psoANN without threading MLP V0.1

May 20, 2022

[ ]:

```
```

[5]:

```
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 psoAnn_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
```

```python
    @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))
```

[6]:

```
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 PSO to get best weights ===============")

start_time = time.time()

a = psoAnn(initialPopSize=100, input_values=input_val,␣
 ↪output_values_expected=output_val, iterations = 100, dimensions = [100,10])

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("PSO")
plt.ylabel("Fitness")
plt.xlabel("Time")
end_time = time.time()
print("Time Taken : ", end_time - start_time)
```

```
Time for inputting data :  0.023538589477539062
============ Calling PSO to get best weights ===============
--------------GENERATION 0-----------
-193.5853982428287
--------------GENERATION 1-----------
-171.32883391496557
--------------GENERATION 2-----------
-173.05082791996438
--------------GENERATION 3-----------
-190.1173849740747
--------------GENERATION 4-----------
-194.22543480573725
--------------GENERATION 5-----------
-192.99996437025362
--------------GENERATION 6-----------
-198.44650465850947
```

```
--------------GENERATION 7-----------
-173.9999965581465
--------------GENERATION 8-----------
-170.9435669717243
--------------GENERATION 9-----------
-116.00000008735586
--------------GENERATION 10-----------
-110.00010962193154
--------------GENERATION 11-----------
-108.99995749187661
--------------GENERATION 12-----------
-106.99999999868518
--------------GENERATION 13-----------
-120.0
--------------GENERATION 14-----------
-131.0
--------------GENERATION 15-----------
-141.9999999438153
--------------GENERATION 16-----------
-135.53764928301277
--------------GENERATION 17-----------
-136.98635100109706
--------------GENERATION 18-----------
-120.9991567790845
--------------GENERATION 19-----------
-102.95252038780423
--------------GENERATION 20-----------
-107.60924956078728
--------------GENERATION 21-----------
-104.06078834725231
--------------GENERATION 22-----------
-109.31140702039819
--------------GENERATION 23-----------
-109.0039600480796
--------------GENERATION 24-----------
-123.06688104180179
--------------GENERATION 25-----------
-150.75650710695763
--------------GENERATION 26-----------
-145.80230039875255
--------------GENERATION 27-----------
-197.39288138940367
--------------GENERATION 28-----------
-199.0606110315642
--------------GENERATION 29-----------
-186.74196602920517
--------------GENERATION 30-----------
-192.88931508370337
```

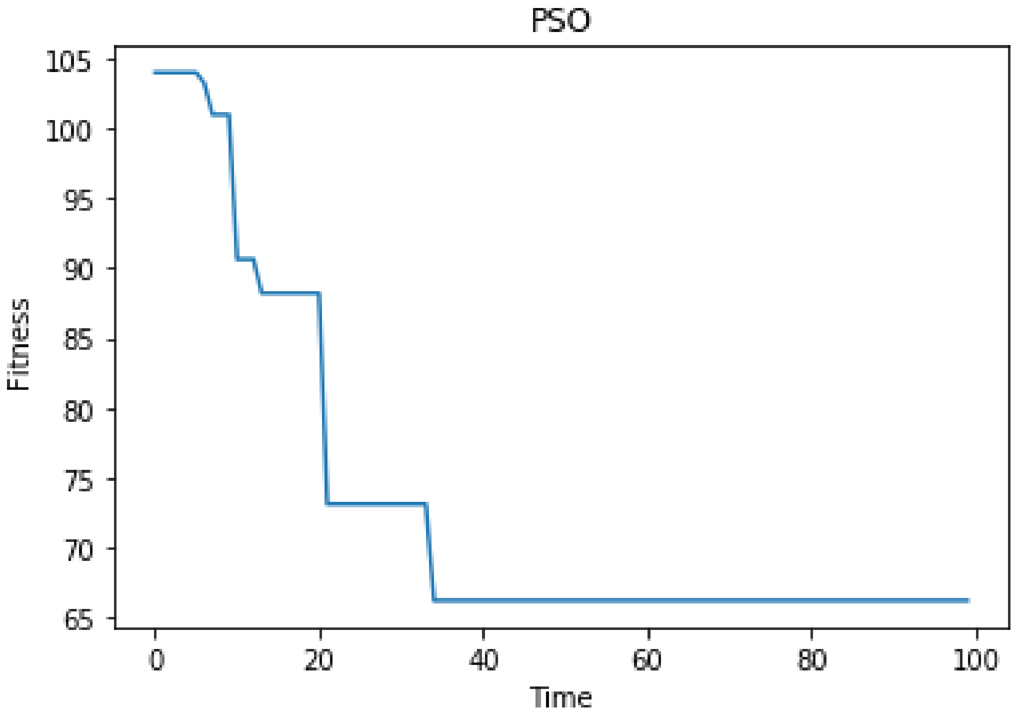
```
--------------GENERATION 31-----------
-194.95638408318226
--------------GENERATION 32-----------
-194.98201696441646
--------------GENERATION 33-----------
-195.87014168695993
--------------GENERATION 34-----------
-195.980303409957
--------------GENERATION 35-----------
-191.16717921155015
--------------GENERATION 36-----------
-190.95766731274452
--------------GENERATION 37-----------
-164.7695895997997
--------------GENERATION 38-----------
-147.77789118797904
--------------GENERATION 39-----------
-140.24608528626567
--------------GENERATION 40-----------
-137.26542821710035
--------------GENERATION 41-----------
-132.27558099002383
--------------GENERATION 42-----------
-132.27558099002383
--------------GENERATION 43-----------
-133.36347314982885
--------------GENERATION 44-----------
-135.36160632900817
--------------GENERATION 45-----------
-134.36360866472853
--------------GENERATION 46-----------
-130.36867611825167
--------------GENERATION 47-----------
-133.7507183560258
--------------GENERATION 48-----------
-136.9248488809402
--------------GENERATION 49-----------
-151.94264663090004
--------------GENERATION 50-----------
-152.94102831466114
--------------GENERATION 51-----------
-152.94102831466114
--------------GENERATION 52-----------
-152.95680517920576
--------------GENERATION 53-----------
-152.95680517920576
--------------GENERATION 54-----------
-152.95680517920576
```

```
--------------GENERATION 55-----------
-152.97081814220718
--------------GENERATION 56-----------
-151.97907833246262
--------------GENERATION 57-----------
-152.97198322152764
--------------GENERATION 58-----------
-152.9721044432835
--------------GENERATION 59-----------
-143.88726102392326
--------------GENERATION 60-----------
-143.6020710863904
--------------GENERATION 61-----------
-150.0677740647825
--------------GENERATION 62-----------
-150.85109176555676
--------------GENERATION 63-----------
-154.75071765684635
--------------GENERATION 64-----------
-153.80056310019097
--------------GENERATION 65-----------
-153.80057411420347
--------------GENERATION 66-----------
-153.80057411420347
--------------GENERATION 67-----------
-153.80057411420347
--------------GENERATION 68-----------
-154.7507183278909
--------------GENERATION 69-----------
-154.7507183278909
--------------GENERATION 70-----------
-154.7507183278909
--------------GENERATION 71-----------
-154.7507183278909
--------------GENERATION 72-----------
-155.70085175409193
--------------GENERATION 73-----------
-155.70085175409193
--------------GENERATION 74-----------
-155.70085175409193
--------------GENERATION 75-----------
-154.7507182731531
--------------GENERATION 76-----------
-154.7507181482935
--------------GENERATION 77-----------
-154.7507181482935
--------------GENERATION 78-----------
-154.69963929588644
```

```
--------------GENERATION 79-----------
-153.65100538482884
--------------GENERATION 80-----------
-156.74219379441158
--------------GENERATION 81-----------
-157.3526793979053
--------------GENERATION 82-----------
-157.3529697995121
--------------GENERATION 83-----------
-146.14896611502041
--------------GENERATION 84-----------
-139.8616188826853
--------------GENERATION 85-----------
-156.70596894763668
--------------GENERATION 86-----------
-156.14384129941016
--------------GENERATION 87-----------
-148.99686752484342
--------------GENERATION 88-----------
-148.99686752484342
--------------GENERATION 89-----------
-148.99686752484342
--------------GENERATION 90-----------
-148.99686752484342
--------------GENERATION 91-----------
-148.99686752484342
--------------GENERATION 92-----------
-148.3504692258899
--------------GENERATION 93-----------
-149.7384839173034
--------------GENERATION 94-----------
-154.1109806684312
--------------GENERATION 95-----------
-153.49110756573373
--------------GENERATION 96-----------
-153.49110756279705
--------------GENERATION 97-----------
-153.49110755756294
--------------GENERATION 98-----------
-153.49110755756294
--------------GENERATION 99-----------
-153.49110755756294
Global :  66.18024896191545
Time taken :  50.03552985191345

 Fitness :  [-66.18024896191545]
 Best Weights :  [-104.85719682   10.03978924  -41.58988218 …  -39.09213131
-5.52965813
```

```
  -41.58402939]
 Dimensions :  [4, 100, 10, 3]
Time Taken :  50.05182385444641
```



```
[7]: 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 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]
 Predictions [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0
 0 0 0 1 1 1 0 1 2 2 1 1 2 1 1 0 1 1 1 2 1 0 1 2 1 0 1 1 1 1 1 1 1 1 1 1 2
 2 2 1 0 2 1 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 0 2 2 0 2 2 2 2 2 2 0 0
 2 2 2 2 2 2 2 2 2]


Confusion Matrix
[[39  1  0]
 [ 5 26  9]
 [ 5  0 35]]


Classification Report
              precision    recall  f1-score   support

     class 0       0.80      0.97      0.88        40
     class 1       0.96      0.65      0.78        40
     class 2       0.80      0.88      0.83        40

    accuracy                           0.83       120
   macro avg       0.85      0.83      0.83       120
weighted avg       0.85      0.83      0.83       120

Time taken =  0.02354288101196289
```

[8]:
```python
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 1 2 2 2 2 2 2 2 2 2 2]

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


Confusion Matrix
[[ 9  1  0]
 [ 1  8  1]
 [ 0  0 10]]
```

```
Classification Report
              precision    recall  f1-score   support

     class 0       0.90      0.90      0.90        10
     class 1       0.89      0.80      0.84        10
     class 2       0.91      1.00      0.95        10

    accuracy                           0.90        30
   macro avg       0.90      0.90      0.90        30
weighted avg       0.90      0.90      0.90        30

Time taken =  0.012535810470581055
```

[ ]: