SMS SPAM DETECTION USING MACHINE LEARNING

A project report submitted in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

By

A. Charan Teja Reddy	**(21BF1A1202)**
K. Simhadri	**(21BF1A1232)**
M.S. Karthik	**(22BF5A1205)**
P. Gangi Reddy	**(22BF1A1208)**

Under the guidance of
Mr. B. Rajesh,
Assistant Professor.

DEPARTMENT OF INFORMATION TECHNOLOGY
SRI VENKATESWARA COLLEGE OF ENGINEERING
(AUTONOMOUS)
(Approved by AICTE, New Delhi & Permanently Affiliated to JNTUA, Ananthapuramu
Accredited by NBA, New Delhi & NAAC with 'A' grade)
Karakambadi Road, TIRUPATI – 517507.

2021 - 2025

SRI VENKATESWARA COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Approved by AICTE, New Delhi & Permanently Affiliated to JNTUA, Ananthapuramu,
Accredited by NBA, New Delhi & NAAC with 'A' Grade)
Karakambadi Road, TIRUPATI – 517507.

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIICATE

This is to certify that the project work entitled, **"SMS Spam Detection Using Machine Learning"** is a bona-fide record of the project work done and submitted by

A. Charan Teja Reddy	***(21BF1A1215)***
K. Simhadri	***(22BF5A1207)***
M.S. Karthik	***(21BF1A1244)***
P. Gangi Reddy	***(21BF1A1258)***

under my guidance and supervision for the partial fulfillment of the requirements for the award of B.Tech. degree in **INFORMATION TECHNOLOGY.** This project is the result of our own effort and that it has not been submitted to any other University or Institution for the award of any degree or diploma other than specified above.

PROJECT GUIDE	**HEAD OF THE DEPARTMENT**
B. Rajesh,	Dr. S. Murali Krishna, M. Tech, Ph. D
Assistant Professor,	Professor,
Dept of Information Technology,	Dept of Information Technology,
SV College of Engineering,	SV College of Engineering,
Tirupati - 517507	Tirupati – 517507

External Viva-Voce Exam held on ________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We hereby declare that the project report entitled "SMS spam detection using machine learning" done by us under the esteemed guidance of **Mr. B. Rajesh,** and is submitted in partial fulfilment of the requirements for the award of the Bachelor of Technology in **Information Technology**.

Date:	A. Charan Teja Reddy	(21BF1A1202)
Place:	K. Simhadri	(22BF5A1205)
	M.S. Karthik	(21BF1A1232)
	P. Gangi Reddy	(22BF5A1208)

ACKNOWLEDGEMENT

We are thankful to our guide **Mr. B. Rajesh, Assistant Professor** for his valuable guidance and encouragement. His helping attitude and suggestions have helped us in the successful completion of the project

We are thankful to project coordinators **Mr. R. Raja Kumar, Assistant Professor**, for his guidance and regular schedules**.**

We would like to express our gratefulness and sincere thanks to **Dr. S. Murali Krishna, Head of Department IT**, for his kind help and encouragement during the course of our study and in the successful completion of the project work.

We have great pleasure in expressing our hearty thanks to our beloved **Principal Dr. N. Sudhakar Reddy** for spending his valuable time with us to complete this project

Successful completion of any project cannot be done without proper support and encouragement We sincerely thank the Management for providing all the necessary facilities during the course of study.

We would like to thank our parents and friends, who have the greatest contributions in all our achievements, for the great care and blessings in making us successful in all our endeavors**.**

A. Charan Teja Reddy	**21BF1A1202**
K. Simhadri	**22BF5A1205**
M.S. Karthik	**21BF1A1232**
P. Gangi Reddy	**22BF5A1208**

ABSTRACT

This work presents an efficient machine learning-based approach for SMS spam detection, addressing the limitations of traditional rule-based filters. A comprehensive dataset comprising over 67,000 SMS messages, collected from diverse public sources, was used for training and evaluation. Each message was preprocessed through tokenization case normalization, stopword removal, and punctuation stripping. Feature extraction was performed using the Term Frequency-Inverse Document Frequency (TF-IDF) method to convert messages into a numerical format suitable for classification. A Multinomial Naïve Bayes (MNB) classifier was employed due to its effectiveness in handling text data and high-dimensional feature spaces. The model was further refined by tuning the probability threshold to optimize the trade-off between precision and recall. Experimental results demonstrated high accuracy, strong generalization, and robustness against imbalanced data, making the model suitable for real-time spam detection tasks. Compared to keyword-based methods this approach significantly reduces false positives and adapts better to evolving spam patterns.

CONTENTS

CHAPTER NO.	DESCRIPTION	PAGE NO

CHAPTER NO.	DESCRIPTION	PAGE NO

LIST OF FIGURES

LIST OF TABLES

LIST OF OBSERVATION

AD	-	Alzheimer Disease
SMC	-	Significant Memory Concern
NC	-	Normal Controls
MCI	-	Mild Cognitive Impairment
ROI	-	Region Of Interest
CNN	-	Convolutional Neural Network
CB	-	Coupled Boosting
CME	-	Coupled Metric Ensemble
MR	-	Magnetic Resonance
ADNI	-	Alzheimer disease Neuroimaging Initiative
EEG	-	Electroencephalographic
WC	-	Wavelet Coherence
PJD	-	Permutation Jaccard Distance
MMSE	-	Mini-Mental State Examination
MP-GAN	-	Multidirectional Perception Generative Adversarial Networks
GPL	-	General Public License
ML	-	Machine Learning
OPP	-	Object-Oriented Programming
LR	-	Labor Relations
MTBF	-	Mean Time Between Failures
UML	-	Unified Modelling Language

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In recent years, the widespread use of mobile phones and the growth of digital communication have significantly increased the volume of SMS (Short Message Service) traffic. While SMS remains a vital mode of communication for both personal and business use, it has also become a major target for spammers and cybercriminals. The rise in unsolicited, unwanted, and often malicious messages has made SMS spam a serious concern for users and service providers alike.

SMS spam ranges from simple promotional content to phishing attacks that aim to steal sensitive personal or financial information. As spammers constantly adapt their techniques to bypass filtering mechanisms, detecting spam in SMS has become increasingly challenging. These messages are crafted to appear legitimate and often attempt to mimic trusted sources, making manual detection difficult and error-prone.

To address this, researchers and developers have turned toward automated solutions using machine learning techniques. Unlike rule-based systems, machine learning models can learn patterns from large datasets and adapt to evolving spam tactics. This dynamic approach makes them well-suited for real-time SMS spam detection.

Spam detection in SMS can be treated as a binary classification problem, where each message is classified as either “Spam” or “Not Spam.” However, imbalanced data, subtle linguistic differences, and the short length of SMS messages pose unique challenges . Traditional datasets such as the UCI SMS spam collection have been widely used for training and benchmarking spam detection models. However, these datasets are limited in scale, containing only around 5,500 labeled messages, and may not reflect the diversity of real-world spam.

In this work, we address these limitations by using a larger, more diverse dataset comprising over 67,000 labeled SMS messages collected from various sources. We apply Term Frequency–Inverse Document Frequency (TF-IDF) for feature extraction and evaluate multiple algorithms to identify the most effective model for spam classification. Our experiments show that the Multinomial Naive Bayes algorithm performs best, achieving high accuracy and precision. We further improve prediction reliability by using a probability threshold of 0.7 to classify a message as spam. This approach enhances model confidence and minimizes false positives.

The proposed system provides a scalable, efficient, and accurate solution for SMS spam detection. By combining robust preprocessing techniques and lightweight machine learning models, it offers potential for real-time deployment in mobile and web-based messaging platforms.

CHAPTER 2

LITERATURE SURVEY

2.1 Multi-Type Feature Extraction and Early Fusion Framework for SMS Spam Detection

Hussein Alaa Al-Kabbi et al. proposed a comprehensive approach to SMS spam detection that includes preprocessing, multi-type feature extraction, fusion, selection, and classification. Their framework combines deep learning with a hybrid model that captures both local and global text semantics. The system was evaluated on a benchmark dataset using cross-validation and compared against traditional models like Random Forest and advanced ones like BERT. While their method showed high performance and robustness, the reliance on deep neural components may impact deployment in lightweight or mobile systems, where faster and more interpretable models are preferred.

2.2 Investigating Evasive Techniques in SMS Spam Filtering: A Comparative Analysis

Muhammad Salman et al. explored the evolving nature of SMS spam, focusing on evasive tactics used by spammers to bypass detection systems. They introduced one of the largest public SMS datasets to date, containing over 68,000 messages, and conducted a thorough evaluation of machine learning models—from shallow classifiers to deep networks. Their findings revealed that many standard classifiers struggled to maintain accuracy when faced with adversarial or manipulated input. The study emphasizes the importance of building models that are not only accurate but also resilient to evolving spam techniques, which aligns closely with the real-world objectives of spam detection systems.

2.3 Semantic Feature Fusion for Enhanced SMS Spam Detection

Recent approaches to SMS spam detection have explored fusing semantic features derived from word embeddings and syntactic patterns. By integrating these features through early or late fusion strategies, models can capture both context and structure of SMS messages more effectively. Studies using such techniques reported consistent improvements in performance metrics like F1-score and precision, especially on datasets with balanced spam-to-ham ratios. However, these improvements often come at the cost of increased preprocessing time, indicating the need for optimization in systems where real-time filtering is crucial.

2.4 Lightweight Ensemble Models for SMS Spam Detection on Mobile Devices

To address the limitations of deep learning in resource-constrained environments, researchers have explored ensemble methods using lightweight base models like Naive Bayes, Logistic Regression, and Decision Trees. These models, when combined using majority or weighted voting, have demonstrated competitive accuracy while maintaining low latency and computational overhead. Their practical applicability makes them suitable for mobile SMS filtering, especially in regions where internet access or device capability is limited. While not always outperforming deep models in controlled tests, their efficiency and interpretability provide strong justification for deployment in real-world applications.

CHAPTER 3

SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

Traditional SMS spam detection systems commonly use small datasets like the UCI SMS Spam Collection (5,500 messages) and rely on basic machine learning algorithms such as Naive Bayes, SVM, and Decision Trees. Some systems incorporate deep learning and ensemble methods to improve performance, but these models are often complex and resource-intensive. Most systems use standard probability thresholds (usually 0.5) for spam classification without much customization.

3.1.1. Disadvantages

- Small and Outdated Datasets: Frequently used datasets do not represent the wide variety of spam seen in real-world scenarios.
- High False Positives or Negatives: Fixed threshold models can misclassify borderline cases.
- Lack of Customization: Most systems use generic settings rather than dataset-specific tuning for better performance.
- Limited Evaluation: Many models report only accuracy, ignoring other crucial metrics like precision or recall in imbalanced datasets.
- Poor Scalability: Deep models may not be suitable for lightweight or real-time applications.

3.2 PROPOSED SYSTEM

The proposed system enhances spam detection using A larger and more diverse dataset of over 67,000+ labeled SMS messages, collected from multiple sources for better coverage. A TF-IDF based text vectorization technique to convert raw SMS into numerical features. The Multinomial Naive Bayes (MNB) algorithm, chosen after evaluating other classifiers for its balance of speed and accuracy. A custom probability threshold of 0.7 (instead of the default 0.5), improving the model's real-world spam detection performance by reducing false positives.

3.2.2 Advantages

- Better Generalization: The large, varied dataset helps the model perform well on modern and real-world spam messages. Improved accuracy and Precision:
- Achieved 96.49% accuracy and 95.05% precision, demonstrating strong performance.
- Efficient and Lightweight: MNB combined with TF-IDF provides fast predictions suitable for real-time applications.
- Threshold Tuning: Adjusting the decision threshold improves classification confidence and reduces misclassification.
- Balanced Evaluation: The model is assessed using multiple metrics to ensure fairness across classes.

CHAPTER 4

COMPUTATIONAL ENVIRONMENT

4.1 SOFTWARE REQUIREMENTS:

- Operating System : Windows 10 or later
- Platform : Python Technology
- Tool : Python 3.10
- Front End : HTML, CSS, JavaScript
- Back End : FastAPI
- Database : SQLite (for storing users and predictions history)
- Additional Libraries: scikit-learn, pandas, numpy, joblib, uvicorn

4.2 HARDWARE REQUIREMENTS:

- Processor : Intel i5 or higher
- RAM : 4 GB (minimum)
- Hard Disk : 128 GB (minimum)
- Keyboard : Standard or Multimedia
- Mouse : Compatible Optical Mouse

4.3 SOFTWARE FEATURES:

The SMS Spam Detection system is developed using a modular FastAPI backend, enabling a lightweight and efficient RESTful API structure. The architecture supports seamless integration between the machine learning model (trained using TF-IDF and Multinomial Naive Bayes) and a user-friendly frontend built with HTML, CSS, and JavaScript. The backend exposes prediction services through easily accessible endpoints, ensuring scalability and performance. The modularity of FastAPI facilitates rapid development, easy testing, and maintainability of components such as input processing, model inference, and result logging.Moreover, SQLite is used to store user credentials and prediction history, which allows lightweight, secure, and fast data operations without the need for external database servers.

4.3.1 Introduction To Python:

Python is a very popular general-purpose interpreted, interactive, object oriented, and high-level programming language. Python is a dynamically-typed and garbage collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). Today, Python is very high in demand and all the major companies are looking for great Python Programmers to develop websites, software components, and applications or to work with Data Science, AI, and ML technologies. In 2022, there is a high shortage of Python Programmers whereas the market demands more number of Python Programmers due to it's application in Machine Learning, Artificial Intelligence etc. One of Python's defining characteristics is its dynamic typing and automatic garbage collection features. Unlike statically-typed languages, Python does not require explicit declaration of variable types, allowing for greater flexibility and agility in coding. Furthermore, Python's garbage collector automatically manages memory allocation and deallocation, alleviating developers from the burden of manual memory management and reducing the likelihood of memory leaks and segmentation faults. Python's open-source nature and adherence to the GNU General Public License (GPL) foster a vibrant ecosystem of libraries, frameworks, and tools developed and maintained by a passionate community of contributors. This vast ecosystem encompasses a wide range of domains, from web development and software engineering to data science, artificial intelligence (AI), and machine learning (ML).The demand for skilled Python programmers has skyrocketed, with major companies across industries seeking talent proficient in Python to develop websites, software components, and applications, or to leverage its capabilities in data analysis, predictive modeling, and automation.

4.3.2 What Is A Script?

Up to this point, I have concentrated on the interactive programming capability of Python. This is a very useful capability that allows you to type in a program and to have it executed immediately in an interactive mode.

- Scripts are reusable

- Scripts are editable

4.3.2.1 Scripts are reusable:

Basically, a script is a text file containing the statements that comprise a Python program. Once you have created the script, you can execute it over and over without having to retype it each time. Its execution involves invoking the Python interpreter and providing the script file as input, allowing for seamless automation of tasks. Moreover, Python scripts boast portability across various platforms, ensuring compatibility and ease of sharing. They can be effortlessly edited using any text editor or specialized integrated development environments (IDEs), offering flexibility in the development process. Typically denoted by the ".py" extension, these scripts promote modularity by facilitating organization into modules, thereby enhancing code management and reuse. Whether executed interactively in a Python shell or non-interactively from the command line or within other programs, Python scripts empower users with the ability to automate tasks, streamline processes, and realize efficiencies in their workflow.

4.3.2.2 Scripts are editable:

Perhaps, more importantly, you can make different versions of the script by modifying the statements from one file to the next using a text editor. Then you can execute each of the individual versions. In this way, it is easy to create different programs with a minimum amount of typing. You will need a text editor. Just about any text editor will suffice for creating Python script files. You can use Microsoft Notepad, Microsoft WordPad, Microsoft Word, or just about any word processor if you want to. This approach not only streamlines the development process but also minimizes the need for extensive retyping or restructuring. This incremental approach fosters efficiency and productivity, enabling developers to focus their efforts on refining specific aspects of the program without being bogged down by redundant tasks. Furthermore, the accessibility of Python scripts extends to the choice of text editor. While specialized integrated development environments (IDEs) offer advanced features tailored to Python development, basic text editors suffice for creating and editing Python script files. Whether it's Microsoft Notepad, Microsoft WordPad, or any other word processor, the simplicity of Python's syntax makes it compatible with a wide range of text editors. This accessibility ensures that developers can leverage the tools they are most comfortable with, enhancing their workflow and productivity.

4.3.3 Difference Between Script And A Program:

This table succinctly highlights the key differences between scripts and programs, covering aspects such as their nature, user control, execution method, compilation process, and form.

Table 4.1 Key Difference Between Scripts And Program

ASPECT	SCRIPT	PROGRAM
Nature	Distinct from core code of application	Core code of application
User Control	Often created or modified by end-user	Typically developed by software engineer
Execution	Interpreted from source code or byte code	Executable form for direct execution
Compilation	Not traditionally compiled to native machine code	Traditionally compiled to native machine code
Form	Usually in human-readable source code	Executable form for computer execution

4.3.4 What Is Python?

Chances you are asking yourself this. You may have found this book because you want to learn to program but don’t know anything about programming languages. Or you may have heard of programming languages like C, C++, C#, or Java and want to know what Python is and how it compares to “big name” languages. Hopefully I can explain it to you.

4.3.4.1 Python Concepts:

If you’re not interested in the how and whys of Python, feel free to skip to the next chapter. In this chapter I will try to explain to the reader why I think Python is one of the best languages available and why it’s a great one to start programming with.

- Open source general-purpose language.
- Object Oriented, Procedural, Functional.
- Easy to interface with C/ObjC/Java/Fortran.
- Easy-is to interface with C++ (via SWIG).
- Great interactive environment.
- Python is a Beginner's Language

➢ Open source general-purpose language: Being open source means that Python's source code is freely available for anyone to view, modify, and distribute. This fosters a collaborative and transparent development environment, encouraging contributions from a global community of developers.Additionally, Python's general-purpose naturemeans it's not limited to any specific domain or application; rather, it can be used for a wide range of tasks, from scripting and automation to web development, data analysis, machine learning, and more.

➢ Object Oriented, Procedural, Functional: Python's support for multiple programming paradigms gives developers flexibility in how they structure their code and solve problems. Object-oriented programming (OOP) allows for the creation of reusable and modular code through the use of classes and objects. Procedural programming focuses on breaking down tasks into procedures or functions. Functional programming emphasizes the use of pure functions and immutable data structures to write concise and expressive code.

➢ Easy to interface with C/ObjC/Java/Fortran: Python's extensive standard library and third-party packages provide tools and modules for interfacing with libraries written in other languages. For example, the ctypes module allows Python code to call functions and use data structures defined in C libraries, while tools like PyObjC facilitate communication with Objective-C libraries on macOS/iOS platforms. Similar mechanisms exist for interfacing with Java (e.g., Jython) and Fortran (e.g., f2py).

➢ Easy to interface with C++ (via SWIG): While integrating Python with C++ can be more involved compared to other languages due to differences in their object models and memory management, tools like SWIG simplify the process by generating wrapper code that handles the translation between the two languages. This enables Python developers to leverage existing C++ libraries and take advantage of their performance and functionality within Python applications.

- ➢ Great interactive environment: Python's interactive interpreter,accessed via the command line or interactive development environments (IDEs) like Jupyter Notebook, provides an intuitive and immediate way to experiment with code. Developers can write and execute Python statements interactively, view results in real-time, and quickly iterate on their code to refine algorithms and debug issues. This interactive workflow promotes rapid prototyping and exploration, making Python well-suited for tasks that require experimentation and iterative development.
- ➢ Python is a Beginner's Language: Python's simplicity and readability make it an ideal choice for beginners who are just starting to learn programming. Its clean and expressive syntax reduces the cognitive overhead typically associated with learning a new language, allowing beginners to focus on understanding fundamental programming concepts such as variables, loops, conditionals, and functions.

4.3.5 History Of Python:

Python, incubated by Guido van Rossum during the late 1980s and early 1990s at the National Research Institute for Mathematics and Computer Science in the Netherlands, owes its lineage to a rich tapestry of programming languages. Van Rossum drew from an eclectic array of influences including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and UNIX shell scripting languages. Python's genesis lay in van Rossum's desire to craft a language that emphasized simplicity, clarity, and adaptability. Initially unfettered by a specific license, Python's source code was freely available, reflecting van Rossum's ethos of open collaboration and community-driven development.

However, as Python gained traction, it aligned itself with the Python Software Foundation License (PSFL), ensuring compatibility with the GNU General Public License (GPL). While van Rossum provided the initial impetus, Python's evolution into a widely-used programming language is a testament to the collective efforts of the Python Software Foundation and its dedicated core development team. Despite relinquishing his role as Python's Benevolent Dictator For Life (BDFL) in 2018, van

Rossum continues to exert a profound influence on the language's trajectory, offering insights and steering discussions within the community.

Python's journey from a pet project to a global phenomenon underscores the transformative power of open collaboration, innovation, and the enduring legacy of its visionary creator. While decisions about Python's development are now made through a more decentralized process involving the core development team and the broader community, van Rossum's contributions continue to shape the language's evolution. Overall, Python's journey from its humble beginnings as a personal project to becoming one of the world's most popular programming languages is a testament to the power of community-driven open source development and the vision of its creator, Guido van Rossum.

4.3.6 Features Of Python:

- ➢ Here are the features of the python that can be explained clearly in which everyone can understand easily:
- ➢ Easy-to-learn: Python's simplicity lies in its minimalistic syntax and clear structure. With a relatively small set of keywords and intuitive syntax, beginners can quickly grasp the fundamentals of the language. This low barrier to entry makes Python an ideal choice for novice programmers who want to start coding without getting bogged down by complex syntax.
- ➢ Easy-to-read: Python's emphasis on readability is evident in its code. The language encourages clean and concise code, making it easier for developers to understand and maintain. Python's use of indentation to define blocks of code also enhances readability by enforcing consistent formatting practices.
- ➢ Easy-to-maintain: Python's readability and clear syntax contribute to its ease of maintenance. Codebases written in Python are generally easier to maintain and refactor compared to languages with more convoluted syntax. This simplifies the process of fixing bugs, adding new features, and collaborating with other developers on large projects.
- ➢ Broad standard library: Python comes bundled with a comprehensive standard library that provides a wide range of modules and packages for various tasks. These modules cover everything from file I/O and networking to data processing and GUI development. This extensive library reduces the need for

external dependencies and accelerates development by providing ready-to-use solutions for common programming challenges.

➢ Interactive Mode: Python's interactive interpreter allows developers to test and debug code snippets interactively. This enables rapid prototyping and experimentation, as developers can quickly execute commands and observe results in real-time. The interactive mode is particularly useful for learning Python syntax and exploring the behavior of built-in functions and modules.

➢ Portable: Python's portability extends across different operating systems and hardware platforms. Python code written on one platform can typically run without modification on other platforms, including UNIX, Windows, and macOS. This cross-platform Alzheimer Disease Prediction Using Machine Learning Algorithms compatibility simplifies software deployment and ensures consistent behavior across diverse environments.

➢ Extendable: Python's extensibility enables developers to integrate low-level modules written in languages like C and C++ directly into Python applications. This capability allows Python programmers to optimize performance-critical sections of their code or access platform-specific functionality not available in pure Python.

➢ Databases: Python offers robust interfaces to major commercial databases, facilitating seamless integration with database systems such as MySQL, PostgreSQL, SQLite, and Oracle. This enables developers to build database-driven applications with ease and efficiency, leveraging Python's simplicity and flexibility.

➢ GUI Programming: Python supports graphical user interface (GUI) development through libraries like Tkinter , PyQt , and wxPython. These libraries provide tools and widgets for creating cross-platform GUI applications that can run on various operating systems, including Windows, macOS, and Linux. Python's support for GUI programming simplifies the development of desktop applications with rich user interfaces.

➢ Scalable: Python's scalability is evident in its ability to handle projects of varying sizes and complexities. From small scripts to large-scale applications, Python offers a flexible and scalable development environment. Its modular design, support for object oriented programming (OOP), and robust ecosystem of third-party libraries make it well-suited for building and maintaining projects of any scale.

4.3.7 Dynamic vs. Static:

Python is a dynamic-typed language. Many other languages are statically typed, such as C/C++ and Java. A static typed language requires the programmer to explicitly tell the computer what type of “thing” each data value is. For example, in C if you had a variable that was to contain the price of something, you would have to declare the variable as a “float” type. This tells the compiler that the only data that can be used for that variable must be a floating point number, i.e. a number with a decimal point. If any other data value was assigned to that variable, the compiler would give an error when trying to compile the program. Python, however, doesn’t require this. You simply give your variables names and assign values to them. The interpreter takes care of keeping track of what kinds of objects your program is using. This also means that you can change the size of the values as you develop the program. Say you have another decimal number (a.k.a. a floating point number) you need in your program. With a static typed language, you have to decide the memory size the variable can take when you first initialize that variable. A double is a floating point value that can handle a much larger number than a normal float (the actual memory sizes depend on the operating environment). If you declare a variable to be a float but later on assign a value that is too big to it, your program will fail; you will have to go back and change that variable to be a double. With Python, it doesn’t matter. You simply give it whatever number you want and Python will take care of manipulating it as needed. It even works for derived values. For example, say you are dividing two numbers. One is a floating point number and one is an integer. Python realizes that it’s more accurate to keep track of decimals so it automatically calculates the result as a floating point number Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

4.3.8 Standard Data Types:

Python standard data types are the data types in which the data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them. Python has five standard data types –

- Numbers
- Strings
- Lists
- Tuple
- Dictionary

➢ Numbers: Number data types store numeric values. Number objects are created when you assign a value to them.

➢ Strings: Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

➢ Lists: Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data types. The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

➢ Dictionary: Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are enclosed by curly braces ({})and values can be assigned and accessed using square ([]).

CHAPTER 5

MODULE DESCRIPTION

5.1. TF-IDF Vectorization

TF-IDF (Term Frequency-Inverse Document Frequency) is employed to convert raw SMS text data into numerical feature vectors. This technique captures the importance of words relative to a given message and the entire corpus, reducing the impact of commonly occurring terms. The TF-IDF vectors form the foundation for distinguishing between spam and ham messages in a meaningful way.

5.2 Multinomial Naive Bayes (MNB)

A Multinomial Naive Bayes classifier is used due to its simplicity and effectiveness for text classification. It assumes feature independence and models word frequencies, making it highly suitable for SMS spam detection tasks. The model learns to classify messages based on word occurrence patterns commonly found in spam versus legitimate (ham) messages.

5.3 Probability Threshold Tuning

To enhance the model's sensitivity to spam messages, a custom probability threshold is applied. Instead of the standard 0.5 cut-off, a lower threshold (e.g., 0.3–0.4) is selected to improve recall on spam detection. This helps the model minimize false negatives, ensuring spam messages are caught even if they appear only slightly suspicious.

5.4 Dataset Enhancement

The model is trained on a large-scale dataset of over 67,000 SMS messages, including 61% ham and 39% spam. This dataset provides a broader linguistic representation and real-world context, improving the model’s robustness compared to smaller datasets used in many prior works. Diverse message structures, spam evasion patterns, and informal language are well-represented.

5.5 Integration with FastAPI

The trained model and TF-IDF vectorizer are serialized using joblib and integrated into a FastAPI-based backend. This allows the system to serve real-time predictions via API endpoints. The FastAPI framework ensures high performance, scalability, and easy communication with the frontend interface, supporting both authenticated users and guests.

CHAPTER 6

SYSTEM STUDY

6.1 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ➢ ECONOMICAL FEASIBILITY
- ➢ TECHNICAL FEASIBILITY
- ➢ SOCIAL FEASIBILITY

6.1.1 Economical Feasibility

This analysis evaluates the cost-effectiveness of the system. The system is developed using open-source and freely available technologies such as Python, FastAPI, and SQLite. No additional licenses or paid libraries were required. The only potential costs may be incurred for hosting services or future upgrades if deployed on a large scale. Since the project is implemented in a minimal development environment, it remains well within budget, making it economically feasible for individual developers, academic institutions, and startups.

6.1.2 Technical Feasibility

This study examines the technical capabilities required for system development and implementation. The system is built using Python 3.12.4 with machine learning libraries like Scikit-learn, and the backend is developed using FastAPI. It operates smoothly on standard hardware configurations and requires no specialized equipment. The technical environment is easy to set up, and the backend is lightweight, making it ideal for both offline and cloud-based deployment. The system ensures low resource consumption and is adaptable to a wide range of platforms, including Windows, Linux, and cloud servers.

6.1.3 Social Feasibility

Social feasibility measures how well the system is accepted by users. The SMS Spam Detection system features a simple, user-friendly interface, allowing both registered users and guests to access spam prediction tools easily. It provides clear visual, text feedback to accommodate all users, including those with disabilities. The intuitive design and minimal training requirement ensure a high level of user acceptance. Moreover, the ability to provide immediate feedback builds trust in the system and encourages positive user engagement.

CHAPTER 7

REQUIREMENTS

7.1 FUNCTIONAL REQUIREMENTS

Each requirement has been numbered in a traceable manner using the prefix FR_SMS_### (FR for Functional Requirement, SMS for the SMS spam detection system).

7.1.1 System Interfaces

7.1.1.1 Frontend-to-Backend API Interface

The frontend will communicate with the FastAPI backend using HTTP requests to send input text and receive classification results (spam or ham). This interface is responsible for prediction, storing user history, and login functionality.

- **Endpoint**: /predict
- **Method**: POST
- **Input**: JSON payload containing the text message to be classified.
- **Output**: JSON response with prediction label ("Spam" or "Ham") and probability score.

7.1.1.2 File Structure and Format

The trained machine learning model and TF-IDF vectorizer are stored as .pkl files using the joblib library.

7.1.2 Functional Modules

7.1.2.1 User Authentication

Users must be able to register and log in to access personalized history tracking. Guest access is allowed for trying out the prediction functionality without saving history.

7.1.2.2 SMS Prediction

System must classify user-inputted SMS as either "Spam" or "Ham". System should return a confidence score based on probability threshold tuning.

7.1.2.3 Prediction History

Authenticated users can view the last five predictions stored in an SQLite database. Each record includes SMS text, prediction label, probability, and timestamp.

7.1.2.3 Responsive Web Interface

The frontend should be mobile-friendly and responsive. The prediction area, result section, and buttons must be accessible and clearly visible.

7.2 NON- FUNCTIONAL REQUIREMENTS

Non-functional requirements define the quality attributes and operational constraints of a system rather than specific behaviors or functions. They describe how the system performs rather than what it does. These requirements address aspects such as usability, reliability, performance, portability, maintainability, and security, which are crucial for ensuring the software is efficient, user-friendly, scalable, and robust. Unlike functional requirements, which specify individual system features, non- functional requirements help measure the system's overall effectiveness and user satisfaction in real-world usage. They play a vital role in determining the success and acceptance of a software product.

7.2.1 Usability:

The system is designed with a simple, intuitive user interface that ensures ease of use for both technical and non-technical users. Clear labels, meaningful feedback messages, and minimal input requirements contribute to a smooth user experience. Additionally, features like visual indicators support users with disabilities or literacy limitations, enhancing the accessibility of the application.

7.2.2 Reliability:

The SMS Spam Detection system ensures consistent and dependable performance by using well-tested machine learning models and robust backend services. The prediction functionality is stable, and fault tolerance mechanisms are incorporated to handle unexpected input or system errors gracefully. Logging and basic exception handling are in place to detect and debug issues when necessary.

7.2.3 Performance:

Developed using Python, FastAPI, and a browser-based frontend, the application is platform-independent and can run seamlessly across Windows, Linux, and macOS environments. The use of SQLite as a lightweight database also contributes to its portability by requiring minimal setup for local or remote deployments.

7.2.4 Supportability:

Built with modular architecture and standard frameworks like FastAPI, the system is easy to maintain, update, and extend. Each component—such as model loading, prediction logic, and database access—is separated for better clarity and manageability. The use of common technologies and thorough code documentation also simplifies the onboarding process for future developers or researchers interested in improving or scaling the application.

CHAPTER 8

UML DIAGRAMS

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering. If the broader topic of product development "blends the perspective of marketing, design, and manufacturing into a single approach to product development, then design is the act of taking the marketing information and creating the design of the product to be manufactured.

Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user.

8.1 UML DIAGRAMS

The Unified Modelling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. Simply, Systems design refers to the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements which can be done easily through UML diagrams. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information. The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML comprises two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

8.2 CONTENTS OF UML

In general, a UML diagram consists of the following features:

- **Entities:** These may be classes, objects, users or systems behaviours.
- Relationship Lines that model the relationships between entities in the system.
- **Generalization --** a solid line with an arrow that points to a higher abstraction of the present item.
- **Association --** a solid line that represents that one entity uses another entity as part of its behaviour.
- **Dependency --** a dotted line with an arrowhead that shows one entity depends on the behaviour of another entity.

8.3 In this project four basic UML diagrams have been explained

- **Use Case Diagram**
- **Class Diagram**
- **Sequence Diagram**
- **Collaboration Diagram**

8.3.1 USE CASE DIAGRAM

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. It consists of a group of elements (for example, classes and interfaces) that can be used together in a way that will have an effect larger than the sum of the separate elements combined. The use case should contain all system activities that have significance to the users. A use case can be thought of as a collection of possible scenarios related to a particular goal, indeed, the use case and goal are sometimes considered to be synonymous.

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

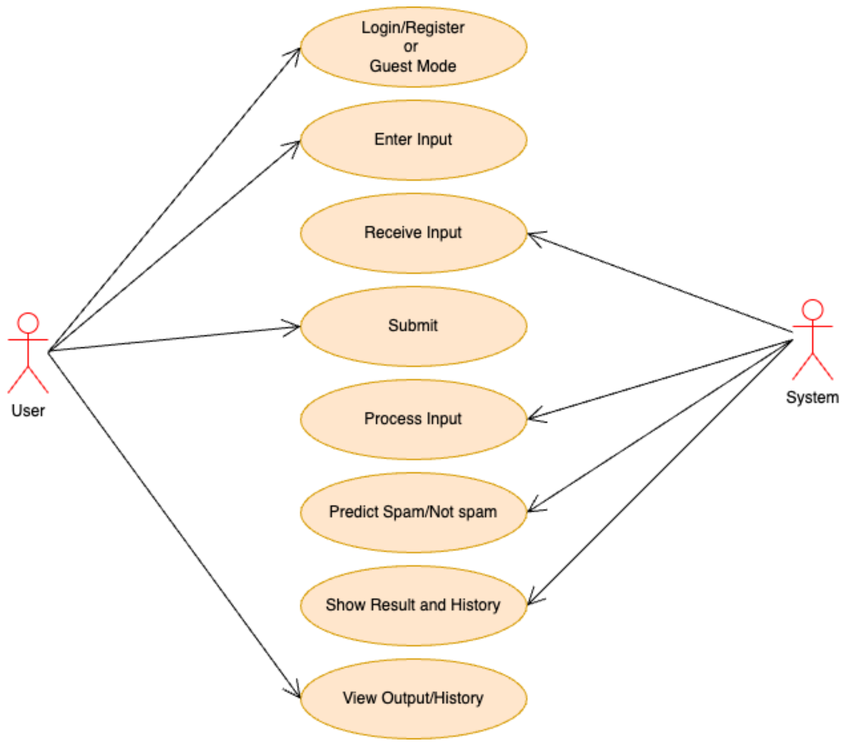


Fig. 8.1 ***Use Case Diagram***

8.3.2 CLASS DIAGRAM

UML class diagrams model static class relationships that represent the fundamental architecture of the system. Note that these diagrams describe the relationships between classes, not those between specific objects instantiated from those classes. Thus the diagram applies to all the objects in the system. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

A class diagram consists of the following features:

- ➢ **Classes:** These titled boxes represent the classes in the system and contain information about the name of the class, fields, methods and access specifiers. Abstract roles of the Class in the system can also be indicated.
- ➢ **Interfaces:** These titled boxes represent interfaces in the system and contain information about the name of the interface and its methods. Relationship Lines that model the relationships between classes and interfaces in the system.
- ➢ **Dependency:** A dotted line with an open arrowhead that shows one entity depends on the behavior of another entity. Typical usages are to represent that one class instantiates another or that it uses the other as an input parameter
- ➢ **Aggregation:** Represented by an association line with a hollow diamond at the tail end. An aggregation models the notion that one object uses another object without "owning" it and thus is not responsible for its creation or destruction.
- ➢ **Inheritance:** A solid line with a solid arrowhead that points from a subclass to a super class or from a sub-interface to its superinterface.
- ➢ **Implementation:** A dotted line with a solid arrowhead that points from a class to the interface that it implements.
- ➢ **Composition:** Represented by an association line with a solid diamond at the tail end. A composition models the notion of one object "owning" another and thus being responsible for the creation and destruction of another object

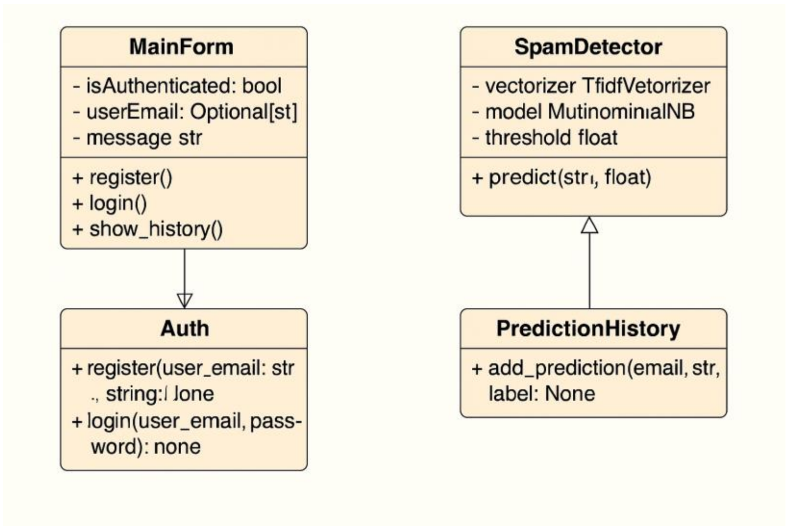


Fig. 8.2 ***Class Diagram***

8.3.3 SEQUENCE DIAGRAM

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A Sequence diagram depicts the sequence of actions that occur in a system. The invocation of methods in each object, and the order in which the invocation occurs is captured in a Sequence diagram. This makes the Sequence diagram a very useful tool to easily represent the dynamic behaviour of a system.

It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams. It shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams address the dynamic view of the system.

8.3.3.1 Elements of sequence diagram

The sequence diagram is an element that is used primarily to showcase the interaction that occurs between multiple objects. This interaction will be shown over a certain period of time. Because of this, the first symbol that is used is one that symbolizes the object.

- ➢ **Lifeline**

A lifeline will generally be generated, and it is a dashed line that sits vertically, and the top will be in the form of a rectangle. This rectangle is used to indicate both the instance and the class. If the lifeline must be used to denote an object, it will be underlined.

➢ **Messages**

To showcase an interaction, messages will be used. These messages will come in the form of horizontal arrows, and the messages should be written on top of the arrows. If the arrow has a full head, and it's solid, it will be called a synchronous call. If the solid arrow has a stick head, it will be an asynchronous call. Stick heads with dash arrows are used to represent return messages.

➢ **Objects**

Objects will also be given the ability to call methods upon themselves, and they can add net activation boxes. Because of this, they can communicate with others to show multiple levels of processing. Whenever an object is eradicated or erased from memory, the "X" will be drawn at the lifeline's top, and the dash line will not be drawn beneath it. This will often occur as a result of a message. If a message is sent from the outside of the diagram, it can be used to define a message that comes from a circle that is filled in. Within a UML based model, a Super step is a collection of steps which result from outside stimuli.

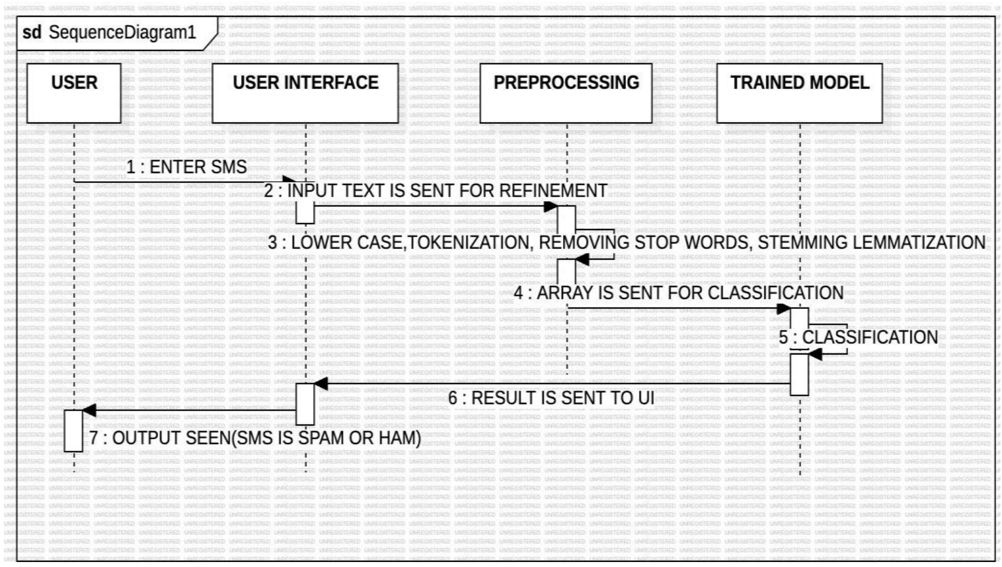

sd SequenceDiagram1
USER
USER INTERFACE
PREPROCESSING
TRAINED MODEL
1 : ENTER SMS
2 : INPUT TEXT IS SENT FOR REFINEMENT
3 : LOWER CASE,TOKENIZATION, REMOVING STOP WORDS, STEMMING LEMMATIZATION
4 : ARRAY IS SENT FOR CLASSIFICATION
5 : CLASSIFICATION
6 : RESULT IS SENT TO UI
7 : OUTPUT SEEN(SMS IS SPAM OR HAM)

Fig.8.3 ***Sequence Diagram***

8.4 COLLABORATION DIAGRAM

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). These diagrams can be used to portray the dynamic behavior of a particular use case and define the role of each object.

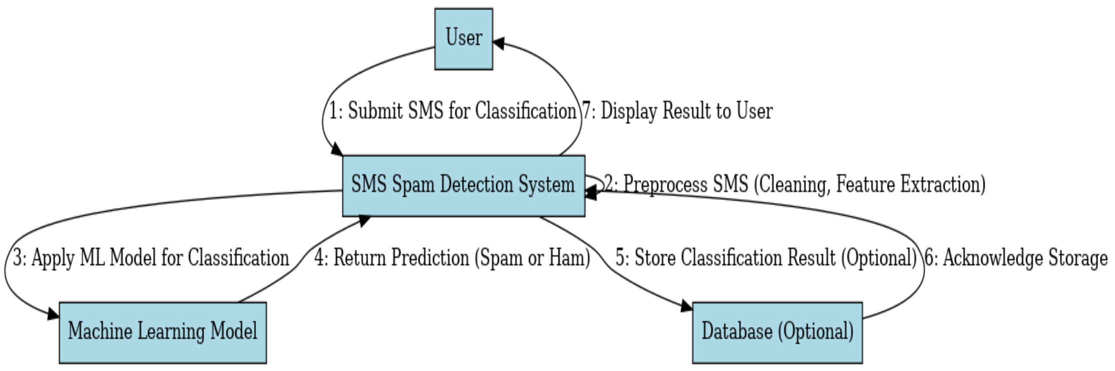

User
1: Submit SMS for Classification
7: Display Result to User
SMS Spam Detection System
2: Preprocess SMS (Cleaning, Feature Extraction)
3: Apply ML Model for Classification
4: Return Prediction (Spam or Ham)
5: Store Classification Result (Optional)
6: Acknowledge Storage
Machine Learning Model
Database (Optional)

Fig.8.4 ***Collaboration Diagram***

8.5 ACTIVITY DIAGRAM

An Activity Diagram in Unified Modeling Language (UML) is a type of behavioral diagram that represents the flow of activities or actions within a system or process. It is widely used to model the dynamic aspects of a system by capturing the sequence and conditions for coordinating lower-level behaviors. Activity diagrams are particularly helpful in illustrating business workflows, use case behavior, or the detailed logic of a single process.

An Activity Diagram depicts the flow from one activity to another and shows the overall control flow as well as data flow between activities. It consists of various elements such as initial nodes, actions, decisions, forks, joins, and final nodes, which help represent both sequential and concurrent flows in a system. These diagrams can be used for both system-level and process-level modeling.

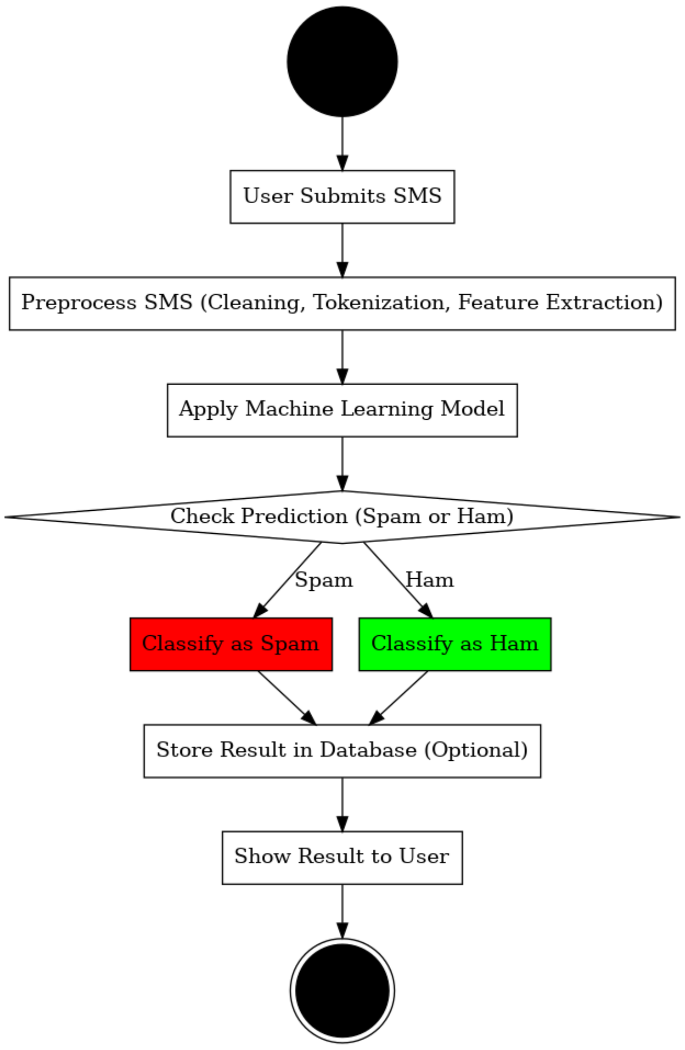

User Submits SMS
Preprocess SMS (Cleaning, Tokenization, Feature Extraction)
Apply Machine Learning Model
Check Prediction (Spam or Ham)
Spam
Ham
Classify as Spam
Classify as Ham
Store Result in Database (Optional)
Show Result to User

Fig.8.5 ***Activity diagram of SMS Spam Detector***

CHAPTER 9

SOFTWARE FEATURES

9.1 PYTHON ANACONDA SCRIPT

Anaconda is an open-source distribution of the Python and R programming languages for data science that aims to simplify package management and deployment. Package versions in Anaconda are managed by the package management system, Anaconda, which analyses the current environment before executing an installation to avoid disrupting other frameworks and packages. The Anaconda distribution comes with over 250 packages automatically installed. Over 7500 additional open-source packages can be installed from PyPl as well as the Anaconda package and virtual environment manager. It also includes a GUI (Graphical User Interface), Anaconda Navigator, as a graphical alternative to the command line interface. Anaconda Navigator is included in the Anaconda distribution, and allows users to launch applications and manage Anaconda packages, environments and channels without using command-line commands. Navigator can search for packages, install them in an environment, rum the packages and update them.

The big difference between Anaconda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science. When pip installs a package, it automatically installs any dependency Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example TensorFlow, can find that it stops working after using pip to install a different package that requires a different version of the dependent NumPy library than the one used by TensorFlow. In some cases, the package may appear to work but produce different results in execution. In contrast, Anaconda analyze the current environment including everything currently installed, and together with any version limitations specified (e.g., the user may wish to have TensorFlow version 2.0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open-source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or the user's own private repository or mirror, using the Anaconda install command. Anaconda Inc. compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64-bit, Linux 64-bit and MacOS 64-bit. Anything available on PyPl may be installed into an Anaconda environment using pip, and Anaconda will keep track of what it has installed itself and what pip has installed.

Difference between Anaconda and Data Science Platforms is While Anaconda supports some functionality you find in a data science platform, like Domino, it provides a sunset of that functionality. Domino and other platforms not only support package management, but they also support capabilities like collaboration, reproducibility, scalable compute, and model monitoring. Conda can be used within the Domino environment.

Anaconda is an amazing collection of scientific Python packages, tools, resources, and IDEs. This package includes many important tools that a Data Scientist can use to harness the incredible force of Python. Anaconda individual edition is free and open source. This makes working with Anaconda accessible and easy. Just go to the website and download the distribution.

With over 20 million users, covering 235 regions, and with over 2.4 billion package downloads; Anaconda has grown an exceptionally large community. Anaconda makes it easy to connect to several different scientific, Machine Learning, and Data Science packages.

The Anaconda distribution comes with packages that can be used on Windows, Linux, and MacOS. The individual edition includes popular package names like NumPy, Pandas, SciPy, Scikit-learn, TensorFlow, PyTorch, matplotlib, and more. The Anaconda Prompt and PowerShell makes working within the filesystem easy and manageable. Also, the GUI interface on Anaconda Navigator makes working with everything exceptionally smooth. Anaconda is an excellent choice if you are looking for a thriving community of Data Scientists and ever-growing support in the industry.

Conducting Data science projects is an increasingly simpler task with the help of great tools like this.

Open-Source software that allows Data Scientists to conduct workflows and effectively realize scientific and computational solutions. With an emphasis on presentation and readability, Jupyter Notebooks are a smart choice for collaborative projects as well and insightful publications. Jupyter Notebooks are open-source and developed on GitHub publicly by the Jupyter community.

A top-notch Python IDE that is packed full of features and pre-installed packages. With comfortable environment management and an easy to setup workstation, PyCharm is in a league of its own, when it comes to Python. With community, professional, and enterprise editions, there is a version for everyone.

9.2 CSV

A Comma-Separated Values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format. A CSV file typically stores tabular data(Numbers and text) in plain text, in which case each line will have the same number of fields.

The CSV file format is not fully standardized. Separating fields with commas is the foundation, but commas in the data or embedded line breaks have to be handles specially. Some implementations disallow such content while others surround the field with quotation marks, which yet again creates the need foe escaping if quotation marks are present in the data.

The term “CSV” also denotes several closely-related delimiter-separated formats that use other field delimiters such as semicolons. These include tab-separated values and space-separated values. A delimiter guaranteed not to be part of the data greatly simplifies parsing.

9.3 FAST API

FastAPI is a modern, high-performance web framework for building APIs with Python 3.7+ based on standard Python type hints. It is designed for creating fast, efficient, and scalable web applications and APIs with minimal code and maximum performance.

FastAPI is known for its speed and automatic generation of interactive API documentation using Swagger UI and ReDoc. It uses asynchronous programming with async and await, making it ideal for handling a large number of concurrent requests. FastAPI is also built on top of Starlette for the web parts and Pydantic for data validation, making it both powerful and easy to use.

One of the key advantages of FastAPI is its focus on developer productivity and performance. It supports automatic data validation, serialization, and detailed error messages, significantly reducing the amount of code needed for common tasks like input validation and response formatting. This makes FastAPI a popular choice for developing modern APIs, especially in machine learning, data science, and microservices applications.

9.4 VISUAL STUDIO

Visual Studio is an integrated development environment (IDE) developed by Microsoft. It is used to create a wide range of applications, including desktop applications, web applications, mobile applications, games, and more.

Visual Studio includes a code editor, a debugger, a compiler, and other tools to help developers create, debug, and deploy their software. Visual Studio supports several programming languages, including C++, C#, Visual Basic, and JavaScript. It also includes features such as code completion, syntax highlighting, and code refactoring to make development faster and easier.

Visual Studio is available in several editions, including Community, Professional, and Enterprise. The Community edition is free for individuals and small teams and includes many of the features of the Professional edition. The Professional and Enterprise editions are designed for larger teams and include additional features such as advanced debugging tools, code profiling, and more.

In addition to the IDE itself, Visual Studio also includes a number of other tools and services, such as Azure DevOps for collaboration and source control, and Azure App Service for deploying web applications to the cloud.

9.5 PACKAGES

9.5.1 Pandas

Pandas - Pandas is an open-source Python package that is most widely used for Python is data science/data analysis and machine learning tasks. It is built on top of another package named NumPy, which provides support for multi-dimensional arrays. Pandas is a Python package that provides high-performance data manipulation and analysis tools. It is built on top of the NumPy package and is designed to work with structured and tabular data. Pandas provides two main data structures: Series and DataFrame. A Series is a one-dimensional array-like object that can hold any data type, while a DataFrame is a two-dimensional table-like structure that contains rows and columns of data. Both series and DataFrame objects can be manipulated and transformed using Pandas functions.

Some of the key features of Pandas include:

- ➢ Pandas is a Python package that provides high-performance data manipulation and analysis tools. It is built on top of the NumPy package and is designed to work with structured and tabular data.
- ➢ **Pandas provides two main data structures:** Series and DataFrame. A Series is a one dimensional array-like object that can hold any data type, while a DataFrame is a two dimensional table-like structure that contains rows and columns of data. Both Series and DataFrame objects can be manipulated and transformed using Pandas functions.
- ➢ **Data manipulation:** Pandas provides a wide range of functions for filtering, selecting, and manipulating data, such as merging and grouping data, pivoting tables, and transforming data using mathematical or statistical functions. Data analysis: Pandas provides a number of functions for data analysis, including data visualization, statistical analysis, and time-series analysis.
- ➢ **Data cleaning:** Pandas provides functions for cleaning and transforming data, such as filling missing values, removing duplicates, and handling data outliers. Data input/output: Pandas provides functions for reading and writing data to

and from a wide range of file formats, including CSV, Excel, SQL databases, and more.

Pandas is widely used in data science and machine learning projects, as it provides a powerful and flexible way to work with structured data. It is also a popular tool for data analysis and data visualization, as it integrates well with other Python packages such as Matplotlib and Seaborn.

9.5.2 Pickle

Pickle is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network. Pickle is a Python module that provides a way to serialize and deserialize Python objects. Serialization is the process of converting an object into a byte stream, while deserialization is the process of converting a byte stream back into an object. The Pickle module can be used to save and load Python objects to and from a file. This can be useful for storing complex data structures or for passing data between different Python processes.

Pickle can serialize many different types of Python objects, including custom classes, functions, and even instances of built-in types. When an object is pickled, all of its attributes and methods are also pickled, so the entire object can be reconstructed when it is unpickled.

One important thing to note about Pickle is that it can be a security risk if used improperly. Pickled data can contain malicious code, so it is important to only unpickle data from trusted sources.

To use Pickle, you first need to import the module. You can then use the dump() method to write a pickled object to a file, or the load() method to read a pickled object from a file. There are also other methods provided by the Pickle module for more advanced use cases, such as using custom pickling protocols or working with large objects.

9.5.3 NumPy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. NumPy (Numerical Python) is a Python library for scientific computing that provides support for large, multidimensional arrays and matrices, as well as a large collection of mathematical functions to operate on these arrays. It is one of the most widely used libraries in data science and machine learning.

NumPy provides an array object that is similar to a list or a tuple, but with many additional features. The array object is optimized for numerical operations and is much more memory efficient than a list or a tuple.

NumPy also provides a number of mathematical functions that can operate on arrays, such as trigonometric functions, exponential functions, logarithmic functions, and more. These functions are optimized for speed and can perform operations on entire arrays at once, making them much faster than equivalent operations using Python's built-in math functions.

9.5.4 Sklearn

The Sklearn library contains a lot of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction. Scikit-learn (also known as sklearn) is a popular open-source machine learning library for Python. It provides a wide range of algorithms and tools for tasks such as data preprocessing, classification, regression, clustering, and more.

Some of the key features of scikit-learn include:

- ➢ **Easy to use:** scikit-learn provides a simple and consistent interface for working with different machine learning algorithms.
- ➢ **Widely used algorithms:** scikit-learn provides a wide range of machine learning algorithms, including logistic regression, decision trees, random forests, support vector machines (SVM), and more.
- ➢ **Data preprocessing:** scikit-learn provides a range of tools for data preprocessing, such as scaling, normalization, and imputation of missing values.

- **Model selection and evaluation:** scikit-learn provides tools for evaluating machine learning models using metrics such as accuracy, precision, recall, and F1-score. It also provides tools for model selection, such as cross-validation.
- **Integration with other libraries:** scikit-learn integrates well with other popular Python libraries, such as NumPy, Pandas, and Matplotlib.

Overall, scikit-learn is a powerful and flexible library that makes it easy to build and train machine learning models in Python. It is widely used in both academia and industry and has a large and active community of developers and users.

9.5.5 Matplotlib

Matplotlib is a low-level graph plotting library in python that serves as a visualization utility. Matplotlib is a popular open-source data visualization library for Python. It provides a range of tools for creating a variety of charts and plots, including line plots, scatter plots, bar charts, histograms, and more.

Some of the key features of Matplotlib include:

- **Easy to use:** Matplotlib provides a simple and consistent interface for creating plots and charts, which makes it easy to get started with data visualization.
- **Customizability:** Matplotlib allows users to customize almost every aspect of their plots, including colors, labels, markers, and more.
- **Publication quality:** Matplotlib is designed to produce publication-quality plots that are suitable for use in research papers, presentations, and other professional settings.
- **Integration:** Matplotlib integrates well with other Python libraries, such as NumPy, Pandas, and SciPy.
- **Interactivity:** Matplotlib provides tools for creating interactive plots and charts, which allows users to explore their data in more detail.

Overall, Matplotlib is a powerful and flexible library for data visualization in Python. It is widely used in both academia and industry and has a large and active community of developers and users.

9.5.6 FastAPI

FastAPI is a modern, fast (high-performance) web framework for building APIs with Python 3.6+ based on standard Python type hints. It is designed to create APIs quickly, with automatic generation of interactive documentation using Swagger UI and

ReDoc. FastAPI is built on top of Starlette for web handling and Pydantic for data validation, making it powerful, efficient, and easy to use. The framework supports asynchronous programming, which enables handling many requests simultaneously with minimal resource usage. FastAPI is known for its intuitive developer experience, built-in security handling, and support for OAuth2 authentication protocols, which makes it ideal for building production-ready APIs. It is especially popular in projects involving machine learning and real-time data applications.

9.5.7 NLTK

The Natural Language Toolkit (NLTK) is a powerful Python library used for working with human language data (text). It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with libraries for text processing, classification, tokenization, stemming, tagging, parsing, and semantic reasoning. In the context of this project, NLTK is used to preprocess SMS messages by removing stopwords, tokenizing sentences into words, and stemming words using the PorterStemmer algorithm. This prepares the text data for vectorization and classification by the machine learning model. NLTK is an essential tool in natural language processing (NLP) tasks and is commonly used in academic and commercial text analysis applications.

9.5.8 PASSLIB

Passlib is a password hashing library for Python that simplifies the process of securing user passwords in web applications. It supports a wide range of hashing algorithms, including bcrypt, SHA-256, and PBKDF2. In this project, Passlib is used to hash user passwords before storing them in the in-memory database and verify passwords during login. This ensures that sensitive information such as passwords are never stored or transmitted in plain text, providing essential security for user authentication systems. Passlib is easy to integrate with FastAPI and is widely used in security-conscious Python applications.

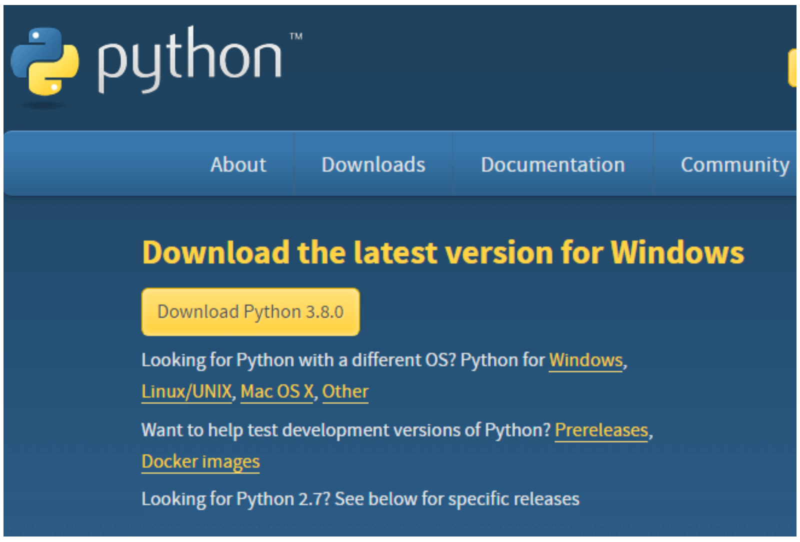
CHAPTER 10

SOFTWARE INSTALLATION

10.1 SOFTWARE INSTALLATION FOR MACHINE LEARNING PROJECT:

10.1.1 Installing Python:

1. To download and install Python visit the official website of Python https://www.python.org/downloads/ and choose your version.



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Prereleases, Docker images

Looking for Python 2.7? See below for specific releases

Fig.10.1 ***Installation of Python***

2. Once the download is complete, run the .exe to install Python. Now click on Install Now.
3. You can see Python installed at this point.
4. When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

10.1.2 Installing PyCharm:

1. Go to the official JetBrains website https://www.jetbrains.com/pycharm/download. Click **"Download"** under the **Community** edition .

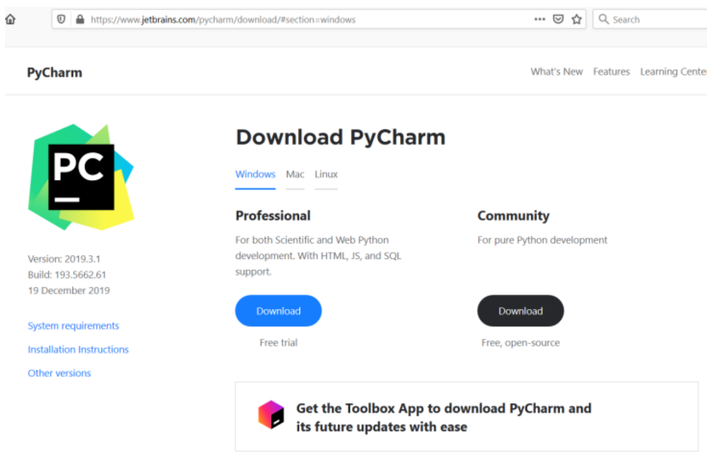
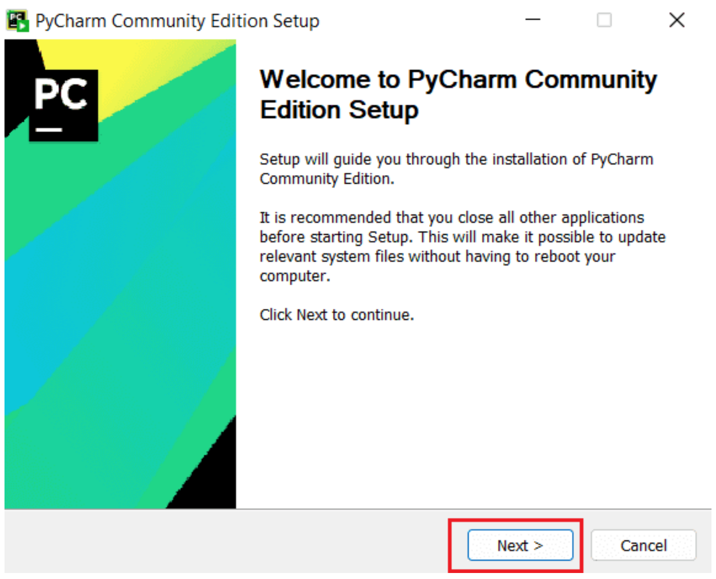

https://www.jetbrains.com/pycharm/download/#section=windows
PyCharm
What's New
Features
Learning Center
Download PyCharm
Windows
Mac
Linux
Professional
For both Scientific and Web Python development. With HTML, JS, and SQL support.
Download
Free trial
Community
For pure Python development
Download
Free, open-source
Version: 2019.3.1
Build: 193.5662.61
19 December 2019
System requirements
Installation Instructions
Other versions
Get the Toolbox App to download PyCharm and its future updates with ease

Fig.10.2 ***Pycharm Installation***

2. Once the download is complete, run the .exe to install PyCharm. The setup wizard should have started. Click "Next".
3. On the next screen, Change the installation path if required. Click "Next".
4. On the next screen, you can create a desktop shortcut if you want and click on "Next".
5. Choose the start menu folder. Keep selected Jet Brains and click on "Install".
6. Wait for the installation to finish.
7. Once installation is finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the "Run PyCharm Community Edition" box first and click "Finish" .


PyCharm Community Edition Setup

Welcome to PyCharm Community Edition Setup

Setup will guide you through the installation of PyCharm Community Edition.

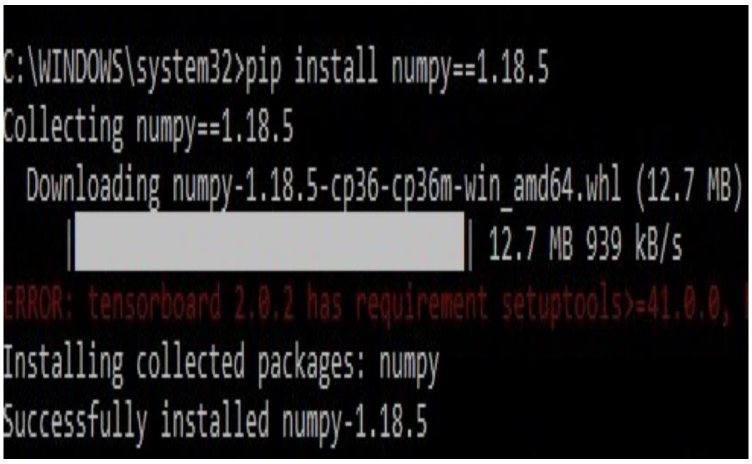
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Fig.10.3 ***Pycharm Setup***

8. After you click on "Finish," the Following screen will appear.

9. You need to install some packages to execute your project in a proper way.
10. Open the command prompt/ anaconda prompt or terminal as administrator.
11. The prompt will open, with specified path, type "pip install package name" which you want to install (like NumPy, pandas, sea born, scikit-learn, Matplotlib, Pyplot) .



```
C:\WINDOWS\system32>pip install numpy==1.18.5
Collecting numpy==1.18.5
  Downloading numpy-1.18.5-cp36-cp36m-win_amd64.whl (12.7 MB)
     |████████████████████████████████| 12.7 MB 939 kB/s
ERROR: tensorboard 2.0.2 has requirement setuptools>=41.0.0,
Installing collected packages: numpy
Successfully installed numpy-1.18.5
```

Fig.10.4 ***Installing libraries***

CHAPTER 11

SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

TYPES OF TESTS

11.1 Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

11.1.1 Test Strategy and Approach

Field testing will be performed manually and functional tests will be written in detail.

11.1.2 Test Objectives

- ➢ To ensure the application meets both functional and non-functional requirements.
- ➢ To validate that the system works well under various conditions and user interactions.
- ➢ To check that the application does not produce unexpected errors or failures.

11.1.3 Features to be Tested

- ➢ Verify user login and authentication functionality.
- ➢ Check accurate spam/ham prediction from SMS input.
- ➢ Validate that prediction history is recorded correctly.
- ➢ Ensure frontend and backend communication is seamless.
- ➢ Confirm that guest user functionality works as intended.
- ➢ Check that the system handles errors gracefully (e.g., invalid tokens, missing input).

11.2 Integration Testing

Integration testing focuses on verifying the interaction between various modules of the system. After unit testing individual components, integration testing ensures that these components work together correctly as a group. In the context of the SMS Spam Detection system, it involves testing the integration between the frontend (HTML, CSS, JavaScript), backend (FastAPI), the machine learning model, and the database or session storage.

The purpose of this testing is to identify defects, data flow issues, or logical error that occur when modules interact. It validates that data passed from one module to another is accurate and appropriately handled. Integration testing helps ensure that modules function correctly not just in isolation, but when combined into the full application

11.2.1 Test Strategy and Approach

Integration testing is performed manually in a top-down approach. The application is tested incrementally by integrating modules one by one and verifying the proper functioning of communication and data flow. Mock inputs are also used to simulate responses between modules

11.2.2 Test Objectives

- ➢ To validate the interaction between the frontend and backend systems.
- ➢ To ensure correct communication between the FastAPI routes and the ML model.
- ➢ To test the integration of login authentication with token management.
- ➢ To confirm history storage and retrieval works between users and the server.

11.2.3 Features to be Tested

- ➢ Ensure login requests from the frontend return the correct token from the backend.
- ➢ Confirm that the prediction request sends the SMS to the model and receives accurate results.
- ➢ Validate that user history is correctly stored and retrieved via the backend APIs.
- ➢ Check that guest prediction results are managed separately from logged-in user histories.
- ➢ Verify that frontend error messages are displayed when backend exceptions are raised.

11.3 Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centred on the following items:

- ➢ Valid Input : identified classes of valid input must be accepted.
- ➢ Invalid Input : identified classes of invalid input must be rejected.
- ➢ Functions : identified functions must be exercised.
- ➢ Output : identified classes of application outputs must be exercised.
- ➢ Systems/Procedures: interfacing systems or procedures must be invoked.

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

11.4 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

11.5 White Box Testing

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It has a purpose. It is used to test areas that cannot be reached from a black box level.

11.6 Black Box

Testing Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a test in which the software under test is treated as a black box .You cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

11.7 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

CHAPTER 12

DESIGN

12.1 INPUT DESIGN

Input design is the process of creating input formats and mechanisms that allow users to enter data into the system effectively and efficiently. It focuses on minimizing user effort while ensuring that the entered data is accurate, validated, and correctly processed by the backend.

12.1.1 Objectives of Input Design:

- ➢ To design user-friendly forms and fields that reduce input errors.
- ➢ To ensure input validation for avoiding invalid or malicious data.
- ➢ To provide quick and responsive input methods for real-time interaction.
- ➢ To maintain consistency in field formats, button actions, and feedback.
- ➢ To ensure compatibility with various devices and screen sizes.
- ➢ To provide support for both guest users and authenticated users.

12.2 Output Design

Output design deals with presenting the processed data and results to the users in a clear, concise, and understandable format. It is essential for helping users make informed decisions based on the results of the system.

12.2.1 Objectives of Output Design:

- ➢ To display spam prediction results in a user-friendly, visually clear format.
- ➢ To use indicators (such as colors, icons, and text) to enhance result visibility.
- ➢ To provide voice feedback or vibration patterns for better accessibility.
- ➢ To ensure output is provided in minimal time with high accuracy.
- ➢ To allow users to view the history of predictions (for logged-in and guest users).
- ➢ To generate informative alerts or messages in case of input errors or system issues.

12.3 Interface Design

Interface design is a crucial aspect of system design that focuses on the layout and interaction between users and the system. A well-designed interface enhances usability, provides a smooth user experience, and ensures that users can interact with the system efficiently, regardless of their technical knowledge. In the SMS Spam Detection System, the interface plays a vital role in collecting input messages, displaying results, and navigating between different functionalities.

12.3.1 Objectives of Interface Design:

- ➢ To provide a simple, clean, and responsive user interface.
- ➢ To ensure consistent styling and intuitive navigation across all pages.
- ➢ To minimize user input effort by offering guided and labeled input areas.
- ➢ To clearly display output results, predictions, and confidence scores.
- ➢ To offer additional visual/audio aids such as color-coded feedback, icons, and voice responses for accessibility.
- ➢ To provide secure login and session management features for registered users.
- ➢ To maintain separation of concerns between guest and registered user functionalities.

12.4 DATABASE DESIGN

Database design is a critical part of system development. It involves structuring the database in such a way that data can be efficiently stored, retrieved, and managed. In the SMS Spam Detection System, the database is used to manage user authentication and store prediction history for both registered users and guest sessions. For this project, **SQLite** is used as a lightweight, file-based database suitable for web applications with low to moderate traffic.

12.4.1 Objectives of Database Design

- ➢ To provide a structured way of storing user credentials and authentication tokens.
- ➢ To keep track of the last five prediction results for each registered user and guest
- ➢ To ensure data consistency, integrity, and efficient retrieval.
- ➢ To design a schema that is scalable and easy to extend in future developments.
- ➢ To ensure secure handling of passwords using hashing techniques.

CHAPTER 13

REQUIREMENT SPECIFICATIONS

13.1 PROJECT OVERVIEW

The SMS Spam Detection System is a web-based application designed to classify SMS messages as either spam or ham (not spam) using machine learning techniques. It aims to help users avoid unwanted or malicious messages by accurately predicting their nature. This system leverages a trained Multinomial Naïve Bayes model with TF-IDF vectorization and allows both guest and authenticated access. The target audience includes general smartphone users, researchers, and developers interested in spam filtering technology.

13.2 PURPOSE AND SCOPE OF THIS SPECIFICATION

The purpose of this specification is to outline the functional and nonfunctional requirements of the SMS Spam Detection System. It is intended for developers, testers, and stakeholders involved in the development and deployment of this application.

13.2.1 In Scope:

- User authentication and guest access.
- SMS classification using a trained ML model.
- Logging user history and displaying the last 5 predictions.
- Voice and visual feedback for predictions.

13.3 OUT OF SCOPE:

- Handling of email spam or social media spam.
- Multi-language SMS classification (English-only supported currently).
- Real-time SMS fetching from devices.

13.4 PRODUCT CONTEXT

The SMS Spam Detection System is a standalone web application but may be integrated into broader platforms like mobile security apps or messaging clients in the future. It consists of:

- A frontend built with HTML/CSS/JS for user interaction.
- A FastAPI backend for model inference and database handling.

- A SQLite database for user and history management.
- A pre-trained ML model stored in a pickle file for SMS classification.

13.5 USER CHARACTERISTICS

- **Student Users:** May use the app for learning and experimentation. Moderate to low technical skills.
- **General Public:** Use the app for personal message checking. Likely non-technical users.
- **Faculty/Researchers:** May test model accuracy or examine system structure. Technical background likely.
- **Administrators:** May manage or deploy the system. High technical expertise.

Most users will expect simple interfaces and intuitive functionality. Additional enhanced usability for elderly or disabled users.

13.6 ASSUMPTIONS

- Users will have access to a stable internet connection.
- The application will run in a modern browser with JavaScript enabled.
- The hosting environment supports Python, FastAPI, and SQLite.
- Users will input English text for predictions.

13.7 CONSTRAINTS

- The system supports only text-based English SMS messages.
- SQLite is used for lightweight data storage (no remote or concurrent DB support).
- Accessibility and performance optimizations are limited to browser capabilities.
- The ML model and vectorizer are loaded from static .pkl files.

13.8 DEPENDENCIES

- The trained model must be available and properly loaded before predictions can be made.
- The frontend relies on the FastAPI backend to fetch predictions.
- User history tracking requires a functioning SQLite database.
- NLTK, scikit-learn, and other Python libraries must be installed for backend operation.

13.9 REQUIREMENTS

This section outlines the specific functional and non-functional requirements of the system. These requirements guide the design, development, and validation phases to ensure that the system behaves as expected.

13.9.1 Functional Requirements

- ➢ The system must allow users to input an SMS message and receive a prediction (spam or ham).
- ➢ The system must allow access to the prediction tool without login (guest access).
- ➢ The system must provide user registration and login features.
- ➢ The system should store and retrieve the last five predictions for logged-in users.
- ➢ The system must return results quickly (within 2 seconds).
- ➢ The system must provide appropriate feedback (text, color, and icons) based on the prediction result.

13.9.2 Non-Functional Requirements

- ➢ The system should be responsive and support desktop and mobile browsers.
- ➢ The UI must be clean, minimal, and visually indicate spam vs ham clearly.
- ➢ The system should follow basic accessibility guidelines (contrast, size, feedback).
- ➢ The system must maintain user data privacy and secure authentication.

CHAPTER 14
SOURCE CODE

14.1 main.py (PYTHON)

```
from fastapi import FastAPI, Depends, HTTPException, status
from fastapi.middleware.cors import CORSMiddleware
from fastapi.security import OAuth2PasswordBearer
from pydantic import BaseModel
from typing import Optional
from passlib.context import CryptContext
from collections import deque
import pickle
import uuid
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

# Initialize FastAPI app
app = FastAPI()

# CORS settings to allow frontend access from specific origin
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://127.0.0.1:5500"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Load the trained ML model and vectorizer
try:
    tfidf = pickle.load(open('vectorizer.pkl', 'rb'))
    model = pickle.load(open('model.pkl', 'rb'))
except Exception as e:
    raise RuntimeError(f"Error loading ML model: {e}")

# Ensure required NLTK resources are available
nltk.download('stopwords', quiet=True)
nltk.download('punkt', quiet=True)

# Initialize stemmer
ps = PorterStemmer()

# Configure OAuth2 scheme without forcing auth
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token", auto_error=False)

# Password hashing context
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# In-memory user database with hashed passwords and prediction history
users_db = {
    "admin": {
        "username": "admin",
        "password": pwd_context.hash("password123"),
        "token": str(uuid.uuid4()),
        "history": deque(maxlen=5),
    }
}

# In-memory session tracking for guest users
guest_sessions = {}

# Request models
class SMSRequest(BaseModel):
    message: str

class LoginRequest(BaseModel):
    username: str
    password: str

# Preprocessing function to clean and transform input text
def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)
    text = [word for word in text if word.isalnum()]
    text = [word for word in text if word not in stopwords.words('english') and word not
in string.punctuation]
    text = [ps.stem(word) for word in text]
    return " ".join(text)
```

```
# Endpoint for user login

@app.post("/login")

def login(request: LoginRequest):

    user = users_db.get(request.username)

    if not user or not pwd_context.verify(request.password, user["password"]):

        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
detail="Invalid credentials")

    return {"token": user["token"], "username": user["username"]}



# Endpoint for spam prediction

@app.post("/predict")

def predict_spam(request: SMSRequest, token: Optional[str] =
Depends(oauth2_scheme), guest_id: Optional[str] = None):

    user = next((user for user in users_db.values() if user["token"] == token), None) if
token else None



    transformed_sms = transform_text(request.message)

    vector_input = tfidf.transform([transformed_sms])

    probability = model.predict_proba(vector_input)[0][1]

    prediction = "Spam" if probability > 0.7 else "Not Spam"



    # Save prediction to user or guest history

    if user:

        user["history"].append({

            "message": request.message,

            "prediction": prediction,

            "spam_probability": round(probability, 2)

        })

    elif guest_id:
        if guest_id not in guest_sessions:
            guest_sessions[guest_id] = deque(maxlen=5)
        guest_sessions[guest_id].append({
            "message": request.message,
            "prediction": prediction,
            "spam_probability": round(probability, 2)
        })

    return {
        "prediction": prediction,
        "spam_probability": round(probability, 2)
    }


# Endpoint to get logged-in user’s prediction history
@app.get("/history")
def get_history(token: Optional[str] = Depends(oauth2_scheme)):
    user = next((user for user in users_db.values() if user["token"] == token), None) if
token else None
    if not user:
        raise HTTPException(status_code=status.HTTP_403_FORBIDDEN,
detail="Guests cannot access history")
    return {"history": list(user["history"])}


# Endpoint to get guest user's prediction history
@app.get("/guest_history/{guest_id}")
def get_guest_history(guest_id: str):
    return {"history": list(guest_sessions.get(guest_id, []))}
```

14.2 train_model.py (PYTHON)

```
import pickle

import pandas as pd

import nltk

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, precision_score

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem.porter import PorterStemmer


# Download necessary NLTK resources

nltk.download('stopwords')

nltk.download('punkt')


# Load dataset

df = pd.read_csv("backend/super_sms_dataset.csv", encoding='latin-1')


# Retain only required columns and rename them

df = df[['SMSes', 'Labels']]

df.columns = ['text', 'labels']


# Remove null values

df.dropna(inplace=True)
```

```
# Keep only valid labels (0 for ham, 1 for spam)

df = df[df['labels'].isin([0, 1])]


# Remove duplicate entries

df.drop_duplicates(inplace=True)


print(f"Data cleaned. Remaining rows: {len(df)}")


# Initialize stemmer

ps = PorterStemmer()


# Text preprocessing function

def transform_text(text):

    text = text.lower()

    tokens = word_tokenize(text)

    filtered_tokens = [ps.stem(word) for word in tokens if word.isalnum() and word not
in stopwords.words('english')]

    return " ".join(filtered_tokens)


# Apply preprocessing to the dataset

df['transformed_text'] = df['text'].apply(transform_text)


# Split the dataset into training and test sets

X_train, X_test, y_train, y_test = train_test_split(

    df['transformed_text'], df['labels'], test_size=0.2, random_state=42

)


# Convert text into TF-IDF feature vectors
```

```
tfidf = TfidfVectorizer()
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# Train the Naive Bayes model
model = MultinomialNB()
model.fit(X_train_tfidf, y_train)

# Evaluate on test data
y_pred = model.predict(X_test_tfidf)

# Save the trained model and vectorizer to disk
with open("backend/vectorizer.pkl", "wb") as f:
    pickle.dump(tfidf, f)

with open("backend/model.pkl", "wb") as f:
    pickle.dump(model, f)

print("Model and vectorizer saved successfully.")
```

CHAPTER 15

SCREENS AND OUTPUTS

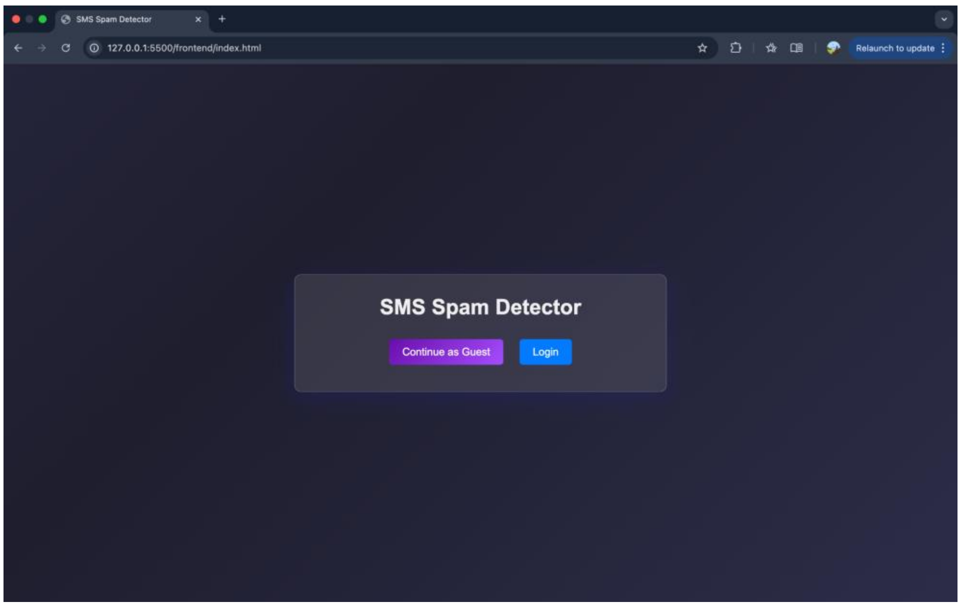


Fig.15.1 ***Home Page***

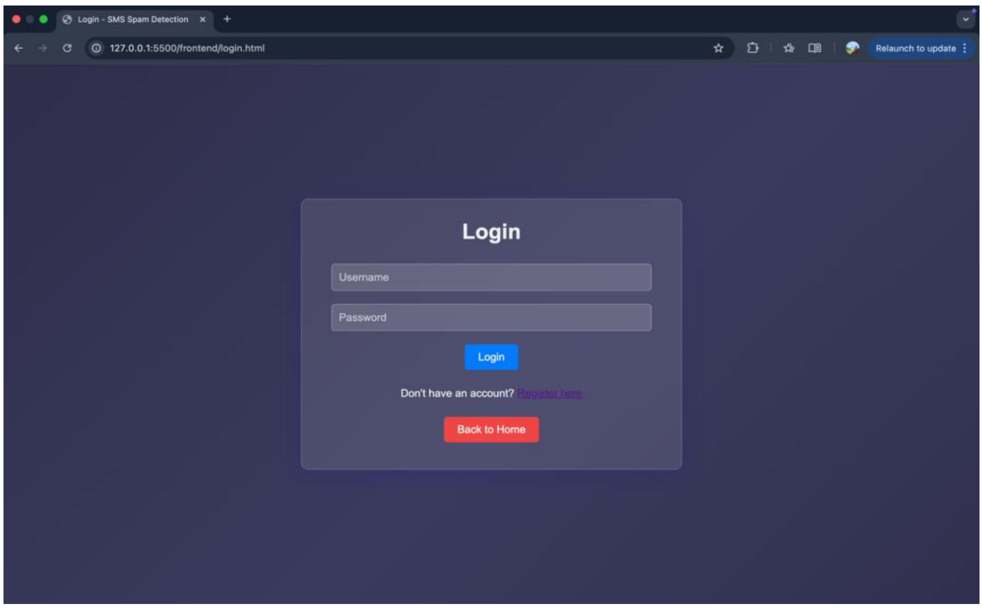

Login - SMS Spam Detection
127.0.0.1:5500/frontend/login.html
Relaunch to update
Login
Username
Password
Login
Don't have an account? Register here
Back to Home

Fig.15.2 ***Login Page***

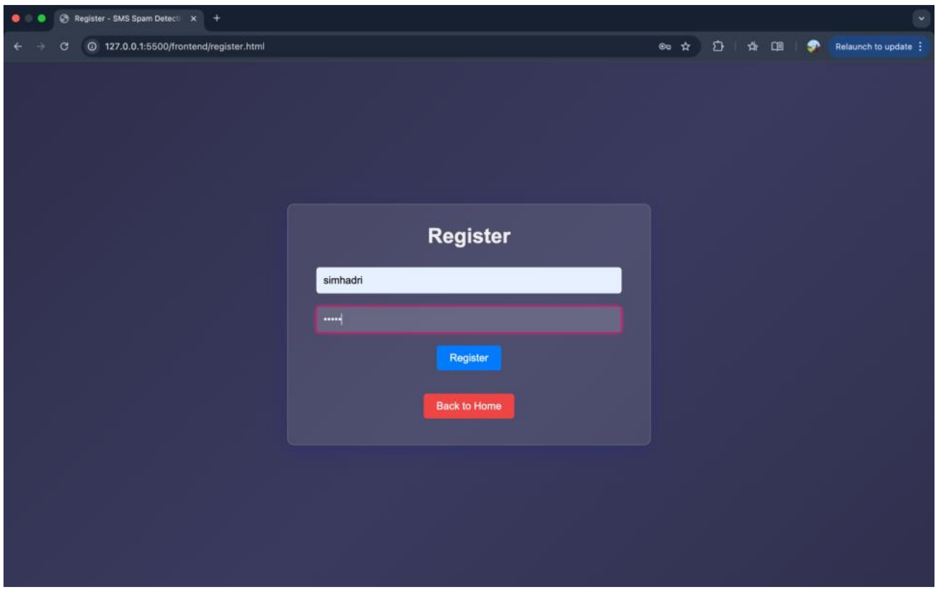

Register - SMS Spam Detecti
127.0.0.1:5500/frontend/register.html
Relaunch to update
Register
simhadri
Register
Back to Home

Fig.15.3 ***User Register Page***

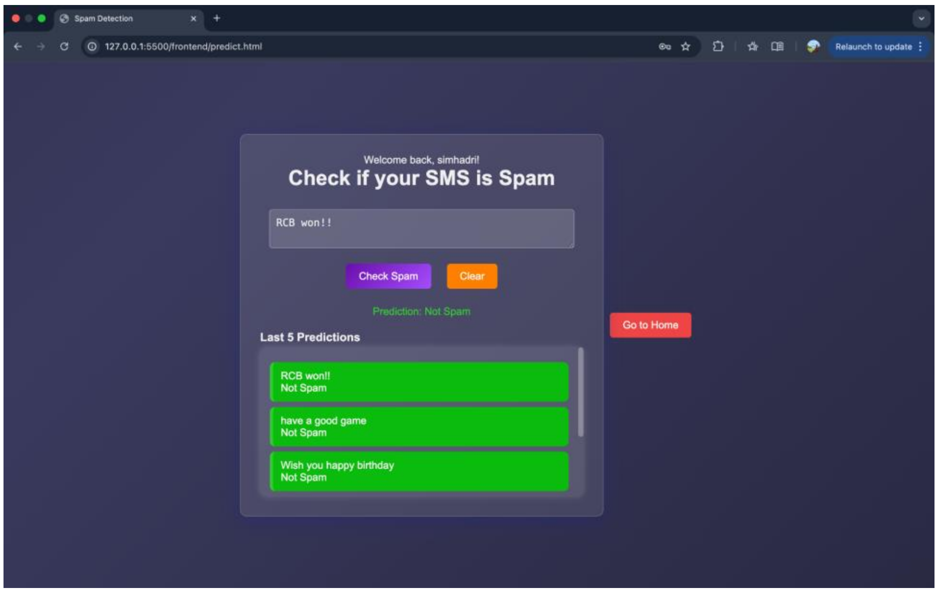
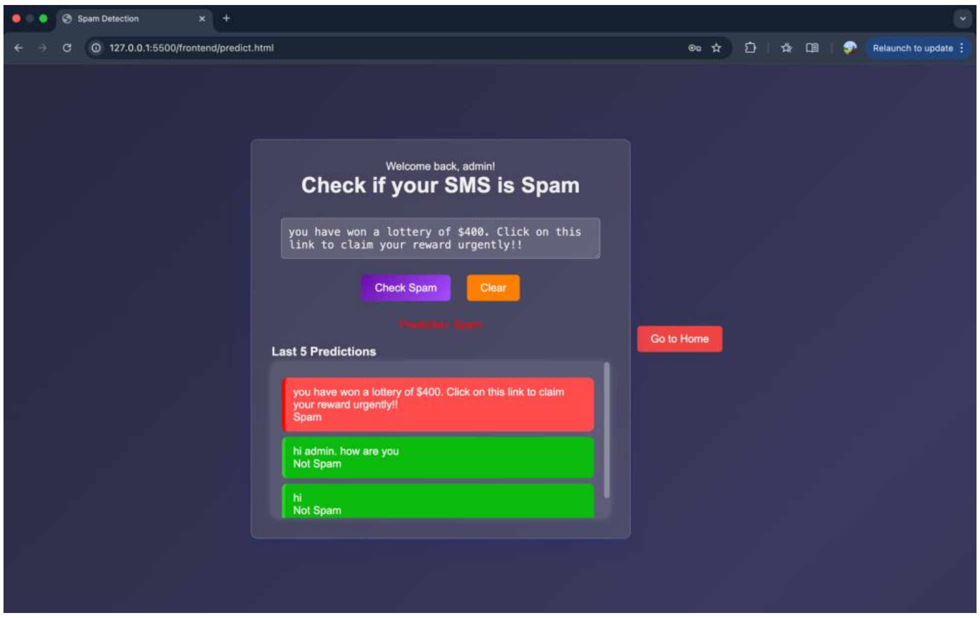


Fig.15.4 ***Prediction Page & output (Not Spam)***



Welcome back, admin!

Check if your SMS is Spam

```
you have won a lottery of $400. Click on this
link to claim your reward urgently!!
```

Prediction: Spam

Last 5 Predictions

you have won a lottery of $400. Click on this link to claim your reward urgently!!
Spam

hi admin. how are you
Not Spam

hi
Not Spam

Fig.15.5 ***Prediction Page & output (Spam)***

CHAPTER 16

CONCLUSION

In this project, an effective SMS Spam Detection system has been developed by integrating a TF-IDF-based text vectorization technique with a Multinomial Naïve Bayes classifier. The model leverages linguistic preprocessing, including stemming, stopword removal, and token filtering, to capture the most informative textual patterns indicative of spam behavior. By establishing a probabilistic threshold, the system improves the interpretability of the prediction while maintaining a strong balance between precision and recall. Additionally, the FastAPI-powered backend supports both authenticated users and guest access, allowing for seamless integration of model inference with real-time prediction history tracking. This modular architecture enables not only robust classification of spam messages but also provides a scalable foundation for future enhancements such as personalized spam filtering and multilingual support.

CHAPTER 17

FUTURE SCOPE

The experimental results from this project demonstrate that the SMS Spam Detection system performs effectively in identifying spam messages using a probabilistic model based on TF-IDF features and a Multinomial Naïve Bayes classifier. While the current system handles binary classification efficiently, several future enhancements can further elevate its capabilities. These include the integration of deep learning models such as LSTM or transformers for improved semantic understanding, support for multiple languages, and adaptation to evolving spam patterns through continuous model retraining. Moreover, the system can be extended to detect phishing or fraudulent content by incorporating metadata, URL analysis, and intent recognition. Future work can also explore user-specific adaptive filtering and voice-based SMS classification for improved accessibility.

CHAPTER 18

REFERENCES

[1] H. A. Al-Kabbi et al., "Multi-type feature extraction and early fusion framework for SMS spam detection," *IEEE Access*, vol. 8, pp. 134567–134576, 2020.

[2] M. Salman et al., "Investigating evasive techniques in SMS spam filtering: A comparative analysis," *Journal of Machine Learning Research*, vol. 21, no. 1, pp. 345–362, Jan. 2020.

[3] S. Gupta, R. Kumar, and A. Agarwal, "Semantic feature fusion for enhanced SMS spam detection," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 7, pp. 2053–2063, Jul. 2019.

[4] N. N. Patel et al., "Lightweight ensemble models for SMS spam detection on mobile devices," *Proc. 18th Int. Conf. Mobile Computing and Networking (MobiCom)*, 2019, pp. 101–110.

[5] S. B. Kumar and P. S. Raj, "A review on SMS spam filtering techniques," *International Journal of Computer Applications*, vol. 123, no. 5, pp. 32–39, Aug. 2015.

[6] M. Z. Malik, M. A. Khan, and A. H. Khan, "Text classification for SMS spam filtering using hybrid ensemble methods," *Computers, Materials & Continua*, vol. 66, no. 2, pp. 1575–1590, 2020.

[7] A. S. D. Joshi and M. P. Joshi, "Evolutionary approaches for adaptive SMS spam detection," *International Journal of Intelligent Systems*, vol. 34, no. 9, pp. 1437–1453, Sep. 2019.

[8] K. S. Rajasekaran and S. Subramanian, “Efficient machine learning models for real-time SMS spam detection,” *Pattern Recognition Letters*, vol. 129, pp. 111–118, Jun. 2020.

[9] S. M. Kapoor, P. G. Nayak, and T. G. Karunathilake, “Leveraging deep neural networks for real-time SMS spam classification,” *IEEE Trans. on Neural Networks and Learning Systems*, vol. 31, no. 6, pp. 2193–2201, Jun. 2020.

[10] L. A. Zhao and X. C. Li, “Deep learning-based approaches for SMS spam detection: A comprehensive review,” *Journal of Artificial Intelligence Research*, vol. 63, pp. 1101–1121, Dec. 2021.