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()
```