**Lab-6**

**Chandra kiran kopparapu**

BL.EN. U4AIE23141

Code:

import os

import numpy as np

import cv2

import zipfile

import math

import matplotlib.pyplot as plt

from skimage.feature import hog

from collections import Counter

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

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

from sklearn.inspection import DecisionBoundaryDisplay

# Debugging function

def debug\_print(msg, var=None):

    print(f"{msg}")

    if var is not None:

        print(var)

# Step 1: Unzip & Load Dataset

dataset\_path = "C:\\AIO\\Semster Files\\SEMSTER - 4\\ML\\Lab Work\\ML\_Assignment\_06\_BL.EN.U4AIE23138\\Dataset.zip"

extract\_path = "C:\\AIO\\Semster Files\\SEMSTER - 4\\ML\\Lab Work\\ML\_Assignment\_06\_BL.EN.U4AIE23138\\Dataset"

with zipfile.ZipFile(dataset\_path, 'r') as zip\_ref:

    zip\_ref.extractall(extract\_path)

debug\_print("Dataset extracted successfully.")

# Step 2: Feature Extraction using HOG

def extract\_hog\_features(image\_path):

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

    if image is None:

        debug\_print(f"Error: Could not read image {image\_path}")

        return None  # Skip unreadable images

    image = cv2.resize(image, (128, 128))

    features, \_ = hog(image, pixels\_per\_cell=(16, 16), cells\_per\_block=(2, 2), visualize=True)

    return features

# Load all images

image\_files = [os.path.join(extract\_path, f) for f in os.listdir(extract\_path) if f.endswith(".jpg")]

debug\_print("Total images found:", len(image\_files))

# Assign Labels

labels = ["Tumor" if "gl" in os.path.basename(f) else "No-Tumor" for f in image\_files]

debug\_print("Sample labels:", labels[:5])

# Encode Labels

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(labels)

# Extract HOG Features

X = np.array([extract\_hog\_features(f) for f in image\_files if extract\_hog\_features(f) is not None])

# Ensure X and y have the same length

if len(X) != len(y):

    debug\_print("Error: Mismatch in features and labels!", (len(X), len(y)))

    exit()

# Split Data (80% Train, 20% Test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

debug\_print("Dataset Loaded and Features Extracted!")

debug\_print(f"Number of Training Samples: {len(y\_train)}")

debug\_print(f"Classes in Training Set: {Counter(y\_train)}")

# A1: Calculate Entropy

def entropy(y):

    counts = Counter(y)

    total = len(y)

    if total == 0:

        return 0

    return -sum((count / total) \* math.log2(count / total) for count in counts.values())

train\_entropy = entropy(y\_train)

debug\_print("Entropy of Dataset:", train\_entropy)

# A2: Calculate Gini Index

def gini\_index(y):

    counts = Counter(y)

    total = len(y)

    return 1 - sum((count / total) \*\* 2 for count in counts.values())

train\_gini = gini\_index(y\_train)

debug\_print("Gini Index of Dataset:", train\_gini)

# A3 & A4: Identify Root Node using Information Gain

def information\_gain(X, y, feature\_index, bins=10):

    """Calculate Information Gain for a given feature."""

    total\_entropy = entropy(y)

    # Binning Continuous Features

    feature\_values = X[:, feature\_index]

    if len(np.unique(feature\_values)) == 1:  # If all feature values are the same

        return 0

    bin\_edges = np.linspace(feature\_values.min(), feature\_values.max(), bins + 1)

    binned\_feature = np.digitize(feature\_values, bin\_edges)

    # Calculate Entropy for each split

    split\_entropy = 0

    for value in np.unique(binned\_feature):

        subset\_y = y[binned\_feature == value]

        if len(subset\_y) == 0:

            continue

        split\_entropy += (len(subset\_y) / len(y)) \* entropy(subset\_y)

    return total\_entropy - split\_entropy

# Find best feature

best\_feature = max(range(X\_train.shape[1]), key=lambda i: information\_gain(X\_train, y\_train, i))

debug\_print(f"Best Root Node Feature: Feature {best\_feature}")

# A5: Build & Train Decision Tree

dt\_classifier = DecisionTreeClassifier(criterion="entropy", random\_state=42)

dt\_classifier.fit(X\_train, y\_train)

# Test Accuracy

y\_pred = dt\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

debug\_print("Decision Tree Accuracy:", accuracy)

# A6: Visualize Decision Tree

plt.figure(figsize=(12, 8))

tree.plot\_tree(dt\_classifier, feature\_names=[f"Feature {i}" for i in range(X\_train.shape[1])], class\_names=["No-Tumor", "Tumor"], filled=True)

plt.title("Decision Tree Visualization")

plt.show()

# Debugging: Check if tree is correctly built

debug\_print("Decision Tree Structure:", dt\_classifier.tree\_)

# A7: Plot Decision Boundaries (Using 2 Features)

if X\_train.shape[1] >= 2:  # Ensure at least 2 features exist

    X\_plot = X\_train[:, :2]  # Take first 2 features

    y\_plot = y\_train

    dt\_2d = DecisionTreeClassifier(criterion="entropy", random\_state=42)

    dt\_2d.fit(X\_plot, y\_plot)

    plt.figure(figsize=(8, 6))

    DecisionBoundaryDisplay.from\_estimator(dt\_2d, X\_plot, response\_method="predict", cmap=plt.cm.Paired, alpha=0.5)

    plt.scatter(X\_plot[:, 0], X\_plot[:, 1], c=y\_plot, edgecolors="k", cmap=plt.cm.Paired)

    plt.title("Decision Boundary of Decision Tree (Using 2 Features)")

    plt.xlabel("Feature 1")

    plt.ylabel("Feature 2")

    plt.show()

else:

    debug\_print("Error: Not enough features to plot decision boundaries.")