ABSTRACT

The rise in technological advancements and Social Networking Sites (SNS) made people more engaged in their virtual lives.Research has revealed that people feel more comfortable posting their feelings, including suicidal thoughts, on SNS than discussing them through face-to-face settings due to the social stigma associated with mental health .Many suicidal deaths can be prevented by understanding how people communicate their distress related thoughts. Early understanding of the risk factors and warning signs can decrease the threshold for suicide and help prevent many deaths. This Project aims to develop a system that can detect suicidal ideation on Twitter tweets using DL models. In this project we trained several Deep learning models such as LSTM(Long Short -Term Memory), GRU(Gated Recurrent unit),andCNN(ConvolutionalNeuralNetwork).

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION

The rise of the social media and the online communities are used to share thoughts about the mental health and express the feelings of suffering, to prevent their suicide it is necessary to detect suicide related post. Twitter is one of the most popular social media sites that help users to share their thoughts and feelings in a real-time .Suicide is one of the significant public health concerns consuming a lot of lives. According to the statistic of the World Health Organization (WHO), around one million people die due to suicide each year, and on average, suicide occurs every 40 seconds. Suicide might be considered as one of the most serious social health problems in the modern society. Many factors can lead to suicide for example personal issues, such as hopelessness, depression, tired, loneliness.

They make statements such as "I want to kill myself", "I hate my life", "I have lived long enough "or “I’m so tired”. It is a common saying that suicide is a permanent solution for dealing with temporary problems. Despite the growing numbers of suicidal cases, it can be prevented to some extent by understanding the risk factors related to suicidal behavior in the early stages of the suicidal process. Our work extracts a number of relevant features to differentiate between suicidal and non-suicidal tweets by the help of a novel feature engineering technique. This paper builds a novel dataset by extracting the tweets related to suicide and non-suicide from Twitter. In the digital age, social media platforms have become a significant medium for self-expression, connecting people, and sharing thoughts and emotions.

The process of suicide starts with suicidal thoughts or ideation. Because mental illness may be diagnosed and treated ,the early identification of warning signs or risk factors may be the most effective way of preventing suicide It then matures to suicidal attempt and finally to the completed suicide. Suicide is the second leading cause of death among people aged between 10 and 34 years. Prevention can be done by reducing the risk factors or by reducing the obstacles to mental health resource. The best way to prevent their suicide is to catch these signals and predict other hidden signals behind their posting content in order to react

to them and take appropriate actions.

In order to save the lives of the people, we need to study the behavior and recent communications performed by them. Finally Detecting and addressing suicidal ideation on social media platforms like Twitter is a crucial application of Machine Learning (ML) and Deep Learning (DL) models. Implementation of DL models for suicidal ideation detection could have profound implications for suicide prevention, offering a lifeline to those in crisis and advancing the integration of technology in mental health support.

1.2 NEED OF THE PROJECT

The advancement of technology and the prevalence of Social Networking Sites (SNS) have led people to be more involved in their online lives. Studies show that individuals find it easier to express their emotions, including suicidal thoughts, on SNS rather than in face-to-face conversations due to the associated social stigma around mental health. Recognizing these distress signals early is crucial in preventing suicides. This project aims to create a system using deep learning models to identify suicidal ideation in Twitter posts, thereby aiding in suicide prevention efforts.

The core objective of this project is to harness the power of deep learning models to address this pressing issue. By developing a sophisticated system, the aim is to detect suicidal ideation embedded within Twitter tweets. By leveraging these advanced algorithms, the project seeks to provide an effective means of early detection, offering a glimmer of hope in the face of a deeply concerning societal challenge. Through this innovative approach, the project endeavors to contribute significantly to suicide prevention efforts, potentially reshaping the way we understand and respond to mental health crises in the digital age.

CHAPTER II

SYSTEM ANALYSIS

2.1 REQUIREMENT ANALYSIS

2.1.1 Non-Functional Requirements

Performance Requirements:

- The system should process and analyze incoming Twitter data in real-time or near-real-time to detect and respond to potential suicidal ideation promptly.
- The model's accuracy in identifying potential suicidal ideation should be consistently high, with minimal false positives and false negatives to avoid unnecessary alerts or missed cases.
- The system should be capable of processing a high volume of tweets per second, ensuring comprehensive coverage and efficient analysis
- Security and data privacy measures to protect sensitive information.

Software Quality Attributes:

Accuracy

The primary goal is to accurately detect suicidal ideation. The software should minimize false positives and false negatives to avoid unnecessary distress or missed cases.

Sensitivity and Specificity:

Sensitivity (true positive rate) measures the ability to correctly identify cases of suicidal ideation, while specificity (true negative rate) measures the ability to correctly exclude non-suicidal cases.

Data Quality:

High-quality training data is essential for model performance. Ensure that data is clean, well-labeled, and representative of the target population.

2.2 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by the software engineers as the starting point for the system design.

In hardware requirement we require all those components which will provide the platform for the development of project. The minimum hardware required for the development of this project is as follows –

- LAPTOP
- HARD DISK : 500 GB - 1 TB
- RAM : 4GB
- PROCESSOR : Intel core i5

2.3 SOFTWARE REQUIREMENTS

The software requirements are the software specifications of the system. It should include both a definition and specification of requirements. It is a set of what the system should do rather than how it should do it. It is useful in estimating cost, planning team activities, performing tasks and tracking the team's progress throughout the development activity.

- OPERATING SYSTEM : WINDOWS 7 AND ABOVE
- INTEGRATED DEVELOPMENT ENVIRONMENT: VISUAL STUDIO.
- TOOLS : GOOGLE COLAB

2.4 EXISTING SYSTEM

In the Existing system various machine learning algorithms and ensemble approaches like Decision trees, Naive Bayes, Multinomial Naive Bayes, Support vector machine (SMO), Regression, Bagging, Random Forest, AdaBoost, voting and Stacking are implemented for classifying the suicide-related tweets into two groups using the real tweets.

After studying the literature, it was found that dataset used in the research was very low as data is not freely available due to its ethical considerations and the studies which were already done in this area of research have not achieved much accuracy and recall.

2.5 PROPOSED SYSTEM

This project aims to develop a system that can detect suicidal ideation on Twitter using Natural Language Processing (NLP) and Deep learning models include Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), Convolutional Neural Networks(CNN).We have used the Twitter Suicide Dataset, which

contains Tweets from individuals who have experienced suicidal thoughts and ideations. The dataset includes more than 1787 Tweets and comments, and it has been labeled as either indicating suicidal ideation or not.

2.6 MODULES

In this project, there are three modules –

1. Data Collection and Preprocessing:
2. Feature Extraction
3. Model Training

Data Collection and Preprocessing Module:

- Data Collection: Gather text or other relevant data from various sources, such as social media Like Twitter
- Data Cleaning: Remove noise, irrelevant information, or personally identifiable data.
- Tokenization: Split text data into individual words or tokens.
- Text Normalization: Convert text to a consistent format (e.g., lowercase).
- Data Splitting: Divide the data into training, validation, and testing sets.

Feature Extraction Module:

Feature extraction helps the models understand and learn meaningful patterns and information from the data.

It is is responsible for transforming the raw input data (typically text or other relevant information) into numerical representations that can be used as input to deep learning models like LSTM, GRU, or CNN.

- Text Cleaning: Lowercasing and handling missing values.
- Tokenization: Splitting the text into words or tokens.
- Stop Words Removal: Removing common English stop words.
- Filtering Words Based on Length: Removing very short words.
- Lemmatization: Reducing words to their base form using the WordNet lemmatizer.

Model Training

Train the deep learning models using the training data and optimize them for the suicidal ideation detection task. Consider using techniques like early stopping and regularization.

CHAPTER III
SYSTEM DESIGN

3.1 SYSTEM ARCHITECTURE

A System Architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

A System Architecture can consist of system components and the sub-systems developed, that will work together to implement the overall system.

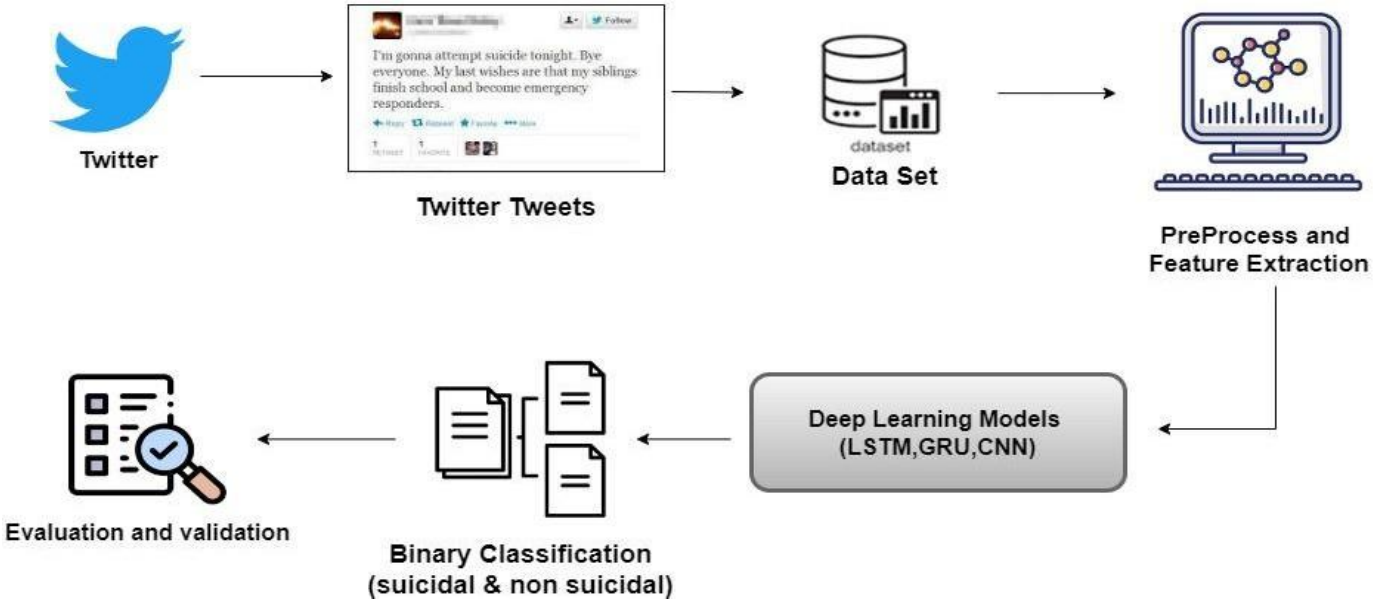


Fig 3.1 System Architecture

3.2 UML DIAGRAMS

UML is an acronym that stands for **Unified Modeling Language**. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques.

It is based on diagrammatic representations of software components. As the old proverb says:

“a picture is worth a thousand words”. By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

Mainly, UML has been used as a general-purpose modeling language in the field of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficacy.

Any complex system is best understood by making some kind of diagrams or pictures. These diagrams have a better impact on our understanding. There are two broad categories of diagrams and they are again divided into subcategories −

- Structural Diagrams
- Behavioral Diagrams

Structural Diagrams

The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable.

These static parts are represented by classes, interfaces, objects, components, and nodes. The four structural diagrams are −

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

Behavioral Diagrams

Any system can have two aspects, static and dynamic. So, a model is considered as complete when both the aspects are fully covered. Behavioral diagrams basically capture the dynamic aspect of a system. UML has the following five types of behavioral diagrams

- Use case diagram
- Sequence diagram

- Collaboration diagram
- Statechart diagram
- Activity diagram

3.2.1 USE CASE DIAGRAM

A use case diagram at the simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

While a use case itself might drill into a lot of detail about every possibility, a use-case diagram can help provide a high-level view of the system. It has been said before that "Use case diagrams are the blueprints for your system". They provide a simplified and graphical representation of what the system must actually do.

Due to their simplistic nature, use case diagrams can be a good communication tool for stakeholders. The drawings attempt to mimic the real world and provide a view for the stakeholder to understand how the system is going to be designed.

The purpose of the use case diagram is simply to provide the high-level view of the system and convey the requirements in laypeople's terms for the stakeholders. Additional diagrams and documentation can be used to provide a complete functional and technical view of the system.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified.

When the initial task is complete, use case diagrams are modelled to present the outside view. In brief, the purpose of use case diagrams can be said to be as follows –

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Shows the interactions among the requirements are actors.

Use-case diagram :

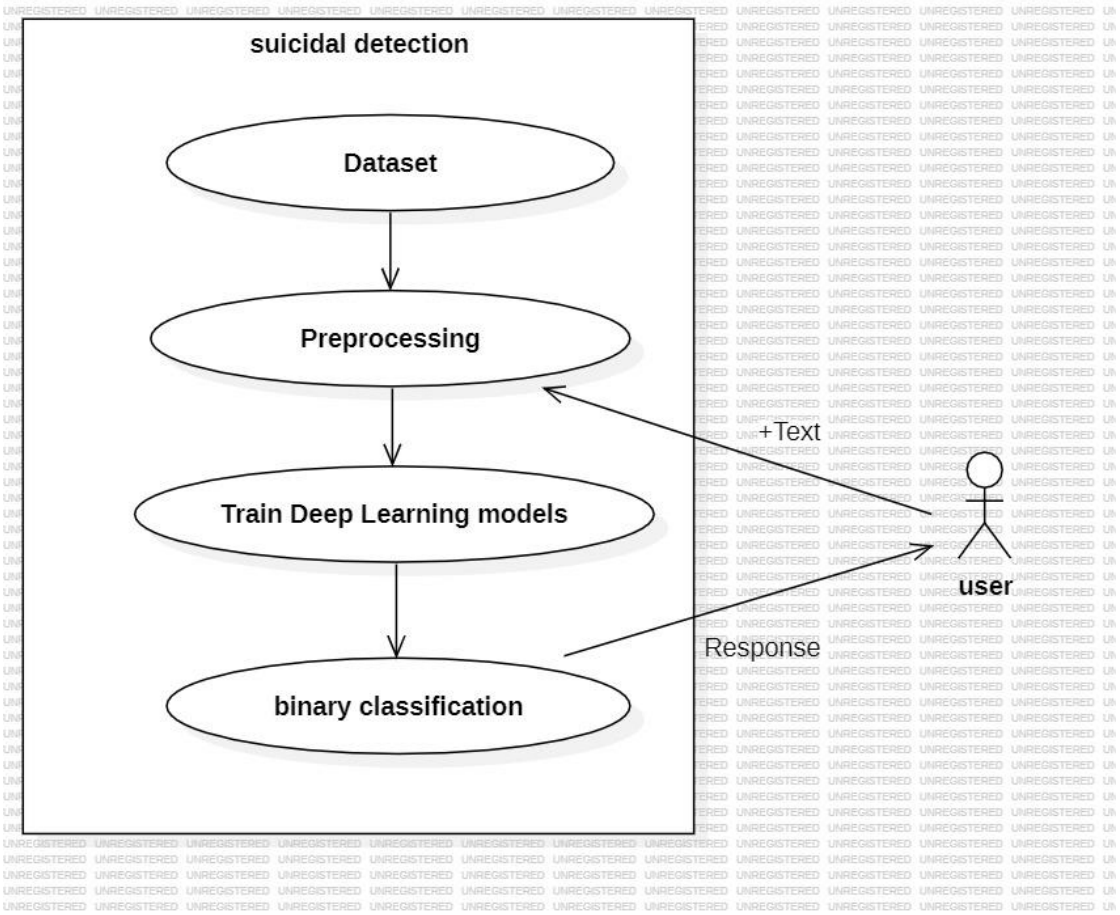


Fig 3.2 Use-case diagram

3.2.2 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.If the lifeline is that of an object, it demonstrates a role. Leaving the instance name blank can represent anonymous and unnamed instances.

Messages, written with horizontal arrows with the message name written above them, display interaction. Solid arrowheads represent synchronous calls, open arrowheads represent asynchronous messages, and dashed lines represent reply messages. If a caller sends a synchronous message, it must wait until the message is done, such as invoking a subroutine. If a caller sends an asynchronous message, it can continue processing and doesn't have to wait for a response.

Asynchronous calls are present in multithreaded applications, event-driven applications and in message-oriented middleware. Activation boxes, or method-call boxes, are opaque rectangles drawn on top of lifelines to represent that processes are being performed in response to the message (Execution Specifications in UML).
Objects calling methods on themselves use messages and add new activation boxes on top of any others to indicate a further level of processing. If an object is destroyed (removed from memory), an X is drawn on the bottom of the lifeline, and the dashed line ceases to be drawn below it. It should be the result of a message, either from the object itself, or another.

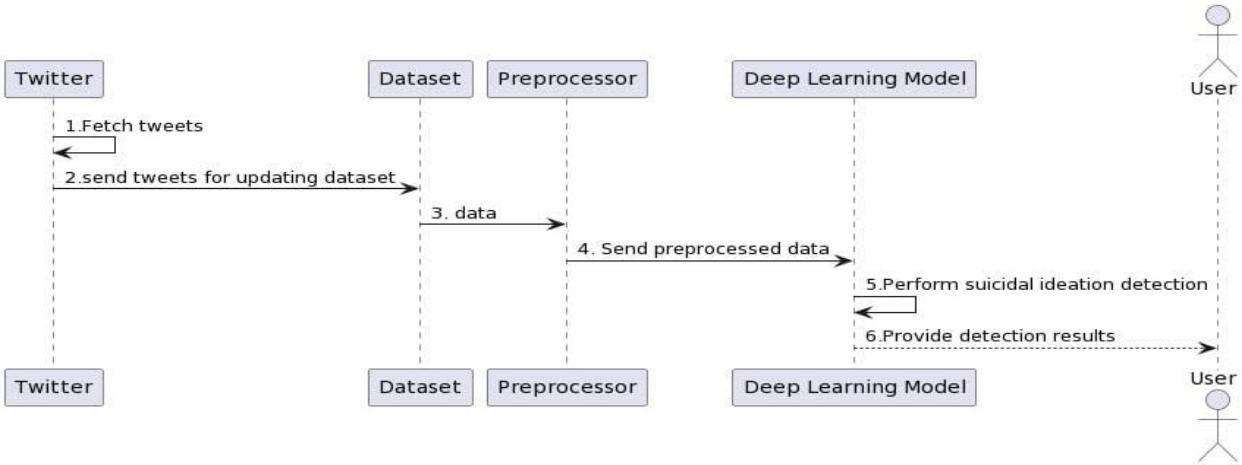


Figure:SequenceDiagram

3.2.3 ACTIVITY DIAGRAM

Activity Diagram is a behavioral diagram that represents the flow of activities in a system or a business process. Activity diagrams are commonly used to model workflows, business processes, software 10 algorithms, and other dynamic aspects of a system. They provide a visual representation of the sequential and parallel activities that make up a complex process. Here's an overview of key elements and concepts related to Activity Diagrams:

ELEMENTS OF AN ACTIVITY DIAGRAM:

- Activity: An activity is a specific task or operation that occurs within a process. Activities are represented by rounded rectangles and are connected by arrows to show the flow of control.

- Action: Actions represent individual steps or computations within an activity. They are typically depicted as rectangles with rounded corners. Actions can be simple, like "print receipt," or complex, involving multiple sub-activities

- Control Flow: Control flow arrows indicate the sequence in which activities are executed. Arrows Connect activities, showing the order of execution. Arrows may have a guard condition that determines under what circumstances they are followed.

- Fork and Join Nodes: Fork nodes (represented by a solid black bar) split a single flow of control into multiple concurrent flows, allowing activities to be executed in parallel. Join nodes (represented by a dashed line) merge multiple flows back into a single control flow.

- Decision Nodes: Decision nodes (diamond shapes) are used to represent points in the process where a decision is made. The outgoing arrows from a decision node have guard conditions that determine which path to follow based on a specific condition

- Initial Node and Final Node: An initial node (a solid black circle) indicates the starting point of the activity diagram. A final node (a solid circle with a dot inside) represents the end of the process or activity.

- Swimlanes: Swimlanes are used to organize activities into different partitions or responsibilities. Each swimlane typically represents a role, department, or system component responsible for certain activities

ACTIVITY DAIAGRAM:

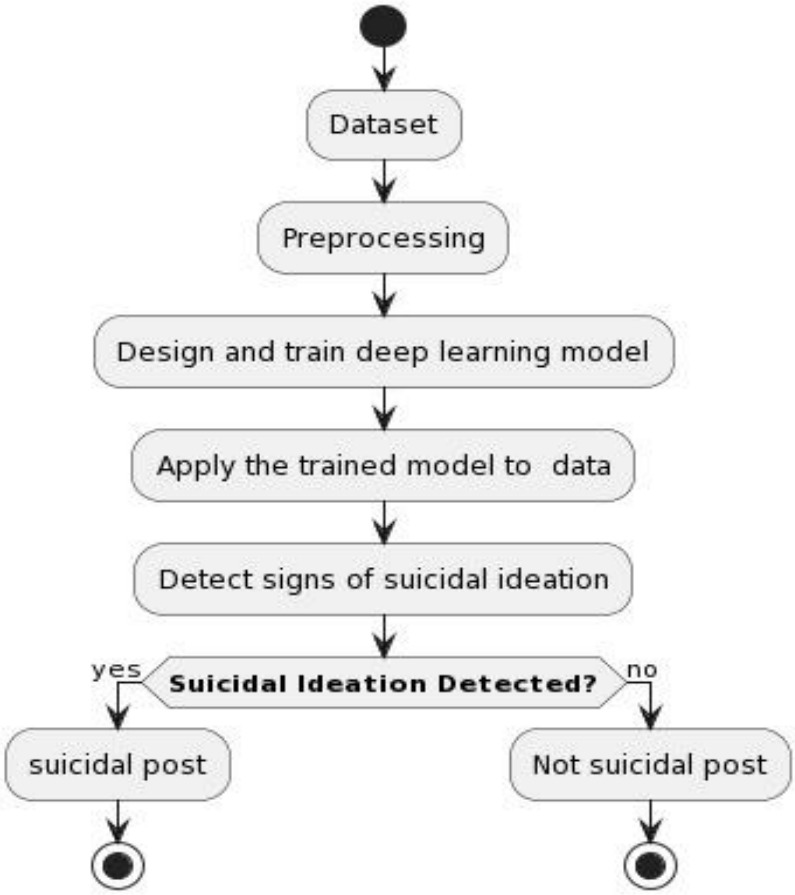


FIGURE: ACTIVITY DIAGRAM

CHAPTER IV
SYSTEM IMPLEMENTATION

4.1 TECHNOLOGIES USED:

4.1.1 PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object- oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it. With Python 2's end-of-life, only Python 3.5.x and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A

non- profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)) Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle- detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the **off-side rule**, which some other languages share, but in most languages, indentation doesn't have any semantic meaning.

4.1.2 GOOGLE COLAB NOTEBOOK

- Google Colab, short for Google Colaboratory, is a free cloud-based platform provided by Google that is popular among developers and researchers for running machine learning and deep learning projects. Google Colab is a powerful and user-friendly platform that simplifies the process of working on data science and machine learning projects, as it provides the necessary tools and resources in a convenient, cloud-based environment.

- Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires

no setup to use, while providing access free of charge to computing resources including GPUs.

- In a Colab notebook, you can write and run Python code in cells. Each cell can contain code or text (Markdown). One of the significant advantages of Google Colab is that it provides free access to GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units). This is extremely useful for accelerating deep learning model training, especially for tasks that require substantial computational power.

- Google Colab comes with pre-installed libraries like NumPy, Pandas, Matplotlib, TensorFlow, and PyTorch, making it well-suited for data analysis, machine learning, and deep learning tasks.

Fig 4.1 Google colab Notebook

- Google Colab runs entirely in the cloud, which means you don't need to install any software on your local machine. To use Google Colab, you need a Google account. You can sign in with your Google credentials, and your Colab notebooks are automatically saved to Google Drive, making it easy to access and share your work.

- Google Colab primarily supports the Python programming language. It provides a Python environment through Jupyter notebooks for data science, machine learning,

and deep learning projects. Users can write and execute Python code in Colab notebooks. you can use a mix of Python and shell commands to perform various tasks within Google Colab.

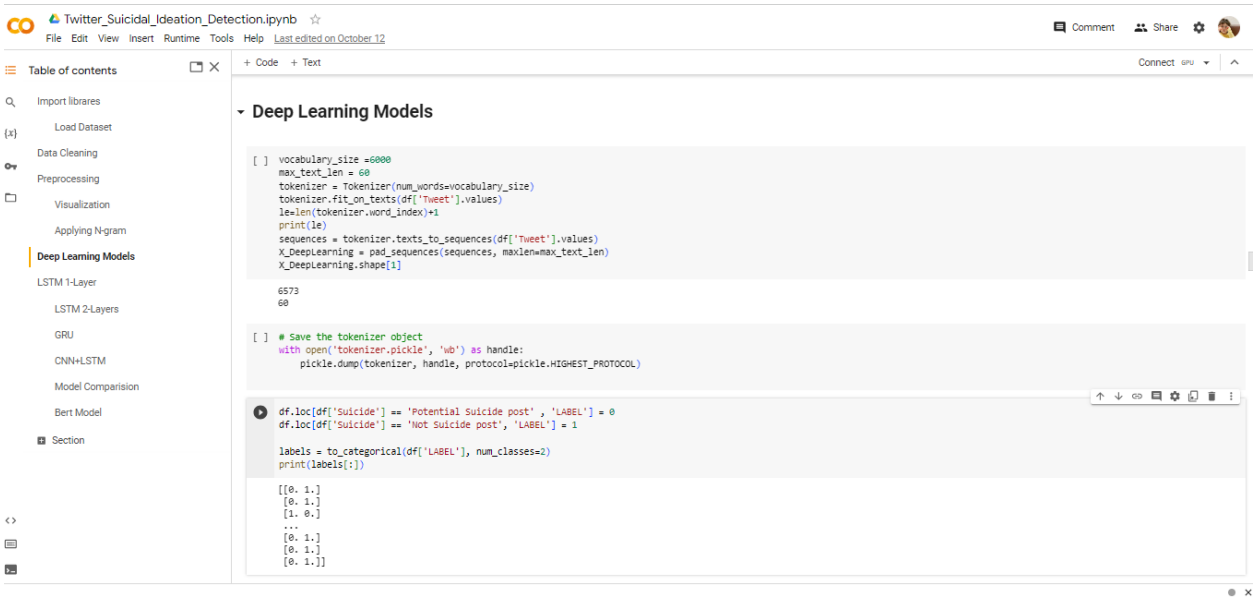


Fig 4.2 Google Colab Notebook Interface

We can easily share your Colab notebooks with collaborators or the public by providing a shareable link. Multiple users can work on the same notebook simultaneously, making it an excellent platform for group projects.

4.1.3 STREAMLIT

Streamlit is an open-source Python library designed to help developers and data scientists create interactive web applications quickly and easily. With its simplicity and Pythonic approach, Streamlit has gained popularity for a wide range of applications in data science, machine learning, and data visualization projects.

- Streamlit is particularly useful for creating interactive data visualizations. Data scientists and analysts can use it to build dashboards, charts, and graphs to convey data insights effectively.It is often used for prototyping and showcasing machine learning models. It allows developers to create user-friendly interfaces for model demos, making it easier to share and explain machine learning concepts and results.

- For teams developing data products, Streamlit is a valuable tool for rapidly

prototyping and testing user interfaces. It allows developers to iterate quickly on the front-end of their products.

- We can easily share your Streamlit applications by deploying them on various platforms, such as Streamlit Sharing, Heroku, or even your own web server.

- The best thing about Streamlit is that we don't even need to know the basics of web development to get started or to create your first web application. So if you're somebody who's into data science and you want to deploy your models easily, quickly, and with only a few lines of code, Streamlit is a good fit.

- One of the important aspects of making an application successful is to deliver it with an effective and intuitive user interface. And Streamlit is a promising open-source Python library, which enables developers to build attractive user interfaces in no time.

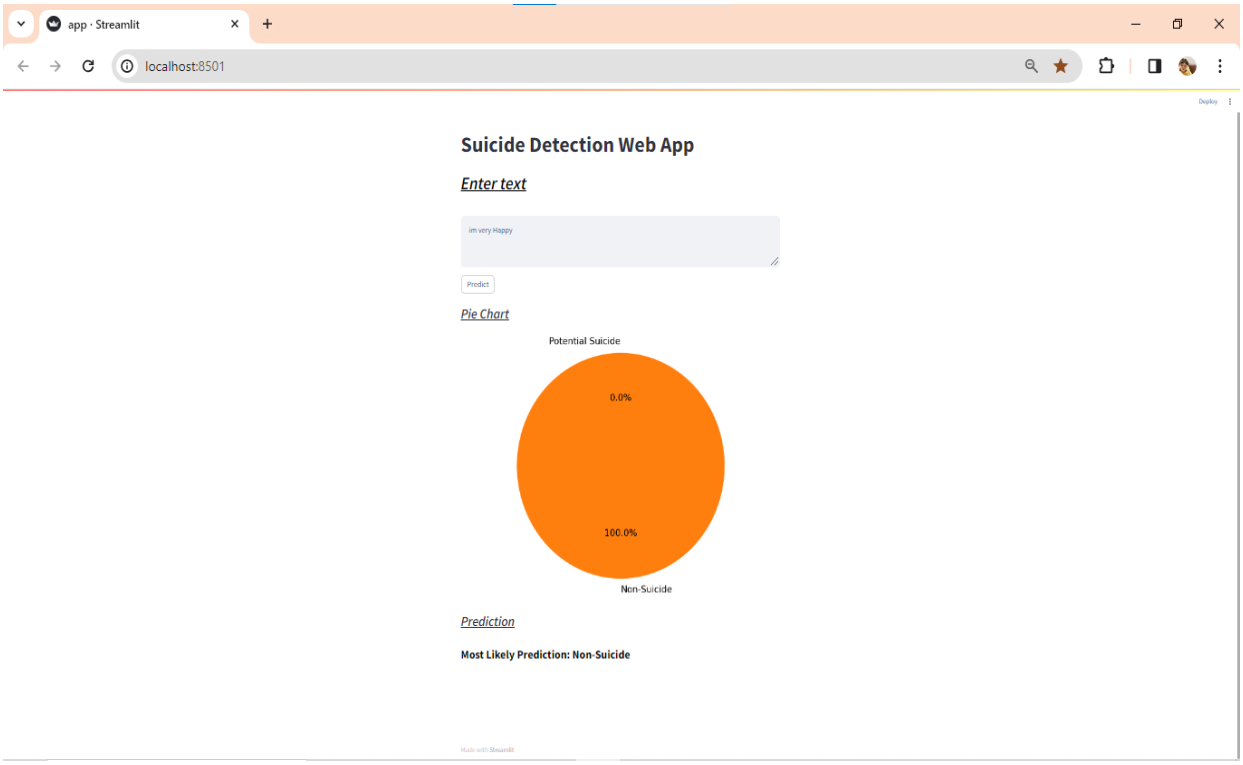


Figure: Streamlit Webapp Interface

4.1.4 VISUAL STUDIO CODE

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages and runtimes (such as C++, C#, Java, Python, PHP, Go, .NET).

Visual Studio is an Integrated Development Environment(IDE) developed by Microsoft to develop Desktop applications, GUI(Graphical User Interface), console, web applications, mobile applications, cloud, and web services, etc. With the help of this IDE, you can create managed code as well as native code. It uses the various platforms of Microsoft software development software like Windows store, Microsoft Silverlight, and Windows API, etc. It is not a language-specific IDE as you can use this to write code in C#, C++, VB(Visual Basic), Python, JavaScript, and many more languages. It provides support for 36 different programming languages. It is available for Windows as well as for macOS.

The first version of VS(Visual Studio) was released in 1997, named as Visual Studio 97 having version number 5.0. The latest version of Visual Studio is 15.0 which was released on March 7, 2017. It is also termed as Visual Studio 2017. The supported .Net Framework Versions in latest Visual Studio is 3.5 to 4.7. Java was supported in old versions of Visual Studio but in the latest version doesn’t provide any support for Java language.

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, Rust, and Julia. It is based on the Electron framework, which is used to develop Node.js web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services).

Out of the box, Visual Studio Code includes basic support for most common programming languages. This basic support includes syntax highlighting, bracket matching,code folding, and configurable snippets. Visual Studio Code also ships with IntelliSense for JavaScript, TypeScript, JSON, CSS, and HTML, as well as debugging support for Node.js.Support for

additional languages can be provided by freely available extensions on the VSCode Marketplace.

Instead of a project system, it allows users to open one or more directories, which can then be saved in workspaces for future reuse. This allows it to operate as a language-agnostic code editor for any language. It supports many programming languages and a set of features that differs per language. Unwanted files and folders can be excluded from the project tree via the settings. Many Visual Studio Code features are not exposed through menus or the user interface but can be accessed via the command palette.

4.2 Deep Learning Models

4.2.1 LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture that has gained popularity in deep learning for its ability to model sequences, time series data, and solve problems where data is dependent on previous information. LSTM networks are designed to overcome some of the limitations of traditional RNNs, such as the vanishing gradient problem, which can hinder learning long-range dependencies in sequential data.

Here are the key characteristics and components of LSTM in deep learning:

- Cell State: LSTMs maintain a cell state, which acts as a conveyor belt, allowing information to flow through the network without being altered. This helps in preserving information over long sequences.

- Three Gates: LSTMs have three types of gates that control the flow of information: the input gate, forget gate, and output gate. These gates are used to manage the cell state and determine what information is added or removed.
    - Input Gate: It decides what new information should be stored in the cell state.
    - Forget Gate: It controls what information should be discarded from the cell state.
    - Output Gate: It determines what the next hidden state and output should be based on the cell state.
- Hidden State: LSTMs also have a hidden state that is passed to the next time step, but it can be influenced by the cell state through the output gate. The hidden state is used to make predictions or

provide information for downstream tasks.

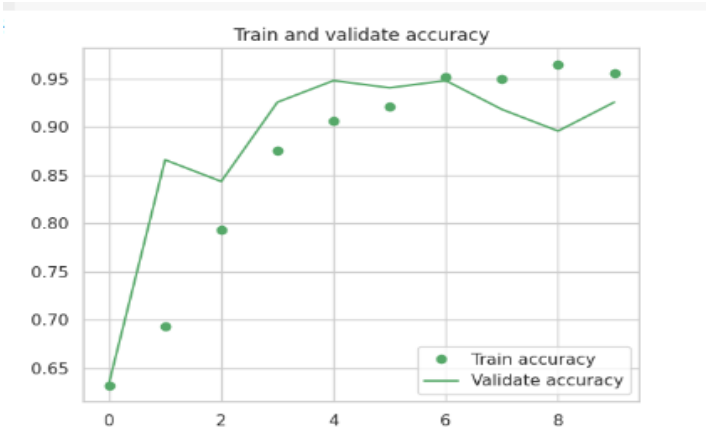
- Activation Functions: LSTMs use various activation functions, such as the sigmoid function for gates and the hyperbolic tangent (tanh) function for the cell state. These functions control the flow of information and prevent it from growing too large or vanishing.

- Long-Range Dependencies: One of the key advantages of LSTMs is their ability to capture long-range dependencies in sequential data. This makes them well-suited for tasks like natural language processing (NLP), speech recognition, and time series forecasting.

- Training: LSTMs are typically trained using gradient-based optimization algorithms, such as backpropagation through time (BPTT). They can be trained to minimize various loss functions, depending on the task, such as mean squared error (MSE) for regression or cross-entropy for classification.

- Variants: Over time, various LSTM variants have been developed, including Bidirectional LSTMs (BiLSTMs), which process sequences in both forward and backward directions, and Stacked LSTMs, which involve multiple LSTM layers stacked on top of each other.

- Applications: LSTMs have been successfully applied in a wide range of applications, including machine translation, sentiment analysis, speech recognition, handwriting recognition, and more. They are particularly useful when dealing with data sequences of varying lengths.



4.2.2 GRU:

Gated Recurrent Unit (GRU) is another type of recurrent neural network (RNN) architecture used in deep learning for sequence modeling and sequential data processing. It is similar in purpose to Long Short-Term Memory (LSTM) networks but has a simpler architecture. GRUs were designed to address some of the complexity and training challenges associated with LSTMs while still being effective in capturing temporal dependencies in sequential data.

Here are the key characteristics and components of GRUs in deep learning:

- Hidden State: Like LSTMs, GRUs maintain a hidden state that is updated at each time step. The hidden state carries information from the past and is used for making predictions or providing information for downstream tasks.

- Update Gate and Reset Gate: Instead of having separate input, forget, and output gates like LSTMs, GRUs use two gates: the update gate and the reset gate. These gates control the flow of information in a more streamlined way.

- Update Gate: The update gate decides what information from the previous hidden state should be passed on to the current hidden state.

- Reset Gate: The reset gate controls how much past information should be forgotten when computing the current hidden state.

- Hidden State Update: The computation of the new hidden state in a GRU is more straightforward and involves a weighted combination of the previous hidden state and a candidate update, which is influenced by the update and reset gates. This simplicity makes GRUs computationally more efficient than LSTMs.

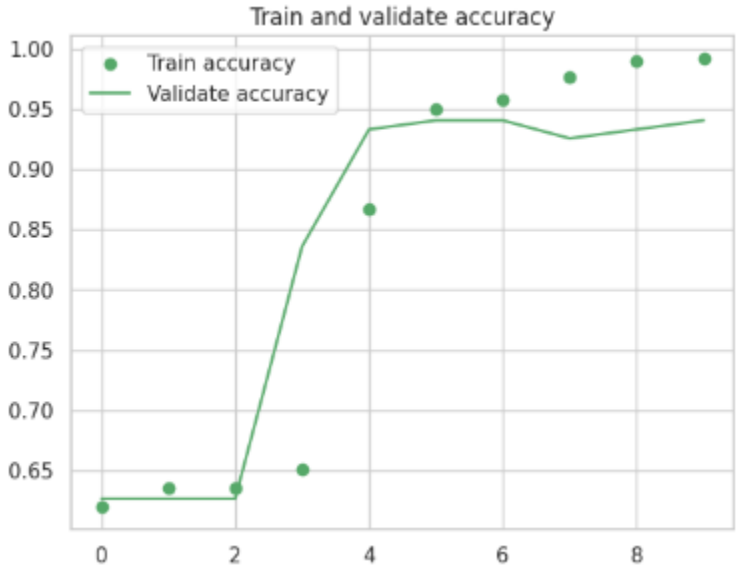
- Activation Functions: GRUs use activation functions, such as the sigmoid function, to control the update and reset gates and the hyperbolic tangent (tanh) function to generate the candidate update. These functions ensure that information is combined in a controlled manner.

- Long-Range Dependencies: Similar to LSTMs, GRUs are capable of capturing long-range dependencies in sequential data. This makes them suitable for tasks like natural language processing (NLP), speech recognition, and time series analysis.

- Training: GRUs are trained using gradient-based optimization algorithms like backpropagation through time (BPTT). The goal is to minimize a loss function suitable for the task, such as mean squared error (MSE) for regression or cross-entropy for classification.

- Simplicity: GRUs are favored when there is a need for a less complex architecture compared to LSTMs. They have fewer parameters and are often easier to train.

- Applications: GRUs have been successfully applied in various applications, including machine translation, sentiment analysis, speech recognition, and video analysis. They are particularly useful for tasks that involve sequential or time-series data.



4.2.3 CNN

Convolutional Neural Network (CNN) is used as part of a deep learning model for suicide ideation detection. CNNs are commonly associated with image processing tasks, but they can also be used for text classification and sequence modeling when combined with techniques like one-dimensional convolutions. Here's how CNNs are used in the provided code:

- Text Preprocessing: Before using CNNs, the text data is preprocessed to convert it into a format suitable for model input. This preprocessing includes tasks like tokenization, removing stopwords, lemmatization, and filtering non-English words.

- Feature Extraction: The text data is represented as a matrix, with rows corresponding to individual texts and columns representing features derived from the text. CountVectorizer and TfidfTransformer are used to convert the text data into a numerical format that can be fed into a deep learning model.

- Embedding Layer: An embedding layer is used to convert the discrete word indices in the text data into dense vector representations. This helps the model learn meaningful representations of words and their contextual relationships. In your code, the embedding layer is used with a specified vocabulary size and embedding dimension.

- Convolutional Layers: Convolutional layers are added to the model architecture to capture local patterns and features in the text data. These layers apply one-dimensional convolutions across the word embeddings, which can be thought of as sliding windows of various sizes to detect patterns in the input.

- Max-Pooling Layers: After the convolutional layers, max-pooling layers are used to downsample the feature maps and retain the most important information. Max-pooling helps reduce the dimensionality of the data while preserving the most relevant features.

- Recurrent Layers (LSTM, GRU): In addition to CNN layers, the model includes recurrent layers like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). These layers are commonly used for sequence modeling and capturing long-range dependencies in the text.

- Fully Connected Layers: The model architecture also includes fully connected (dense) layers for making final predictions. These layers can perform classification tasks by learning to map the extracted features to the output classes.

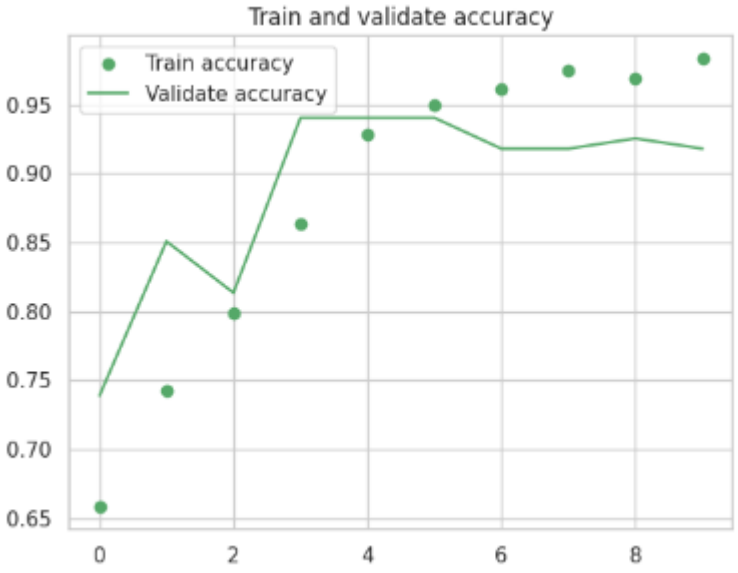
- Activation Functions: Activation functions like ReLU (Rectified Linear Unit) are used in the

convolutional and fully connected layers to introduce non-linearity to the model.

- Training and Evaluation: The model is trained using an appropriate loss function, such as binary cross-entropy, and an optimization algorithm. It's evaluated using metrics like accuracy.

- Predictions: The trained model can be used to predict whether a given input text is indicative of potential suicide ideation or not. The model provides a probability score for each class, and a threshold is applied to make the final classification decision.

In our project we used CNN with LSTM



4.3 SAMPLE CODE

#IMPORTING LIBRARES

```
!pip install Keras-Preprocessing
import numpy as np
import pandas as pd
import nltk
import tensorflow as tf
from nltk.corpus import stopwords
from nltk.corpus import reuters
```

```
from nltk.corpus import brown
from nltk.corpus import gutenberg
from nltk.tokenize import RegexpTokenizer
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import pickle
import joblib
from collections import Counter
from textblob import Word
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC, SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier,
AdaBoostClassifier, VotingClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_score, f1_score, recall_score
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, load_model
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras_preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from keras.layers import Activation, Dense, Embedding, LSTM, SpatialDropout1D, Dropout,
Flatten, GRU, Conv1D, MaxPooling1D, Bidirectional
from wordcloud import WordCloud,ImageColorGenerator
from PIL import Image
import urllib
```

```
import requests
from keras.regularizers import l2
!pip install ktrain
import ktrain
from ktrain import text
sns.set()
%matplotlib inline
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('gutenberg')
nltk.download('brown')
nltk.download("reuters")
nltk.download('words')

# LOAD DATASET
df=pd.read_csv("Twitter_Suicide_Dataset.csv", engine='python', encoding='UTF-8')
df=df.replace('Potential Suicide post ','Potential Suicide post')
df

 #DATA CLEANING
  df['Tweet']=df['Tweet'].fillna("")
  df.isna().sum()
#PREPROCESSING
 #Convert to lower case
df['lower_case']= df['Tweet'].apply(lambda x: x.lower())
#Tokenize
tokenizer = RegexpTokenizer(r'\w+')
df['Special_word'] = df.apply(lambda row: tokenizer.tokenize(row['lower_case']), axis=1)
#Stop words remove
stop = stopwords.words('english')
stop.remove("not")
stop.remove("here")
```

```
stop.remove("some")
df['stop_words'] = df['Special_word'].apply(lambda x: [item for item in x if item not in stop])
df['stop_words'] = df['stop_words'].astype('str')
#Filter words based on length
df['short_word'] = df['stop_words'].str.findall('\w{3,}')
df['string']=df['short_word'].str.join(' ')
#Removing non-english words(mention,emoji,link,special characters etc..)
words = set(nltk.corpus.words.words())
for w in reuters.words():
  words.add(w)
for w in brown.words():
  words.add(w)
for w in gutenberg.words():
  words.add(w)
df['NonEnglish'] = df['string'].apply(lambda x: " ".join(x for x in x.split() if x in words))
#Lemmatization
df['tweet'] = df['NonEnglish'].apply(lambda x: " ".join([Word(word).lemmatize() for word in
x.split()])).
```

#VISUALIZATION

```
fig = plt.figure(figsize=(14,7))
df['length'] = df.tweet.str.split().apply(len)
ax1 = fig.add_subplot(122)
sns.histplot(df['length'], ax=ax1,color='green')
describe = df.length.describe().to_frame().round(2)
ax2 = fig.add_subplot(121)
ax2.axis('off')
font_size = 14
bbox = [0, 0, 1, 1]
```

```
table = ax2.table(cellText = describe.values, rowLabels = describe.index, bbox=bbox,
colLabels=describe.columns)
table.set_fontsize(font_size)
fig.suptitle('Distribution of text length for positive sentiment tweets.', fontsize=16)
plt.show()
```

#APPLYING N-GRAM

```
x_train, x_test, y_train, y_test = train_test_split(df["tweet"],df["Suicide"], test_size = 0.25,
random_state = 42)
count_vect = CountVectorizer(ngram_range=(1, 2))
transformer = TfidfTransformer(norm='l2',sublinear_tf=True)
x_train_counts = count_vect.fit_transform(x_train)
x_train_tfidf = transformer.fit_transform(x_train_counts)
x_test_counts = count_vect.transform(x_test)
x_test_tfidf = transformer.transform(x_test_counts)
print (x_train_tfidf.shape,x_test_tfidf.shape, y_train.shape, y_test.shape)
```

#LSTM 1 LAYER

```
epochs = 10
emb_dim = 120
batch_size = 50
model_lstm1 = Sequential()
model_lstm1.add(Embedding(vocabulary_size,emb_dim,
input_length=X_DeepLearning.shape[1]))
model_lstm1.add(SpatialDropout1D(0.8))
model_lstm1.add(Bidirectional(LSTM(300, dropout=0.5, recurrent_dropout=0.5)))
model_lstm1.add(Dropout(0.5))
model_lstm1.add(Flatten())
model_lstm1.add(Dense(64, activation='relu'))
model_lstm1.add(Dropout(0.5))
model_lstm1.add(Dense(2, activation='softmax'))
model_lstm1.compile(optimizer=tf.optimizers.Adam(),loss='binary_crossentropy',
```

```
metrics=['acc'])
print(model_lstm1.summary())
checkpoint_callback = ModelCheckpoint(filepath="lastm-1-layer-best_model.h5",
save_best_only=True, monitor="val_acc", mode="max", verbose=1)
early_stopping_callback = EarlyStopping(monitor="val_acc", mode="max", patience=10,
verbose=1, restore_best_weights=True)
reduce_lr_callback = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5,
verbose=1, mode="min", min_delta=0.0001, cooldown=0, min_lr=0)
callbacks=[checkpoint_callback, early_stopping_callback, reduce_lr_callback]
results_1 = model_lstm1.evaluate(XX_test, y_test, verbose=False)
print(f'Test results - Loss: {results_1[0]} - Accuracy: {100*results_1[1]}%')
acc = history_lstm1.history['acc']
val_acc = history_lstm1.history['val_acc']
loss = history_lstm1.history['loss']
val_loss = history_lstm1.history['val_loss']
plt.plot( acc, 'go', label='Train accuracy')
plt.plot( val_acc, 'g', label='Validate accuracy')
plt.title('Train and validate accuracy')
plt.legend()
plt.figure()
plt.plot( loss, 'go', label='Train loss')
plt.plot( val_loss, 'g', label='Validate loss')
plt.title('Train and validate loss')
plt.legend()
plt.show()
# Load tokenizer object
with open('/content/tokenizer.pickle', 'rb') as handle:
    tokenizers = pickle.load(handle)

model = load_model('/content/lastm-1-layer-best_model.h5')
#model.save('/content/drive/MyDrive/Colab_Notebooks/DL Model/Twitter Suicide Ideation
Detection/lstm 1-layer.h5')
twt = ['i will not kill myself ']
```

```
twt = tokenizers.texts_to_sequences(twt)
twt = pad_sequences(twt, maxlen=60, dtype='int32')
predicted = model.predict(twt,batch_size=1,verbose = True)
if(np.argmax(predicted) == 0):
    print("Potential Suicide Post")
elif (np.argmax(predicted) == 1):
    print("Non Suicide Post")

#LSTM 2-LAYER
epochs = 10
emb_dim = 120
batch_size = 50
model_lstm2 = Sequential()
model_lstm2.add(Embedding(vocabulary_size,emb_dim
,input_length=X_DeepLearning.shape[1]))
model_lstm2.add(SpatialDropout1D(0.8))
model_lstm2.add(Bidirectional(LSTM(200, dropout=0.5, recurrent_dropout=0.5,
return_sequences= True)))
model_lstm2.add(Dropout(0.5))
model_lstm2.add(Bidirectional(LSTM(300, dropout=0.5, recurrent_dropout =0.5)))
model_lstm2.add(Dropout(0.5))
model_lstm2.add(Flatten())
model_lstm2.add(Dense(64, activation='relu'))
model_lstm2.add(Dropout(0.5))
model_lstm2.add(Dense(2, activation='softmax'))
model_lstm2.compile(optimizer=tf.optimizers.Adam(),loss='binary_crossentropy',
metrics=['acc'])
print(model_lstm2.summary())
checkpoint_callback = ModelCheckpoint(filepath="lastm-2-layer-best_model.h5",
save_best_only=True, monitor="val_acc", mode="max", verbose=1)
early_stopping_callback = EarlyStopping(monitor="val_acc", mode="max", patience=10,
verbose=1, restore_best_weights=True)
reduce_lr_callback = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5,
```

```
verbose=1, mode="min", min_delta=0.0001, cooldown=0, min_lr=0)
callbacks2=[checkpoint_callback, early_stopping_callback, reduce_lr_callback]
history_lstm2 = model_lstm2.fit(XX_train, y_train, epochs=epochs, batch_size=batch_size,
validation_split=0.1, callbacks=callbacks2)
results_2 = model_lstm2.evaluate(XX_test, y_test, verbose=False)
print(f'Test results - Loss: {results_2[0]} - Accuracy: {100*results_2[1]}%')
acc = history_lstm2.history['acc']
val_acc = history_lstm2.history['val_acc']
loss = history_lstm2.history['loss']
val_loss = history_lstm2.history['val_loss']
plt.plot( acc, 'go', label='Train accuracy')
plt.plot( val_acc, 'g', label='Validate accuracy')
plt.title('Train and validate accuracy')
plt.legend()
plt.figure()
plt.plot( loss, 'go', label='Train loss')
plt.plot( val_loss, 'g', label='Validate loss')
plt.title('Train and validate loss')
plt.legend()
plt.show()
# Load tokenizer object
with open('/content/tokenizer.pickle', 'rb') as handle:
    tokenizers = pickle.load(handle)
model = load_model('/content/lastm-2-layer-best_model.h5')
#model.save('/content/drive/MyDrive/Colab_Notebooks/DL Model/Twitter Suicide Ideation
Detection/lstm 2-layer.h5')
twt = ["i will not kill myself. "]
twt = tokenizers.texts_to_sequences(twt)
twt = pad_sequences(twt, maxlen=60, dtype='int32')
predicted = model.predict(twt,batch_size=1,verbose = True)
if(np.argmax(predicted) == 0):
    print("Potential Suicide Post")
elif (np.argmax(predicted) == 1):
```

```
    print("Non Suicide Post")


#GRU
epochs = 10
emb_dim = 120
batch_size = 50
model_gru = Sequential()
model_gru.add(Embedding(vocabulary_size,emb_dim
,input_length=X_DeepLearning.shape[1]))
model_gru.add(SpatialDropout1D(0.5))
model_gru.add(GRU(units=16,  dropout=0.2, recurrent_dropout=0.2,
kernel_regularizer=l2(0.01)))
model_gru.add(Dropout(0.5))
model_gru.add(Dense(228, activation='relu', kernel_regularizer=l2(0.01)))
model_gru.add(Dropout(0.5))
model_gru.add(Dense(2, activation='softmax'))
model_gru.compile(optimizer=tf.optimizers.Adam(),loss='binary_crossentropy', metrics=['acc'])
print(model_gru.summary())
checkpoint_callback = ModelCheckpoint(filepath="gru-best_model.h5", save_best_only=True,
monitor="val_acc", mode="max", verbose=1)
early_stopping_callback = EarlyStopping(monitor="val_acc", mode="max", patience=10,
verbose=1, restore_best_weights=True)
reduce_lr_callback = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5,
verbose=1, mode="min", min_delta=0.0001, cooldown=0, min_lr=0)
callbacks3=[checkpoint_callback, early_stopping_callback, reduce_lr_callback]
history_gru = model_gru.fit(XX_train, y_train, epochs=epochs,
batch_size=batch_size,validation_split=0.1, callbacks=callbacks3)
results_3 = model_gru.evaluate(XX_test, y_test, verbose=False)
print(f'Test results - Loss: {results_3[0]} - Accuracy: {100*results_3[1]}%')
acc = history_gru.history['acc']
val_acc = history_gru.history['val_acc']
loss = history_gru.history['loss']
val_loss = history_gru.history['val_loss']
```

```
    plt.plot( acc, 'go', label='Train accuracy')
    plt.plot( val_acc, 'g', label='Validate accuracy')
    plt.title('Train and validate accuracy')
    plt.legend()
    plt.figure()
    plt.plot( loss, 'go', label='Train loss')
    plt.plot( val_loss, 'g', label='Validate loss')
    plt.title('Train and validate loss')
    plt.legend()
    plt.show()
    # Load tokenizer object
    with open('/content/tokenizer.pickle', 'rb') as handle:
        tokenizers = pickle.load(handle)
    model = load_model('/content/gru-best_model.h5')
    #model.save('/content/drive/MyDrive/Colab_Notebooks/DL Model/Twitter Suicide Ideation
    Detection/gru-best_model.h5')
    twt = ["i will not kill myself."]
    twt = tokenizers.texts_to_sequences(twt)
    twt = pad_sequences(twt, maxlen=60, dtype='int32')
    predicted = model.predict(twt,batch_size=1,verbose = True)
    if(np.argmax(predicted) == 0):
        print("Potential Suicide Post")
    elif (np.argmax(predicted) == 1):
        print("Non Suicide Post")
#CNN+LSTM
epochs = 10
emb_dim = 120
batch_size = 50
model_cl = Sequential()
model_cl.add(Embedding(vocabulary_size,emb_dim, input_length=X_DeepLearning.shape[1]))
model_cl.add(SpatialDropout1D(0.8))
model_cl.add(Conv1D(filters=64, kernel_size=6, padding='same', activation='relu'))
```

```
model_cl.add(MaxPooling1D(pool_size=2))
model_cl.add(Conv1D(filters=32, kernel_size=6, activation='relu'))
model_cl.add(MaxPooling1D(pool_size=2))
model_cl.add(Bidirectional(LSTM(100, dropout=0.5, recurrent_dropout=0.5,
return_sequences=True)))
model_cl.add(Dropout(0.5))
model_cl.add(Bidirectional(LSTM(400, dropout=0.5, recurrent_dropout=0.5)))
model_cl.add(Dropout(0.5))
model_cl.add(Flatten())
model_cl.add(Dense(64, activation='relu'))
model_cl.add(Dropout(0.5))
model_cl.add(Dense(2, activation='softmax'))
model_cl.compile(optimizer='adam',loss='binary_crossentropy', metrics=['acc'])
print(model_cl.summary())
checkpoint_callback = ModelCheckpoint(filepath="cnn+lastm-best_model.h5",
save_best_only=True, monitor="val_acc", mode="max", verbose=1)

early_stopping_callback = EarlyStopping(monitor="val_acc", mode="max", patience=10,
verbose=1, restore_best_weights=True)
reduce_lr_callback = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=5, verbose=1,
mode="min", min_delta=0.0001, cooldown=0, min_lr=0)
callbacks=[checkpoint_callback, early_stopping_callback, reduce_lr_callback]
history_cl = model_cl.fit(XX_train, y_train, epochs=epochs,
batch_size=batch_size,validation_split=0.1, callbacks=callbacks)
results_4 = model_cl.evaluate(XX_test, y_test, verbose=False)
print(f'Test results - Loss: {results_4[0]} - Accuracy: {100*results_4[1]}%')
acc = history_cl.history['acc']
val_acc = history_cl.history['val_acc']
loss = history_cl.history['loss']
val_loss = history_cl.history['val_loss']
plt.plot( acc, 'go', label='Train accuracy')
plt.plot( val_acc, 'g', label='Validate accuracy')
plt.title('Train and validate accuracy')
```

```
plt.legend()
plt.figure()
plt.plot( loss, 'go', label='Train loss')
plt.plot( val_loss, 'g', label='Validate loss')
plt.title('Train and validate loss')
plt.legend()
plt.show()
# Load tokenizer object
with open('tokenizer.pickle', 'rb') as handle:
    tokenizers = pickle.load(handle)
model = load_model('/content/cnn+lastm-best_model.h5')
model_cl.save('/content/drive/MyDrive/Colab_Notebooks/DL Model/Twitter Suicide Ideation
Detection/CNN+LSTM.h5')
twt = ['I will not kill myself']
twt = tokenizer.texts_to_sequences(twt)
twt = pad_sequences(twt, maxlen=60, dtype='int32')

predicted = model.predict(twt,batch_size=1,verbose = True)
if(np.argmax(predicted) == 0):
    print("Potential Suicide Post")
elif (np.argmax(predicted) == 1):
    print("Non Suicide Post")
```

#MODEL COMPARISION

```
results=pd.DataFrame({'Model':['LSTM-1 Layer','LSTM-2 Layer','GRU','CNN+LSTM'],
                      'Accuracy Score':[results_1[1],results_2[1],results_3[1],results_4[1]]})
result_df=results.sort_values(by='Accuracy Score', ascending=False)
result_df=result_df.set_index('Model')
result_df
```

app.py

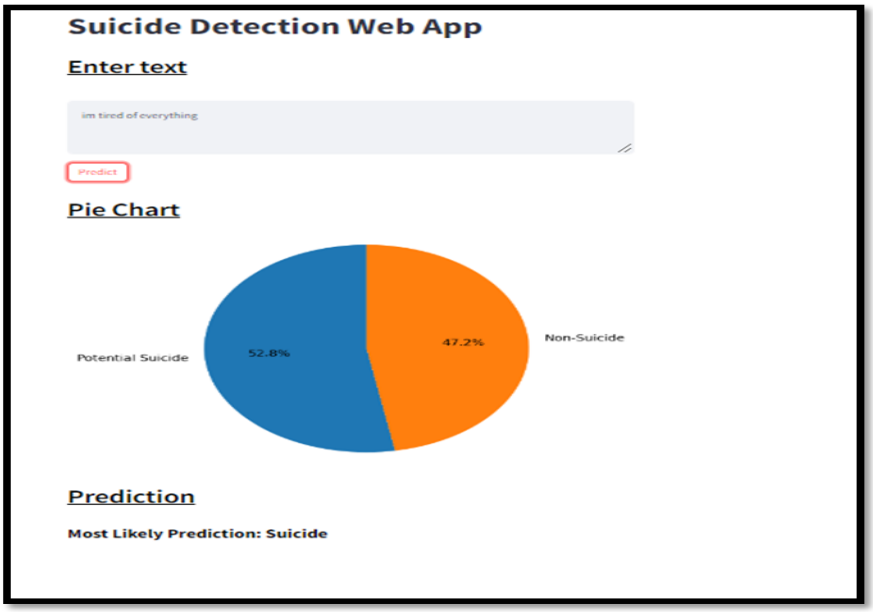
```
import pickle
```

```
import streamlit as st
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np
import matplotlib.pyplot as plt

# Load tokenizer object
with open('tokenizer.pickle', 'rb') as handle:
    tokenizers = pickle.load(handle)

models = {
    'LASTM 1 LAYER MODEL': load_model('lastm-1-layer-best_model.h5'),
    'LASTM 2 LAYER MODEL': load_model('lastm-2-layer-best_model.h5'),
    'GRU MODEL': load_model('gru-best_model.h5'),
    'CNN + LSTM MODEL': load_model('cnn+lastm-best_model.h5'),
}
st.title("Suicide Detection Web App")
st.markdown("<h2 style='color: black; font-style: italic; text-decoration: underline;'>Enter
text</h2>", unsafe_allow_html=True)
text = st.text_area("")
if st.button("Predict"):
    if text:
        predictions = {}
        max_percentage = 0
        most_likely_prediction = ''

 for model_name, model in models.items():
            twt = [text]
            twt = tokenizers.texts_to_sequences(twt)
            twt = pad_sequences(twt, maxlen=60, dtype='int32')
            predicted = model.predict(twt, batch_size=1, verbose=True)
            percentage = predicted[0][0]  # Percentage of Potential Suicide Post
```

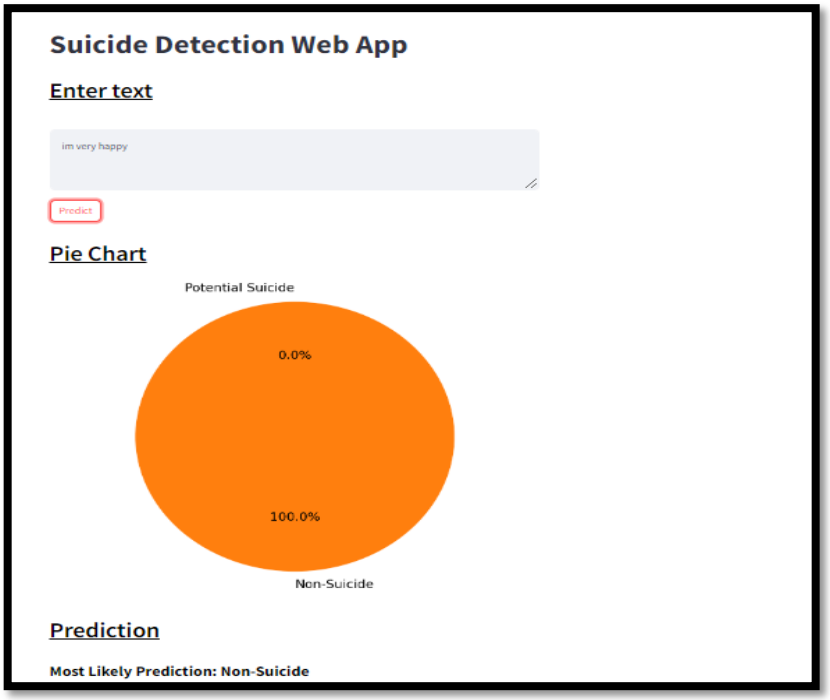
```
        if percentage > max_percentage:
            max_percentage = percentage
            most_likely_prediction = "Suicide" if percentage > 0.5 else "Non-Suicide"
        st.markdown("<h3 style='font-style: italic; text-decoration: underline;'>Pie Chart</h3>",
    unsafe_allow_html=True)

        # Create and display a pie chart
        fig, ax = plt.subplots()
        labels = ['Potential Suicide', 'Non-Suicide']
        sizes = [max_percentage, 1 - max_percentage]
        ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
        ax.axis('equal')
        st.pyplot(fig)
        st.markdown("<h3 style='font-style: italic; text-decoration: underline;'>Prediction</h3>",
    unsafe_allow_html=True)

# Display the most likely prediction in large fon
st.markdown(f"<h1 style='font-size: 24px; color: black;'>Most Likely Prediction:
{most_likely_prediction}</h1>", unsafe_allow_html=True)
```

4.4 OUTPUT SCREENS:

OUTPUT1: Suicidal Text Detection



Output2:Non Suicidal Text Detection:



Chapter-5

System Testing

5.1 PURPOSE OF TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests; each test type addresses a specific testing requirement. Testing and test design are parts of quality assurance that should also focus on bug prevention. A prevented bug is better than a detected and corrected bug. Testing consumes at least half of the time and work required to produce a functional program. History reveals that even well written programs still have 1-3 bugs per hundred statements. Testing is the process of executing a program with the aim of finding errors. To make our software perform well it should be error-free. If testing is done successfully, it will remove all the errors from the software.

5.2 TESTING STRATEGIES

In order to uncover the errors, present in different phases we have the concept of levels of testing. The Software testing has a prescribed order with the following list of software testing categories arranged in chronological order for our project testing and to generate test cases for output. These are the steps taken to fully test new software in preparation for marketing it:

Types of testing

- ***Unit testing*** performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.
- ***Integration testing*** done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.
- ***System testing*** done by a professional testing agent on the completed software product before it is introduced to the market.

- ***Acceptance testing*** - beta testing of the product done by the actual end users.

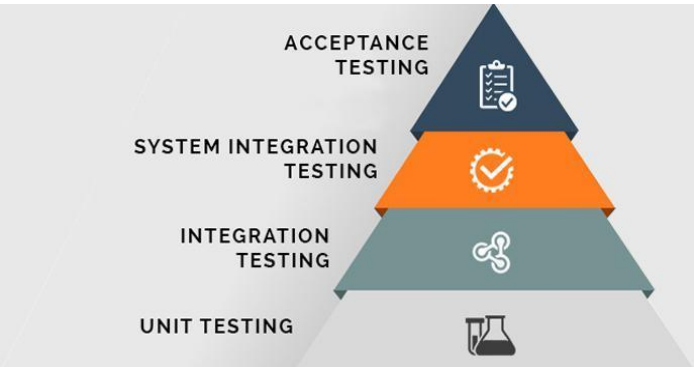


Fig 5.1 Types of Testing

5.2.1 UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software i.e., the module. Using the detailed design and the process specifications testing is done to uncover errors within the boundary of the module. All modules must be successful in the unit test before the start of the integration testing begins. It has been seen that each activity class runs after its development using unit testing.

Unit testing is performed at the first stage of testing as it is performed first of all other testing processes. Unit testing is also known as white box testing. So it's generally performed by developers. In our Project Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document. The three Modules of Fake Face Detection GANs, Phase I & Phase II individually tested in terms of detection of defects. Used to test each one of those functions, behavior is tested.

Here the Unit Testing Techniques are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behavior of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.

The Code coverage techniques used in our project under Unit Testing are listed below:

- Statement Coverage
- Decision Coverage

- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

In this project, the unit testing was performed on each module separately to find any defects in the code. The testing performed was manual walkthrough in the code of each module whether the functional call was correct or not, whether the attributes of functions called were accurate or not. By unit testing we found that without proper execution of each individual block of code we cannot further execute the project. Each module should give an accurate result and then should be moved towards the next testing process. A piece of code cannot work if the file mentioned was not found in mentioned directory, by Unit Testing we can find such faults. Through unit testing of this project we found that each individual module or block of code was working without any faults.

Below we have listed types of Unit testing in our project typically used:

Key Unit Tests:

- Data Preprocessing Unit Test: Ensures data cleaning, tokenization, stop-word removal, and other preprocessing steps are functioning correctly.
- Feature Extraction Unit Test: Validates feature extraction, including word count and n-gram generation.
- Model Training Unit Test: Tests each deep learning model's training process and model saving.
- Model Prediction Unit Test: Validates that models can make predictions based on input data.
- Data Loading Unit Test: Ensures that data can be loaded successfully.

5.2.2 INTEGRATION TESTING

Integration testing is the testing where multiple modules are tested to verify if different pieces of the modules are working together as per expectation or not. In case of Integration testing multiple modules get integrated and are tested as a single module so testers focus more on integrated functionality rather on internal design of the application.

In our Project the Integration testing is performed after all modules get integrated are done with unit tested i.e. Integration testing is done after unit testing and before System testing.Although each of our software module is unit tested, defects still exist for various reasons like below

- A Module, understanding and programming logic may differ. to verify the software modules, work in unity.
- At the time of module development, there are wide chances of change in requirements by the Users. These new requirements may not be unit tested.
- Interfaces of the software modules with the dataset are sometimes erroneous.
- External Hardware interfaces, if any, could be erroneous
- Exception handling causes issues in execution sometimes.

Integration Test Case differs from other test cases in the sense it focuses mainly on the interfaces & flow of data/information between the modules. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Through integration testing we made sure each module was integrated properly and there were no errors in integrating the modules. The information was passed smoothly from one module to other modules without missing any data.

Below we have listed types of integration testing in our project typically used:

Key Integration Tests:

- Data Pipeline Integration Test: Validates the complete data preprocessing pipeline from data loading to feature extraction.
- Model Integration Test: Ensures that deep learning models integrate into the system and make predictions correctly.
- Model Selection Integration Test: Verifies the correct selection of the best-performing model.
- Web Interface Integration Test: Tests the integration between the web interface and the deep learning models.

5.2.3 SYSTEM TESTING

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

In our project System Testing involves testing the software code for following

- Testing the fully integrated modules including external peripherals in order to check how components interact with one another and with the system as a whole. This is to support an End-to-End testing scenario.
- Verified with thorough testing of every input in individual modules to check for desired outputs.
- Tested for user's experience and expectations with the integrated modules and System Application.

Here the entire software system is tested. The reference document for this process is a requirements document and the goal was to see if software meets its requirements.

Below we have listed types of system testing in our project typically used:

Key System Tests:

- End-to-End Testing: Conducts complete tests from user input through the web interface to predictions to ensure the entire system works correctly.
- User Interface Testing: Verifies that users can interact with the system effectively.
- Performance Testing: Evaluates the system's response time and scalability under different workloads.
- Usability Testing: Collects user feedback to assess the system's usability and user-friendliness.

CHAPTER VI

CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

- Our project demonstrates the feasibility of using NLP and DL models to detect suicidal ideation on Twitter Tweets.
- The models we developed achieved high accuracy indicating their potential usefulness in identifying individuals who may be at risk of suicide.
- These findings suggest that NLP and DL models have the potential to contribute to suicide prevention efforts by identifying individuals who may need help and support.

6.2 FUTURE SCOPE

1. **Multilingual Support:** Extend the system to detect suicidal ideation in multiple languages. This would require creating or adapting models to work with various languages, as well as language detection for input text.
2. **Hybrid Models:** Investigate the combination of different deep learning architectures to create hybrid models that may improve detection accuracy.
3. **Natural Language Understanding (NLU):** Incorporate NLU techniques to understand the nuances and context within text. This can improve the accuracy of identifying ambiguous or sarcastic statements.
4. **Data Enrichment:** Integrate data from various sources, such as geolocation and user demographics, to gain a more comprehensive understanding of user behavior.
5. **User Support Integration:** Connect the system with support services and helplines, allowing it to automatically refer at-risk individuals to the appropriate resources.
6. **Feedback Mechanisms:** Implement mechanisms for feedback and improvement through user and professional input. This can help the system adapt and improve over time.
7. **Mobile Applications:** Develop mobile applications that incorporate the suicide ideation detection system, making it more accessible to users and enabling real-time assistance.

APPENDIX

REFERENCES

Papers and articles

1) Rabani, S. T., Khan, Q. R., &Khanday, A. M. U. D. (2020). Detection of suicidal ideation on Twitter using machine learning &ensemble approaches. Baghdadsciencejournal, 17(4), 1328- 1328.
2) Desmet B, Hoste V. Online suicide prevention through optimised text classification. Inf Sci (Ny). 2018; 439–440:61–78.
3) Vioules MJ, Moulahi B, Aze J, Bringay S. Detection of suicide-related posts in Twitter data streams. IBM J Res Dev.2018; 62(1):7:1-7:12.
4) De Choudhury M, Kıcıman E. The Language of Social Support in Social Media and its Effect on Suicidal Ideation Risk. Proc. Int AAAI Conf Weblogs Soc Media Int AAAI Conf Weblogs Soc Media.2017; 2017.
5) AL-Jumaili AS. A Hybrid Method of Linguistic and Statistical Features for Arabic Sentiment Analysis. Baghdad Sci J.2020; 17(1(Suppl.)):0385.
6) Dar AR, Ravindran D. Fog computing resource optimization: A review on current scenarios and resource management. Baghdad Sci J.2019; 16(2):419–27.
7) Joseph A, Ramamurthy B. Suicidal behavior prediction using data mining techniques. Int J Mech Eng Technol.2018;9(4):293–301.
8) Coppersmith G, NgoK, Leary R, Wood A. Exploratory Analysis ofSocialMediaPriortoa Suicide Attempt. Proc Third Work Comput Lingusitics Clin Psychol.2016; 106–17.
9) Anand N, Goyal D, Kumar T. Analyzing and Preprocessing the Twitter Data for Opinion Mining. In: Lecture Notes in Networks and Systems. Springer; 2018. P.213–21.
10) Dar AR, Ravindran D. Fog computing resource optimization: A review on current scenarios and resource management. Baghdad Sci J.2019;16(2):419–27.
11) Sikander D, Arvaneh M, Amico F, Healy G, Ward T, Kearney D, et al. Predicting risk of suicide using resting state heart rate. In: 2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, APSIPA 2016. Institute of Electrical and Electronics Engineers Inc.; 2017.
12) Varathan KD, Talib N. Suicide detection system based on Twitter. Proc 2014 Sci Inf Conf SAI 2014.2014; 785–8.