Q5.ANN

Build an artificial neural network by implementing backpropogation algorithm and test the same using appropriate datasets

CODE:

```
import numpy as np

x = np.array(([2,9],[1,5],[3,6]), dtype=float)
y = np.array(([92],[86],[89]), dtype=float)

x = x/np.amax(x,axis=0)
y = y/100

def sigmoid(x):
   return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
   return x * (1 - x)

epoch = 7000
lr = 0.5
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh = np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh = np.random.uniform(size=(1,hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout = np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
   hinp1 = np.dot(x,wh)
   hinp = hinp1 + bh
   hlayer_act = sigmoid(hinp)

   outinp1 = np.dot(hlayer_act,wout)
   outinp = outinp1 + bout
   output = sigmoid(outinp)

   EO = y - output
```

```
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad

    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) * lr
    wh += x.T.dot(d_hiddenlayer) * lr

print("Input: \n" + str(x))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n", output)
```

OUTPUT:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.8966801 ]
 [0.87985564]
 [0.89282493]]
```

Q6. Naive Bayesian Classifier Algorithm

Write a program lo implement the naive Bayesian classifier for a sample training data set stored as a CSV file. Compute the accuracy of the classifier considering few test data sets

CODE:

```
import random
import math
import pandas as pd

def load_data(filename):
    """Loads data from a CSV file and ensures all columns are numeric."""
    df = pd.read_csv(filename)

    # Convert categorical data to numeric if necessary
    for column in df.columns:
        if df[column].dtype == 'object':
            df[column] = df[column].astype('category').cat.codes

    # Ensure all columns are numeric
    df = df.apply(pd.to_numeric, errors='coerce')

    # Drop rows with NaN values that could not be converted
    df = df.dropna()

    return df.values.tolist()

def split_dataset(dataset, split_ratio):
    """Splits a dataset into training and testing sets."""
    train_size = int(len(dataset) * split_ratio)
    train_set = []
    copy = list(dataset)
    while len(train_set) < train_size:
        index = random.randrange(len(copy))
        train_set.append(copy.pop(index))
    return [train_set, copy]

def separate_by_class(dataset):
    """Separates a dataset into classes."""
    separated = {}
```

```
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]  # Assuming the class label is the last element
        if class_value not in separated:
            separated[class_value] = []
        separated[class_value].append(vector)
    return separated

def mean(numbers):
    """Calculates the mean of a list of numbers."""
    return sum(numbers) / float(len(numbers)) if numbers else None

def stddev(numbers):
    """Calculates the standard deviation of a list of numbers."""
    if not numbers or len(numbers) <= 1:
        return None
    avg = mean(numbers)
    variance = sum((x - avg) ** 2 for x in numbers) / float(len(numbers) - 1)
    return math.sqrt(variance)

def summarize(dataset):
    """Summarizes a dataset by calculating the mean and standard deviation of
each attribute."""
    summaries = [(mean(attribute), stddev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]  # Remove the summary for the class label
    return summaries

def summarize_by_class(dataset):
    """Summarizes a dataset by class."""
    separated = separate_by_class(dataset)
    summaries = {}
    for class_value, instances in separated.items():
        summaries[class_value] = summarize(instances)
    return summaries

def calculate_probability(x, mean, stddev):
    """Calculates the probability of a value given a normal distribution."""
    if stddev == 0:
        return 0
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stddev, 2))))
```

```
    return (1 / (math.sqrt(2 * math.pi) * stddev)) * exponent

def calculate_class_probabilities(summaries, input_vector):
    """Calculates the probability of each class given an input vector."""
    probabilities = {}
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = 1
        for i in range(len(class_summaries)):
            mean, stddev = class_summaries[i]
            x = input_vector[i]
            probabilities[class_value] *= calculate_probability(x, mean, stddev)
    return probabilities

def predict(summaries, input_vector):
    """Predicts the class label of a new input vector based on the summaries of the
training data."""
    probabilities = calculate_class_probabilities(summaries, input_vector)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_label = class_value
            best_prob = probability
    return best_label

def get_predictions(summaries, test_set):
    """Predicts the class labels for a set of new input vectors."""
    predictions = []
    for i in range(len(test_set)):
        result = predict(summaries, test_set[i])
        predictions.append(result)
    return predictions

def get_accuracy(test_set, predictions):
    """Calculates the accuracy of a set of predictions."""
    correct = 0
    for i in range(len(test_set)):
        if test_set[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test_set))) * 100.0
```

```
def split_data(dataset, split_ratio):
    """Splits data into training and testing sets."""
    return split_dataset(dataset, split_ratio)

def main():
    """Machine Learning Classification Script"""
    filename = "C:/Users/DELL/OneDrive/Desktop/New
folder/newclass/trial/tennisdata.csv"
    split_ratio = 0.67

    # Load the dataset from the CSV file
    dataset = load_data(filename)

    if len(dataset) == 0:
        print("Error: Empty dataset")
        return

    # Split the dataset into training and testing sets
    training_set, testing_set = split_data(dataset, split_ratio)

    if len(training_set) == 0:
        print("Error: Empty training set")
        return

    print(f"Split {len(dataset)} rows into training: {len(training_set)} and test:
{len(testing_set)} rows")

    # Summarize the data by class
    summaries = summarize_by_class(training_set)

    # Get predictions for the testing set
    predictions = get_predictions(summaries, testing_set)

    # Calculate the accuracy
    accuracy = get_accuracy(testing_set, predictions)

    print(f"Accuracy of the classifier is: {accuracy:.3f}%")

if __name__ == "__main__":
    main()
```

Dataset:

Outlook	Temperatu	Humidity	Windy	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rainy	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

OUTPUT:

```
rive/Desktop/New folder/newclass/trial/ml6.py"
Split 14 rows into training: 9 and test: 5 rows
Accuracy of the classifier is: 40.000%
PS C:\Users\DELL\OneDrive\Desktop\New folder> &
rive/Desktop/New folder/newclass/trial/ml6.py"
Split 14 rows into training: 9 and test: 5 rows
Accuracy of the classifier is: 100.000%
```