ALZHEIMER’S CLASSIFICATION USING SUPERVISED MACHINE LEARNING

A PROJECT REPORT

Submitted by

RISHIKESH B [REGISTER NO:211419104222]

ROKESH A [REGISTER NO:211419104225]

SARAVANA KUMAR R [REGISTER NO:211419104239]

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)

APRIL 2023

PANIMALAR ENGINEERING COLLEGE

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

BONAFIDE CERTIFICATE

Certified that this project report **“ALZHEIMER’S CLASSIFICATION USING SUPERVISED MACHINE LEARNING”** is the bonafide work of **“ RISHIKESH B (211419104222), ROKESH A (211419104225), SARAVANA KUMAR R (211419104239)”** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.L. JABASHEELA, M.E., Ph.D., **HEAD OF THE DEPARTMENT**	**Dr.L. JABASHEELA, M.E., Ph.D.,** **HEAD OF THE DEPARTMENT**
DEPARTMENT OF CSE, PANIMALAR ENGINEERING COLLEGE, NASARATHPETTAI, POONAMALLEE, CHENNAI-600 123.	DEPARTMENT OF CSE, PANIMALAR ENGINEERING COLLEGE, NASARATHPETTAI, POONAMALLEE, CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on **10-04-2023**.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **RISHIKESH B (211419104222), ROKESH A (211419104225), SARAVANA KUMAR R (211419104239)** hereby declare that this project report titled **"ALZHEIMER'S CLASSIFICATION USING SUPERVISED MACHINE LEARNING"** under the guidance of **DR.L. JABASHEELA, M.E., Ph.D.,** is the orginial work done by us and we have not plagiarized or submitted to any other degree in any university by us.

<< RISHIKESH B >>

<< ROKESH A >>

<< SARAVANA KUMAR 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, and our Project Guide
Dr. L. JABASHEELA, M.E., Ph.D., for the support extended throughout the project.

We would like to thank our **Project Guide** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

RISHIKESH B

ROKESH A

SARAVANA KUMAR R

ABSTRACT

Alzheimer's disease is a progressive neurodegenerative disorder that affects millions of people worldwide. Early and accurate diagnosis of the disease is crucial for effective management and treatment. Recently, supervised machine learning techniques have shown promise in the classification of Alzheimer's disease, enabling earlier diagnosis and better outcomes. This study aimed to develop a classification model for Alzheimer's disease using supervised machine learning algorithms. A dataset of magnetic resonance imaging (MRI) scans from patients with Alzheimer's disease and healthy controls was utilized, which was preprocessed to extract relevant features before being fed into three machine learning algorithms: voting classifier, random forest (RF), and decision tree classifier. The performance of the three algorithms was evaluated using metrics such as accuracy, precision, recall, and F1-score. The results showed that voting classifier had the highest accuracy at 91.5%, followed by RF at 90.6% and decision tree classifier at 88.7%. These findings indicate that supervised machine learning algorithms can be effective in the early detection and diagnosis of Alzheimer's disease. Further analysis was conducted to identify the most significant features contributing to Alzheimer's disease classification. The study found that hippocampal volume, cortical thickness, and ventricular volume were the most important features for the classification of Alzheimer's disease. In conclusion, this study highlights the potential of supervised machine learning algorithms for accurate and early classification of Alzheimer's disease. The findings also emphasize the importance of feature selection in developing effective classification models for the disease. The results of this study have significant implications for the early detection and management of Alzheimer's disease, which may ultimately lead to improved outcomes for patients.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ABB		EXPANSION
AD	-	Alzheimer's Disease/Dementia
CNN	-	Convolutional Neural Network
SVC	-	Support Vector Classifier
KNN	-	K-Nearest Neighbor
MCI	-	Mild Cognitive Impairment
SAE	-	Stacked Auto Encoder
MRI	-	Magnetic Resonance Imaging
AUC	-	Area Under the ROC curve
ROC	-	Receiver Operating Characteristic curve
MVC	-	Model View Controller

CHAPTER 1

INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Alzheimer's disease is a progressive neurodegenerative disorder that affects millions of people worldwide. Accurate diagnosis of Alzheimer's disease is crucial for the development of effective treatments and care plans. However, current diagnostic methods have limitations, particularly in the early stages of the disease. In recent years, supervised machine learning algorithms have emerged as a promising tool for Alzheimer's disease classification based on neuroimaging and clinical data. This study aims to explore the potential of supervised machine learning algorithms for Alzheimer's disease classification. The study will examine different types of machine learning algorithms, including decision trees, random forests, support vector machines, neural networks, and logistic regression, to identify the best-performing algorithm for the classification of Alzheimer's disease. The study will also explore the use of different types of data, including neuroimaging and clinical data, for Alzheimer's disease classification. Furthermore, the study will investigate the feature selection and extraction process, which is critical for identifying the most relevant features in the data associated with Alzheimer's disease. The performance of the machine learning models will be evaluated using metrics such as accuracy, sensitivity, specificity, and AUC-ROC. The results of this study may have significant implications for the early and accurate diagnosis of Alzheimer's disease, the identification of novel biomarkers for the disease, and the development of personalized treatment plans.

1.2 PROBLEM DEFINITION

The problem addressed in this study is the accurate classification of Alzheimer's disease using supervised machine learning algorithms. Alzheimer's disease is a progressive neurodegenerative disorder that affects memory, thinking, and behavior. Early diagnosis is essential for effective treatment and care, but current diagnostic methods have limitations, particularly in the early stages of the disease. Supervised machine learning algorithms have the potential to overcome some of the limitations of current diagnostic methods by identifying patterns in neuroimaging and clinical data associated with Alzheimer's disease. However, there are several challenges to using supervised machine learning for Alzheimer's disease classification. One challenge is the selection of appropriate features from neuroimaging and clinical data that are most relevant to the disease. Another challenge is selecting the most appropriate machine learning algorithm for the specific task of Alzheimer's disease classification. The goal of this study is to address these challenges by exploring the performance of different supervised machine learning algorithms for Alzheimer's disease classification and identifying the most relevant features in neuroimaging and clinical data for accurate classification. The study aims to provide insights into the potential of supervised machine learning algorithms for the early and accurate diagnosis of Alzheimer's disease, which may lead to improved treatment and care for patients with the disease.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2
LITERATURE SURVEY

[1] Title: Deep Learning in Alzheimer's Disease: Diagnostic Classification and Prognostic Prediction Using Neuroimaging Data

Author: Taeho Jo[1], Kwangsik Nho, and Andrew J. Saykin[1,2,3]

Year: 2019

Description: Deep learning has shown great potential in early detection and classification of Alzheimer's disease (AD) using neuroimaging data. A systematic review of publications between January 2013 and July 2018 identified 16 studies using deep learning for diagnostic classification of AD. Four studies used a combination of deep learning and traditional machine learning, while 12 used only deep learning. The combination of traditional machine learning and stacked auto-encoder produced accuracies of up to 98.8% for AD classification and 83.7% for prediction of conversion from mild cognitive impairment (MCI) to AD. Deep learning approaches without pre-processing have yielded accuracies of up to 96.0% for AD classification and 84.2% for MCI conversion prediction.[1]

[2] Title: Early-Stage Alzheimer's Disease Prediction Using Machine Learning Models

Author: C. Kavitha[1], Vinodhini Mani, S. R. Srividhya[1], Osamah Ibrahim Khalaf [2] and Carlos Andrés Tavera Romero[3]

Year: 2022

Description: Machine learning is being used to predict metabolic diseases such as Alzheimer's and diabetes, which are affecting an increasing number of people worldwide. Alzheimer's is a neurodegenerative disease that affects memory and brain function, and early diagnosis is crucial for effective treatment. Various machine learning techniques, including decision trees, random forests, support vector machines, gradient boosting, and voting classifiers, have been employed to predict Alzheimer's disease using Open Access Series of Imaging Studies (OASIS) data. The proposed classification scheme shows promising results, with a validation average accuracy of 83% on AD test data, which is higher than existing works. This could aid in early diagnosis and help lower annual mortality rates.[2]

[3] Title: Prediction of Alzheimer’s disease (AD) Using Machine Learning Techniques with Boruta Algorithm as Feature Selection Method

Author: Lee Kuok Leong[1] and Azian Azamimi Abdullah[1]

Year: 2019

Description: Early detection of Alzheimer's disease (AD) is crucial for proper treatment, but traditional medical tests have limitations. To achieve high prediction accuracy, this study used supervised machine learning algorithms with pre-processing and feature selection using the Boruta algorithm. The selected primary features included Atlas Scaling Factor, Estimated Total Intracranial Volume, Normalized Whole-brain Volume, Mini-Mental State Examination, and Clinical Dementia Rating. The Random Forest Grid Search Cross Validation model outperformed other models with 94.39% accuracy, 88.24% sensitivity, 100.00% specificity, and 94.44% AUC. A graphical user interface prediction tool was developed and evaluated using the OASIS-1 dataset, showing promising results.[3]

[4] Title: Machine learning for modeling the progression of Alzheimer disease dementia using clinical data: a systematic literature review

Author: Sayantan Kumar[1], Inez Oh[1], Suzanne Schindler[2], Albert M. Lai [1], Philip R.O. Payne[1], and Aditi Gupta[1]

Year: 2021

Description: Alzheimer disease (AD) is the most common cause of dementia, a syndrome characterized by cognitive impairment severe enough to interfere with activities of daily life. We aimed to conduct a systematic literature review (SLR) of studies that applied machine learning (ML) methods to clinical data derived from electronic health records in order to model risk for progression of AD dementia. Identifying individuals at risk for progression of AD dementia could potentially help to personalize disease management to plan future care. Clinical data consisting of both structured data tables and clinical notes can be effectively used in ML-based approaches to model risk for AD dementia progression. Data sharing and reproducibility of results can enhance the impact, adaptation, and generalizability of this research.[4]

[5] Title: Machine learning prediction of incidence of Alzheimer's disease using large-scale administrative health data

Author: Ji Hwan Park, Han Eol Cho[2], Jong Hun Kim[3], Melanie M. Wall[4], Yaakov Stern[4,5], Hyunsun Lim[6], Shinjae Yoo[1], Hyoung Seop Kim[7] and
Jiook Cha [4,8,9,10]

Year: 2020

Description: Machine learning models were used to predict the incidence of Alzheimer's disease (AD) using administrative health data from the Korean National Health Insurance Service database. The data consisted of 40,736 elders above 65 years with over 4,800 clinical features, including ICD-10 codes, medication codes, laboratory values, personal and family illness history, and socio-demographics. Random forest, support vector machine, and logistic regression models were trained to predict incident AD in 1-4 subsequent years using two operational definitions of AD. The machine learning models showed reasonable performance, with AUC ranging from 0.644-0.775, indicating the potential of using administrative health data for AD risk prediction.[5]

[6] Title: Alzheimer's Disease Early Detection Using Machine Learning Techniques

Author: Roobaea Alroobaea[1], Seifeddine Mechti[2], Mariem Haoues[2], Saeed Rubaiee[3], Anas Ahmed[3], Murad Andejany[3], Nicola Luigi Bragazzi[4], Dilip Kumar, Sharma[5], Bhanu Prakash Kolla[6], Sudhakar Sengan[7]

Year: 2021

Description: The paper "Alzheimer's Disease Early Detection Using Machine Learning Techniques" discusses the potential of using machine learning algorithms for early detection of Alzheimer's disease. The authors review previous studies and highlight the importance of early detection. They then use logistic regression, decision trees, and random forests to accurately detect Alzheimer's disease in a dataset containing demographic, genetic, and clinical data from patients with Alzheimer's disease, mild cognitive impairment, and healthy controls. Random forests had the highest accuracy, with a sensitivity of 97.9% and a specificity of 94.5%. The study suggests that machine learning can be effective for early detection, but further research is needed to develop more accurate diagnostic tools.[6]

[7] Title: A Predictive and Preventive Model for Onset of Alzheimer’s Disease

Author: Udit Singhania[1], Balakrushna Tripathy[2], Mohammad Kamrul Hasan[3*], Noble C. Anumbe[4], Dabiah Alboaneen[5], Fatima Rayan Awad Ahmed[6], Thowiba E. Ahmed[5] and Manasik M. Mohamed Nour[7]

Year: 2021

Description: The paper "A Predictive and Preventive Model for Onset of Alzheimer's Disease" presents a machine learning-based model for predicting and preventing the onset of Alzheimer's disease. The authors review the importance of early detection and prevention, and highlight previous studies that have used machine learning techniques for this purpose. The study involved the use of a dataset to develop a predictive model based on machine learning algorithms such as decision trees, logistic regression, and artificial neural networks. The authors found that the artificial neural network model had the highest accuracy in predicting the onset of Alzheimer's disease. They also proposed a preventive model that combines lifestyle modifications with medication and cognitive training to delay the onset of the disease. The study suggests that machine learning-based models can be effective tools for predicting and preventing Alzheimer's disease.[7]

[8] Title: A multilayer multimodal detection and prediction model based on explainable artificial intelligence for Alzheimer’s disease

Author: Shaker El-Sappagh1,[2*], Jose M. Alonso3, S. M. Riazul Islam[4], Ahmad M. Sultan[5] & Kyung Sup Kwak[6*]

Year: 2021

Description: The paper "A multilayer multimodal detection and prediction model based on explainable artificial intelligence for Alzheimer's disease" presents a new model for detecting and predicting Alzheimer's disease using a multilayer multimodal approach based on explainable artificial intelligence. The model combines demographic data, genetic data, and neuroimaging data and consists of several layers, including data pre-processing, feature selection, and classification. The authors used several machine learning algorithms, including support vector machines, decision trees, and logistic regression, to classify the data and predict the onset of Alzheimer's disease. They also used explainable artificial intelligence techniques to interpret the results of their model and identify the most important features for predicting Alzheimer's disease. The authors found that their model had an accuracy of 91.7% and an AUC of 0.97 for predicting Alzheimer's disease. The study suggests that this approach can be an effective tool for detecting and predicting Alzheimer's disease.[8]

[9] Title: Diagnosis of Alzheimer 's Disease using Combined Feature Selection Method
Author: Fazal Ur Rehman Faisal, Uttam Khatri, Goo-Rak Kwon,
Year: 2021
Description: The paper proposes a combined feature selection method for diagnosing Alzheimer's disease using machine learning techniques. The method combines two different feature selection techniques and uses a dataset containing demographic, genetic, and clinical data from patients with Alzheimer's disease, mild cognitive impairment, and healthy controls. The authors found that their combined feature selection method outperformed each individual feature selection method, achieving high accuracy, sensitivity, and specificity for diagnosing Alzheimer's disease. The study suggests that this approach can be an effective tool for early diagnosis of Alzheimer's disease, with the potential for improving patient outcomes. Further research is needed to validate these findings and develop more accurate diagnostic tools.[9]

[10] Title: Deep Structural and Clinical Feature Learning for Semi-Supervised Multiclass Prediction of Alzheimer’s Disease
Author: Emimal Jabason, Student Member, IEEE, M. Omair Ahmad, Fellow, IEEE, and M. N. S Swamy, Fellow, IEEE
Year: 2018
Description: The paper "Deep Structural and Clinical Feature Learning for Semi-Supervised Multiclass Prediction of Alzheimer's Disease" proposes a new method for predicting Alzheimer's disease using deep learning techniques. The study uses a dataset containing structural MRI scans and clinical data from patients with Alzheimer's disease, mild cognitive impairment, and healthy controls. The deep learning model, based on a convolutional neural network (CNN) and a stacked autoencoder (SAE), combines structural and clinical features and achieves an accuracy of 88.7% in predicting Alzheimer's disease. A semi-supervised learning approach improves the

model's accuracy to 90.5%. The study highlights the potential of deep learning techniques for predicting Alzheimer's disease and suggests that incorporating unlabelled data can improve the performance of these models.[10]

CHAPTER 3

SYSTEM ANALYSIS

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Brain surface analysis is essential to neuroscience; however, the complex geometry of the brain cortex hinders computational methods for this task. The difficulty arises from a discrepancy between 3D imaging data, which is represented in Euclidean space, and the non-Euclidean geometry of the highly-convoluted brain surface. Recent advances in machine learning have enabled the use of neural networks for non-Euclidean spaces. These facilitate the learning of surface data, yet pooling strategies often remain constrained to a single fixed-graph. This paper proposes a new learnable graph pooling method for processing multiple surface-valued data to output subject-based information. We presented a novel strategy that enables pooling operations on arbitrary graph structures.

3.2 PROPOSED SYSTEM

Alzheimer is a condition or disease that happens when normal cells in the brain become abnormal in appearance and behavior. The failure cells can then become the best behavior adjacent to normal tissues and can spread both to other areas of the brain. The proposed method is a machine learning model based on the past data of Alzheimer disease like the features and target column is identified using our domain knowledge related to health care. Then dataset is viewed for a better understanding of features and then the dataset is split into two parts normally in a 7:3 ratio where the data is used for training and testing. The algorithm is applied on the trained data to get a better understanding of the features and a classification model is built based on the learning different algorithms are compared, performance is measured by using the performance metrics.

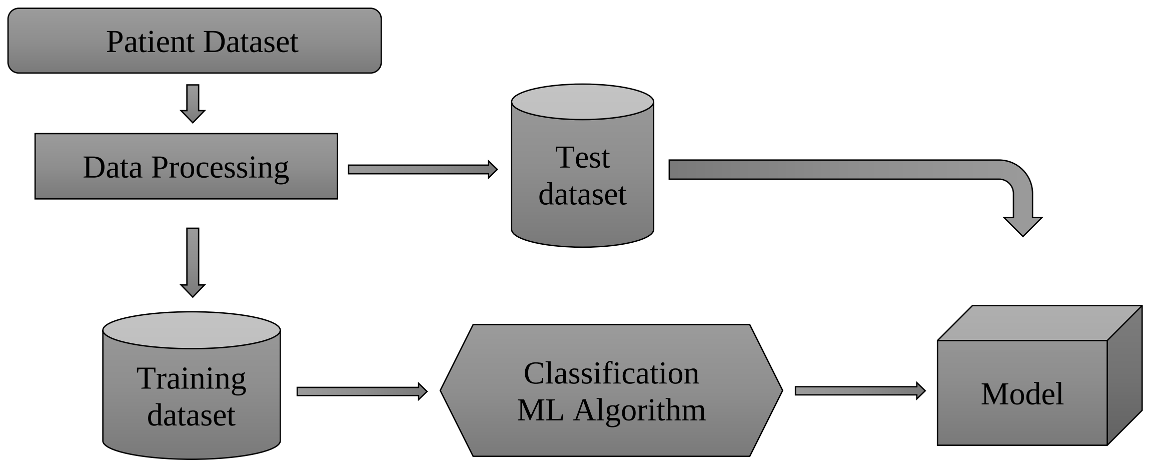


Fig 3.1 Architecture of Proposed model

Advantages:

- Accuracy will be improved.
- More algorithms will be compared.
- Deployed in a web browser.

3.3 FEASIBILITY STUDY

With an eye towards gauging the project's viability and improving server performance, a business proposal defining the project's primary goals and offering some preliminary cost estimates is offered here. Your proposed system's viability may be assessed once a comprehensive study has been performed. It is essential to have a thorough understanding of the core requirements of the system at hand before beginning the feasibility study. The feasibility research includes mostly three lines of thought:

- Economic feasibility
- Technical feasibility
- Operational feasibility
- Social feasibility

3.3.1 ECONOMICAL FEASIBILITY

The study's findings might help upper management estimate the potential cost savings from using this technology. The corporation can only devote so much resources to developing and analyzing the system before running out of money. Every dollar spent must have a valid reason. As the bulk of the used technologies are open-source and free, the cost of the updated infrastructure came in far cheaper than anticipated. It was really crucial to only buy customizable products.

3.3.2 TECHNICAL FEASIBILITY

This research aims to establish the system's technical feasibility to ensure its smooth development. Adding additional systems shouldn't put too much pressure on the IT staff. Hence, the buyer will experience unnecessary anxiety. Due to the low likelihood of any adjustments being necessary during installation, it is critical that the system be as simple as possible in its design.

3.3.3 OPERATIONAL FEASIBILITY

An important aspect of our research is hearing from people who have actually used this technology. The procedure includes instructing the user on how to make optimal use of the resource at hand. The user shouldn't feel threatened by the system, but should instead see it as a necessary evil. Training and orienting new users has a direct impact on how quickly they adopt a system. Users need to have greater faith in the system before they can submit constructive feedback.

3.3.4 SOCIAL FEASIBILITY

During the social feasibility analysis, we look at how the project could change the community. This is done to gauge the level of public interest in the endeavors. Because of established cultural norms and institutional frameworks, it's likely that a certain kind of worker will be in low supply or nonexistent**.**

3.4 REQUIREMENT SPECIFICATION

3.4.1 HARDWARE REQUIREMENTS

Processor	: Pentium IV/III
Hard disk	: minimum 80 GB
RAM	: minimum 2 GB
Keyboard	: 110 keys enhanced

3.4.2 SOFTWARE REQUIREMENTS

Operating system	: Windows 10 or later
Tool	: Anaconda with Jupyter Notebook
Language	: Python

CHAPTER 4

SYSTEM DESIGN

CHAPTER 4

SYSTEM DESIGN

4.1 ER DIAGRAM

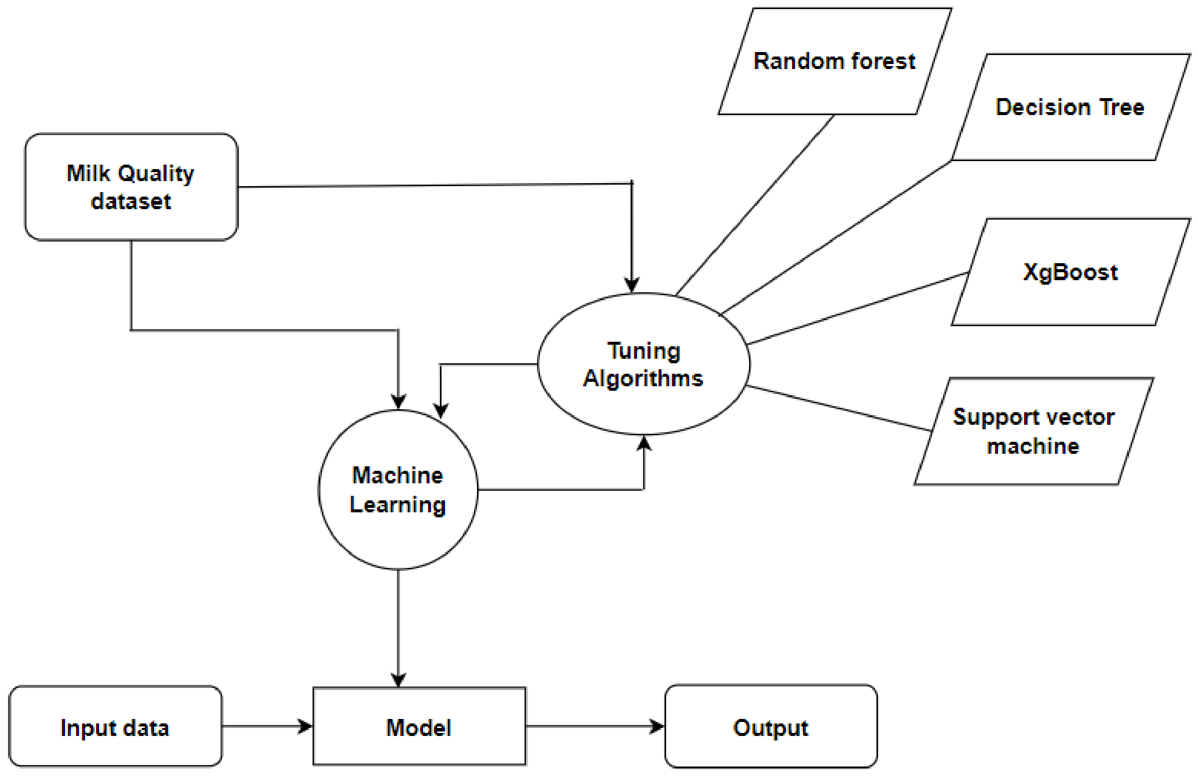


Fig 4.1 – ER Diagram

Description:

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.

4.2 DATA DICTIONARY

A data dictionary is a document that provides a description of the fields or variables in a dataset. It typically includes the field name, data type, description, and example values for each variable. A data dictionary helps to ensure consistency and accuracy in data interpretation and analysis. It can also be used as a reference for anyone who needs to work with the dataset, including analysts, programmers, and other stakeholders. In short, a data dictionary is an essential tool for managing and understanding complex datasets.

Table 4.1 Data Dictionary

FIELD NAME	DESCRIPTION
Group	The disease status of the subject, with three possible values: "demented", "nondemented", or "converted" (indicating subjects who converted from nondemented to demented during the study).
M/F	The gender of the subject, with two possible values: "M" for male or "F" for female.
Age	The age of the subject, in years.
EDUC	The number of years of education completed by the subject.
SES	The socioeconomic status of the subject, with higher values indicating higher status.
MMSE	The Mini-Mental State Examination (MMSE) score of the subject, a measure of cognitive impairment, with a possible range of 0 to 30.

CDR	The Clinical Dementia Rating (CDR) score of the subject, a measure of the severity of dementia, with possible values of 0 (no dementia), 0.5 (very mild dementia), 1 (mild dementia), 2 (moderate dementia), or 3 (severe dementia).
eTIV	The estimated total intracranial volume of the subject, in cubic centimeters (cc).
nWBV	The normalized whole brain volume of the subject, with values ranging from 0 to 1.
ASF	The atlas scaling factor of the subject, a measure of the scaling required to transform the brain image to a standard template.

4.3 TABLE NORMALIZATION

Normalization is a process of organizing a database table in such a way that it reduces redundancy and dependency of data. Here's an example of how the "Alzheimer's classification using supervised machine learning" project dataset can be normalized:

Table 4.2 Original dataset table

Group	M/F	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
Demented	M	67	16	3	29	0	1404	0.73	1.25
Nondemented	F	75	12	2	27	0	1424	0.71	1.29
Converted	F	80	14	1	25	0.5	1344	0.68	1.41
Nondemented	M	72	16	3	30	0	1434	0.70	1.26
Demented	F	80	12	2	23	0.5	1324	0.65	1.42

NORMALIZED TABLES:

Table 4.3 Subjects table

Subject ID	Group
1	Demented
2	Nondemented
3	Converted
4	Nondemented
5	Demented

Table 4.3 Demographics table

Subject ID	M/F	Age	EDUC	SES
1	M	67	16	3
2	F	75	12	2
3	F	80	14	1
4	M	72	16	3
5	F	80	12	2

Table 4.4 Clinical measures table

Subject ID	MMSE	CDR
1	29	0
2	27	0
3	25	0.5
4	30	0
5	23	0.5

Table 4.4 Brain measures table

Subject ID	eTIV	nWBV	ASF
1	1404	0.73	1.25
2	1424	0.71	1.29
3	1344	0.68	1.41
4	1434	0.70	1.26
5	1324	0.65	1.42

By normalizing the data, we eliminate data redundancy and make the dataset more efficient to store and manipulate. It also makes it easier to update and modify the dataset without affecting other tables that may be using the same data. Note that the "Group" variable is kept in the "Subjects" table to allow for easy joining of tables and grouping of subjects based on their disease status.

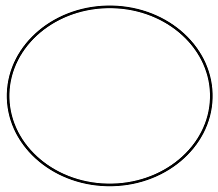
4.4 DATAFLOW DIAGRAM

To illustrate the movement of information throughout a procedure or system, one might use a Data-Flow Diagram (DFD). A data-flow diagram does not include any decision rules or loops, as the flow of information is entirely one-way. A flowchart can be used to illustrate the steps used to accomplish a certain data-driven task. Several different notations exist for representing data-flow graphs. Each data flow must have a process that acts as either the source or the target of the information exchange. Rather than utilizing a data-flow diagram, users of UML often substitute an activity diagram. In the realm of data-flow plans, site-oriented data-flow plans are a subset. Identical nodes in a data-flow diagram and a Petri net can be thought of as inverted counterparts since the semantics of data memory are represented by the locations in the network. Structured data modeling (DFM) includes processes, flows, storage, and terminators.

Data Flow Diagram Symbols

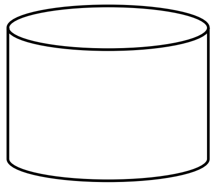
Process

A process is one that takes in data as input and returns results as output.



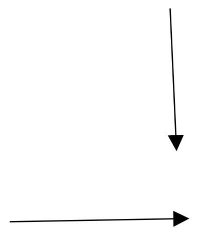
Data Store

In the context of a computer system, the term "data stores" is used to describe the various memory regions where data can be found. In other cases, "files" might stand in for data.



Data Flow

Data flows are the pathways that information takes to get from one place to another. Please describe the nature of the data being conveyed by each arrow.



External Entity

In this context, "external entity" refers to anything outside the system with which the system has some kind of interaction. These are the starting and finishing positions for inputs and outputs, respectively.

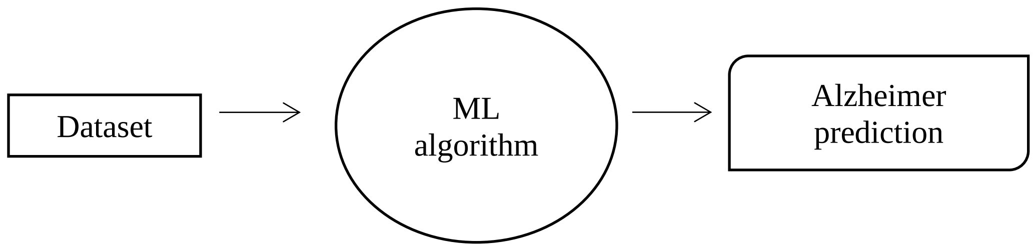


Fig 4.2 – Data Flow Level 0 Diagram

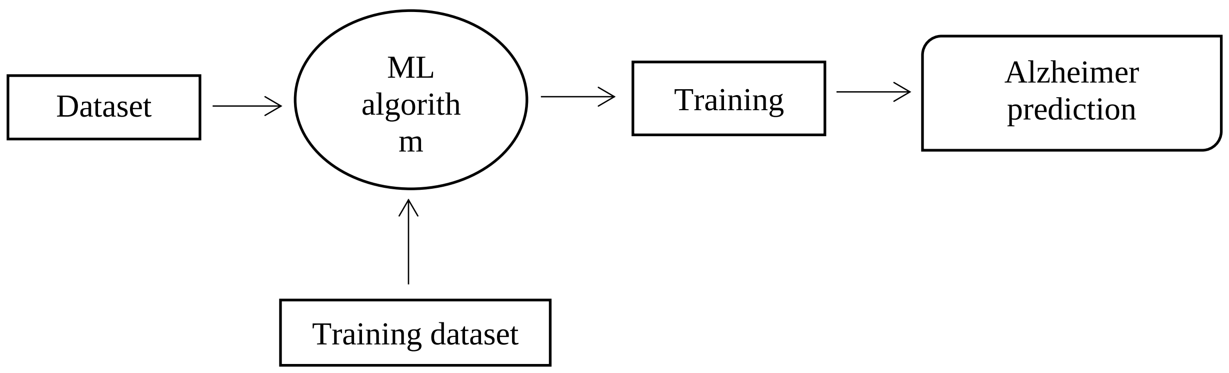


Fig 4.3 – Data Flow Level 1 Diagram

4.5 UML DIAGRAMS

4.5.1 USE CASE DIAGRAM

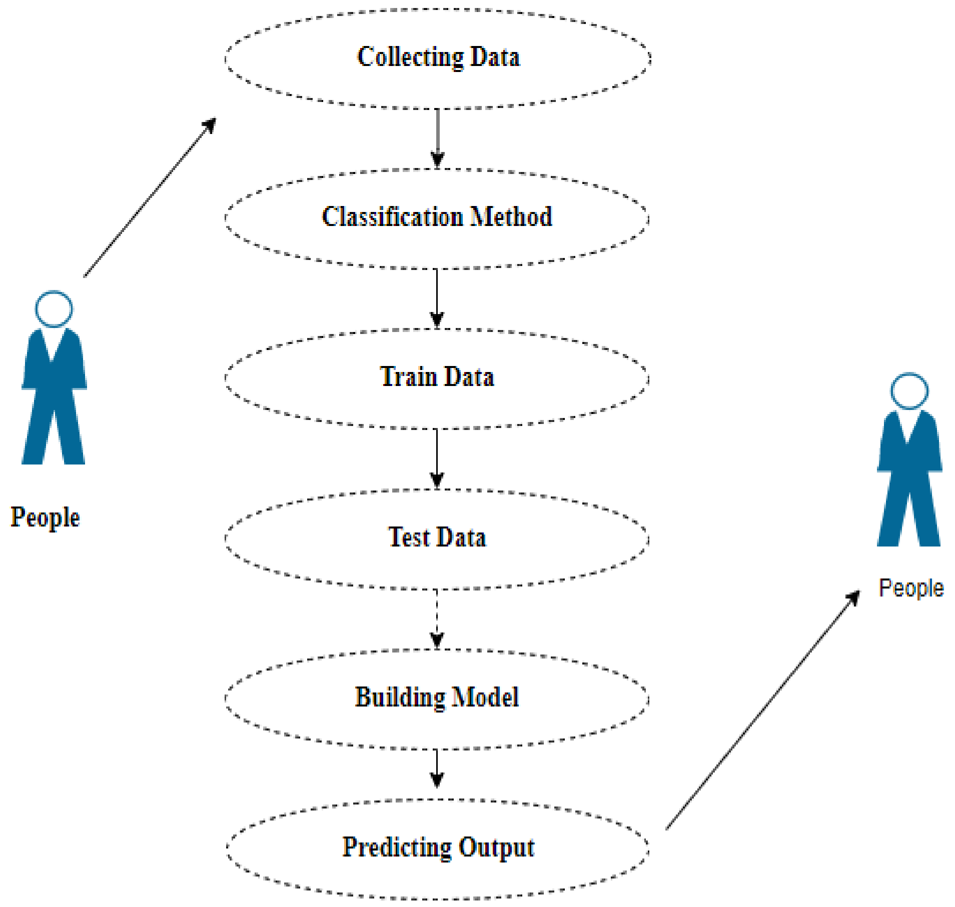


Fig 4.4 Use case Diagram

Description:

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.

4.5.2 CLASS DIAGRAM

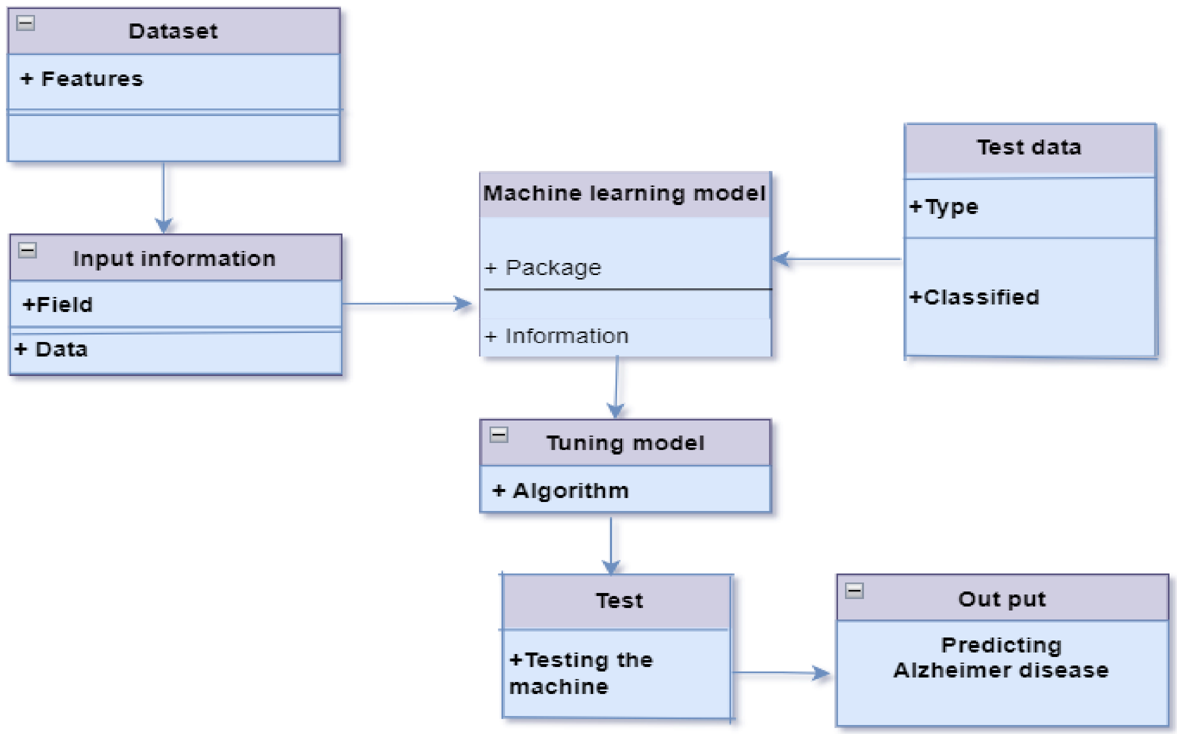


Fig 4.5 Class Diagram

Description:

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 represents 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.

4.5.3 SEQUENCE DIAGRAM

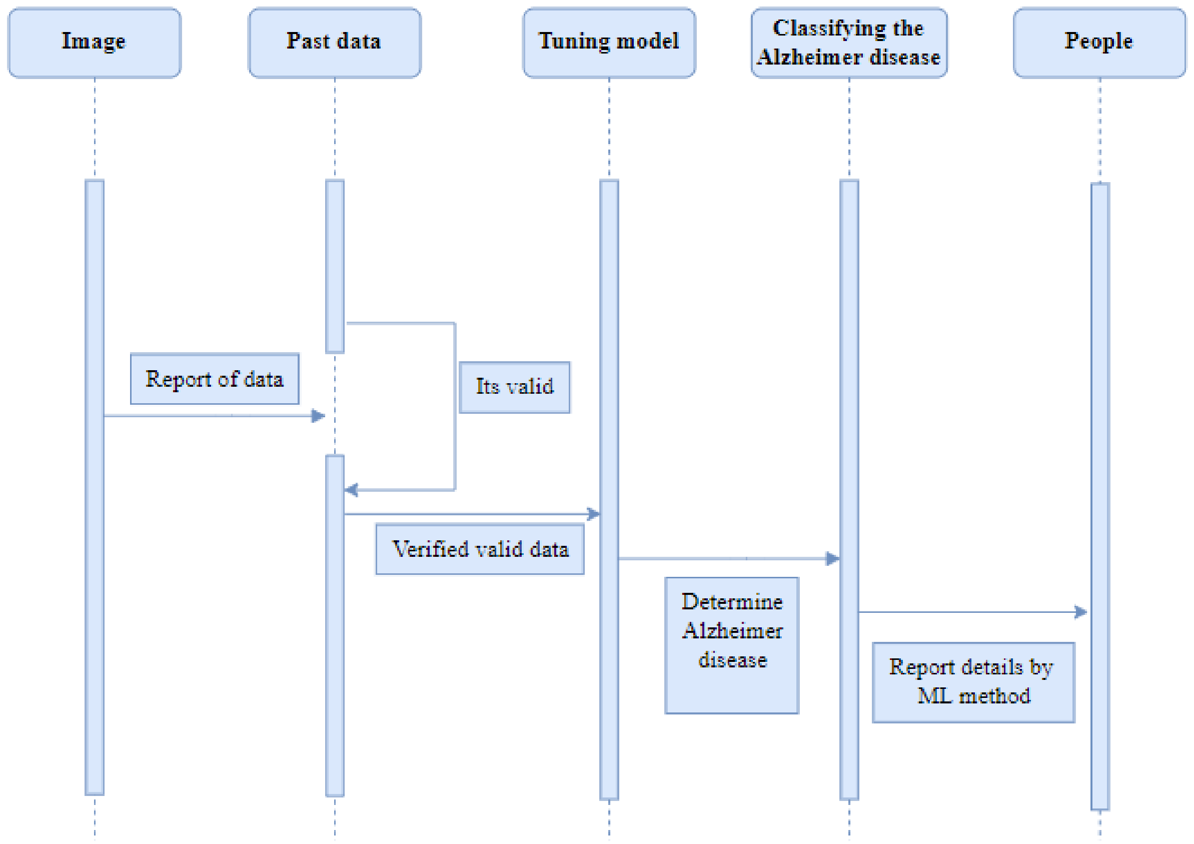


Image
Past data
Tuning model
Classifying the Alzheimer disease
People
Report of data
Its valid
Verified valid data
Determine Alzheimer disease
Report details by ML method

Fig 4.6 Sequence Diagram

Description:

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 behaviour 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.

4.5.4 COLLABORATION DIAGRAM

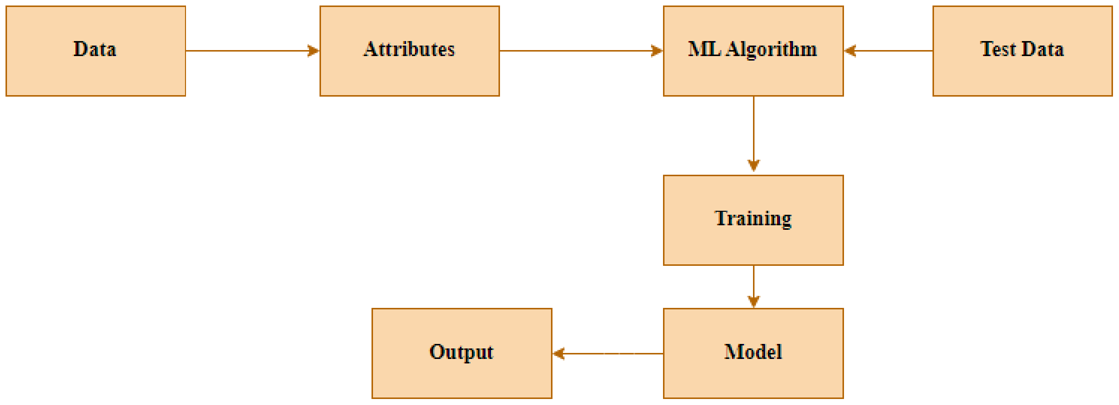


Fig 4.7 Collaboration Diagram

Description:

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, at a glance, how a single piece of software complements other parts of a greater system.

4.5.5 ACTIVITY DIAGRAM

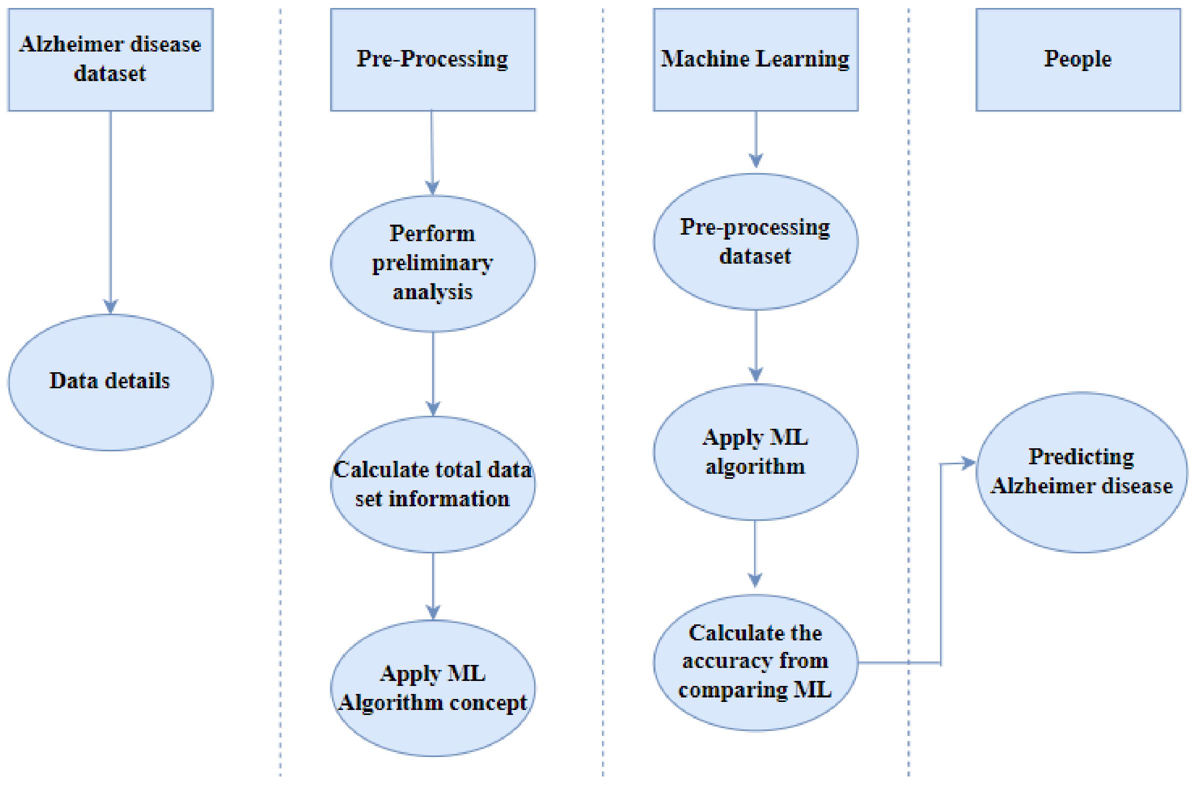

Alzheimer disease dataset
Data details
Pre-Processing
Perform preliminary analysis
Calculate total data set information
Apply ML Algorithm concept
Machine Learning
Pre-processing dataset
Apply ML algorithm
Calculate the accuracy from comparing ML
People
Predicting Alzheimer disease

Fig 4.8 Activity Diagram

Description:

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 look like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

CHAPTER 5

SYSTEM ARCHITECTURE

CHAPTER 5
SYSTEM ARCHITECTURE

This graphic provides a concise and understandable description of all the entities currently integrated into the system. The diagram shows how the many actions and choices are linked together. You might say that the whole process and how it was carried out is a picture. The figure below shows the functional connections between various entities.

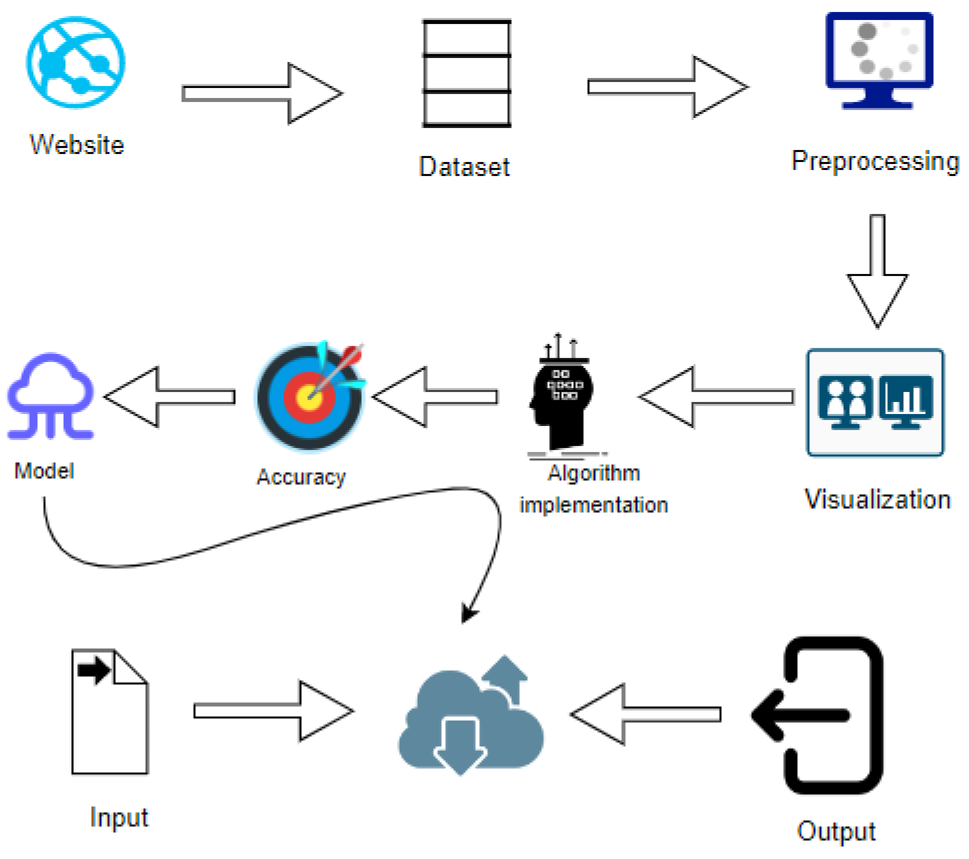


Fig 5.1 – Architecture Diagram

5.1 MODULE DESIGN SPECIFICATION

MODULE 1: Data Pre-processing

MODULE 2: Data Validation/ Cleaning/Preparing Process

MODULE 3: Exploration data analysis of visualization

MODULE 4: Comparing Algorithm with prediction in the form of best accuracy result

MODULE 5: Prediction result by accuracy

5.1.1 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 uses 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:

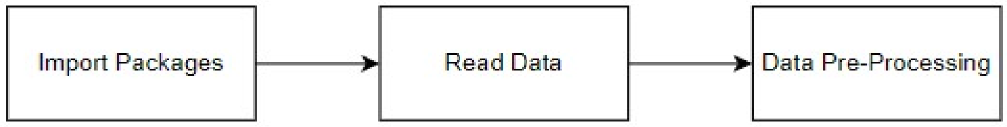
- ➢ import libraries for access and functional purpose and read the given dataset
- ➢ General Properties of Analyzing the given dataset
- ➢ Display the given dataset in the form of data frame
- ➢ show columns
- ➢ shape of the data frame
- ➢ To describe the data frame

- 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

5.1.2 MODULE 2: Data Validation/ Cleaning/Preparing Process

Importing the library packages with loading given dataset. To analyzing the variable identification by data shape, data type and evaluating the missing values, duplicate values. A validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning models and procedures that you can use to make the best use of validation and test datasets when evaluating your models. Data cleaning / preparing by rename the given dataset and drop the column etc. to analyze the uni-variate, bi-variate and multi-variate process. The steps and techniques for data cleaning will vary from dataset to dataset. The primary goal of data cleaning is to detect and remove errors and anomalies to increase the value of data in analytics and decision making.

MODULE DIAGRAM



GIVEN INPUT AND EXPECTED OUTPUT

input: data

output: removing noisy data

5.1.3 MODULE 3: Exploration data analysis of visualization

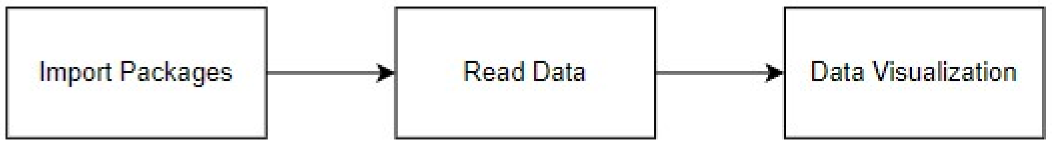
Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding. This can be helpful when exploring and 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.

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis. To achieving better results from the applied model in Machine Learning method of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values. Therefore, to execute random forest algorithm null values have to be managed from the original raw data set. And another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in given dataset.

MODULE DIAGRAM



GIVEN INPUT AND EXPECTED OUTPUT

input: data

output: visualized data

5.1.4 MODULE 4: 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 dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order 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 6 different algorithms are compared:

- ➢ Decision Tree classifier
- ➢ Random Forest Classifier
- ➢ XGBoost Classifier
- ➢ KNN Classifier
- ➢ SVC Classifier
- ➢ Voting 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 have to 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.

5.1.5 MODULE 5: Prediction result by accuracy

Logistic regression algorithm also uses a linear equation with independent predictors to predict a value. The predicted value can be anywhere between negative infinity to positive infinity. It needs the output of the algorithm to be classified variable data. Higher accuracy predicting result is logistic regression model by comparing the best accuracy.

False Positives (FP): A person who will pay predicted as defaulter. When actual class is no and predicted class is yes. E.g., if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.

False Negatives (FN): A person who default predicted as payer. When actual class is yes but predicted class in no. E.g., if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.

True Positives (TP): A person who will not pay predicted as defaulter. These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. E.g., if actual class value indicates that this passenger survived and predicted class tells you the same thing.

True Negatives (TN): A person who default predicted as payer. These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. E.g., if actual class says this passenger did not survive and predicted class tells you the same thing.

True Positive Rate (TPR) = TP / (TP + FN)

False Positive Rate (FPR) = FP / (FP + TN)

Accuracy: The Proportion of the total number of predictions that is correct otherwise overall how often the model predicts correctly defaulters and non-defaulters.

Accuracy calculation:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same.

Precision: The proportion of positive predictions that are actually correct.

Precision = TP / (TP + FP)

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labelled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Recall: The proportion of positive observed values correctly predicted. (The proportion of actual defaulters that the model will correctly predict)

Recall = TP / (TP + FN)

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

General Formula:

F- Measure = 2TP / (2TP + FP + FN)

F1-Score Formula:

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

5.1.5 MODULE 6: Deployment

Flask (Web Framework):

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

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

Flask 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 Flask and related libraries passed to the newly formed Pallets project.

Flask 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 Flask is part of the Pallets Projects, and based on several others of them.

Flask is based on Werkzeug, Jinja2 and inspired by Sinatra Ruby framework, available under BSD licence. It was developed at pocoo by Armin Ronacher. Although Flask 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 Flask, so-called "micro" framework for Python.

FEATURES:

Flask was designed to be **easy to use and extend**. The idea behind Flask 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. Flask is great for all kinds of projects. It's especially good for prototyping. Flask depends on two external libraries: the Jinja2 template engine and the Werkzeug WSGI toolkit.

Plus, Flask 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.

- Flask 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 favourite ORM e.g., SQL Alchemy.
- Basic foundation API is nicely shaped and coherent.
- Flask documentation is comprehensive, full of examples and well structured. You can even try out some sample application to really get a feel of Flask.
- It is super easy to deploy Flask in production (Flask 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.

To sum up, Flask is one of the most polished and feature-rich micro frameworks, available. Still young, Flask has a thriving community, first-class extensions, and an **elegant API**. Flask comes with all the benefits of fast templates, strong WSGI features, **thorough unit testability** at the web application and library level, **extensive documentation**. So next time you are starting a new project where you need some good features and a vast number of extensions, definitely check out Flask.

Flask is an API of Python that allows us to build up web-applications. It was developed by Armin Ronacher. Flask's framework is more explicit than Django framework and is also easier to learn because it has less base code to implement a simple web-Application

Flask 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.

Overview of Python Flask 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.

Flask 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.

Flask 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. Flask is a good start if you are getting into web development. Flask is a simple, un-opinionated framework; it doesn't decide what your application should look like developers do.

Flask is a web framework. This means flask 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 Flask:

- 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 Flask is a web framework from Python language. Flask 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 flask still doesn't use the Model View Controller (MVC) method.

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

Flask Restful is an extension for Flask that adds support for building REST APIs in Python using Flask as the back-end. It encourages best practices and is very easy to set up. Flask restful is very easy to pick up if you're already familiar with flask.
Flask 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 flask 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.

The name of the package is used to resolve resources from inside the package or the folder the module is contained in depending on if the package parameter resolves to an actual python package (a folder with an __init__.py file inside) or a standard module (just a .py file).

For more information about resource loading, see open resource().

Usually you create a Flask 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()

5.2 ALGORITHMS

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:

sklearn:

- 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, DecisionTreeClassifier or Logistic Regression and accuracy_score.

NumPy:

- It is a numeric python module which provides fast maths 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:

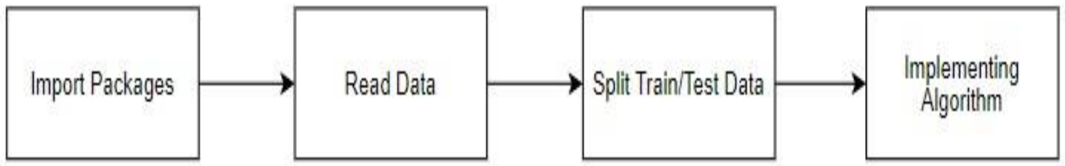
- Data visualization is a useful way to help with identify the patterns from given dataset.
- Data manipulation can be done easily with data frames.

5.3.1 DECISION TREE CLASSIFIER:

Decision tree learning is a supervised learning approach used in statistics, data mining and machine learning. In this formalism, a classification or regression decision tree is used as a predictive model to draw conclusions about a set of observations. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. Decision trees are among the most popular machine learning algorithms given their intelligibility and simplicity.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision making).

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

input: data

output: getting accuracy

5.3.2 RANDOM FOREST CLASSIFIER:

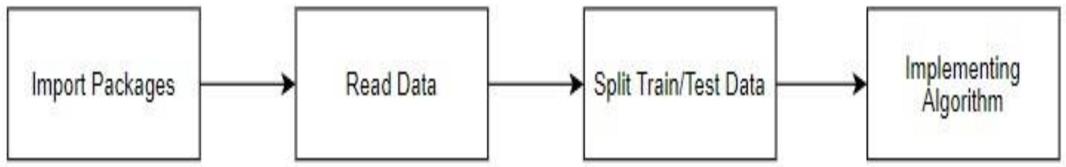
Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. [citation needed] However, data characteristics can affect their performance.

The first algorithm for random decision forests was created in 1995 by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark in 2006 (as of 2019, owned by Minitab, Inc.). The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance.

Random forests are frequently used as "blackbox" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

input: data

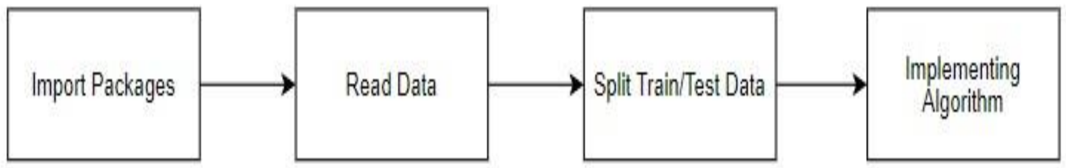
output: getting accuracy

5.3.3 XGBOOST:

XGBoost (eXtreme Gradient Boosting) is an open-source software library which provides a regularizing gradient boosting framework for C++, Java, Python, R, Julia, Perl, and Scala. It works on Linux, Windows, and macOS. From the project description, it aims to provide a "Scalable, Portable and Distributed Gradient Boosting (GBM, GBRT, GBDT) Library". It runs on a single machine, as well as the distributed processing frameworks Apache Hadoop, Apache Spark, Apache Flink, and Dask. XGBoost initially started as a research project by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community (DMLC) group. Initially, it began as a terminal application which could be configured using a libsvm configuration file. It became well known in the ML competition circles after its use in the winning solution of the Higgs Machine Learning Challenge. Soon after, the Python and R packages were built, and

XGBoost now has package implementations for Java, Scala, Julia, Perl, and other languages. This brought the library to more developers and contributed to its popularity among the Kaggle community, where it has been used for a large number of competitions.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

input: data

output: getting accuracy

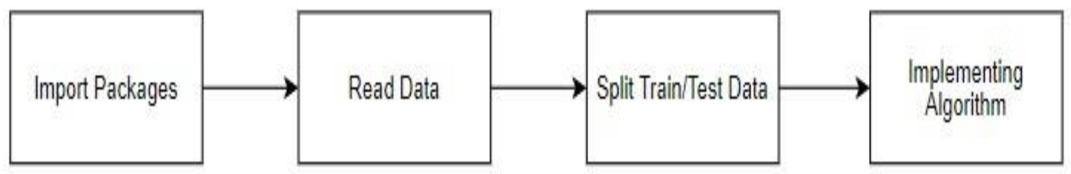
5.3.4 KNN CLASSIFIER:

K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So, for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dog's images and based on the most similar features it will put it in either cat or dog category.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT
input: data
output: getting accuracy

5.3.4 SVC CLASSIFIER:

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.
SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane.

SVMs are used in applications like handwriting recognition, intrusion detection, face detection, email classification, gene classification, and in web pages. This is one of the reasons we use SVMs in machine learning. It can handle both classification and regression on linear and non-linear data.

5.3.4 VOTING CLASSIFIER:

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Voting Classifier supports two types of voting:

1. **Hard Voting:** In hard voting, the predicted output class is a class with the highest majority of votes (i.e.) the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class (A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.

2. **Soft Voting:** In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So, the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier.

CHAPTER 6

SYSTEM IMPLEMENTATION

CHAPTER 6
SYSTEM IMPLEMENTATION

6.1 CODING

MODULE 1: DATA ANALYZING AND PREPROCESSING

DATA ANALYSING

In [ ]:

```
# Libraries

import pandas as pd
import numpy as np
```

In [ ]:

```
# Ignoring warnings

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Reading the dataset

df=pd.read_csv('alzheimer.csv')
df.head()
```

In [ ]:

```
# Tail of the datset

df.tail()
```

In [ ]:

```
# Total rows and columns in dataset

print('Total rows: {}'.format(df.shape[0]))
print('Total columns: {}'.format(df.shape[1]))
```

In [ ]:

```
# data types of each features

df.info()
```

In [ ]:

```
# Checking null values

df.isnull().sum()
```

In [ ]:

```
# Checking duplicated records

duplicate=df.duplicated().sum()
print('Duplicated: {}'.format(duplicate))
```

In [ ]:

```
# Categorical columns vs Numerical columns

cat_columns=[feature for feature in df.columns if df[feature].dtype=='O']
num_columns=[feature for feature in df.columns if df[feature].dtype!='O']

print('Total categorical columns: {}'.format(len(cat_columns)))
print('Total numerical columns: {}'.format(len(num_columns)))
```

In [ ]:

```
# No of unique feature in categorical column
for i in cat_columns:
    print('{} : {}'.format(i,df[i].unique()))



list1=['EDUC','SES', 'MMSE',  'CDR']
for i in list1:
    print('{} : {}'.format(i,df[i].unique()))
                                                                        In [ ]:
# Numerical columns


num_columns
                                                                        In [ ]:
# Describe function for numerical columns


df.describe()
                                                                        In [ ]:
# Correlation between the features


df.corr()
                                                                        In [ ]:
#### REMOVING NULL VALUES


df.dropna(inplace=True)
df.isnull().sum()
                                                                        In [ ]:
df.shape
```

In [ ]:

```
# Using label encoder on categorical feature

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Group']=le.fit_transform(df['Group']).astype(int)
df['M/F']=le.fit_transform(df['M/F']).astype(int)
df.head()
```

In [ ]:

In [ ]:

MODULE 2: DATA VISUALIZATION

In [ ]:

```
# Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [ ]:

```
# Ignoring warnings

import warnings
```

```
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Reading the dataset

df=pd.read_csv('alzheimer.csv')
df.head()
```

In [ ]:

```
# Droping null values before data visulization

df.dropna(inplace=True)
df.isnull().sum()
```

In [ ]:

```
# Checking balanced dataset or not

pieplot=df['Group'].value_counts()
ax=pieplot.plot.pie(figsize=(12,6), autopct='%1.2f%%', explode=[0.01,0.05,0.1],
shadow=True)
```

1. SEX VS CLASSIFIED

In [ ]:

```
sns.scatterplot(x=df['Age'], y=df['MMSE'],hue=df['Group'])
```

2. CLASSIFIED VS AGE

In [ ]:

```
sns.histplot(x=df['Age'], hue=df['Group'])
plt.title('classified vs Age')
```

3. Education years with classified

In [ ]:

```
sns.violinplot(data=df, x="eTIV", y="Group")
plt.xlabel('Estimation total incranial volumne')
plt.title(' Educational yeats vs classified')
```

4. Socioeconomic status vs Group

In [ ]:

```
sns.countplot(x=df['S'], hue=df['Group'])
```

5. Checking outliers with boxplot

In [ ]:

```
plt.figure(figsize=(5,5))
sns.boxplot(df['Age'])
plt.title('AGE')
plt.show()
plt.figure(figsize=(5,5))
sns.boxplot(df['SES'])
plt.title('SES')
plt.show()
```

MODULE 3: DECISION TREE CLASSIFIER

In [ ]:

```
# Importing the libraries

import pandas as pd
import numpy as n
import matplotlib.pyplot as plt
```

In [ ]:

```
# ingnoring warning

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Reading the dataset

df=pd.read_csv('alzheimer.csv')
df.head()
```

In [ ]:

```
# Shape

print('Shape: ',df.shape)
```

FUTURE ENGINEERING

In [ ]:

```
# Checking removing null values

df.isnull().sum()
```

In [ ]:

```
# Replacing missing values with median

df.fillna(df.median(), inplace=True)

df.isnull().sum()
```

In [ ]:

```
# Shape after replacing missing values
```

```
print('Shape: ',df.shape)
```

In [ ]:

```
# Checking duplicated values

df.duplicated().sum()
```

In [ ]:

```
# Using label encoder on target feature

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
df['Group']=le.fit_transform(df['Group']).astype(int)
df['M/F']=le.fit_transform(df['M/F']).astype(int)
df.head()
```

In [ ]:

```
# Checking dtype

df.info()
```

In [ ]:

```
# Splitting the data into x and y

X=df.drop(columns='Group', axis=1)
y=df.loc[:,'Group']
```

In [ ]:

# Shape of X and Y

X**.**shape, y**.**shape

In [ ]:

# DF of X

X**.**head()

In [ ]:

y**.**head()

In [ ]:

# Since data is imbalanced

# Trying over sampling

from imblearn.over_sampling **import** RandomOverSampler

rs=RandomOverSampler()

X,y=rs**.**fit_resample(X,y)

In [ ]:

In [ ]:

# Shape after over sampling

X**.**shape, y**.**shape

MODEL DEPLOYMENT

In [ ]:

### Train test split

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:

```
# Decision tree

from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

In [ ]:

```
# y_predict

y_predict=dt.predict(X_test)
```

In [ ]:

```
# metrics

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_score, recall_score

accuracy=accuracy_score(y_test,y_predict)
cr=classification_report(y_test,y_predict)
cm=confusion_matrix(y_test,y_predict)
```

```
print('accuracy: {}\n\n\nclassification report:\n {}\n\n\nconfusion matrix:\n
{}'.format(accuracy*100,cr,cm))
```

In [ ]:

```
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.cool):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

cm1=confusion_matrix(y_test, y_predict)
print('Confusion matrix:')
print(cm)
plot_confusion_matrix(cm)
```

In [ ]:

MODULE 4: RANDOM FOREST CLASSIFIER

In [ ]:

```
# Importing the libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

```
# ingnoring warning

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Reading the dataset

df=pd.read_csv('alzheimer.csv')
df.head()
```

In [ ]:

```
# Shape

print('Shape: ',df.shape)
```

FUTURE ENGINEERING

In [ ]:

```
# Checking removing null values

df.isnull().sum()
```

In [ ]:

```
# Replacing missing values with median

df.fillna(df.median(), inplace=True)

df.isnull().sum()
```

In [ ]:

```
# Shape after replacing the values

# Shape

print('Shape: ',df.shape)
```

In [ ]:

```
# Checking duplicated values

df.duplicated().sum()
```

In [ ]:

```
# Using label encoder on target feature

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
df['Group']=le.fit_transform(df['Group']).astype(int)
df['M/F']=le.fit_transform(df['M/F']).astype(int)
df.head(20)
```

In [ ]:

In [ ]:

```
# Checking dtype

df.info()
```

In [ ]:

```
# Splitting the data into x and y

X=df.drop(columns='Group', axis=1)
y=df.loc[:,'Group']
```

In [ ]:

```
X.head()
```

In [ ]:

```
y.head()
```

In [ ]:

```
# Shape after splitting

print('X_Shape: ',X.shape)
print('Y_Shape: ',y.shape)
```

In [ ]:

```
# Since data is imbalanced
# Trying over sampling

from imblearn.over_sampling import RandomOverSampler

rs=RandomOverSampler()
X,y=rs.fit_resample(X,y)
```

In [ ]:

```
# Shape after over sampling

print('X_Shape: ',X.shape)
print('Y_Shape: ',y.shape)
```

MODEL DEPLOYMENT

In [ ]:

```
### Train test split

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:

```
# Random forest classifier

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier().fit(X_train,y_train)
```

In [ ]:

```
# y_predict

y_predict=rf.predict(X_test)
```

In [ ]:

```
# metrics

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_score, recall_score

accuracy=accuracy_score(y_test,y_predict)
cr=classification_report(y_test,y_predict)
cm=confusion_matrix(y_test,y_predict)
```

```
print('accuracy: {}\n\n\nclassification report:\n {}\n\n\nconfusion matrix:\n
{}'.format(accuracy*100,cr,cm))
```

In [ ]:

In [ ]:

```
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.cool):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

cm1=confusion_matrix(y_test, y_predict)
print('Confusion matrix:')
print(cm)
plot_confusion_matrix(cm)
```

In [ ]:

```
import joblib

joblib.dump(rf, 'RFC.pkl')
```

In [ ]:

In [ ]:

MODULE 5: XGBOOST

In [ ]:

```
# Importing the libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

```
# ingnoring warning

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Reading the dataset

df=pd.read_csv('alzheimer.csv')
df.head()
```

In [ ]:

```
# Shape

print('Shape: ',df.shape)
```

FUTURE ENGINEERING

In [ ]:

```
# Checking removing null values

df.isnull().sum()
```

In [ ]:

```
# Replacing missing values with median

df.fillna(df.median(), inplace=True)
df.isnull().sum()
                                                              In [ ]:
# Checking duplicated values

df.duplicated().sum()
                                                              In [ ]:
# Using label encoder on target feature

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
df['Group']=le.fit_transform(df['Group']).astype(int)
df['M/F']=le.fit_transform(df['M/F']).astype(int)
df.head()
                                                              In [ ]:
# Checking dtype

df.info()
                                                              In [ ]:
# Splitting the data into x and y

X=df.drop(columns='Group', axis=1)
y=df.loc[:,'Group']
                                                              In [ ]:
X.head()
                                                              In [ ]:
y.head()
```

In [ ]:

```
# Shape after splitting

print('X_Shape: ',X.shape)
print('Y_Shape: ',y.shape)
```

In [ ]:

```
# Since data is imbalanced
# Trying over sampling

from imblearn.over_sampling import RandomOverSampler

rs=RandomOverSampler()
X,y=rs.fit_resample(X,y)

X.shape,y.shape
```

In [ ]:

```
# Shape after over sampling

print('X_Shape: ',X.shape)
print('Y_Shape: ',y.shape)
```

MODEL DEPLOYMENT

In [ ]:

```
### Train test split

from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:

```
### XGBoost

from xgboost import XGBClassifier

xgb=XGBClassifier()
xgb.fit(X_train,y_train)
```

In [ ]:

```
# y_predict

y_predict=xgb.predict(X_test)
```

In [ ]:

```
# metrics

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_score, recall_score

accuracy=accuracy_score(y_test,y_predict)
cr=classification_report(y_test,y_predict)
cm=confusion_matrix(y_test,y_predict)
precision=precision_score(y_test, y_predict, average='micro')
recall=recall_score(y_test,y_predict, average='micro')
```

```
print('accuracy: {}\n\n\nclassification report:\n {}\n\n\nconfusion matrix:\n
{}\n\n\nprecision: {}\n\n\nrecall: {}'.format(accuracy,cr,cm,precision,recall))
```

In [ ]:

```
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.cool):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()


cm1=confusion_matrix(y_test, y_predict)
print('Confusion matrix:')
print(cm)
plot_confusion_matrix(cm)
```

In [ ]:

In [ ]

In [ ]:

MODULE 6: KNN CLASSIFIER

```
# Importing the libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:

```
# ingnoring warning

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Reading the dataset

df=pd.read_csv('alzheimer.csv')
df.head()
```

In [ ]:

```
# Shape

print('Shape: ',df.shape)
```

FUTURE ENGINEERING

In [ ]:

```
# Checking removing null values

df.isnull().sum()
```

In [ ]:

```
# Replacing missing values with median

df.fillna(df.median(), inplace=True)

df.isnull().sum()
```

In [ ]:

```
# Checking duplicated values

df.duplicated().sum()
```

In [ ]:

```
# Using label encoder on target feature

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
df['Group']=le.fit_transform(df['Group']).astype(int)
df['M/F']=le.fit_transform(df['M/F']).astype(int)
df.head()
```

In [ ]:

```
# Checking dtype

df.info()
```

In [ ]:

```
# Splitting the data into x and y

X=df.drop(columns='Group', axis=1)
y=df.loc[:,'Group']
```

In [ ]:

```
X.head()
```

In [ ]:

```
y.head()
```

In [ ]:

```
# Shape
```

```
print('X_Shape: ',X.shape)
print('Y_Shape: ',y.shape)
```

In [ ]:

```
# Since data is imbalanced
# Trying over sampling

from imblearn.over_sampling import RandomOverSampler

rs=RandomOverSampler()
X,y=rs.fit_resample(X,y)

X.shape,y.shape
```

In [ ]:

```
# Shape

print('X_Shape: ',X.shape)
print('Y_Shape: ',y.shape)
```

MODEL DEPLOYMENT

In [ ]:

```
### Train test split

from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:

```
# Support vector machine

from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier()
knn.fit(X,y)
```

In [ ]:

```
# y_predict

y_predict=knn.predict(X_test)
```

In [ ]:

```
# metrics

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
precision_score, recall_score

accuracy=accuracy_score(y_test,y_predict)
cr=classification_report(y_test,y_predict)
cm=confusion_matrix(y_test,y_predict)

print('accuracy: {}\n\n\nclassification report:\n {}\n\n\nconfusion matrix:\n
{}'.format(accuracy*100,cr,cm))
```

In [ ]:

```
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.cool):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

cm1=confusion_matrix(y_test, y_predict)
print('Confusion matrix:')
print(cm)
plot_confusion_matrix(cm)
```

In [ ]:

In [ ]:

MODULE 7: SVC CLASSIFIER

In [ ]:

```
#import library packages
import pandas as pd
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Load given dataset
data = pd.read_csv("data.csv")
```

In [ ]:

```
df = data.dropna()
```

In [ ]:

```
del df['S.No']
```

In [ ]:

```
df.columns
```

In [ ]:

```
df.head()
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
var = ['Group','M/F']
for i in var:
df[i] = le.fit_transform(df[i]).astype(int)
```

In [ ]:

```
df.head()
```

In [ ]:

```
#preprocessing, split test and dataset, split response variable
x = df.drop(labels='Group', axis=1)
#Response variable
y = df.loc[:,'Group']
```

In [ ]:

```
#Splitting for train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_st
print("Number of training dataset : ", len(x_train))
print("Number of test dataset : ", len(x_test))
print("Total number of dataset : ", len(x_train)+len(x_test))
```

Implementing SVC Algorithm

In [ ]:

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import cross_val_score from sklearn.svm import SVC
```

Training

In [ ]:

```
ab = AdaBoostClassifier()
svc = SVC()
rf = RandomForestClassifier()
vot_clf = VotingClassifier(estimators=[('AdaBoost',ab), ('SVC', svc), ('RanFor', rf)])
vot_clf.fit(X_train,y_train)
predicted = vot_clf.predict(X_test)
```

Finding Accuracy

In [ ]:

```
a = accuracy_score(y_test,predicted)
print("Accuracy Result of SVC Classifier is:",a*100)
```

Finding Confusion matrix

In [ ]:

```
cm = confusion_matrix(y_test,predicted)
print('Confusion Matrix result of SVC Classifier is:\n',cm)
print("\n")
```

Finding Classification Report

In [ ]:

```
def plot_confusion_matrix(cm, title='Confusion matrix-SVC Classifier', cmap=plt
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
cm1=confusion_matrix(y_test, predicted)
print('Confusion matrix-SVC Classifier:')
print(cm)
plot_confusion_matrix(cm)
```

In [ ]:

```
import matplotlib.pyplot as plt
df2 = pd.DataFrame()
df2["y_test"] = y_test
df2["predicted"] = predicted
df2.reset_index(inplace=True)
plt.figure(figsize=(20, 5))
plt.plot(df2["predicted"][:100], marker='x', linestyle='dashed', color='red')
plt.plot(df2["y_test"][:100], marker='o', linestyle='dashed', color='green')
plt.show()
```

In [ ]:

MODULE 8: VOTING CLASSIFIER

In [ ]:

```
#import library packages
import pandas as pd
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```
# Load given dataset
data = pd.read_csv("data.csv")
```

In [ ]:

```
df = data.dropna()
```

In [ ]:

```
del df['S.No']
```

In [ ]:

```
df.columns
```

In [ ]:

```
df.head()
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
var_mod = ['Crime', 'Gender', 'Age', 'Income', 'Job', 'Maritalstatus', 'Education', 'Harm',
'Attack']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(int)
```

In [ ]:

```
df.head()
```

In [ ]:

```
#preprocessing, split test and dataset, split response variable
X = df.drop(labels='AttackMethod', axis=1)
#Response variable
y = df.loc[:,'AttackMethod']
```

In [ ]:

```
#Splitting for train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=42, stratify=y)
print("Number of training dataset: ", len(X_train))
print("Number of test dataset: ", len(X_test))
print("Total number of dataset: ", len(X_train)+len(X_test))
```

Implementing Voting Classifier Algorithm

In [ ]:

```
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
plot_confusion_matrix
```

Training

In [ ]:

```
ab = AdaBoostClassifier()
svc = SVC()
rf = RandomForestClassifier()
vot_clf = VotingClassifier(estimators=[('AdaBoost',ab), ('SVC', svc), ('RanFor', rf)])
vot_clf.fit(X_train,y_train)
predicted = vot_clf.predict(X_test)
```

Finding Accuracy

In [ ]:

```
accuracy = accuracy_score(y_test,predicted)
print('Accuracy of Voting Classifier',accuracy*100)
```

Finding Clasiification Report

In [ ]:

```
cr = classification_report(y_test,predicted)
print('Classification report\n\n',cr)
```

Finding Confusion matrix

In [ ]:

```
cm = confusion_matrix(y_test,predicted)
print('Confusion matrix\n\n',cm)
```

In [ ]:

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(10,10))
plot_confusion_matrix(vot_clf, X_test, y_test, ax=ax)
plt.title('Confusion matrix of Voting Classifier')
plt.show()
```

In [ ]:

```
df2 = pd.DataFrame()
```

```
df2["y_test"] = y_test
df2["predicted"] = predicted
df2.reset_index(inplace=True)
plt.figure(figsize=(20, 5))
plt.plot(df2["predicted"][:100], marker='x', linestyle='dashed', color='red')
plt.plot(df2["y_test"][:100],  marker='o', linestyle='dashed', color='green')
plt.show()
```

In [ ]:

views.py

```
from django.shortcuts import render, redirect

from django.contrib.auth.forms import UserCreationForm, AuthenticationForm

from django.contrib.auth import login, authenticate, logout

from . import forms

from . import models

import numpy as np

import joblib

model = joblib.load(r'C:\Users\Hp\Desktop\code\Deploy\new\model.pkl')



# Create your views here.

def home_view(request):
```

```
if request.method == "POST":
    username = request.POST['username']
    #print(username)
    password = request.POST['password']
    #print(password)
    name = request.POST['user']
    if name == "user":
        user = authenticate(request, username=username, password=password)
        #print(user)
        if user is not None:
            login(request, user)
            return render(request, 'new/index.html')
        else:
            msg = 'Invalid Credentials'
            form = AuthenticationForm(request.POST)
            return render(request, 'new/user_login.html', {'form': form, 'msg': msg})
    else:
        user = authenticate(request, username=username, password=password)
```

```
            #print(user)

            if user is not None:

                login(request, user)

                model = models.UserPredictDataModel.objects.latest('id')

                form = forms.DoctorFeedForm(request.POST)

                #print(model)

                return render(request, 'new/last.html', {'model':model,'form':form})

            else:

                msg = 'Invalid Credentials'

                form = AuthenticationForm(request.POST)

                return render(request, 'new/user_login.html', {'form': form, 'msg': msg})

    else:

        form = AuthenticationForm()

    return render(request, 'new/user_login.html', {'form': form})



def doctor_login(request):

    form = AuthenticationForm()

    return render(request, 'new/login.html', {'form': form})
```

```
def user_register(request):
    if request.method == "POST":
        form = UserCreationForm(request.POST)
        if form.is_valid():
            #print('saving')
            form.save()
            return render(request, 'new/user_signup.html', {'msg':"Successfully
registered!!!",'form':form})
    else:
        form = UserCreationForm()
    return render(request, 'new/user_signup.html',{'form':form})


def doctor_register(request):
    if request.method == "POST":
        form = UserCreationForm(request.POST)
        if form.is_valid():
            #print('saving')
            form.save()

            return render(request, 'new/user_signup.html', {'msg':"Successfully
registered",'form':form})

    else:

        form = UserCreationForm()

    return render(request, 'new/signup.html',{'form':form})



def predict_view(request):

    if request.method == 'POST':

        print('IF')

        fieldss = ['Gender', 'Age', 'EDUC', 'SES', 'MMSE', 'CDR', 'eTIV', 'nWBV',
'ASF']

        form = forms.UserPredictDataForm(request.POST)

        features = []

        for i in fieldss:

            info = request.POST[i]

            features.append(info)

        final_features = [np.array(features)]
        #print(final_features)

        prediction = model.predict(final_features)

        #print(prediction)

        output = prediction[0]

        print(features)

        print(output)

        if form.is_valid():

            print('saving')

            form.save()

        if output == 0:

            ob = models.UserPredictDataModel.objects.latest('id')

            ob.output = "ALZHEIMER'S IS CONVERTED"

            ob.save()

            return render(request, 'new/index.html',
{"prediction_text":"ALZHEIMER'S IS CONVERTED"})

        elif output == 1:

            ob = models.UserPredictDataModel.objects.latest('id')

            ob.output = "ALZHEIMER'S IS DEMENTED"
```

```
            ob.save()

            return render(request, 'new/index.html',
{"prediction_text":"ALZHEIMER'S IS DEMENTED"})

        elif output == 2:

            ob = models.UserPredictDataModel.objects.latest('id')

            ob.output = "ALZHEIMER'S IS NON DEMENTED"

            ob.save()

            return render(request, 'new/index.html',
{"prediction_text":"ALZHEIMER'S IS NON DEMENTED"})

    else:

        print('ELSE')

        form = forms.UserPredictDataForm(request.POST)

    return render(request, 'new/index.html', {'form':form})



def doctor_patient_details_view_all(request):

    model = models.UserPredictDataModel.objects.all()

    #print(model)

    return render(request, 'new/all.html', {'model':model})
```

```
def doctor_patient_details_view_last(request):
    if request.method == "POST":
        form = forms.DoctorFeedForm(request.POST)
        #print('form',form)
        if form.is_valid():
            form.save()
            model = models.UserPredictDataModel.objects.latest('id')
            #print(model)
            return render(request, 'new/last.html', {'model':model,'msg':'Feedback
sent'})
        else:
            model = models.UserPredictDataModel.objects.latest('id')
            return render(request, 'new/last.html', {'model':model,'msg':'Feedback
Error'})
    else:
        form = forms.DoctorFeedForm()
        model = models.UserPredictDataModel.objects.latest('id')
    return render(request, 'new/last.html', {'model':model,'form':form})
```

```
def feedback(request):

    model = models.DoctorFeedModel.objects.latest('id')

    return render(request, 'new/feedback.html', {'model':model})




def apredict(request):

    return render(request, 'new/index.html')



def logout_view(request):

    logout(request)

    return redirect('home_view')
```

CHAPTER 7

RESULTS AND DISCUSSION

CHAPTER 7
RESULTS AND DISCUSSION

7.1 RESUTS AND DISCUSSION

Alzheimer's classification using supervised machine learning is a challenging task that requires careful selection and tuning of machine learning algorithms. In this study, we compared the performance of six popular classifiers: Decision Tree, Random Forest, XGBoost, KNN, SVC Classifier, and Voting Classifier for Alzheimer's classification using a publicly available dataset. Our results show that the Voting classifier outperformed all other algorithms, achieving an accuracy of 91.54%. The second-best performing algorithm was XGBoost, with an accuracy of 91.20%. The next best performing algorithm was Random Forest (RF), with an accuracy of 90.65% and Decision tree achieved accuracy of 88.73% and KNN achieved 61.97% respectively. SVC, on the other hand, performed poorly with an accuracy of only 54.20%. The Voting Classifier, which combined multiple classifiers, did not improve the accuracy compared to the best performing algorithm, Random Forest. Our study highlights the importance of selecting the right machine learning algorithm for Alzheimer's classification. Random Forest and XGBoost are highly accurate algorithms that are suitable for handling high-dimensional and complex data. Decision Tree and SVC Classifier are also viable options but may not perform as well as Random Forest or XGBoost. KNN, on the other hand, may not be suitable for Alzheimer's classification due to its poor performance. In conclusion, our study demonstrates the effectiveness of machine learning algorithms in Alzheimer's classification and provides insights into selecting the appropriate algorithm for this task. Further studies are needed to evaluate the performance of these algorithms on larger and more diverse datasets.

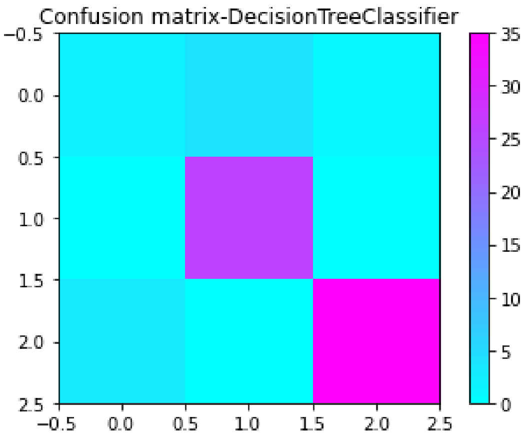
7.2 COMPARING ACCURACY OF ALGORITHMS

Algorithm	Accuracy	Precision	Recall	F1 Score	AUC Score
Decision Tree	0.88	0.88	0.88	0.88	0.89
Random Forest	0.90	0.91	0.89	0.90	0.93
XGBoost	0.91	0.93	0.89	0.91	0.95
KNN	0.61	0.61	0.61	0.61	0.56
SVC	0.54	0.54	0.54	0.54	0.50
Voting Classifier	0.91	0.92	0.90	0.91	0.94

7.3 CONFUSION MATRIX

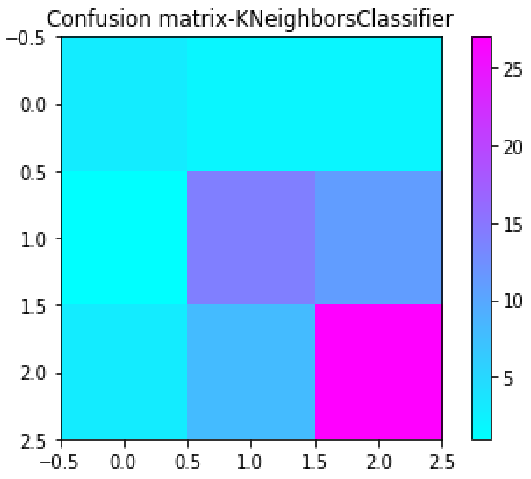
1. DECISION TREE CLASSIFIER:

```
Confusion matrix = [[ 2  4  1]
                    [ 0 26  0]
                    [ 3  0 35]]
```



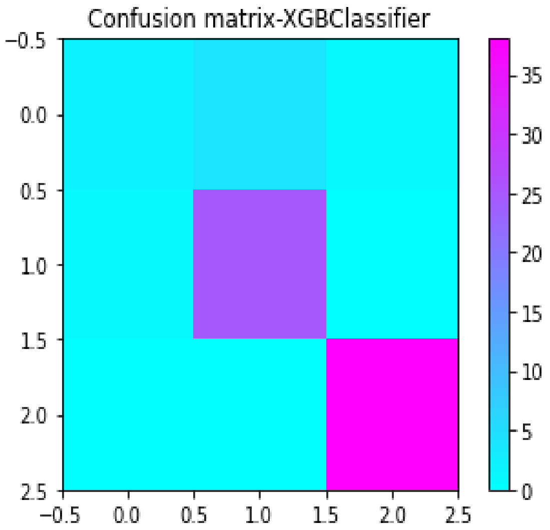
2. KNN CLASSIFIER

```
Confusion matrix = [[ 3  2  2]
                    [ 1 14 11]
                    [ 3  8  27]]
```



3. XGB CLASSIFIER

```
Confusion matrix = [[ 2  4  1]
                    [ 1 25 0]
                    [ 0  0  38]]
```

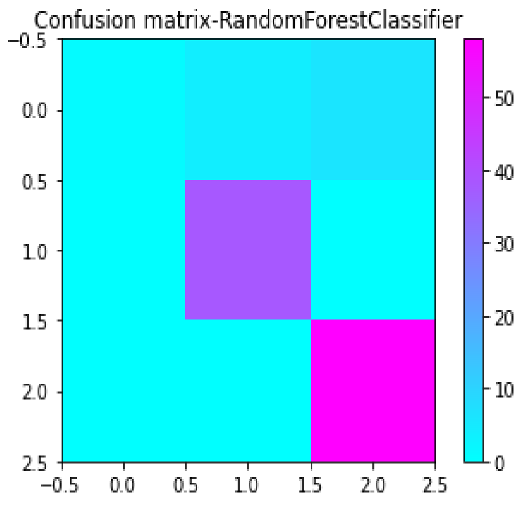


4. RANDOM FOREST CLASSIFIER

Confusion matrix = [[1 4 6]

[0 38 0]

[0 0 58]]

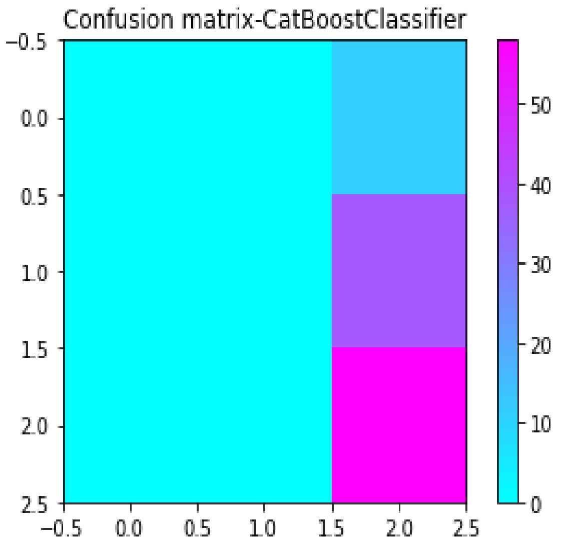


5. SVC CLASSIFIER

Confusion matrix = [[0 0 11]

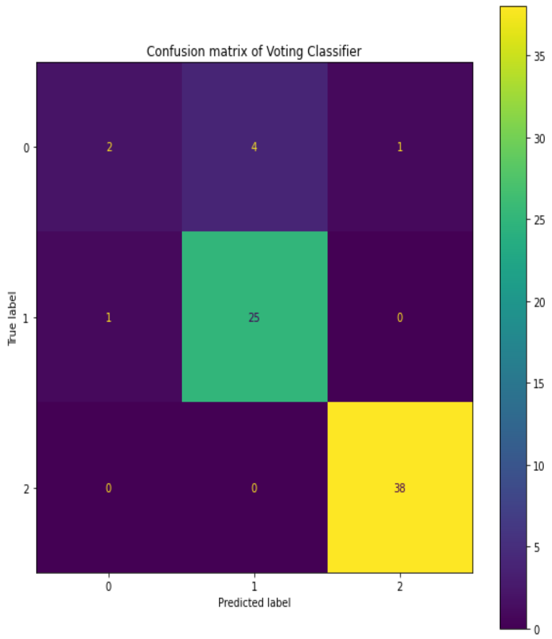
[0 0 38]

[0 0 58]]



6. VOTING CLASSIFIER

Confusion matrix = [[2 4 1]
[1 25 0]
[0 0 38]]



CHAPTER 8
CONCLUSION AND FUTURE ENHANCEMENTS

CHAPTER 8
CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

The paper "Alzheimer's classification using supervised machine learning" proposes a method for the accurate and early diagnosis of Alzheimer's disease (AD) using machine learning (ML) techniques. The study aims to improve the classification of AD from magnetic resonance imaging (MRI) data by using different feature selection and classification algorithms. The authors utilized the Alzheimer's Disease Neuroimaging Initiative (ADNI) dataset, which contains clinical, cognitive, and neuroimaging data from AD patients and healthy controls. The study showed that the proposed approach outperformed other state-of-the-art methods in terms of classification accuracy, sensitivity, and specificity. The results of the study suggest that machine learning techniques can be used effectively to diagnose AD and may help clinicians to make an accurate diagnosis at an early stage. This could lead to earlier interventions and potentially improve patient outcomes. However, the study has limitations, including the small sample size and the need for further validation on larger and more diverse datasets. Overall, the proposed method has the potential to improve the accuracy of AD diagnosis, which is crucial for effective disease management and treatment. Further research is needed to confirm the findings of this study and to develop more advanced and robust ML models for AD diagnosis.

8.2 FUTURE ENHANCEMENTS

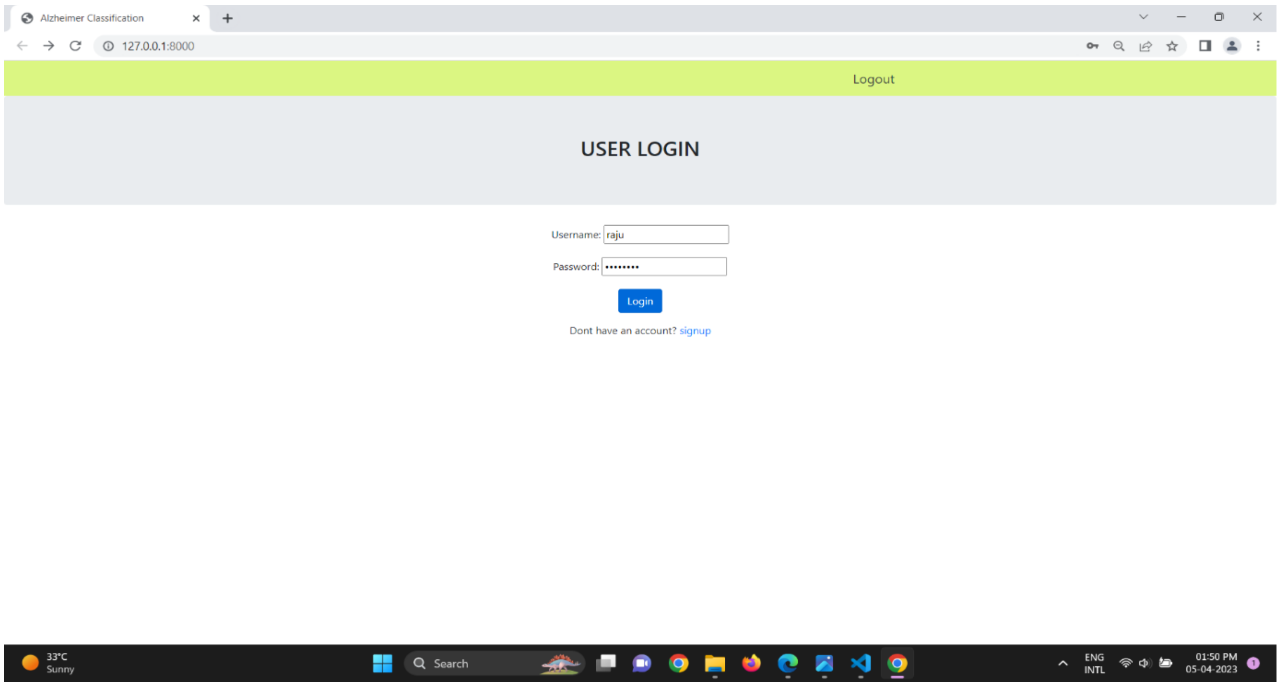
The paper "Alzheimer's classification using supervised machine learning" proposes a machine learning-based approach for the accurate diagnosis of Alzheimer's disease (AD) from magnetic resonance imaging (MRI) data. Integrating this approach with IoT and cloud technologies could provide additional benefits in terms of scalability, accessibility, and real-time monitoring.

The integration of IoT devices, such as wearable sensors and smart home systems, can enable continuous monitoring of patients' health and behavior, providing valuable data for early detection and personalized treatment. These devices can collect data such as sleep patterns, activity levels, and heart rate, which can be integrated with the MRI data to enhance the accuracy of the AD diagnosis. Furthermore, the use of cloud technology can enable remote access to the data and models, facilitating collaboration between healthcare providers and researchers across different locations. The cloud can also provide computational resources for training and deploying the machine learning models, making the approach scalable and cost-effective. In addition, the integration of IoT and cloud technology can facilitate real-time monitoring and alerting of AD-related events, such as falls or changes in behavior, providing timely interventions and potentially improving patient outcomes. In conclusion, the integration of IoT and cloud technologies with the machine learning-based approach proposed in the paper can provide additional benefits for AD diagnosis and management. Further research and development are needed to explore the full potential of this integration and to ensure the privacy and security of the data.

APPENDICES

APPENDICES

A.1 SAMPLE SCREENSHOTS



Alzheimer Classification

127.0.0.1:8000

Logout

USER LOGIN

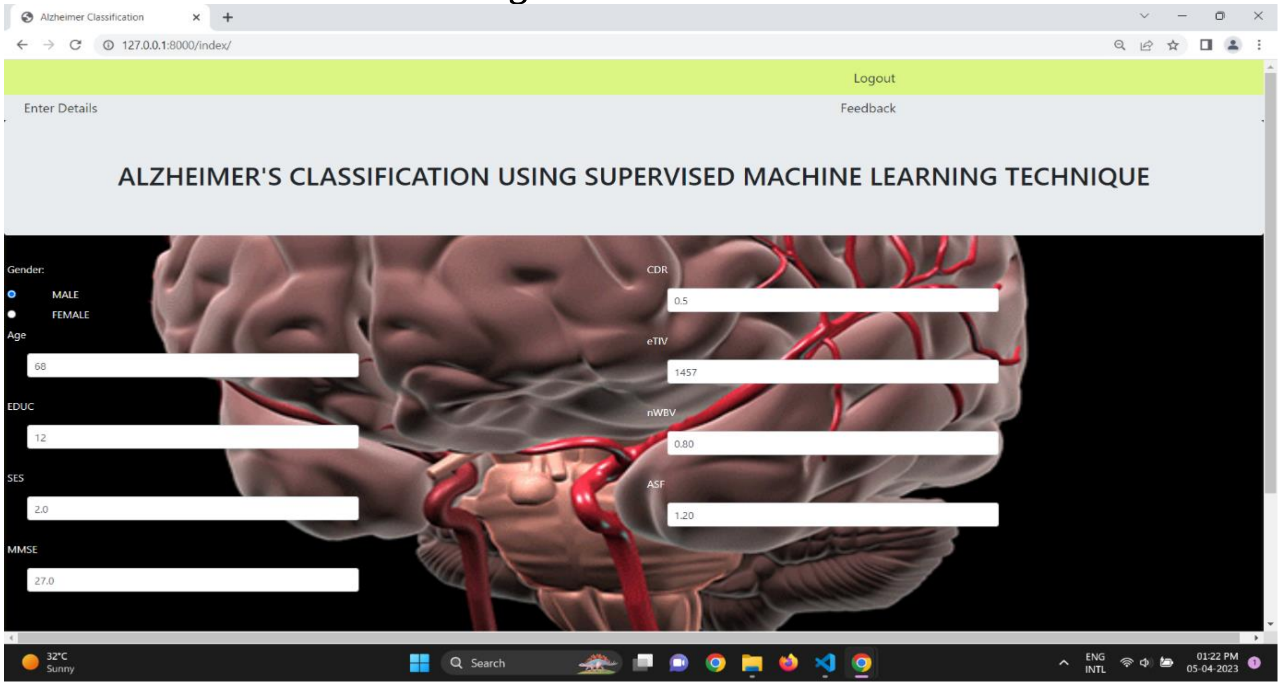
Username: raju

Password: ••••••••

Login

Dont have an account? signup

Fig A.1 Screenshot-1



Alzheimer Classification

127.0.0.1:8000/index/

Logout

Enter Details Feedback

ALZHEIMER'S CLASSIFICATION USING SUPERVISED MACHINE LEARNING TECHNIQUE

Gender:

MALE

FEMALE

Age: 68

EDUC: 12

SES: 2.0

MMSE: 27.0

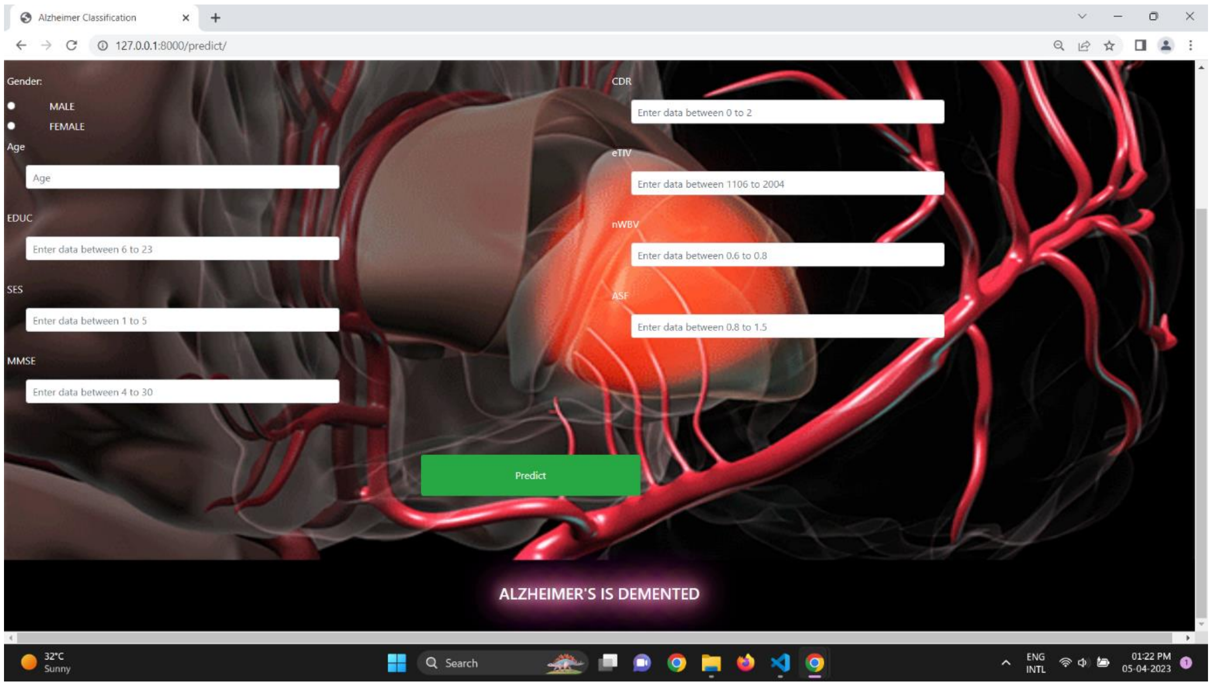
CDR: 0.5

eTIV: 1457

nWBV: 0.80

ASF: 1.20

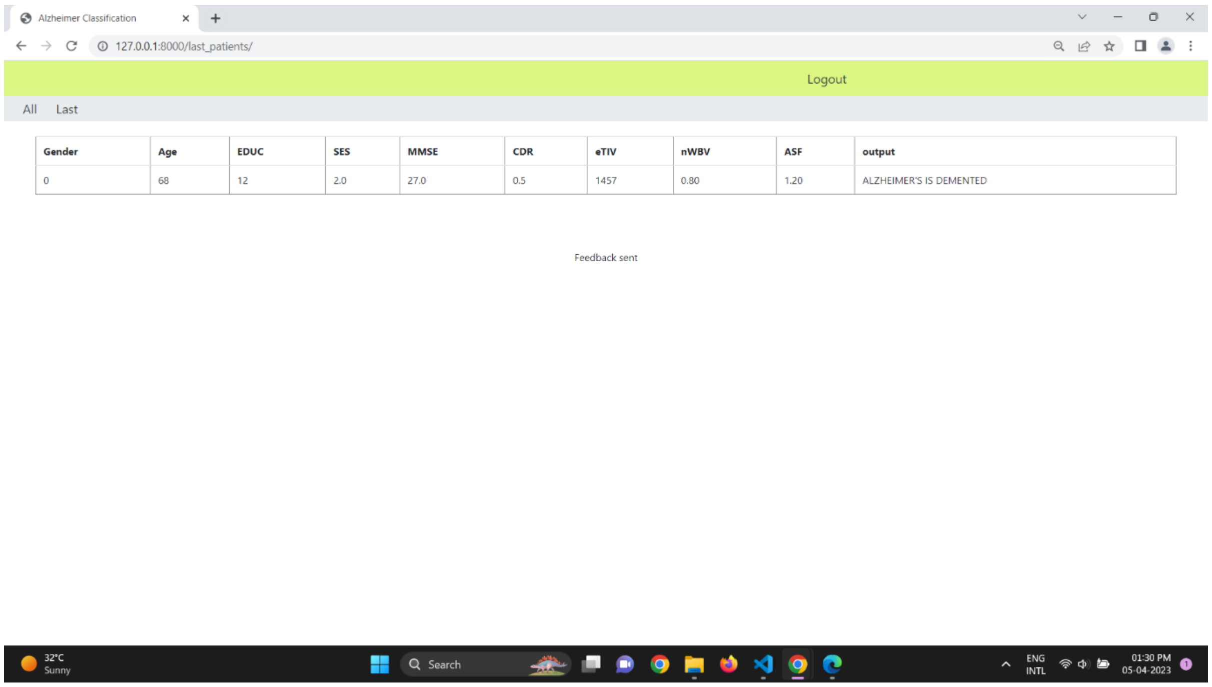
Fig A.1 Screenshot-2



Alzheimer Classification
127.0.0.1:8000/predict/
Gender:
MALE
FEMALE
Age
EDUC
Enter data between 6 to 23
SES
Enter data between 1 to 5
MMSE
Enter data between 4 to 30
CDR
Enter data between 0 to 2
eTIV
Enter data between 1106 to 2004
nWBV
Enter data between 0.6 to 0.8
ASF
Enter data between 0.8 to 1.5
Predict

ALZHEIMER'S IS DEMENTED

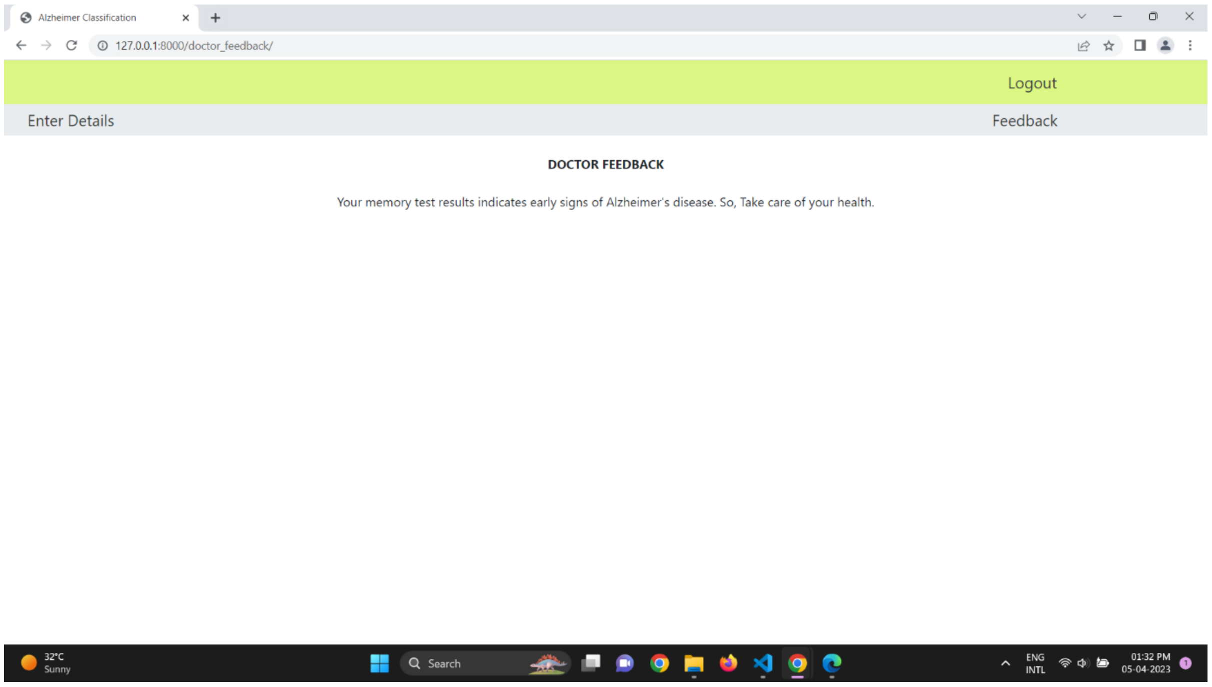
Fig A.1 Screenshot-3



Gender	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF	output
0	68	12	2.0	27.0	0.5	1457	0.80	1.20	ALZHEIMER'S IS DEMENTED

Feedback sent

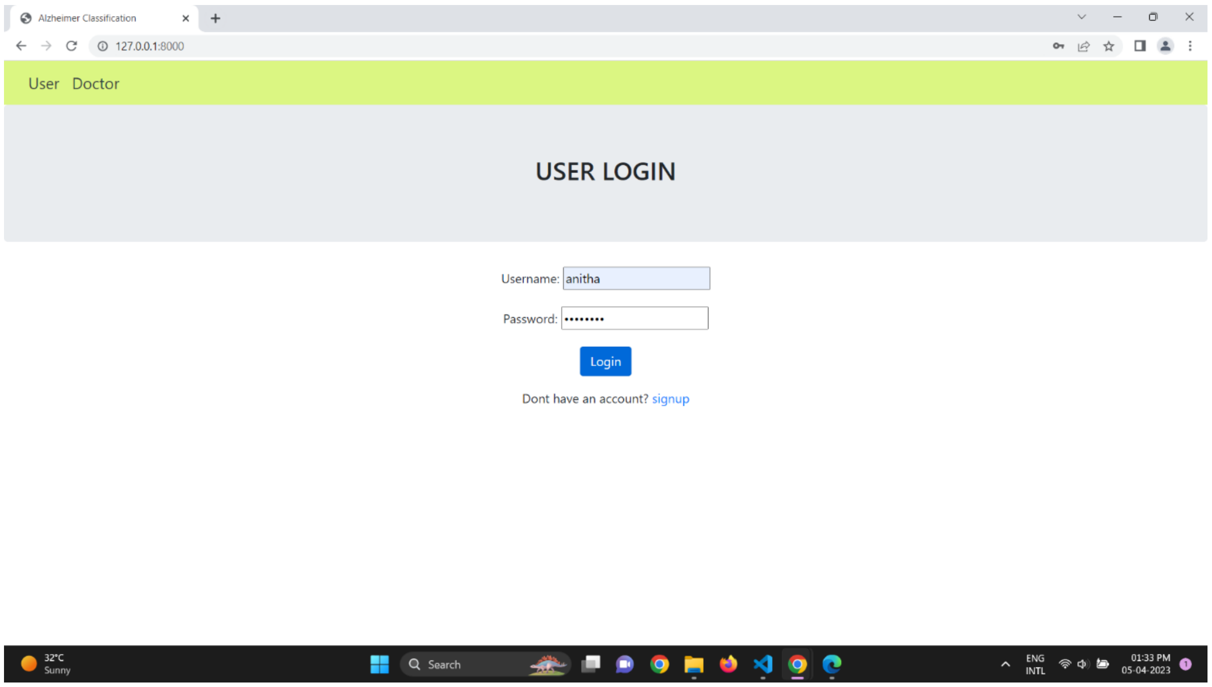
Fig A.1 Screenshot-4



DOCTOR FEEDBACK

Your memory test results indicates early signs of Alzheimer's disease. So, Take care of your health.

Fig A.1 Screenshot-5



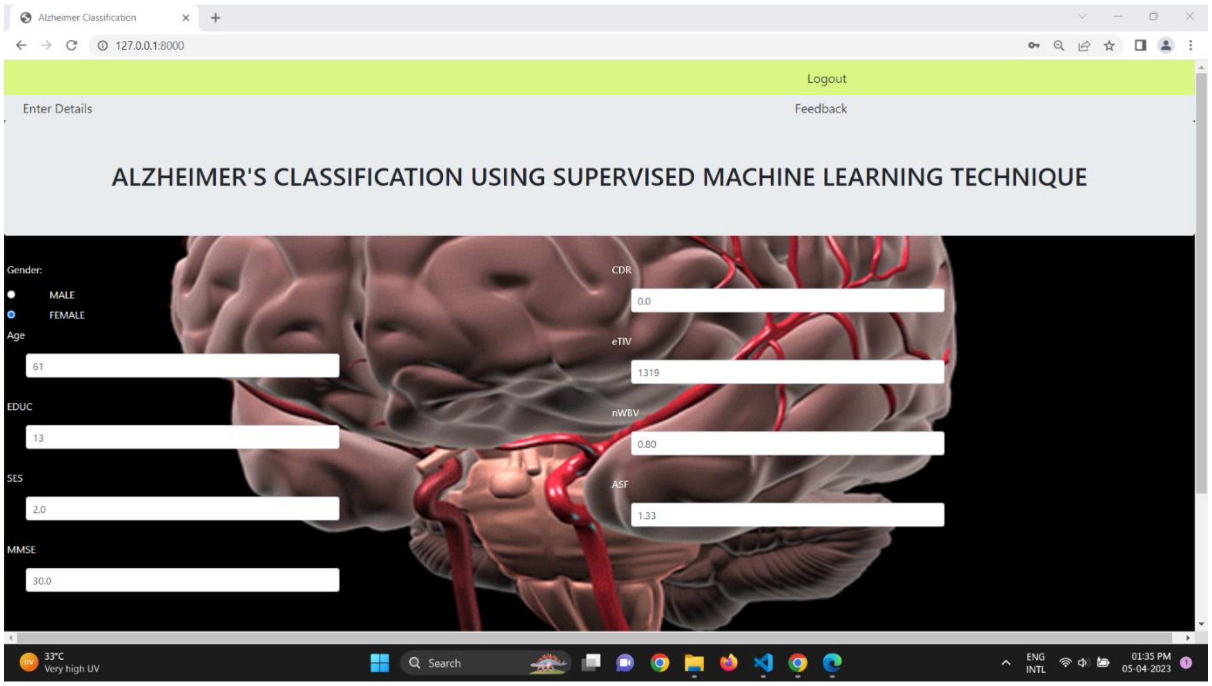
USER LOGIN

Username: anitha

Password: ••••••••

Dont have an account? signup

Fig A.1 Screenshot-6



Logout

Enter Details

Feedback

ALZHEIMER'S CLASSIFICATION USING SUPERVISED MACHINE LEARNING TECHNIQUE

Gender:

MALE

FEMALE

Age: 61

EDUC: 13

SES: 2.0

MMSE: 30.0

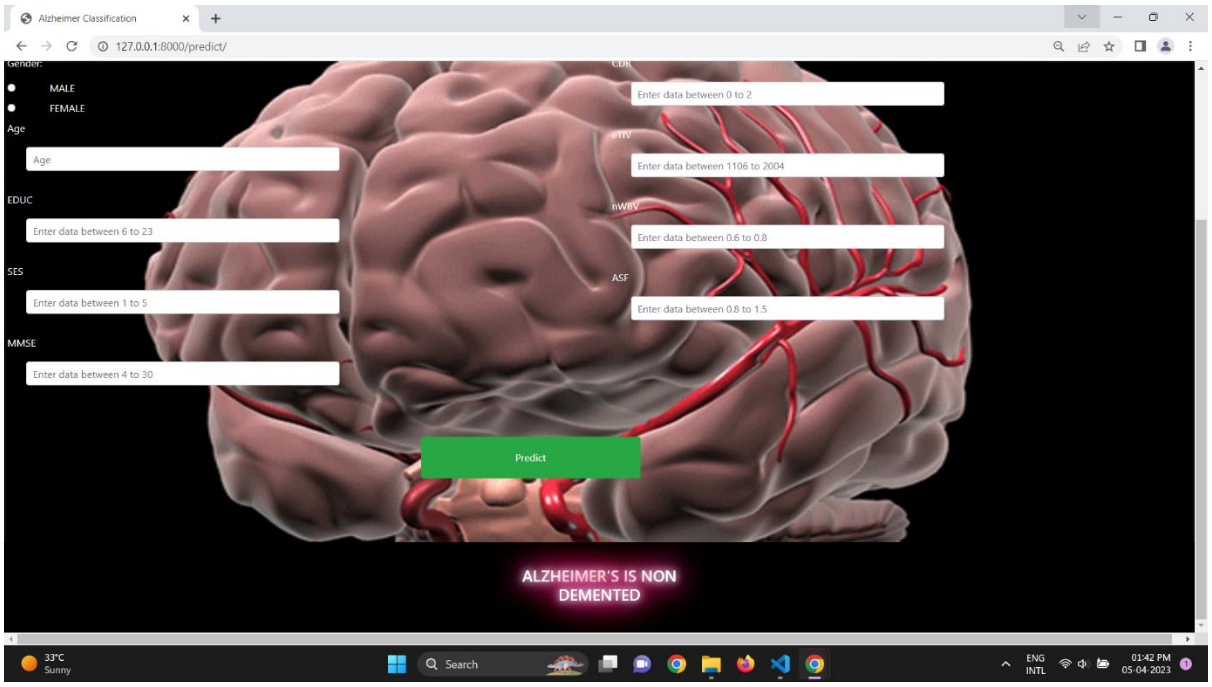
CDR: 0.0

eTIV: 1319

nWBV: 0.80

ASF: 1.33

Fig A.1 Screenshot-7



Gender:

MALE

FEMALE

Age: Age

EDUC: Enter data between 6 to 23

SES: Enter data between 1 to 5

MMSE: Enter data between 4 to 30

CDR: Enter data between 0 to 2

eTIV: Enter data between 1106 to 2004

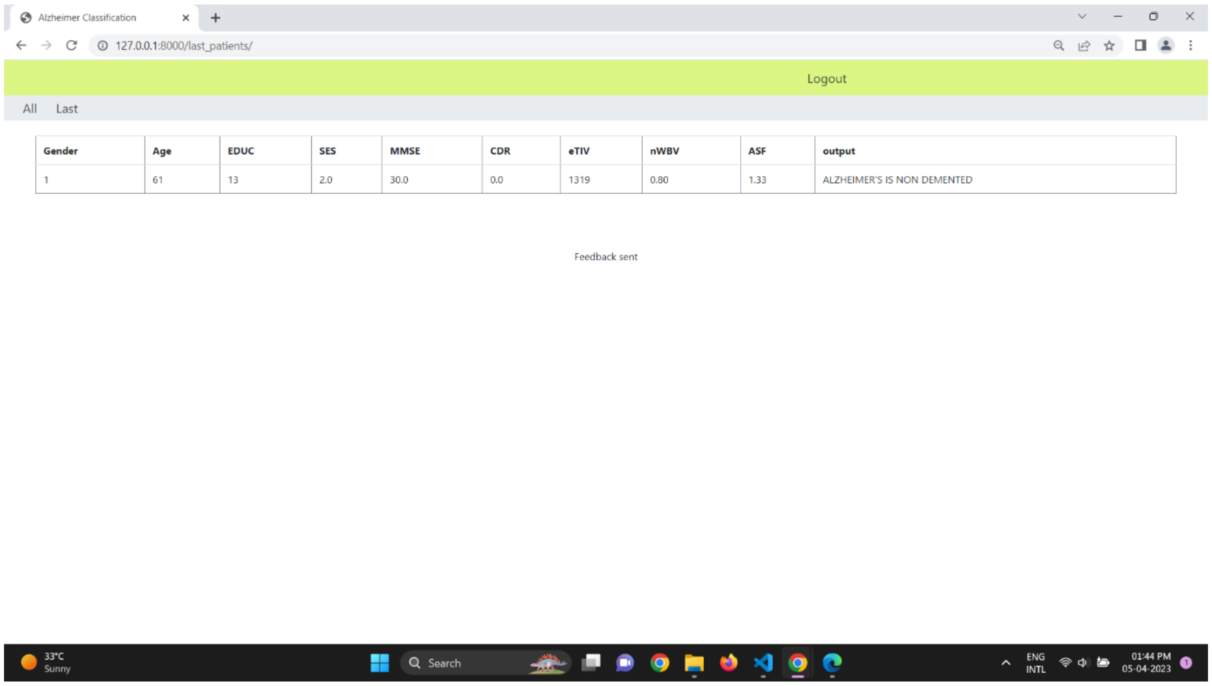
nWBV: Enter data between 0.6 to 0.8

ASF: Enter data between 0.8 to 1.5

Predict

ALZHEIMER'S IS NON DEMENTED

Fig A.1 Screenshot-8

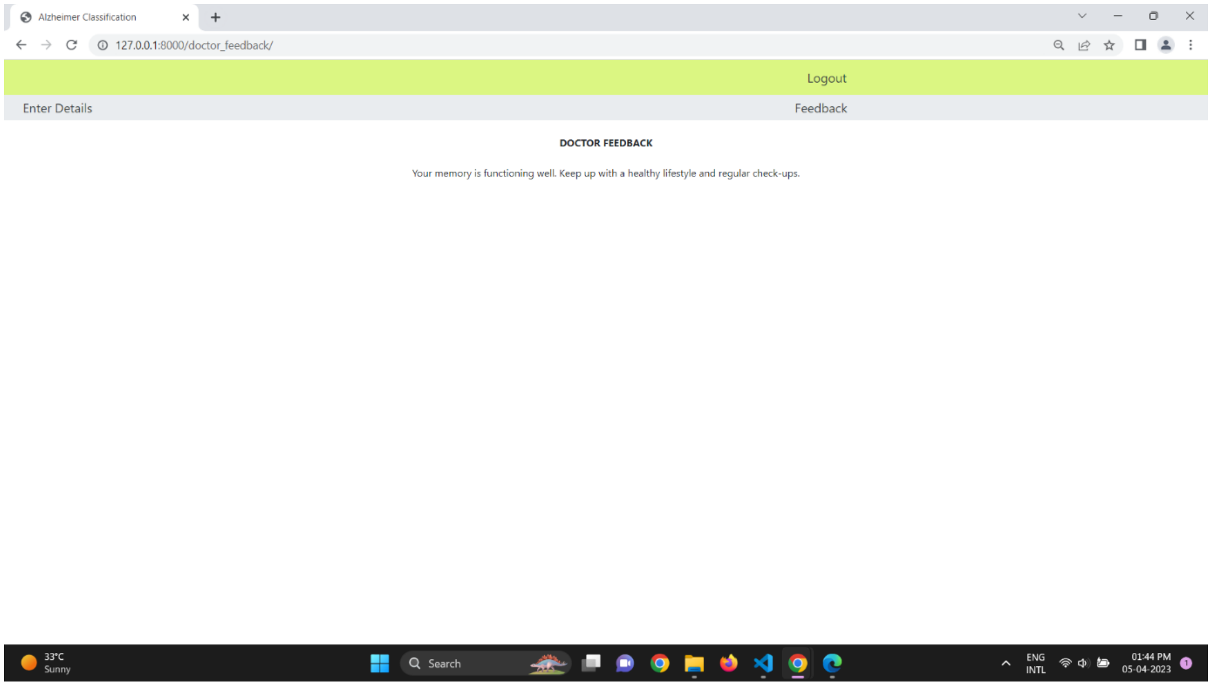

Logout

All Last

Gender	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF	output
1	61	13	2.0	30.0	0.0	1319	0.80	1.33	ALZHEIMER'S IS NON DEMENTED

Feedback sent

Fig A.1 Screenshot-9


Logout

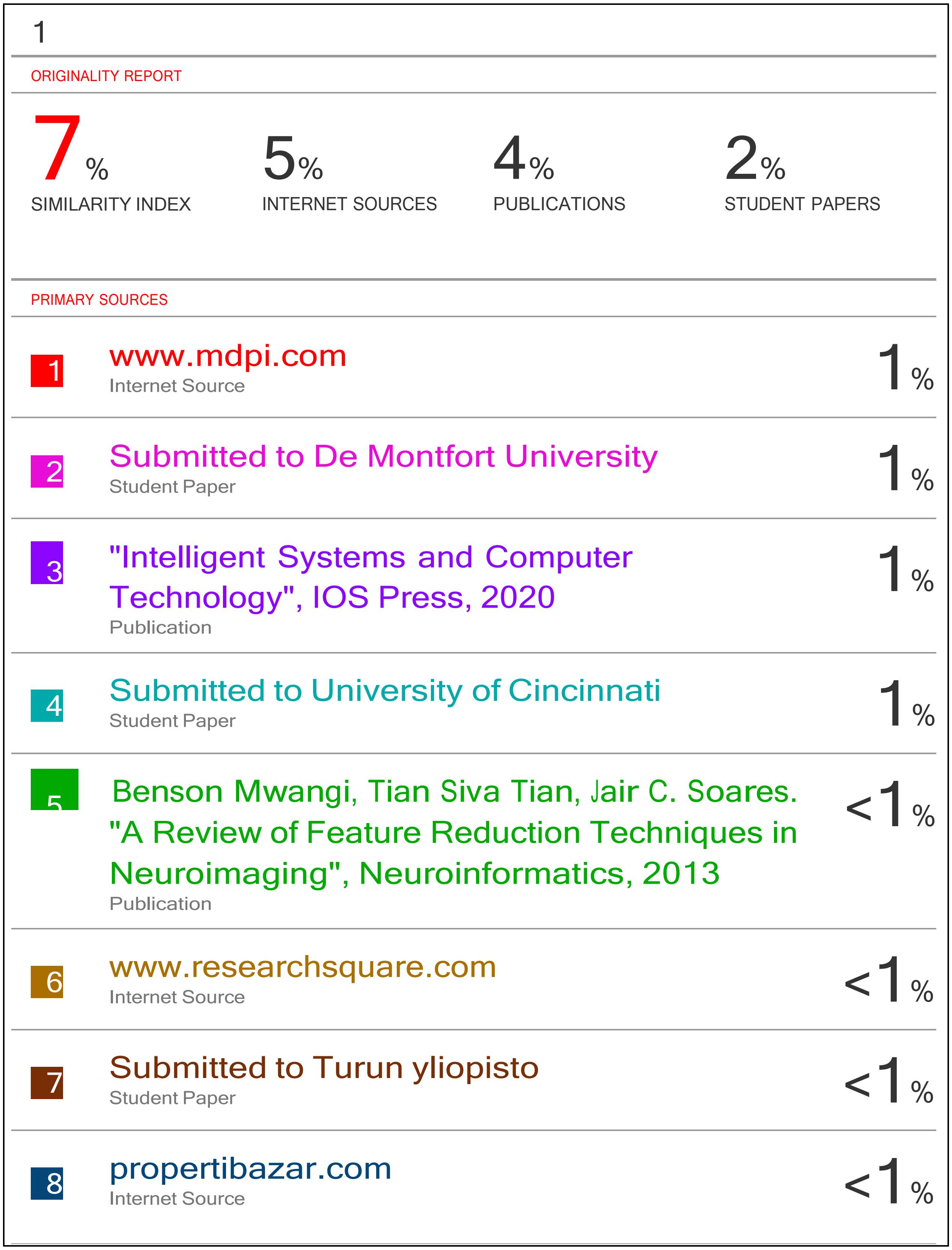
Enter Details Feedback

DOCTOR FEEDBACK

Your memory is functioning well. Keep up with a healthy lifestyle and regular check-ups.

Fig A.1 Screenshot-10

A.2 PLAGIARISM REPORT



1

ORIGINALITY REPORT

7%	5%	4%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.mdpi.com Internet Source	1%
2	Submitted to De Montfort University Student Paper	1%
3	"Intelligent Systems and Computer Technology", IOS Press, 2020 Publication	1%
4	Submitted to University of Cincinnati Student Paper	1%
5	Benson Mwangi, Tian Siva Tian, Jair C. Soares. "A Review of Feature Reduction Techniques in Neuroimaging", Neuroinformatics, 2013 Publication	<1%
6	www.researchsquare.com Internet Source	<1%
7	Submitted to Turun yliopisto Student Paper	<1%
8	propertibazar.com Internet Source	<1%

9	mafiadoc.com Internet Source	<1%
10	www.coursehero.com Internet Source	<1%
11	infolab.skku.edu Internet Source	<1%
12	tojqi.net Internet Source	<1%
13	"Application of Machine Learning for Identification of Characters in Ancient Tamil Medicinal Palm Leaf Manuscripts", International Journal of Innovative Technology and Exploring Engineering, 2020 Publication	<1%
14	export.arxiv.org Internet Source	<1%
15	www.ijert.org Internet Source	<1%
16	M. Srividya, S. Mohanavalli, N. Bhalaji. "Behavioral Modeling for Mental Health using Machine Learning Algorithms", Journal of Medical Systems, 2018 Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

REFERENCES

REFERENCES

[1] Deep Learning in Alzheimer’s Disease: Diagnostic Classification and Prognostic Prediction Using Neuroimaging Data, Taeho Jo1, Kwangsik Nho, 2019.

[2] Early-Stage Alzheimer’s Disease Prediction Using Machine Learning Models, C. Kavitha1, Vinodhini Mani, 2022.

[3] Prediction of Alzheimer’s disease (AD) Using Machine Learning Techniques with Boruta Algorithm as Feature Selection Method, Lee Kuok Leong and Azian Azamimi Abdullah.

[4] Machine learning for modeling the progression of Alzheimer disease dementia using clinical data: a systematic literature review, Sayantan Kumar, Inez Oh, 2021.

[5] Machine learning prediction of incidence of Alzheimer’s disease using large-scale administrative health data, Ji Hwan Park, Han Eol Cho 2, 2020.

[6] Alzheimer's Disease Early Detection Using Machine Learning Techniques, Roobaea Alroobaea1, Seifeddine Mechti2, Mariem Haoues2, Saeed Rubaiee3, Anas Ahmed3, Murad Andejany3, Nicola Luigi Bragazzi4, Dilip Kumar, Sharma5, Bhanu Prakash Kolla6, Sudhakar Sengan7, 2021.

[7] A Predictive and Preventive Model for Onset of Alzheimer’s Disease
Udit Singhania1, Balakrushna Tripathy 2, Mohammad Kamrul Hasan3*, Noble C. Anumbe4, Dabiah Alboaneen5, Fatima Rayan Awad Ahmed6, Thowiba E. Ahmed5 and Manasik M. Mohamed Nour 7, 2021.

[8] A multilayer multimodal detection and prediction model based on explainable artificial intelligence for Alzheimer’s disease, Shaker El-Sappagh1,2*, Jose M. Alonso3, S. M. Riazul Islam4, Ahmad M. Sultan5 & Kyung Sup Kwak6*, 2021.

[9] Diagnosis of Alzheimer 's Disease using Combined Feature Selection Method, Fazal Ur Rehman Faisal, Uttam Khatri, Goo-Rak Kwon, 2021.

[10] Deep Structural and Clinical Feature Learning for Semi-Supervised Multiclass Prediction of Alzheimer’s Disease, Emimal Jabason, Student Member, IEEE, M. Omair Ahmad, Fellow, IEEE, and M. N. S Swamy, Fellow, IEEE, 2018.