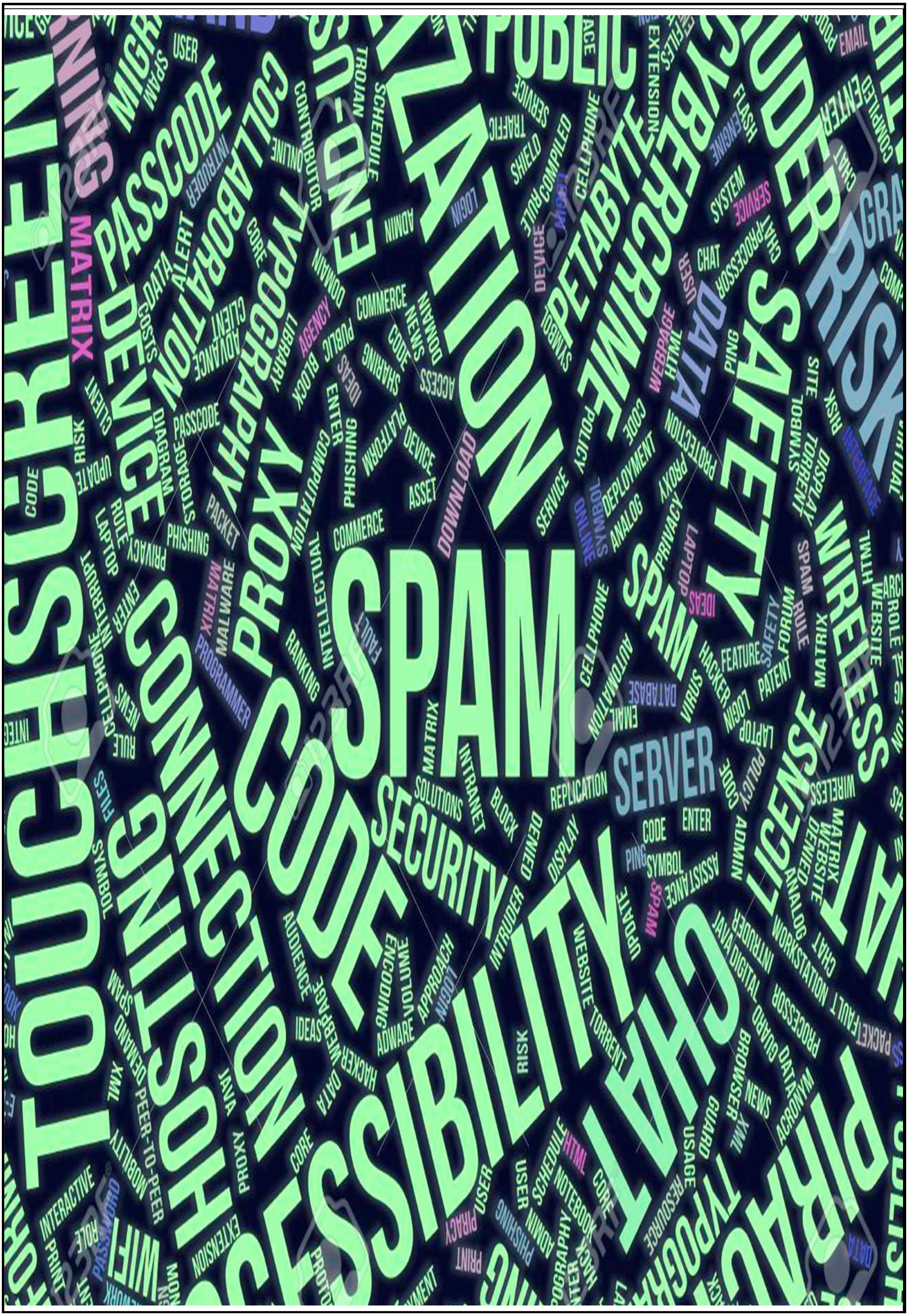

SPAM
SERVER
SECURITY
CODE
PROXY
PASSCODE
PUBLIC
SAFETY
WIRELESS
LICENSE
CONNECTION
DEVICE
MATRIX
DOWNLOAD
COMMERCE
REPLICATION
CELLPHONE
MALWARE
INTELLECTUAL
TYPOGRAPHY
PETABYTE
DATA
RISK
CHAT
WIFI
PHISHING

“Anekant Education Society’s”

Tuljaram Chaturchand College of Arts, Science and Commerce,

Baramati. (Empowered Autonomous)

DEPARTMENT OF STATISTICS

"Optimizing Email Spam Detection and Natural Language Processing with Keyword Driven Multi-Class Labeling."

MSc - II (Statistics)

Submitted By

Patil Om Sanjay (Roll No. 16688)

Ahiwale Sajjan Rajendra (Roll No. 16715)

Mane Ajay Appa (Roll No. 16681)

UNDER THE GUIDANCE OF

Dr. N. A. Jagtap

M.Sc. II (Academic Year 2024-25)

“Anekant Education Society’s”

Tuljaram Chaturchand College of Arts, Science and Commerce,

Baramati. (Empowered Autonomous)

DEPARTMENT OF STATISTICS

MSc - II (Statistics)

CERTIFICATE

Date : 23 / 04 / 2025

This is to certify that the project entitled **"Optimizing Email Spam Detection and Natural Language Processing with Keyword Driven Multi-Class Labeling."** Submitted by **Mr. Ahiwale Sajjan Rajendra** for the award of the degree of Master of Science in Statistics by the Department of Statistics, Tuljaram Chaturchand college, Baramati. In partial fulfillment of course, MSc – II in Statistics in Academic year 2024-2025.

Dr. N. A. Jagtap		**Prof. Dr. V. C. Kakade**
Project Guide	**Examiner**	**Head of Department**

INDEX

ACKNOWLEDGEMENT

The project is an attempt every student to put his efforts into achieving some objectives under a study and concluding with some valid results. While preparing our project report we receive endless help from number of people. This report would be incomplete if we don't convey our sincere thanks to all those who were involved.

We take great satisfaction in completing our project, **"Optimizing Email Spam Detection and Natural Language Processing with Keyword Driven Multi-Class Labeling."** at the Department of Statistics, Tuljaram Chaturchand College, Baramati, during the academic year 2024–2025.

We extend our heartfelt gratitude to **Prof. Dr. A. S. Jagtap** (Principal, Tuljaram Chaturchand College of Arts, Science and Commerce, Baramati.) for his invaluable support and encouragement throughout our project.

Our sincere thanks go to **Prof. Dr. V. C. Kakade**, Head, Department of Statistics, for providing the necessary resources, valuable insights, and continuous cooperation during the course of our work.

A special note of appreciation to our mentors **Dr. N. A. Jagtap** whose valuable guidance, constructive suggestions, and constant encouragement have been instrumental in the successful completion of our project. Their insights have enhanced our understanding and application of statistical techniques in research.

We also extend our sincere gratitude to all the faculty members and non-teaching staff of the Department of Statistics for their continuous support, advice, and motivation. Their contributions, both direct and indirect, have helped us immensely throughout this journey.

Lastly, we acknowledge and appreciate the support of all individuals who contributed to this project in any capacity. Your encouragement and assistance have been invaluable in the successful completion of this research.

Thank you all for your inspiration and guidance!

ABSTRACT

Email communication becomes an absolutely necessary tool in professional and personal setting, encouraging seamless interaction and information exchange. However, the increasing prevalence of SPAM Email represents important issues, such as security threats, inbox ingestion and critical communication obstacles. SPAM messages often include phishing attempts, rogue systems, and distribution of malware. This represents the risks of individuals and organizations. To mitigate these threats, an automated SPAM filter system was developed that uses machine learning and regular classification. These systems effectively reduce SPAM, but they are not perfect, and legitimate emails are often misclassified, recognized SPAM (false positives), or not recognized harmful messages (false negatives).

NLP allows for a deeper understanding of email text by analyzing language patterns, semantic structures and contextual relationships, improving classification accuracy. Additionally, various machine learning algorithms are implemented and compared, such as random forests, support vector machine (SVMs) and deep learning models, are evaluated using critical performance metrics such as accuracy, recall, F1 score and accuracy.

As Email communication continues to be an integral part of modern life, ensuring the effectiveness of SPAM detection methods by incorporating advances NLP techniques and machine learning models. By analyzing linguistic patterns, contextual relationships and classification performance, the research aims to enhance the reliability of SPAM filters while minimizing false positives and false negatives. The outcomes of this study are expected to contribute significantly to the field of Email security, providing valuable insights for developing more sophisticated and efficient SPAM detection systems.

In today's digital world, people receive a huge number of emails daily, including personal messages, spam, social updates, promotions, and service notifications, making manual sorting time-consuming and frustrating. To address this, we developed a simple and fast keyword-based email classification system that automatically categorizes emails into five groups Spam, Promotion, Social, Updates, and Primary based on predefined keyword lists found in the subject and body of the email. This rule-based approach checks for specific words related to each category and assigns the email accordingly, offering a user-friendly solution without the need for complex machine learning algorithms. The system performed well in reducing inbox clutter and organizing emails efficiently. However, it also has some limitations, such as misclassifying emails that use unexpected language. To improve accuracy in the future, the keyword lists can be expanded or the system can be enhanced by integrating machine learning techniques. Overall, this project provides a practical starting point for efficient email management and opens the door for further development.

Keywords: Email SPAM detection, Natural Language Processing (NLP), Machine Learning Algorithms, Data Mining, Deep learning.

MOTIVATION

Email has become an essential part of our daily lives, helping us stay connected, collaborate on projects, and share important information. However, the increasing flood of spam emails has turned this convenience into a challenge. From annoying advertisements to dangerous scams, spam emails not only waste time but also pose serious security threats, such as phishing attacks, financial fraud, and malware infections.

Many spam filters are designed to block unwanted emails, but they are not perfect. Sometimes, important emails get wrongly classified as spam (false positives), causing missed opportunities and communication breakdowns. On the other hand, some spam messages slip through the filter (false negatives), exposing users to potential risks. This problem makes it clear that current spam detection methods need improvement.

This research is motivated by the need to develop a smarter and more reliable spam filtering system. By using Natural Language Processing (NLP) and advanced machine learning techniques, we aim to improve how emails are analyzed and classified. NLP helps computers understand text more accurately by recognizing patterns in language, meaning spam can be detected more effectively. We will test different machine learning models, including Random Forest, Support Vector Machines (SVMs), and Deep Learning, to find the best approach for improving email security.

Our goal is simple to make spam filters smarter, reduce errors, and ensure that important emails are never lost while blocking actual spam effectively. In doing so, we hope to create a more secure, efficient, and hassle-free email experience for everyone.

Managing a large number of emails every day can be challenging, especially when they come from different sources like social media, online shopping, newsletters, work, and unknown senders. Important emails often get lost among spam, promotions, and updates, making it harder for users to focus on what really matters. While major email platforms use advanced algorithms to sort emails automatically, building such systems from scratch can be difficult for beginners or small projects with limited resources.

This inspired us to create a simple and fast keyword-based classification system that can help organize emails effectively without using complex machine learning models. We wanted to develop a method that anyone with basic programming knowledge could understand and implement. By using predefined keywords to detect the type of email based on its subject and body, we aimed to offer a lightweight solution for email categorization. Our goal was to reduce inbox clutter, save users time, and create a foundation that can later be improved using smarter technologies.

This project is also motivated by the increasing importance of email filtering in areas like cybersecurity, productivity, and digital communication. With better email organization, users can respond faster to important messages and avoid the risks of missing out on urgent updates or falling for spam and phishing emails.

INTRODUCTION

Email communication has become an important aspect of modern life, encouraging professional cooperation, personal connections and exchange of important information. The ever-growing spread of spam emails hampers this comfort and highlights users with unrelated, unwanted, and malicious content. To combat this challenge, automated spam filter systems are widely used and effectively block important parts of spam messages. Despite their use, these systems are not without flaws. The persistent issue with this function is the occurrence of false positives that leads to legitimate emails as Spam was misclassified and completely deleted into the spam folder. This misclassification can lead to a missed business opportunity, communication with obstacles, and potential economic losses.

E-mail is an important part of everyday life and helps people stay relevant, collaborate on projects and share important information. From annoying ads to harmful scams, email management can be frustrating due to the rise in unwanted news from Spam E-mails. To improve this, spam filters have been developed to automatically identify and block unnecessary emails. These filters are effective, but not perfect. The main issue is incorrectly positive outcomes when important e-miles are misidentified and lost or ignored. This can lead to missed business opportunities, communication envelopes, and even economic losses. This study examines how natural language processing (NLP) improves spam recognition systems by analyzing email content more accurately. With NLP, computers can understand human language and allow spam to more effectively distinguish spam from actual messages. Additionally, various techniques for machine learning have been tested, including random forests, vector machines (SVMs), and deep learning models, to find the best approach to spam filtering.

NLP provides a powerful device for analyzing the textual content of emails, allowing you to gain a deeper understanding of the language patterns and properties that distinguish spam from legitimate messages. By including NLP, this study aims to improve the accuracy and memory of spam filters, reducing the reduction in false positive aspects and false negatives. In addition to NLP, this study evaluates and compares several algorithms of machine learning to identify the most effective approach to spam detection.

Algorithms such as Random Forest, Support Vector Machines (SVMs), and Deep Learning Models are implemented and analyzed to determine their strengths and limitations. Important power metrics such as accuracy, recall, F1 score, and accuracy lead to the best possible model selection. By addressing dual goals to minimize false positives and improve identification effectiveness, this study should provide a comprehensive framework for improving the security and efficiency of electronic communications.

The purpose of this study is to improve the reliability of spam filters by evaluating these models using critical performance measures such as accuracy, recall, and accuracy. The purpose of this study is to make spam filters more intelligent and efficient, reduce errors, and improve email security. By combining NLP with machine learning, this study aims to develop a more sophisticated filter system that minimizes false alarms and at the same time ensures that actual spam is correctly identified. This creates a smoother and more secure email experience for users.

Email communication is an important part of modern life and acts as a critical mode of interaction for individual, professional and business purposes. Given the increased reliance on emails for information, cooperation and exchange of transactions, ensuring a seamless and secure electronic experience is a priority. The efficiency and reliability of electronic communications, which is at stake, is the overwhelming presence of spam emails. In many cases, unstable and unrelated spam messages abuse security threats such as incoming calls as well as phishing attacks, financial fraud, and the distribution of malware. The global rise in spam email requires the development of effective filtering mechanisms to protect users from these threats and ensure smooth communications.

These systems use predefined rules, heuristics, and algorithms to identify and retrieve spam messages from legitimate emails using machine learning. These filters have significantly improved the email experience with reduced spam load, but are free of flaws. One of the most urgent concerns related to spam filters is the creation of false positive aspects of legitimate emails that are misclassified as spam. If your important email is incorrect and is redirected or deleted to a spam folder, this may miss the possibility that this could lead to communication and financial losses. Conversely, false negative cases in which SPAM Email sends filters via email and terminates in your inbox can lead users to security risks. As a result, the effectiveness of spam filters depends on the ability to minimize both false positives and false negatives, while simultaneously maintaining high accuracy.

However, Spammer has developed a method to bypass these rules using the boring method. Change spelling, insert random characters, and use highly developed language patterns. As a result, rule based filters have become increasingly ineffective when adapting to the development of spam tactics. Algorithms such as Naive Bayes, Decision-Making, Support Vector Machine (SVM) have been frequently used in spam filter applications.

These models analyze a variety of features, including email content, metadata, and behaviors to distinguish between spam and legitimate messages. Despite its advantages, spam-based spam filters with machine learning bases still face challenges in handling complex, dynamic spam patterns that lead to continuation of false positives and false negatives. NLP technology allows spam filters to

analyze the textual content of Email in greater detail by understanding semantics, contextual relationships, and linguistic structure meanings.

By using NLP, spam detection systems can go beyond simple keywords and instead evaluate the overall meaning of email. NLP has proven to be extremely effective for text classification tasks and is a valuable device for recognizing spam. Using NLP technology, spam filter email content can be analyzed at a lower level, identifying unique patterns and properties of spam and legitimate email. Text preprocessing steps help eliminate noise, standardize text, and improve the output of the model, such as deleting stop words, trucking, and lemmatization.

Emails are an essential part of modern communication, whether for personal use, business, or marketing. However, the large number of emails people receive every day can make it hard to find important messages quickly. Emails often include spam, promotional offers, social media alerts, and updates from different services. Without proper organization, the inbox can become overwhelming.

To solve this problem, email platforms usually use automated filters that sort messages into different folders. In this project, we developed a basic keyword based email classification system that can automatically sort emails into five common categories:

- Spam – unwanted or harmful emails
- Promotion – marketing and sales-related messages
- Social – notifications from social media platforms
- Updates – service alerts or app updates
- Primary – personal or important emails

The main idea is simple: we use lists of keywords for each category. For example, if an email contains words like "limited offer", "discount", or "sale", it likely belongs to the Promotion category. Similarly, words like "friend request", "like", or "comment" might indicate the email is from a social media platform, and it should go into the Social category.

Our system checks both the subject and body of each email for these keywords. If matches are found, the email is classified accordingly. This rule-based approach is easy to build, quick to run, and doesn't require a lot of computational resources or technical setup.

While the results are promising, the method also has its challenges. If the email uses different words not included in the keyword list, the system might misclassify it. To improve accuracy in the future, we can make the keyword lists smarter or even use machine learning models that can learn from past email patterns. this keyword-based classification system is a good starting point for organizing emails more effectively. It helps users save time and makes their inbox cleaner and easier to manage.

Naïve Bayes showed modest performance with 63% overall accuracy. It handled Promotion and Social emails moderately well but struggled significantly with Spam and Updates classification. Despite achieving 66% precision, its recall was the lowest at 34%, indicating a high number of false negatives. This makes it less suitable for spam-heavy environments, where identifying actual spam is critical. Its simplicity and speed are advantageous, but it underperforms in complex, multi-category classification tasks. While useful as a baseline model, Naïve Bayes may require improvements or enhancements for better accuracy and coverage in real-world email classification scenarios.

SVM achieved a strong performance with an overall accuracy of 87%. It excelled in identifying Spam and Updates emails with high precision (84%), though recall was moderate (45%), suggesting it missed some relevant messages. SVM performed reliably across all categories but requires more computation than simpler models. Its strength lies in separating categories with clear

boundaries, making it suitable for structured spam classification. However, it may need tuning to improve recall, especially when detecting subtle or ambiguous email types. Overall, SVM offers a balanced yet conservative approach, emphasizing correctness over completeness in email classification.

KNN performed effectively with 91% overall accuracy and balanced precision and recall (both above 60%). It was particularly strong in Spam and Updates detection but showed slight weaknesses in classifying Social emails. KNN benefits from simplicity and requires no training time, but it is sensitive to data scaling and computationally expensive during prediction. Its ability to generalize well across categories makes it a suitable option for multi-class classification. Despite minor inconsistencies, KNN is a reliable algorithm that provides competitive performance across categories, especially when supported by proper preprocessing and optimal value selection for neighbors (k).

Random Forest was the most accurate model, achieving 92% overall accuracy with high precision (89%) and recall (63%). It demonstrated robust classification across all email categories, particularly excelling in detecting Spam, Updates, and Social emails. As an ensemble method, it mitigates overfitting and captures complex patterns effectively. Its strength in balancing performance metrics makes it ideal for real-world multi-class classification tasks. Random Forest's interpretability and stability are also valuable, providing reliable insights into feature importance. Due to its consistent and superior performance, it stands out as the best model for email classification among the compared algorithms.

Logistic Regression yielded 89% overall accuracy with 74% precision and 52% recall. It performed well in detecting Primary and Spam emails but showed limitations in classifying Social emails accurately. As a linear model, it is interpretable and efficient, making it a suitable baseline for text classification. However, its relatively lower recall implies a higher rate of false negatives, which may hinder its reliability in spam detection scenarios. Logistic Regression is a practical choice for scenarios prioritizing precision and simplicity but might benefit from enhancements or combination with other models to handle complex and imbalanced datasets better.

Decision Tree achieved a strong 90% accuracy with excellent recall (65%) and precision (85%). It performed well in identifying Primary and Spam categories, although it underperformed in the Social category. Its ability to model non-linear relationships makes it a versatile option, but it is prone to overfitting on small or noisy datasets. With appropriate pruning or ensemble methods, Decision Trees can become highly effective. Its interpretability makes it valuable for understanding decision rules in classification tasks. Overall, it is a competitive model that offers strong performance and insights, especially when combined with boosting or bagging techniques.

AdaBoost provided competitive performance with 90% accuracy, 80% precision, and 60% recall. It handled Primary, Promotions, and Spam emails well but severely underperformed in Social email classification (21%). As an ensemble technique, AdaBoost improves weak learners and increases classification robustness. Its precision-recall balance is commendable, though low Social category performance suggests it may struggle with imbalanced or less distinct classes. AdaBoost is suitable for high-accuracy environments with structured data and consistent feature importance. However, improvements or hybrid models may be needed to ensure consistent performance across all categories in more diverse or nuanced email datasets.

OBJECTIVES

1) To select the best performing algorithm for email spam detection based on the obtained results.
2) To explore and implement natural language processing (NLP) methods in spam detection to enhance text analysis and classification.
3) To develop and implement a keyword-based email classification system that categorizes emails into distinct categories (Spam, Promotion, Social, Updates, and Primary) using predefined keyword lists for subject and body text analysis.

REVIEW OF LITERATURE

1. Email Spam Detection Using Machine Learning Algorithms:

Several researchers have explored different machine learning methods to detect spam emails effectively. Nikhil Kumar and his team from Delhi Technological University compared various algorithms like Naïve Bayes, Random Forest, and ensemble techniques to find the most accurate spam classifier. Their study concluded that while Naïve Bayes performs well, ensemble methods such as bagging and boosting offer better accuracy and reliability, especially in filtering spam content rather than just relying on domain-based detection. However, their system had limitations due to the smaller dataset used, which affected the generalizability of their findings.

2. Email Spam Detection Using Machine Learning:

Study by Anitha Reddy and her students from Sreyas Institute applied Natural Language Processing (NLP) techniques such as tokenization, stop-word removal, and stemming to clean email data before applying machine learning models. They tested several algorithms and found that Naïve Bayes achieved the highest accuracy at 99%, making it their preferred model. This research highlights how simple preprocessing steps combined with lightweight models can result in strong performance, making this method suitable for practical email filtering tasks.

Machine learning and natural language processing (NLP) effectively enable email spam classification by leveraging supervised algorithms like Naïve Bayes, SVM, and KNN. Preprocessing techniques such as tokenization, stop-word removal, and stemming enhance model performance. Among the models tested, Naïve Bayes achieved the highest accuracy (99%), followed by SVM (98%) and KNN (97%), making Naïve Bayes the preferred choice. These methods not only automate spam detection but also enhance security and productivity in email communication, with potential extensions to combat sophisticated threats like phishing.

3. Email Spam Filter using Machine Learning :

Hazel Murphy designed a spam detection system that aimed to improve user productivity by reducing time spent on managing spam emails. Her system demonstrated excellent performance, achieving 98.56% accuracy, 0.997 precision, and a recall of 0.99174. These results show that her implementation of the Naïve Bayes model was highly effective at identifying spam while minimizing the risk of flagging legitimate emails. This study emphasizes the importance of balancing precision and recall to ensure accurate and reliable spam filtering.

The metrics provided from the functional specification document show the success of the spam detection feature:

- Specificity: 0.9448, indicating that only 0.5662% of non-spam emails are misclassified as spam.
- Precision: 0.997, meaning only 0.1% of emails are wrongly classified as spam.
- Accuracy: 0.9856, demonstrating that the Naïve Bayes model classifies emails correctly with minimal false negatives (8).

- Recall: 0.99174, ensuring that the tool correctly identifies nearly all spam emails, with an emphasis on high precision to avoid misclassifying important non-spam emails.

4. SPAM DETECTION REPORT:

Nimrat Virk and Kingsley Okeke focused on feature selection and classification using several machine learning models. Their study found that the k-Nearest Neighbour (KNN) method consistently provided the best results, especially when more attributes were used in training. Their findings support the idea that increasing the number of meaningful features can improve spam detection performance. Their use of classification algorithms like Decision Trees, Bayes Net, and Neural Networks adds depth to the research by showing different approaches to spam detection.

In this project, feature selection was used to identify key words that differentiate between spam and ham emails. Among the classifiers tested, the k-Nearest Neighbour (k=3) method consistently achieved the highest classification accuracy. Additionally, increasing the number of attributes generally improved accuracy, with potential for further improvement beyond 94.5%. Overall, the k-NN method demonstrated superior performance in spam classification, and increasing feature attributes could enhance the model's accuracy even further.

5. Spam Detection in Emails using Machine Learning :

Lastly, Prazwal Thakur and Kartik Joshi experimented with more advanced models like Recurrent Neural Networks (RNNs) alongside traditional ones such as SVM and Decision Trees. Their results showed that RNNs outperformed other models in terms of accuracy and were better at handling unstructured text data. This study demonstrates how deep learning models are evolving to offer even greater accuracy in spam classification, especially when combined with proper preprocessing steps. Overall, these studies show that while basic models like Naïve Bayes work well, combining them with advanced methods and thoughtful data preparation can lead to more powerful and accurate email spam detection systems.

This study explored various machine learning methods for spam filtering, focusing on the evolution of spam detection systems and their effectiveness. The raw, unstructured email data was pre-processed through techniques such as stemming, stop word removal, and tokenization to prepare it for analysis. After dimensionality reduction, different machine learning models were applied and their performance compared based on accuracy and precision. The findings suggest that Recurrent Neural Networks (RNNs) outperformed other models, achieving the highest accuracy across various datasets and scenarios. This highlights the potential of RNNs as a highly effective method for spam detection, offering a significant improvement over traditional approaches.

METHODOLOGY

6.1 Data Description :

We have downloaded MBOX files from the Gmail accounts of 10 individuals, then used Python to convert each MBOX file into an Excel format. After that, we combined all the individual Excel files into one consolidated dataset, resulting in a collection of over 55,436 emails. The dataset under analysis processed using the Python engine `'openpyxl'`.

Users Email id :

1. ompatil1101@gmail.com
2. ajaymane5885@gmail.com
3. sd307907@gmail.com
4. ahiwale.sa074@gmail.com
5. naynaom8649@gmail.com
6. patilsanjay965@gmail.com
7. spatil9108@gmail.com
8. opykaz1008@gmail.com
9. oppatil1603@gmail.com
10. srujanshaktishramik2008@gmail.com

Each email has five primary columns:

1. **Date** – The timestamp of the email.
2. **From** – Sender's email address.
3. **To** – Receiver's email address.
4. **Subject** – The subject line of the email.
5. **Body** – The content/message of the email.

Dataset Shape :

- Total Rows: 55,436
- Total Columns: 5

6.2 Data Cleaning & Preprocessing :

a. **Remove duplicates, nulls, and irrelevant entries:**

This step ensures the dataset is clean by deleting repeated emails, empty fields, or unnecessary information that could negatively affect the analysis or model accuracy.

b. Normalize text (lowercase, remove special characters):

Text normalization makes the email content uniform by converting all letters to lowercase and removing symbols or special characters like “@”, “#”, or “!” which may not add meaning.

c. Tokenization:

Tokenization is the process of breaking down the email text into smaller units, like individual words or phrases, to make it easier for a machine to understand and analyze.

d. Stop word Removal:

Stop words are common words like "the", "is", "at", which don’t add much meaning in spam detection. Removing them helps the model focus on more meaningful words.

e. Stemming:

Stemming reduces words to their base form by cutting off prefixes or suffixes. For example, "playing", "played", and "plays" are all reduced to "play".

f. Lemmatization:

Lemmatization also reduces words to their base form, but it uses grammar rules to return actual dictionary words. For example, "running" becomes "run" and "better" becomes "good".

6.3 Data Visualization :

1. Word Cloud :

A word cloud shows the most common words used in all the emails. The words that appear more often look bigger and bolder in the image. For example, if words like “win”, “free”, or “click” are big, it might mean the emails are spam. This helps us easily spot keywords without reading each email.

2. Distribution of Email Body Length :

This chart shows how long the emails are by counting the number of words or characters in each email body. It helps us see if most emails are short, medium, or long. It also shows if spam emails are usually longer or shorter than normal emails. This gives us clues that can help in identifying spam.

3. Top 10 Email Senders :

This chart shows the top 10 people or companies who sent the most emails in our dataset. This helps us understand who the most active senders are. If one sender appears too

Often, and their emails look like spam, we can investigate further. This is useful for spotting repeated spam or finding important senders.

6.4 Natural Language Processing :

1. Text Preprocessing :

Definition :

Text preprocessing refers to the set of techniques applied to raw textual data to convert it into a clean, structured, and analyzable form.

Preprocessing Techniques Definitions :

1. **Lowercasing:** Converts all characters to lowercase to ensure uniformity.
2. **Special Character Removal:** Removes non-alphanumeric characters using regular expressions.
3. **Whitespace Normalization**: Removes unnecessary spaces, tabs, and line breaks.

2. Tokenization

Definition:

Tokenization is the process of splitting a text into individual words, phrases, or other meaningful elements called tokens.

3. Stopword Removal

Definition:

Stopwords are commonly used words in a language that carry minimal semantic value (e.g., "is", "the", "at", "in").

1. **Stemming and Lemmatization**

These two techniques help in word normalization.

2. **Stemming:**

It reduces a word to its root form by chopping off suffixes. For instance, “advertising,” “advertised,” and “advertisement” may all be reduced to “advertis”.

3. **Lemmatization:**

A more sophisticated version of stemming that returns actual words. For example, “running” becomes “run” and “better” becomes “good”.

➢ Feature Engineering for Email Analysis

Definition:

Feature engineering is the process of creating new input features from existing data to improve the performance of machine learning models.

Features Used in the Project:

1. **Subject Length** – The number of characters in the subject line. Spam subjects are often short and urgent.
2. **Word Count** – Total number of words in both subject and body.
3. **Link Count** – Number of hyperlinks present. Spam emails often contain many links.
4. **Special Character Count** – Frequency of symbols like $, %, !, which are common in spam.
5. **Month & Day of Week** – Temporal analysis to detect if spam peaks during specific periods.

6.5 Feature Engineering:

a. Subject Length, Word Count, Link Count, Special Characters :

These are new features created from the original email data. counting the number of characters in the subject, total words in the email, how many links it contains, and how often special characters like "$", "!", or "%" appear. These details help the model understand patterns often found in spam.

b. Add Date-Based Features (Month, Day of Week) :

By analyzing the date an email was sent, we can extract the month or day of the week to see if spam emails are more common during certain times, like weekends or holidays.

c. Keyword Categories (Spam, Social, Promotion, etc.)

Specific words like "win", "free", or "urgent" are often found in spam. Similarly, words like "Instagram", "Facebook" may relate to social emails. Grouping these words into categories helps the model classify emails more accurately.

1. **Spam Keywords** – Terms like "win", "free", "prize", etc., identified in the text.
2. **Promotion Keywords** – Words related to discounts, deals, and offers.

3. **Social Keywords** – References to platforms like Facebook, Instagram, Twitter.
4. **Update Keywords** – Informative or notification-based terms.
5. **Default to Primary** – If no keywords are detected, classify as a primary category.

- **Label Encoding :**

Label encoding is a method to convert text labels into numbers so that machine learning models can understand them. In this case, we change the email category "ham" (not spam) to 0 and "spam" (unwanted or junk email) to 1.

- **Text Vectorization :**

Machine learning models are unable to interpret raw text directly, so tools like Count Vectorizer and Tfidf Vectorizer are used to convert email content into numerical values. These tools analyze the frequency of words and transform the text into a numerical matrix that models can effectively process and learn from.

- **Data Splitting :**

The dataset was divided into training and test sets:

- Training Set: 70%
- Testing Set: 30%

6.6 Machine Learning :

Machine Learning is a branch of Artificial Intelligence that focuses on building systems that can learn from data, identify patterns, and make decisions with minimal human intervention. Instead of being explicitly programmed, the machine improves its performance based on experience.

Types of Machine Learning :

1. **Supervised Learning**

In supervised learning, the model is trained on a labeled dataset, meaning that each input has a corresponding correct output. The goal is to learn a mapping from inputs to outputs so the model can predict the output for new, unseen data. Common algorithms used in supervised learning include Linear Regression, Logistic Regression, Decision Trees, Random Forests, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Naive Bayes.

2. **Unsupervised Learning**

Unsupervised learning deals with data that has no labels. The model tries to discover hidden patterns, groupings, or structures within the data. It is useful for clustering and dimensionality reduction tasks. Popular algorithms in this category are K-Means Clustering, Hierarchical Clustering and Principal Component Analysis (PCA).

3. Semi-Supervised Learning

Semi-supervised learning uses a combination of a small amount of labeled data and a large amount of unlabeled data. It falls between supervised and unsupervised learning and is often used when labeling data is expensive or time-consuming. Examples of algorithms used include Semi-Supervised Support Vector Machines, Label Propagation, and Self training methods.

6.7 Statistical Techniques of Machine Learning Algorithms :

6.7.1 Naive Bayes :

A probabilistic classifier based on Bayes' theorem. Assumes independence between words (features). Used for spam detection, sentiment analysis, etc. Works well with small datasets and high-dimensional data. Fast and simple to implement. Variants include Multinomial and Bernoulli Naive Bayes.

Naive Bayes is like a statistical rule we use to guess which group something belongs to, based on what we already know. Imagine you're a teacher who gets essays from students and wants to decide whether the essay is written in a positive or negative tone. You look at the words in the essay: if it has words like "good", "excellent", "happy", it's probably positive. If it has "bad", "sad", "terrible", it's likely negative.

Statistically, Naive Bayes applies Bayes' Theorem**,** which is all about calculating the probability of something being true given some evidence. In this case, it calculates the probability that an email is spam, given that it contains certain words. The "naive" part comes from assuming that all the words in the email are independent of each other, even though that may not be true (but it still works well).

6.7.2 Logistic Regression :

A linear model that predicts probability of a class. Uses a sigmoid function to map input to output. Effective for binary classification problems. Handles high-dimensional data like text features (TF-IDF). Not ideal for non-linear relationships. Good baseline model in NLP pipelines.

Logistic Regression may sound like it does regression, but it's actually used for classification deciding between categories like "yes" or "no", "spam" or "not spam". It's like when you try to predict the chance of rain based on temperature, humidity, and wind. If the chance is over 50%, you say, "It will rain"; otherwise, "It won't".

In statistics, logistic regression uses a logic function (or sigmoid curve), which takes any number (from minus infinity to plus infinity) and squashes it between 0 and 1**.** This result can be interpreted as a probability**.** So, from a statistical point of view, logistic regression estimates the relationship between one or more independent variables (X) and a binary dependent variable (Y) using the log-odds of the probability of Y.

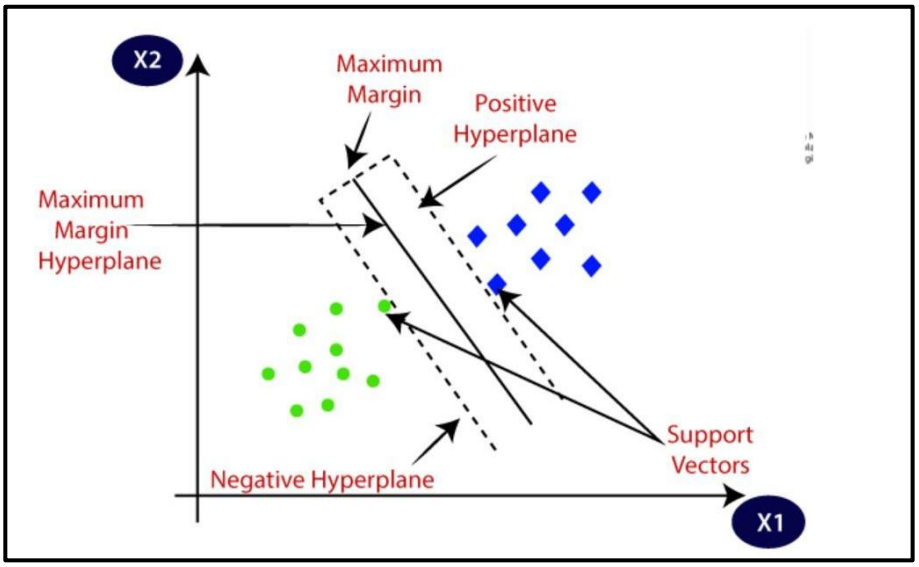
6.7.3 Support Vector Machine (SVM) :

Finds the best hyperplane to separate classes. Maximizes margin between data points and decision boundary. Performs well on text data with clear class separation. Effective with sparse data like TF-IDF vectors. Can use kernel trick for non-linear data. Works well for small to medium-sized datasets.

Support Vector Machine is a bit like trying to draw a line between two groups of points on a graph say, blue circles and red squares. But you don't want just any line; you want the one that is as far away as possible from both groups. This way, it's more likely that new points will be correctly classified.

In statistical terms, SVM tries to find a hyperplane (a boundary) that best separates the data into two categories. It focuses on the data points that are closest to the boundary these are called support vectors. These critical points define where the boundary should be.

If the data can't be separated with a straight line, SVM uses mathematical tricks (called kernels) to map the data into higher dimensions where it can draw a better boundary. This is similar to what we do in statistics when we apply transformations to non-linear data to make it easier to analyze.



6.7.4 K-Nearest Neighbors (KNN) :

Classifies based on majority of 'k' nearest samples. Distance metric (e.g., Euclidean) measures similarity. Non-parametric, no training phase required. Slow with large datasets due to distance calculations. Sensitive to feature scaling and irrelevant data. Rarely used for NLP, but useful in small scale tasks.

KNN is one of the simplest ways to make a decision: it says, “Let’s look around and see what the neighbors are doing.” Imagine you just moved to a new city and you see that most of the people near your home eat at a particular restaurant. You’d probably guess it’s a good place to eat and decide to go there too. That’s what KNN does.

In statistical language, KNN is a non-parametric method, meaning it doesn’t make any assumptions about the data distribution. It doesn’t build a model during training it just stores the training data. Then when you give it a new data point to classify, it calculates the distance between that point and all others in the training set. It selects the K closest points and votes on which category the new point belongs to, based on the majority. In terms of statistics, it uses a distance measure (like Euclidean distance) to define similarity and applies a simple rule: majority wins.

6.7.5 Random Forest:

An ensemble of decision trees built on random samples. Uses majority voting to improve accuracy and reduce overfitting. Performs well on text classification tasks. Handles both numerical and categorical data. Slower than simpler models but more accurate.

This algorithm considers numerous decision trees, thus forming a forest. It is also called an ensemble of decision tree algorithms. This can be used for classification as well regression. This algorithm tries to find out best feature randomly among all the features. In our experiment, we have used 100 decision trees and Gini for impurity index.

Random Forest is one of the most effective machines learning models for predictive statistics, making it an industrial machine learning task. The random forest model is a type of add-on model that creates predictions by combining decisions from a sequence of basic models. where the ultimate model is the total of the basic simple models.

Here, the division of each foundation is a simple decision tree. This deep method of using multiple models to obtain better guessing performance is called model integration. In random forests, all base models are built separately using a separate sample of data.

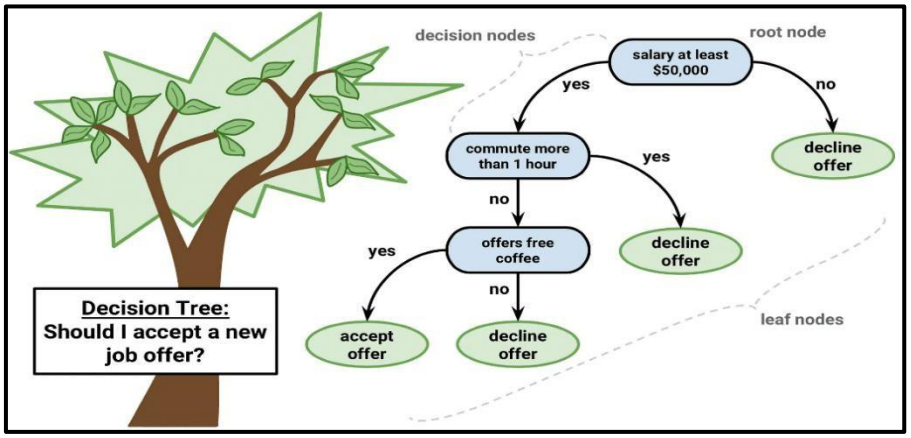
6.7.6 Decision Tree:

A decision tree is a flowchart like tree structure where each internal node denotes the feature, branches denote the rules and the leaf nodes denote the result of the algorithm. It is a versatile supervised machine-learning algorithm, which is used for both classification and regression problems. It is one of the very powerful algorithms. And it is also used in Random Forest to train on different subsets of training data. Splits data into branches based on feature values. Simple, interpretable, and easy to visualize. Can overfit if not pruned or regularized. Useful for feature importance and rule-based modelling. Faster training compared to ensemble models. Not ideal alone for complex NLP tasks.

Decision Tree Terminologies

Some of the common Terminologies used in Decision Trees are as follows:

- Root Node: It is the topmost node in the tree, which represents the complete dataset. It is the starting point of the decision-making process.
- Decision/Internal Node: A node that symbolizes a choice regarding an input feature. Branching off of internal nodes connects them to leaf nodes or other internal nodes.
- Leaf/Terminal Node: A node without any child nodes that indicates a class label or a numerical value.
- Splitting: The process of splitting a node into two or more sub-nodes using a split criterion and a selected feature.
- Branch/Sub-Tree: A subsection of the decision tree starts at an internal node and ends at the leaf nodes.
- Parent Node: The node that divides into one or more child nodes.
- Child Node: The nodes that emerge when a parent node is split.
- Impurity: A measurement of the target variable's homogeneity in a subset of data. It refers to the degree of randomness or uncertainty in a set of examples. The Gini index and entropy are two commonly used impurity measurements in decision trees for classifications task
- Variance: Variance measures how much the predicted and the target variables vary in different samples of a dataset. It is used for regression problems in decision trees. Mean squared error, Mean Absolute Error, Friedman's, or Half Poisson deviance are used to measure the variance for the regression tasks in the decision tree.
- Information Gain: Information gain is a measure of the reduction in impurity achieved by splitting a dataset on a particular feature in a decision tree. The splitting criterion is determined by the feature that offers the greatest information gain, It is used to determine the most informative feature to split on at each node of the tree, with the goal of creating pure subsets
- Pruning: The process of removing branches from the tree that do not provide any additional information or lead to overfitting



6.7.7 AdaBoost:

AdaBoost, short for Adaptive Boosting, is one of the most powerful ensembles learning techniques in machine learning. The main goal of AdaBoost is to improve the performance of simple, weak models by combining many of them into a strong, accurate model. At its core, AdaBoost works by building a sequence of models, each trying to fix the mistakes made by the ones before it. This adaptive nature learning from past errors and focusing more on difficult cases is what makes it both unique and effective.

From a statistical point of view, AdaBoost works by maintaining a set of weights for the training data. Initially, all data points are treated equally. A weak learner typically a decision stump (a decision tree with only one split) is trained on the data. This weak learner makes predictions, and some of these will be wrong. AdaBoost increases the weights of the incorrectly classified instances, which means the next model pays more attention to these hard-to-classify points. At the same time, it reduces the weights for the points that were classified correctly, since they are considered easier and less important for the next round. This process continues for many iterations, and in each one, the model becomes better at focusing on the most difficult cases.

The concept of "weak learners" is central to AdaBoost. A weak learner is a model that performs only slightly better than random guessing. For example, in binary classification (where the goal is to choose between two categories), random guessing would achieve around 50% accuracy. A weak learner might get 55% or 60% accuracy not great, but better than chance. On its own, it's not good enough. But AdaBoost builds a strong classifier by combining a large number of such weak learners. After each round, the algorithm calculates the accuracy of the learner and gives it a weight based on its performance. Better models are given more importance in the final combination, while weaker ones contribute less. The beauty of AdaBoost lies in its flexibility and adaptiveness. It doesn't need the weak learner to be very complex something as simple as a one-level decision tree can work well. And it automatically focuses on the hardest parts of the dataset, adapting its attention to where it's most needed. This makes AdaBoost particularly effective in situations where some classes are harder to distinguish, or when the data is noisy but still contains useful patterns.

6.8 Confusion Matrix :

The following are some of the Terminologies of Confusion Matrix :

a. True Positives (TP): These are cases in which we predicted yes, and they have encountered fraud.
b. True Negatives (TN): We predicted no, and they don't have encountered fraud.
c. False Positives (FP): We predicted yes, but they have not encountered fraud. (Also known as a "Type I error.")
d. False Negatives (FN): We predicted no, but they actually have encountered Fraud. (Also known as a "Type II error.")

1. Accuracy : Accuracy allows for the comparison of different models trained on the same dataset. Models with higher accuracy values are generally preferred.
2. Precision : Precision provides insight into the quality of positive predictions made by a model. A high precision indicates that when the model predicts a positive class, it is likely to be correct.
3. Recall : Recall measures a model's ability to correctly identify all relevant positive cases. A high recall means the model successfully finds most of the actual positive instances.
4. F1 Score : The F1 score is commonly used evaluating the performance of binary classification models. It is especially useful when there is a trade-off between precision and recall.

EXPLORATORY DATA ANALYSIS

7.1 Keywords Classification :

Date splitting in the `value_counts()` output of your Data Frame shows the distribution of different email labels. Here's a simple breakdown:

- **Spam**: 32,790 emails
- **Primary**: 15,784 emails
- **Updates**: 2,507 emails
- **Promotion**: 2,281 emails
- **Social**: 2,074 emails

We can use NLP techniques like tokenization (splitting text into words), removing common words (like "the", "is", etc.), and reducing words to their base form (using stemming or lemmatization). We can also use TF-IDF to turn the email text into numbers, which helps identify important words for classification. This will help the model better understand the content of emails and classify them into categories like Spam, Promotion, or Primary.

Keyword-Based Classification basic keyword-based classification was used to label emails into categories such as Spam, Primary, Promotions, Updates, and Social. Specific keyword lists were created for each category. Manual Label Mapping for binary classification, the labels were mapped as "spam" or "ham" based on the occurrence of spam-related keywords in the email content.

spam_keywords = ["unsubscribe", "win", "click here", "free", "claim", "urgent", "limited offer", "winner", "guaranteed", "risk-free", "exclusive", "money", "claim your prize", "offer expires", "free trial", "credit card", "cash prize", "unsecured", "no payment required"]

promo_keywords = ["offer", "deal", "discount", "sale", "coupon", "promo", "clearance", "bargain", "flash sale", "hot deal", "special offer", "limited time", "today only", "big discount", "buy now", "shop today", "online exclusive", "exclusive deal", "seasonal sale"]

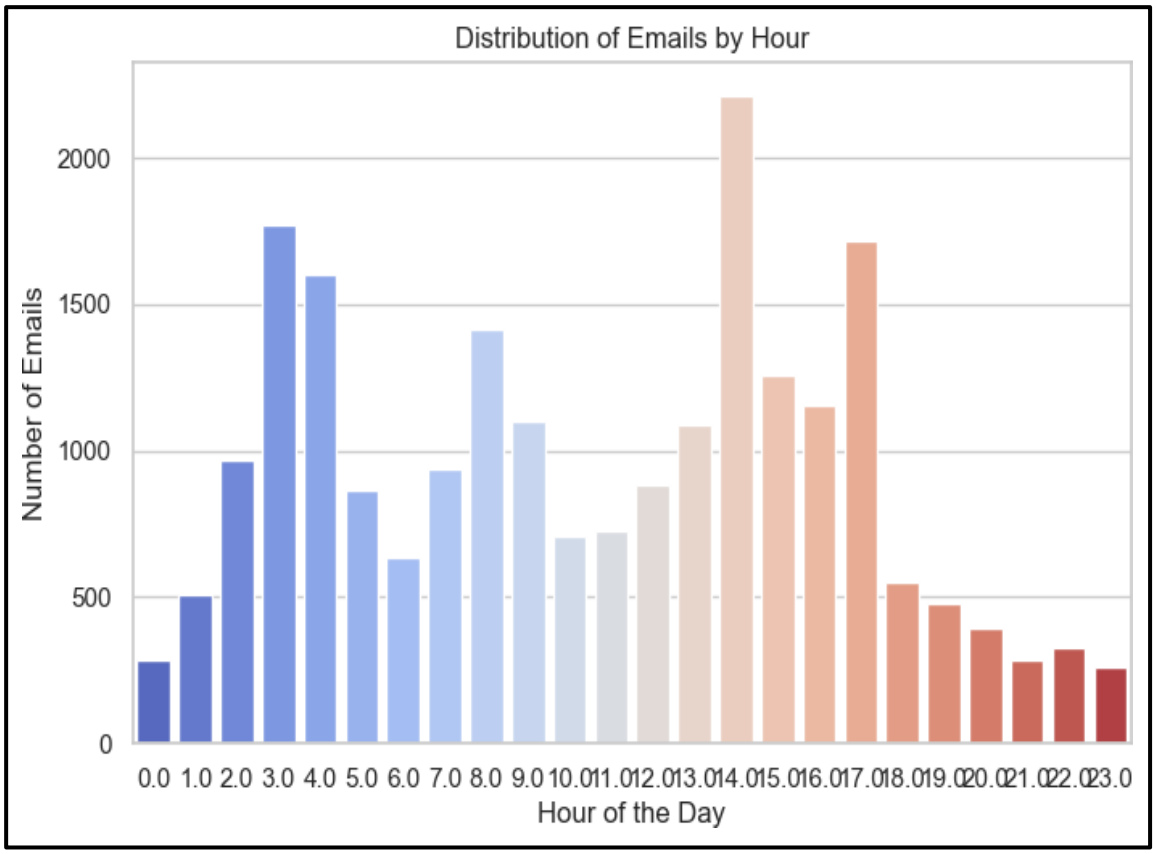
social_keywords = ["friend request", "like", "comment", "follow", "connection", "new follower", "your post", "social invite", "message request", "share your status", "new like", "new comment", "friend suggestion", "tagged in photo", "social media update", "profile update", "join the group"]

update_keywords = ["update", "newsletter", "report", "summary", "digest", "alert", "news", "update notification", "policy update", "account update", "system update", "upgrade", "change", "announcement", "new feature", "release notes", "latest news"]

primary_keywords = ["meeting", "appointment", "invoice", "schedule", "reminder", "password", "receipt", "confirmation", "team", "project", "task", "follow-up"]

7.2 Bar Charts :

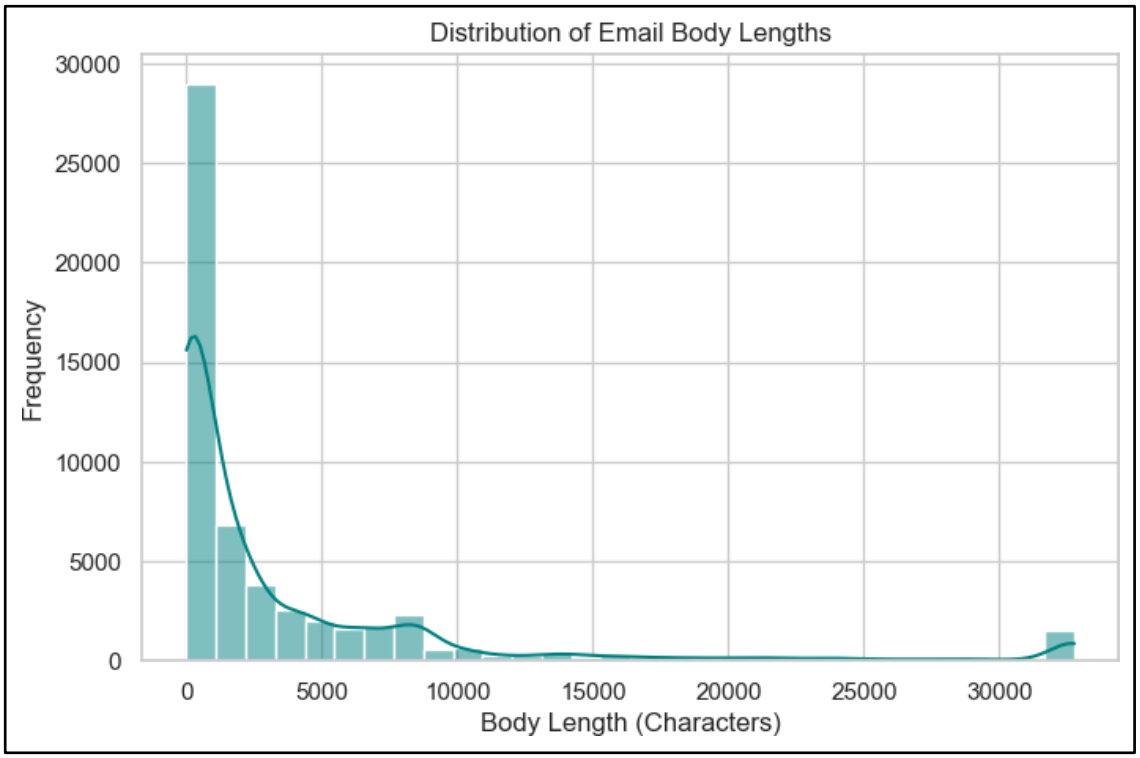
7.2.1 Analysis of Email Activity Patterns in the Day :



The chart shows that email activity peaks at 2:00 PM, indicating heavy usage during working hours. There is another noticeable spike around 3:00–4:00 AM, likely due to automated or scheduled emails.

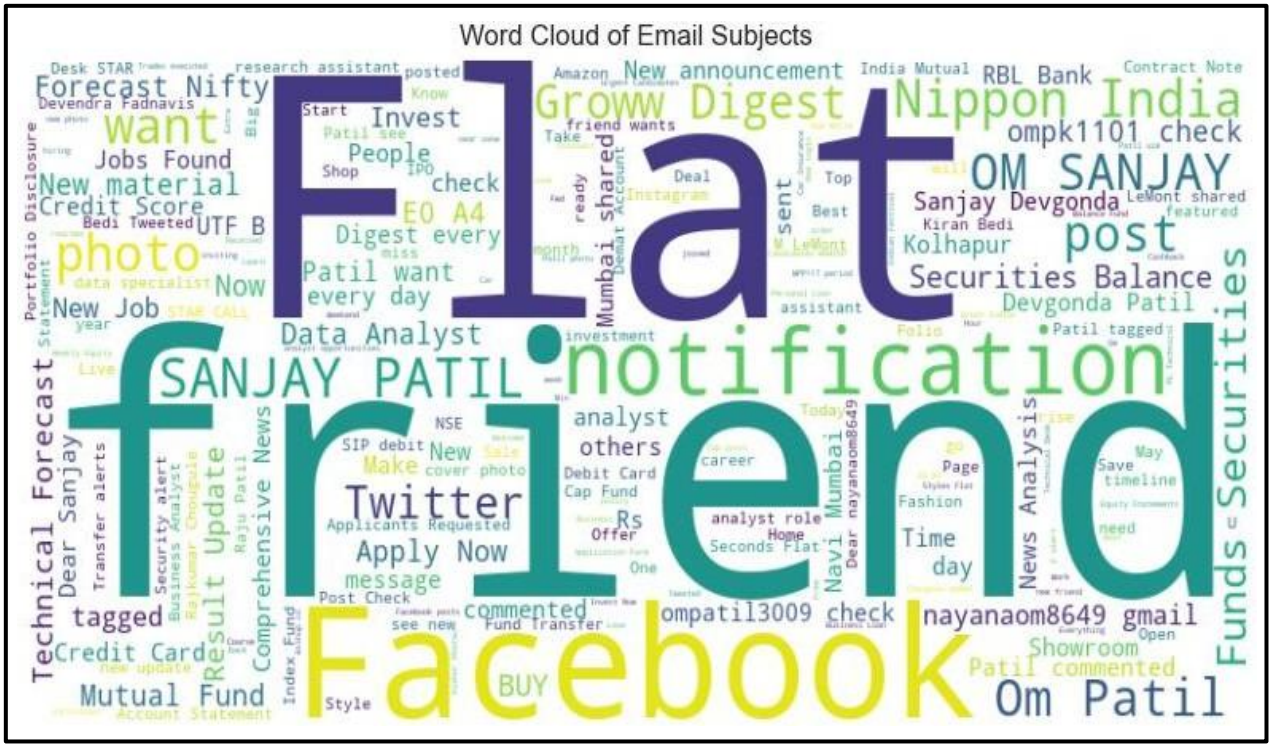
Email volume gradually increases from early morning and remains high until late afternoon. After 6:00 PM, the number of emails drops steadily, with minimal activity at night. This pattern highlights a mix of human and automated email behavior across different times of the day.

7.2.2 Distribution of Email Body Lengths :



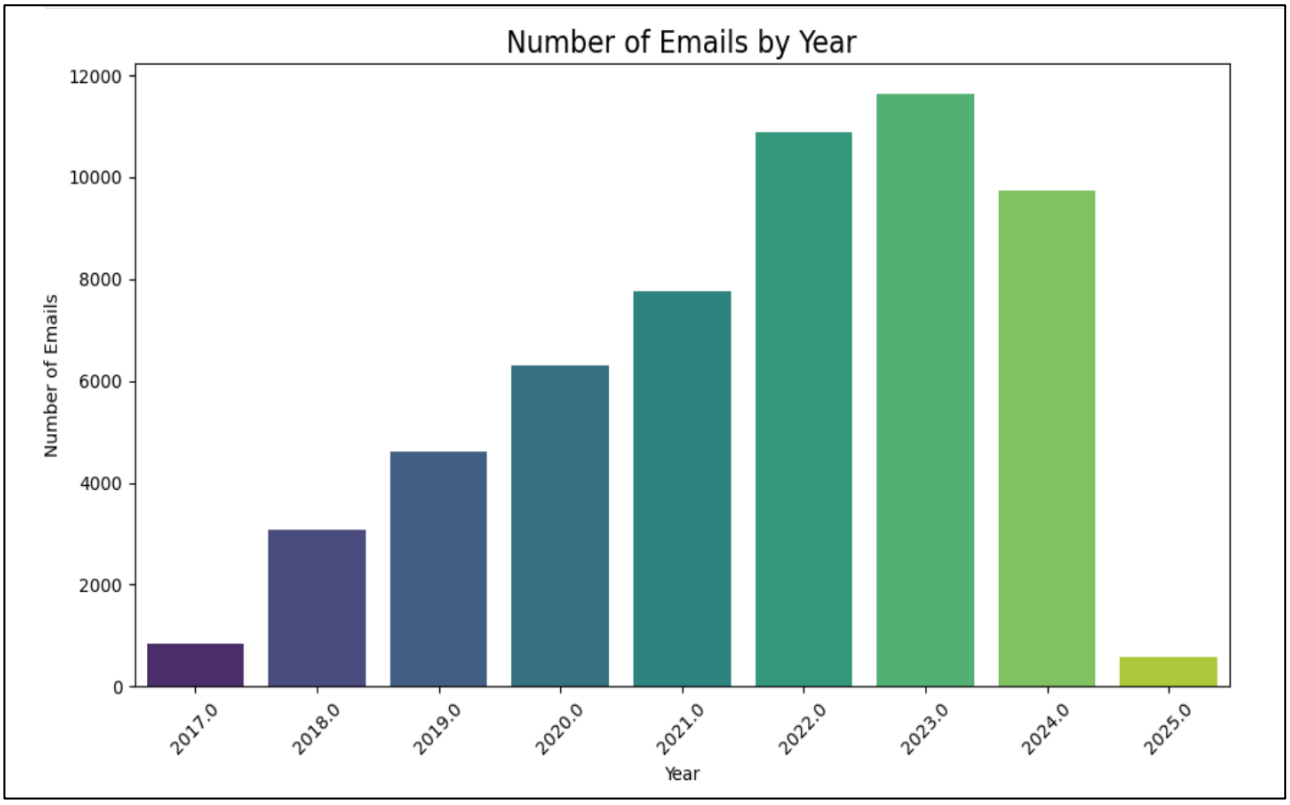
The chart shows that most emails have short body lengths, with a large number containing fewer than 1,000 characters. There is a sharp drop in frequency as the body length increases, indicating that long emails are less common. A few emails have exceptionally large body lengths, exceeding 30,000 characters, which could be detailed reports or spam. The distribution is right-skewed, meaning that while short emails are typical, there are some extreme outliers.

7.2.3 Email Subject Line Word Cloud Analysis :



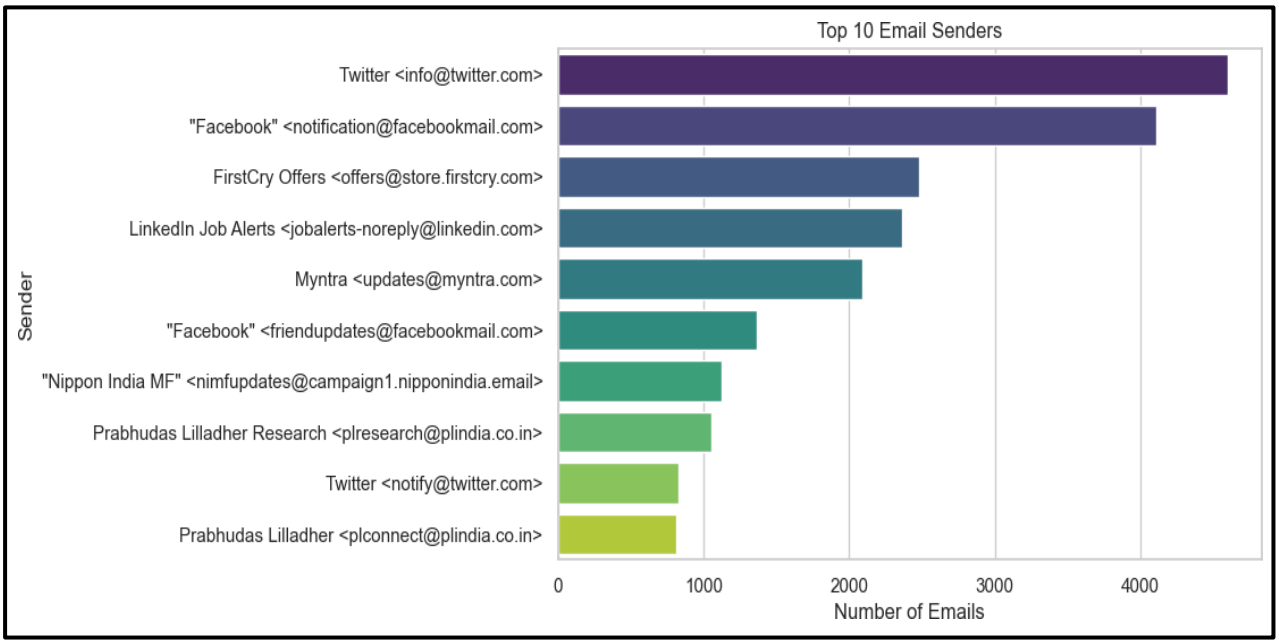
The word cloud highlights the most frequent words appearing in email subject lines, with larger words indicating higher frequency. Words like "Flat," "friend," "Facebook," "notification," and "Sanjay Patil" appear prominently, suggesting repeated topics or sources. Social media-related terms such as "Facebook," "Twitter," and "tagged" indicate a high number of social notification emails. There are also professional and financial terms like "Data Analyst," "Funds," "Invest," and "Credit Card," pointing to job or investment-related emails.

7.2.4 Yearly Email Volume Trend Analysis (2017-2025) :



The chart shows a steady increase in email volume from 2017 to 2023, peaking in 2023 with over 11,000 emails. This trend reflects growing digital communication, possibly due to increased online activity, subscriptions, and promotions. A slight drop is observed in 2024, but the volume still remains high compared to earlier years. Email data for 2025 appears very low, likely due to partial data collection up to the current date. This yearly trend analysis helps understand the growth of email usage over time and supports planning for storage or filtering solutions.

7.2.5 Top Email Sources and Their Distribution :



The chart shows that Twitter and Facebook are the top two sources of emails, each sending over 4,000 messages. FirstCry Offers and LinkedIn Job Alerts are also significant senders, indicating a mix of promotional and professional content. Other frequent senders include Myntra, Nippon India Mutual Fund, and Prabhudas Lilladher, reflecting marketing and financial updates. Most of these emails appear to be automated notifications or newsletters rather than personal communication.

- **Email Volume Growth and Management Trends (2017-2025) :**

	Year	Email_Count
0	2017.0	837
1	2018.0	3081
2	2019.0	4602
3	2020.0	6294
4	2021.0	7749
5	2022.0	10877
6	2023.0	11644
7	2024.0	9750
8	2025.0	582

A small drop is observed in 2024 with 9,750 emails, yet the volume remains significantly high. The count for 2025 is much lower (582) as the year is still in progress and data is partial. Overall, the trend shows significant growth in email volume over time, highlighting the need for effective email management strategies.

- **Monthly Email Traffic Analysis :**

	Month	Email_Count
0	January	4684
1	February	4294
2	March	4026
3	April	3948
4	May	4899
5	June	4557
6	July	4957
7	August	5157
8	September	4866
9	October	5029
10	November	4128
11	December	4871

Email traffic is relatively consistent across months, with August (5157) and October (5029) receiving the highest number of emails. The lowest email volume is seen in April (3948), possibly due to seasonal or holiday factors. Mid-year months like May, June, and July also maintain high activity, showing steady communication. Even December, despite holiday seasons, has a high email count (4871), indicating year-end notifications. Overall, there's no extreme fluctuation, suggesting continuous email activity throughout the year.

- **Weekly Email Activity Patterns :**

	Day_of_Week	Email_Count
0	Monday	8228
1	Tuesday	8052
2	Wednesday	8281
3	Thursday	8629
4	Friday	8583
5	Saturday	6901
6	Sunday	6742

Email activity peaks on Thursday (8629) and Friday (8583), showing strong end-of-week communication. Wednesday (8281) and Monday (8228) also see high traffic, indicating a steady mid-week workflow. Tuesday (8052) is slightly lower but still part of the busy workweek trend. Email volume drops significantly on Saturday (6901) and is lowest on Sunday (6742), reflecting weekend slowdowns. This pattern suggests most emails are sent during standard business days, with a decline over the weekend.

DATA ANALYSIS

8.1 Machine Learning Algorithm with outputs :

8.1.1 Naive Bayes Classifier:

The Naive Bayes classifier showed strong performance in identifying Spam emails (95% recall) but struggled with other categories like Primary and Social, leading to a high false positive rate for Spam. With an overall accuracy of 63%, it proves to be a fast and effective baseline for spam detection but lacks precision in multi-class email classification. Improving the feature set and addressing class imbalances could enhance its performance in future iterations. Naive Bayes is like a fast, simple rule-based filter good for spam, but weak in other category.

```
Naive Bayes Results:
                precision    recall  f1-score   support

       Primary       0.62      0.13      0.22      5164
     Promotion       0.74      0.41      0.53       746
        Social       0.65      0.10      0.18       587
          Spam       0.62      0.95      0.75      9747
       Updates       0.69      0.12      0.21       387

      accuracy                           0.63     16631
     macro avg       0.66      0.34      0.38     16631
  weighted avg       0.63      0.63      0.54     16631

[[ 680    6    6 4469    3]
 [  51  305    2  387    1]
 [  39    2   60  486    0]
 [ 300   98   24 9307   18]
 [  23    0    0  316   48]]
```

Metric	Value (%)
Primary Accuracy	62
Promotion Accuracy	74
Social Accuracy	65
Spam Accuracy	62
Updates Accuracy	69
Overall Accuracy	63
Overall Precision	66
Overall Recall	34

8.1.2 Logistic Regression:

The Logistic Regression model achieved a high overall accuracy of 89%, performing especially well in identifying Spam and Primary emails with strong precision and recall. However, it struggled with categories like Promotion, Social, and Updates, likely due to overlapping features or limited representation. Despite a convergence warning, the model delivered reliable results, showing that with enhanced preprocessing or more advanced features, its classification performance across all categories can be further improved. Logistic Regression is more like a smart assistant slower, but much more accurate in sorting emails into the right box.

```
Logistic Regression Results:
                precision    recall  f1-score   support

       Primary       0.78      0.96      0.86      5164
     Promotion       0.79      0.41      0.54       746
        Social       0.55      0.14      0.22       587
          Spam       0.97      0.96      0.96      9747
       Updates       0.61      0.14      0.23       387

      accuracy                           0.89     16631
     macro avg       0.74      0.52      0.56     16631
  weighted avg       0.88      0.89      0.87     16631

[[4977    5   15  159    8]
 [ 374  309   10   47    6]
 [ 445    6   80   51    5]
 [ 267   66   41 9356   17]
 [ 293    6    0   32   56]]
```

Metric	Value (%)
Primary Accuracy	78
Promotion Accuracy	79
Social Accuracy	55
Spam Accuracy	97
Updates Accuracy	61
Overall Accuracy	89
Overall Precision	74
Overall Recall	52

8.1.3 SVM (Support Vector Machine):

The SVM model demonstrated strong performance with an overall accuracy of 87%, particularly excelling in Spam and Updates classification. Its high precision in Spam detection (96%) is notable. However, it struggled with Social emails, leading to a lower recall for that category. With better feature engineering or class balancing, SVM could improve its performance on multi-class email classification. SVM is like a reliable, sharp classifier, great at detecting clear distinctions, but needs some fine-tuning to handle more nuanced categories.

```
SVM Results:
                precision    recall  f1-score   support

       Primary       0.74      0.96      0.83      5164
     Promotion       0.84      0.10      0.19       746
        Social       0.76      0.11      0.20       587
          Spam       0.96      0.95      0.95      9747
       Updates       0.92      0.11      0.20       387

      accuracy                           0.87     16631
     macro avg       0.84      0.45      0.47     16631
  weighted avg       0.88      0.87      0.84     16631

[[4957    1    0  206    0]
 [ 573   78    1   91    3]
 [ 435    3   66   82    1]
 [ 446   11   20 9270    0]
 [ 315    0    0   28   44]]
```

Metric	Value (%)
Primary Accuracy	74
Promotion Accuracy	84
Social Accuracy	76
Spam Accuracy	96
Updates Accuracy	92
Overall Accuracy	87
Overall Precision	84
Overall Recall	45

8.1.4 KNN (K-Nearest Neighbours):

KNN delivered an impressive accuracy of 91%, with strong performance in Spam detection and relatively high recall for other categories. It performed well across most classes but had some issues with misclassifying Social emails. KNN is a simple yet effective model, especially for well-defined categories, but its performance could benefit from addressing class imbalances and refining the distance metric.

```
KNN Results:
              precision    recall  f1-score   support

     Primary       0.81      0.98      0.89      5164
   Promotion       0.80      0.46      0.58       746
      Social       0.72      0.25      0.37       587
        Spam       0.98      0.96      0.97      9747
     Updates       0.89      0.49      0.63       387

    accuracy                           0.91     16631
   macro avg       0.84      0.63      0.69     16631
weighted avg       0.91      0.91      0.90     16631

[[5047    9    5  103    0]
 [ 355  340   10   35    6]
 [ 393    1  145   45    3]
 [ 250   73   40 9369   15]
 [ 176    0    2   19  190]]
```

Metric	Value (%)
Primary Accuracy	81
Promotion Accuracy	80
Social Accuracy	72
Spam Accuracy	98
Updates Accuracy	89
Overall Accuracy	91
Overall Precision	84
Overall Recall	63

8.1.5 Random Forest:

Random Forest outperformed the other models with an accuracy of 92%, showing exceptional performance in classifying Spam and Updates, while maintaining solid results across all categories. Its ability to handle large, complex datasets and provide reliable classification makes it an excellent choice for this task. However, further tuning could improve its recall for categories with fewer instances.

Random Forest is like a strong, dependable decision-maker, handling complexity well but with room for fine-tuning in edge cases.

```
              precision    recall  f1-score   support

     Primary       0.83      0.97      0.89      5164
   Promotion       0.85      0.47      0.61       746
      Social       0.88      0.26      0.40       587
        Spam       0.97      0.97      0.97      9747
     Updates       0.96      0.52      0.67       387

    accuracy                           0.92     16631
   macro avg       0.90      0.64      0.71     16631
weighted avg       0.92      0.92      0.91     16631

[[5029    1    0  134    0]
 [ 331  352    3   55    5]
 [ 374    0  154   59    0]
 [ 174   59   18 9492    4]
 [ 166    1    0   19  201]]
```

Metric	Value (%)
Primary Accuracy	83
Promotion Accuracy	85
Social Accuracy	88
Spam Accuracy	97
Updates Accuracy	96
Overall Accuracy	92
Overall Precision	89
Overall Recall	63

8.1.6 Decision Tree:

The Decision Tree model achieved a good overall accuracy of 90%, excelling in Spam detection, but struggled with Social emails. It tended to overfit the training data, leading to lower performance on unseen data. Decision Tree is a quick and interpretable model, but its simplicity can lead to poor generalization if not properly tuned. It's like a decisive helper who makes quick choices but sometimes misses the subtle details.

```
Decision Tree Results:
              precision    recall  f1-score   support

     Primary       0.79      0.98      0.87      4728
   Promotion       0.79      0.52      0.63       689
      Social       0.78      0.29      0.42       612
        Spam       0.98      0.96      0.97      9888
     Updates       0.90      0.53      0.66       714

    accuracy                           0.90     16631
   macro avg       0.85      0.65      0.71     16631
weighted avg       0.91      0.90      0.90     16631

Confusion Matrix:
 [[4611   14   10   80   13]
 [ 285  357    9   33    5]
 [ 407    3  177   22    3]
 [ 253   75   26 9512   22]
 [ 316    4    4   13  377]]
```

Metric	Value (%)
Primary Accuracy	79
Promotion Accuracy	79
Social Accuracy	42
Spam Accuracy	96
Updates Accuracy	66
Overall Accuracy	90
Overall Precision	85
Overall Recall	65

8.1.7 AdaBoost:

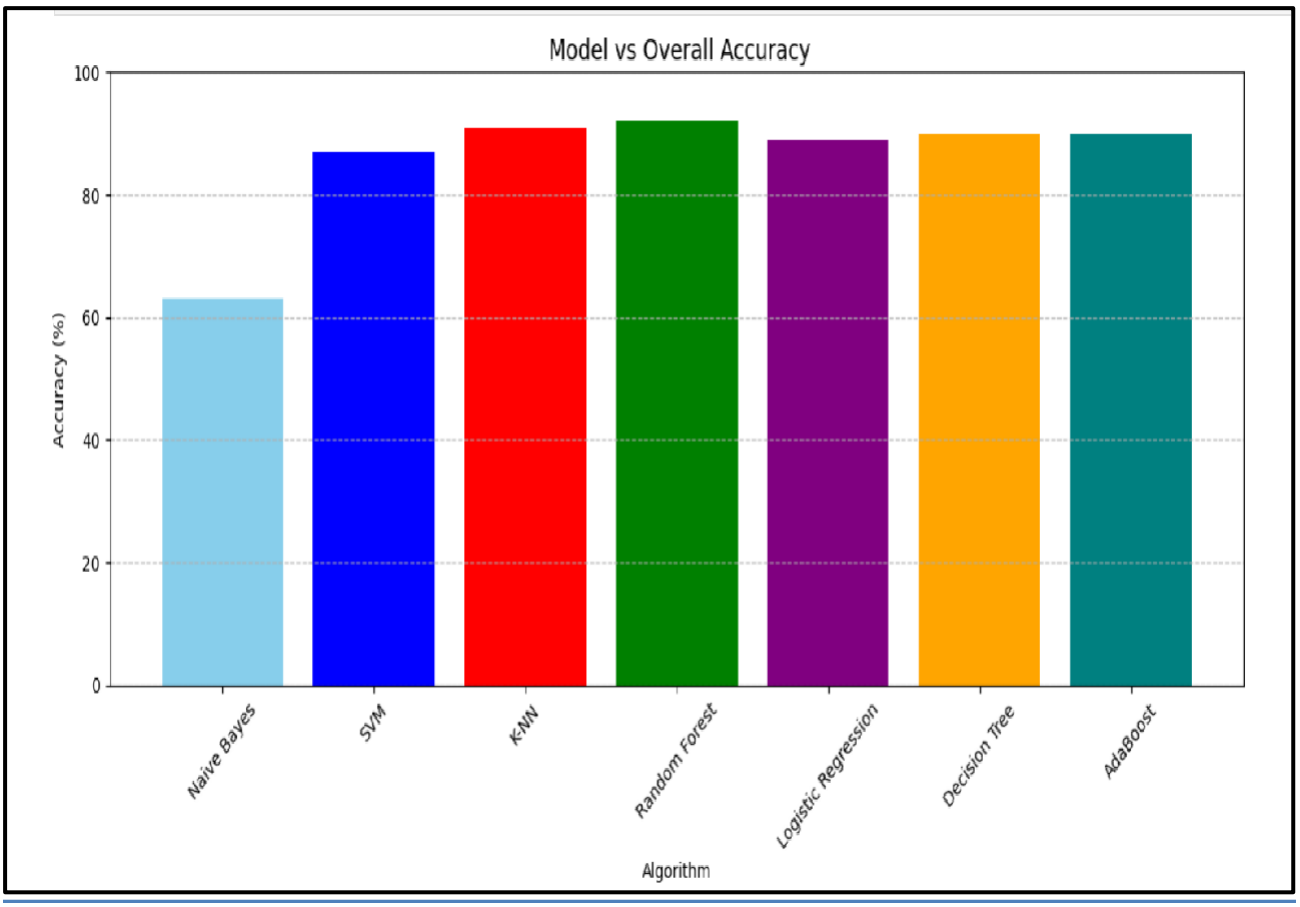
AdaBoost performed with an overall accuracy of 90%, excelling in Spam classification, but had difficulty with Social emails, showing a low recall for that category. It is a powerful ensemble method that works well with weak learners, but its performance on underrepresented classes like Social could be improved with better feature representation or resampling techniques.

AdaBoost is like a team of helpers working together, doing great on the big tasks, but struggling when the data is more complicated or imbalanced.

```
AdaBoost Results:
                precision    recall  f1-score   support

       Primary       0.82      0.98      0.89      5164
     Promotion       0.80      0.45      0.58       746
        Social       0.51      0.13      0.21       587
          Spam       0.97      0.96      0.96      9747
       Updates       0.91      0.45      0.60       387

      accuracy                           0.90     16631
     macro avg       0.80      0.60      0.65     16631
  weighted avg       0.90      0.90      0.89     16631

Confusion Matrix:
 [[5051    1   27   83    2]
 [ 330  338    7   68    3]
 [ 361    9   79  138    0]
 [ 248   77   43 9367   12]
 [ 184    0    0   28  175]]
```

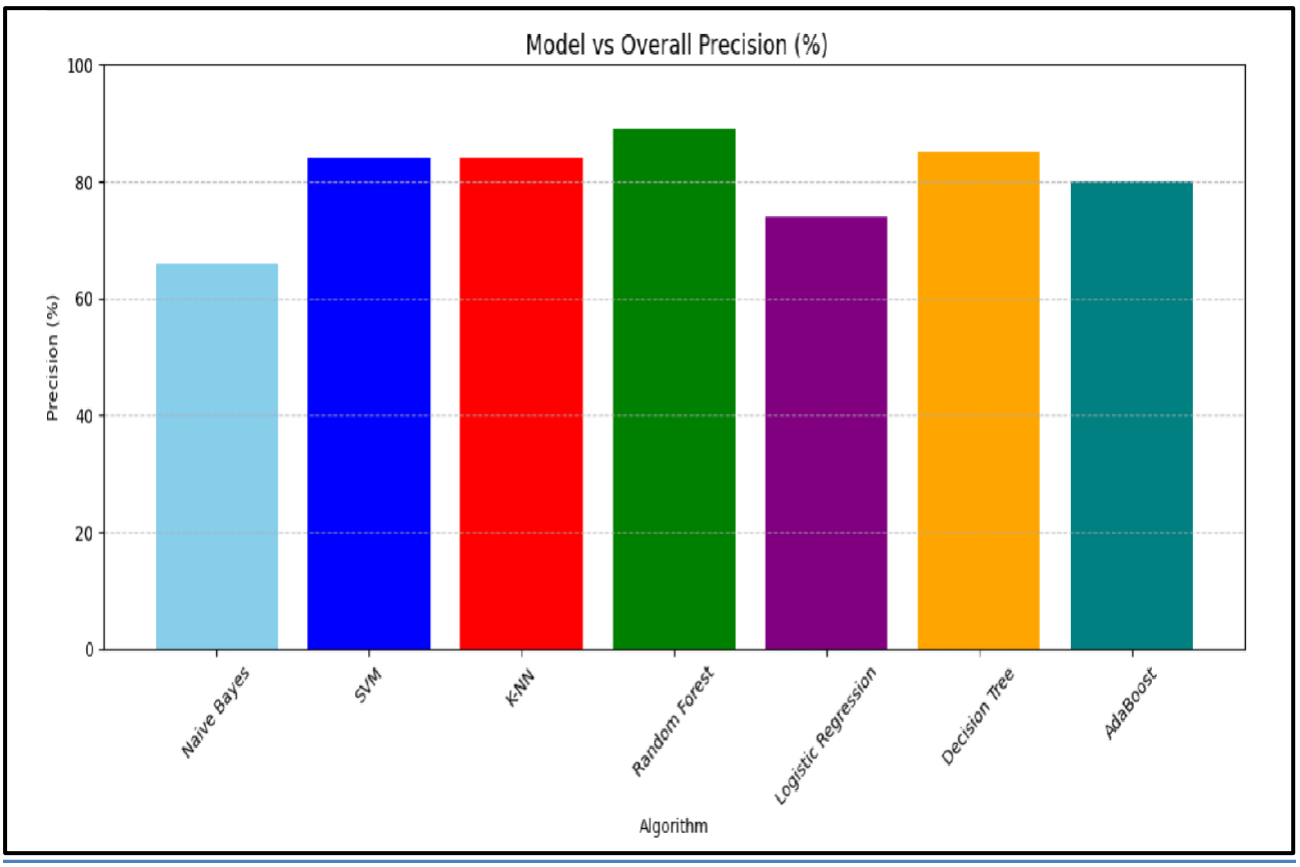
Metric	Value (%)
Primary	82
Promotion	80
Social	21
Spam	96
Updates	60
Overall Accuracy	90
Overall Precision	80
Overall Recall	60

➢ Model Comparison Based on Overall Accuracy (%) :



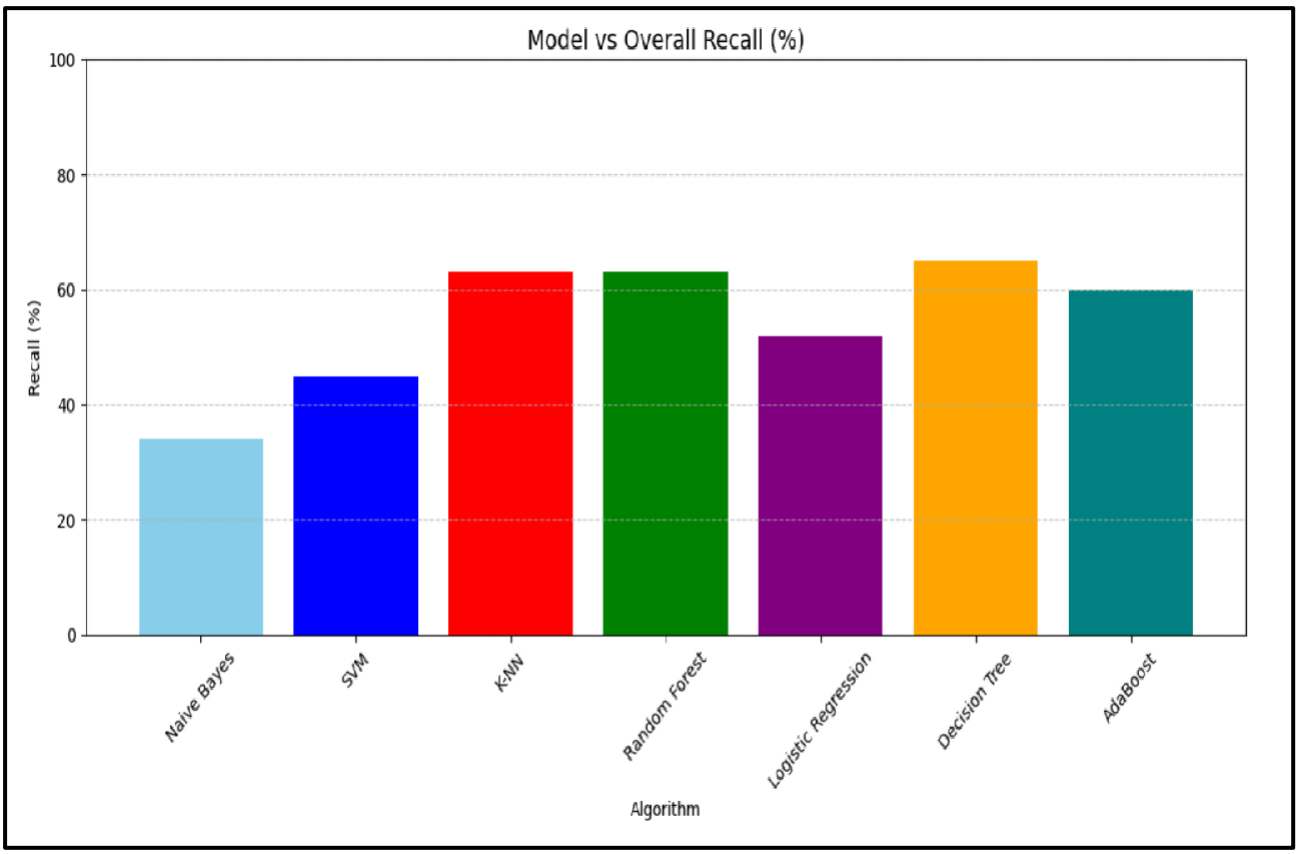
The graph shows that Random Forest achieved the highest accuracy among all models, making it the most reliable for classifying emails correctly. KNN, Decision Tree, AdaBoost, and Logistic Regression also performed very well with high accuracy. SVM gave good results but was slightly behind these models. Naive Bayes had the lowest accuracy, indicating it made the most mistakes in classification. Overall, Random Forest is the best choice, while Naive Bayes is less effective for this task.

➢ Model Comparison Based on Overall Precision (%) :



The graph shows that Random Forest has the highest overall precision, meaning it makes the most correct predictions with fewer false alarms. SVM, KNN, and Decision Tree also performed well, showing strong and reliable precision. AdaBoost gave decent results but was slightly behind the top models. Logistic Regression had lower precision, indicating more incorrect predictions. Naive Bayes had the lowest precision, making it the least reliable in terms of predicting the correct email categories.

➢ Model Comparison Based on Overall Recall (%) :



The graph shows that the Decision Tree model had the highest overall recall, meaning it was best at correctly identifying relevant emails. KNN and Random Forest also had strong recall, capturing most of the correct email categories. AdaBoost and Logistic Regression performed moderately well, but Logistic Regression missed more relevant emails. SVM had lower recall, and Naive Bayes had the lowest, indicating it failed to correctly detect many emails in their actual categories.

➢ Evaluation Metrics of Various Machine Learning Models for Email Classification Tasks :

[57]:

	Algorithm	Primary (%)	Promotion (%)	Social (%)	Spam (%)	Updates (%)	Overall Accuracy (%)	Overall Precision (%)	Overall Recall (%)
0	Naive Bayes	62	74	65	62	69	63	66	34
1	SVM	74	84	76	96	92	87	84	45
2	K-NN	81	80	72	98	89	91	84	63
3	Random Forest	83	85	88	97	96	92	89	63
4	Logistic Regression	78	79	55	97	61	89	74	52
5	Decision Tree	79	79	42	96	66	90	85	65
6	AdaBoost	82	80	21	96	60	90	80	60

This table compares different machine learning algorithms based on how well they classify emails into categories like Primary, Promotion, Social, Spam, and Updates. Random Forest performed the best overall, with the highest accuracy (92%) and strong precision and recall. KNN also showed excellent results, especially in detecting Spam (98%) and Updates (89%). SVM was good at identifying Spam and Updates but had lower recall.

Logistic Regression had high accuracy (89%) but struggled with the Social category (55%). Decision Tree performed well overall but was weak in identifying Social emails. Naive Bayes, while fast, had the lowest performance across all major metrics. AdaBoost was decent but had a major drop in detecting Social emails (only 21%).

CONCLUSION

1. Using NLP techniques like tokenization, stemming, and TF-IDF improved the models ability to analyze and classify email content. These methods enhanced feature quality, leading to better spam detection accuracy.
2. The keyword based email classification system effectively categorizes emails into Spam**,** Promotion**,** Social**,** Updates, and Primary. This approach provides a fast and simple method for sorting emails, though future improvements could include refining the keyword lists or incorporating machine learning for better accuracy.
3. After evaluating multiple machine learning models, Random Forest achieved the highest accuracy of 92%, making it the most effective model for email spam detection. It consistently outperformed other algorithms in both precision and recall across multiple email categories.
4. The Naive Bayes model has fast and simple, with strong spam recall (95%), but lowest overall accuracy (63%). Struggles significantly with non-spam categories,
5. The Logistic Regression model has high accuracy (89%) with reliable performance on Spam and Primary emails. Weaker on Social and Promotions due to overlapping features
6. The SVM (Support Vector Machine) model has accurate in Spam (96%) and Updates classification, achieving 87% overall accuracy. Requires better tuning for nuanced classes like Social.
7. The KNN model has achieved 91% accuracy with excellent Spam (98%) and Updates classification. Performance drops slightly when dealing with similar or overlapping data points.
8. The Random Forest model has best performer with 92% accuracy, strong across all categories, especially Spam and Updates. handles complex data well, though can be improved for underrepresented classes.
9. The Decision Tree model has easy to interpret and fast, with good accuracy (90%) in Spam detection. Tends to overfit and performs poorly on Social emails.
10. The AdaBoost model has reaches 90% accuracy and excels at Spam classification. Struggles with imbalanced classes like Social (21% recall), needing better data handling.

SCOPE

Scope :

- Multilingual and Multimodal Spam Detection the project will explore detecting spam in emails written in different languages and incorporating non-textual features like email headers and metadata (e.g. sender reputation) to enhance detection accuracy.
- Adaptive Spam Detection Model the focus will be on creating a model that adapts to new spam tactics over time, utilizing techniques like semi-supervised learning to continuously improve the system as spam evolves.
- This task focuses on deleting emails from the Promotion and Social categories that were received on specific dates, helping to reduce clutter and reclaim storage.
- It involves filtering emails based on category and date, and optionally backing them up before permanent deletion for safety and reference.

LIMITATIONS

Limitations :

- Dataset Limitations the performance of the models will be constrained by the quality and size of the labeled dataset available for training and testing. Any bias or imbalance in the dataset may affect the model's generalizability.

- Feature Extraction some NLP techniques (e.g. stop word removal, stemming) may inadvertently discard useful information, which could affect the model's ability to accurately differentiate between spam and legitimate emails.

- Computational Complexity & Model Interpretability more advanced models like Random Forest or SVM may require higher computational resources, making real-time processing in large scale systems challenging.

REFERENCE

1. Subhashini, G., & Mahalakshmi, G. (2024). Advanced SMS Spam Detection Using Integrated Feature Extraction. 4th International Conference on Smart Systems and Inventive Technology (ICSSIT). IEEE.
2. Alharbi, F., & Lee, S. (2020). Adaptive spam detection framework using NLP and classification-based techniques. Journal of Information Security and Applications, 54, 102581.
3. Zhang, Y., Jin, R., & Zhou, Z. (2017). Understanding NLP-based spam detection using keyword and semantic cues. Knowledge-Based Systems, 126, 70–84.
4. Ravindran, P., & Chitrakala, S. (2021). Hybrid Approach for Multi-Class Spam Filtering Using TF-IDF and BERT. Expert Systems with Applications,
5. Olayemi, S., & Weli, V. (2023). Natural Language Processing for Email Spam Detection Using Pretrained Embeddings. Data & Knowledge Engineering, 142, 101121.
6. Kumar, A., & Singh, R. (2023). Spam Email Detection Using Natural Language and Machine Learning Classifiers. International Journal of Information Technology.
7. Pillai, A., & Nair, R. (2021). Spam Classification Using NLP and Deep Learning. Amrita Vishwa Vidyapeetham, Coimbatore.
8. Saxena, R., & Sharma, A. (2022). Hybrid Feature Selection for Email Spam Filtering Using Keyword Extraction. Indian Journal of Computer Science and Engineering, 13(2), 118–126.
9. Singh, H., & Kaur, R. (2021). Feature-Rich Multi-Class Spam Filter for Emails Using TF-IDF and NLP. National Conference on Recent Innovations in Science and Technology, Punjab.
10. Joshi, R., & Mehta, S. (2023). Email Spam Filtering Based on POS-Tag Enrichment and Keyword Boosting. AICTE Sponsored Project Report, Gujarat Technological University.

APPENDIX

Code :

- **Python Code for NLP method with Data Cleaning & Preprocessing :**

```
import pandas as pd
import numpy as np
import re
import string
# Load dataset with proper encoding
df = pd.read_excel(r'E:\OM Research Project\Combined_Data.xlsx', engine='openpyxl')
df.dropna(inplace=True)
def preprocess_text(text):
    text = text.lower()
    text = re.sub(f"[{string.punctuation}]", "", text) # Remove punctuation
    return text
df.head(4)
df['processed_text'] = df['Body'].apply(preprocess_text)
df.shape
df.isnull().sum()
df.info()
df.dtypes
# Ensure TensorFlow is installed
%pip install tensorflow==2.10.0

# Ensure TensorFlow is imported correctly
 try:
        from tensorflow.keras.preprocessing.text import Tokenizer
        from tensorflow.keras.preprocessing.sequence import pad_sequences

        # Tokenization for LSTM
        tokenizer = Tokenizer(num_words=5000)
        tokenizer.fit_on_texts(df['processed_text'])
        sequences = tokenizer.texts_to_sequences(df['processed_text'])
        X_lstm = pad_sequences(sequences, maxlen=50)
except ImportError as e:
        print("Error importing TensorFlow:", e)
        print("Please ensure TensorFlow is installed correctly and is compatible with your
system.")
# Load dataset with proper encoding
df = pd.read_excel(r'E:\OM Research Project\Combined_Data.xlsx')

# Print dataset size
print("Dataset Shape:", df.shape)  # Should print (578, X) if all rows are loaded
print(df.head())  # Show first few rows
# Check if the dataset is loading properly
print("Total Rows in Dataset:", df.shape[0])
```

```
# Define spam keywords
spam_keywords = ["win", "free", "winner", "click", "prize", "money", "urgent"]

# Function to classify emails
def classify_email(text):
    if pd.isna(text): # Handle missing values
        return "ham"
    text = text.lower()  # Convert to lowercase
    if any(word in text for word in spam_keywords):
        return "spam"
    return "ham"

# Apply classification to ALL rows (not just one row)
df["category"] = df.apply(lambda row: classify_email(str(row["Subject"]) + " " +
str(row["Body"])), axis=1)

# Convert category to numerical (ham=0, spam=1)
df["category"] = df["category"].map({"ham": 0, "spam": 1})

# Verify all rows are classified
print(df[["Subject", "Body", "category"]].head())  # Show first few rows
print("Total Rows Classified:", df.shape[0])  # Should be 578+ rows

# Save the classified dataset
df.to_csv("classified_emails.csv", index=False)

df.category.value_counts()
```

- **Data Visualization :**

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

# Get the top 10 senders
top_senders = df['From'].value_counts().head(10)

# Plot
plt.figure(figsize=(8, 5))
sns.barplot(x=top_senders.values, y=top_senders.index, palette="viridis")
plt.title("Top 10 Email Senders")
plt.xlabel("Number of Emails")
plt.ylabel("Sender")
plt.show()

# Generate word cloud
subject_text = " ".join(df['Subject'].dropna().tolist())
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(subject_text)



# Plot
```

```
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title("Word Cloud of Email Subjects")
plt.axis("off")
plt.show()
No description has been provided for this image
# Compute body length
df['Body_Length'] = df['Body'].fillna("").apply(len)

# Plot
plt.figure(figsize=(8, 5))
sns.histplot(df['Body_Length'], bins=30, kde=True, color="teal")
plt.title("Distribution of Email Body Lengths")
plt.xlabel("Body Length (Characters)")
plt.ylabel("Frequency")
plt.show()
No description has been provided for this image
# Convert Date column to datetime
df['Date'] = pd.to_datetime(df['Date'], errors='coerce', utc=True)
df['Hour'] = df['Date'].dt.hour

# Plot
plt.figure(figsize=(8, 5))
sns.countplot(x='Hour', data=df, palette="coolwarm")
plt.title("Distribution of Emails by Hour")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Emails")
plt.show()

# Function to automatically label emails based on keywords
def auto_label(subject, body):
    # Convert to lowercase for uniformity
    subject = str(subject).lower()
    body = str(body).lower()


    # Spam keywords
    spam_keywords = ["unsubscribe", "win", "click here", "free", "claim", "urgent", "limited
offer",
                     "winner", "guaranteed", "risk-free", "exclusive", "money", "claim your prize",
                     "offer expires", "free trial", "credit card", "cash prize", "unsecured", "no payment
required"]
    if any(keyword in subject or keyword in body for keyword in spam_keywords):
        return "Spam"

    # Promotion keywords
    promo_keywords = ["offer", "deal", "discount", "sale", "coupon", "promo", "clearance",
"bargain",
                      "flash sale", "hot deal", "special offer", "limited time", "today only", "big discount",
                      "buy now", "shop today", "online exclusive", "exclusive deal", "seasonal sale"]
```

```
    if any(keyword in subject or keyword in body for keyword in promo_keywords):
        return "Promotion"

    # Social keywords
    social_keywords = ["friend request", "like", "comment", "follow", "connection", "new
follower",
                       "your post", "social invite", "message request", "share your status", "new like",
                       "new comment", "friend suggestion", "tagged in photo", "social media update",
                       "profile update", "join the group"]
    if any(keyword in subject or keyword in body for keyword in social_keywords):
        return "Social"

    # Update keywords
    update_keywords = ["update", "newsletter", "report", "summary", "digest", "alert", "news",
                       "update notification", "policy update", "account update", "system update",
"upgrade",
                       "change", "announcement", "new feature", "release notes", "latest news"]
    if any(keyword in subject or keyword in body for keyword in update_keywords):
        return "Updates"

    # Default to Primary (Ham)
    return "Primary"

# Apply the automatic labeling function
df['Label'] = df.apply(lambda row: auto_label(row['Subject'], row['Body']), axis=1)

# Display some labeled data samples to verify
df[['From', 'Subject', 'Body', 'Label']].head(20)
# Display the updated dataset with the new labels
# Create a new dataframe with the new column 'Label'
df1 = df.copy()

# Display the updated dataset with the new labels
df1.head()

# First, make sure 'Year' column is extracted
df['Year'] = df['Date'].dt.year

# Now group by year and count
emails_per_year = df.groupby('Year').size().reset_index(name='Email_Count')

# Display the result
print(emails_per_year)

# Extract month name
df['Month'] = df['Date'].dt.month_name()

# Group by month and count
emails_per_month = df.groupby('Month').size().reset_index(name='Email_Count')
```

```
# Sort by calendar month order (optional)
from pandas.api.types import CategoricalDtype
month_order = CategoricalDtype(
    ['January', 'February', 'March', 'April', 'May', 'June',
     'July', 'August', 'September', 'October', 'November', 'December'],
    ordered=True
)
df['Month'] = df['Month'].astype(month_order)
emails_per_month = df.groupby('Month').size().reset_index(name='Email_Count')

# Display result
print(emails_per_month)

# Extract the day of the week
df['Day_of_Week'] = df['Date'].dt.day_name()

# Group by day and count
emails_by_day = df.groupby('Day_of_Week').size().reset_index(name='Email_Count')

# Optional: Sort days in natural week order
from pandas.api.types import CategoricalDtype
day_order = CategoricalDtype(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'],
    ordered=True
)
df['Day_of_Week'] = df['Day_of_Week'].astype(day_order)
emails_by_day = df.groupby('Day_of_Week').size().reset_index(name='Email_Count')

# Display result
print(emails_by_day)

plt.figure(figsize=(10, 6))
sns.barplot(data=emails_per_year, x='Year', y='Email_Count', palette='viridis')

plt.title('Number of Emails by Year', fontsize=16)
plt.xlabel('Year')
plt.ylabel('Number of Emails')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

df['Label'].value_counts()

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stopwords.words('english')


import pandas as pd
```

```
import re

def preprocess_text(text):
    if pd.isna(text):
        return ""
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\W', ' ', text)  # Remove special characters
    text = re.sub(r'\s+', ' ', text).strip() # Remove extra spaces
    return text

df['processed_text'] = df['Body'].apply(preprocess_text)  # Ensure correct column name
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform(df['processed_text'])
X.shape
Y = df['Label'].values
Y

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.30,random_state=40)
```

- **By Naive Bayes :**

```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(X_train, Y_train)

# Predicting the Test set results
Y_pred = classifier.predict(X_test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix(Y_test, Y_pred)
print("Naive Bayes Results:\n", classification_report(Y_test, Y_pred))

# Compute confusion matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

- **By Logistic Regression :**

```
from sklearn.linear_model import LogisticRegression  # Import the missing module
# Logistic Regression Model
lr_model = LogisticRegression()
lr_model.fit(X_train, Y_train)

# Evaluation
Y_pred = lr_model.predict(X_test)
print("Logistic Regression Results:\n", classification_report(Y_test, Y_pred))

# Compute confusion matrix
```

```
cm = confusion_matrix(Y_test, Y_pred)
print(cm)

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
model.fit(X_train, Y_train)

# Step 7: Evaluate Model
Y_pred = model.predict(X_test)
print(classification_report(Y_test,  Y_pred))

cm = confusion_matrix(Y_test, Y_pred)
print(cm)
```

- **By Support Vector Machine :**

```
from sklearn.svm import SVC

# Support Vector Machine Model
svm_model = SVC()
svm_model.fit(X_train, Y_train)

Y_pred_svm = svm_model.predict(X_test)
print("SVM Results:\n", classification_report(Y_test, Y_pred_svm))

# Compute confusion matrix
cm_svm = confusion_matrix(Y_test, Y_pred_svm)
print(cm_svm)
```

- **By K-Nearest Neighbors :**

```
from sklearn.neighbors import KNeighborsClassifier

# K-Nearest Neighbors Model
knn_model = KNeighborsClassifier() # You can tune the number of neighbors
knn_model.fit(X_train, Y_train)

# Evaluation
Y_pred_knn = knn_model.predict(X_test)
print("KNN Results:\n", classification_report(Y_test, Y_pred_knn))

# Compute confusion matrix
cm_knn = confusion_matrix(Y_test, Y_pred_knn)
print(cm_knn)
```

- **By Decision Tree :**

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Decision Tree Model
dt_model = DecisionTreeClassifier() # You can tune parameters like max_depth, criterion
dt_model.fit(X_train, Y_train)
```

```
# Evaluation
Y_pred_dt = dt_model.predict(X_test)
print("Decision Tree Results:\n", classification_report(Y_test, Y_pred_dt))

# Compute confusion matrix
cm_dt = confusion_matrix(Y_test, Y_pred_dt)
print("Confusion Matrix:\n", cm_dt)
```

- **By AdaBoost :**

```
from sklearn.ensemble import AdaBoostClassifier

# AdaBoost with Decision Tree
adaboost_model = AdaBoostClassifier()
adaboost_model.fit(X_train, Y_train)

# Predictions
Y_pred_adaboost = adaboost_model.predict(X_test)

# Evaluation
print("AdaBoost Results:\n", classification_report(Y_test, Y_pred_adaboost))
print("Confusion Matrix:\n", confusion_matrix(Y_test, Y_pred_adaboost))
```

- **Algorithm Comparison Table :**

```
import pandas as pd
from tabulate import tabulate

# Define data for different models
data = {
   "Algorithm": ["Naive Bayes", "SVM", "K-NN", "Random Forest", "Logistic Regression",
"Decision Tree", "AdaBoost" ],
   "Primary (%)": [62, 74, 81, 83, 78, 79, 82],
   "Promotion (%)": [74, 84, 80, 85, 79, 79, 80],
   "Social (%)": [65, 76, 72, 88, 55, 42, 21],
   "Spam (%)": [62, 96, 98, 97, 97, 96, 96],
   "Updates (%)": [69, 92, 89, 96, 61, 66, 60],
   "Overall Accuracy (%)": [63, 87, 91, 92, 89, 90, 90],
   "Overall Precision (%)": [66, 84, 84, 89, 74, 85, 80],
   "Overall Recall (%)": [34, 45, 63, 63, 52, 65, 60]
}

# Create DataFrame
df = pd.DataFrame(data)
df
```

- **Overall Algorithm Comparison :**

```
# Plot Overall Accuracy comparison
x = ["Naive Bayes", "SVM", "K-NN", "Random Forest", "Logistic Regression", "Decision Tree",
"AdaBoost"]
```

```
y = [63, 87, 91, 92, 89, 90, 90]

plt.figure(figsize=(12, 6))
plt.bar(x, y, color=['skyblue', 'blue', 'red', 'green', 'purple', 'orange', 'teal'])

plt.title(" Model vs Overall Accuracy", fontsize=14)
plt.xlabel("Algorithm")
plt.ylabel("Accuracy (%)")
plt.ylim(0, 100)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```
No description has been provided for this image
```
# Plot Overall Accuracy comparison
x = ["Naive Bayes", "SVM", "K-NN", "Random Forest", "Logistic Regression", "Decision Tree",
"AdaBoost"]
y = [66, 84, 84, 89, 74, 85, 80]

plt.figure(figsize=(12, 6))
plt.bar(x, y, color=['skyblue', 'blue', 'red', 'green', 'purple', 'orange', 'teal'])

plt.title(" Model vs Overall Precision (%)", fontsize=14)
plt.xlabel("Algorithm")
plt.ylabel("Precision (%)")
plt.ylim(0, 100)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```
No description has been provided for this image
```
# Plot Overall Accuracy comparison
x = ["Naive Bayes", "SVM", "K-NN", "Random Forest", "Logistic Regression", "Decision Tree",
"AdaBoost" ]
y = [34, 45, 63, 63, 52, 65, 60]

plt.figure(figsize=(12, 6))
plt.bar(x, y, color=['skyblue', 'blue', 'red', 'green', 'purple', 'orange', 'teal'])

plt.title("Model vs Overall Recall (%)", fontsize=14)
plt.xlabel("Algorithm")
plt.ylabel("Recall (%)")
plt.ylim(0, 100)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

- **Email Classify :**

```
import pandas as pd

# Sample email data (replace with your actual data or load from Excel)
data = {
    'Subject': [
        "Congratulations! You won a cash prize",
        "Flash Sale - 50% OFF today only!",
        "Your friend liked your post",
        "Weekly newsletter update",
        "Meeting scheduled for Monday"
    ],
    'Body': [
        "Click here to claim your prize. No payment required.",
        "Don't miss our hot deal! Limited time offer, shop now.",
        "John commented on your post. View it on your profile.",
        "Here is your weekly newsletter with all the latest news.",
        "Reminder: Team meeting is scheduled for 10 AM Monday."
    ]
}

df = pd.DataFrame(data)

# Define keyword lists
spam_keywords = [
    "unsubscribe", "win", "click here", "free", "claim", "urgent", "limited offer",
    "winner", "guaranteed", "risk-free", "exclusive", "money", "claim your prize",
    "offer expires", "free trial", "credit card", "cash prize", "unsecured",
    "no payment required"
]

promo_keywords = [
    "offer", "deal", "discount", "sale", "coupon", "promo", "clearance", "bargain",
    "flash sale", "hot deal", "special offer", "limited time", "today only",
    "big discount", "buy now", "shop today", "online exclusive", "exclusive deal",
    "seasonal sale"
]

social_keywords = [
    "friend request", "like", "comment", "follow", "connection", "new follower",
    "your post", "social invite", "message request", "share your status", "new like",
    "new comment", "friend suggestion", "tagged in photo", "social media update",
    "profile update", "join the group"
]

update_keywords = [
    "update", "newsletter", "report", "summary", "digest", "alert", "news",
    "update notification", "policy update", "account update", "system update",
    "upgrade", "change", "announcement", "new feature", "release notes", "latest news"
]

primary_keywords = [
```

```
    "meeting", "appointment", "invoice", "schedule", "reminder", "password",
    "receipt", "confirmation", "team", "project", "task", "follow-up"
]

# Function to classify emails
def classify_email(subject, body):
    if not isinstance(subject, str):
        subject = ''
    if not isinstance(body, str):
        body = ''

    content = (subject + ' ' + body).lower()

    if any(keyword in content for keyword in spam_keywords):
        return "Spam"
    elif any(keyword in content for keyword in promo_keywords):
        return "Promotion"
    elif any(keyword in content for keyword in social_keywords):
        return "Social"
    elif any(keyword in content for keyword in update_keywords):
        return "Updates"
    elif any(keyword in content for keyword in primary_keywords):
        return "Primary"
    else:
        return "Primary"  # default fallback

# Apply classification
df['Label'] = df.apply(lambda row: classify_email(row['Subject'], row['Body']), axis=1)

# Display results
print(df[['Subject', 'Body', 'Label']])
```