**Ques 9: Write a program to demonstrate the working of Decision Tree classifier.**

import math

data = [

{'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High', 'Wind': 'Weak', 'PlayTennis': 'No'},

{'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High', 'Wind': 'Strong', 'PlayTennis': 'No'},

{'Outlook': 'Overcast', 'Temperature': 'Hot', 'Humidity': 'High', 'Wind': 'Weak', 'PlayTennis': 'Yes'},

{'Outlook': 'Rainy', 'Temperature': 'Mild', 'Humidity': 'High', 'Wind': 'Weak', 'PlayTennis': 'Yes'},

{'Outlook': 'Rainy', 'Temperature': 'Cool', 'Humidity': 'Normal', 'Wind': 'Weak', 'PlayTennis': 'Yes'},

{'Outlook': 'Rainy', 'Temperature': 'Cool', 'Humidity': 'Normal', 'Wind': 'Strong', 'PlayTennis': 'No'},

{'Outlook': 'Overcast', 'Temperature': 'Cool', 'Humidity': 'Normal', 'Wind': 'Strong', 'PlayTennis': 'Yes'},

{'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'High', 'Wind': 'Weak', 'PlayTennis': 'No'},

{'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'Normal', 'Wind': 'Weak', 'PlayTennis': 'Yes'},

{'Outlook': 'Rainy', 'Temperature': 'Mild', 'Humidity': 'Normal', 'Wind': 'Weak', 'PlayTennis': 'Yes'},

{'Outlook': 'Sunny', 'Temperature': 'Mild', 'Humidity': 'Normal', 'Wind': 'Strong', 'PlayTennis': 'Yes'},

{'Outlook': 'Overcast', 'Temperature': 'Mild', 'Humidity': 'High', 'Wind': 'Strong', 'PlayTennis': 'Yes'},

{'Outlook': 'Overcast', 'Temperature': 'Hot', 'Humidity': 'Normal', 'Wind': 'Weak', 'PlayTennis': 'Yes'},

{'Outlook': 'Rainy', 'Temperature': 'Mild', 'Humidity': 'High', 'Wind': 'Strong', 'PlayTennis': 'No'}

]

**# Define Entropy Function**

def entropy(data, target\_attribute):

label\_counts = {}

for record in data:

label = record[target\_attribute]

if label not in label\_counts:

label\_counts[label] = 0

label\_counts[label] += 1

total = len(data)

entropy = 0.0

for key in label\_counts:

probability = label\_counts[key] / total

entropy -= probability \* math.log2(probability)

return entropy

## #Define the Information Gain Function

def information\_gain(data, attribute, target\_attribute):

total\_entropy = entropy(data, target\_attribute)

attribute\_values = set(record[attribute] for record in data)

weighted\_entropy = 0.0

for value in attribute\_values:

subset = [record for record in data if record[attribute] == value]

subset\_entropy = entropy(subset, target\_attribute)

weighted\_entropy += (len(subset) / len(data)) \* subset\_entropy

return total\_entropy - weighted\_entropy

## #Define the ID3 Algorithm Function

def id3(data, available\_features, target\_attribute):

target\_labels = [record[target\_attribute] for record in data]

if len(set(target\_labels)) == 1:

return target\_labels[0]

if not available\_features:

return max(set(target\_labels), key=target\_labels.count)

best\_feature = max(available\_features, key=lambda feature: information\_gain(data, feature, target\_attribute))

tree = {best\_feature: {}}

available\_features = [feature for feature in available\_features if feature != best\_feature]

for value in set(record[best\_feature] for record in data):

subtree\_data = [record for record in data if record[best\_feature] == value]

subtree = id3(subtree\_data, available\_features, target\_attribute)

tree[best\_feature][value] = subtree

return tree

features = ['Outlook', 'Temperature', 'Humidity', 'Wind']

target = 'PlayTennis'

decision\_tree = id3(data, features, target)

print("Decision Tree:")

print(decision\_tree)

**Ques 10: WAP to Implement CNN for Image Classification**

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

train\_dir = 'xray\_dataset\_covid19/train'

validation\_dir = 'xray\_dataset\_covid19/train'

train\_datagen = ImageDataGenerator(rescale=1./255)

validation\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(64, 64),

batch\_size=32,

class\_mode='binary' # Use 'categorical' for multi-class classification

)

validation\_generator = validation\_datagen.flow\_from\_directory (validation\_dir, target\_size = (64, 64), batch\_size=32,class\_mode='binary')

**# Building the Model**

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(64, 64, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(1, activation='sigmoid') # Use 'softmax' for multi-class classification

])

**# Compiling the Model**

model.compile(optimizer=Adam(learning\_rate=1e-4),

loss='binary\_crossentropy', # Use 'categorical\_crossentropy' for multi-class classification

metrics=['accuracy'])

**# Training the Model**

history = model.fit(

train\_generator,

steps\_per\_epoch=train\_generator.samples // train\_generator.batch\_size,

epochs=10,

validation\_data=validation\_generator,

validation\_steps=validation\_generator.samples // validation\_generator.batch\_size

)

**# Evaluating the Model**

import matplotlib.pyplot as plt

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

**Ques 11: WAP to Implement RNN for Text Classification**

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing import sequence

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

import matplotlib.pyplot as plt

**# Load and Preprocess the Data**

# Set parameters

max\_features = 10000 # Number of words to consider as features

maxlen = 500 # Cut texts after this number of words (among top max\_features most common words)

batch\_size = 32

# Load the data

print('Loading data...')

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

print(len(x\_train), 'train sequences')

print(len(x\_test), 'test sequences')

print('Pad sequences (samples x time)')

x\_train = sequence.pad\_sequences(x\_train, maxlen=maxlen)

x\_test = sequence.pad\_sequences(x\_test, maxlen=maxlen)

print('x\_train shape:', x\_train.shape)

print('x\_test shape:', x\_test.shape)

**#** **Build the RNN Model**

model = Sequential()

model.add(Embedding(max\_features, 128,))

model.add(SimpleRNN(128))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

print(model.summary())

**# Train the Model**

print('Training model...')

history = model.fit(x\_train, y\_train,batch\_size=batch\_size,epochs=5,validation\_split=0.2)

**# Evaluate the Model**

print('Evaluate model...')

score = model.evaluate(x\_test, y\_test,batch\_size=batch\_size)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

**# Plot Training History**

# Plot training & validation accuracy values

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight\_layout()

plt.show()