psoANN 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 psoAnn_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
```

```python
    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
```

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

```python
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, m=10, 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 :  0.015381574630737305
============ Calling PSO to get best weights ===============
--------------GENERATION 0-----------
-248.51308995240814
--------------GENERATION 1-----------
-249.9974025700157
--------------GENERATION 2-----------
-249.1693744959391
--------------GENERATION 3-----------
-249.9974467734275
--------------GENERATION 4-----------
-249.00000000001853
--------------GENERATION 5-----------
-237.96758065438723
--------------GENERATION 6-----------
-215.99999999999508
```

```
--------------GENERATION 7-----------
-204.00001653324261
--------------GENERATION 8-----------
-203.92682667652082
--------------GENERATION 9-----------
-206.82436412821843
--------------GENERATION 10-----------
-202.99999722948934
--------------GENERATION 11-----------
-150.999935747535
--------------GENERATION 12-----------
-149.99999999999915
--------------GENERATION 13-----------
-150.0
--------------GENERATION 14-----------
-150.0
--------------GENERATION 15-----------
-150.0
--------------GENERATION 16-----------
-150.0
--------------GENERATION 17-----------
-149.99999999996624
--------------GENERATION 18-----------
-145.96337158842206
--------------GENERATION 19-----------
-149.99971511840522
--------------GENERATION 20-----------
-148.390944310847
--------------GENERATION 21-----------
-137.23216815581006
--------------GENERATION 22-----------
-128.00746423932443
--------------GENERATION 23-----------
-137.99963219007728
--------------GENERATION 24-----------
-143.40372811150294
--------------GENERATION 25-----------
-167.00697245747614
--------------GENERATION 26-----------
-167.00927856274234
--------------GENERATION 27-----------
-170.00902514703466
--------------GENERATION 28-----------
-168.72655680769233
--------------GENERATION 29-----------
-168.86044327348242
--------------GENERATION 30-----------
-170.64790427755884
```

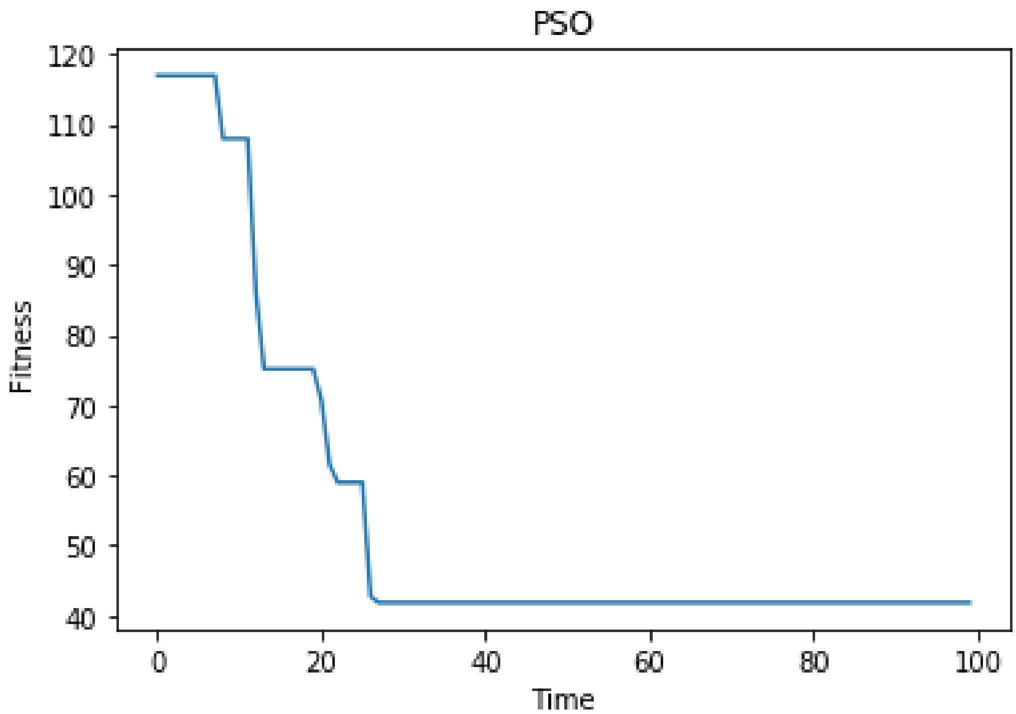
```
--------------GENERATION 31-----------
-160.52724784196366
--------------GENERATION 32-----------
-146.9452066865872
--------------GENERATION 33-----------
-141.741666415709
--------------GENERATION 34-----------
-141.6810335614381
--------------GENERATION 35-----------
-141.6810335614381
--------------GENERATION 36-----------
-140.87211432141098
--------------GENERATION 37-----------
-140.94593889116712
--------------GENERATION 38-----------
-143.78533709558195
--------------GENERATION 39-----------
-143.95371299177813
--------------GENERATION 40-----------
-144.0518908321367
--------------GENERATION 41-----------
-144.35142773322832
--------------GENERATION 42-----------
-145.50954748756055
--------------GENERATION 43-----------
-145.50954748756055
--------------GENERATION 44-----------
-145.50954748756055
--------------GENERATION 45-----------
-145.50954748756055
--------------GENERATION 46-----------
-145.50954748756055
--------------GENERATION 47-----------
-145.50954748756055
--------------GENERATION 48-----------
-145.50954748756055
--------------GENERATION 49-----------
-145.50954748756055
--------------GENERATION 50-----------
-145.50954748756055
--------------GENERATION 51-----------
-145.50954748756055
--------------GENERATION 52-----------
-140.40489298712
--------------GENERATION 53-----------
-140.40489298712
--------------GENERATION 54-----------
-140.40489298712
```

```
--------------GENERATION 55-----------
-140.40489298712
--------------GENERATION 56-----------
-140.40489298712
--------------GENERATION 57-----------
-140.40489298705023
--------------GENERATION 58-----------
-140.40489293311717
--------------GENERATION 59-----------
-143.9915981335279
--------------GENERATION 60-----------
-143.99159790831382
--------------GENERATION 61-----------
-143.99159790831382
--------------GENERATION 62-----------
-143.99159813350943
--------------GENERATION 63-----------
-143.99159813350943
--------------GENERATION 64-----------
-143.99159813350943
--------------GENERATION 65-----------
-143.99159813350943
--------------GENERATION 66-----------
-143.99159813350943
--------------GENERATION 67-----------
-143.99159813350943
--------------GENERATION 68-----------
-143.99159813350943
--------------GENERATION 69-----------
-143.99159813350943
--------------GENERATION 70-----------
-143.99159813350943
--------------GENERATION 71-----------
-143.99159813350943
--------------GENERATION 72-----------
-143.99159813350943
--------------GENERATION 73-----------
-143.99159813350943
--------------GENERATION 74-----------
-143.99159813350943
--------------GENERATION 75-----------
-143.99159813350943
--------------GENERATION 76-----------
-143.99159813350943
--------------GENERATION 77-----------
-143.99159813350943
--------------GENERATION 78-----------
-143.99159813350943
```

```
--------------GENERATION 79-----------
-140.40489293311003
--------------GENERATION 80-----------
-140.40489293311003
--------------GENERATION 81-----------
-140.40489293311003
--------------GENERATION 82-----------
-140.40489293311003
--------------GENERATION 83-----------
-140.40489293311003
--------------GENERATION 84-----------
-140.40489293311003
--------------GENERATION 85-----------
-140.40489293311003
--------------GENERATION 86-----------
-140.40489293311003
--------------GENERATION 87-----------
-140.40489293311003
--------------GENERATION 88-----------
-140.40489293311003
--------------GENERATION 89-----------
-140.40489293311003
--------------GENERATION 90-----------
-143.99159813353864
--------------GENERATION 91-----------
-143.99159813353864
--------------GENERATION 92-----------
-143.99159813353864
--------------GENERATION 93-----------
-143.99159813353864
--------------GENERATION 94-----------
-143.99159813353864
--------------GENERATION 95-----------
-143.99159813353864
--------------GENERATION 96-----------
-143.9916052272588
--------------GENERATION 97-----------
-143.9916052272588
--------------GENERATION 98-----------
-143.9916052272588
--------------GENERATION 99-----------
-143.9916052272588
Global :  41.80745683206267
Time taken :  72.68625950813293

 Fitness :  [-41.80745683206267]
 Best Weights :  [-86.23843964 -58.80409054  54.30552104 …   0.22068293
47.92238716
```

```
 57.78789971]
 Dimensions :  [4, 100, 10, 3]
Time Taken :  72.72134065628052
```



[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 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]
```

```
 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 0 1 1 2 1 1 1 2 1 1 2 1 2 1 1 2 1 2 2 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 2
 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 1]


Confusion Matrix
[[40  0  0]
 [ 0 25 15]
 [ 0  5 35]]


Classification Report
              precision    recall  f1-score   support

     class 0       1.00      1.00      1.00        40
     class 1       0.83      0.62      0.71        40
     class 2       0.70      0.88      0.78        40

    accuracy                           0.83       120
   macro avg       0.84      0.83      0.83       120
weighted avg       0.84      0.83      0.83       120

Time taken =  0.01929616928100586
```

[4]:
```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 0 0 0 0 0 0 0 2 2 1 1 2 1 2 1 2 1 2 2 2 2 2 1 2 1 2 2]


Confusion Matrix
[[10  0  0]
 [ 0  5  5]
 [ 0  2  8]]
```

```
Classification Report
              precision    recall  f1-score   support

     class 0       1.00      1.00      1.00        10
     class 1       0.71      0.50      0.59        10
     class 2       0.62      0.80      0.70        10

    accuracy                           0.77        30
   macro avg       0.78      0.77      0.76        30
weighted avg       0.78      0.77      0.76        30

Time taken =  0.010092973709106445
```

[ ]:

```

```