**LIBRARY FILES**

hugchat==0.3.1

joblib==1.3.2

pandas==2.0.3

Requests==2.31.0

streamlit==1.27.2

scikit-learn==1.2.2

**ALZHEIMER DETECTION**

# %%

# import opencv

import cv2

# store list

images = []

lables = []

## AD Patient Brain Scan with PreProcessing of the Image ##

for i in range(1, 172):

    img = cv2.imread(f"C:\\Alzheimer-Disease-Prediction-master\\Alzheimers-ADNI\\train\\Final AD JPEG\\AD ({i}).jpg")

    gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

    gray = gray/255

    gray = cv2.resize(gray,(200, 200))

    images.append(gray)

    lables.append(0)  # AD

# %%

# Sample Images

import matplotlib.pyplot as plt

plt.figure(figsize = (15,15))

for i in range(20):

    plt.subplot(4, 5, i + 1)

    plt.imshow(images[10 + i\*3])

plt.show()

# %%

## CN Patient Brain Scan with PreProcessing of the Image ##

for i in range(1, 581):

    img = cv2.imread(f"C:\\Alzheimer-Disease-Prediction-master\\Alzheimers-ADNI\\train\\Final CN JPEG\\CN ({i}).jpg")

    gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

    gray = gray/255

    gray = cv2.resize(gray,(200, 200))

    images.append(gray)

    lables.append(1)  # CN

# %%

# Sample Images

plt.figure(figsize = (15,15))

for i in range(20):

    plt.subplot(4, 5, i + 1)

    plt.imshow(images[200 + i\*3])

plt.show()

# %%

## EMCI Patient Brain Scan with PreProcessing of the Image ##

for i in range(1, 241):

    img = cv2.imread(f"C:\\Alzheimer-Disease-Prediction-master\\Alzheimers-ADNI\\train\\Final EMCI JPEG\\EMCI ({i}).jpg")

    gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

    gray = gray/255

    gray = cv2.resize(gray,(200, 200))

    images.append(gray)

    lables.append(2)  # EMCI

# %%

# Sample Images

plt.figure(figsize = (15,15))

for i in range(20):

    plt.subplot(4, 5, i + 1)

    plt.imshow(images[780 + i\*3])

plt.show()

# %%

## LMCI Patient Brain Scan with PreProcessing of the Image ##

for i in range(1, 73):

    img = cv2.imread(f"C:\\Alzheimer-Disease-Prediction-master\\Alzheimers-ADNI\\train\\Final LMCI JPEG\\LMCI ({i}).jpg")

    gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

    gray = gray/255

    gray = cv2.resize(gray,(200, 200))

    images.append(gray)

    lables.append(3)  # LMCI

# %%

# Sample Images

plt.figure(figsize = (15,15))

for i in range(20):

    plt.subplot(4, 5, i + 1)

    plt.imshow(images[995 + i\*3])

plt.show()

# %%

## LMCI Patient Brain Scan with PreProcessing of the Image ##

for i in range(1, 234):

    img = cv2.imread(f"C:\\Alzheimer-Disease-Prediction-master\\Alzheimers-ADNI\\train\\Final MCI JPEG\\MCI ({i}).jpg")

    gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

    gray = gray/255

    gray = cv2.resize(gray,(200, 200))

    images.append(gray)

    lables.append(4)  # MCI

# %%

# Sample Images

plt.figure(figsize = (15,15))

for i in range(20):

    plt.subplot(4, 5, i + 1)

    plt.imshow(images[1080 + i\*3])

plt.show()

# %%

# Shape of the Images

print(f"Shape of each image is = {images[1000].shape}")

# %%

# Convert The List into Array Format  #

import numpy as np

train\_feature = np.array(images)

lables = np.array(lables)

# %%

## Display Array Shape ##

print(f"image dataset shape = {train\_feature.shape}")

print(f"lable dataset shape = {lables.shape}")

# %% [markdown]

# # Split The Dataset into Test and Train

# %%

from sklearn.model\_selection import train\_test\_split

train\_features, test\_features, train\_target, test\_target = train\_test\_split(train\_feature,lables,test\_size=0.12)

print(f"train\_features shape = {train\_features.shape}")

print(f"test\_features shape = {test\_features.shape}")

print(f"train\_target shape = {train\_target.shape}")

print(f"test\_target shape = {test\_target.shape}")

# %% [markdown]

# # Deep Learning CNN Model Architecture

# %%

# MODEL ARCHITECTURE

import keras

import tensorflow

from tensorflow.keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout

# ONE NOT ENCODING

train\_target = to\_categorical(lables)

# MODEL LAYERS

model=Sequential()

# Convolutional Layers

model.add(Conv2D(25, kernel\_size = (3,3), strides = (1,1), padding = 'same', activation = 'relu', input\_shape = (200, 200, 1)))

model.add(Conv2D(75, kernel\_size = (3,3), strides = (1,1), padding = 'same', activation = 'relu'))

model.add(MaxPooling2D(pool\_size = (2,2)))

model.add(Flatten())

model.add(Dense(500,activation='relu'))

model.add(Dropout(0.25))

model.add(Dense(250,activation='relu'))

model.add(Dropout(0.25))

model.add(Dense(100,activation='relu'))

model.add(Dense(5,activation='softmax'))

model.summary()

# %%

# MODEL COPILE

from tensorflow.keras.optimizers import Adam

model.compile(Adam(learning\_rate = 0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

# %%

from tensorflow.keras.models import load\_model

# Load the saved model

model = load\_model(r"C:\Users\AMREEN\Alzheimers\_Prediction\_System-main\your\_model\_name.h5")

# %%

# Test our Model

predictions = model.predict(test\_features)

# %%

# Accuracy of Our Model #

wrong = 0

for i in range(156) :

    test = np.argmax(predictions[i])

    train = test\_target[i]

    if (test != train):

        wrong += 1

print(f"Model Accuracy : {((156 - wrong)/156)\*100} %")

# %%

import cv2

import numpy as np

# Preprocess input image(s)

def preprocess\_image(image\_path):

    img = cv2.imread(image\_path)

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    gray = gray / 255

    gray = cv2.resize(gray, (200, 200))

    # Reshape to match model input shape

    processed\_img = gray.reshape(1, 200, 200, 1)

    return processed\_img

# Function to predict Alzheimer's

def predict\_alzheimer(image\_path):

    processed\_img = preprocess\_image(image\_path)

    prediction = model.predict(processed\_img)

    class\_label = np.argmax(prediction)

    if class\_label == 0:

        return "AD (Alzheimer's Disease)"

    elif class\_label == 1:

        return "CN (Cognitively Normal)"

    elif class\_label == 2:

        return "EMCI (Early Mild Cognitive Impairment)"

    elif class\_label == 3:

        return "LMCI (Late Mild Cognitive Impairment)"

    elif class\_label == 4:

        return "MCI (Mild Cognitive Impairment)"

# Example usage:

image\_path = r'C:\Users\AMREENAS\OneDrive\Desktop\helo.jpeg'

result = predict\_alzheimer(image\_path)

print("Predicted class:", result)

# %%

import cv2

import numpy as np

import streamlit as st

# Load your model

# Replace this with your actual model loading code

model = None

# Preprocess input image(s)

def preprocess\_image(image):

    img = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    gray = gray / 255

    gray = cv2.resize(gray, (200, 200))

    # Reshape to match model input shape

    processed\_img = gray.reshape(1, 200, 200, 1)

    return processed\_img

# Function to predict Alzheimer's

def predict\_alzheimer(image):

    processed\_img = preprocess\_image(image)

    prediction = model.predict(processed\_img)

    class\_label = np.argmax(prediction)

    if class\_label == 0:

        return "AD (Alzheimer's Disease)"

    elif class\_label == 1:

        return "CN (Cognitively Normal)"

    elif class\_label == 2:

        return "EMCI (Early Mild Cognitive Impairment)"

    elif class\_label == 3:

        return "LMCI (Late Mild Cognitive Impairment)"

    elif class\_label == 4:

        return "MCI (Mild Cognitive Impairment)"

# Main function for Streamlit app

def main():

    st.title("Alzheimer's Disease Prediction")

    st.write("Upload an image to predict the result")

    # File uploader

    uploaded\_file = st.file\_uploader("Choose an image...", type=["jpg", "jpeg"])

    if uploaded\_file is not None:

        # Display uploaded image

        image = np.array(bytearray(uploaded\_file.read()), dtype=np.uint8)

        st.image(image, caption='Uploaded Image', use\_column\_width=True)

        # Predict result

        result = predict\_alzheimer(image)

        st.write("Predicted class:", result)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# %%

**ALZHEIMER PREDICTION**

# %% [markdown]

# <img src="assets/images/banner.jpg" alt="banner" style="background-color: #ecf0f1; border-radius: 5px; text-align:center; color:#080205; font-weight:800; text-shadow: 2px 2px 2px #888888;">

# %% [markdown]

# <div style="padding: 10px; border-radius: 5px; color:#080205; font-family: 'Times New Roman', Times, serif;">

#     <h1 style="font-weight:800; font-size:30px;">Introduction</h1>

#     Alzheimer's disease is a complex neurodegenerative disorder that affects millions of people worldwide. Early detection and prediction of Alzheimer's can lead to better management and treatment outcomes. This prediction system utilizes a machine learning model trained on a dataset of relevant features to provide predictions about the likelihood of Alzheimer's disease.

#     <h1 style="font-weight:800; font-size:30px;">About Alzheimer's Disease</h1>

#     Alzheimer's disease (AD) is a progressive neurodegenerative disease. Though best known for its role in declining memory function, symptoms also include: difficulty thinking and reasoning, making judgements and decisions, and planning and performing familiar tasks. It may also cause alterations in personality and behavior. The cause of AD is not well understood. There is thought to be a significant hereditary component. For example, a variation of the APOE gene, APOE e4, increases risk of Alzheimer's disease.

#     <h1 style="font-weight:800; font-size:30px;">Purpose of the project</h1>

#     The purpose of this project proposal is to develop a machine learning model for the early prediction of Alzheimer's disease. Alzheimer's disease is a devastating neurodegenerative disorder that affects millions of individuals worldwide. Early detection is crucial for better patient care and the development of potential interventions. This project aims to leverage machine learning techniques to create a predictive model that can identify individuals at risk of Alzheimer's disease based on relevant data.

#     <h1 style="font-weight:800; font-size:30px;">Potential Impact</h1>

#     The potential impact of this project on the issue of Alzheimer's disease is significant:

#     - Early prediction of Alzheimer's disease can lead to timely interventions, potentially slowing down the progression of the disease.

#     - Accurate prediction models can aid in identifying suitable candidates for clinical trials and research studies.

#     - Providing a tool for early prediction can raise awareness about Alzheimer's disease and encourage individuals to seek early medical evaluation.

#     <br><br>

#     The model will be trained on a dataset collected from <a href="https://adni.loni.usc.edu/">Alzheimer’s Disease Neuroimaging Initiative (ADNI)</a>. This dataset is a comprehensive collection of clinical, imaging, and genetic data from individuals with Alzheimer's disease.

# </div>

#

# <h2 style="color: #000; font-family: 'Times New Roman', Times, serif;"><b>Variable Descriptions</b></h2>

# <table style="color: #000; width: 100%; border-collapse: collapse; margin-bottom: 20px;">

#     <tr>

#         <th style="color: #fff; background-color: #080205; border: 1px solid #fff; padding: 8px; text-align: left;">Variable</th>

#         <th style="color: #fff; background-color: #080205; border: 1px solid #fff; padding: 8px; text-align: left;">Description</th>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">index</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">A numerical index for each row in the dataframe.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">directory.id</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">A unique identifier for each row in the dataframe.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">Subject</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">A unique identifier for each subject in the dataframe.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">RID</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">A numerical identifier for each subject in the dataframe.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">image.data.id</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">A numerical identifier for each image in the dataframe.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">Modality</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The type of imaging used to collect the data (MRI).</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">Visit</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The visit number associated with the data.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">Acq.Date</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The date on which the data was acquired.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">DX.bl</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The diagnosis at baseline (AD, LMCI, or CN).</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">EXAMDATE</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The date of the exam associated with the data.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">AGE</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The age of the subject at the time of the exam.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">PTGENDER</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The gender of the subject.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">PTEDUCAT</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The educational level of the subject.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">PTETHCAT</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The ethnicity of the subject (Hisp/Latino or Not Hisp/Latino).</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">PTRACCAT</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The race of the subject (White).</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">APOE4</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The APOE4 genotype of the subject (0, 1, or 2).</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">MMSE</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The Mini-Mental State Examination score of the subject.</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">imputed\_genotype</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">Whether or not the genotype was imputed (True or False).</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">APOE Genotype</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The APOE genotype of the subject (3,3; 3,4; 4,3; 4,4).</td>

#     </tr>

#     <tr>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">Dx Codes for Submission</td>

#         <td style="border: 1px solid #fff; padding: 8px; text-align: left;">The diagnosis code for submission (AD, MCI, or CN).</td>

#     </tr>

# </table>

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Importing the libraries

# </h1>

# %%

import joblib

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear\_model import RidgeClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

# %%

import pandas as pd

data = pd.read\_csv('C:\Alzheimers\_Prediction\_System-main-20240313T053759Z-002\Alzheimers\_Prediction\_System-main\data\ADNI\_Training\_Q3\_APOE\_CollectionADNI1Complete 1Yr 1.5T\_July22.2014.csv')

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Preprocessing

# </h1>

# %% [markdown]

# <div style="text-align:center; color:black; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Checking for NULL values

# </div>

# %%

data.isnull().sum()

# %% [markdown]

# <div style="text-align:center; color:black; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Deleting NULL values

# </div>

# %%

data = data.dropna()

data.isnull().sum().sum()

# %%

data.head(3)

# %% [markdown]

# <div style="text-align:center; color:black; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Declaring Features (X) and Target (y) columns

# </div>

# %%

X = data

Y = data['DX.bl']

# del data

remove\_columns = list(X.columns)[0:9]

remove\_columns.append('Dx Codes for Submission')

print('Removing columns:', remove\_columns)

X = X.drop(remove\_columns, axis=1)

features = list(X.columns)

X.head(5)

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Exploratory Data Analysis (EDA)

# </h1>

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.5em;">

#     Relationship between age and diagnosis (DX.bl) by plotting a scatterplot of age against diagnosis.

# </div>

# %%

plt.scatter(data['AGE'], data['DX.bl'])

plt.xlabel('Age')

plt.ylabel('Diagnosis')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Counts of gender and ethnicity across different diagnoses

# </div>

# %%

gender\_ethnicity\_counts = data.groupby(['DX.bl', 'PTGENDER', 'PTETHCAT']).size().reset\_index(name='count')

print(gender\_ethnicity\_counts)

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Relationship between APOE4 genotype and diagnosis (DX.bl)

# </div>

# %%

sns.countplot(x='APOE4', hue='DX.bl', data=data)

plt.title('Relationship between APOE4 genotype and diagnosis')

plt.xlabel('APOE4 Genotype')

plt.ylabel('Count')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Scatterplot of MMSE score against diagnosis

# </div>

# %%

plt.scatter(data['MMSE'], data['DX.bl'])

plt.xlabel('MMSE Score')

plt.ylabel('Diagnosis')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Plotting the counts of imputed genotype (imputed\_genotype) across different diagnoses (DX.bl)

# </div>

# %%

counts = data.groupby(['DX.bl', 'imputed\_genotype']).size().reset\_index(name='counts')

plt.figure(figsize=(10,6))

ax = sns.barplot(x="DX.bl", y="counts", hue="imputed\_genotype", data=counts)

ax.set\_title('Distribution of Imputed Genotype Across Different Diagnoses')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.5em;">

#     Relationship between APOE Genotype and diagnosis (DX.bl) by plotting a bar chart of APOE Genotype against diagnosis.

# </div>

# %%

ax = data.groupby(['APOE Genotype', 'DX.bl']).size().unstack().plot(kind='bar', figsize=(10,6))

ax.set\_title('APOE Genotype vs Diagnosis')

ax.set\_xlabel('APOE Genotype')

ax.set\_ylabel('Diagnosis')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Categorical Variable Distributions

# </div>

# %%

numerical\_vars = ['AGE', 'MMSE', 'PTEDUCAT']

cat\_vars = list(set(features) - set(numerical\_vars))

print('Categorical variable distributions:\n')

for var in cat\_vars:

    print('\nDistribution of', var)

    print(X[var].value\_counts())

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Numerical Variable Distributions

# </div>

# %%

print('Numerical Var Distributions:\n')

for var in numerical\_vars:

    plt.hist(X[var], bins=10)

    plt.title(var + ' Distribution')

    plt.show()

    print(X[var].describe())

# %% [markdown]

# <!-- <p style="color: #3498db; font-size: 18px; font-weight: bold;">Stylish Text 1</p>

# <p style="font-family: 'Courier New', monospace; font-size: 20px; color: #2ecc71;">Stylish Text 2</p>

# <p style="text-transform: uppercase; letter-spacing: 2px; font-size: 16px; color: #e74c3c;">Stylish Text 3</p>

# <p style="font-style: italic; font-size: 22px; color: #f39c12;">Stylish Text 4</p>

# <p style="text-decoration: underline; font-size: 18px; color: #9b59b6;">Stylish Text 5</p>

# <p style="font-family: 'Times New Roman', serif; font-size: 24px; color: #34495e;">Stylish Text 6</p>

# <p style="font-weight: 700; font-size: 20px; color: #1abc9c;">Stylish Text 8</p>

# <p style="text-shadow: 2px 2px 4px #888888; font-size: 18px; color: #c0392b;">Stylish Text 9</p>

# <p style="font-size: 22px; color: #3498db; transform: rotate(15deg);">Stylish Text 10</p> -->

# %% [markdown]

# <div style="text-align:center; color:orange; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 1.7em;">

#     Analyzing target (y) distribution

# </div>

# %%

plt.bar(Y.value\_counts().index, Y.value\_counts())

plt.show()

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Preprocessing

# </h1>

# %% [markdown]

# <div style="text-align:center; color:yellow; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2em;">

#     Converting the Categorical Variables to 1-Hot

# </div>

# %%

for var in cat\_vars:

    print(f'Converting {var} to 1-hot encoding')

    one\_hot\_df = pd.get\_dummies(X[var], prefix=var)

    X = pd.concat([X, one\_hot\_df], axis=1)

    X.drop(var, axis=1, inplace=True)

X.head(4)

# %% [markdown]

# <div style="text-align:center; color:yellow; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2em;">

#     Normalization function

# </div>

# %%

def normalize(X):

    X = np.array(X)

    means = np.mean(X, axis=0)

    stds = np.std(X, axis=0)

    normalized\_X = (X - means) / stds

    print('Normalizing:')

    for i, (mean, std) in enumerate(zip(means, stds)):

        print(f'  Variable {i+1}: Mean = {mean:.2f}, Standard Deviation = {std:.2f}')

    return normalized\_X

# %% [markdown]

# <div style="text-align:center; color:yellow; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2em;">

#     Splitting the dataset

# </div>

# %%

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=.2)

num\_test = X\_test.shape[0]

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Comparing between models

# </h1>

# %%

X\_train.columns = X\_train.columns.astype(str)

X\_test.columns = X\_test.columns.astype(str)

# %% [markdown]

# ### Checking each model's accuracy

# %% [markdown]

# <div style="text-align:center; color:lightgreen; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2.0em;">

#     Linear Discriminant Analysis

# </div>

# %%

lda\_clf = LinearDiscriminantAnalysis(solver='svd', store\_covariance=False, tol=0.0001)

lda\_clf.fit(X\_train, y\_train)

y\_pred = lda\_clf.predict(X\_test)

print('Cross Validation:', format(cross\_val\_score(lda\_clf, X\_train, y\_train, cv=5).mean(), '.2%'))

print('Accuracy:', format(accuracy\_score(y\_test, y\_pred), '.2%'))

print('Precision:', format(precision\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Recall:', format(recall\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('F1 Score:', format(f1\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Confusion Matrix:')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, cmap='Blues')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:lightgreen; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2.0em;">

#     Logistic Regression

# </div>

# %%

log\_clf = LogisticRegression(solver='lbfgs', penalty='l2', max\_iter=1000000, multi\_class='multinomial')

log\_clf.fit(X\_train, y\_train)

y\_pred = log\_clf.predict(X\_test)

print('Cross Validation:', format(cross\_val\_score(log\_clf, X\_train, y\_train, cv=5).mean(), '.2%'))

print('Accuracy:', format(accuracy\_score(y\_test, y\_pred), '.2%'))

print('Precision:', format(precision\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Recall:', format(recall\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('F1 Score:', format(f1\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Confusion Matrix:')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, cmap='Blues')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:lightgreen; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2.0em;">

#     Random Forest Classifier

# </div>

# %%

rf\_clf = RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,

                                criterion='gini', max\_depth=None, max\_features='sqrt',

                                max\_leaf\_nodes=None, max\_samples=None,

                                min\_impurity\_decrease=0.0, min\_samples\_leaf=1,

                                min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

                                n\_estimators=100, n\_jobs=-1, oob\_score=False,

                                random\_state=123, verbose=0, warm\_start=False)

rf\_clf.fit(X\_train, y\_train)

y\_pred = rf\_clf.predict(X\_test)

print('Cross Validation:', format(cross\_val\_score(rf\_clf, X\_train, y\_train, cv=5).mean(), '.2%'))

print('Accuracy:', format(accuracy\_score(y\_test, y\_pred), '.2%'))

print('Precision:', format(precision\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Recall:', format(recall\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('F1 Score:', format(f1\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Confusion Matrix:')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, cmap='Blues')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:lightgreen; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2.0em;">

#     Ridge Classifier

# </div>

# %%

rg\_clf = RidgeClassifier(alpha=1.0, copy\_X=True, fit\_intercept=True, random\_state=123, solver='auto', tol=0.0001)

rg\_clf.fit(X\_train, y\_train)

y\_pred = rg\_clf.predict(X\_test)

print('Cross Validation:', format(cross\_val\_score(rg\_clf, X\_train, y\_train, cv=5).mean(), '.2%'))

print('Accuracy:', format(accuracy\_score(y\_test, y\_pred), '.2%'))

print('Precision:', format(precision\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Recall:', format(recall\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('F1 Score:', format(f1\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Confusion Matrix:')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, cmap='Blues')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:lightgreen; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2.0em;">

#     Gradient Boosting Classifier

# </div>

# %%

gb\_clf = GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', init=None,

                            learning\_rate=0.1, loss='log\_loss', max\_depth=3,

                            max\_features=None, max\_leaf\_nodes=None,

                            min\_impurity\_decrease=0.0, min\_samples\_leaf=1,

                            min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

                            n\_estimators=100, n\_iter\_no\_change=None,

                            random\_state=123, subsample=1.0, tol=0.0001,

                            validation\_fraction=0.1, verbose=0,

                            warm\_start=False)

gb\_clf.fit(X\_train, y\_train)

y\_pred = gb\_clf.predict(X\_test)

print('Cross Validation:', format(cross\_val\_score(gb\_clf, X\_train, y\_train, cv=5).mean(), '.2%'))

print('Accuracy:', format(accuracy\_score(y\_test, y\_pred), '.2%'))

print('Precision:', format(precision\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Recall:', format(recall\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('F1 Score:', format(f1\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Confusion Matrix:')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, cmap='Blues')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:lightgreen; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2.0em;">

#     Multi-layer Perceptron (MLP) Classifier

# </div>

# %%

mlp\_clf = MLPClassifier(hidden\_layer\_sizes=(15, 10), alpha=3, learning\_rate='adaptive', max\_iter=100000)

mlp\_clf.fit(X\_train, y\_train)

y\_pred = mlp\_clf.predict(X\_test)

print('Cross Validation:', format(cross\_val\_score(mlp\_clf, X\_train, y\_train, cv=5).mean(), '.2%'))

print('Accuracy:', format(accuracy\_score(y\_test, y\_pred), '.2%'))

print('Precision:', format(precision\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Recall:', format(recall\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('F1 Score:', format(f1\_score(y\_test, y\_pred, average='weighted'), '.2%'))

print('Confusion Matrix:')

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, cmap='Blues')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# %% [markdown]

# <div style="text-align:center; color:lightgreen; font-weight:600; font-family: 'Courier New', Times, serif; font-size: 2.0em;">

#     Conclusion

# </div>

# %% [markdown]

# <!-- <center>

#   <table>

#     <thead>

#       <tr>

#         <th>Model</th>

#         <th>Cross Val</th>

#         <th>Accuracy</th>

#         <th>Precision</th>

#         <th>Recall</th>

#         <th>F1 Score</th>

#       </tr>

#     </thead>

#     <tbody>

#       <tr>

#         <td>Linear Discriminant Analysis</td>

#         <td>69.86</td>

#         <td>66.67</td>

#         <td>67.52</td>

#         <td>66.67</td>

#         <td>66.62</td>

#       </tr>

#       <tr style="background-color: green;">

#         <td>Logistic Regression</td>

#         <td>70.66</td>

#         <td>68.25</td>

#         <td>68.63</td>

#         <td>68.25</td>

#         <td>68.12</td>

#       </tr>

#       <tr>

#         <td>Random Forest Classifier</td>

#         <td>65.66</td>

#         <td>64.29</td>

#         <td>63.94</td>

#         <td>64.29</td>

#         <td>63.87</td>

#       </tr>

#       <tr>

#         <td>Ridge Classifier</td>

#         <td>64.87</td>

#         <td>63.49</td>

#         <td>65.01</td>

#         <td>63.49</td>

#         <td>63.51</td>

#       </tr>

#       <tr>

#         <td>Gradient Boosting Classifier</td>

#         <td>66.27</td>

#         <td>64.29</td>

#         <td>65.17</td>

#         <td>64.29</td>

#         <td>64.07</td>

#       </tr>

#       <tr>

#         <td>MLP Classifier</td>

#         <td>51.28</td>

#         <td>63.29</td>

#         <td>63.61</td>

#         <td>63.49</td>

#         <td>62.97</td>

#       </tr>

#     </tbody>

#   </table>

# </center> -->

#

# Choosing Logistic Regression with validation Score .67

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Model Selection

# </h1>

# %% [markdown]

# ### Testing Logistic Regression Model

# %%

log\_clf\_preds = log\_clf.predict(X\_test)

log\_clf\_accuracy = (log\_clf\_preds == y\_test)

print('Test Accuracy =', format( np.sum(log\_clf\_accuracy)/num\_test, '.2%'))

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Performance Analysis

# </h1>

# %%

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

log\_clf = LogisticRegression(solver='lbfgs', penalty='l2', max\_iter=1000000, multi\_class='multinomial')

log\_clf.fit(X\_train, y\_train)

log\_clf\_preds = log\_clf.predict(X\_test)

print('Logistic Regression:')

print('Accuracy:', format(accuracy\_score(y\_test, log\_clf\_preds), '.2%'))

print('Precision:', format(precision\_score(y\_test, log\_clf\_preds, average='weighted'), '.2%'))

print('Recall:', format(recall\_score(y\_test, log\_clf\_preds, average='weighted'), '.2%'))

print('F1 Score:', format(f1\_score(y\_test, log\_clf\_preds, average='weighted'), '.2%'))

# %% [markdown]

# <h1 style="background-color: #080205; padding: 10px; border-radius: 5px; text-align:center; color:#ecf0f1; font-weight:600; text-shadow: 2px 2px 2px #888888;">

#     Exporting the final model

# </h1>

# %%

# joblib.dump(log\_clf, 'model/alzheimer\_model.pkl')

**GENERATIVE CHATBOT**

import streamlit as st

from hugchat import hugchat

from hugchat.login import Login

NEWS\_API = "y886b1766bdc4af69e19811aef4dc9e8"

NEWS\_API\_KEY = ["NEWS\_API"]

KEYWORD = "alzheimer"

HF\_EMAIL = "amreenrafiq10@gmail.com"

HF\_PASS = "v/PkBzwiJdbW@4!"

BASE\_PROMPT = "Hello"

# Login Credentials

hf\_email = HF\_EMAIL

hf\_pass = HF\_PASS

flag = 0

def chat\_bot():

    # Store LLM generated responses

    if "messages" not in st.session\_state.keys():

        st.session\_state.messages = [{"role": "assistant", "content": "Hi! How may I help you?"}]

    # Display chat messages

    for message in st.session\_state.messages:

        with st.chat\_message(message["role"]):

            st.write(message["content"])

    # Function for generating LLM response

    def generate\_response(prompt\_input, email, passwd):

        # Hugging Face Login

        sign = Login(email, passwd)

        cookies = sign.login()

        # Create ChatBot

        chatbot = hugchat.ChatBot(cookies=cookies.get\_dict())

        global flag

        if flag<1:

            flag+=1

            prompt\_input=BASE\_PROMPT+prompt\_input

        return str(chatbot.chat(prompt\_input)).strip('`')

    # User-provided prompt

    if prompt := st.chat\_input(disabled=not (hf\_email and hf\_pass)):

        st.session\_state.messages.append({"role": "user", "content": prompt})

        with st.chat\_message("user"):

            st.write(prompt)

    # Generate a new response if last message is not from assistant

    if st.session\_state.messages[-1]["role"] != "assistant":

        with st.chat\_message("assistant"):

            with st.spinner("Thinking..."):

                response = generate\_response(prompt, hf\_email, hf\_pass)

                st.write(response)

        message = {"role": "assistant", "content": response}

        st.session\_state.messages.append(message)

**ALZNEWSFEED**

import html

import requests

import streamlit as st

KEYWORD = "alzheimer"

DEFAULT\_IMAGE = "C:/Users/AMREEN/Alzheimers\_Prediction\_System-main/assets/images/default.webp"

NEWS\_API\_KEY = "c886b1766bdc4af69e19811aef4dc9e8"

def \_get\_news():

    response = requests.get(f'https://newsapi.org/v2/everything?q={KEYWORD}&apiKey={NEWS\_API\_KEY}&language=en&searchIn=title').json()

    return response.get('articles', [])

def news\_page():

    news\_articles = \_get\_news()

    if not news\_articles:

        st.write("No news articles found.")

        return

    for random\_news in news\_articles:

        st.write(f"""<h2>{html.unescape(random\_news['title'])}</h2><br>""", unsafe\_allow\_html=True)

        if random\_news.get("urlToImage"):

            st.image(random\_news["urlToImage"])

        else:

            st.image(DEFAULT\_IMAGE)

        st.write(f"""

            <h5>{random\_news['description']}</h5>

            Link : <a href="{random\_news['url']}">{random\_news['url'][:80]}...</a><br>

            Author : {random\_news['author']}, &nbsp; <i>{random\_news['publishedAt'][:10]}</i>

            <hr>

            """, unsafe\_allow\_html=True)

**STREAMLIT APP**

import base64

import streamlit as st

from config import \*

from streamlit\_pages.\_home\_page import home\_page

from streamlit\_pages.\_predict\_alzheimer import prediction\_page

from streamlit\_pages.\_latest\_news import news\_page

from streamlit\_pages.\_team\_members import team\_members

from streamlit\_pages.\_chat\_page import chat\_bot

from streamlit\_pages.\_detect\_alzheimer import alzheimer\_detection

# SETTING PAGE CONFIG

st.set\_page\_config(

    page\_title="Alzheimer's Prediction Systems",

    page\_icon=":brain:",

)

st.markdown(f"<style>{CSS}</style>", unsafe\_allow\_html=True)

def set\_page\_background(png\_file):

    @st.cache\_data()

    def get\_base64\_of\_bin\_file(bin\_file):

        with open(bin\_file, 'rb') as f:

            data = f.read()

        return base64.b64encode(data).decode()

    bin\_str = get\_base64\_of\_bin\_file(png\_file)

    page\_bg\_img = f'''

        <style>

        .stApp {{

            background-image: url("data:image/png;base64,{bin\_str}");

            }}

        </style>

    '''

    st.markdown(page\_bg\_img, unsafe\_allow\_html=True)

set\_page\_background(BACKGROUND)

# STREAMLIT APP

st.sidebar.image(SIDE\_BANNER)

st.sidebar.title("Alzheimer's Prediction System")

app\_mode = st.sidebar.selectbox(

    "Please navigate through the different sections of our website from here",

    ["Home", "Predict Alzheimer's", "ChatBot", "Latest News", "Team Members", "Detect Alzheimer"]

)

st.sidebar.write("""

# Disclaimer

The predictions provided by this system are for informational purposes only. Consult a healthcare professional for accurate diagnosis and advice.

# Contact

For inquiries, you can mail us [here](amreenrafiq10@gmail.com).

""")

def main():

    if app\_mode == "Home":

        home\_page()

    if app\_mode == "Predict Alzheimer's":

        prediction\_page()

    if app\_mode == "ChatBot":

        chat\_bot()

    if app\_mode == "Latest News":

        news\_page()

    if app\_mode == "Team Members":

        team\_members()

    if app\_mode == "Detect Alzheimer":

        alzheimer\_detection()

if \_\_name\_\_ == "\_\_main\_\_":

    main()