

PREDICTION OF HATE SPEECH AND DEPRESSION CLASSIFICATION BY USING ML TECHNIQUES

A PROJECT REPORT

Submitted by

PAVITHRA B (211420104190)

PRIYADHARSHINI R (211420104206)

RAJA RAJESWARI R (211420104335)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MARCH 2024

PANIMALAR ENGINEERING COLLEGE
(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **PREDICTION OF HATE SPEECH AND DEPRESSION CLASSIFICATION BY USING ML TECHINQUES** is the Bonafide work of **PAVITHRA B (211420104190), PRIYADHARSHINI R (211420104206), RAJA RAJESWARI R (211420104335)** who carried out the project work under my supervision.

SIGNATURE

Dr.L.JABA SHEELA, M.E.,Ph.D.,
PROFESSOR,
HEAD OF THE DEPARTMENT

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

Ms.S.YAMUNA DEVI, M.Tech.,
SUPERVISOR,
ASSISTANT PROFESSOR

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) were examined in the Project Viva-Voce Examination
held on

INTERNAL EXAMINER

EXTERNAL EXAMIER

DECLARATION BY THE STUDENT

We **PAVITHRA B (211420104190)**, **PRIYADHARSHINI R (211420104206)**, **RAJA RAJESWARI R (211420104335)** hereby declare that this project report titled **PREDICTION OF HATE SPEECH AND DEPRESSION CLASSIFICATION** under the guidance of **Ms. S.YAMUNA DEVI, M.Tech.**, is the original work done by us and we have plagiarized or submitted to any other degree in any university by us.

PAVITHRA B

PRIYADHARSHINI R

RAJA RAJESWARI R

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A.,Ph.D.**, for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors **Tmt.C.VIJAYARAJESWARI, Dr.C.SAKTHI KUMAR,M.E.,Ph.D.**, and **Dr.SARANYASREE SAKTHI KUMAR B.E.,M.B.A.,Ph.D.**, for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI,M.E.,Ph.D.**, who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr. L.JABA SHEELA, M.E.,Ph.D.**, for the support extended throughout the project.

We would like to thank our Project Guide **Ms.S.YAMUNA DEVI, M.Tech.**, and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

PAVITHRA B (211420104190)

PRIYADHARSHINI R (211420104206)

RAJA RAJESWARI (211420104335)

ABSTRACT

The rise of social media platforms has brought about both positive and negative impacts on society. One significant challenge is the prevalence of hate speech and its potential contribution to mental health issues such as depression. This abstract presents an overview of the classification of hate speech and depression using machine learning techniques. To address the problem of hate speech, a machine learning-based approach is proposed. By employing supervised learning algorithms, a model is trained on a labelled dataset containing instances of hate speech to identify patterns and features indicative of hate speech. The effectiveness of the model is evaluated using appropriate performance metrics, aiming to achieve accurate hate speech detection. Furthermore, the abstract highlights the classification of depression using machine learning methods. By leveraging both supervised and unsupervised learning techniques, a model is developed to analyze textual data, particularly social media posts.

Keywords: Hate speech, depression, classification, machine learning, supervised learning, unsupervised learning, social media, online environment, mental health.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	V
	LIST OF FIGURES	IX
	LIST OF SYMBOLS	X
1.	INTRODUCTION	
	1.1 Overview	01
	1.2 Scope of the project	01
	1.3 Objectives of the project	02
	1.4 Problem Definition	02
2.	LITERATURE SURVEY	
	2.1 Literature Survey 1	03
	2.2 Literature Survey 2	04
	2.3 Literature Survey 3	05
	2.4 Literature Survey 4	06
	2.5 Literature Survey 5	07
3.	SYSTEM ANALYSIS	
	3.1 Existing System	08
	3.2 Feasibility Study	10

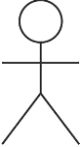
3.3.1 Data Wrangling	10
3.3.2 Data Collection	10
3.3.3 Preprocessing	10
3.3.4 Building the classification model	11
3.3.5 Construction of a Predictive Model	11
3.4 System Requirements	13
3.4.1 Functional Requirements	13
3.4.2 Non-Functional Requirements	13
3.4.3 Environment Requirements	14
3.5 Software Description	15
3.5.1 Anaconda Navigator	16
3.5.2 Jupyter Notebook	17
3.5.3 Python	20
4. SYSTEM DESIGN	
4.1 ER Diagram	27
4.2 Work Flow Diagram	29
4.3 Usecase Diagram	31
4.4 Class Diagram	32
4.5 Activity Diagram	33
4.6 Sequence Diagram	34
4.7 Collaboration Diagram	35
5. SYSYEM ARCHITECTURE	
5.1 Algorithm and Techniques	36
5.2 Module Description	37

	5.2.1 Data Pre-processing	40
	5.2.2 Data Visualization	42
	5.2.3 Decision Tree Classifier	44
	5.2.4 Random Forest Classifier	47
6.	SYSTEM IMPLEMENTATION	
	6.1 Server Side	63
7.	TESTING AND PERFORMANCE ANALYSIS	74
8.	CONCLUSION	
	8.1 Results and Discussions	78
	8.2 Conclusion and Future Enhancement	78
	APPENDICES	
	A.1 Sample Screens	79
	A.2 Plagiarism Report	83
	A.3 Paper Publication	100
	REFERENCES	101

LIST OF FIGURES

FIG NO	TITLE	PAGE NO
3.3.5.1	Process of Dataflow Diagram	12
4.1	ER Diagram	28
4.2	Work Flow Diagram	30
4.3	Usecase Diagram	31
4.4	Class Diagram	32
4.5	Activity Diagram	33
4.6	Sequence Diagram	34
4.7	Collaboration Diagram	35
5.1	Architecture Diagram	36
7.1	Confusion Matrix For Decision Tree Classifier Algorithm	76
7.2	Confusion Matrix For Random Forest Algorithm	77

LIST OF SYMBOLS

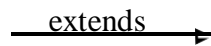
S.NO	NAME	NOTATION	DESCRIPTION
1.	Class	<div><div><div>+ <i>public</i></div><div>- <i>private</i></div></div><div><div><i>Class Name</i></div><div>- <i>attribute</i></div><div>- <i>attribute</i></div></div></div>	Represents a collection of similar entities grouped together.
2.	Association	<div><div>Class A</div><div>NAME</div><div>Class B</div></div> <div><div>Class A</div><div></div><div>Class B</div></div>	Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		It aggregates several classes into a single class.
4	Aggregation	<div><div>Class A</div><div>↑</div><div>Class B</div></div> <div><div>Class A</div><div>↑</div><div>Class B</div></div>	Interaction between the system and external environment

5. Relation
(uses)

uses

Used for additional
process communication.

6. Relation
(extends)



The Extended relationship is
used when one use case is
similar to another use case but
does a bit more.

7. Communication



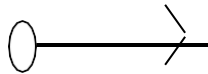
Communication between
various use cases.

8. State



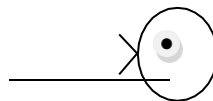
State of the processes.

9. Initial State



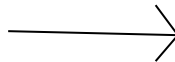
The Initial state of the object

10. Final state



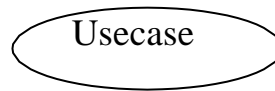
The Final state of the object

11. Control flow



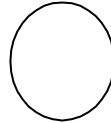
Represents various control
flows between the states.

12. Use case



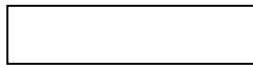
Interaction between the system and external environment.

13. Data
Process/State



A circle in DFD represents a state or process which has been triggered due to some event or action.

14. External entity



Represents external entities such as keyboards, sensors, etc.

15. Transition



Represents communication that occurs between processes.

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

Hate speech and depression state of person is a real problem in today's world, and it has become more extensive and harder to identify. A major challenge is to detect it in the early phase. Another challenge in this is the unavailability or the shortage of labelled data for training the detection models.

1.2 SCOPE OF THE PROJECT

The scope of hate speech and depression detection using NLP encompasses vital aspects of online safety, mental health support, and content moderation. In the realm of hate speech, it involves safeguarding social media spaces by identifying and curtailing discriminatory language, cyberbullying, and harassment. Additionally, it extends to legal compliance, ensuring adherence to hate speech laws and regulations while fostering a positive online community. On the mental health front, NLP facilitates the early detection of individuals expressing signs of depression in their online communication, enabling timely intervention and the provision of mental health resources. The challenges involve navigating ethical considerations, preserving user privacy, addressing cultural sensitivities, and continually improving models to adapt to evolving language nuances. Technologically, it requires advanced NLP techniques, machine learning models, and a focus on explainability to enhance accuracy and transparency in predictions, particularly in the sensitive domain of mental health. In essence, the scope of NLP in hate speech and depression detection is integral to creating safer digital spaces and offering timely support for mental well-being.

1.3 OBJECTIVES OF THE PROJECT

The objective of this project is to develop and implement Natural Language Processing (NLP) models for effective detection of hate speech and signs of depression in online communication. The goal is to enhance online safety by mitigating harmful content, fostering positive digital communities, and providing timely support for mental well-being.

1.4 PROBLEM DEFINITION

The Problem of this project is to develop and implement Natural Language Processing (NLP) models for effective detection of hate speech and signs of depression in online communication. The goal is to enhance online safety by mitigating harmful content, fostering positive digital communities, and providing timely support for mental health well-being.

CHAPTER 2

LITERATURE SURVEY

INTRODUCTION

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time. Its goal is to bring the reader up to date with current literature on a topic and form the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

2.1 TITLE: Interpretation of Depression Detection Models via Feature Selection Methods

AUTHOR: Sharifa Alghowinem , Tom Gedeon , Senior Member, IEEE, Roland Goecke , Senior Member, IEEE, Jeffrey F. Cohn , and Gordon Parker

YEAR: 2023

Given the prevalence of depression worldwide and its major impact on society, several studies employed artificial intelligence modelling to automatically detect and assess depression. However, interpretation of these models and cues are rarely discussed in detail in the AI community, but have received increased attention lately. In this article, we aim to analyse the commonly selected features using a proposed framework of several feature selection methods and their effect on the classification results, which will provide an interpretation of the depression detection model. The developed framework aggregates and selects the most promising features for

modelling depression detection from 38 feature selection algorithms of different categories. Using three real-world depression datasets, 902 behavioral cues were extracted from speech behavior, speech prosody, eye movement and head pose. To verify the generalizability of the proposed framework, we applied the entire process to depression datasets individually and when combined. The results from the proposed framework showed that speech behavior features (e.g. pauses) are the most distinctive features of the depression detection model. From the speech prosody modality, the strongest feature groups were F0, HNR, formants, and MFCC, while for the eye activity modality they were left-right eye movement and gaze direction, and for the head modality it was yaw head movement. Modelling depression detection using the selected features (even though there are only 9 features) outperformed using all features in all the individual and combined datasets. Our feature selection framework did not only provide an interpretation of the model, but was also able to produce a higher accuracy of depression detection with a small number of features in varied datasets. This could help to reduce the processing time needed to extract features and creating the model.

2.2 TITLE: Ordinal Logistic Regression With Partial Proportional Odds for Depression Prediction

AUTHOR: Sadari Jayawardena , Member, IEEE, Julien Epps , Member, IEEE, and Eliathamby Ambikairajah , Senior Member, IEEE

YEAR: 2023

Like many psychological scales, depression scales are ordinal in nature. Depression prediction from behavioral signals has so far been posed either as classification or regression problems. However, these naive approaches have fundamental issues because they are not focused on ranking, unlike ordinal regression, which is the most appropriate approach. Ordinal regression to date has comparatively few methods when compared with other branches in machine learning, and its usage is limited to specific research domains. Ordinal Logistic Regression (OLR) is one such method,

which is an extension for ordinal data of the well-known logistic regression, but is not familiar in speech processing, affective computing or depression prediction. The primary aim of this article is to investigate proportionality structures and model selection for the design of ordinal regression systems within the logistic regression framework. A new greedy-based algorithm for partial proportional odds model selection (GREP) is proposed that allows the parsimonious design of effective ordinal logistic regression models, which avoids an exhaustive search and outperforms model selection using the Brant test. Evaluations on the DAIC-WOZ and AVID depression corpora show that OLR models exploiting GREP can outperform two competitive baseline systems (GSR and CNN), in terms of both RMSE and Spearman correlation.

2.3 TITLE: Machine Learning Models for Hate Speech and Offensive Language Identification for Indo-Aryan Language: Hindi

AUTHOR: Purva Mankar , Akshaya Gangurde , Deptii Chaudhari and Ambika Pawar

YEAR: 2023

Automated recognition and detection of Hate Speech and Offensive language on different Online Social Networks, mainly Twitter, presents a challenge to the community of Artificial Intelligence and Machine Learning. Unfortunately, sometimes these ideas communicated via the internet are intended to promote or incite hatred or humiliation of an individual, community, or even organizations. The HASOC shared task is to attempt to automatically detect abusive language on Twitter in English and Indo-Aryan Languages like Hindi. To participate in this task and provide our input, we Team Data Pirates presented several machine learning models for Hindi Subtasks. The datasets provided allowed the development and testing of supervised machine learning techniques. The top 2 performing models for sub-task A were Naïve Bayes and Logistic Regression with the same Macro F1 score of 0.7394. The top 2 performing models for sub-task B were Logistic Regression and Cat Boost, with Macro F1 scores of 0.4828 and 0.4709, respectively. This overview intends to provide detailed understandings and to analyze the outcomes.

2.4 TITLE: Classifying Hate Speech Using a Two-Layer Model**AUTHOR:** Yiwen Tang & Nicole Dalzell**YEAR:** 2019

Social media and other online sites are being increasingly scrutinized as platforms for cyberbullying and hate speech. Many machine learning algorithms, such as support vector machines, have been adopted to create classification tools to identify and potentially filter patterns of negative speech. While effective for prediction, these methodologies yield models that are difficult to interpret. In addition, many studies focus on classifying comments as either negative or neutral, rather than further separating negative comments into subcategories. To address both of these concerns, we introduce a two-stage model for classifying text. With this model, we illustrate the use of internal lexicons, collections of words generated from a pre classified training dataset of comments that are specific to several subcategories of negative comments. In the first stage, a machine learning algorithm classifies each comment as negative or neutral, or more generally target or nontarget. The second stage of model building leverages the internal lexicons (called L2CLs) to create features specific to each subcategory. These features, along with others, are then used in a random forest model to classify the comments into the subcategories of interest. We demonstrate our approach using two sets of data. Supplementary materials for this article are available online.

2.5 TITLE: Automatic Hate Speech Detection Using Deep Neural Networks and Word Embedding**AUTHOR:** Olumide Ebenezer Ojo , Thang-Hoang Ta1, Alexander Gelbukh , Hiram Calvo , Grigori Sidorov , Olaronke Oluwayemisi Adebajji**YEAR:** 2022

Hatred spreading through the use of language on social media platforms and in online groups is becoming a well-known phenomenon. By comparing two text

representations: bag of words (BOW) and pre-trained word embedding using GLOVE, we used a binary classification approach to automatically process user contents to detect hate speech. The Naive Bayes Algorithm (NBA), Logistic Regression Model (LRM), Support Vector Machines (SVM), Random Forest Classifier (RFC) and the one-dimensional Convolutional Neural Networks (1D-CNN) are the models proposed. With a weighted macro-F1 score of 0.66 and a 0.90 accuracy, the performance of the 1D-CNN and GLOVE embeddings was best among all the models.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Physiological studies have shown that there are some differences in speech and facial activities between depressive and healthy individuals. Based on this fact, we propose a novel spatio-temporal attention (STA) network and a multimodal attention feature fusion (MAFF) strategy to obtain the multimodal representation of depression cues for predicting the individual depression level. Specifically, we first divide the speech amplitude spectrum/video into fixed-length segments and input these segments into the STA network, which not only integrates the spatial and temporal information through attention mechanism, but also emphasizes the audio/ video frames related to depression detection. The audio/video segment-level feature is obtained from the output of the last full connection layer of the STA network. Second, this article employs the eigen evolution pooling method to summarize the changes of each dimension of the audio/video segment-level features to aggregate them into the audio/video level feature. Third, the multimodal representation with modal complementary information is generated using the MAFF and inputs into the support vector regression predictor for estimating depression severity. Experimental results on the AVEC2013 and AVEC2014 depression databases illustrate the effectiveness of our method.

Disadvantages:

- They focused on only Depression.
- They did not compared more than an algorithms to getting better accuracy level.
- Accuracy was low.
- More time complex to do that.

3.2 PROPOSED SYSTEM

The proposed system for hate speech and depression classification integrates state-of-the-art natural language processing. These models have demonstrated remarkable success in capturing contextual information and understanding language nuances. The system utilizes a large, labeled dataset of hate speech and depression instances to fine-tune the pre-trained NLP model. The proposed system for hate speech and depression classification integrates various machine learning techniques and feature engineering approaches to classify individuals at risk of depression.

Supervised learning algorithms, such as Support Vector Machines, Random Forests, or deep learning architectures like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM), are employed to train models on labeled datasets. These models analyze individuals' textual data, incorporating sentiment analysis, linguistic patterns, and topic modeling, along with other relevant features derived from user posts.

To capture the temporal nature of depression symptoms, recurrent neural network architectures like RNN and LSTM can be employed to model sequential dependencies in the text. This enables the system to capture the contextual and temporal aspects of user expressions related to depression.

Advantages:

- We apply NLP techniques for predict Hate speech & depression.
- Accuracy was improved.
- We build a production level application for deployment purpose.
- We improved performance level.

3.3 FEASIBILITY STUDY

3.3.1 Data Wrangling

In this section of the report will load in the data, check for cleanliness, and then trim and clean given dataset for analysis. Make sure that the document steps carefully and justify for cleaning decisions.

3.3.2 Data Collection

The data collection process for predictive tasks typically involves splitting a dataset into a Training set and a Test set, commonly utilizing an 80:20 or 70:30 ratio. The gathered text data is then preprocessed, including tasks such as tokenization, stemming, and vectorization, to transform the raw text into a format suitable for machine learning algorithms.

3.3.3 Preprocessing

The data which was collected might contain missing values that may lead to inconsistency. To gain better results data need to be preprocessed so as to improve the efficiency of the algorithm. The outliers have to be removed and also variable conversion need to be done.

3.3.4 Building the classification model

The prediction of Hate speech and depression classification, a high accuracy prediction model is effective because of the following reasons: It provides better results in classification problem.

- It is strong in preprocessing outliers, irrelevant variables, and a mix of continuous, categorical and discrete variables.
- It produces out of bag estimate error which has proven to be unbiased in many tests and it is relatively easy to tune with.

3.3.5 Construction of a Predictive Model

Machine learning needs data gathering have lot of past data. Data gathering have sufficient historical data and raw data. Before data pre-processing, raw data can't be used directly. It's used to pre-process then, what kind of algorithm with model. Training and testing this model working and predicting correctly with minimum errors. Tuned model involved by tuned time to time with improving the accuracy.

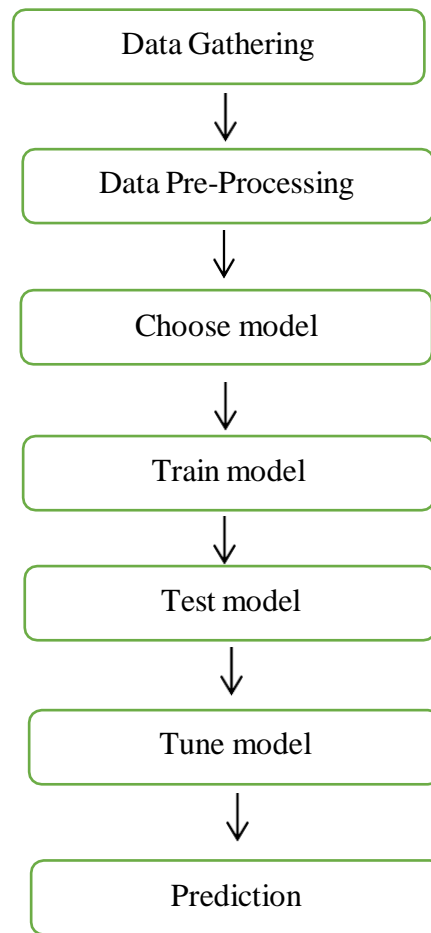


Fig 3.3.5.1 Process of dataflow diagram

3.4 SYSTEM REQUIREMENTS

Requirements are the basic constraints that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements
2. Non-Functional requirements
3. Environment requirements
 - A. Hardware requirements
 - B. software requirements

3.4.1 Functional requirements:

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like sk-learn, pandas, numpy, matplotlib and seaborn.

3.4.2 Non-Functional Requirements:

Process of functional steps,

1. Problem Statement
2. Preparing data
3. Evaluating algorithms
4. Improving results
5. Prediction the result

3.4.3 Environment Requirements:

1. Software Requirements:

Operating System : Windows 10 or later

Tool : Anaconda with Jupyter Notebook

2. Hardware requirements:

Processor : Intel i3

Hard disk : minimum 80 GB

RAM : minimum 2 GB

3.5 SOFTWARE DESCRIPTION

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system “Conda”. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400 packages as well as the Conda package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the Conda install command or using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the `Conda build` command, and can be shared with others by uploading them to Anaconda Cloud, Pip or other repositories.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

3.5.1 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- Spyder
- PyCharm
- VSCode
- Glueviz
- Orange 3 App
- RStudio

- Anaconda Prompt (Windows only)
- Anaconda PowerShell (Windows only)

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Anaconda comes with many built-in packages that you can easily find with `conda list` on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

Conda

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Mini conda, and Anaconda Repository.

Anaconda is freely available, open source distribution of python and R programming languages which is used for scientific computations. If you are doing any machine learning or deep learning project then this is the best place for you. It consists of many software's which will help you to build your machine learning project and deep learning project. These software's have great graphical user interface and these will make your work easy to do. You can also use it to run your python script. These are the software carried by anaconda navigator.

3.5.2 Jupyter Notebook

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and

to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

Launching Jupyter Notebook App:

Launching Jupyter Notebook App: The Jupyter Notebook App can be launched by clicking on the *Jupyter Notebook* icon installed by Anaconda in the start menu (Windows) or by typing in a terminal (*cmd* on Windows): “jupyter notebook”

When started, the Jupyter Notebook App can access only files within its start-up folder (including any sub-folder). No configuration is necessary if you place your notebooks in your home folder or subfolders. Otherwise, you need to choose a Jupyter Notebook App start-up folder which will contain all the notebooks.

Executing a notebook:

Download the notebook you want to execute and put it in your notebook folder (or a sub-folder of it).

- ❖ Launch the jupyter notebook app
- ❖ In the Notebook Dashboard navigate to find the notebook: clicking on its name will open it in a new browser tab.
- ❖ Click on the menu *Help -> User Interface Tour* for an overview of the Jupyter Notebook App user interface.

- ❖ You can run the notebook document step-by-step (one cell a time) by pressing *shift + enter*.
- ❖ You can run the whole notebook in a single step by clicking on the menu *Cell - > Run All*.
- ❖ To restart the kernel (i.e. the computational engine), click on the menu *Kernel - > Restart*. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc...).

File Extension

An **IPYNB** file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

Working Process

- Download and install anaconda and get the most useful package for machine learning in Python.
- Load a dataset and understand its structure using statistical summaries and data visualization.
- Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

When you are applying machine learning to your own datasets, you are working on a project. A machine learning project may not be linear, but it has a number of well-known steps:

- Define Problem.
- Prepare Data.
- Evaluate Algorithms.
- Improve Results.
- Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

3.5.3 Python

Introduction

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its 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 structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

History

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator for Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a 5-member "Steering Council" to lead the project. As of 2021, the current members of this council are Barry Warsaw, Brett Cannon, Carol Willing, Thomas Wouters, and Pablo Galindo Salgado.

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.6.x and later are supported.

Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues, leading to possible remote code execution and web cache poisoning.

Design Philosophy & Feature

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 meta-programming and meta-objects (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's design offers some support for functional programming in the Lisp tradition. It has filter, map and reduce functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible (with modules). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard

library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the C-Python reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is *pythonic* is to say that it uses Python idioms well, that it is natural or shows fluency in the language, and that it conforms to Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as *Pythonistas*.

Syntax and Semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are allowed but are rarely, if ever, used. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation

Main article : Python syntax and semantics & Indentation

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 does not have any semantic meaning. The recommended indent size is four spaces.

Expressions

Some Python expressions are similar to those found in languages such as C and Java, while some are not:

- Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) `//` and floating-point division. Python also uses the `**` operator for exponentiation.
- From Python 3.5, the new `@` infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.
- From Python 3.8, the syntax: `=`, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.
- Python uses the words `and`, `or`, `not` for its Boolean operators rather than the symbolic `&&`, `||`, `!` Used in Java and C.
- Python has a type of expression termed a list comprehension as well as a more general expression termed a generator expression.
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.

- Conditional expressions in Python are written as `x if c else y` (different in order of operands from the `c ? x : y` operator common to many other languages).
- Python has a "string format" operator `%`. This functions analogously to `printf` format strings in C, e.g. `"spam=%s eggs=%d" % ("blah",2)` evaluates to `"spam=blah eggs=2"`. In Python 3 and 2.6+, this was supplemented by the `format()` method of the `str` class, e.g. `"spam={0} eggs={1}".format("blah",2)`. Python 3.6 added "f-strings": `blah = "blah"; eggs = 2; spam={blah} eggs={eggs}'`
- Strings in Python can be concatenated, by "adding" them (same operator as for adding integers and floats). E.g. `"spam" + "eggs"` returns `"spam eggs"`. Even if your strings contain numbers, they are still added as strings rather than integers. E.g. `"2" + "2"` returns `"2"`.
- Python has various kinds of string literals:
 - Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (`\`) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".
 - Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
 - Raw string varieties, denoted by prefixing the string literal with an `r`. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.
- Python has array index and array slicing expressions on lists, denoted as `a[Key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for

example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

- List comprehensions vs. for-loops
- Conditional expressions vs. if blocks
- The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a=1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c==1) {...}` is syntactically valid (but probably unintended) C code but `if c=1: ...` causes a syntax error in Python.

Methods

Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby). Apart from this Python also provides methods, sometimes called d-under methods functionality of custom class to support native functions such as `print`, `length`, `comparison`, support for arithmetic operations, type conversion, and many more.

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically-typed, Python is strongly-typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

The long-term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named mypy supports compile-time type checking.

CHAPTER 4

SYSTEM DESIGN

4.1 ER DIAGRAM

An Entity Relationship Diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts, or events within that system. An ERD is a data modeling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

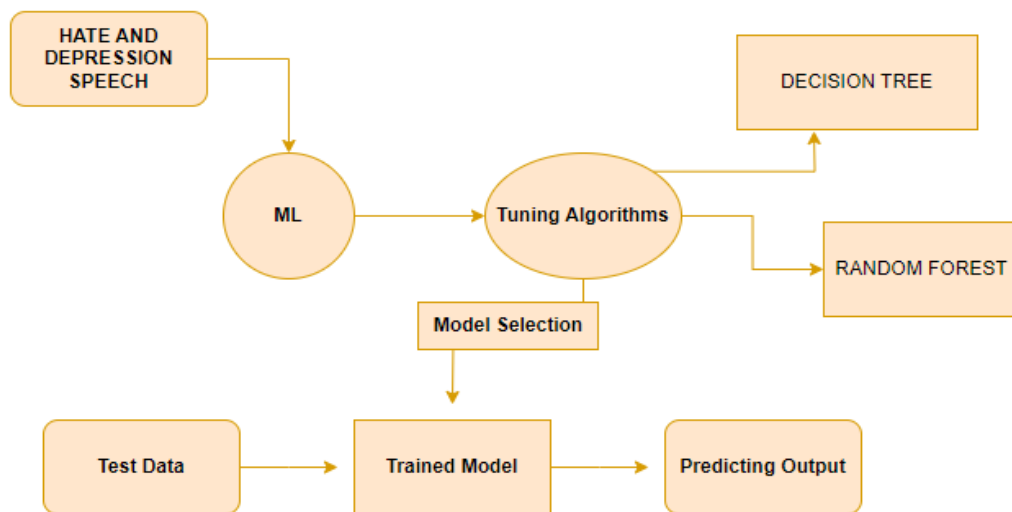


Fig 4.1 ER Diagram For Prediction Of Hate Speech And Depression Classification By Using ML Techniques.

4.2 WORK FLOW DIAGRAM

A Data-Flow Diagram is a way of representing a flow of data through a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. There are several notations for displaying data-flow diagrams. For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes. The dataflow diagram is part of the structured-analysis modelling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data-flow plan is a site-oriented data-flow plan.

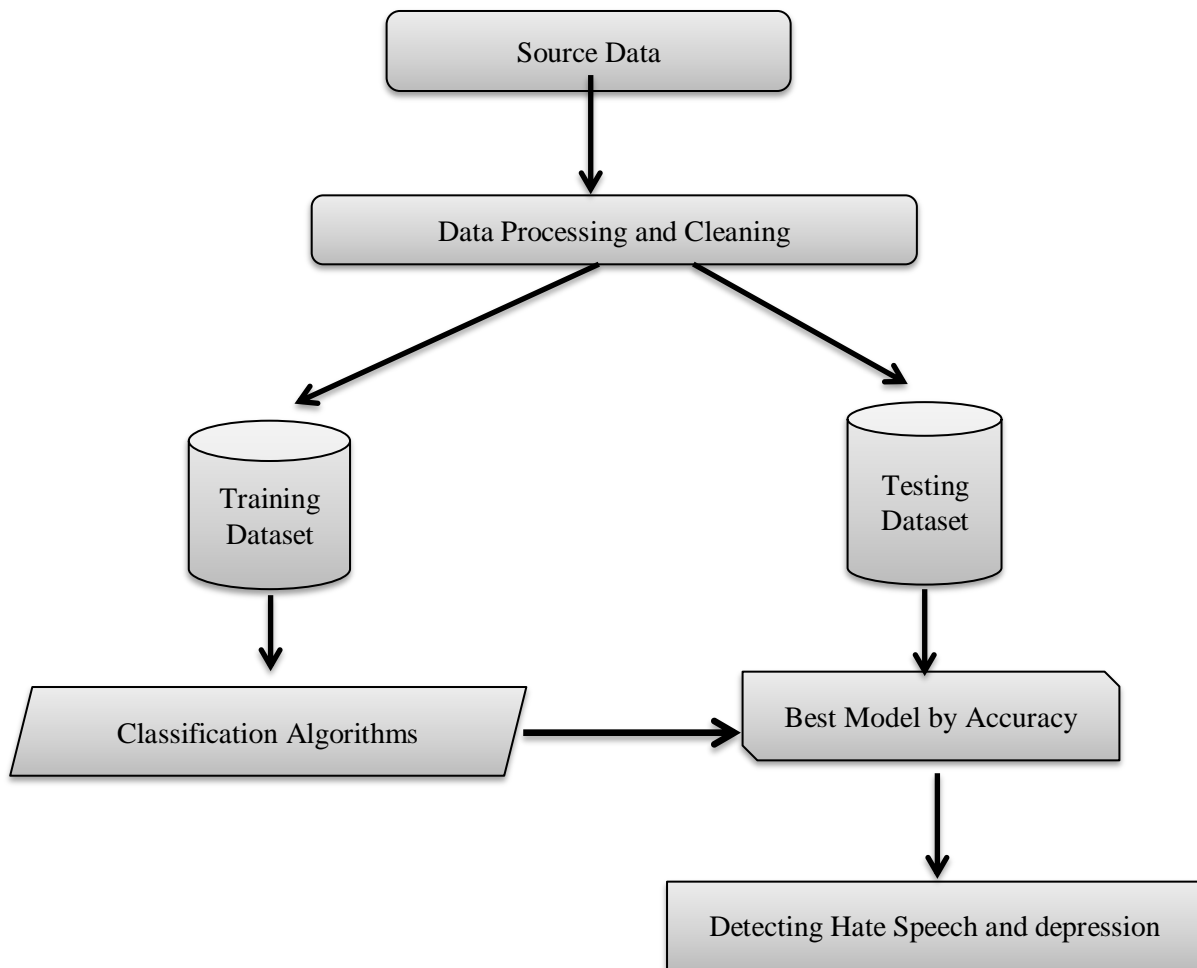


Fig 4.2 Work Flow Diagram For Prediction Of Hate Speech And Depression Classification By Using ML Techniques

4.3 USECASE DIAGRAM

Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

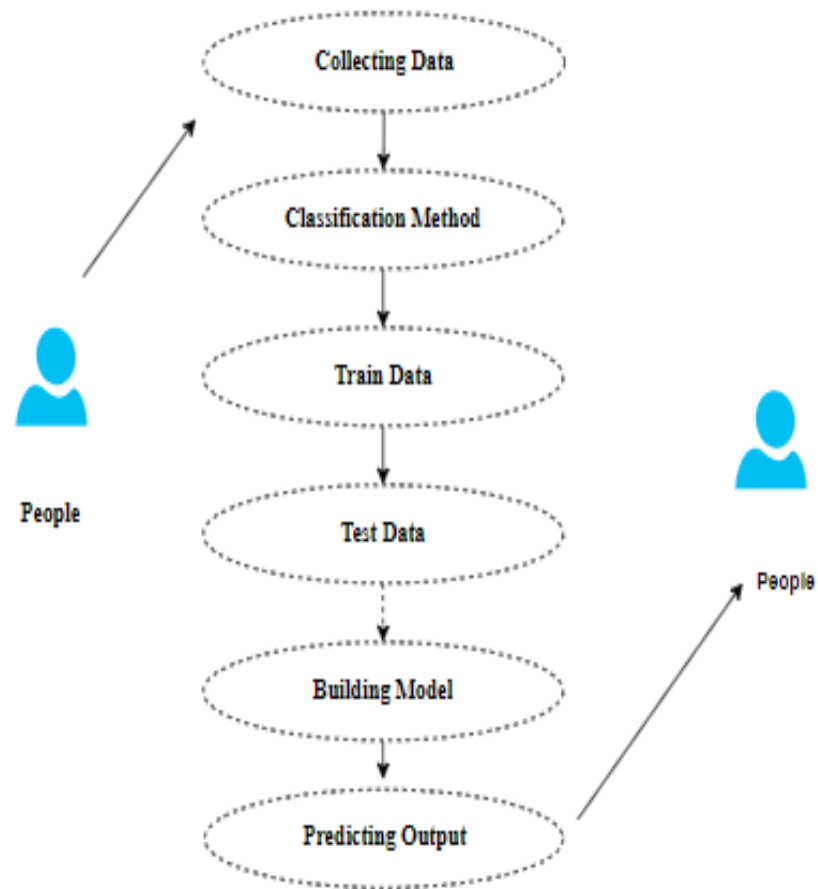


Fig 4.3 Use Case Diagram For Prediction Of Hate Speech and Depression Classification By Using ML Techniques.

4.4 CLASS DIAGRAM

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance. Responsibility (attributes and methods) of each class should be clearly identified for each class. Minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

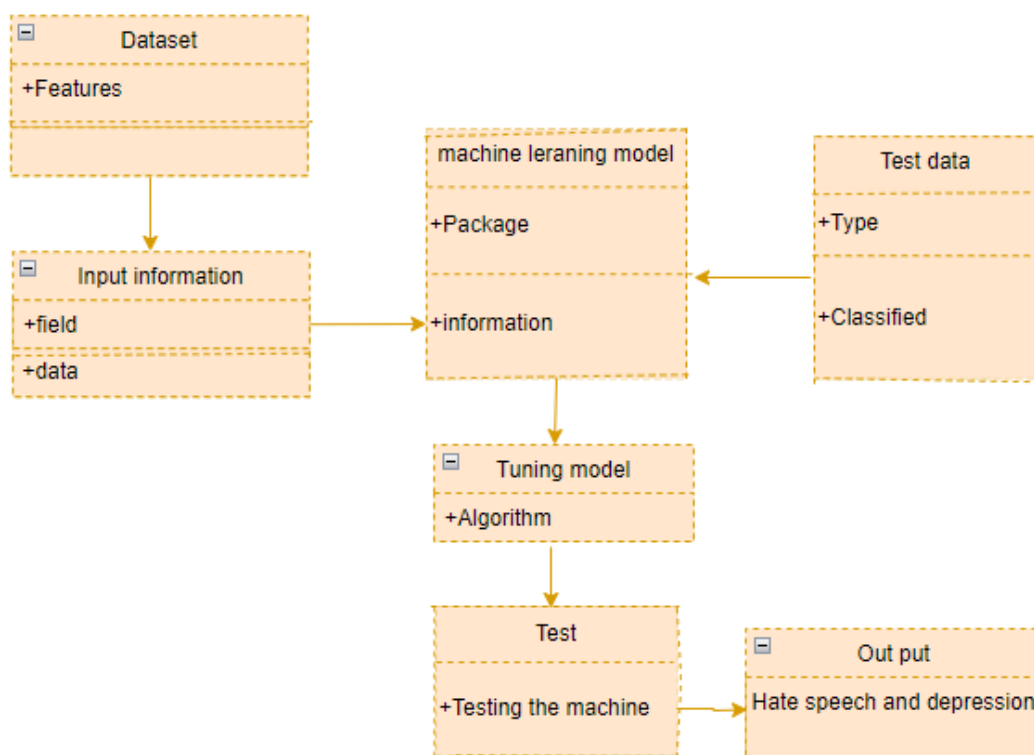


Fig 4.4 Class Diagram For Prediction Of Hate Speech And Depression Classification By Using ML Techniques

4.5 ACTIVITY DIAGRAM

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

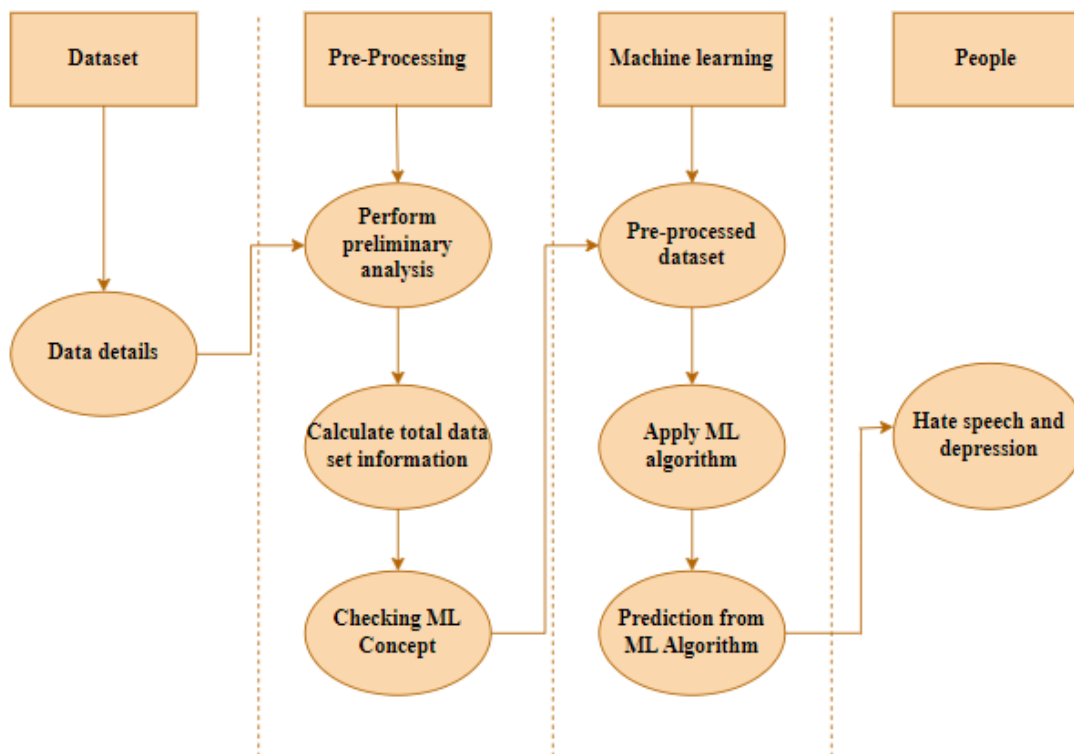


Fig 4.5 Activity Diagram For Prediction Of Hate Speech And Depression Classification By Using ML Techniques

4.6 SEQUENCE DIAGRAM

Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behavior within your system. Other dynamic modelling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

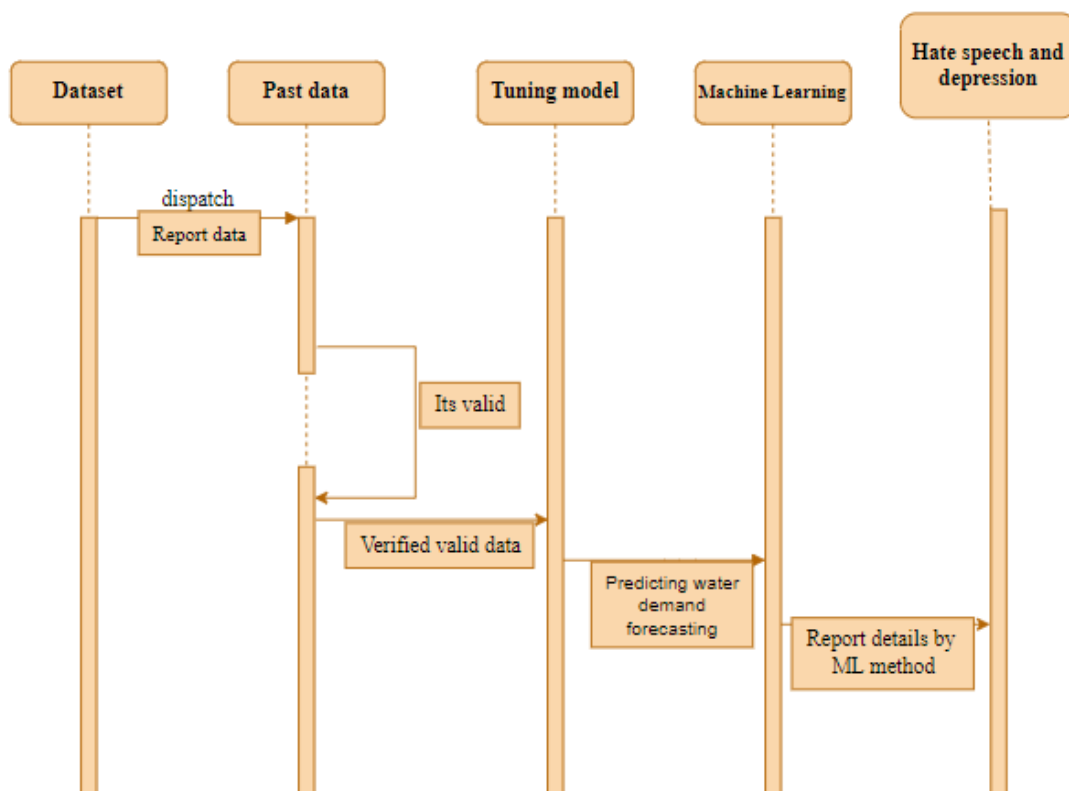


Fig 4.6 Sequence Diagram For Prediction Of Hate Speech And Depression Classification By Using ML Techniques

4.7 COLLABORATION DIAGRAM

A collaboration diagram is a type of visual presentation that shows how various software objects interact with each other within an overall IT architecture and how users (like doctor or patient) can benefit from this collaboration. A collaboration diagram often comes in the form of a visual chart that resembles a flow chart. It can show, briefly, how a single piece of software complements other parts of a greater system.

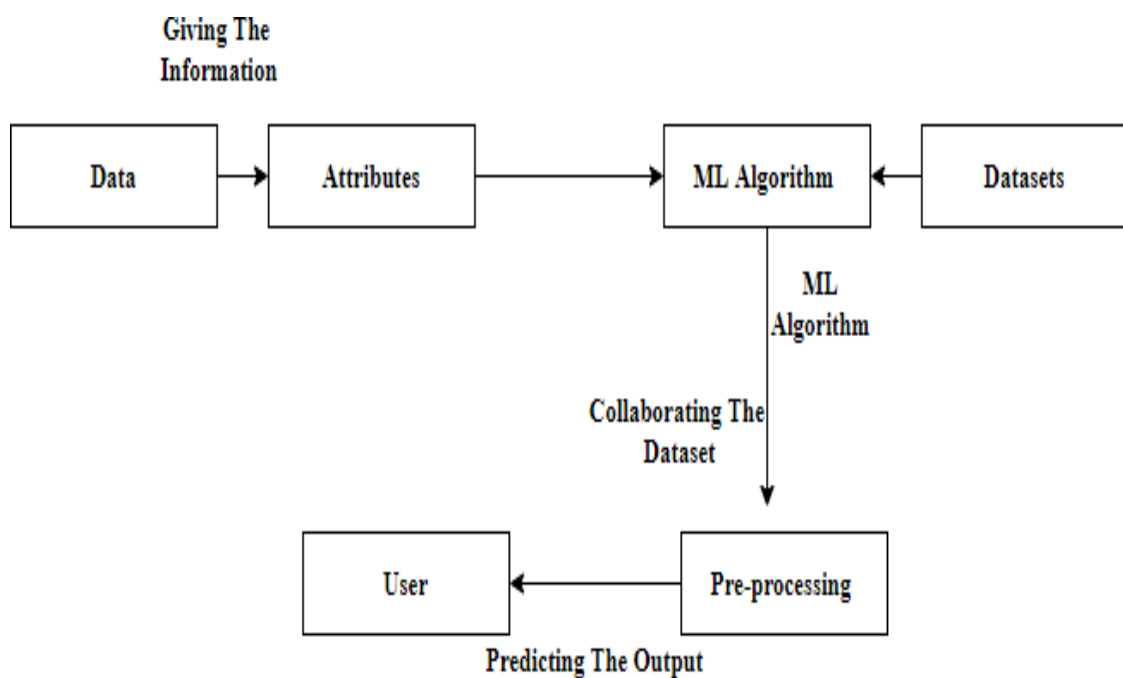


Fig 4.7 Collaboration Diagram For Prediction Of Hate Speech And Depression Classification By Using ML Techniques

CHAPTER 5

SYSTEM ARCHITECTURE

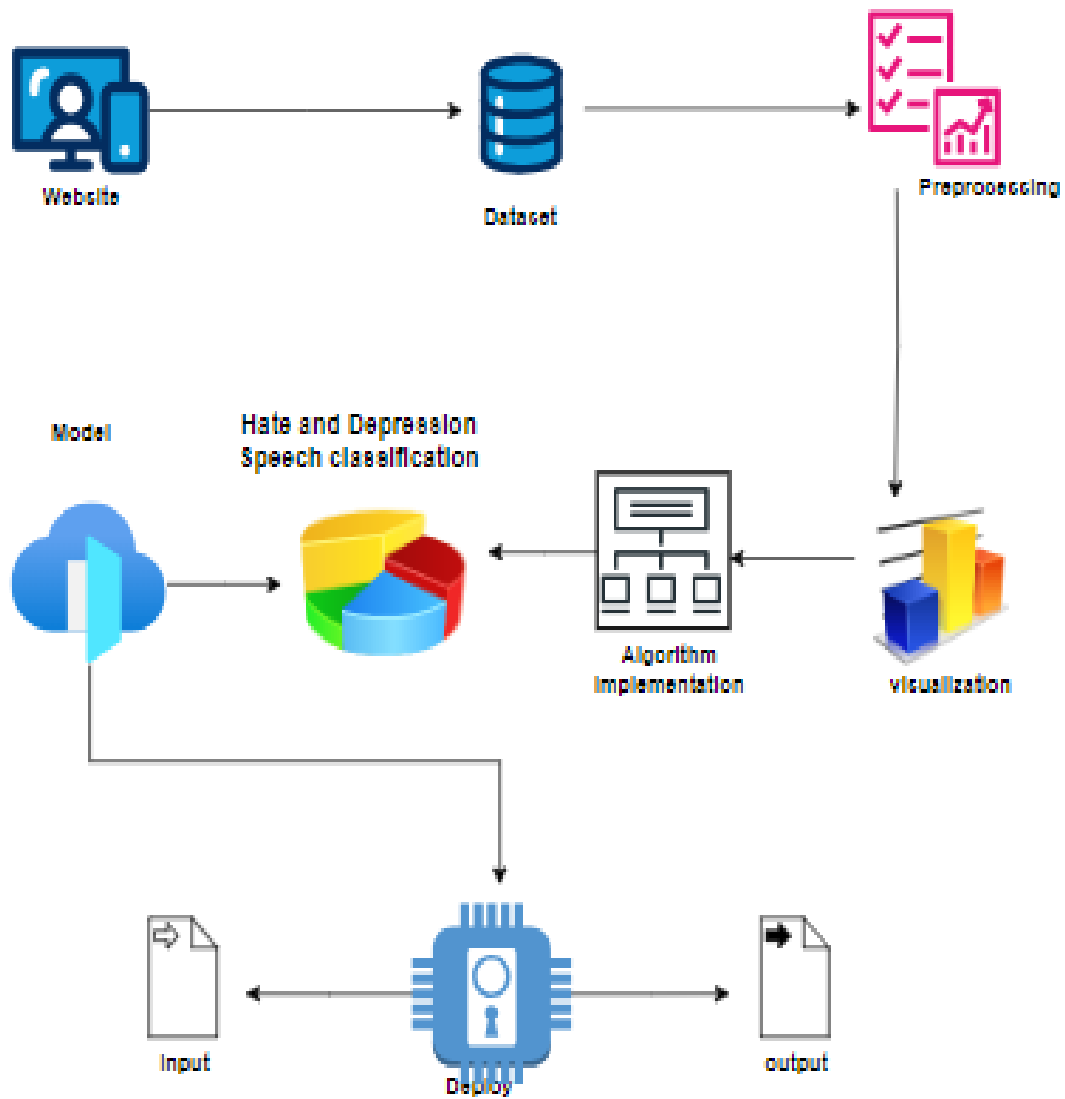


Fig 5.1 Architecture Diagram For Prediction Of Hate Speech And Depression Classification By Using ML Techniques

5.1 ALGORITHM AND TECHNIQUES

In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc. In Supervised Learning, algorithms learn from labelled data. After understanding the data, the algorithm determines which label should be given to new data based on pattern and associating the patterns to the unlabeled new data.

Used Python Packages:

Sk Learn:

- In python, sklearn is a machine learning package which include a lot of ML algorithms.
- Here, we are using some of its modules like `train_test_split`, `Decision Tree Classifier` or `Logistic Regression` and `accuracy_score`.

NumPy:

- It is a numeric python module which provides fast math functions for calculations.
- It is used to read data in numpy arrays and for manipulation purpose.

Pandas:

- Used to read and write different files.
- Data manipulation can be done easily with data frames.

Matplotlib:

- Data visualization is a useful way to help with identify the patterns from given dataset.
- Data manipulation can be done easily with data frames.

Comparing Algorithm with prediction in the form of best accuracy result

It is important to compare the performance of multiple different machine learning algorithms consistently and it will discover to create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. It can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare. Each model will have different performance characteristics. Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. It needs to be able to use these estimates to choose one or two best models from the suite of models that you have created. When have a new data set, it is a good idea to visualize the data using different techniques to look at the data from different perspectives. The same idea applies to model selection. You should use several different ways of looking at the estimated accuracy of your machine learning algorithms to choose the one or two to finalize. A way to do this is to use different visualization methods to show the average accuracy, variance, and other properties of the distribution of model accuracies.

In the next section you will discover exactly how you can do that in Python with scikit-learn. The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data and it can achieve this by forcing each algorithm to be evaluated on a consistent test harness.

In the example below 3 different algorithms are compared:

- BernoulinNB
- Random forest Classifier
- Ridge Classifier

The K-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithm is evaluated in precisely the same way. Before that comparing algorithm, building a Machine Learning Model using install Scikit-Learn libraries. In this library package must done preprocessing, linear model with logistic regression method, cross validating by KFold method, ensemble with random forest method and tree with decision tree classifier. Additionally, splitting the train set and test set. To predicting the result by comparing accuracy.

5.2 MODULE DESCRIPTION

5.2.1 Module 1 - Data Pre-processing

Validation techniques in machine learning are used to get the error rate of the Machine Learning (ML) model, which can be considered as close to the true error rate of the dataset. If the data volume is large enough to be representative of the population, you may not need the validation techniques. However, in real-world scenarios, to work with samples of data that may not be a true representative of the population of given dataset. To finding the missing value, duplicate value and description of data type whether it is float variable or integer. The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper parameters.

The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. It as machine learning engineers use this data to fine-tune the model hyper parameters. Data collection, data analysis, and the process of addressing data content, quality, and structure can add up to a time-consuming to-do list. During the process of data identification, it helps to understand your data and its properties; this knowledge will help you choose which algorithm to use to build your model.

A number of different data cleaning tasks using Python's Pandas library and specifically, it focus on probably the biggest data cleaning task, missing values and it able to more quickly clean data. It wants to spend less time cleaning data, and more time exploring and modeling.

Some of these sources are just simple random mistakes. Other times, there can be a deeper reason why data is missing. It's important to understand these different types of missing data from a statistics point of view. The type of missing data will influence how to deal with filling in the missing values and to detect missing values, and do some basic imputation and detailed statistical approach for dealing with missing data.

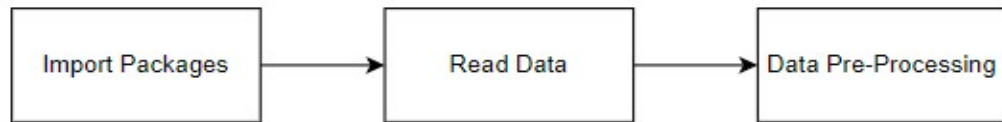
Before, joint into code, it's important to understand the sources of missing data. Here are some typical reasons why data is missing:

- User forgot to fill in a field.
- Data was lost while transferring manually from a legacy database.
- There was a programming error.
- Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.

Variable identification with Uni-variate, Bi-variate and Multi-variate analysis:

- Import libraries for access and functional purpose and read the given dataset
- General Properties of Analyzing the given dataset
- Display the given dataset in the form of data frame
- Show columns
- Shape of the data frame
- To describe the data frame
- Checking data type and information about dataset
- Checking for duplicate data
- Checking Missing values of data frame
- Checking unique values of data frame
- Checking count values of data frame
- Rename and drop the given data frame
- To specify the type of values
- To create extra columns

MODULE DIAGRAM



➤ GIVEN INPUT EXPECTED OUTPUT

- Input: data
- Output: removing noisy data

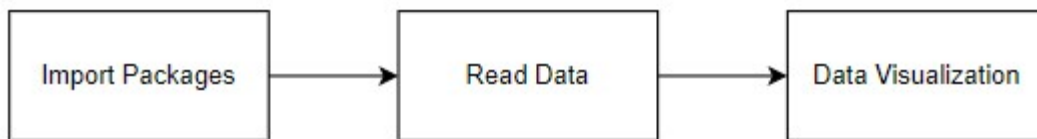
5.2.2 Module 2 - Data Visualization

Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding. This can be helpful when exploring and getting to know a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral and stakeholders than measures of association or significance. Data visualization and exploratory data analysis are whole fields themselves and it will recommend a deeper dive into some the books mentioned at the end.

Sometimes data does not make sense until it can look at in a visual form, such as with charts and plots. Being able to quickly visualize of data samples and others is an important skill both in applied statistics and in applied machine learning. It will discover the many types of plots that you will need to know when visualizing data in Python and how to use them to better understand your own data.

- How to chart time series data with line plots and categorical quantities with bar charts.
- How to summarize data distributions with histograms and box plots.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

Input: data

Output: visualized data

ALGORITHM IMPLEMENTATION

Certainly! In Natural Language Processing (NLP) for text classification, we typically start by splitting our dataset into a Training set and a Test set, usually in an 80:20 or 70:30 ratio. For traditional machine learning, we can use algorithms like Random Forest, Logistic Regression, Decision Trees, and Support Vector Classifier (SVC). We convert the text data into numerical features using techniques like TF-IDF. For neural network approaches, we use frameworks like TensorFlow and Keras. The text is tokenized and converted into sequences, and a neural network, often employing layers like Embedding and LSTM, is trained on the Training set. After training, we evaluate the model on the Test set to measure its accuracy and performance using metrics such as precision, recall, and F1 score. This process helps us determine how well our NLP model can predict and classify text data accurately.

In evaluating the performance of Natural Language Processing (NLP) models for text classification, several metrics are commonly employed to assess how well the model generalizes to unseen data. Key metrics include accuracy, precision, recall, and F1 score. Accuracy measures the overall correctness of predictions, while precision focuses on the proportion of true positives among all instances predicted as positive, assessing the model's ability to avoid false positives. Recall, on the other hand, calculates the proportion of true positives among all actual positive instances, indicating the model's ability to capture relevant cases without missing them. F1 score, which considers both precision and recall, provides a balanced assessment of a model's performance.

In practical terms, a high accuracy score does not necessarily guarantee a robust model, especially in imbalanced datasets where one class dominates. Precision and recall offer a more nuanced understanding of the model's strengths and weaknesses. For instance, in applications where false positives have serious consequences, prioritizing precision may be crucial, while in scenarios where false negatives are costly, recall becomes a priority.

When interpreting these metrics, it's essential to consider the specific goals and requirements of the NLP task at hand. Balancing precision and recall often involves a trade-off, and the choice of an appropriate metric depends on the specific context and consequences of false positives and false negatives in the given application.

The below 2 different algorithms are compared:

- DTC
- RFC

5.2.3 Decision Tree Algorithm

Decision Trees are a class of supervised machine learning algorithms used for both classification and regression tasks. They are versatile and can be applied to various domains, including finance, healthcare, and marketing. Decision Trees build models by recursively splitting the data based on feature conditions, creating a tree-like structure.

Key Points about Decision Trees

1. Decision-Making Process

- Decision Trees make decisions by asking a series of questions based on input features.
- Each internal node in the tree represents a decision based on a particular feature, and each branch represents an outcome.
- Leaf nodes represent the final decision or prediction.

2. Feature Selection

- At each node, the algorithm selects the feature that provides the best split, aiming to maximize information gain (for classification) or minimize variance (for regression).
- Popular criteria for measuring impurity or information gain include Gini impurity, entropy, or mean squared error.

3. Interpretability

- One of the significant advantages of Decision Trees is their interpretability. The resulting tree structure is human-readable and provides insights into the decision-making process.
- Decision Trees can be visualized, making it easy to understand how the model

arrived at a particular decision.

4. Handling Categorical and Numeric Data

- Decision Trees can handle both categorical and numeric data.
- For categorical features, the tree creates branches for each category.
- For numeric features, the tree selects optimal thresholds to split the data.

5. Overfitting

- Decision Trees are prone to overfitting, especially when the tree depth is too large.
- Techniques like pruning, setting a maximum depth, or using ensemble methods (e.g., Random Forests) can help mitigate overfitting.

6. Ensemble Methods

- Decision Trees can be combined into ensemble methods like Random Forests or Gradient Boosting, enhancing predictive performance and robustness.

7. Applications

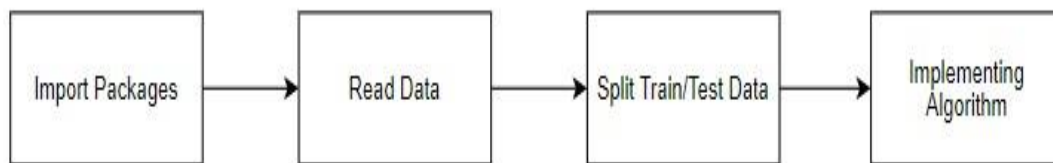
- Decision Trees are widely used in various domains, including finance for credit scoring, healthcare for diagnosis, and marketing for customer segmentation.
- Their simplicity and interpretability make them suitable for scenarios where understanding the decision-making process is crucial.

8. Limitations

- Decision Trees may struggle with capturing complex relationships in data, especially when the underlying patterns are not hierarchical.
- They are sensitive to small changes in the data, leading to different tree structures.
- Prone to overfitting, especially with deep trees.

In summary, Decision Trees provide a transparent and interpretable approach to decision-making, making them valuable for certain applications. However, addressing their limitations, such as overfitting, can be crucial for ensuring optimal performance.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

Input: data

Output: getting accuracy

5.2.4 Module 4 - Random forest Classifier

Random Forest is an ensemble learning method that leverages multiple decision trees to enhance predictive performance and reduce overfitting. It is widely used for both classification and regression tasks, offering robustness and improved generalization compared to individual decision trees.

Key Points about Random Forest:

1. Ensemble Learning

- Random Forest is an ensemble of decision trees, where each tree is trained on a random subset of the data and features.
- The final prediction is typically an average (regression) or majority vote (classification) of the individual tree predictions.

2. Decision Trees in Random Forest

- The decision trees in a Random Forest are constructed independently of each other.
- Each tree is trained on a different bootstrap sample (randomly sampled with replacement) from the original dataset.
- Additionally, at each split in a tree, only a random subset of features is considered.

3. Reducing Overfitting

- The ensemble nature of Random Forest helps mitigate overfitting, a common issue with individual decision trees.
- By combining predictions from multiple trees, the model becomes more robust and less sensitive to noise or outliers in the data.

4. Feature Importance

- Random Forest provides a measure of feature importance based on how much each feature contributes to the model's accuracy.
- This information can be valuable for feature selection and understanding the most influential factors in the data.

5. High Predictive Accuracy

- Random Forest often achieves high predictive accuracy, making it a popular choice for various applications.
- It can capture complex relationships in the data and handle both numerical and categorical features.

6. Scalability

- Random Forest is parallelizable, making it suitable for large datasets.
- The training process can be distributed across multiple processors or machines.

7. Applications

- Random Forest is used in diverse fields, including finance for fraud detection,

healthcare for disease diagnosis, and ecology for species classification.

- Its ability to handle high-dimensional data and provide robust predictions makes it applicable to various domains.

8. Hyper-parameter Tuning

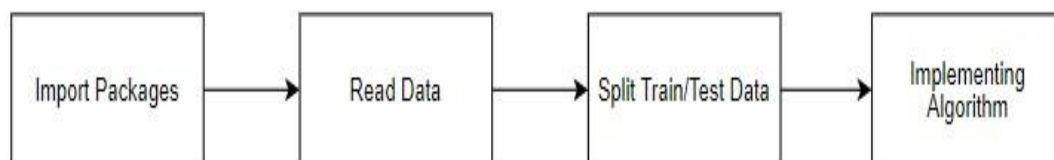
- Random Forest has hyper-parameters that can be fine-tuned for optimal performance, such as the number of trees, maximum depth of trees, and the size of the feature subset.

9. Limitations

- Although Random Forest is robust, it may not perform as well as more complex models on certain tasks.
- The interpretability of a Random Forest model is lower than that of individual decision trees.

In summary, Random Forest is a powerful and versatile machine learning algorithm that addresses some of the limitations of individual decision trees. It is known for its robustness, high predictive accuracy, and applicability to various domains. When facing complex tasks or large datasets, Random Forest can be an effective choice for building accurate and stable predictive models.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

Input: data

Output: getting accuracy

Deployment

Django (Web Framework)

Django is a micro web framework written in Python.

It is classified as a micro-framework because it does not require particular tools or libraries.

It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

However, Django supports extensions that can add application features as if they were implemented in Django itself.

Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Django was created by Armin Ronacher of Pocoo, an international group of Python enthusiasts formed in 2004. According to Ronacher, the idea was originally an April Fool's joke that was popular enough to make into a serious application. The name is a play on the earlier Bottle framework.

When Ronacher and Georg Brand created a bulletin board system written in Python, the Pocoo projects Werkzeug and Jinja were developed.

In April 2016, the Pocoo team was disbanded and development of Django and related libraries passed to the newly formed Pallets project.

Django has become popular among Python enthusiasts. As of October 2020, it has second most stars on GitHub among Python web-development frameworks, only slightly behind Django, and was voted the most popular web framework in the Python Developers Survey 2018.

The micro-framework Django is part of the Pallets Projects, and based on several others of them.

Django is based on Werkzeug, Jinja2 and inspired by Sinatra Ruby framework, available under BSD license. It was developed at Pocono by Armin Ronacher. Although Django is rather young compared to most Python frameworks, it holds a great promise and has already gained popularity among Python web developers. Let's take a closer look into Django, so-called "micro" framework for Python.

FEATURES

Django was designed to be easy to use and extend. The idea behind Django is to build a solid foundation for web applications of different complexity. From then on you are free to plug in any extensions you think you need. Also you are free to build your own modules. Django is great for all kinds of projects. It's especially good for prototyping. Django depends on two external libraries: the Jinja2 template engine and the Werkzeug WSGI toolkit.

Still the question remains why use Django as your web application framework if we have immensely powerful Django, Pyramid, and don't forget web mega-framework Turbo-gears? Those are supreme Python web frameworks BUT out-of-the-box Django is pretty impressive too with it's: Built-In Development server and Fast debugger

- Integrated support for unit testing
- RESTful request dispatching
- Uses Jinja2 Templating
- Support for secure cookies
- Unicode based
- Extensive Documentation
- Google App Engine Compatibility
- Extensions available to enhance features desired

Plus Django gives you so much more CONTROL on the development stage of your project. It follows the principles of minimalism and let you decide how you will build your application.

- Django has a lightweight and modular design, so it easy to transform it to the web framework you need with a few extensions without weighing it down
- ORM-agnostic: you can plug in your favorite ORM e.g. SQLAlchemy.
- Basic foundation API is nicely shaped and coherent.
- Django documentation is comprehensive, full of examples and well structured. You can even try out some sample application to really get a feel of Django.
- It is super easy to deploy Django in production (Django is 100% WSGI 1.0 compliant”)
- HTTP request handling functionality
- High Flexibility

The configuration is even more flexible than that of Django, giving you plenty of solution for every production need.

Overview of Python Django Framework Web apps are developed to generate content based on retrieved data that changes based on a user’s interaction with the site. The server is responsible for querying, retrieving, and updating data. This makes web applications to be slower and more complicated to deploy than static websites for simple applications.

Django is an excellent web development framework for REST API creation. It is built on top of Python which makes it powerful to use all the python features.

Django is used for the backend, but it makes use of a templating language called Jinja2 which is used to create HTML, XML or other markup formats that are returned to the user via an HTTP request.

Django is considered to be more popular because it provides many out of box

features and reduces time to build complex applications. Django is a good start if you are getting into web development. Django is a simple, un-opinionated framework; it doesn't decide what your application should look like developers do.

Django is a web framework. This means Django provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, and a wiki or go as big as a web-based calendar application or a commercial website.

Advantages of Django

- Higher compatibility with latest technologies.
- Technical experimentation.
- Easier to use for simple cases.
- Codebase size is relatively smaller.
- High scalability for simple applications.
- Easy to build a quick prototype.
- Routing URL is easy.
- Easy to develop and maintain applications.

Framework Django is a web framework from Python language. Django provides a library and a collection of codes that can be used to build websites, without the need to do everything from scratch. But Framework Django still doesn't use the Model View Controller (MVC) method.

Django-RESTful is an extension for Django that provides additional support for building REST APIs. You will never be disappointed with the time it takes to develop an API. Django-Restful is a lightweight abstraction that works with the existing ORM/libraries. Django-RESTful encourages best practices with minimal setup.

Django is a web framework for Python, meaning that it provides functionality for

building web applications, including managing HTTP requests and rendering templates and also we can add to this application to create our API.

Start Using an API

1. Most APIs require an API key. ...
2. The easiest way to start using an API is by finding an HTTP client online, like REST-Client, Postman, or Paw.
3. The next best way to pull data from an API is by building a URL from existing API documentation.

The Django object implements a WSGI application and acts as the central object. It is passed the name of the module or package of the application. Once it is created it will act as a central registry for the view functions, the URL rules, template configuration and much more.

Usually you create a Django instance in your main module or in the `__init__.py` file of your package.

Parameters

- **rule** (*str*) – The URL rule string.
- **endpoint** (*Optional[str]*) – The endpoint name to associate with the rule and view function. Used when routing and building URLs. Defaults to `view_func.__name__`.
- **view_func** (*Optional[Callable]*) – The view function to associate with the endpoint name.
- **provide_automatic_options** (*Optional[bool]*) – Add the `OPTIONS` method and respond to `OPTIONS` requests automatically.
- **options** (*Any*) – Extra options passed to the Rule object.

Return type -- None

After_Request(f)

Register a function to run after each request to this object.

The function is called with the response object, and must return a response object.

This allows the functions to modify or replace the response before it is sent.

If a function raises an exception, any remaining after request functions will not be called. Therefore, this should not be used for actions that must execute, such as to close resources. Use `teardown_request()` for that.

Parameters:

f (*Callable*[[*Response*], *Response*])

Return type

Callable[[*Response*], *Response*]

after_request_funcs: *t.Dict*[*AppOrBlueprintKey*,

t.List[*AfterRequestCallable*]]

A data structure of functions to call at the end of each request, in the format {scope: [functions]}. The scope key is the name of a blueprint the functions are active for, or None for all requests.

To register a function, use the `after_request()` decorator.

This data structure is internal. It should not be modified directly and its format may change at any time.

app_context()

Create an `AppContext`. Use as a with block to push the context, which will make `current_app` point at this application.

An application context is automatically pushed by `RequestContext.push()` when handling a request, and when running a CLI command. Use this to manually create a context outside of these situations.

With `app.app_context()`:

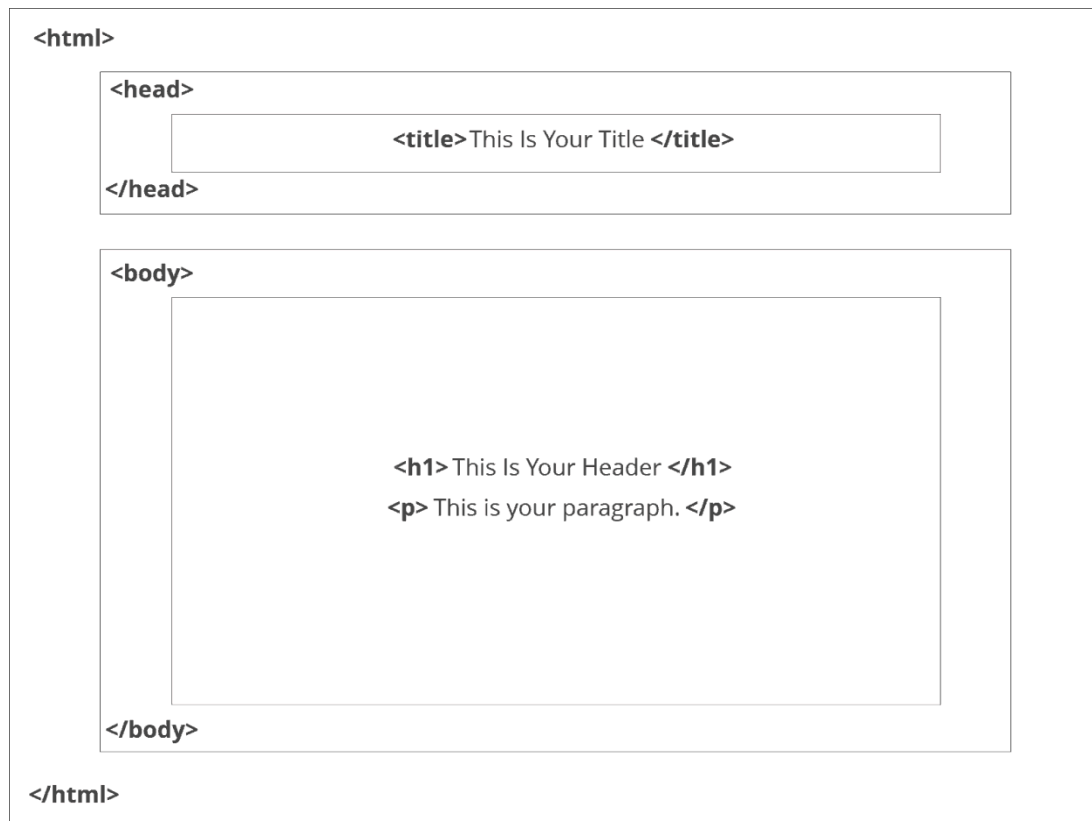
`Init_db()`

HTML INTRODUCTION

HTML stands for Hyper Text Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

Basic Construction of an HTML Page

These tags should be placed underneath each other **at the top of every HTML page** that you create.



`<!DOCTYPE html>` — This tag **specifies the language** you will write on the page. In this case, the language is HTML 5.

`<html>` — This tag signals that from here on we are going to write in HTML code.

`<head>` — This is where all the **metadata for the page** goes — stuff mostly meant for search engines and other computer programs.

`<body>` — This is where the **content of the page** goes.

Further Tags

Inside the `<head>` tag, there is one tag that is always included: `<title>`, but there are others that are just as important:

`<title>`

This is where we **insert the page name** as it will appear at the top of the browser

window or tab.

<meta>

This is where information *about* the document is stored: character encoding, name (page context), description.

HeadTag

<head>

<title>My First Webpage</title>

<meta charset="UTF-8">

<meta name="description" content="This field contains information about your page. It is usually around two sentences long.">.

<meta name="author" content="Conor Sheils">

</header>

Adding Content

Next, we will make<body> tag.

The HTML <body> is where we add the content which is designed for viewing by human eyes.

This includes **text**, **images**, **tables**, **forms** and everything else that we see on the internet each day.

Add HTML Headings To Web Page

In HTML, [headings](#) are written in the following elements:

- <h1>
- <h2>
- <h3>
- <h4>

- <h5>
- <h6>

As you might have guessed <h1> and <h2> should be used for the most important titles, while the remaining tags should be used for sub-headings and less important text.

Search engine bots use this order when deciphering which information is most important on a page.

Creating Your Heading

Let's try it out. On a new line in the HTML editor, type:

```
<h1> Welcome To My Page </h1>
```

And hit save. We will save this file as “index.html” in a new folder called “my webpage.”

Add Links In HTML

As you may have noticed, the internet is made up of lots of links.

Almost everything you click on while surfing the web is a link **takes you to another page** within the website you are visiting or to an external site.

Links are included in an attribute opened by the <a> tag. This element is the first that we've met which uses an attribute and so it **looks different to previously mentioned tags**.

```
<a href=http://www.google.com>Google</a>
```

CASCADING STYLE SHEETS

CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colours, layout, and fonts, thus making our web pages presentable to the users. CSS is designed to make style sheets for the

web. It is independent of HTML and can be used with any XML-based markup language. Now let's try to break the acronym:

- Cascading: Falling of Styles
- Style: Adding designs/Styling our HTML tags
- Sheets: Writing our style in different documents

CSS Syntax

```
Selector {  
Property 1 : value;  
Property 2 : value;  
Property 3 : value;  
}
```

For example:

```
h1  
{  
Color: red;  
Text-align: center;  
}  
#unique  
{  
color: green;  
}
```

- Selector: selects the element you want to target
- Always remains the same whether we apply internal or external styling
- There are few basic selectors like tags, id's, and classes
- All forms this key-value pair
- Keys: properties(attributes) like color, font-size, background, width, height, etc

- Value: values associated with these properties

CSS Comment

- Comments don't render on the browser
- Helps to understand our code better and makes it readable.
- Helps to debug our code
- Two ways to comment:
 - Single line

Inline CSS

- Before CSS this was the only way to apply styles
- Not an efficient way to write as it has a lot of redundancy
- Self-contained
- Uniquely applied on each element
- The idea of separation of concerns was lost
- Example:

`<h3 style = "color: red"> Have a great day </h3>`

`<p style = "color: green"> I did this, I did that </p>`

Internal CSS

- With the help of style tag, we can apply styles within the HTML file
- Redundancy is removed
- But the idea of separation of concerns still lost
- Uniquely applied on a single document
- Example:

`<style>`

```
H1{
```

```
Color: red;
```

```
}
```

```
</style>
```

```
<h3> Have a great day </h3>
```

External CSS

- With the help of <link> tag in the head tag, we can apply styles
- Reference is added
- File saved with .css extension
- Redundancy is removed
- The idea of separation of concerns is maintained
- Uniquely applied to each document

Example:

```
<head>
```

```
<link rel= "stylesheet" type= "text/css" href= "name of the CSS file">
```

```
</head>
```

```
h1{
```

```
color: red;  //.css file
```

```
}
```

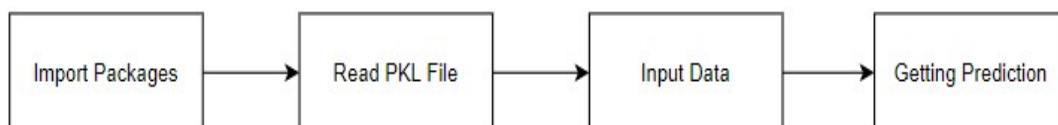
CSS Box Model

- Every element in CSS can be represented using the BOX model
- It allows us to add a border and define space between the content
- It helps the developer to develop and manipulate the elements
- It consists of 4 edges
 - Content edge – It comprises of the actual content
 - Padding edge – It lies in between content and border edge
 - Border edge – Padding is followed by the border edge
 - Margin edge – It is an outside border and controls the margin of the element

Deploying the model predicting output

In this module the trained machine learning model is converted into pickle data format file (.pkl file) which is then deployed for providing better user interface and predicting the hate and depression of a person.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

Input: data values

Output: predicting output

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 SERVER SIDE

Module – 1

```
import pandas as pd
import numpy as np
Data = pd.read_csv('DEPRESSION.csv')
Data.head()
Data.tail()
Data.shape
Data = Data.dropna()
Data.shape
Data.size
Data.isnull().sum()
Data.info()
Data.columns
Data['LABEL'].unique()
Data['LABEL'].value_counts()
Data.groupby('LABEL').describe()
##### BEFORE LABEL ENCODER
Data.head()
from sklearn.preprocessing import LabelEncoder
var_mod = ['TEXT','LABEL']
le = LabelEncoder()
for i in var_mod:
    Data[i] = le.fit_transform(Data[i]).astype(int)
```

```
#### AFTER LABEL ENCODER
```

```
Data.head()
```

```
Data.duplicated()
```

```
Data.duplicated().sum()
```

```
Data = Data.drop_duplicates()
```

```
Data.duplicated().sum()
```

Module - 2

DATA VISUALIZATION AND DATA ANALYSIS

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
Data = pd.read_csv('DEPRESSION.csv')
```

```
Data.head()
```

```
Data.tail()
```

```
sns.countplot(x='LABEL',data=Data)
```

```
plt.hist(Data['LABEL'],color='green')
```

```
Data['LABEL'].plot(kind='density')
```

```
sns.displot(Data['LABEL'], color='purple')
```

```
sns.violinplot(Data['LABEL'], color='yellow')
```

```
sns.ecdfplot(Data['LABEL'], color='blue')
```

```
sns.distplot(Data['LABEL'], color='RED')
```

```
import re
```

```
from nltk.corpus import stopwords
```

```
from nltk.stem import PorterStemmer
```

```
from nltk.tokenize import word_tokenize
```

```
# Define preprocess function for text preprocessing
```

```
def preprocess_text(text)
```

```
# Check for NaN values and handle them
```

```

if pd.isnull(text)
return ""

# Convert to lowercase
text = text.lower()

# Remove special characters and digit
text = re.sub(r'^a-zA-Z\s|', "", text)

# Tokenization and remove stop word
stop_words = set(stopwords.words('english'))
words = [word for word in word_tokenize(text) if word not in stop_words]

# Stemming
ps = PorterStemmer()
words = [ps.stem(word) for word in words]

# Join the preprocessed words back into a single string
preprocessed_text = ' '.join(words)

return preprocessed_text

Data['TEXT'] = Data['TEXT'].apply(preprocess_text)

from sklearn.model_selection import train_test_split
X,X_test,y,y_test = train_test_split(Data.loc[:, 'TEXT'],Data['LABEL'],test_size=0.2)

from wordcloud import WordCloud
import matplotlib.pyplot as plt

ham=' '.join(X.loc[y==0, 'TEXT'].values)

ham_text = WordCloud(background_color='white',max_words=2000,width =
800, height = 800).generate(ham)

plt.figure(figsize=[10,30])

plt.imshow(ham_text,interpolation='bilinear')

plt.title('NOT IN DEPRESSION')

plt.axis('off')

spam=' '.join(X.loc[y==1, 'TEXT'].values)

spam_text = WordCloud(background_color='RED',max_words=2000,width =
800, height = 800).generate(spam)

```

```
plt.figure(figsize=[10,30])
plt.imshow(spam_text, interpolation='bilinear')
plt.axis('off')
plt.title('DEPRESSION CONDITION')
```

Module-3:

DECISION TREE CLASSIFIER ALGORITHM

```
import pandas as pd
import numpy as np
# Step 1: Load the CSV dataset
Data = pd.read_csv('DEPRESSION.csv')
Data.head()
Data.tail()
Data['LABEL'].value_counts()
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
# Define preprocess function for text preprocessing
def preprocess_text(text):
    # Check for NaN values and handle them
    if pd.isnull(text):
        return ""
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and digits
    text = re.sub(r'^a-zA-Z\s', "", text)
    # Tokenization and remove stop words
    stop_words = set(stopwords.words('english'))
    words = [word for word in word_tokenize(text) if word not in stop_words]

    # Stemming
```

```

ps = PorterStemmer()
words = [ps.stem(word) for word in words]
# Join the preprocessed words back into a single string
preprocessed_text = ''.join(words)
return preprocessed_text

# Step 2: Data Preprocessing

Data['TEXT'] = Data['TEXT'].apply(preprocess_text)

# Step 3: Feature Extraction (TF-IDF)

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()

x1 = tfidf_vectorizer.fit_transform(Data['TEXT'])

# Assuming you have a column named 'label' containing the target labels
y1 = Data['LABEL']

# import imblearn

# from imblearn.over_sampling import RandomOverSampler

# from collections import Counter

# ros =RandomOverSampler(random_state=42)

# x,y=ros.fit_resample(x1,y1)

# print("OUR DATASET COUNT      : ", Counter(y1))

# print("OVER SAMPLING DATA COUNT : ", Counter(y))

# Step 5: Splitting Data

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x1, y1, test_size=0.2,

```



```

random_state=42)

from sklearn.tree import DecisionTreeClassifier

# Step 6: Machine Learning Model (Naive Bayes)

DTC = DecisionTreeClassifier()

# Step 7: Training the Model

DTC.fit(x_train, y_train)

# Step 8: Evaluation

predicted = DTC.predict(x_test)

from sklearn.metrics import accuracy_score
AC = accuracy_score(y_test,predicted)
print("THE ACCURACY SCORE OF DECISION TREE CLASSIFIER
IS :",AC*100)

from sklearn.metrics import hamming_loss
HL = hamming_loss(y_test,predicted)
print("THE HAMMING LOSS OF DECISION TREE CLASSIFIER
IS :",HL*100)

from sklearn.metrics import precision_score
PR = precision_score(y_test,predicted)
print('THE PRECISION SCORE OF DECISION TREE
CLASSIFIER:\n\n',PR*100)

from sklearn.metrics import recall_score
RE = recall_score(y_test,predicted)
print('THE RECALL SCORE OF DECISION TREE
CLASSIFIER:\n\n',RE*100)

from sklearn.metrics import f1_score
F1 = f1_score(y_test,predicted)

```

```

print('THE F1 SCORE OF DECISION TREE CLASSIFIER:\n\n',F1*100)
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test,predicted)
print('THE CONFUSION MATRIX SCORE OF DECISION TREE
CLASSIFIER:\n\n\n',CM)
import matplotlib.pyplot as plt
cm=confusion_matrix(y_test, predicted)
print('THE CONFUSION MATRIX SCORE OF DECISION TREE
CLASSIFIER:\n\n')
print(cm)
print("\n\nDISPLAY CONFUSION MATRIX OF DECISION TREE CLASSIFIER:
\n\n")
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, predicted, labels=DTC.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=DTC.classes_)
disp.plot()
plt.show()
def graph():
import matplotlib.pyplot as plt
data=[AC]
alg=" DECISION TREE CLASSIFIER "
plt.figure(figsize=(5,5))
b=plt.bar(alg,data,color=("YELLOW"))
plt.title("THE ACCURACY SCORE OF DECISION TREE CLASSIFIER IS\n\n\n")
plt.legend(b,data,fontsize=9)
graph()
import joblib
joblib.dump(DTC, 'MODEL1.pkl')
tfidf_vectorizer.fit_transform(Data['TEXT'])
joblib.dump(tfidf_vectorizer, 'TF1.pkl')

```

Module-4

RANDOM FOREST CLASSIFIER ALGORITHM

```
import pandas as pd
import numpy as np
# Step 1: Load the CSV dataset
Data = pd.read_csv('HATE.csv',usecols=['TEXT','LABEL'])
Data.head()
Data = Data.dropna()
Data.tail()
Data['LABEL'].value_counts()
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
# Define preprocess function for text preprocessing
def preprocess_text(text):
# Check for NaN values and handle them
    if pd.isnull(text):
        return ""
# Convert to lowercase
    text = text.lower()
# Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', "", text)

# Tokenization and remove stop words
    stop_words = set(stopwords.words('english'))
    words = [word for word in word_tokenize(text) if word not in stop_words]
# Stemming
    ps = PorterStemmer()
    words = [ps.stem(word) for word in words]
```

```

# Join the preprocessed words back into a single string
preprocessed_text = ''.join(words)
return preprocessed_text

# Step 2: Data Preprocessing
Data['TEXT'] = Data['TEXT'].apply(preprocess_text)

# Step 3: Feature Extraction (TF-IDF)
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
x1 = tfidf_vectorizer.fit_transform(Data['TEXT'])

# Assuming you have a column named 'label' containing the target labels
y1 = Data['LABEL']

# import imblearn
# from imblearn.over_sampling import RandomOverSampler
# from collections import Counter
# ros =RandomOverSampler()
# x,y=ros.fit_resample(x1,y1)
# print("OUR DATASET COUNT      : ", Counter(y1))
# print("OVER SAMPLING DATA COUNT : ", Counter(y))

# Step 5: Splitting Data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x1, y1, test_size=0.1,
random_state=42, stratify=y1)
from sklearn.ensemble import RandomForestClassifier

# Step 6: Machine Learning Model (Naive Bayes)
RFC = RandomForestClassifier()
RFC.fit(x_train, y_train)
predictedRFC = RFC.predict(x_test)
from sklearn.metrics import accuracy_score
AC = accuracy_score(y_test,predictedRFC)

```

```

print("THE ACCURACY SCORE OF RANDOM FOREST CLASSIFIER IS
:",AC*100)

from sklearn.metrics import hamming_loss
HL = hamming_loss(y_test,predictedRFC)
print("THE HAMMING LOSS OF RANDOM FOREST CLASSIFIER
IS :",HL*100)

from sklearn.metrics import classification_report
CL = classification_report(y_test,predictedRFC)
print("THE CLASSIFICATION REPORT OF RANDOM FOREST
CLASSIFIER:\n\n',CL)

from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test,predictedRFC)
print("THE CONFUSION MATRIX SCORE OF RANDOM FOREST
CLASSIFIER:\n\n\n',CM)

import matplotlib.pyplot as plt
cm=confusion_matrix(y_test, predictedRFC)
print("THE CONFUSION MATRIX SCORE OF RANDOM FOREST
CLASSIFIER:\n\n')
print(cm)
print("\n\nDISPLAY CONFUSION MATRIX OF RANDOM FOREST
CLASSIFIER: \n\n")

from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, predictedRFC, labels=RFC.classes_)
disp =ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=RFC.classes_)
disp.plot()
plt.show()
def graph():
import matplotlib.pyplot as plt
data=[AC]
alg="RANDOM FOREST CLASSIFIER"

```

```
plt.figure(figsize=(5,5))
b=plt.bar(alg,data,color=("VIOLET"))
plt.title("THE ACCURACY SCORE OF RANDOM FOREST CLASSIFIER
IS\n\n\n")
plt.legend(b,data,fontsize=9)
graph()
import joblib
joblib.dump(RFC, 'MODEL2.pkl')
tfidf_vectorizer.fit_transform(Data['TEXT'])
joblib.dump(tfidf_vectorizer, 'TF2.pkl')
```

CHAPTER 7

TESTING

7.1 Introduction

The main objective of testing is to uncover errors from the system. For the uncovering process we have to give proper input data to the system. So we should have more conscious to give input data. It is important to give correct inputs to efficient testing.

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspes conditions. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works.

Inadequate testing or non-testing leads to errors that may appear few months later.

TYPES OF TESTING

Unit testing verification efforts on the smallest unit of software design, module. This is known as “Module Testing”. The modules are tested separately. This testingis carried out during programming stage itself. In these testing steps, each module is found to be working satisfactorily as regard to the expected output from the module.

BLACK BOX TESTING

Black box testing, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

WHITE-BOX TESTING

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing).

GREY BOX TESTING

Grey box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. To test the Web Services application usually the Grey box testing is used. Grey box testing is performed by end-users and also by testers and developers.

INTEGRATION TESTING

Integration testing is a systematic technique for constructing tests to uncover error associated within the interface. In the project, all the modules are combined and then the entire programmer is tested as a whole. In the integration-testing step, all the error uncovered is corrected for the next testing steps.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

ACCEPTANCE TESTING

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

PERFORMANCE AND ANALYSIS

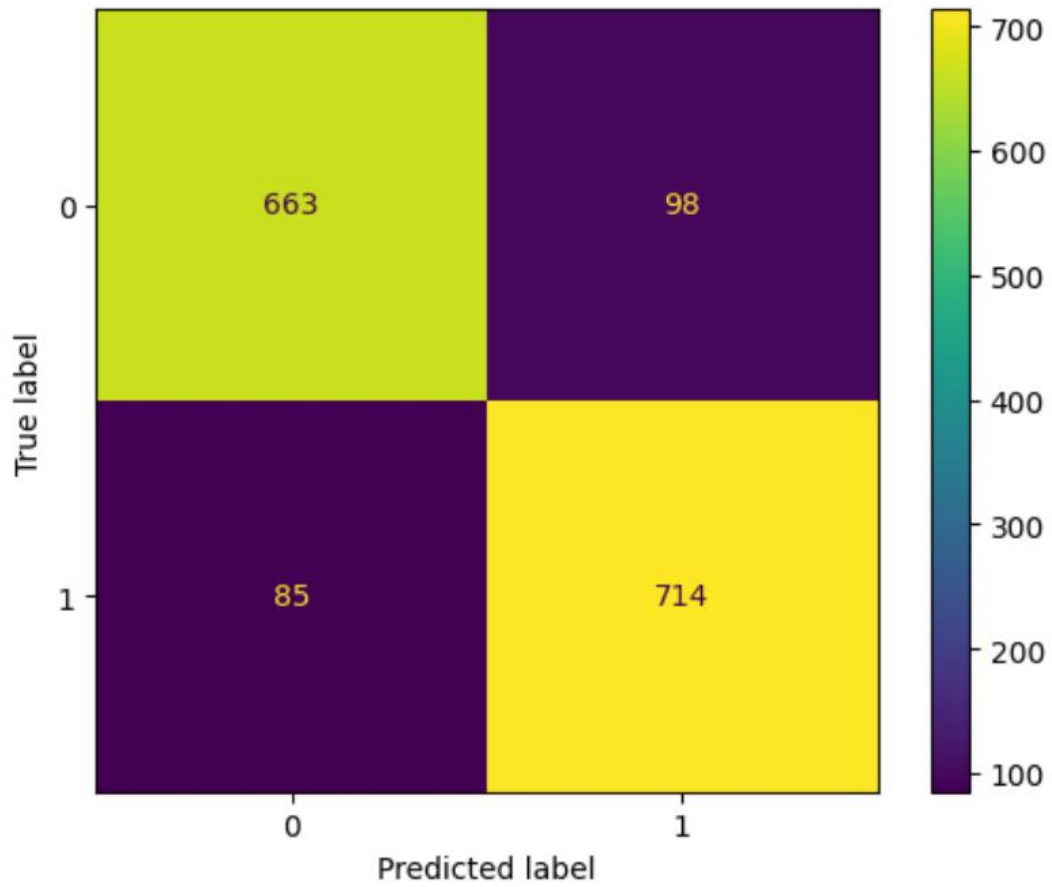


Fig 7.1 Confusion Matrix For Decision Tree Classifier Algorithm

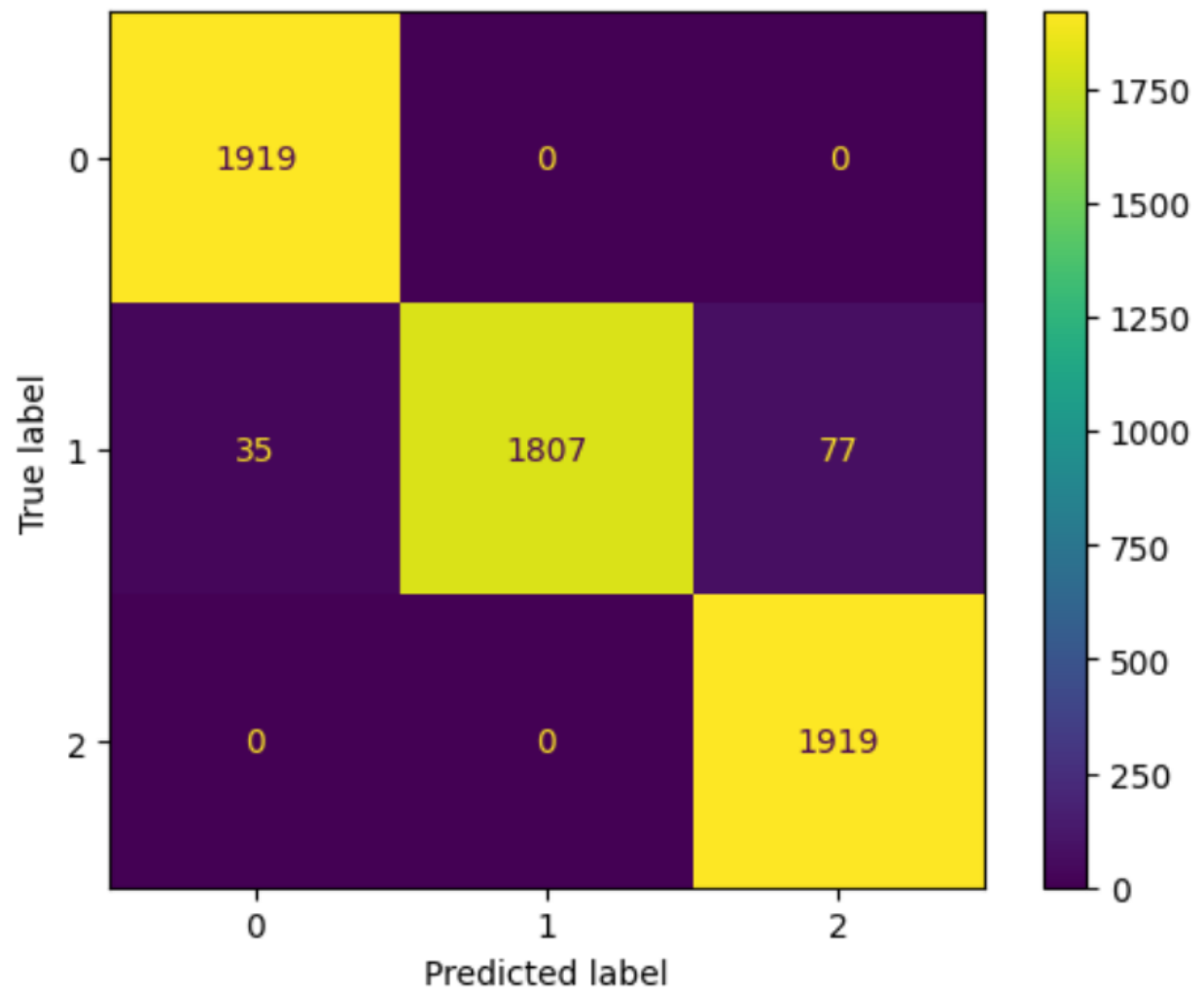


Fig 7.2 Confusion Matrix For Random Forest Algorithm

CHAPTER 8

CONCLUSION

8.1 RESULTS AND DISCUSSIONS

The analytical process started from data cleaning and processing, missing values and finally model building and evaluation. The best accuracy on public test is high accuracy score will be find out. This application can help to find the prediction of any hate speech and depression context.

8.2 CONCLUSION AND FUTURE ENHANCEMENTS

Leveraging Natural Language Processing (NLP) for hate speech and depression detection holds immense promise in creating safer online environments and fostering mental health support. By employing advanced linguistic analysis and machine learning models, these applications contribute to the prevention of online toxicity, cyberbullying, and discrimination. Simultaneously, the early identification of individuals expressing signs of depression allows for timely intervention and the provision of mental health resources. However, the implementation of such technology necessitates careful consideration of ethical concerns, user privacy, cultural sensitivities, and the ongoing refinement of models to keep pace with evolving language nuances. The continuous pursuit of technological advancements and interdisciplinary collaboration is crucial for maximizing the positive impact of

FUTURE WORK

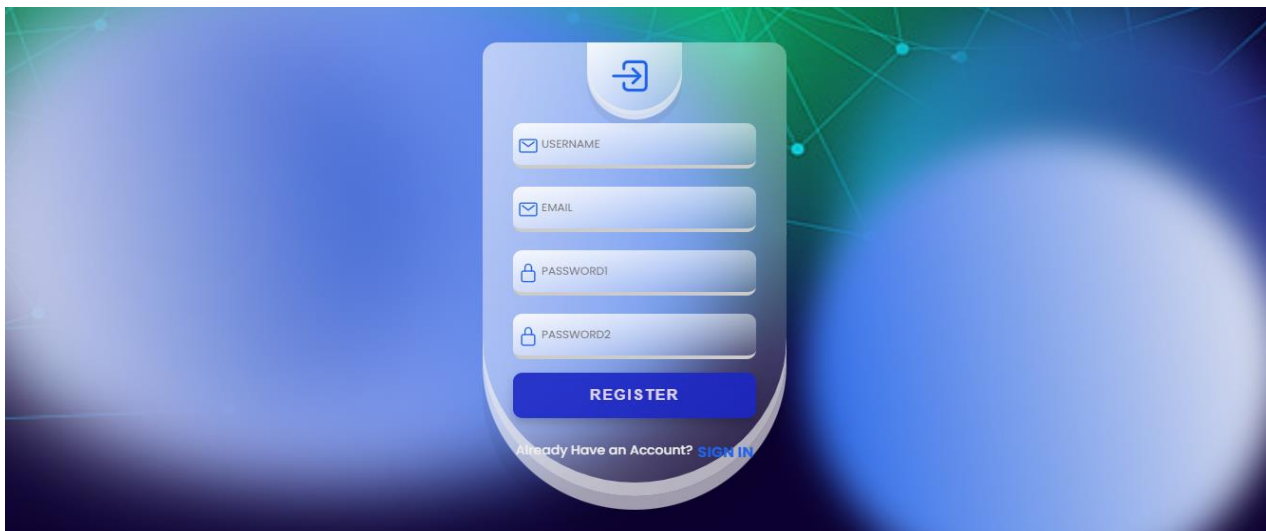
- ❖ Future work involves integrating multimodal data for comprehensive hate speech and depression detection, combining textual, visual, and audio cues
- ❖ Further research should focus on fine-tuning NLP models for cultural and linguistic diversity, ensuring accuracy across diverse online contexts

APPENDICES

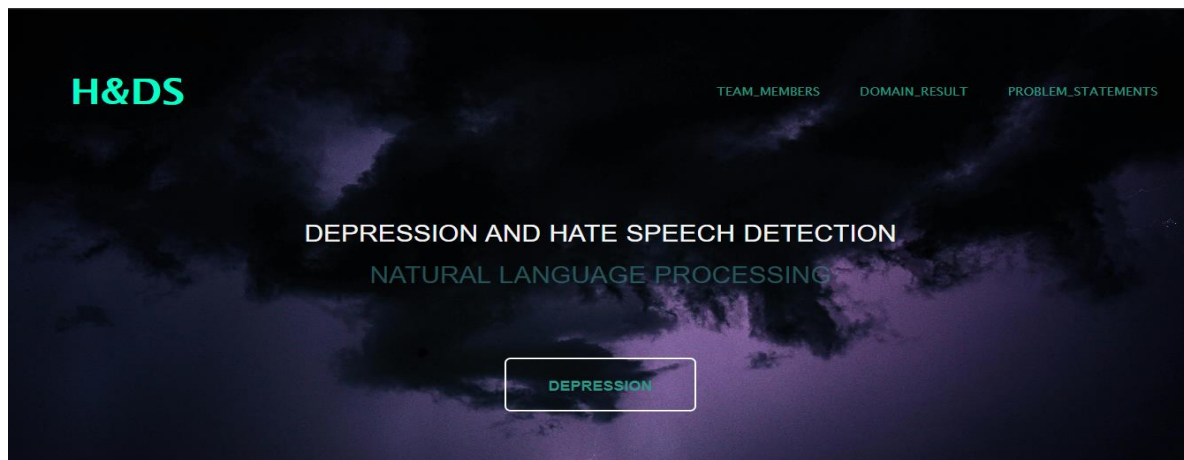
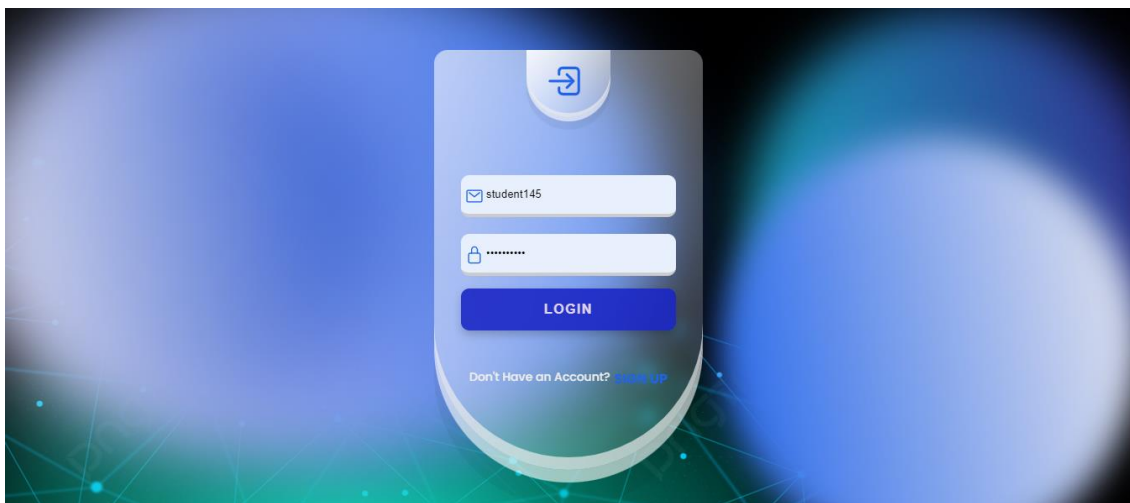
A.1 SAMPLE SCREENS

Hate and Depression Speech Classification

[Explore](#)



A registration form with a blue and green background. The form is contained within a rounded rectangle with a blue border. It features a blue icon of a document with a checkmark at the top. Below the icon are four input fields: USERNAME, EMAIL, PASSWORD1, and PASSWORD2. Each field has a small icon on the left: an envelope for USERNAME and EMAIL, and a padlock for PASSWORD1 and PASSWORD2. Below the input fields is a blue button labeled REGISTER. At the bottom of the form, there is a link: Already Have an Account? [SIGN IN](#).



BASEPAPER DEMERITS :

- They focused on only Depression.
- They did not compared more than an algorithms to getting better accuracy level.
- Accuracy was low.
- More time complex to do that.

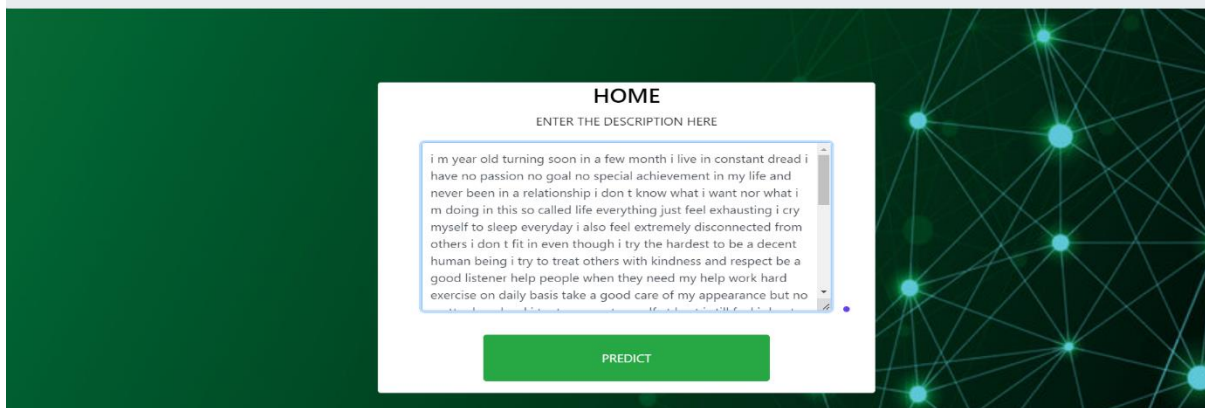
PROPOSED SYSTEM MERITS :

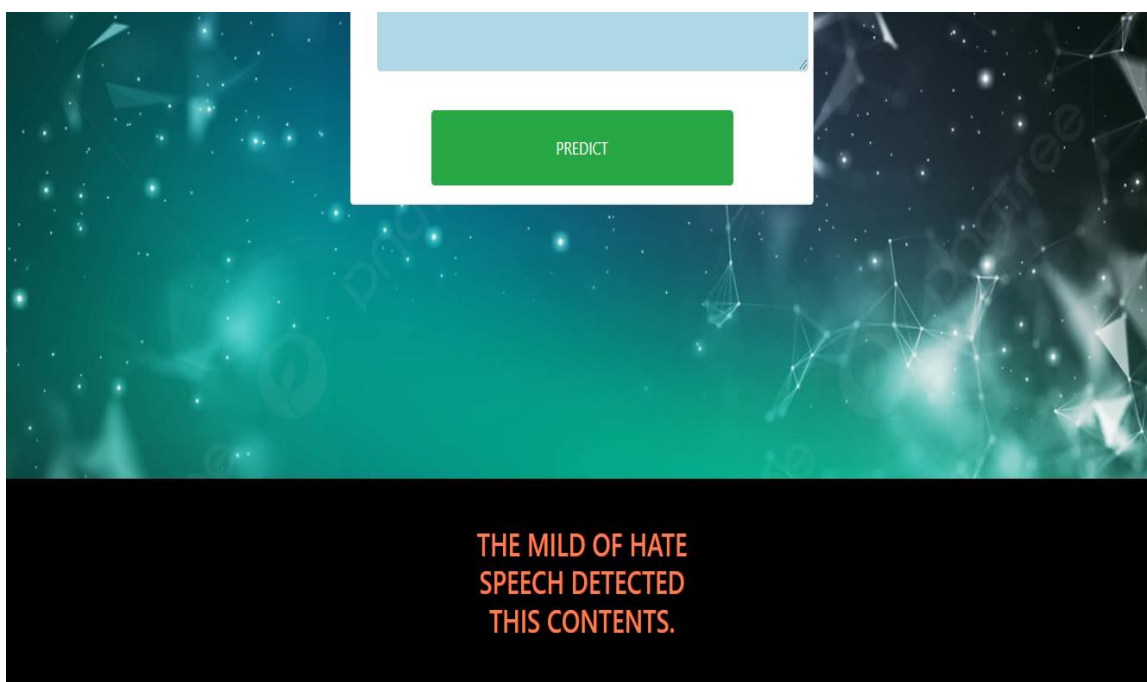
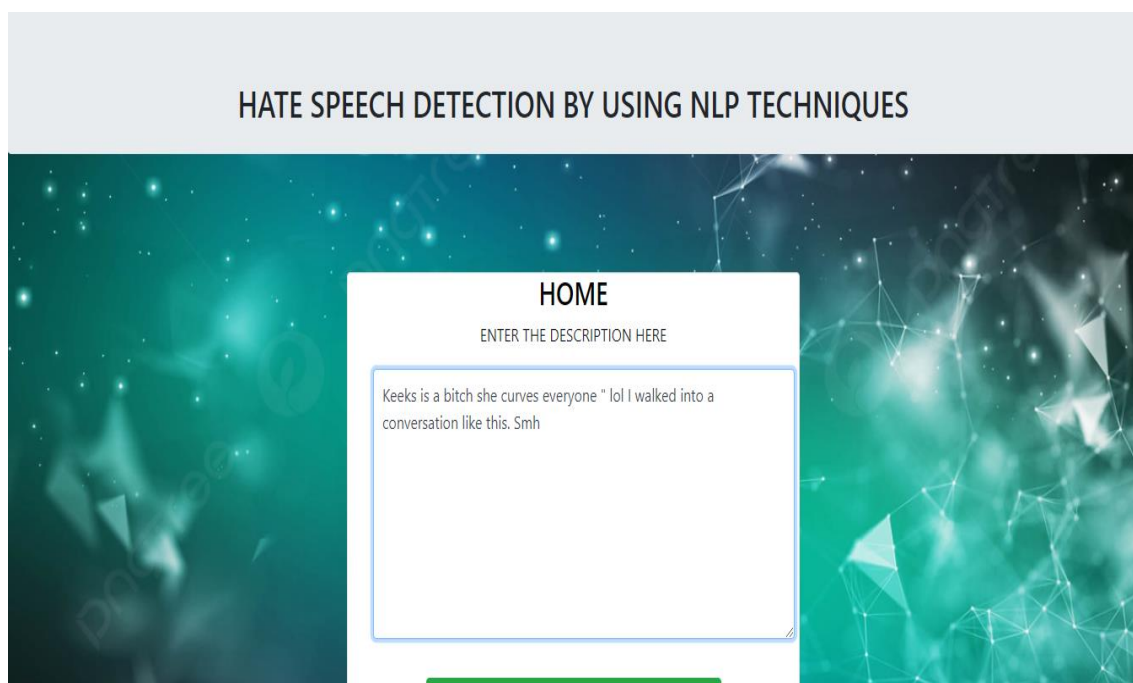
- We apply nlp techniques for predict Hate speech & depression.
- Accuracy was improved.
- We build a production level application for deployment purpose.
- We improved performance level.

DEPRESSION DETECTION BY USING NLP TECHNIQUES



DEPRESSION DETECTION BY USING NLP TECHNIQUES





A.3 Publication

International Conference on Advances in Computing, Communication and Applied Informatics
(ACCAI-2024)



Dear B PAVITHRA,

Title: PREDICTION OF HATE SPEECH AND DEPRESSION CLASSIFICATION BY USING ML TECHNIQUES

Paper ID: ACCAI-24--001

Your Paper has been successfully submitted.

If you need more clarification, please send mail to accai@stjosephs.ac.in.

Close

REFERENCES

- [1] P. H. Soloff et al., “Self-mutilation and suicidal behaviour in borderline personality disorder,” *J. Pers. Disorders*, vol. 8 , no. 4, pp. 257–267, 1994.
- [2] World Health Organization, “Depression and other common mental disorders: Global health estimates,” World Health Organization, pp. 7–24, 2017.
- [3] A. J. Flint et al., “Abnormal speech articulation, psychomotor retardation, and subcortical dysfunction in major depression,” *J. Psychiatric Res.*, vol. 27, no. 3, pp. 309–319, 1993.
- [4] A. Korszun, “Facial pain, depression and stress—Connections and directions,” *J. Oral Pathol. Med.*, vol. 31, no. 10, pp. 615–619, 2002.
- [5] A. McPherson and C. R. Martin, “A narrative review of the beck depression inventory (BDI) and implications for its use in an alcohol-Dependent population,” *J. Psychiatric Mental Health Nursing*, vol. 17, no. , pp. 19–30, 2010.
- [6] Jiawei Du and Kai Yang, “Depression Detection in NLP”, *IEEE Trans. Control Netw. Syst.*, vol. 4, no. 1, pp. 93– 105, Jun 2023.
- [7] Murtaza Ahmed Siddiqi and Wooguil Pak, “Tier-Based Optimization for Synthesized Network Intrusion Detection System”, *IEEE Trans. Cybern.*, vol. 50, no. 7, pp. 2996–3008, Jul. 2023.
- [8] J.-J. Yan and G.-H. Yang, “Hate Speech Detection in Tweets”: A multiobserver approach,” *IEEE Trans. Cybern.*, vol. 53, no. 3, pp. 1447–1459, Mar. 2023.
- [9] Mohamed Hadi Habaebi and Robiah Ahmad, “Hate Speech Detection”, *Int. J. Syst. Sci.*, vol. 53, no. 10, pp. 2247– 2259, Jul. 2022.
- [10] Taehoon Kim and Wooguil Pak, “Detection of Depression Classifier”, *Int. J. Netw. Dyn. Intell.*, vol. 1, no. 1, pp. 111–119, Dec. 2022.

