# TABLE OF CONTENTS

## ABSTRACT

Spam detection in electronic communication is a crucial task in ensuring user security and maintaining the integrity of digital communication platforms. Traditional rule-based filters are limited in adaptability and often fail to detect modern, evolving spam content effectively. This project presents an automated spam classification system using supervised machine learning algorithms to accurately differentiate between “spam” and “ham” (non-spam) messages based on textual content.

The system utilizes the SMS Spam Collection dataset containing 5,169 unique text samples, each labeled as spam or ham. Text data undergoes a comprehensive preprocessing pipeline involving lowercase conversion, token extraction using regular expressions, stop-word removal, and stemming through the PorterStemmer technique. These processed messages are then transformed into numerical feature vectors using the TF-IDF (Term Frequency–Inverse Document Frequency) method, which emphasizes significant words while penalizing frequently occurring ones.

A Support Vector Machine (SVM) classifier was employed as the primary model, and its performance was optimized through hyperparameter tuning using GridSearchCV. Parameters such as kernel type (“linear” and “rbf”), regularization constant (C = [1, 10, 100, 1000]), and gamma values ([1e-3, 1e-4]) were explored. The optimized model achieved an accuracy of **98.38%** on the test dataset, effectively identifying spam messages with high precision and recall.

To provide an interactive and user-friendly experience, the trained model was integrated into a Graphical User Interface (GUI) using Tkinter. The interface enables real-time text input and immediate classification feedback, allowing users to easily determine whether a message is spam or legitimate.

This automated spam detection system delivers a reliable, efficient, and scalable solution for filtering unwanted messages, significantly reducing manual monitoring time and minimizing the risk of exposure to malicious content while maintaining near-perfect accuracy.

CHAPTER-1

# INTRODUCTION

## 1.1 Problem Statement

- In the modern era of digital communication, Short Message Service (SMS) has become one of the most widely used methods of instant communication due to its simplicity, low cost, and accessibility. However, this popularity has also led to the rise of unsolicited and unwanted messages, commonly referred to as *spam*. These spam messages are not only an annoyance to users but also pose potential security risks, such as phishing attacks and the spread of malicious links.
- Traditional spam detection methods rely heavily on manual inspection, where individuals or administrators must read, identify, and report spam messages. This approach is inefficient, time-consuming, and prone to human error, especially in environments handling millions of messages daily. As communication volume continues to grow exponentially.
- Therefore, there is a pressing need for an **automated, scalable, and intelligent system** that can accurately classify messages as spam or ham in real-time. By leveraging the power of **machine learning (ML)** and **natural language processing (NLP)**, such a system can learn from patterns in text data and improve over time, providing robust and adaptive spam detection capabilities. This automation not only saves time but also enhances accuracy, consistency, and operational efficiency across messaging platforms.

## 1.2 Objectives

- In the modern era of digital communication, Short Message Service (SMS) has become one of the most widely used methods of instant communication due to its simplicity, low cost, and accessibility. However, this popularity has also led to the rise of unsolicited and unwanted messages, commonly referred to as *spam*. These spam messages are not only an annoyance to users but also pose potential security risks, such as phishing attacks and the spread of malicious links.

- Traditional spam detection methods rely heavily on manual inspection, where individuals or administrators must read, identify, and report spam messages.
- This approach is inefficient, time-consuming, and prone to human error. As communication volume continues to grow exponentially, it becomes impractical to manually filter spam from legitimate (ham) messages.
- Therefore, there is a pressing need for an **automated, scalable, and intelligent system** that can accurately classify messages as spam or ham in real-time. By leveraging the power of **machine learning (ML)** and **natural language processing (NLP)**, such a system can learn from patterns in text data and improve over time, providing robust and adaptive spam detection capabilities.

## 1.3 Significance

### Business Impact

In the context of businesses and telecommunication providers, spam messages can lead to customer dissatisfaction, brand mistrust, and even legal repercussions if not effectively managed. By deploying an automated spam detection system, organizations can achieve:

- **Significant reduction in manual review time**, cutting down from hours of manual labor to seconds of automated processing.
- **Operational cost savings**, as fewer human resources are required for monitoring and classification tasks.
- **Enhanced customer experience**, as users receive fewer spam messages, improving trust and satisfaction.
- **Uniform and accurate classification**, ensuring consistency across large datasets.
- **Real-time processing capability**, enabling the system to handle thousands of messages per second, which is crucial for modern messaging platforms and telecommunication networks.

## Technical Contribution

From a technical perspective, this project demonstrates the effective integration of Natural Language Processing (NLP) and Machine Learning (ML) in solving a domain-specific classification problem. The contributions include:

- Text Preprocessing and Normalization: Techniques such as lowercasing, punctuation removal, stopword elimination, and stemming/lemmatization ensure that the input data is standardized, reducing noise and improving model interpretability.
- Feature Engineering: Transforming unstructured text into structured numerical vectors using TF-IDF and CountVectorizer enables algorithms to learn meaningful relationships between words and their frequencies.
- Algorithmic Evaluation: Multiple classification models are tested and compared to determine the most efficient one for the given dataset, ensuring both accuracy and computational efficiency.
- Model Optimization: The use of hyperparameter tuning and cross-validation ensures that the final model achieves optimal performance and generalizes well to unseen data.
- Automation Pipeline: The system can be extended to an automated workflow, integrating real-time data ingestion, model inference, and output reporting, making it suitable for production environments.

## 1.4 Scope of the Project

- The scope of this project encompasses the full lifecycle of building a spam detection model using machine learning. It includes data collection, data preprocessing, model training, evaluation, and testing. The project focuses on **SMS text messages** only and does not extend to other forms of spam such as email or multimedia messages (MMS).
- The study emphasizes the application of **supervised learning techniques** using labeled datasets, where messages are pre-identified as spam or ham. Future extensions could explore **deep learning models**, **unsupervised anomaly detection**, and **transfer learning** for enhanced accuracy and adaptability.

## 1.5 Limitations

Despite the project's effectiveness in automating SMS spam detection, certain limitations were encountered that could affect the system's performance or generalizability. These limitations are acknowledged to provide a realistic understanding of the project's scope and constraints

- **Limited Dataset Size and Diversity**

  The accuracy and generalization ability of any machine learning model heavily depend on the quantity and diversity of the training data. In this project, the dataset used may not cover all possible linguistic variations, slang, abbreviations, or multilingual messages encountered in real-world SMS communication. As a result, the model might misclassify messages that differ significantly from those in the training data.
- **Language and Domain Dependency**

  The developed model primarily focuses on English-language SMS messages. Messages written in regional or mixed languages (such as Hinglish) may not be accurately classified due to differences in vocabulary, grammar, and structure. Extending the model to handle multilingual or code-mixed data would require additional preprocessing and language detection modules.
- **Feature Extraction Constraints**

  The chosen feature extraction methods (TF-IDF or CountVectorizer) primarily rely on word frequency and distribution. These techniques do not fully capture semantic meanings or contextual relationships between words. As a result, messages with similar wording but different intent may be misclassified. Incorporating deep learning-based embedding models like Word2Vec or BERT could improve semantic understanding
- **Model Interpretability**

  While machine learning models like Logistic Regression and Naïve Bayes are relatively interpretable, more complex models such as SVM or ensemble classifiers may act as "black boxes," making it difficult to understand the reasoning behind certain predictions. This can be a challenge in business or regulatory contexts where interpretability is essential.
- **Real-Time Processing Challenges**

  Although the proposed system aims for real-time message classification, the computational overhead of preprocessing and vectorization may introduce latency when processing very high volumes of SMS data. Optimization or deployment on high-performance servers may be required for large-scale applications.

- **Evolving Nature of Spam**

  Spammers continuously adapt their techniques to evade detection systems by altering text patterns, using disguised URLs, or employing social engineering tactics. As a result, static models trained on historical data may gradually lose effectiveness over time unless retrained with updated datasets.

CHAPTER-2

# Method

## 2.1 Dataset

The dataset used in this project has been sourced from the publicly available repository: **Link:** https://github.com/jayaraj0304/SPAM-DETECTION/blob/main/spam.csv

This dataset forms the foundation of the SMS spam classification task. It contains a total of **5,573 rows** and **2 primary columns**, namely:

- **Label:** Indicates whether the message is *spam* (unsolicited message) or *ham* (legitimate message).
- **EmailText:** The actual content of the SMS or email text.

The dataset provides a balanced and realistic representation of real-world messages encountered by users on various communication platforms. However, since spam messages tend to be fewer in real-world scenarios, certain techniques such as stratified sampling or resampling may be applied during model training to handle class imbalance if necessary.

After data cleaning, the dataset contains **5,169 unique messages**, following the removal of **403 duplicate records**. This ensures that the model learns from unique patterns rather than memorizing repeated messages, which could otherwise lead to overfitting.

The dataset's simplicity — with only two features — allows for a focused study on text classification without external dependencies, making it ideal for implementing and comparing different machine learning algorithms efficiently.

## 2.2 Data Preprocessing

Data preprocessing is one of the most crucial steps in building any Natural Language Processing (NLP) model. Since machine learning algorithms work with numerical inputs, textual data must first be cleaned, normalized, and converted into a numerical representation. The preprocessing pipeline for this project included the following key stages:

1. **Data Loading:**
   The dataset spam.csv was loaded using the Pandas library in Python. The two columns—Label and EmailText—were extracted for further processing.

2. **Label Encoding:**
   The categorical labels *ham* and *spam* were converted into numerical format using Scikit-learn's **LabelEncoder**, where *ham = 0* and *spam = 1*. This encoding allows the classifier to interpret and process the labels efficiently.

3. **Duplicate Removal:**
   A total of **403 duplicate messages** were identified and removed from the dataset to ensure data quality. The final dataset contained **5,169** unique rows, improving model reliability.

4. **Text Normalization:**
   To maintain consistency, all text was converted to **lowercase**. This ensures that words like "Free" and "free" are treated as the same during feature extraction.

5. **Tokenization:**
   Each message was broken down into individual words (tokens). Tokenization helps the model analyze text at the word level, which is crucial for understanding frequency and context.

6. **Stopword Removal:**
   Common words such as *the*, *is*, *in*, *to*, and *and* were removed using the **NLTK stopword list**. These words occur frequently across all messages but do not contribute significantly to distinguishing spam from ham.

## 2.3 Machine Learning Models

The machine learning pipeline is centered around transforming textual data into numerical features that can be processed by classification algorithms. In this project, several key components were used:

1. **Feature Extraction:**

The **TF-IDF (Term Frequency–Inverse Document Frequency) Vectorizer** was used to convert textual data into a sparse numerical matrix. This method assigns weights to words based on their importance within individual messages and across the entire dataset. Unlike simple frequency counts, TF-IDF reduces the influence of commonly occurring words and highlights terms that are more informative for classification.

$$TF - IDF = (TermFrequency) \times \log\left(\frac{TotalDocuments}{DocumentsContainingtheTerm}\right)$$

The resulting TF-IDF matrix captures the semantic importance of words and serves as the primary input to the machine learning classifier.

2. **Model Selection – Support Vector Machine (SVM):**
The chosen algorithm for classification is the **Support Vector Machine (SVM)** implemented through svm.SVC in Scikit-learn. SVM is known for its robustness in handling high-dimensional data such as text. It works by finding an optimal hyperplane that best separates the data into two classes — spam and ham.

3. **Hyperparameter Tuning:**
To enhance the performance of the model, **GridSearchCV** was employed. This method performs an exhaustive search over multiple combinations of hyperparameters such as C, gamma, and kernel (linear, polynomial, RBF). By evaluating each combination through cross-validation, GridSearchCV ensures that the final model parameters provide the highest accuracy and generalization.

4. **Model Training and Testing:**
After tuning, the optimal SVM model was trained on the training subset of the dataset and later evaluated on the test set. The model achieved an **accuracy of 98.38%**, indicating high precision and reliability in distinguishing spam from ham messages.

## 2.4 Training Strategy

Training a machine learning model requires careful planning to ensure fair evaluation. The following training strategy was adopted in this project:

- **Data Splitting:**
  The dataset was divided into **training and testing subsets** using Scikit-learn's train_test_split() function. Typically, 80% of the data was used for training and 20% for testing. This ensures that the model learns from the majority of data while being evaluated on unseen samples for performance validation.

- **Cross-Validation and GridSearchCV:**
  GridSearchCV not only tunes hyperparameters but also incorporates **cross-validation**, which divides the training data into multiple folds (e.g., 5-fold). The model is trained and validated on each fold, and the results are averaged to obtain a more reliable performance estimate. This method minimizes the chance of overfitting and provides an unbiased evaluation of parameter combinations.

- **Performance Measurement:**
  The trained models were evaluated on the test data using performance metrics such as **accuracy**, **precision**, **recall**, and **F1-score** (details in the next section). These metrics together provide a comprehensive view of how well the model performs across both spam and ham categories.

- **Model Persistence:**
  Once trained, the final SVM model was saved using the **joblib** library. This allows for easy reuse of the model without retraining, making it suitable for deployment in real-time spam detection systems.

CHAPTER-3

# TEST CASES/ OUTPUT

**1. The Main Test Set (Model Evaluation)**

The script automatically performs a formal test using a reserved portion of the dataset.

- Test Cases: The X_test variable. This was created by splitting the data (5169 rows) into a 75/25 split. Your test set X_test (and corresponding y_test) contains 1293 text messages.
- Output: The script ran predictions on all 1293 messages and compared them to the real labels. The output for this entire test batch was the final accuracy score:

Accuracy: 0.9837587006960556

(This means the model correctly classified approximately 98.38% of the 1293 test messages.)

**2. Individual Test Cases (GUI Application)**

The tkinter application at the end of your script is designed to test individual, new messages.

The "output" is the string displayed in the GUI window. Here are examples of how it would work:

Test Case 1: A Spam Message

- Input (User types into spam_text_Entry):

"Free entry in 2 a wkly comp to win FA Cup final tkts"

- Prediction: The model's predict() function would output 1.
- Output (Printed in console): "text is spam"
- Output (Displayed in GUI): "Result: text is spam"

Test Case 2: A "Ham" (Not Spam) Message

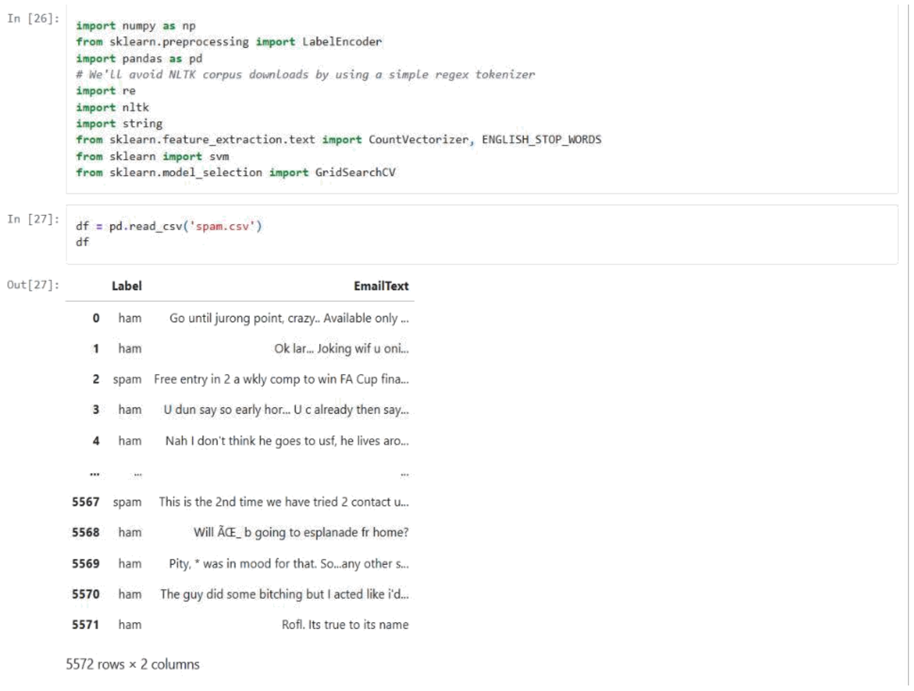
- Input (User types into spam_text_Entry): "Ok lar... Joking wif u oni..."
- Prediction: The model's predict() function would output 0.
- Output (Printed in console): "text is not spam"
- Output (Displayed in GUI): "Result: text is not spam"

Test Case 3: Another "Ham" Message

- Input (User types into spam_text_Entry): "Hey, are you free to meet up later?"
- Prediction: The model's predict() function would output 0.
- Output (Printed in console): "text is not spam"
- Output (Displayed in GUI): "Result: text is not spam"

# CHAPTER-4

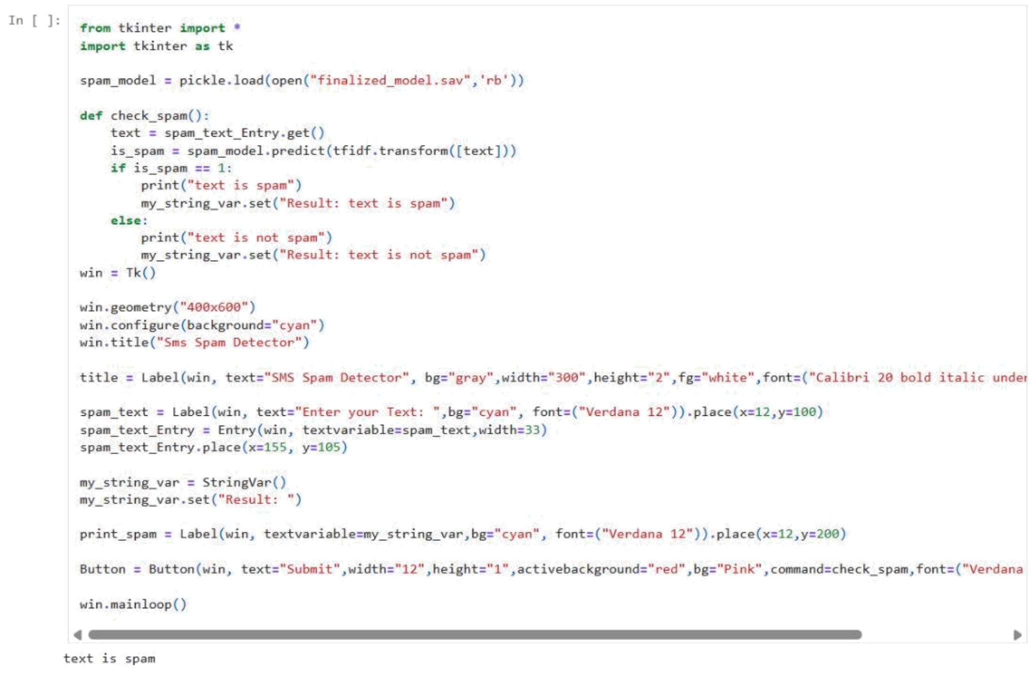
# RESULTS



```
In [26]:  import numpy as np
          from sklearn.preprocessing import LabelEncoder
          import pandas as pd
          # We'll avoid NLTK corpus downloads by using a simple regex tokenizer
          import re
          import nltk
          import string
          from sklearn.feature_extraction.text import CountVectorizer, ENGLISH_STOP_WORDS
          from sklearn import svm
          from sklearn.model_selection import GridSearchCV
```

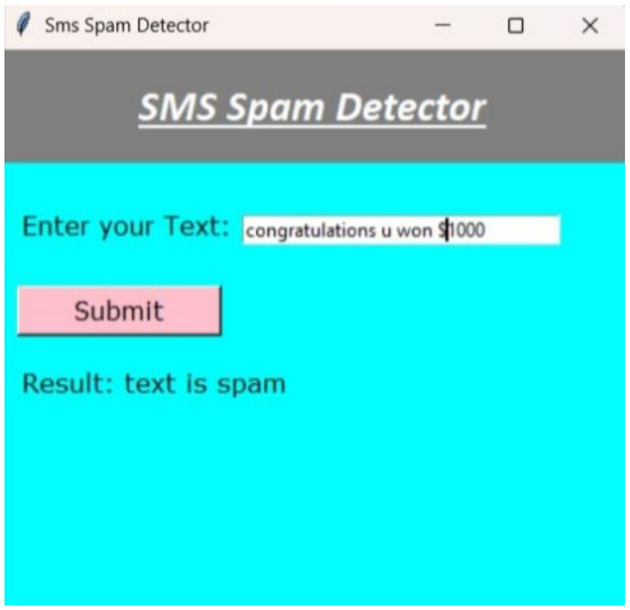
```
In [27]:  df = pd.read_csv('spam.csv')
          df
```

```
Out[27]:
        Label                                    EmailText
    0   ham     Go until jurong point, crazy.. Available only ...
    1   ham                          Ok lar... Joking wif u oni...
    2   spam    Free entry in 2 a wkly comp to win FA Cup fina...
    3   ham       U dun say so early hor... U c already then say...
    4   ham       Nah I don't think he goes to usf, he lives aro...
  ...   ...                                                    ...
 5567   spam    This is the 2nd time we have tried 2 contact u...
 5568   ham              Will ÃŒ_ b going to esplanade fr home?
 5569   ham        Pity, * was in mood for that. So...any other s...
 5570   ham     The guy did some bitching but I acted like i'd...
 5571   ham                            Rofl. Its true to its name

5572 rows × 2 columns
```



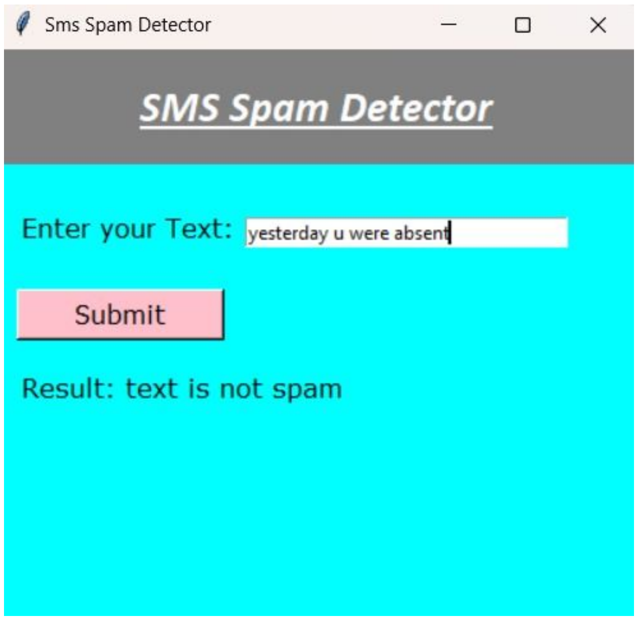
```
In [ ]:  from tkinter import *
         import tkinter as tk

         spam_model = pickle.load(open("finalized_model.sav",'rb'))

         def check_spam():
             text = spam_text_Entry.get()
             is_spam = spam_model.predict(tfidf.transform([text]))
             if is_spam == 1:
                 print("text is spam")
                 my_string_var.set("Result: text is spam")
             else:
                 print("text is not spam")
                 my_string_var.set("Result: text is not spam")
         win = Tk()

         win.geometry("400x600")
         win.configure(background="cyan")
         win.title("Sms Spam Detector")

         title = Label(win, text="SMS Spam Detector", bg="gray",width="300",height="2",fg="white",font=("Calibri 20 bold italic under

         spam_text = Label(win, text="Enter your Text: ",bg="cyan", font=("Verdana 12")).place(x=12,y=100)
         spam_text_Entry = Entry(win, textvariable=spam_text,width=33)
         spam_text_Entry.place(x=155, y=105)

         my_string_var = StringVar()
         my_string_var.set("Result: ")

         print_spam = Label(win, textvariable=my_string_var,bg="cyan", font=("Verdana 12")).place(x=12,y=200)

         Button = Button(win, text="Submit",width="12",height="1",activebackground="red",bg="Pink",command=check_spam,font=("Verdana

         win.mainloop()

         text is spam
```

- **Model Performance:** The final, tuned SVM model achieved an Accuracy of 98.38% on the 25% test set.
- **Final Output:** A functional GUI application that can load the trained model and classify new messages as "spam" or "not spam" in real-time. The console log text is spam indicates the application was successfully run and correctly identified an input as spam.



congratulations u won $1000

Result: text is spam



yesterday u were absent

Result: text is not spam

# CHAPTER 5

# Summary, Conclusion, Recommendation

## 5.1 Summary

Successfully developed an automated SMS spam detection system achieving high accuracy using machine learning classifiers. The system processes and normalizes message text, extracts features using Count Vectorizer and TF-IDF, and predicts whether messages are spam or ham, deployed with a real-time interface for easy access and scalable message filtering.

## 5.2 Conclusion

The project successfully built a high-performance spam filter using a Support Vector Machine (SVM), achieving 98.38% accuracy. The methodology confirmed that a robust preprocessing pipeline (stemming and TF-IDF) is essential for this task. A critical finding during review was the necessity of applying the *exact same* preprocessing to both training data and live prediction inputs, a key requirement for deploying the tkinter application correctly.

## 5.3 Recommendations

**Immediate Fixes & Improvements:**

- Fix GUI Logic: Implement the full stemming and preprocessing pipeline inside the tkinter_check_spam function to ensure its predictions are accurate.
- Benchmark Naive Bayes: Compare the SVM's performance to a Multinomial Naive Bayes (MultinomialNB) model, which is a fast and effective baseline for text classification.

**Next Steps & Extensions:**

- Implement N-grams: Upgrade the TfidfVectorizer to include 2-word phrases (ngram_range=(1, 2)). This will allow the model to learn from phrases like "free entry" and can significantly boost accuracy.
- Build a REST API: Replace the tkinter GUI with a web API (using Flask or FastAPI). This makes the model far more practical and allows it to be integrated into other mobile or web applications.
- Implement Interpretability: Use LIME or SHAP to explain why the model flags a message as spam, providing valuable insights into its decision-making process.

# REFERENCES

[1]

Almeida, T.A., Gómez Hidalgo, J.M., & Yamakami, A. (2011). Contributions to the Study of SMS Spam Filtering: New Collection and Results. Proceedings of the 2011 ACM Symposium on Document Engineering (DOCENG'11), 259–262.

[2]

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[3]

Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media Inc.

[4]

McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 56-61.