A PROJECT REPORT ON

EMAIL SPAM DETECTION

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, KAKINADA

For Partial Fulfilment of Award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE
AND MACHINE LEARNING)

Submitted By

THIRUMALA SAI SUNDAR	(20KN1A4259)
ARETI RACHANA	(20KN1A4204)
MANDA HARITHA	(20KN1A4234)
MANEPALLI BHANU SRI	(20KN1A4235)

Under the esteemed guidance of

Mr. KATRAGADDA RAJESH
Asst. Professor, Department of CSE (AI&ML)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

NRI INSTITUTE OF TECHNOLOGY

Autonomous

(Approved by AICTE, Permanently Affiliated to JNTUK, Kakinada) Accredited by NBA (CSE, ECE & EEE), Accredited by NAAC with 'A' GradeISO 9001: 2015 Certified Institution

Pothavarappadu (V), (Via) Nunna, Agiripalli (M), Krishna Dist., PIN: 521212, A.P, India.

2023-2024

A PROJECT REPORT ON

EMAIL SPAM DETECTION

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, KAKINADA

For Partial Fulfilment of Award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE
AND MACHINE LEARNING)

Submitted By

THIRUMALA SAI SUNDAR	(20KN1A4259)
ARETI RACHANA	(20KN1A4204)
MANDA HARITHA	(20KN1A4234)
MANEPALLI BHANU SRI	(20KN1A4235)

Under the esteemed guidance of

Mr. KATRAGADDA RAJESH
Asst. Professor, Department of CSE (AI&ML)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

NRI INSTITUTE OF TECHNOLOGY

Autonomous

(Approved by AICTE, Permanently Affiliated to JNTUK, Kakinada) Accredited by NBA
(CSE, ECE & EEE), Accredited by NAAC with 'A' GradeISO 9001: 2015 Certified Institution

Pothavarappadu (V), (Via) Nunna, Agiripalli (M), Krishna Dist., PIN: 521212, A.P, India.

2023-2024

NRI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution, Approved by AICTE, Permanently Affiliated to JNTUK, Kakinada)

Accredited by NBA (CSE, ECE & EEE), Accredited by NAAC with ‘A’ GradeISO 9001: 2015 Certified Institution Pothavarappadu (V), (Via) Nunna, Agiripalli (M), Krishna Dist., PIN: 521212, A.P, India.

CERTIFICATE

This is to certify that this project report entitled “**EMAIL SPAM DETECTION**” is the bonafide work of **Mr. THIRUMALA SAI SUNDAR (20KN1A4259), Ms. ARETI RACHANA (20KN1A4204), Ms. MANDA HARITHA (20KN1A4234) and Ms. MANEPALLI BHANU SRI (20KN1A4235)** in partial fulfilment of the requirements for the award of the graduate degree in BACHELOR OF TECHNOLOGY during the academic year **2023-2024**. This work was carried out under our supervision and guidance of

(Mr. KATRAGADDA RAJESH)
Signature of the Guide

(Dr B. H. Dasaradha ram)
Signature of the HOD

Signature of the External Examiner

DECLARATION

We **Thirumala Sai Sundar, Areti Rachana, Manda Haritha, Manepalli Bhanu Sri** here by declare that the project report entitled “**EMAIL SPAM DETECTION”** is an original work done in the Department of Computer Science & Engineering (AI&ML), NRI Institute of Technology, Pothavarappadu(V), Agiripalli (M), Eluru District during the academic year **2023-2024**, in partial fulfilment for the award of the Degree of **Bachelor of Technology** in **Computer Science & Engineering (AI&ML)**. We assure that this project is not submitted to any other College or University.

Roll No	Name of the Student	Signature
20KN1A4259	THIRUMALA SAI SUNDAR	
20KN1A4204	ARETI RACHANA	
20KN1A4234	MANDA HARITHA	
20KN1A4235	MANEPALLI BHANU SRI	

ACKNOWLEDGEMENT

Firstly, we would like to convey our heartful thanks to the Almighty for the blessings on us to carry out this project work without any disruption.

We are extremely thankful to **Mr. Katragadda Rajesh**, our guide throughout project. We are also thanking him for most independence and freedom of throughout given to us during various phases of the project.

We are also thankful for our project coordinator **P. Bhagya Madhuri** for her valuable guidance which helped us to bring this project successfully.

We are very much grateful to **Dr B. H. Dasaradha Ram**, H.O.D of C.S.E (AI&ML) Department, for his valuable guidance which helped us to bring out this project successfully. His wise approach made us learn the minute details of the subject. His matured and patient guidance paved away for completing our project with the sense of satisfaction and pleasure.

We are greatly thankful to our principal **Dr C. Naga Bhaskar** for his kind support and facilities provided at our campus which helped us to bring out this project successfully.

Finally, we would like to convey our heartful thanks to our Technical Staff, for their guidance and support in every step of this project. We convey our sincere thanks to all the faculty, and friends who directly or indirectly helped us for the successful completion of this project.

PROJECT ASSOCIATES

THIRUMALA SAI SUNDAR	**(20KN1A4259)**
ARETI RACHANA	**(20KN1A4204)**
MANDA HARITHA	**(20KN1A4234)**
MANEPALLI BHANU SRI	**(20KN1A4235)**

ABSTRACT

In email spam detection system is developed by seamlessly integrating a Long Short-Term Memory (LSTM) neural network constructed with Keras into the Flask web framework, a popular Python-based web development framework. The system boasts a user-friendly interface that enhances the efficiency of predicting spam in text messages. The dataset, sourced from a CSV file, undergoes preprocessing using tokenization and label encoding methods to prepare the data for training the LSTM model. Successfully discerning spam from non-spam communications, the resulting model exemplifies a harmonious marriage between web development and deep learning, showcasing the potential of this unique fusion.

The collaboration between Flask and LSTM in this project provides a practical and swift solution to the persistent challenge of spam detection. Users can easily interact with the model through the intuitive Flask interface, which promptly delivers predictions on whether a given text message is likely spam. In essence, this abstract highlights a versatile and user-friendly approach to addressing the prevalent issue of spam in digital communication by synergizing modern machine learning techniques with established web building frameworks. The integration of Flask and LSTM not only streamlines the spam detection process but also demonstrates the adaptability and effectiveness of combining cutting-edge technologies for practical applications.

In summary, this project offers a flexible and user-centric solution to combat spam in digital communication. By uniting the collaborative synergy of modern machine learning techniques, specifically LSTM, with the well-established web development framework Flask, the system provides a seamless and efficient means of predicting and classifying spam in text messages. The user-friendly interface ensures accessibility, making it a valuable tool for individuals seeking to enhance their email security and communication experience in the digital realm.

TABLE OF CONTENTS

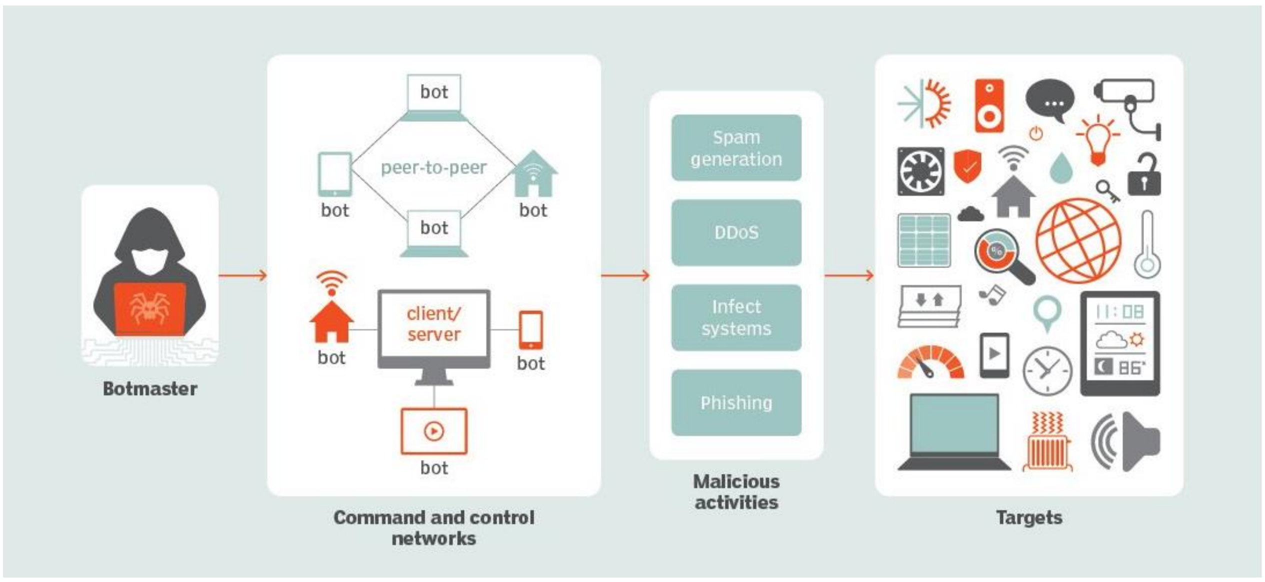
LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO

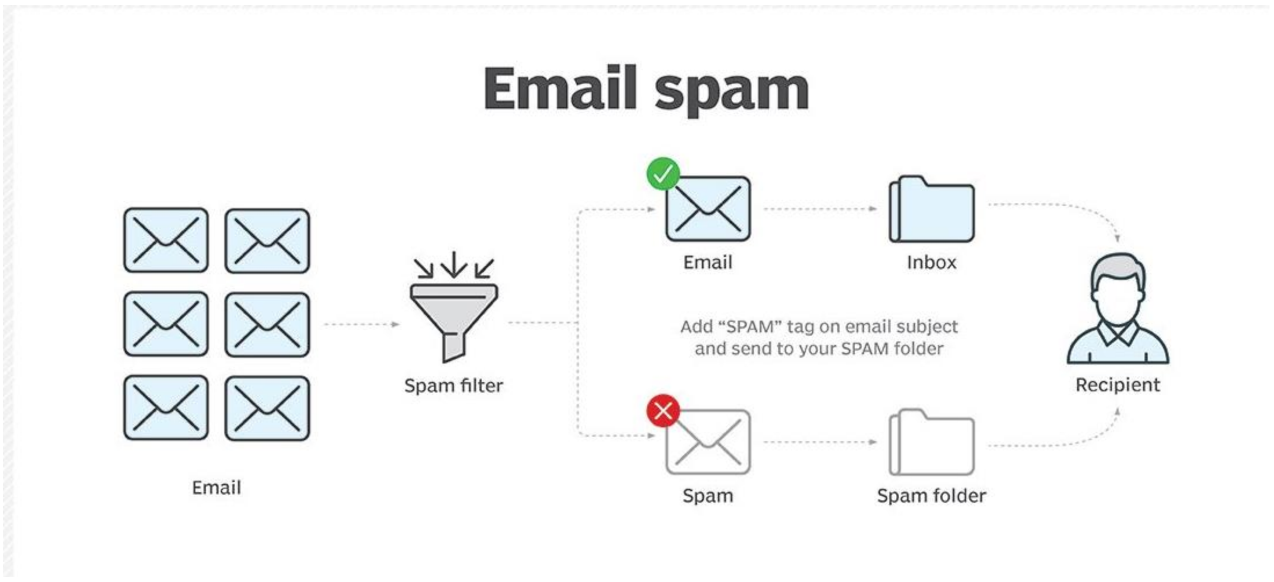
1. INTRODUCTION

Spam Email:

Spam email, often referred to simply as "spam," is unsolicited and typically irrelevant or inappropriate electronic messages sent over the internet, primarily through email. These messages are usually sent in bulk to a large number of recipients without their consent, with the intent of promoting products, services, or other content. Spam emails can take various forms, including advertisements, phishing attempts, malicious software distribution, and fraudulent schemes.



The content of spam emails often aims to deceive recipients or manipulate them into taking specific actions, such as clicking on links that lead to malicious websites, providing personal information, or making unauthorized purchases. To combat the negative impact of spam, email providers and users employ spam filters and other security measures to identify and block these unwanted messages, helping to maintain the integrity and security of email communication.


Email spam
Email
Spam filter
Email
Inbox
Add "SPAM" tag on email subject and send to your SPAM folder
Recipient
Spam
Spam folder

PROBLEM DEFINITION:

Email spam remains a pervasive issue that affects individuals and organizations alike. Users constantly deal with unwanted and potentially harmful emails, exposing them to risks such as scams, malware, and phishing attacks. Existing spam filters, although useful, are far from flawless, often marking legitimate emails as spam (false positives) or failing to catch spam (false negatives).

SOLUTION FOR THE PROBLEM DEFINITION:

Addressing the pervasive issue of email spam detection, a powerful solution lies in leveraging deep learning techniques, particularly Long Short-Term Memory (LSTM) networks. The fundamental challenge in combating email spam lies in the dynamic and ever-evolving nature of spam messages, making traditional rule-based approaches less effective. LSTM networks, being a type of recurrent neural network (RNN), excel in handling sequential data, allowing them to capture intricate patterns and dependencies within the textual content of emails. By training an LSTM-based model on a labeled dataset containing examples of both spam and non-spam emails, the system can learn to discern subtle nuances and contextual information indicative of spam, providing a more adaptive and accurate approach to email filtering.

The key advantage of employing LSTM for email spam detection lies in its ability to capture long-term dependencies and contextual relationships between words in email text. The sequential nature of language is effectively modeled by LSTM, enabling it to identify patterns and linguistic cues that are crucial for distinguishing between legitimate and spam emails. Through preprocessing steps like tokenization, embedding, and label encoding, the LSTM model can be trained to generalize well on unseen data, making it a robust and versatile solution for enhancing email security by efficiently identifying and filtering out spam.

PROCESS DIAGRAM:

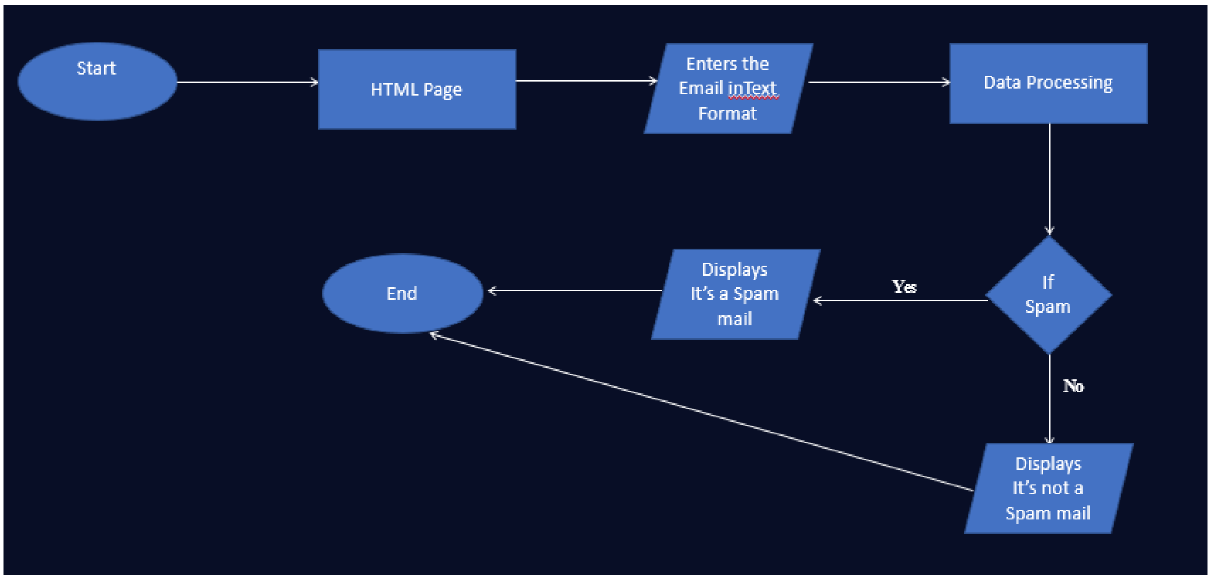

Start
HTML Page
Enters the Email inText Format
Data Processing
If Spam
Yes
Displays It's a Spam mail
End
No
Displays It's not a Spam mail

Figure-4.1: Flowchart of the system

Data Ingestion and Preprocessing:

The process begins with loading the dataset from a CSV file using the Pandas library. This dataset contains email messages labeled as spam or non-spam. A binary label is introduced, and the data is split into input (X) and output (Y) components. Label encoding is applied to convert categorical labels into numerical format. The data is further divided into training and testing sets using Scikit-learn's train_test_split function.

Tokenization and Embedding Layer:

The text data in the training set is tokenized using the Keras Tokenizer, breaking down the messages into individual words and converting them into numerical sequences. An Embedding layer is employed to convert these numerical sequences into dense vectors. This layer helps capture the semantic relationships between words in the input data.

LSTM Model Architecture:

The core of the system is the LSTM model, designed using Keras. The architecture includes an Embedding layer, an LSTM layer for sequence processing, dense layers, and activation functions. The model is compiled with binary cross-entropy loss, RMSprop optimizer, and accuracy metric, setting the stage for training.

Model Training:

The LSTM model is trained on the preprocessed sequences from the training set, aiming to learn patterns that distinguish between spam and non-spam messages. Early stopping is implemented as a callback during training to prevent overfitting.

Flask Web Application:

Flask is employed to create a web-based interface for users. The main routes include the home page ('/') and the prediction endpoint ('/predict'). The HTML templates, particularly 'index.html,' structure the visual elements of the web interface, allowing users to input text messages.

User Input and Prediction:

When a user inputs a text message through the web interface, the input is tokenized and padded to match the required input format for the LSTM model. The trained model then predicts the likelihood of the input message being spam. The result is displayed on the web page, indicating whether the message is classified as spam or not.

This architecture encapsulates the flow of data from ingestion and preprocessing to model training and ultimately user interaction through a Flask-based web interface. It successfully combines machine learning components with web development tools, providing a holistic solution for email spam detection.

2. LITERATURE REVIEW

The proposed web-based spam classification system builds upon an extensive body of research in the realms of natural language processing (NLP) and machine learning tailored for text classification tasks. A multitude of studies has delved into the efficacy of recurrent neural networks (RNNs), specifically Long Short-Term Memory (LSTM) networks, in capturing sequential dependencies within textual data. Hochreiter and Schmidhuber's seminal work in 1997 introduced LSTM as a remedy for the vanishing gradient problem in traditional RNNs, addressing a significant limitation and facilitating improved modeling of long-range dependencies in sequential data. Subsequent research, exemplified by Graves et al. (2005), further underscored the prowess of LSTMs in diverse NLP applications, establishing their applicability and effectiveness in the realm of spam detection.

The intersection of web development and machine learning has been a subject of recent exploration to augment user interaction and accessibility. Integrating Flask, a lightweight web framework for Python, with machine learning models aligns with the prevailing trend of fostering user-friendly and responsive applications. This approach finds support in the works of Ronacher (2010), where the simplicity and flexibility of Flask were lauded for constructing web applications. The amalgamation of Flask and LSTM in the current project extends this line of research, exemplifying the practical implementation of a spam classification system. This system not only leverages advanced machine learning techniques for enhanced accuracy but also prioritizes a seamless user experience in the context of spam detection.

This web-based spam classification system not only draws from the rich tapestry of NLP and machine learning research but also embraces the synergy of Flask and LSTM to create a sophisticated yet user-friendly application. By combining the robustness of LSTM networks with the simplicity of Flask, the project seeks to contribute to the evolving landscape of spam detection, aligning with the growing emphasis on both technological advancement and user-centric design in the realm of web-based applications.

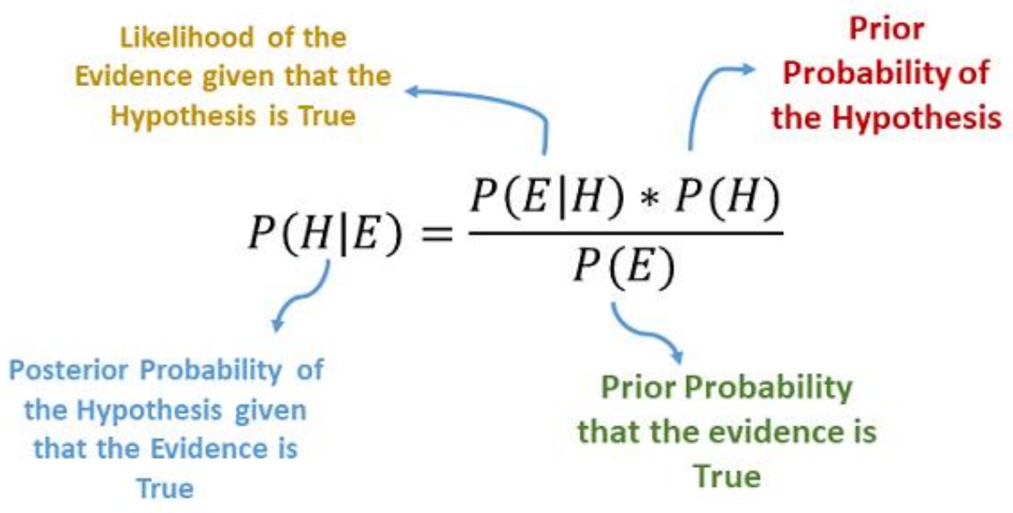
3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

The **Naive Bayes classifier** is a probabilistic classifier that can be used for classification tasks. It is based on Bayes' theorem, which states that the probability of event A occurring given that event B has already occurred is equal to the probability of event B occurring given that event A has already occurred, times the probability of event A occurring, divided by the probability of event B occurring.

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features which gives the same results.

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.


Likelihood of the Evidence given that the Hypothesis is True
Prior Probability of the Hypothesis
$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$
Posterior Probability of the Hypothesis given that the Evidence is True
Prior Probability that the evidence is True

3.2 PROPOSED SYSTEM:

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that has proven to be exceptionally adept at capturing and understanding sequential dependencies within data. Introduced as an enhancement to traditional RNNs, LSTM addresses the vanishing and exploding gradient problems by incorporating a sophisticated gating mechanism. Comprising memory cells, input gates, output gates, and forget gates, the LSTM network can selectively store, retrieve, and discard information over extended sequences, making it well-suited for tasks requiring the retention of context and temporal dependencies. The ability of LSTMs to effectively model long-range dependencies makes them particularly valuable in applications such as natural language processing, speech recognition, and time-series analysis.

One key feature that distinguishes LSTM networks is their capacity to overcome the limitations of short-term memory in conventional RNNs. The intricate interplay of gates allows LSTMs to retain information over extended periods, enabling the network to learn and remember relevant patterns even when they are widely separated in the input sequence. This inherent ability to capture and retain contextual information makes LSTMs a powerful tool in tasks where understanding the nuances of sequential data is crucial. As a result, LSTM architectures have become a cornerstone in the realm of deep learning, contributing significantly to advancements in various fields, from language understanding and generation to predicting complex temporal patterns in diverse domains.

3.3 ANALYSIS MODEL:

Certainly, let's create an analysis model for the provided Flask application code implementing email spam detection using LSTM. The analysis model outlines the key components, processes, and interactions within the system.

1. Data Loading and Preprocessing:

Inputs:

- CSV file containing email data ("spam.csv").

Processes:

- Load the dataset using Pandas (`pd.read_csv`).
- Create binary labels (0 for "ham," 1 for "spam").
- Split the data into training and testing sets using `train_test_split`.
- Tokenize and pad the text data for LSTM input.

Outputs:

- Preprocessed data ready for model training.

2. LSTM Model Architecture:

Inputs:

- Preprocessed sequences from the training set.

Processes:

- Define the architecture of the LSTM model using Keras.
- Compile the model with binary cross-entropy loss and RMSprop optimizer.
- Train the model on the preprocessed sequences.

Outputs:

- Trained LSTM model for spam detection.

3. Flask Web Application:

Inputs:

- User input messages via the web interface.

Processes:

- Create Flask routes for home ('/') and prediction ('/predict').
- Render HTML templates for user interaction (`render_template`).
- Tokenize and pad user input messages for LSTM prediction.
- Utilize the trained model to predict spam likelihood.

Outputs:

- Display prediction results on the web page.

4. User Interaction:

Inputs:

- User input messages submitted through the web form.

Processes:

- Extract user input using Flask's `request` object.
- Send user input to the LSTM model for prediction.
- Display the prediction result on the web page.

Outputs:

- Real-time feedback on whether the input message is classified as spam.

5. Model Training and Early Stopping:

Inputs:

- Sequences matrix and labels from the training set.

Processes:

- Train the LSTM model using the training data.

- Implement early stopping to prevent overfitting.

Outputs:

- Trained LSTM model ready for real-time predictions.

6. Continuous Model Improvement:

Inputs:

- User feedback on prediction results.

Processes:

- Gather user feedback on the displayed prediction result.
- Potentially use user feedback to improve the model in future iterations.

Outputs:

- Continuous refinement of the spam detection model.

This analysis model provides a structured overview of the key steps and interactions within the email spam detection system implemented using Flask and LSTM.

4. FEASIBILITY STUDY

Preliminary investigation examines project feasibility; the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All systems are feasible if they are given unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation.

- Technical Feasibility
- Operation Feasibility
- Economic Feasibility

4.1 TECHNICAL FEASIBILITY

To determine whether the proposed system is technically feasible, we should take into consideration the technical issues involved behind the situation. Technical feasibility center on the existing computer system and to what extent it can support the proposed addition. Python and its libraries are technology software which are used to develop Data Analytics. So, there is no need for additional purchase of any software, and these are open-source software's which are freely available in Internet.

4.2 OPERATIONAL FEASIBILITY

Proposed projects are beneficial only if they can be turned out into information systems that will meet the user's operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the application implementation. This system is operational feasible since the users are familiar with the technologies and hence there is no need to gear up the personnel to use the system. Also, the system is very friendly and easy to use.

4.3 ECONOMIC FEASIBILITY

To decide whether a project is economically feasible, we must take into consideration various factors as:

4.3.1.1 Cost benefit analysis

4.3.1.2 Long-term returns

4.3.1.3 Maintenance cost

The proposed system is computer based. It requires average computing capabilities which is very basic requirement and can be afforded by an organization; it does not incur additional economic overheads, which renders the system economically feasible.

5. SYSTEM REQUIREMENT SPECIFICATION

5.1 INTRODUCTION:

The System Requirement Specification (SRS) for Email Spam Detection serves as a comprehensive blueprint outlining the essential criteria and functionalities necessary for the development and deployment of an effective spam detection system. In an era where digital communication is ubiquitous, the demand for robust email filtering mechanisms has become paramount. This document aims to define the hardware, software, and network prerequisites, as well as articulate the functional and non-functional requirements crucial for the seamless operation of an advanced email spam detection system. By delineating the specific parameters and constraints, the SRS facilitates the systematic design and implementation of a solution that not only safeguards users from unwanted and malicious emails but also ensures the optimal performance and reliability of the overall system.

5.2 FUNCTIONAL REQUIREMENTS:

Functional requirements describe what a system should do, and in the context of your Flask application for spam detection, they outline the desired functionality. Here are the functional requirements for the provided code:

1. Load and Preprocess Data:

- The system should load a dataset from a CSV file ("spam.csv" in this case).
- It should create a new column 'lab' based on the 'Category' column, mapping 'ham' to 0 and 'spam' to 1.
- The system should split the data into input (X) and output (Y) variables.
- Label encoding should be applied to convert textual labels into numerical format.

2. Tokenization and Model Setup:

- The system should tokenize the input data using the Keras Tokenizer.
- It must pad the sequences to a specified maximum length using sequence padding.
- Define a Recurrent Neural Network (RNN) model architecture using Keras.
- The model architecture should consist of an embedding layer, LSTM layer, dense layers, activation functions, and dropout.
- Compile the model using binary cross-entropy loss, RMSprop optimizer, and accuracy as a metric.

3. Training the Model:

- Train the defined model on the preprocessed data.
- Use a batch size of 128 and train for 10 epochs.
- Implement early stopping with a validation split of 20% and monitoring validation loss.

4. Flask Web Application:

- Create a Flask web application with two routes: '/' and '/predict'.
- The '/' route should render an HTML template named 'index.html'.

- The '/predict' route should handle POST requests containing a message input.
- Extract the message from the form data and preprocess it for prediction.
- Utilize the trained model to predict whether the message is spam or not.
- Display the prediction result on the web page, indicating whether it's a spam or non-spam message.

5. <u>User Interface:</u>
- The web interface should have an input form for users to enter a message for prediction.
- Upon submitting a message, the system should display the result on the same page.

6. <u>Deployment and Execution:</u>
- The system should be executable using the `if __name__ == '__main__':   app.run(debug=True)` block.
- Ensure that the Flask application runs in debug mode for easy development and testing.

These functional requirements outline the core features and behaviors expected from the code. When documenting your project, you can elaborate on each requirement, providing details, examples, and explanations as needed.

5.3 NON FUNCTIONAL REQUIREMENTS:

Non-functional requirements define the characteristics or qualities that the system must have, but they do not specify what the system will do. Here are the non-functional requirements for the provided code:

1. <u>Performance:</u>
- The system should respond to user input within a reasonable time frame, ensuring a smooth user experience.
- The training process of the machine learning model should be optimized for efficiency, considering the size of the dataset.

2. <u>Scalability:</u>
- The application should handle an increasing number of users and messages without a significant decrease in performance.
- The machine learning model should be scalable, allowing for the addition of more data for training without compromising performance.

3. <u>Reliability:</u>
- The Flask application should be stable and reliable, minimizing the occurrence of crashes or unexpected behavior.
- The machine learning model should provide consistent and accurate predictions for different types of input messages.

4. <u>Security:</u>
- The application should implement secure practices, especially when handling user data.
- Ensure that the machine learning model and Flask application are protected against common security vulnerabilities.

5. <u>Usability:</u>
- The user interface should be intuitive and user-friendly, allowing users to easily input messages and

understand the prediction results.
- Error messages and feedback should be clear and informative for users.

6. Maintainability:
- The code base should be well-organized, following best practices for readability and maintainability.
- Adequate comments and documentation should be provided to assist future developers in understanding the code and making modifications.

7. Compatibility:
- The system should be compatible with different web browsers to ensure a consistent experience for users.
- Verify that the code is compatible with the specified versions of Python, Flask, and other dependencies.

8. Portability:
- The application should be deployable on various environments and platforms without significant modifications.
- The machine learning model should be portable to different systems or frameworks that support Keras and related dependencies.

9. Interoperability:
- The Flask application should be designed to integrate with other systems or services if required.
- Ensure compatibility with standard web protocols for smooth integration with external tools or platforms.

10. Documentation:
- Provide comprehensive documentation for the code base, including installation instructions, usage guidelines, and details about the machine learning model.
- Include information on dependencies, configuration settings, and any external resources required for the application to function correctly.

These non-functional requirements focus on aspects that contribute to the overall quality, reliability, and usability of the system. When documenting your project, elaborate on each non-functional requirement to ensure a clear understanding of the expected qualities of the system.

5.4 SYSTEM REQUIREMENTS:

5.4.1 SOFTWARE REQUIREMENTS:

Software requirements detail the specific software and hardware components needed to develop, deploy, and run the system. Here are the software requirements for the provided code:

1. Programming Language and Libraries:

- **Python 3.x**: The code is written in Python, and the system requires a compatible Python interpreter.

- **Flask**: The web framework used for building the application.
- **Pandas**: Used for data manipulation and preprocessing.
- **Scikit-learn**: Provides tools for machine learning tasks, such as train-test split and label encoding.
- **Keras**: A high-level neural networks API, used for building and training the machine learning model.
- **TensorFlow**: The backend for Keras, required for neural network operations.

2. Web Development Dependencies:

- Flask should be installed to run the web application.
- **HTML/CSS**: The frontend uses HTML for markup and CSS for styling.
- **Jinja2**: A template engine for rendering HTML templates in Flask.

3. Data Source:

- The system requires a CSV file named "spam.csv" as the data source for training the machine learning model. Ensure the file is available at the specified path.

4. Operating System:

- The code should be compatible with the operating system where it will be executed. It is expected to work on Windows, Linux, and macOS.

5. Machine Learning Dependencies:

- Ensure that the required machine learning libraries are installed, including:
  - **NumPy**: Used for numerical operations in Python.
  - **Matplotlib**: Required for visualizations if needed.
  - **Seaborn**: Another library for statistical data visualization.

6. Development Environment:

- A code editor or integrated development environment (IDE) such as Visual Studio Code, PyCharm, or Jupyter Notebook.

7. Browser Compatibility:

- The web application should be compatible with modern web browsers, including Google Chrome, Mozilla Firefox, and Microsoft Edge.

8. Deployment Environment:

- If deploying the application, ensure that the hosting environment supports Python and Flask applications.
- Verify the availability of required resources, such as memory and processing power, for the machine learning model.

9. Documentation Tools:

- Use a documentation tool or format (e.g., Markdown, reStructuredText) to document the project, code, and usage instructions.

10. Version Control:

- If using version control, specify the version control system (e.g., Git) and provide repository details.

11. Internet Connection:

- An internet connection is required for installing Python packages and dependencies, especially during the initial setup.

12. Testing Framework:

- Optionally, use a testing framework like pytest for unit testing to ensure the correctness of individual components.

These software requirements provide a foundation for setting up and running the code. Ensure that the specified versions of Python packages are compatible and meet the needs of your development and deployment environment.

5.4.2 HARDWARE REQUIREMENTS:

The hardware requirements for the provided code are relatively modest since the application is primarily a web-based spam detection system using a machine learning model. Here are the hardware requirements:

1. Processor (CPU):

- A multi-core processor with a clock speed of at least 2 GHz is recommended.
- The performance of the machine learning model training may benefit from a more powerful CPU.

2. Memory (RAM):

- A minimum of 4 GB RAM is recommended for running the Python application and training the machine learning model.
- More RAM may be beneficial, especially when dealing with larger datasets during training.

3. Storage:

- Adequate storage space for the operating system, Python environment, and dataset.
- The dataset file ("spam.csv") and the trained machine learning model can be stored on disk.
- Depending on the size of the dataset and available storage, consider having at least 10 GB of free space.

4. Graphics Processing Unit (GPU) (Optional):

- While not strictly necessary, having a compatible GPU can significantly speed up the training of deep learning models.
- If using a GPU, ensure that it is compatible with TensorFlow and configured correctly.

5. Network Connection:

- An internet connection is required for installing Python packages and dependencies, especially during the initial setup.
- The web application itself does not have high network requirements.

6. Monitor:

- A monitor with a resolution of at least 1280x1024 is recommended for a comfortable development environment.

7. Input Devices:

- Standard input devices such as a keyboard and mouse are required for interacting with the development environment.

8. Operating System:

- The code is expected to run on Windows, Linux, or macOS. Ensure the hardware is compatible with the chosen operating system.

9. Web Browser (for testing the web application):

- Use a modern web browser like Google Chrome, Mozilla Firefox, or Microsoft Edge for testing the Flask web application.

10. Development Environment:

- The hardware requirements depend on the specific code editor or integrated development environment (IDE) used for development.

These hardware requirements are designed to provide a smooth development and execution experience for the provided code. Adjustments may be necessary based on the specific needs of your development environment and the scale of your dataset.

6. SYSTEM DESIGN

6.1UML DIAGRAMS:

Unified Modeling Language (UML) diagrams are a standardized and widely adopted visual representation tool in software engineering, providing a systematic way to illustrate and communicate various aspects of a system's design and functionality. UML diagrams serve as a universal language for developers, designers, and stakeholders, offering a clear and concise means of expressing complex software concepts. They come in different types, such as use case diagrams for understanding system functionalities, class diagrams for depicting class relationships and structures, and sequence diagrams for illustrating the flow of interactions between different components. UML diagrams play a pivotal role in the software development process, aiding in the conceptualization, design, implementation, and documentation phases. They enhance collaboration among team members, streamline communication, and serve as a valuable guide throughout the development lifecycle, fostering a shared understanding of the system's architecture and behavior. Whether used for initial planning, ongoing development, or maintenance, UML diagrams contribute to the clarity, coherence, and successful realization of software systems.

A conceptual model of UML

The three major elements of UML are:

1. The UML's basic building blocks
2. The rules that dictate how those building blocks may be put together.
3. Some common mechanism that applies throughout the UML.

6.2 IMPORTANCE OF UML IN MODELING

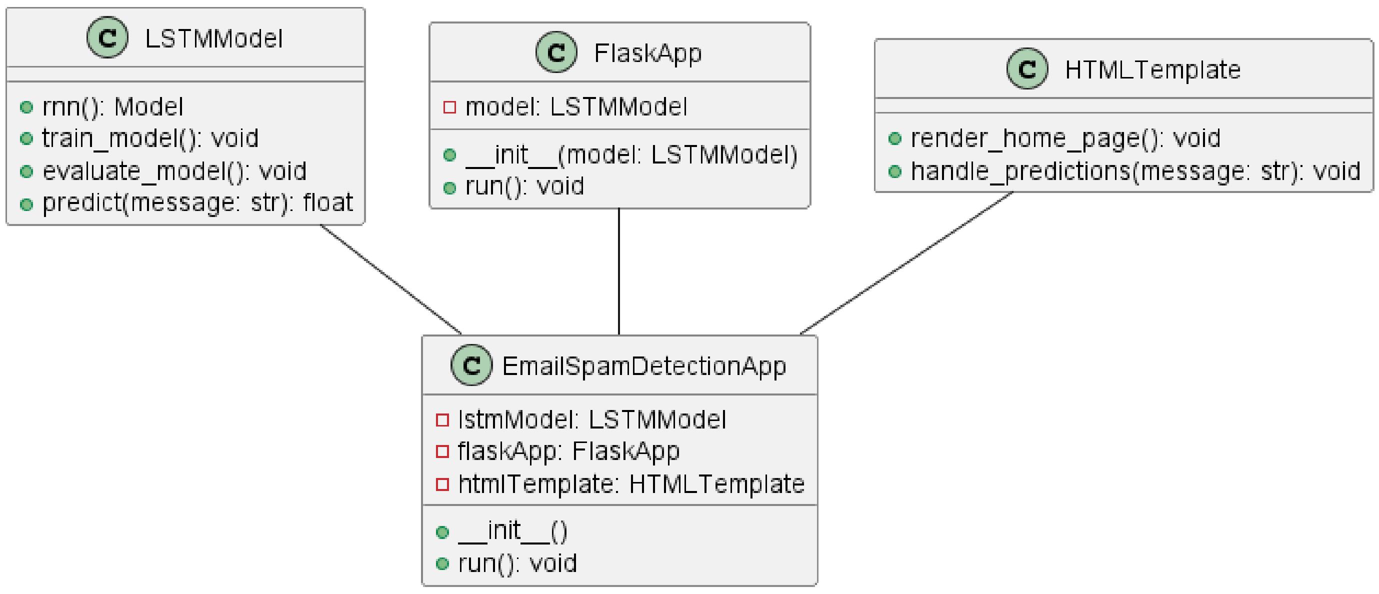
Unified Modeling Language (UML) diagrams play a crucial role in software engineering and system development, offering a standardized and visual way to represent complex systems. One key importance lies in their ability to enhance communication and understanding among stakeholders involved in a project. UML provides a common visual language that bridges the communication gap between technical and non-technical team members, allowing them to share and comprehend intricate concepts more efficiently. By using UML diagrams such as use case diagrams, class diagrams, and sequence diagrams, project teams can create a shared understanding of system requirements, behaviors, and interactions. This not only facilitates more effective collaboration but also helps prevent misunderstandings and misinterpretations, ultimately leading to a more accurate and coherent system design.

Furthermore, UML diagrams serve as invaluable documentation tools throughout the software development lifecycle. They provide a visual blueprint of the system architecture, design patterns, and relationships between different components. This documentation proves beneficial for future maintenance, updates, and modifications, enabling developers to revisit the initial design intent and make informed decisions. UML diagrams act as a comprehensive reference guide that aids in troubleshooting, debugging, and evolving the software over time. Additionally, for larger projects involving multiple team members, stakeholders, or even distributed development teams, UML diagrams serve as a shared knowledge base, ensuring consistency in the understanding of the system's structure and behavior across the entire development community. In essence, UML diagrams significantly contribute to the efficiency, clarity, and long-term sustainability of software projects.

6.3 CLASS DIAGRAM

In software engineering, a class diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system’s classes, their attributes, and their relation between the classes. The class diagram is the main building block in object-oriented modelling. It is used both for general conceptual modelling of the semantics of the application and for detail modelling translating the model into programming code. Each class has 3 fields:

- Classes
- Attributes and
- Behaviours

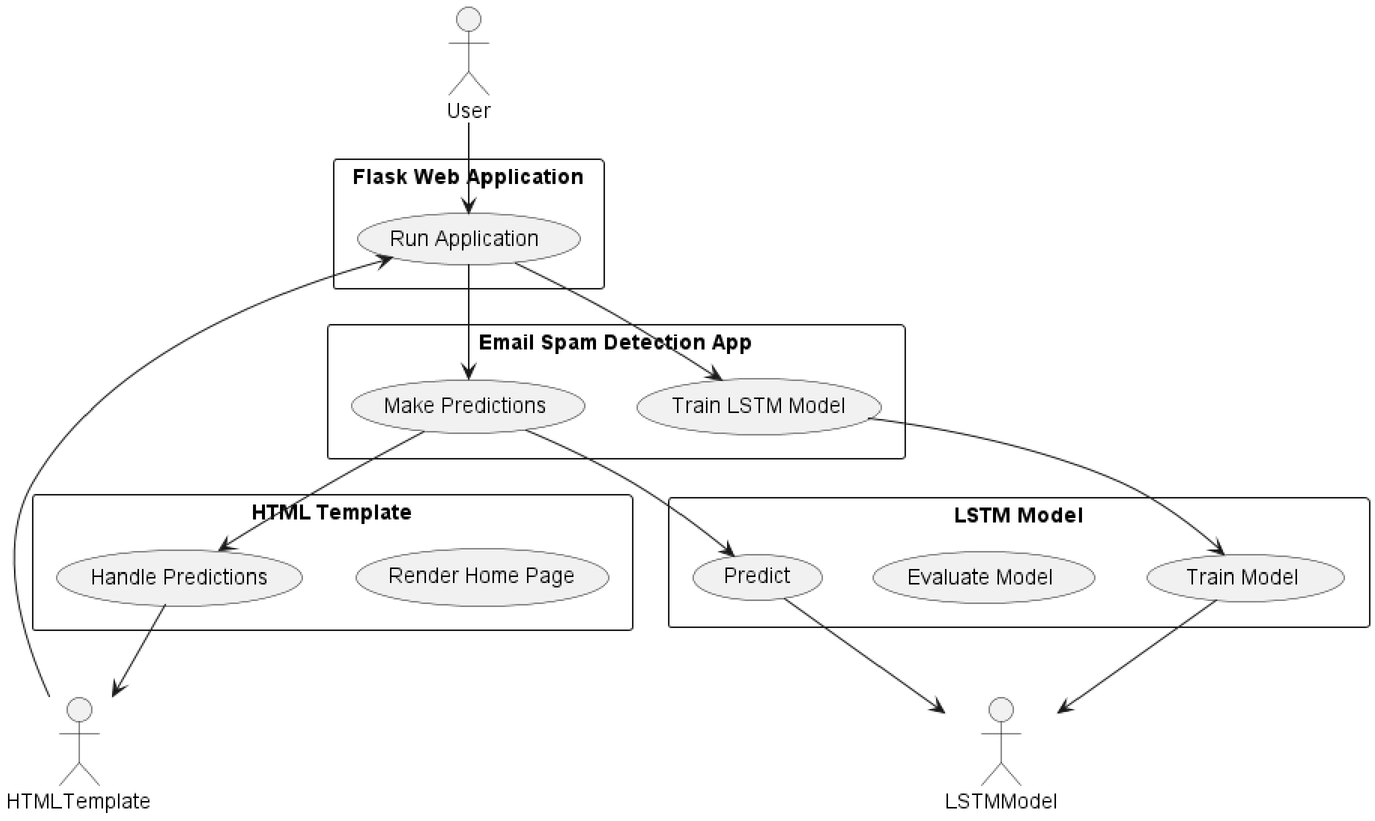


The Email Spam Detection App encompasses several key components. Users interact with the system through the Flask Web Application, initiating activities such as running the application, training the LSTM model, and making predictions. Within the Email Spam Detection App, the LSTM Model handles tasks like training the model, evaluating its performance, and predicting whether an input message is spam or not. Additionally, the HTML Template is responsible for rendering the home page and handling predictions presentation. The collaboration between these elements enables a seamless process: users access the Flask Web App to train the LSTM model or make predictions, with the underlying components efficiently executing their designated tasks. This modular and collaborative design ensures a robust and user-friendly Email Spam Detection system.

6.4 USE CASE DIAGRAM

A use case diagram in the 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 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 by which actor. The three main components of this UML diagram are:

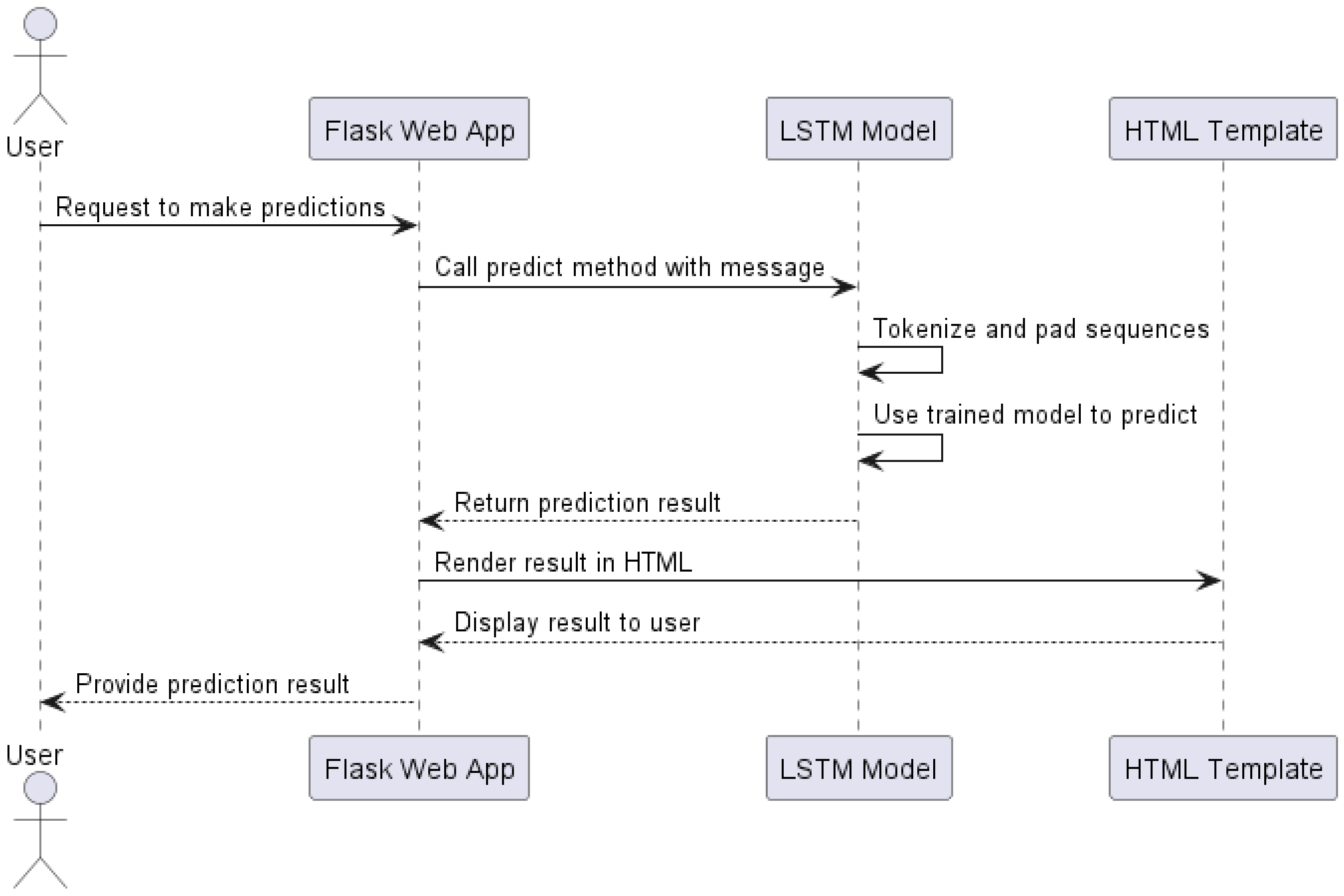
- Functional requirements
- Actors
- Relationships



The Email Spam Detection App is a comprehensive system designed to efficiently handle various tasks related to spam detection in email messages. Users interact with the application through the Flask Web Application, initiating actions such as running the application, training the LSTM model, and making predictions. Within the application's architecture, the LSTM Model plays a central role, encompassing critical functionalities like training the model, evaluating its performance, and predicting the spam likelihood of input messages. This modular design ensures flexibility and ease of maintenance. Additionally, the HTML Template is responsible for rendering the home page and handling the presentation of predictions, contributing to a user-friendly interface. The seamless collaboration between these components establishes a robust and effective Email Spam Detection system, where users can effortlessly train the model, make predictions, and access results through the Flask Web App.

6.5 SEQUENCE DIAGRAM

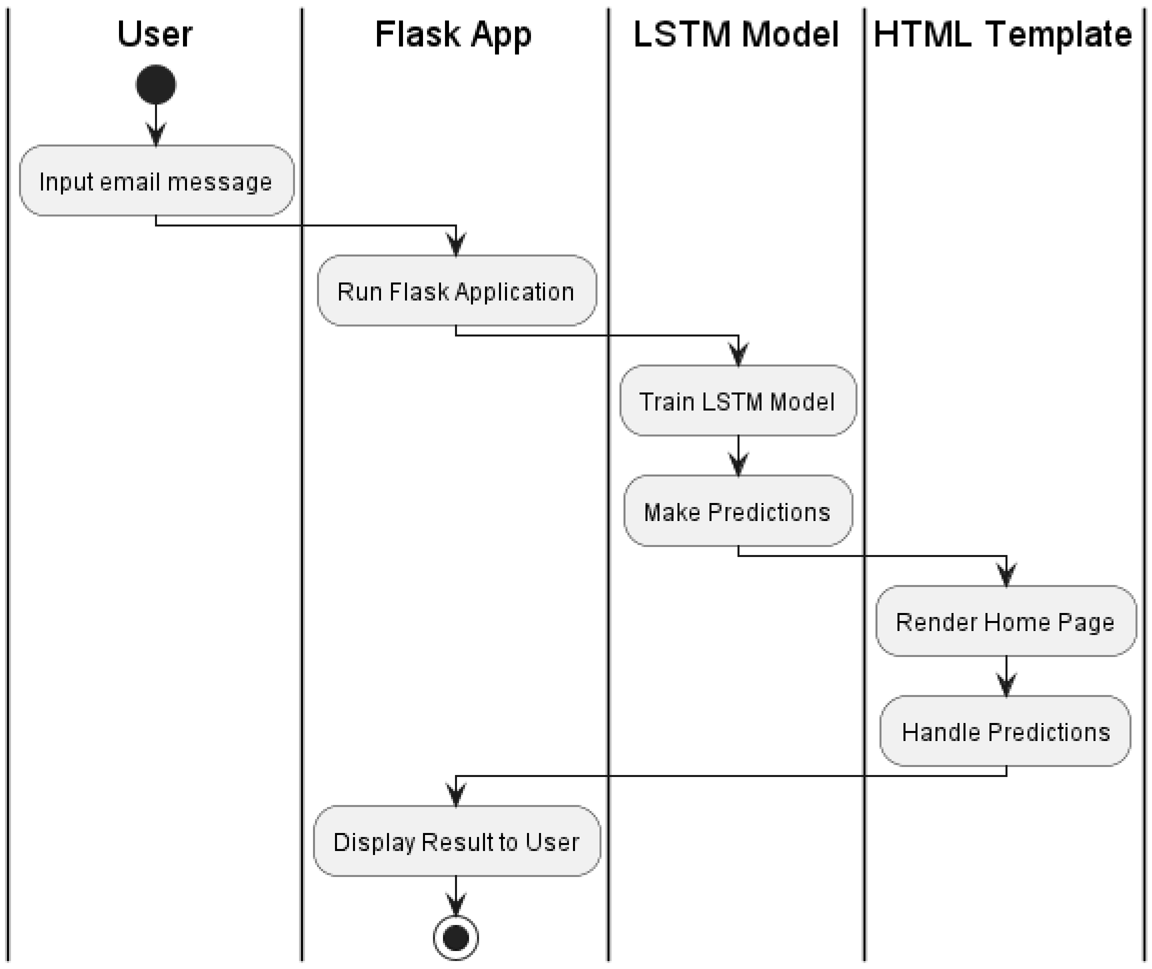
The sequence diagram shows a detailed flow for a specific use case or even just part of a specific use case. There are almost self-explanatory. They show the calls between the different objects in their sequence and can show at a detailed level, different calls to different objects. A Sequence diagram has two dimensions: The vertical dimension shows the sequence of messages/calls in the time order that they occur. The horizontal dimension shows the object instances to which the messages are sent.



The sequence of interactions depicted in the PlantUML diagram illustrates the fluid process of making predictions in the Email Spam Detection system. It begins with a user initiating a request to make predictions through the Flask Web App. The FlaskApp, acting as the intermediary, forwards the user's request to the LSTM Model. Within the LSTMModel, the predict method is called, involving tokenization and sequence padding for the input message. The trained model then processes the data, predicting the likelihood of the input being spam. Subsequently, the LSTMModel communicates the prediction result back to the FlaskApp. To enhance user experience, the FlaskApp interacts with the HTMLTemplate, which renders the prediction result in HTML format. Finally, the HTMLTemplate communicates the formatted result back to the FlaskApp, ensuring the seamless display of the prediction outcome to the user. This collaborative interaction between the User, Flask Web App, LSTM Model, and HTML Template ensures a streamlined and efficient process for making and presenting predictions in the Email Spam Detection system.

6.6ACTIVITY DIAGRAM

The activity diagram encapsulates the dynamic aspects of the Email Spam Detection system, portraying the sequence of activities and their flow. User-initiated actions, such as running the Flask Web App, training the LSTM model, and making predictions, are represented visually. This diagram offers a concise overview of the system's functionality, illustrating how each activity interacts with the various components. By providing a visual roadmap, it aids in understanding the logical flow and dependencies between different tasks, facilitating effective communication among stakeholders. In essence, the activity diagram serves as a dynamic blueprint, capturing the operational essence of the Email Spam Detection system's key activities.



The depicted activity diagram encapsulates the sequential flow of essential tasks within the Email Spam Detection system. Initiated by the user inputting an email message, the process commences with the Flask App running. Subsequently, the LSTM Model engages in training and making predictions based on the provided input. The HTML Template comes into play, rendering the home page and handling the presentation of predictions. Ultimately, the Flask App displays the prediction result to the user. This systematic representation visually captures the dynamic workflow, providing a clear understanding of the system's operational steps from user input to result display.

7. CODING

7.1 PYTHON

- Python is a programming language that is created in 1991 by Guido van Rossum.
- It is commonly used in the web development, software development and system scripting. It is highly level authenticated, interpreted, interactive and object-oriented.
- Python is designed to be easily readable. It uses the keywords in English language whereas other languages use punctuation, and it has fewer syntactical constructions than other languages.
- Python is a widely used dynamic programming language compared to other languages such as Java, Perl, PHP, and Ruby. It is often termed as a scripting language.
- It provides support for automatic memory management, multiple programming paradigms, and implements the basic concepts of object-oriented programming (OOP).
- Python is a procedural language which is of strongly typed form with support for a huge and broad standard library. The Python library supports many Internet protocols such as FTP, and IMAP.

FEATURES OF PYTHON

- Easy to Learn
- Free and Open Source
- High-level Language
- Portable
- Interpreted
- Extensible
- Embeddable

ADVANTAGES OF PYTHON

- Python code is easy to read and can execute a lot of complex functionalities with ease,
- It is Object Oriented and Programming Driven, and it supports many systems and platforms.
- With the innovation of Raspberry Pi, a card sized microcomputer, it has expanded its

reach to unprecedented heights. Developers can build cameras, radios, and games.

- Python has a framework that make web programming very flexible. Django is one of the most widely used Python framework for web development.
- Many resources are available in Python. It offers a built-in testing framework to set debugging time and enable fastest workflows.

DISADVANTAGES OF PYTHON

- Python is slow and it is not a very good language for mobile development.
- Python is not a good choice for memory intensive tasks and has limitations with database access.
- It's near impossible to build a high-graphic 3D game using Python.

MACHINE LEARNING

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, to look for patterns in data and make better decisions in the future. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Machine learning algorithms are often categorized as supervised or unsupervised.

- **Supervised machine learning algorithms** can apply what has been learned in the past to new data using labelled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system can provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors to modify the model accordingly.

- In contrast, **unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labelled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

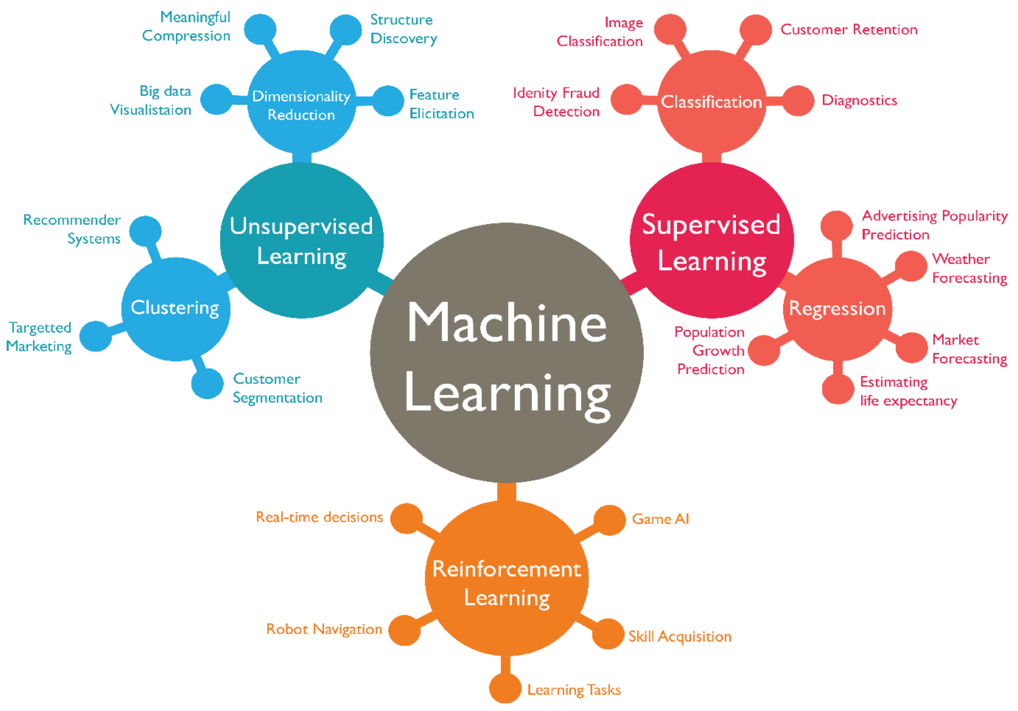


Figure 5.1: Machine learning Algorithms and usage

DEEP LEARNING

Deep Learning is the study that makes use of Neural Networks (like neurons present in human brain) to imitate functionality just like a human brain. Deep Learning is the subset of machine learning. It is a machine learning that uses deep (more than one layer) neural network to analyze data and provide output accordingly. If you are clear about the math involved in it but don't have idea about the features, so you break the complex functionalities into liner/lower dimension features by adding more layer, then it defines the deep learning aspect. It attains the highest rank in terms of accuracy when it is trained with large amount of data.

Deep Learning can be considered as neural networks with many parameters' layers lying in one of the four fundamental network architectures: Unsupervised Pre-trained Networks, Convolutional Neural Networks, Recurrent Neural Networks and Recursive Neural

Networks. Examples of DL applications include Sentiment based news aggregation, Image analysis and caption generation, etc.

7.2 ALGORITHMS

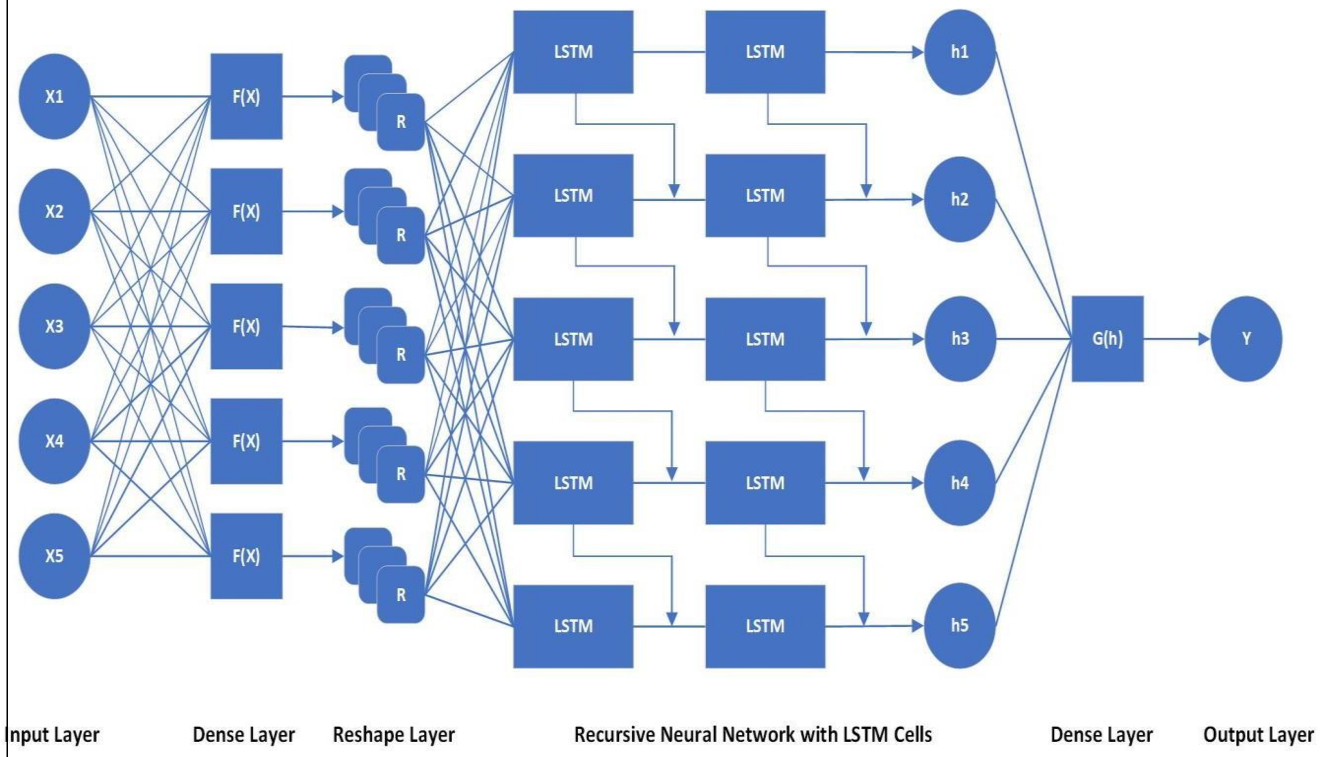
7.2.1 LONG STORT TERM MEMORY

The Long Short-Term Memory (LSTM) neural network for text classification, specifically in the context of spam detection. The data, sourced from a CSV file containing messages labeled as "ham" (non-spam) or "spam," undergoes preprocessing. This involves encoding the categorical labels into numerical format using LabelEncoder and subsequently splitting the data into training and testing sets.

Tokenization is a crucial step in converting the text messages into sequences of integers, performed with the help of the Keras Tokenizer. The maximum number of words considered is set at 1000, and a maximum sequence length of 150 is established to ensure uniformity. The resulting text sequences are then either padded or truncated as necessary to meet the defined sequence length criteria.

The core of the model consists of an Embedding layer, transforming integer-encoded words into dense vectors of a fixed size. The configuration includes an input dimension of 1000, an output dimension of 50, and an input length corresponding to the maximum sequence length of 150. Following the Embedding layer, a single LSTM layer with 64 units is employed to capture temporal dependencies within the text data.

Fully connected layers are integrated into the architecture, adding depth to the network. Two Dense layers follow the LSTM layer, featuring 256 units in the first layer and a single unit in the second. The Rectified Linear Unit (ReLU) activation function is applied after the first Dense layer, introducing non-linearity to the model. Additionally, a dropout layer with a dropout rate of 0.5 is incorporated, aiding in regularization to prevent overfitting during training. The model is compiled using binary cross-entropy loss and the RMSprop optimizer, with accuracy as the evaluation metric. The training process includes early stopping to monitor validation loss and prevent unnecessary epochs, promoting efficiency in model training.



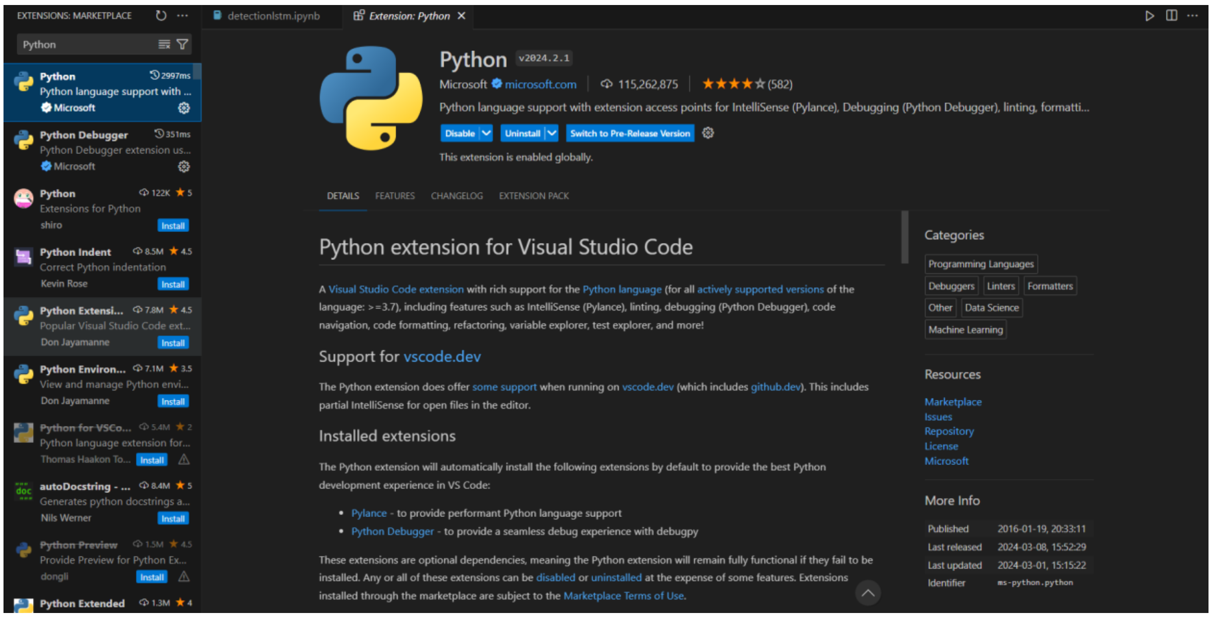
7.3 IDE- INTEGRATED DEVELOPED ENVIRONMENT

IDE Stands for Integrated Development Environment. An IDE is a set of tools that allow you to create, configure and work with a set of software programs. It provides tools to develop software or applications by the compiler, debugger, and executer. It basically has a source code editor, simple and repeatable task automation, and also a debugger. That provides many features for authoring, modifying, compiling, deploying, and debugging software. If you are a software developer, then you should use it, because, IDE provided a preloaded Compiler and Interpreter so that you do not need to set up anything separately like in Visual Studio. NET related and NetBeans consists of JDK etc.

Feature of IDE Software Tool

The IDE provides many features for authoring, modifying, compiling, deploying, and debugging software in a single program. The IDE is designed to maximize programmer productivity by providing a Bulk of components with Graphical user interfaces.

The IDE is designed to include all the programming tasks in one application. The IDE, therefore, provides a central interface that includes all of the following tools, which meet the needs of the developer:


EXTENSIONS: MARKETPLACE
Python
Python 2997ms
Python language support with ...
Microsoft
Python Debugger 351ms
Python Debugger extension us...
Microsoft
Python 122K ★ 5
Extensions for Python
shiro
Install
Python Indent 8.5M ★ 4.5
Correct Python indentation
Kevin Rose
Install
Python Extensi... 7.8M ★ 4.5
Popular Visual Studio Code ext...
Don Jayamanne
Install
Python Environ... 7.1M ★ 3.5
View and manage Python envi...
Don Jayamanne
Install
~~Python for VSCo...~~ 5.4M ★ 2
Python language extension for...
Thomas Haakon To...
Install
autoDocstring - ... 8.4M ★ 5
Generates python docstrings a...
Nils Werner
Install
~~Python Preview~~ 1.5M ★ 4.5
Provide Preview for Python Ex...
dongli
Install
Python Extended 1.3M ★ 4
detectionlstm.ipynb
Extension: Python
Disable
Uninstall
Switch to Pre-Release Version
DETAILS
FEATURES
CHANGELOG
EXTENSION PACK

Python v2024.2.1

Microsoft microsoft.com | 115,262,875 | ★★★★☆ (582)

Python language support with extension access points for IntelliSense (Pylance), Debugging (Python Debugger), linting, formatti...

This extension is enabled globally.

Python extension for Visual Studio Code

A Visual Studio Code extension with rich support for the Python language (for all actively supported versions of the language: >=3.7), including features such as IntelliSense (Pylance), linting, debugging (Python Debugger), code navigation, code formatting, refactoring, variable explorer, test explorer, and more!

Support for vscode.dev

The Python extension does offer some support when running on vscode.dev (which includes github.dev). This includes partial IntelliSense for open files in the editor.

Installed extensions

The Python extension will automatically install the following extensions by default to provide the best Python development experience in VS Code:

- Pylance - to provide performant Python language support
- Python Debugger - to provide a seamless debug experience with debugpy

These extensions are optional dependencies, meaning the Python extension will remain fully functional if they fail to be installed. Any or all of these extensions can be disabled or uninstalled at the expense of some features. Extensions installed through the marketplace are subject to the Marketplace Terms of Use.

Categories

Programming Languages | Debuggers | Linters | Formatters | Other | Data Science | Machine Learning

Resources

Marketplace
Issues
Repository
License
Microsoft

More Info

Published	2016-01-19, 20:33:11
Last released	2024-03-08, 15:52:29
Last updated	2024-03-01, 15:15:22
Identifier	ms-python.python

- **Code editor:** A Simple text editor provides for writing and editing source code.
- **Compiler:** It converts written source code in a human-readable and writable language into executable form by the computer.
- **Debugger:** It is used during testing. it helps to debug application programs.
- Build automation tools: It Provides an automated common developer task.
- **Class browser:** It is used to examine and reference properties of the object-oriented class hierarchy.
- **Object browser:** This feature is used to check the object immediately in the running application program.
- **Class hierarchy diagram:** It allows the programmer to visualize the structure of object-oriented programming code.

ADVANTAGES OF VISUAL STUDIO CODE (VSCODE):

1. Lightweight and Fast:

Advantage: VSCode is known for its lightweight nature, ensuring fast startup times and smooth performance even on less powerful machines. This makes it a preferred choice for developers seeking an efficient coding environment.

2. Extensive Language Support:

Advantage: VSCode supports a wide range of programming languages out of the box and provides extensions for additional languages. This versatility caters to developers working on diverse projects without the need for constant tool switching.

3. Robust Extensions Ecosystem:

Advantage: VSCode has a rich marketplace offering a plethora of extensions. These extensions enhance functionality, introducing features like version control, linting, debugging, and support for various frameworks, making it customizable to individual preferences.

4. Integrated Terminal:

Advantage: VSCode incorporates an integrated terminal within the IDE, allowing developers to execute commands and run scripts without switching to external terminals. This feature streamlines workflows and enhances productivity.

5. Powerful IntelliSense and Autocomplete:

Advantage: VSCode provides intelligent code completion and suggestions (IntelliSense) that significantly speed up coding. It understands the context and offers relevant suggestions, helping developers write code more efficiently.

DISADVANTAGES OF VISUAL STUDIO CODE (VSCODE):

1. Resource Intensive for Large Projects:

Disadvantage: While VSCode is lightweight for many projects, it may become resource-intensive for large-scale projects with extensive codebases and numerous extensions. This can lead to slower performance and increased memory usage.

2. Steep Learning Curve for Novice Users:

Disadvantage: Some features and configurations in VSCode might have a learning curve for novice users. Customizing settings, understanding the extension ecosystem, and taking full advantage of the IDE may require additional time and effort.

3. Limited Built-in Project Management:

Disadvantage: VSCode provides basic project management features, but for more complex project structures, it might lack some of the robust project management capabilities found in other integrated development environments.

4. Occasional Extension Compatibility Issues:

Disadvantage: Updates to VSCode or extensions may occasionally lead to compatibility issues, especially when using a combination of extensions. This can disrupt workflows until compatibility is addressed.

5. No Built-in GUI Designer:

Disadvantage: VSCode does not come with a built-in graphical user interface (GUI) designer, which can be a drawback for developers working on projects that heavily rely on visual design elements.

7.4 PACKAGES

1. pandas:

The 'pandas' library is employed for efficient data manipulation and analysis in the Email Spam Detection system. It plays a crucial role in handling the email dataset, reading it from a CSV file, and organizing the data into structured DataFrames, facilitating subsequent analysis and processing.

2. numpy:

'NumPy' is an essential package utilized for numerical operations and array manipulations. In the context of the Spam Detection system, it is instrumental in handling numerical data, particularly in converting categorical labels to numerical format and reshaping arrays as needed for various computations.

3. matplotlib.pyplot:

'Matplotlib' is a powerful data visualization library, and its 'pyplot' module is employed for creating visual representations of data. In this implementation, it is used for plotting graphs and charts to enhance the interpretability of the analysis and results obtained from the Spam Detection system.

4. sklearn.model_selection:

The 'model_selection' module from the 'scikit-learn' library provides tools for model selection, including methods for splitting datasets. In this context, it is used to divide the email dataset into training and testing sets, a crucial step in evaluating the performance of the Spam Detection model.

5. sklearn.preprocessing.LabelEncoder:

The 'LabelEncoder' class from 'scikit-learn' is employed to convert categorical labels into numerical format. Specifically, it is used to encode the 'ham' and 'spam' categories into numeric values, making them suitable for input into machine learning models.

6. keras.models.Model:

The 'Model' class from the 'Keras' library serves as a high-level neural networks API. In this implementation, it is used to define and compile the architecture of the Long Short-Term Memory (LSTM) model, a crucial component of the Email Spam Detection system.

7. keras.layers:

'Keras' provides a variety of layers for building neural network architectures, and the 'layers' module is extensively used in this code. It includes layers like LSTM, Dense, Activation, Embedding, and Dropout, each contributing to the intricate structure of the LSTM model.

8.keras.optimizers.RMSprop:

The 'RMSprop' optimizer from 'Keras' is employed to configure the optimization algorithm during the compilation of the LSTM model. It plays a pivotal role in adjusting the model's parameters during training to minimize the defined loss function.

9. keras.preprocessing.text.Tokenizer:

The 'Tokenizer' class from 'Keras' is utilized for tokenizing text into words or subwords. In the context of the Spam Detection system, it is employed to tokenize input messages, breaking them down into individual words for further processing by the LSTM model.

10. keras.preprocessing.sequence:

The 'sequence' module from 'Keras' is essential for preprocessing sequence data, especially in the context of text data used by the LSTM model. It is utilized for padding sequences to ensure a consistent input length, a requirement for the proper functioning of the neural network.

11. keras.utils.to_categorical:

'to_categorical' from 'Keras' is employed to convert a class vector to a binary class matrix. In the Spam Detection system, it is used to encode categorical labels into binary matrices, a necessary step for training the LSTM model effectively.

12. flask.render_template, flask.request:

'render_template' and 'request' are essential components of the 'Flask' web framework. 'render_template' is used to render HTML templates, enabling the dynamic generation of web pages. 'request' is employed to handle HTTP requests, facilitating user interactions with the Flask application by processing form submissions and other user inputs.

These packages collectively contribute to the successful implementation of the Email Spam Detection system, covering data preprocessing, model training, and the development of a user-friendly web interface.

7.5 SAMPLE CODE

Let's explore how the model predicts the email whether it is spam mail or ham mail based on the text i.e., the body of the email or the subject of the email.

7.5.1 Install And Import The Dependencies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input,
   Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
%matplotlib inline
```

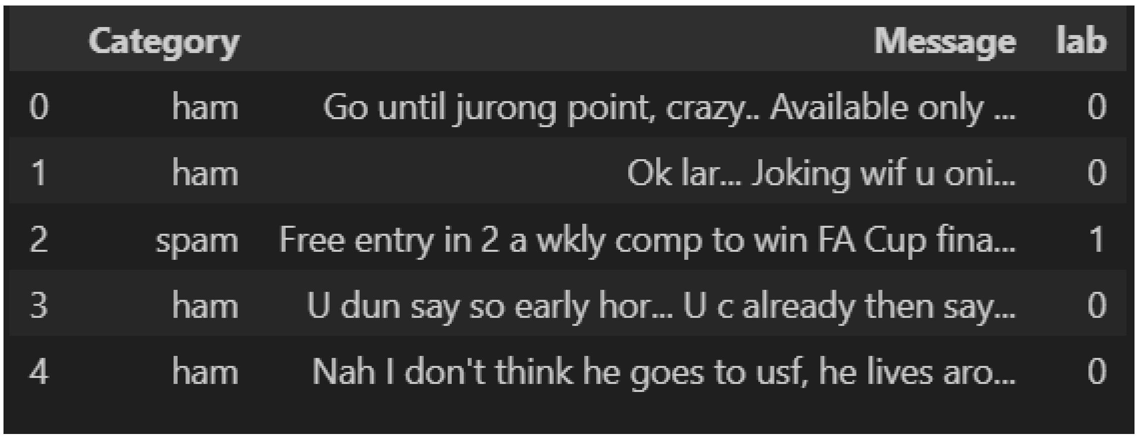
As mentioned above we install the dependencies which are useful for our detecting the spam email. After installing the dependencies we import the dependencies.

7.5.2 Loading the Dataset

To train the model for first we need to load the data set. This is the second step of the process.

```
data = pd.read_csv("C:\\Users\\SUNDAR\\Desktop\\codes\\major project\\spam.csv",encoding
= 'latin1')

data['lab'] = data["Category"].replace({"ham":0,"spam":1})
data.head()
```

In the above step the dataset names spam.csv was loaded using the pandas package.



	Category	Message	lab
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

The above photo is the output. It shows the first 5 rows of the dataset as it was mentioned as **data.head().**

7.5.3 Converting categorical variables to numerical variables

```
X = data.Message
Y = data.Category
le = LabelEncoder()#converts categorical variables to numerical format
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)
```

The provided code snippet is a Python script for preprocessing data before training a machine learning model. It involves extracting features (messages) and target labels (categories) from a dataset. The `LabelEncoder` from scikit-learn is used to convert categorical category labels into numerical format. The target variable is reshaped to ensure it is a 2D array, a common requirement for machine learning models. This preparation is crucial for training models that expect numerical input and target variables.

7.5.4 Splitting The Data For Training And Testing

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.20)
```

The code snippet utilizes the `train_test_split` function from scikit-learn to divide a dataset into training and testing sets. The feature set (`X`) and target variable (`Y`) are split into training (`X_train`, `Y_train`) and testing (`X_test`, `Y_test`) sets, with 20% of the data allocated for testing. This separation is crucial for assessing the model's performance on unseen data during the machine learning process.

7.5.5 Tokenization

```
#spliting into words using tokenization
#tokens- words
max_words = 1000
max_len  = 150
tok = Tokenizer(num_words = max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
sequences_matrix  =  sequence.pad_sequences(sequences,  maxlen  =
   max_len)
```

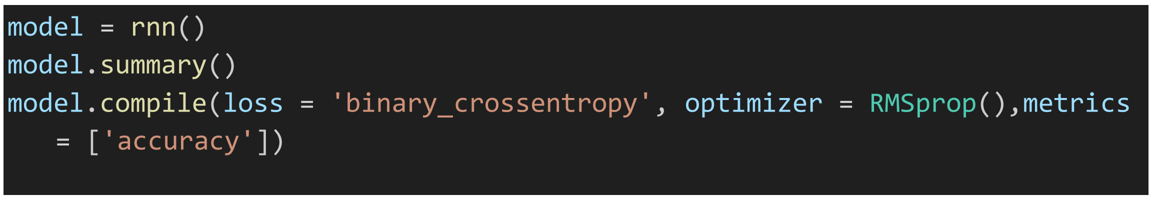
The above step preprocesses text data for neural networks using Keras. It limits the vocabulary to the top 1000 words, sets a maximum sequence length of 150, and utilizes a Tokenizer to convert text in the training set (`X_train`) into integer sequences. The Tokenizer's vocabulary is updated based on the training data, assigning unique integer indices to words. Finally, sequences are padded or truncated to a uniform length of 150. These steps are essential for inputting consistent and formatted text data into neural networks, facilitating effective natural language processing tasks with a focus on limited vocabulary and uniform sequence length.

7.5.6 Building A Model

```
#Embedding layer - used for neural networks on text data.It input data be integer
#each word is encoded as unique integer...it can be used to load a pre-trained word
embedding model- a type of transfer learning
def rnn():
    inputs = Input(name = 'inputs',shape = [max_len])
    layer = Embedding(max_words,50,input_length=max_len)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256,name = 'FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(1,name = 'out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs = inputs, outputs = layer)
    return model
```

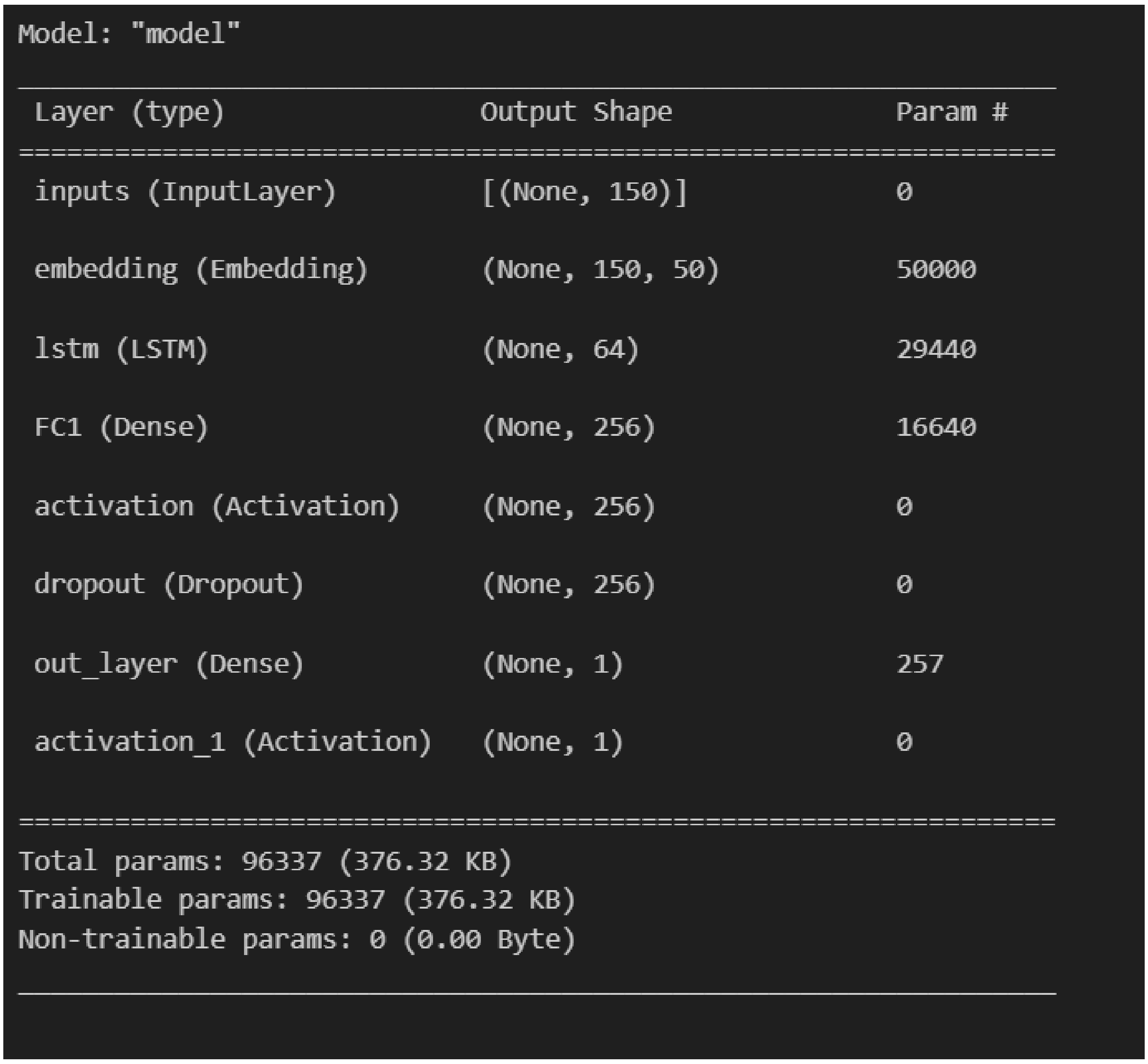
The above step defines a Recurrent Neural Network (RNN) model for binary classification using Keras. The model architecture consists of an input layer for sequences of integers, an embedding layer to convert input sequences into dense vectors, an LSTM layer with 64 units for capturing sequential dependencies, a dense layer with 256 units and ReLU activation, a dropout layer to prevent overfitting, and a final dense layer with a single unit and sigmoid activation for binary classification. The model is constructed using the Keras functional API, allowing for flexible and customizable architectures. Overall, this RNN aims to learn patterns in sequential data, making it suitable for tasks such as natural language processing or time-series analysis.

7.5.7 Model Architecture



```
model = rnn()
model.summary()
model.compile(loss = 'binary_crossentropy', optimizer = RMSprop(),metrics
   = ['accuracy'])
```

The output of the above code is:



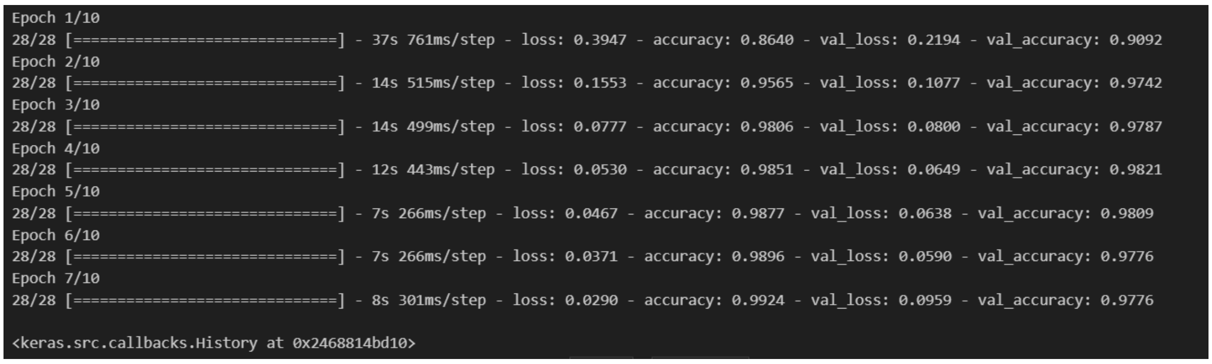
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inputs (InputLayer)         [(None, 150)]             0

 embedding (Embedding)       (None, 150, 50)           50000

 lstm (LSTM)                 (None, 64)                29440

 FC1 (Dense)                 (None, 256)               16640

 activation (Activation)     (None, 256)               0

 dropout (Dropout)           (None, 256)               0

 out_layer (Dense)           (None, 1)                 257

 activation_1 (Activation)   (None, 1)                 0

=================================================================
Total params: 96337 (376.32 KB)
Trainable params: 96337 (376.32 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

This step involves creating an instance of the RNN model using the rnn() function, displaying a summary of the model architecture, and then compiling the model for training.After this compilation step, the model is ready to be trained using training data and labels. You would typically use the fit method to train the model on your data, specifying the number of epochs, batch size, and validation data if needed.

7.5.8 Training the LSTM model

```
model.fit(sequences_matrix,Y_train,batch_size=128,epochs = 10,validation_split=0.2,
callbacks=[EarlyStopping(monitor ='val_loss',min_delta=0.0001)])
```

The output of the above snippet is:



```
Epoch 1/10
28/28 [==============================] - 37s 761ms/step - loss: 0.3947 - accuracy: 0.8640 - val_loss: 0.2194 - val_accuracy: 0.9092
Epoch 2/10
28/28 [==============================] - 14s 515ms/step - loss: 0.1553 - accuracy: 0.9565 - val_loss: 0.1077 - val_accuracy: 0.9742
Epoch 3/10
28/28 [==============================] - 14s 499ms/step - loss: 0.0777 - accuracy: 0.9806 - val_loss: 0.0800 - val_accuracy: 0.9787
Epoch 4/10
28/28 [==============================] - 12s 443ms/step - loss: 0.0530 - accuracy: 0.9851 - val_loss: 0.0649 - val_accuracy: 0.9821
Epoch 5/10
28/28 [==============================] - 7s 266ms/step - loss: 0.0467 - accuracy: 0.9877 - val_loss: 0.0638 - val_accuracy: 0.9809
Epoch 6/10
28/28 [==============================] - 7s 266ms/step - loss: 0.0371 - accuracy: 0.9896 - val_loss: 0.0590 - val_accuracy: 0.9776
Epoch 7/10
28/28 [==============================] - 8s 301ms/step - loss: 0.0290 - accuracy: 0.9924 - val_loss: 0.0959 - val_accuracy: 0.9776

<keras.src.callbacks.History at 0x2468814bd10>
```

The provided information outlines the training configuration for a Recurrent Neural Network (RNN) using the Keras library. The `sequences_matrix` represents preprocessed training data consisting of integer sequences derived from text, while `Y_train` contains corresponding binary labels for a binary classification task. During training, batches of 128 samples are processed in each iteration, with the entire dataset passed forward and backward through the neural network for 10 epochs. Additionally, 20% of the training data is reserved for validation to monitor model performance. The training process incorporates the EarlyStopping callback, which halts training if the validation loss fails to improve by at least 0.0001 over a specified number of epochs, thus mitigating the risk of overfitting. This comprehensive setup aims to optimize the RNN model's performance while preventing unnecessary training epochs.

7.5.9 Testing The Model

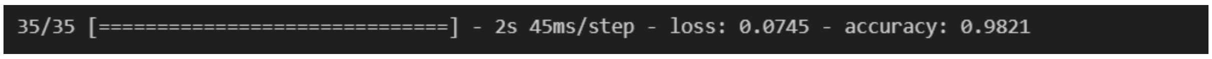
```
test_sequences = tok.texts_to_sequences(X_test)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen = max_len)
```

The provided code prepares the test data for evaluation by the trained Recurrent Neural Network (RNN). It utilizes the Tokenizer (`tok`) previously fitted on the training data to convert text sequences in `X_test` into sequences of integers. The resulting sequences are then padded or truncated to match the specified maximum sequence length (`max_len`). This ensures that the test data is formatted in the same way as the training data, allowing for consistent input into the RNN model. The processed test sequences are stored in the `test_sequences_matrix` variable, which can be used for evaluating the RNN's performance on the unseen test data.

7.5.10 Checking The Accuracy Of A Model

```
acc = model.evaluate(test_sequences_matrix,Y_test)
```

The Accuracy of the LSTM model is:

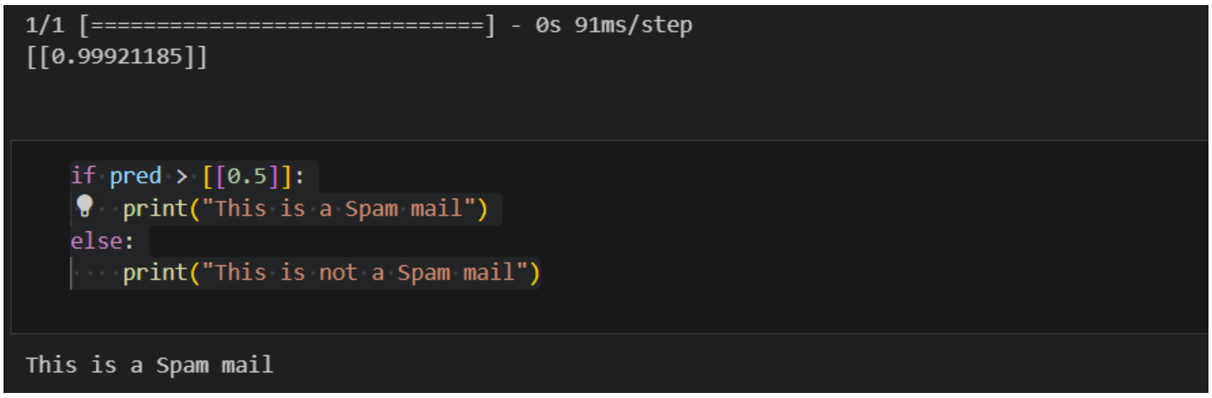


```
35/35 [==============================] - 2s 45ms/step - loss: 0.0745 - accuracy: 0.9821
```

7.5.11 Predicting The Given Email Is Spam Or Not

```
test_content = [input()]

textx = tok.texts_to_sequences(test_content)
textx = sequence.pad_sequences(textx,maxlen=max_len)
pred = model.predict(textx)
print(pred)
```

Here the input is given by the user. The input will be passed to the lstm model and predicts whether the user’s email is spam mail or ham mail.



```
1/1 [==============================] - 0s 91ms/step
[[0.99921185]]
```

```
if pred > [[0.5]]:
    print("This is a Spam mail")
else:
    print("This is not a Spam mail")
```

```
This is a Spam mail
```

Based on the output it is clear that the user’s mail is **spam mail.**

7.5.12 Building a Flask Application

Flask Application:

```
from flask import Flask, render_template, request
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping

app = Flask(__name__)

# Load the data and preprocess
data = pd.read_csv("C:\\Users\\SUNDAR\\Desktop\\codes\\majorproject\\spam.csv",
encoding='latin1')
data['lab'] = data["Category"].replace({"ham": 0, "spam": 1})
X = data.Message
Y = data.Category
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1, 1)
X_train, _, Y_train, _ = train_test_split(X, Y, test_size=0.20)
# Tokenization and model setup
max_words = 1000
max_len = 150
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences_matrix =
sequence.pad_sequences(tok.texts_to_sequences(X_train),maxlen=max_len)

def rnn():
    inputs = Input(name='inputs', shape=[max_len])
    layer = Embedding(max_words, 50, input_length=max_len)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256, name='FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(1, name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs, outputs=layer)
    model.compile(loss='binary_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])
    return model

model = rnn()
model.fit(sequences_matrix, Y_train, batch_size=128, epochs=10, validation_split=0.2,
          callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)])
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
```

```
def predict():
    if request.method == 'POST':
        message = request.form['message']
        textx = tok.texts_to_sequences([message])
        textx = sequence.pad_sequences(textx, maxlen=max_len)
        pred = model.predict(textx)
        result = "This is a Spam mail" if pred > 0.5 else "This is not a Spam mail"
        return render_template('index.html', result=result, message=message)

if __name__ == '__main__':
    app.run(debug=True)
```

The above code is Flask code i.e **app.py**. This flask application is connected to a frontend website i.e., **index.html**

HTML Page(User Interface):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>batch-6</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background-color: #f2f2f2;
            background-image:
url("https://as2.ftcdn.net/v2/jpg/05/36/59/41/1000_F_536594133_tME8MiBDJ5nZF4KU6vmUMaSr6
CTubfnQ.jpg");

            color: #333;
            margin: 0;
            padding: 0;
        }

        h1 {
            text-align: center;
            color: hsl(215, 19%, 88%);
        }

        .form-section {
            max-width: 400px;
            margin: 20px auto;
            padding-top: 10px;
            padding-bottom: 20px;
            padding-right: 90px;
            padding-left:90px;
            height: 550px;
            background-color: rgb(211, 233, 241);
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            text-align: center;
            border: 2px solid black;
            box-shadow: 3px 6px 40px #000;

        }

        label {
            display: block;
            margin-bottom: 8px;
            color: black;
        }


        input {
            width: 100%;
            padding: 10px;
            margin-bottom: 15px;
            text-align: center;
            border: 1px solid black;
            box-sizing: border-box;
            background-color: rgb(236, 236, 238);
            box-shadow:3px 6px 40px rgb(61, 62, 62);
            border: 3px solid transparent;
            border-image: linear-gradient(to bottom right, #046576 0%, #2c90fc 25%,
#a432e6 50%,  #154f8d 75%, #510dd8 100%);
            border-image-slice: 1;
            height: 50px;
            width: 400px;

        }

        button {
            top:50%;
           background-color:#0a0a23;
           color: #fff;
           border:none;
           border-radius:10px;
           box-shadow: 0px 0px 2px 2px rgb(0,0,0);
           width: 90px;
           height: 30px;
           margin-top: 10px;
           margin-bottom: 10px;
        }

        button:hover {
            background-color:rgb(166, 224, 244);
            transform: scale(1.1);
            transition: 0.1s;
            color: black;
            /* box-shadow:3    px 6px 40px rgb(248, 250, 251);   */
        }

        p {
            margin-top: 20px;
            color: #555;
        }
        .con{
            padding-top: 13px;
            color: black;
        }

        .form-section img{
            padding-top: 20px;
            padding-bottom: 20px;
            height: 30vh;
            width : 50vh;
            margin-top:1px;

        }
         .img1:hover{
         transform: scale(1.1);
         transition: 0.1s;

       }
    </style>
</head>
<body>
    <h1>Email Spam Detection</h1>
    <h1> IV-II Major Project (2024)</h1>

    <div class="form-section">

        <form method="post" action="{{ url_for('predict') }}">
            <img src="https://editor.analyticsvidhya.com/uploads/51577istockphoto-
875618304-612x612.jpg" alt="Image Description" class="img1"/>
            <label for="message">Enter your message:</label>

            <input type="text" id="message" placeholder="Enter your message...📨👈"
name="message" required>
            <button type="submit">Predict</button>
        </form>
        <div class="con"></div>
        {% if result %}
            <p>Input Message: {{ message }}</p>
            <p>Prediction: {{ result }}</p>
        {% endif %}
    </div>
</body>
</html>
```

By using the flask application the user will be able to interact easily and can enter his email in HTML page. After Entering the email the user will get the prediction whether the entered email is spam email or ham email. Then the user will get to know that he/she received email is spam mail or not.

8. TESTING

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects). Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors.

8.1 System Testing:

System testing for email spam detection using LSTM (Long Short-Term Memory) involves evaluating the overall performance, reliability, and functionality of the system. Here is a comprehensive system testing plan for email spam detection using LSTM:

- **Integration Testing:** Ensure seamless integration with other components, such as data input modules and result output interfaces. Verify that the system can handle real-time data feeds.

- **Functional Testing:** Test the core functionalities of the system, including- Email classification (spam or non-spam). Model adaptation to evolving spam patterns. Handling of false positives and false negatives.

- **Performance Testing:** Evaluate the system's performance under various loads and conditions. Measure response times for both training and inference phases.Assess resource utilization (CPU, memory) during peak loads.

- **User Acceptance Testing (UAT):** Involve end-users in the testing process to gather feedback on the system's performance and usability.

8.2Module Testing:

Module testing, also known as unit testing, involves testing individual components or modules of a system in isolation to ensure they function as intended. In the context of email spam detection using LSTM, various modules contribute to the overall functionality of the system. Below is a module testing plan for key components:

- **Inference Module:**

  Objective: Ensure that the module accurately classifies emails as spam or non-spam during the inference phase.

  Test Cases:

  - Test the module with known spam and non-spam emails.
  - Evaluate the module's performance on a diverse set of test cases.
  - Validate the handling of edge cases, such as emails with unusual content.
  - Confirm that the module produces the expected output format.

- **Integration with External Systems Module:**

  Objective: Confirm seamless integration with external systems, such as email clients or databases.

  Test Cases:

  - Validate the compatibility with different email clients and platforms.
  - Test the module's ability to handle real-time data feeds.
  - Confirm that integration does not compromise the overall system performance.

- **Model Updating Module:**

  Objective: Ensure that the system can update the LSTM model to adapt to evolving spam patterns.

  Test Cases:

  - Simulate the introduction of new labeled data for model updating.
  - Verify that the updated model maintains or improves performance.
  - Test the robustness of the system during the model updating process.

9. RESULTS

Upon accessing the deployed web application, users are greeted with a user-friendly interface designed to detect the entered email is spam mail or not.

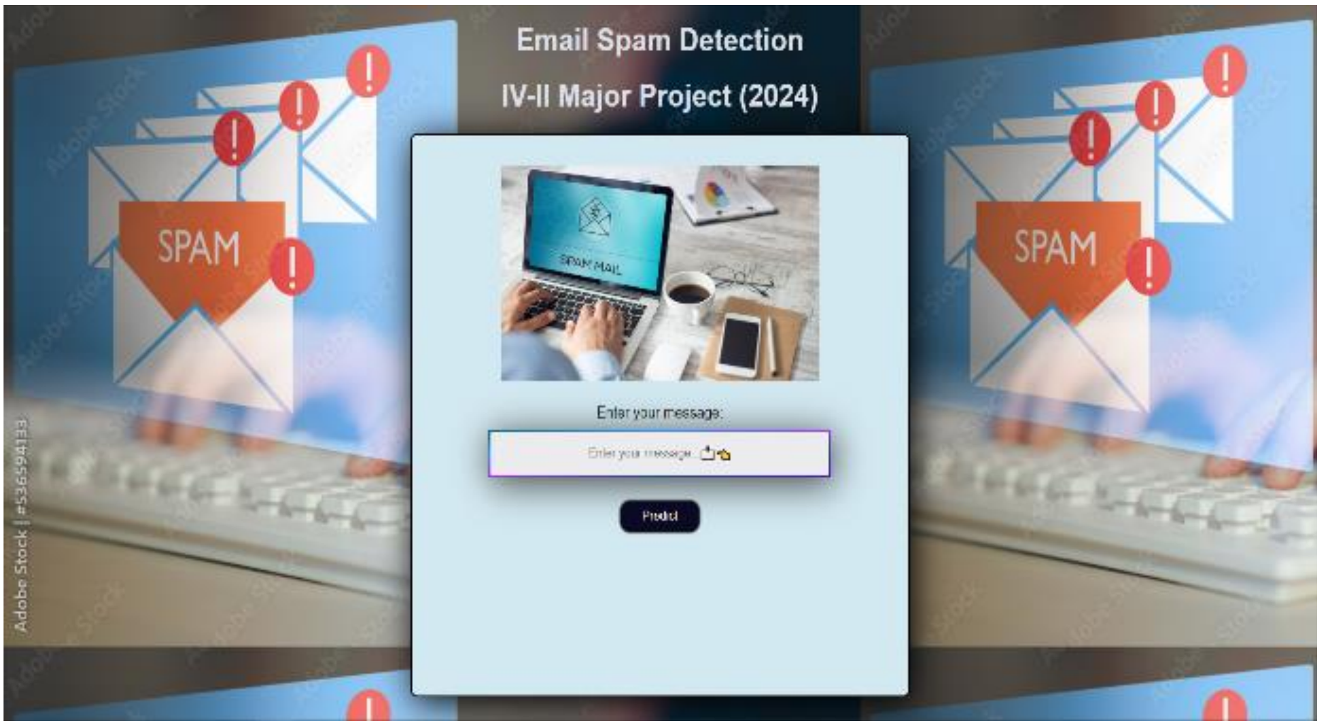


Figure-9.1 Home page

Users can enter their email messages in the box of the web application.

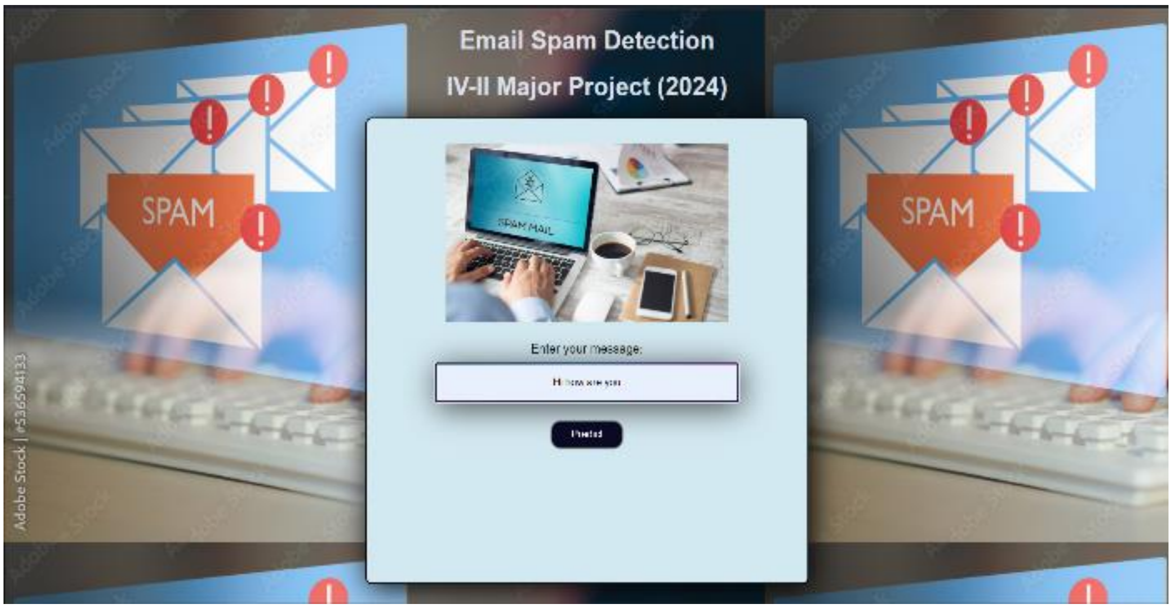
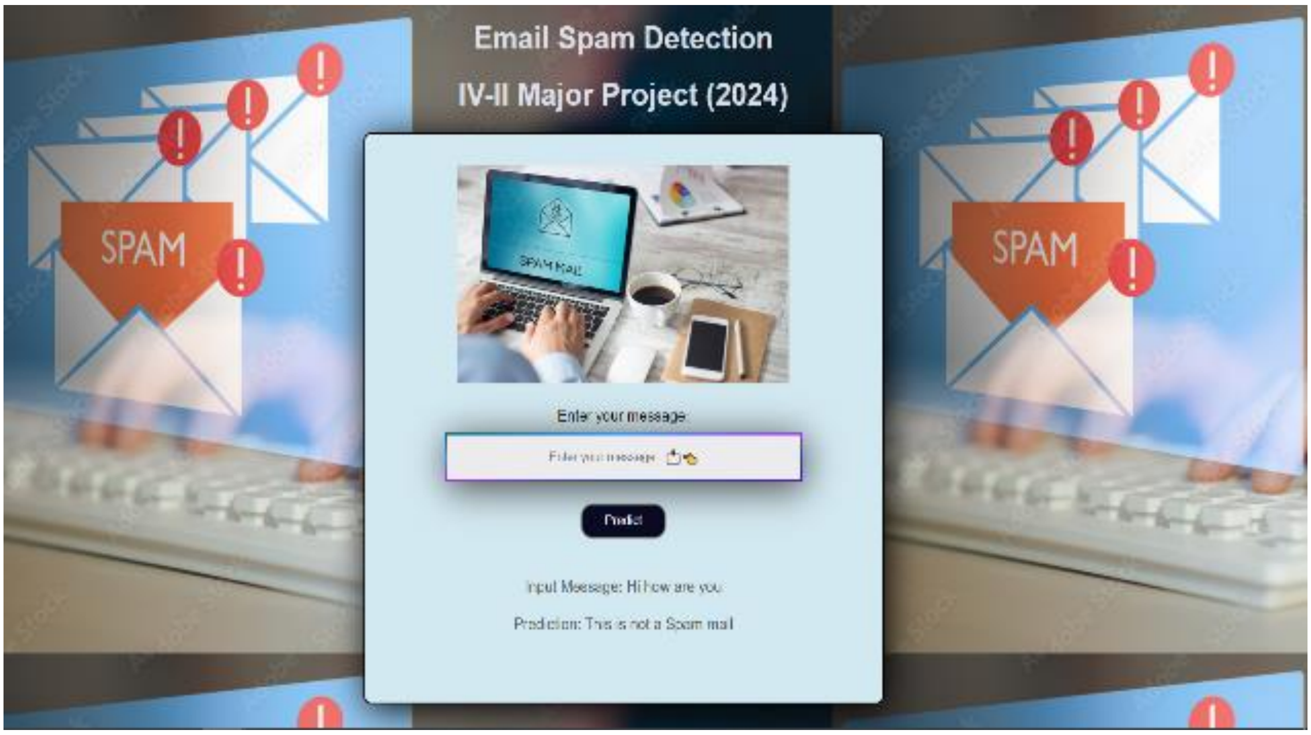


Figure-9.2 Entering the text(Body of an email)

After submission, the model uses the trained LSTM network to process the input, and the webpage dynamically displays the prediction.



Input Message: Hi how are you

Prediction: This is not a Spam mail

Fig-3: Predicted Result

The Flask web application incorporates the predictions of the model, enabling users to interact with it without requiring extensive technical knowledge. The result is a user-centric web environment where machine learning—more precisely, LSTM—is applied practically for email spam detection.

10.CONCLUSION

The implementation of email spam detection using Long Short-Term Memory (LSTM) networks, as demonstrated in the provided code, represents a significant advancement in the realm of email security. Leveraging the power of deep learning, specifically LSTM architecture, enables the model to capture intricate sequential patterns within the text data, thereby enhancing the accuracy of spam classification. The utilization of Flask for the web application interface ensures a user-friendly and accessible platform for individuals seeking reliable spam detection.

In the process of developing and training the LSTM model, the code showcases a comprehensive approach to data preprocessing, model construction, and training. The incorporation of early stopping mechanisms in the training phase is crucial in preventing overfitting, contributing to the model's generalization capability. The system's real-time predictions through the Flask web application illustrate the seamless integration of the trained LSTM model into an interactive environment, enabling users to assess the spam likelihood of their messages on-the-fly.

Furthermore, the modular structure of the code facilitates ease of testing, maintenance, and potential future enhancements. The division of the system into distinct modules, such as data loading, LSTM model architecture, and user interaction, allows for systematic testing and iterative improvements. The system's adaptability to user feedback positions it as a dynamic solution, capable of continuous refinement based on real-world interactions. The combination of deep learning techniques and web application development showcased in this code provides a robust foundation for an effective and scalable email spam detection system.

As we conclude, the code not only represents a functional email spam detection solution but also serves as a valuable template for future research and development in the domain of email security. The successful integration of LSTM networks and Flask demonstrates the potential for creating sophisticated yet accessible tools that contribute to a safer online communication environment. This code lays the groundwork for further exploration, optimization, and potential integration into broader email security frameworks.

11.FUTURE SCOPE

Further development of the spam detection model is where the offered code will be most useful. This is investigating cutting edge methods in machine learning and natural language processing (NLP) to enhance the predictive power of the model. Intricate patterns and semantic subtleties in text data may be captured by the system by experimentation with sophisticated neural network topologies, such as transformer-based models like BERT or GPT. Accurate and reliable spam detection will be enhanced by ongoing optimization through hyper parameter adjustments and fine-tuning on a variety of datasets.

Future directions worth exploring include extending the application's functionality to support several languages and customizing it for various cultural contexts. Training the model on datasets reflecting linguistic variances is necessary to incorporate multilingual support. It will also make the model more flexible to take into account cultural variations in spam patterns and communication approaches. Due to the global diversity of languages and communication conventions, this comprehensive approach guarantees that the spam detection system becomes more inclusive and successful.

Opportunities for the code's integration with new trends arise from the advancement of technology. Scalability and accessibility can be improved by cloud deployment on systems like AWS, Google Cloud, or Azure. Additionally, investigating partnerships with email service providers to smoothly incorporate the spam detection technology into current email platforms can have a big effect on user experience. Furthermore, thinking about how the program may be integrated into mobile platforms—for example, by creating a mobile application—would increase the system's reach and give consumers easy-to-use spam detection capabilities while they're on the road. Retaining user confidence in the rapidly changing digital communication ecosystem will require adopting privacy-focused technologies and upholding strict security protocols.

12.REFERENCES

1. Thashina Sultana, K A Sapnaz, Fathima Sana, Mrs. Jamedar Najath, Dept. of Computer Science and Engineering
Yenepoya Institute of Technology
Moodbidri, India.

2. O. Saad, A. Darwish and R. Faraj, "A survey of machine learning techniques for Spam filtering", *Int. J. Comput. Sci. Netw. Secur.*, vol. 12, pp. 66, Feb. 2012.

3. M. K. Paswan, P. S. Bala and G. Aghila, "Spam filtering: Comparative analysis of filtering techniques", *Proc. Int. Conf. Adv. Eng. Sci. Manage. (ICAESM)*, pp. 170-176, Mar. 2012.

4. R. Islam and Y. Xiang, "Email classification using data reduction method", *Proc. 5th Int. ICST Conf. Commun. Netw. China*, pp. 1-5, Aug. 2010

5. N. Banu and M. Banu, "A Comprehensive Study of Phishing Attacks", *Int. J. Comput. Sci. nf. Technol.*, vol. 4, no. 6, pp. 783-786, 2013.

6. J. Dean, “Large scale deep learning,” in Proceedings of the Keynote GPU Technical Conference, San Jose, CA, USA, 2015.
7. J. K. Kruschke and T. M. Liddell, “Bayesian data analysis for newcomers,” Psychonomic Bulletin & Review, vol. 25, no. 1, pp. 155–177, 2018.
8. K. S. Adewole, N. B. Anuar, A. Kamsin, K. D. Varathan, and S. A. Razak, “Malicious accounts: dark of the social networks,” Journal of Network and Computer Applications, vol. 79, pp. 41–67, 2017
9. N. Kumar and S. Sonowal, “Email spam detection using machine learning algorithms,” in Proceedings of the 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), pp. 108–113, Coimbatore, India, 2020
10. A. J. Saleh, A. Karim, B. Shanmugam et al., “An intelligent spam detection model based on artificial immune system,” Information, vol. 10, no. 6, p. 209, 2019.
11. E. Blanzieri and A. Bryl, E-mail Spam Filtering with Local SVM Classifiers, University of Trento, Trento, Italy, 2008