ffaANN MLP V0.1

May 20, 2022

[ ]:

[1]:
```
import math
import numpy as np
import pandas as pd
# import seaborn as sns
# import matplotlib.pyplot as plt

# from sklearn.preprocessing import LabelEncoder
# from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn import preprocessing
from scipy.special import expit

from numpy.random import default_rng

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

import time

from ffaAnn_thread_V_I import *



class MultiLayerPerceptron():
    # ================== Activation Functions ================ #

    # accepts a vector or list and returns a list after performing␣
 ↪corresponding function on all elements

    @staticmethod
    def sigmoid(vectorSig):
        """returns 1/(1+exp(-x)), where the output values lies between zero and␣
 ↪one"""
        sig = expit(vectorSig)
        return sig
```

```
    @staticmethod
    def binaryStep(x):
        """ It returns '0' is the input is less then zero otherwise it returns␣
↪one """
        return np.heaviside(x, 1)

    @staticmethod
    def linear(x):
        """ y = f(x) It returns the input as it is"""
        return x

    @staticmethod
    def tanh(x):
        """ It returns the value (1-exp(-2x))/(1+exp(-2x)) and the value␣
↪returned will be lies in between -1 to 1"""
        return np.tanh(x)

    @staticmethod
    def relu(x):  # Rectified Linear Unit
        """ It returns zero if the input is less than zero otherwise it returns␣
↪the given input"""
        x1 = []
        for i in x:
            if i < 0:
                x1.append(0)
            else:
                x1.append(i)

        return x1

    @staticmethod
    def leakyRelu(x):
        """ It returns zero if the input is less than zero otherwise it returns␣
↪the given input"""
        x1 = []
        for i in x:
            if i < 0:
                x1.append((0.01 * i))
            else:
                x1.append(i)

        return x1

    @staticmethod
    def parametricRelu(self, a, x):
```

```
        """ It returns zero if the input is less than zero otherwise it returns␣
↪the given input"""
        x1 = []
        for i in x:
            if i < 0:
                x1.append((a * i))
            else:
                x1.append(i)

        return x1

    @staticmethod
    def softmax(self, x):
        """ Compute softmax values for each sets of scores in x"""
        return np.exp(x) / np.sum(np.exp(x), axis=0)

    # ============ Activation Functions Part Ends ============= #

    # ================= Distance Calculation ================== #

    @staticmethod
    def chebishev(self, cord1, cord2, exponent_h):
        dist = 0.0
        if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) ==␣
↪float and type(cord2) == float))):
            dist = math.pow((cord1 - cord2), exponent_h)
        else:
            for i, j in zip(cord1, cord2):
                dist += math.pow((i - j), exponent_h)
        dist = math.pow(dist, (1.0 / exponent_h))
        return dist

    @staticmethod
    def minimum_distance(self, cord1, cord2):
        # min(|x1-y1|, |x2-y2|, |x3-y3|, ...)
        dist = float('inf')
        if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) ==␣
↪float and type(cord2) == float))):
            dist = math.fabs(cord1 - cord2)
        else:
            for i, j in zip(cord1, cord2):
                temp_dist = math.fabs(i - j)
                if (temp_dist < dist):
                    dist = temp_dist
        return dist

    @staticmethod
```

```
    def maximum_distance(self, cord1, cord2):
        # max(|x1-y1|, |x2-y2|, |x3-y3|, ...)
        dist = float('-inf')
        if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) ==␣
↪float and type(cord2) == float))):
            dist = math.fabs(cord1 - cord2)
        else:
            for i, j in zip(cord1, cord2):
                temp_dist = math.fabs(i - j)
                if (temp_dist > dist):
                    dist = temp_dist
        return dist

    @staticmethod
    def manhattan(self, cord1, cord2):
        # |x1-y1| + |x2-y2| + |x3-y3| + ...
        dist = 0.0
        if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) ==␣
↪float and type(cord2) == float))):
            dist = math.fabs(cord1 - cord2)
        else:
            for i, j in zip(cord1, cord2):
                dist += math.fabs(i - j)
        return dist

    @staticmethod
    def eucledian(self, cord1, cord2):
        dist = 0.0
        if ((type(cord1) == int and type(cord2) == int) or ((type(cord1) ==␣
↪float and type(cord2) == float))):
            dist = math.pow((cord1 - cord2), 2)
        else:
            for i, j in zip(cord1, cord2):
                dist += math.pow((i - j), 2)
        return math.pow(dist, 0.5)

    # =========== Distance Calculation Ends ============== #

    def __init__(self, dimensions=(8, 5), all_weights=(0.1, 0.2),␣
↪fileName="iris"):

        """
        Args:
            dimensions : dimension of the neural network
            all_weights : the optimal weights we get from the bio-algoANN models
        """
```

```
        self.allPop_Weights = []
        self.allPopl_Chromosomes = []
        self.allPop_ReceivedOut = []
        self.allPop_ErrorVal = []

        self.all_weights = all_weights

        self.fitness = []

        # ================== Input dataset and corresponding output ==========================

        self.fileName = fileName
        self.fileName += ".csv"
        data = pd.read_csv(self.fileName)

        classes = []
        output_values_expected = []
        input_values = []

        # ~~~~ encoding ~~~~#

        # labelencoder = LabelEncoder()
        # data[data.columns[-1]] = labelencoder.fit_transform(data[data.columns[-1]])

        # one hot encoding - for multi-column
        # enc = OneHotEncoder(handle_unknown='ignore')
        # combinedData = np.vstack((data[data.columns[-2]], data[data.columns[-1]])).T
        # print(combinedData)
        # y = enc.fit_transform(combinedData).toarray()
        # y = OneHotEncoder().fit_transform(combinedData).toarray()

        #
        y = LabelBinarizer().fit_transform(data[data.columns[-1]])
        # print(y)

        # ~~~~ encoding ends~~~~#

        for j in range(len(data)):
            output_values_expected.append(y[j])

        # print(output_values_expected)

        input_values = []
        for j in range(len(data)):
```

```
            b = []
            for i in range(1, len(data.columns) - 1):
                b.append(data[data.columns[i]][j])
            input_values.append(b)

        self.X = input_values[:]
        self.Y = output_values_expected[:]

        # input and output
        self.X = input_values[:]
        self.Y = output_values_expected[:]

        self.dimension = dimensions
        # print(self.dimension)

        # ================ Finding Initial Weights ================ #

        self.pop = []  # weights
        reshaped_all_weights = []
        start = 0
        for i in range(len(self.dimension) - 1):
            end = start + self.dimension[i + 1] * self.dimension[i]
            temp_arr = self.all_weights[start:end]
            w = np.reshape(temp_arr[:], (self.dimension[i + 1], self.
↪dimension[i]))
            reshaped_all_weights.append(w)
            start = end
        self.pop.append(reshaped_all_weights)

        self.init_pop = self.all_weights

    # ================ Initial Weights Part Ends ================ #


    def Predict(self, chromo):
        # X, Y and pop are used
        self.fitness = []
        total_error = 0
        m_arr = []
        k1 = 0
        for i in range(len(self.dimension) - 1):
            p = self.dimension[i]
            q = self.dimension[i + 1]
            k2 = k1 + p * q
            m_temp = chromo[k1:k2]
            m_arr.append(np.reshape(m_temp, (p, q)))
            k1 = k2
```

```
        y_predicted = []
        for x, y in zip(self.X, self.Y):

            yo = x

            for mCount in range(len(m_arr)):
                yo = np.dot(yo, m_arr[mCount])
                yo = self.sigmoid(yo)

            # converting to sklearn acceptable form
            max_yo = max(yo)
            for y_vals in range(len(yo)):
                if(yo[y_vals] == max_yo):
                    yo[y_vals] = 1
                else:
                    yo[y_vals] = 0
            y_predicted.append(yo)
        return (y_predicted, self.Y)

    def main(self):
        Y_PREDICT, Y_ACTUAL = self.Predict(self.init_pop)
        Y_PREDICT = np.array(Y_PREDICT)
        Y_ACTUAL = np.array(Y_ACTUAL)

        n_classes = 3

        label_binarizer = LabelBinarizer()
        label_binarizer.fit(range(n_classes))
        Y_PREDICT = label_binarizer.inverse_transform(np.array(Y_PREDICT))
        Y_ACTUAL = label_binarizer.inverse_transform(np.array(Y_ACTUAL))

        # find error

        print("\n Actual / Expected", Y_ACTUAL)
        print("\n Predictions", Y_PREDICT)
        print("\n\nConfusion Matrix")
        print(confusion_matrix(Y_ACTUAL, Y_PREDICT))

        print("\n\nClassification Report")
        target_names = ['class 0', 'class 1', 'class 2']
        print(classification_report(Y_ACTUAL, Y_PREDICT,␣
↪target_names=target_names))
```

[2]:
```
start_time = time.time()
i = InputData(fileName="iris")
input_val, output_val = i.main()
```

```
end_time = time.time()
print("Time for inputting data : ", end_time - start_time)

print("============ Calling FFA to get best weights ===============")

start_time = time.time()
a = ffaAnn(initialPopSize=100, m=10, dimensions = [100,10],␣
 ↪input_values=input_val, output_values_expected=output_val, iterations = 100)

fit, b, weights, dim = a.main()

end_time = time.time()
print("Time taken : ", end_time - start_time)

print("\n Fitness : ", fit, "\n Best Weights : ", weights, "\n Dimensions : ",␣
 ↪dim)



import matplotlib.pyplot as plt
x=b[:]
z=[i for i in range(0,100)]
plt.plot(z,x)

plt.title("Firefly Algorithm")
plt.ylabel("Fitness")
plt.xlabel("Iterations")
end_time = time.time()
print("Time Taken : ", end_time - start_time)
```

```
Time for inputting data :  0.01881694793701172
============ Calling FFA to get best weights ===============
--------------GENERATION 0-----------
--------------GENERATION 1-----------
--------------GENERATION 2-----------
--------------GENERATION 3-----------
--------------GENERATION 4-----------
--------------GENERATION 5-----------
--------------GENERATION 6-----------
hi
hi
hi
hi
hi
hi
hi
hi
hihi
```

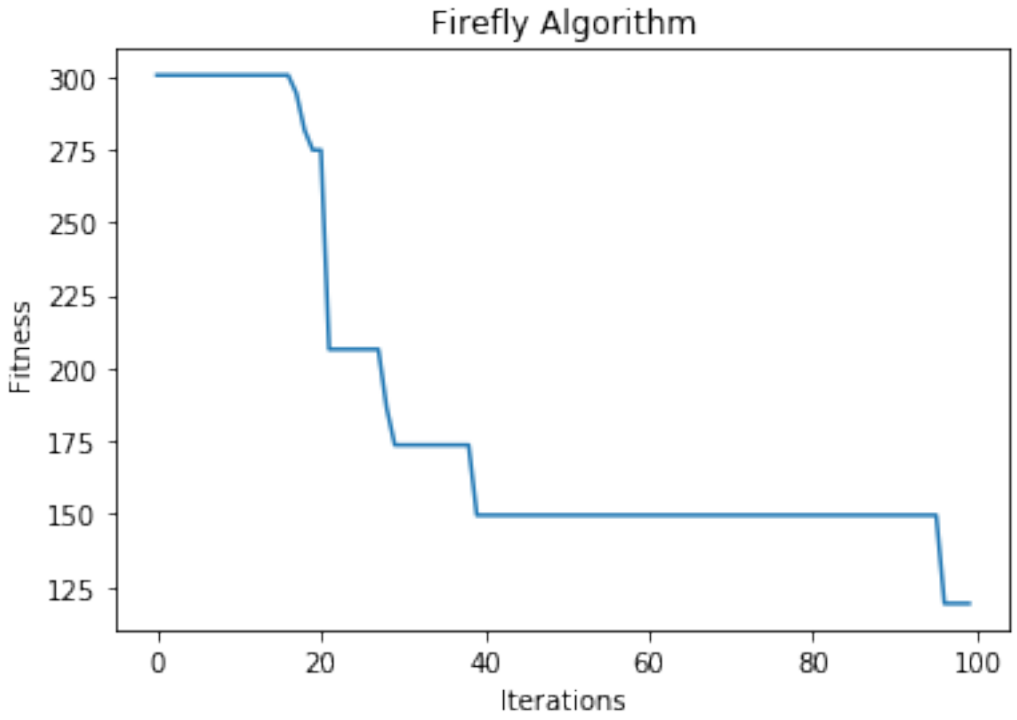
```
hi
----------GENERATION 7----------
----------GENERATION 8----------
----------GENERATION 9----------
----------GENERATION 10----------
----------GENERATION 11----------
----------GENERATION 12----------
----------GENERATION 13----------
----------GENERATION 14----------
----------GENERATION 15----------
----------GENERATION 16----------
----------GENERATION 17----------
----------GENERATION 18----------
----------GENERATION 19----------
----------GENERATION 20----------
----------GENERATION 21----------
----------GENERATION 22----------
----------GENERATION 23----------
----------GENERATION 24----------
----------GENERATION 25----------
----------GENERATION 26----------
----------GENERATION 27----------
----------GENERATION 28----------
----------GENERATION 29----------
----------GENERATION 30----------
----------GENERATION 31----------
----------GENERATION 32----------
----------GENERATION 33----------
----------GENERATION 34----------
----------GENERATION 35----------
----------GENERATION 36----------
----------GENERATION 37----------
----------GENERATION 38----------
----------GENERATION 39----------
----------GENERATION 40----------
----------GENERATION 41----------
----------GENERATION 42----------
----------GENERATION 43----------
----------GENERATION 44----------
----------GENERATION 45----------
----------GENERATION 46----------
----------GENERATION 47----------
----------GENERATION 48----------
----------GENERATION 49----------
----------GENERATION 50----------
----------GENERATION 51----------
----------GENERATION 52----------
```

```
--------------GENERATION 53----------
--------------GENERATION 54----------
--------------GENERATION 55----------
--------------GENERATION 56----------
--------------GENERATION 57----------
--------------GENERATION 58----------
--------------GENERATION 59----------
--------------GENERATION 60----------
--------------GENERATION 61----------
--------------GENERATION 62----------
--------------GENERATION 63----------
--------------GENERATION 64----------
--------------GENERATION 65----------
--------------GENERATION 66----------
--------------GENERATION 67----------
--------------GENERATION 68----------
--------------GENERATION 69----------
--------------GENERATION 70----------
--------------GENERATION 71----------
--------------GENERATION 72----------
--------------GENERATION 73----------
--------------GENERATION 74----------
--------------GENERATION 75----------
--------------GENERATION 76----------
--------------GENERATION 77----------
--------------GENERATION 78----------
--------------GENERATION 79----------
--------------GENERATION 80----------
--------------GENERATION 81----------
--------------GENERATION 82----------
--------------GENERATION 83----------
--------------GENERATION 84----------
--------------GENERATION 85----------
--------------GENERATION 86----------
--------------GENERATION 87----------
--------------GENERATION 88----------
--------------GENERATION 89----------
--------------GENERATION 90----------
--------------GENERATION 91----------
--------------GENERATION 92----------
--------------GENERATION 93----------
--------------GENERATION 94----------
--------------GENERATION 95----------
--------------GENERATION 96----------
--------------GENERATION 97----------
--------------GENERATION 98----------
--------------GENERATION 99----------
Fitness :  119.30006253385986
```

```
Time taken :  173.24341106414795

 Fitness :  -119.30006253385986
 Best Weights :  [0, 0, 0, 4, 0, -9, 10, 16, -17, -9, 0, 0, 18, 16, 0, 4, -2, 0,
19, 20, 4, -15, 10, 0, -1, 4, 0, 0, 0, 2, -1, -18, 0, 0, -2, 6, 7, -29, 0, 17,
4, 2, 14, 5, 0, 4, -22, -4, 5, 0, -32, 2, -5, 7, 0, -25, 14, 4, 3, 20, 0, 4, -2,
13, 13, 2, 16, -12, -5, 2, 0, -2, 0, -7, -2, 6, -14, 4, 7, 0, 0, -9, -6, 0, 0,
4, -10, -18, -12, 1, 0, -8, 0, 2, 7, -29, 0, 27, 4, -11, -6, 4, 2, 0, -2, 0, 2,
0, -6, 0, -14, 0, 0, 6, 2, 0, 0, 14, -9, 6, -11, 0, 5, 5, -4, -10, 11, 2, -7, 0,
-5, -2, 1, 11, 10, 0, -18, 14, 5, 9, -15, 2, 0, -2, 7, 22, 0, 21, 2, -5, 32, 4,
0, 6, 2, 8, 6, -9, 0, -11, 4, 0, -4, 0, 10, 4, 2, 9, -20, 0, -14, -9, 6, 0, 2,
-6, 7, -4, 5, 1, -2, -22, -14, 0, 4, 2, -7, -4, -2, 0, -12, 9, 4, 7, -10, -24,
14, -5, 0, 4, 5, -5, -2, 12, -10, -2, 4, -9, -16, -9, -14, 0, -5, -2, 4, 17,
-12, -4, 7, 8, 0, -24, 10, 12, -11, 7, -20, -13, 0, 0, 0, 13, 9, 0, -4, 0, 3, 0,
8, 0, -2, 4, 13, 9, -5, -28, 0, 11, -4, -9, 5, 0, 10, 2, -4, -8, 2, 0, 2, -5, 2,
-16, 0, -18, 5, -43, -2, 1, -3, 0, -7, -16, 15, -24, 18, -3, -2, 0, 20, -5, 2,
0, -5, 11, 0, -2, 10, -12, 0, -11, 10, -17, -4, 2, 7, -11, 6, 0, -17, 0, 5, 3,
19, 2, -3, 7, -8, -6, 0, -11, 2, 0, 0, 0, 5, -12, -6, 5, 0, 0, 11, 12, 0, 22,
-2, -13, 0, 0, 2, 0, 4, -2, 20, 0, 1, 0, 12, -15, 0, 4, -8, 0, 18, 10, 2, 0, -5,
-19, -11, 6, 9, 0, 2, 32, 20, 17, -9, -19, 3, 0, 10, 0, -2, -8, 0, 2, 4, 6, 10,
12, -7, 2, -7, 0, 6, 20, 7, -11, -18, 0, 5, -2, -6, -7, -28, 0, 4, -5, 11, -2,
-11, 0, 0, 4, -2, 8, 5, -14, 0, 29, 7, -4, 0, -12, 0, -14, -11, -4, 2, 12, -8,
0, 10, 0, -8, 2, 2, -12, -9, 4, -9, -10, -5, 2, 0, 0, 5, -18, 11, 7, 5, 0, 28,
11, -19, -5, 18, 0, -6, 0, -5, -3, 0, 5, 8, 16, -2, 4, -14, 2, 4, 7, 6, -28, 0,
0, -2, -22, -6, 0, 2, 0, 0, -15, 6, -11, -2, 31, -8, 0, 17, 2, 0, -20, -14, -29,
5, 0, 7, 5, 7, 12, 9, 0, -6, -4, 5, 2, 2, -4, 5, 5, 20, 2, 15, -8, 0, 0, -8, 0,
0, 4, -13, 2, 0, 7, 12, 6, 8, -13, 8, 0, 0, 11, 0, 0, 3, 10, 16, -6, -25, 8, 9,
4, -11, -2, -9, -6, 19, 2, 0, -20, 5, -17, 0, 3, 11, 10, -16, -3, -9, -11, 0, 0,
-4, 18, 2, 9, 4, 0, 0, 9, 2, -4, -11, 5, 13, 18, 12, -7, -2, -29, -8, 2, 18, 0,
-4, -20, 7, 0, 0, -5, 9, 0, 4, 0, 2, -4, 0, 19, -23, 5, -6, 2, -11, 29, -20, 12,
0, -17, -11, 22, -5, 10, 9, -8, 7, 15, -11, 0, -2, 0, 11, -2, 0, 2, 0, 18, 7, 4,
10, 6, 0, 0, -3, 2, -9, -10, 0, -13, 0, 0, 2, 16, 19, -2, 18, -4, -11, 2, 4, 14,
2, -8, 9, 2, -10, 6, -6, 0, 0, -22, -11, -18, 0, 22, -2, 0, 7, 2, 4, 11, 19, 1,
24, -4, 5, -2, -10, 0, 10, -18, 8, -5, 12, 5, 11, 5, -19, 10, -7, 0, -4, 4, -8,
-33, -9, 0, 2, 14, 0, 0, -17, 8, -2, 5, -13, 0, 4, -8, 0, 8, -2, -5, 10, -5, 4,
0, 5, -13, -13, 3, 0, -10, 16, -15, -4, 7, -18, 0, 2, -2, 0, 1, -4, -10, 0, 7,
0, -17, -11, 3, -14, 7, -3, 6, -15, -4, 0, -5, 7, 0, 14, 0, 11, 0, 4, -17, 8, 0,
0, 6, 0, 20, 0, -6, -23, -6, 15, 21, 0, -15, 0, -2, -9, -23, 9, -3, 8, 8, 0, 0,
2, 0, -3, 0, 8, 0, 2, 5, 0, 12, -5, -4, 0, -15, 0, 0, -31, -8, 4, -6, -13, 0, 2,
-6, -27, 0, -2, 0, -12, 15, 0, 6, 0, -22, -9, 15, 5, 0, 4, 35, -2, 0, 0, 3, -2,
-18, 5, 7, 0, 17, 0, -5, -2, 0, 4, -7, -16, -14, 9, 8, 0, 0, -15, 4, 15, -8, 0,
0, 2, -19, -13, 0, 6, 0, 17, -4, 16, 0, 2, 2, -5, 0, 0, -2, -18, -13, 0, -2, 6,
-2, -4, 6, 2, -11, 24, 0, -5, 4, -1, -19, -18, 19, -3, 7, -5, 13, 21, 0, 0, -5,
7, 6, 12, 20, 0, 16, 1, -9, 0, 19, -3, -2, 1, 4, 0, 21, 0, 7, 9, 0, -16, 0, -5,
-15, 0, 0, -6, -34, 0, 5, -7, 4, 15, 7, 0, -18, -10, 11, -10, 12, -18, 17, 34,
0, -2, -2, -18, 0, -12, -2, 13, 32, 38, -5, 0, 0, 6, 8, 10, -4, -20, 2, 17, 0,
2, 21, 2, 5, -14, 0, 0, 6, -20, 0, -17, 10, 5, 0, 2, 2, -7, 0, 10, -3, -2, -2,
-18, 10, 0, 0, -9, -2, 5, 8, -4, 2, 5, 0, 9, 5, -6, 23, -22, 22, 0, 0, 5, -4,
```

```
-7, -16, -34, 6, -2, 0, 5, -2, -8, -7, 13, -2, 2, -2, 2, 0, -5, -5, -19, -7, -6,
-6, -8, 11, -26, 15, 1, -8, 6, -9, 3, 0, 19, 0, 6, -15, -10, 4, -15, -2, -11, 3,
-30, 6, 0, -6, 0, 12, 2, -7, -11, -8, 2, 0, 0, 12, 0, -2, 4, 9, 8, 0, 4, 2, 0,
-7, 10, -5, -25, -2, 11, -4, 0, 0, -12, -6, -5, -5, -4, 0, 0, 10, 2, -9, -6, -9,
0, -14, 10, -9, 0, 23, -8, 17, -4, -5, -2, -13, 10, -2, -15, 4, -5, 2, 2, -11,
4, 4, -4, -2, 0, -8, -22, -9, 0, 0, -2, 0, 7, 0, -7, -11, -7, -6, 4, 5, 7, -12,
0, 0, 4, -7, 2, -9, -7, 0, 0, 2, 0, 0, -7, -11, 14, 0, -2, -2, 0, 0, -11, 0, -5,
-9, 0, 2, 6, 0, 21, 0, 2, 18, -6, 4, -2, 10, -4, 0, 4, 2, -15, 7, -21, 22, 0, 5,
6, 0, 11, -4, 2, 5, -28, -4, 9, -2, 5, 2, 0, 12, 14, 5, 0, -4, 31, -4, 0, 11,
-4, -5, 4, 4, 7, 8, 6, -14, 0, -12, -5, -4, 8, -4, 19, 0, -4, 9, 0, -4, -7, 2,
0, -10, 0, 0, 0, -12, -2, 20, 2, 1, 5, 0, -11, -3, 2, 0, 0, 0, -2, 22, 13, 5,
-5, 0, -10, -3, 0, 8, 4, -28, 8, 19, -9, -9, -6, -6, 4, -2, -8, -20, -24, -5,
10, 0, 0, 15, -26, 2, 6, -11, -2, 29, 0, 0, 0, -11, 2, 0, -4, -10, 0, 0, 4, 9,
0, -7, 10, 2, 5, 4, 15, 3, -33, -11, 0, 9, 2, -15, 9, -1, 0, -6, -5, 2, -7, -11,
0, -9, 11, 17, -11, -2, -4, 5, 4, 0, 29, 2, -10, 4, -5, 15, 2, 5, 0, 5, 0, 2, 4,
5, -6, 0, 7, 2, -19, 0, -16, 2, 6, -2, 0, 0, 0, 0, 0, -3, 39, 0, -11, 14, -9, 0,
0, -10, -2, 0, 0, 0, 0, 1, -2, 2, 17, -9, 10, 7, -11, 7, -6, 7, 0, 0, -15, 2, 0,
-8, 0, 0, -12, 0, 0, 16, 0, -8, 0, -1, 8, 0, 11, 11, 0, -6, 8, -2, -1, 2, 2, -9,
0, 2, -9, 12, -14, -10, 0, -9, 10, 2, 0, 4, 8, 5, -21, 2, 1, 0, -10, -13, 3, -2,
5, 0, 0, -2, 10, -18, 16, -1, -3, 23, 18, -11, -15, 4, 0, 8, 5, -5, -13, 0, 1,
-8, -6, 0, 1, 0, -17, 10, 5, 10, 7, -9]
 Dimensions :  [4, 100, 10, 3]
Time Taken :  173.281170129776
```



[3]:
```
print("\n\n============= MLP Program Begins ============")

start_time = time.time()
print("Training")
m = MultiLayerPerceptron(fileName="iris_train", dimensions=dim,␣
 ↪all_weights=weights)
m.main()
end_time = time.time()
print("Time taken = ", end_time - start_time)
```

```
============= MLP Program Begins ============
Training

 Actual / Expected [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]

 Predictions [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2
 2 2 2 2 2 2 2 2 2]


Confusion Matrix
[[40  0  0]
 [ 0 38  2]
 [ 0  1 39]]


Classification Report
              precision    recall  f1-score   support

     class 0       1.00      1.00      1.00        40
     class 1       0.97      0.95      0.96        40
     class 2       0.95      0.97      0.96        40

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120

Time taken =  0.037489891052246094
```

[4]:
```
start_time = time.time()
print("Testing")
m = MultiLayerPerceptron(fileName="iris_test", dimensions=dim,␣
 ↪all_weights=weights)
m.main()

end_time = time.time()
print("Time taken = ", end_time - start_time)
```

```
Testing

 Actual / Expected [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]

 Predictions [0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2]


Confusion Matrix
[[ 9  1  0]
 [ 0 10  0]
 [ 0  0 10]]


Classification Report
              precision    recall  f1-score   support

     class 0       1.00      0.90      0.95        10
     class 1       0.91      1.00      0.95        10
     class 2       1.00      1.00      1.00        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Time taken =  0.016193866729736328
```

[ ]: