**Lab-9**

Chandra kiran kopparapu

BL.EN. U4AIE23141

Code:

import numpy as np

import os

import cv2

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

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, StackingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neural\_network import MLPClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import classification\_report, accuracy\_score

import lime

import lime.lime\_tabular

# === Path to dataset ===

DATASET\_PATH = "C:\\AIO\\Semster Files\\SEMSTER - 4\\ML\\Lab Work\\ML\_Assignment\_09\_BL.EN.U4AIE23138\\Dataset"

# === Function to load and split into 2 classes (artificial for small dataset) ===

def load\_dataset(path: str, size: int = 64):

    images = []

    for fname in os.listdir(path):

        img\_path = os.path.join(path, fname)

        img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

        if img is not None:

            img = cv2.resize(img, (size, size)).flatten()

            images.append(img)

    features = np.array(images)

    count = len(features)

    # Artificially assign half as class 0 and half as class 1

    labels = np.zeros(count, dtype=int)

    labels[count // 2:] = 1

    return features, labels

# === A1: Stacking Classifier ===

def create\_stacking\_classifier(X\_train, y\_train):

    base\_classifiers = [

        ('rf', RandomForestClassifier(n\_estimators=50, random\_state=42)),

        ('svm', SVC(probability=True, random\_state=42)),

        ('mlp', MLPClassifier(max\_iter=300, random\_state=42)),

        ('gnb', GaussianNB())

    ]

    final\_estimator = LogisticRegression()

    model = StackingClassifier(

        estimators=base\_classifiers,

        final\_estimator=final\_estimator,

        cv=2  # Fix: Use 2-fold CV due to small dataset

    )

    model.fit(X\_train, y\_train)

    return model

# === A2: Pipeline ===

def create\_pipeline(X\_train, y\_train):

    pipe = Pipeline(steps=[

        ('scaler', StandardScaler()),

        ('clf', MLPClassifier(hidden\_layer\_sizes=(64,), max\_iter=300, random\_state=42))

    ])

    pipe.fit(X\_train, y\_train)

    return pipe

# === A3: LIME ===

def explain\_with\_lime(pipeline, X\_train, X\_test):

    explainer = lime.lime\_tabular.LimeTabularExplainer(

        training\_data=X\_train,

        feature\_names=[f'pixel\_{i}' for i in range(X\_train.shape[1])],

        class\_names=['Glioma\_0', 'Glioma\_1'],

        mode='classification'

    )

    explanation = explainer.explain\_instance(X\_test[0], pipeline.predict\_proba)

    # Save explanation as HTML with proper encoding

    html\_path = "lime\_explanation.html"

    with open(html\_path, "w", encoding="utf-8") as f:

        f.write(explanation.as\_html())

    print(f"LIME explanation saved to: {html\_path}")

    print("Open this file in your browser to view the explanation.")

    return html\_path

# === MAIN ===

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

    X, y = load\_dataset(DATASET\_PATH)

    if len(np.unique(y)) < 2:

        raise ValueError("Dataset must have at least two classes to perform classification.")

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(

        X, y, test\_size=0.3, random\_state=42, stratify=y

    )

    # A1

    print("\n--- A1: Stacking Classifier ---")

    stacking\_model = create\_stacking\_classifier(X\_train, y\_train)

    y\_pred\_stack = stacking\_model.predict(X\_test)

    print("Accuracy:", accuracy\_score(y\_test, y\_pred\_stack))

    print(classification\_report(y\_test, y\_pred\_stack))

    # A2

    print("\n--- A2: Pipeline with MLP ---")

    pipeline\_model = create\_pipeline(X\_train, y\_train)

    y\_pred\_pipe = pipeline\_model.predict(X\_test)

    print("Accuracy:", accuracy\_score(y\_test, y\_pred\_pipe))

    print(classification\_report(y\_test, y\_pred\_pipe))

    # A3

    print("\n--- A3: LIME Explanation ---")

    explain\_with\_lime(pipeline\_model, X\_train, X\_test)