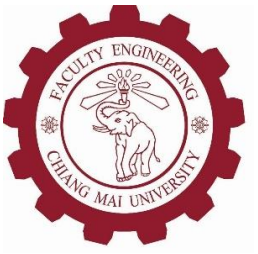


รายงาน

Computer Assignment 4

โดย

นายศุภกฤต ก๋องคำ

650610858

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา 261456

สาขาวิชาวิศวกรรมหุ่นยนต์และปัญญาประดิษฐ์

ภาคเรียนที่1 ปีการศึกษา 2566

มหาวิทยาลัยเชียงใหม่

จงเขียน program สำหรับการ Train Multilayer Perceptron โดยใช้ Particle Swarm Optimization (PSO) สำหรับการทำ prediction Benzene concentration โดยเป็นการ predict 5 วันล่วงหน้า และ 10 วันล่วงหน้า โดยให้ใช้ attribute เบอร์ 3,6,8,10,11,12,13 และ 14 เป็น input ส่วน desire output เป็น attribute เบอร์ 5

ปัญหา :

ให้ทำการทดลองกับ AirQualityUCI (Air Quality Data Set จาก UCI Machine learning Repository) โดยที่ data set นี้มีทั้งหมด 9358 sample และมี 14 attribute ดังนี้

0 Date (DD/MM/YYYY)

1 Time (HH.MM.SS)

2 True hourly averaged concentration CO in mg/m^3 (reference analyzer)

3 PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)

4 True hourly averaged overall Non Metanic HydroCarbons concentration in microg/m^3 (reference analyzer)

5 True hourly averaged Benzene concentration in microg/m^3 (reference analyzer)

6 PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)

7 True hourly averaged NOx concentration in ppb (reference analyzer)

8 PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)

9 True hourly averaged NO2 concentration in microg/m^3 (reference analyzer)

10 PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)

11 PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)

12 Temperature in Â°C

13 Relative Humidity (%)

14 AH Absolute Humidity

ให้ทำการทดลองโดยใช้ 10% cross validation เพื่อทดสอบ validity ของ network ที่ได้ และให้ทำการเปลี่ยนแปลงจำนวน hidden layer และ nodes

หลักการ:

แบ่งข้อมูลเป็น 2 fold คือ 1:9 และ 9:1 โดย 9 เป็น Train และ 1 เป็น Test จากนั้นสุ่ม particles และนำไปหาค่า MAE ผ่าน MLP และให้ PSO หาคำตอบที่ดีที่สุด คือ best MAE และ best weights จากนั้นนำ best weight มาทดลองกับ Test case ทำการทดลองปรับค่าของโมเดลและสังเกต

1.ขั้นตอนการสร้างโปรแกรม

โดยทำการคัดลอกข้อมูลจากไฟล์ .xlsx มาอยู่ในไฟล์ .txt

1360	150	11.9	1046	166	1056	113	1692	1268	13.6	48.9	0.7578
1292	112	9.4	955	103	1174	92	1559	972	13.3	47.7	0.7255
1402	88	9.0	939	131	1140	114	1555	1074	11.9	54.0	0.7502
1376	80	9.2	948	172	1092	122	1584	1203	11.0	60.0	0.7867
1272	51	6.5	836	131	1205	116	1490	1110	11.2	59.6	0.7888
1197	38	4.7	750	89	1337	96	1393	949	11.2	59.2	0.7848
1185	31	3.6	690	62	1462	77	1333	733	11.3	56.8	0.7603
1136	31	3.3	672	62	1453	76	1333	730	10.7	60.0	0.7702
1094	24	2.3	609	45	1579	60	1276	620	10.7	59.7	0.7648
1010	19	1.7	561	-200	1705	-200	1235	501	10.3	60.2	0.7517
1011	14	1.3	527	21	1818	34	1197	445	10.1	60.5	0.7465
1066	8	1.1	512	16	1918	28	1182	422	11.0	56.2	0.7366
1052	16	1.6	553	34	1738	48	1221	472	10.5	58.1	0.7353
1144	29	3.2	667	98	1490	82	1339	730	10.2	59.6	0.7417
1333	64	8.0	900	174	1136	112	1517	1102	10.8	57.4	0.7408
1351	87	9.5	960	129	1079	101	1583	1028	10.5	60.6	0.7691

1.1) อ่านค่าจากไฟล์ข้อมูล .txt

- แยก 90% ของข้อมูลเพื่อใช้สำหรับ Train
- แยก 10% ของข้อมูลเพื่อใช้สำหรับ Test
- แยกเป็น 2 fold

```
def data_read(file_path):
    data = []
    with open(file_path, 'r') as file:
        lines = file.readlines()
        for line in lines:
            line = line.strip().split(',')
            data.append(line)
    return data

source_data = 'airQ.txt'
data = data_read(source_data)
data = np.array([line[0].split('\t') for line in data])
data = data.astype(float)
#data = data // 2000
#print(data[0][1])
k=0.1
output_data = data[:, 2]
input_data = np.delete(data,(1,2,4,6), axis=1)
fold_bes_mae = []
fold_test_mae =[]
```

```
for fold in range(1,3):
    print(fold)
    point1 = (input_data.shape[0])*k
    point2 = (output_data.shape[0])*k
    point1 = int(point1)
    point2 = int(point2)
    input_data_train = input_data[point1:,:]
    input_data_test = input_data[:point1,:]
    output_data_train =output_data[point2:]
    output_data_test = output_data[:point2]

    if fold == 2:
        input_data_train = input_data[:((input_data.shape[0])-point1),:]
        input_data_test = input_data[((input_data.shape[0])-point1):,:]
        output_data_train =output_data[:((output_data.shape[0])-point1)]
        output_data_test = output_data[((output_data.shape[0])-point1):]

    ##-----------------------------------------------------
```

1.2) สร้าง Function หา MAE จากโมเดล MLP

- ใช้ Particles เป็น Initialize weights
- Sigmoid Activation Function

```python
##-----------------------------------------------------------
def find_mae(weights,input_data_train, output_data_train):

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

    weights_input_hidden = weights[:input_nodes * hidden_nodes].reshape((input_nodes, hidden_nodes))
    weights_hidden_output = weights[input_nodes * hidden_nodes:].reshape((hidden_nodes, output_nodes))

    # Implement the forward pass of the MLP
    hidden_layer_input = np.dot(input_data_train, weights_input_hidden)
    hidden_layer_output = activation(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
    output_layer_output = output_layer_input  # No activation function for output

    # Calculate MAE on the validation data
    absolute_errors = np.abs(output_layer_output - output_data_train)
    mae = np.mean(absolute_errors)

    return mae
```

1.3) สร้าง Function PSO เพื่อหาคำตอบ

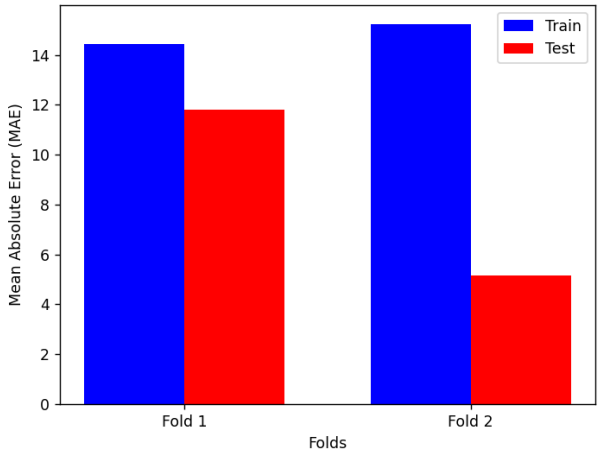
- สุ่ม initialize particles
- Update positions
- Find best positions

```python
def optimize_mlp(input_data_train, output_data_train):
    c1 = 1.5
    c2 = 1.5
    w = 0.7
    def initialize_particles(num_particles, num_dimensions):
        return np.random.uniform(-1, 1, (num_particles, num_dimensions))

    def update_particles(particles, velocities, best_positions, global_best_position, c1, c2, w):
        for i in range(particles.shape[0]):  # ลูปตามจำนวนพาทิเคิล
            # Generate random values for r1 and r2
            r1 = np.random.rand()
            r2 = np.random.rand()

            # Check if global_best_position is not None
            if global_best_position is not None:
                # Update velocity using the PSO equation
                cognitive_velocity = c1 * r1 * (best_positions[i] - particles[i])
                social_velocity = c2 * r2 * (global_best_position - particles[i])
                velocities[i] = w * velocities[i] + cognitive_velocity + social_velocity

                # Update particle position using the updated velocity
                particles[i] = particles[i] + velocities[i]

                # Apply bounds to particle position
                particles[i] = np.maximum(particles[i], lb)
                particles[i] = np.minimum(particles[i], ub)

    lb = np.full(num_dimensions, -1)  # Lower bound for weights
    ub = np.full(num_dimensions, 1)   # Upper bound for weights

    particles = initialize_particles(num_particles, num_dimensions)
```

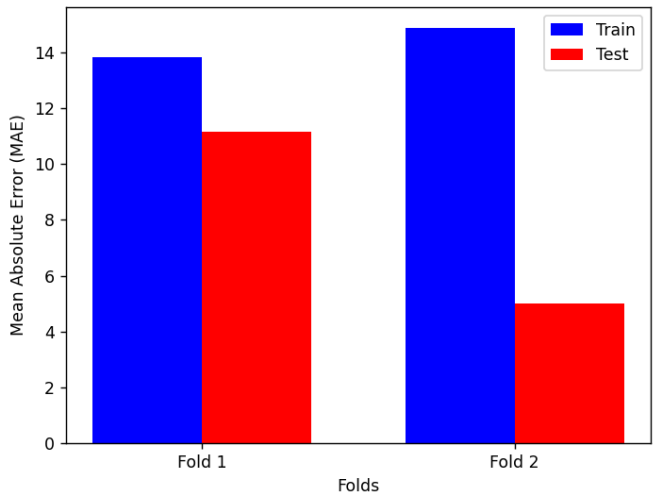
```
    ub = np.full(num_dimensions, 1)   # Upper bound for weights

    particles = initialize_particles(num_particles, num_dimensions)
    velocities = np.zeros((num_particles, num_dimensions))
    best_positions = particles.copy()
    best_mae = np.full(num_particles, float('inf'))
    global_best_position = None
    global_best_mae = float('inf')

    for iteration in range(num_iterations):
        update_particles(particles, velocities, best_positions, global_best_position, c1, c2, w)
        for i in range(num_particles):
            mae = find_mae(particles[i],input_data_train, output_data_train)
            if mae < best_mae[i]:
                best_mae[i] = mae
                best_positions[i] = particles[i]
            if mae < global_best_mae:
                global_best_mae = mae
                global_best_position = particles[i]

    return global_best_mae, global_best_position


best_mae, best_weights = optimize_mlp(input_data_train, output_data_train)
#print("Best Mean Absolute Error:", best_mae)
#print("Best Position:", best_weights)
test_case_mae = find_mae(best_weights,input_data_test,output_data_test)
#print("Test Case Mae = ",test_case_mae)
fold_bes_mae.append(best_mae)
fold_test_mae.append(test_case_mae)
```

2.ทำการทดลอง

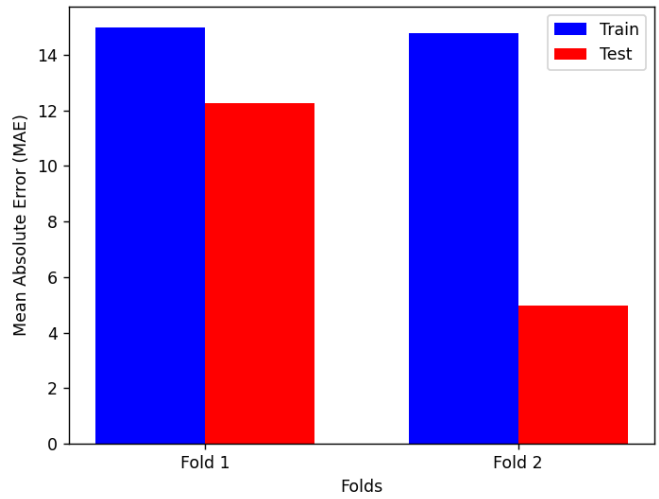
โดยการทดลองจะแสดงเป็นกราฟโดยสังเกตผลกระทบที่เกิดจากการเปลี่ยนแปลงค่าของโมเดล



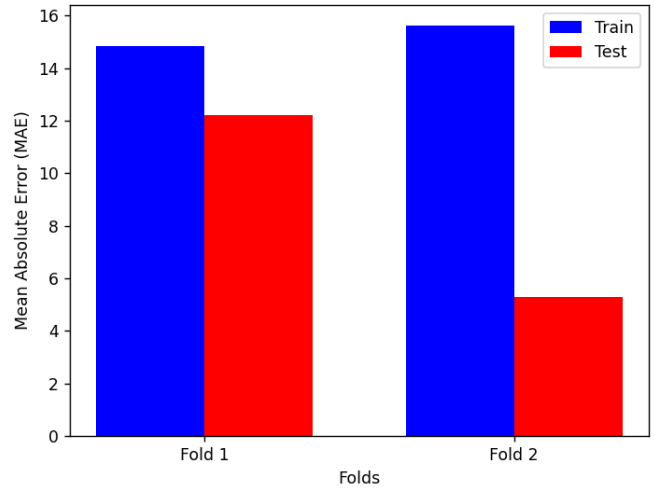

```
##-------------------------------------
input_nodes = input_data.shape[1]
hidden_nodes = 8
output_nodes = 1
num_particles = 10
num_dimensions = (input_nodes * hidden_
num_iterations = 10
##-------------------------------------
```




```
##---------------------------------------
input_nodes = input_data.shape[1]
hidden_nodes = 12
output_nodes = 1
num_particles = 15
num_dimensions = (input_nodes * hidden_no
num_iterations = 10
##---------------------------------------
```




```
##---------------------------------------
input_nodes = input_data.shape[1]
hidden_nodes = 8
output_nodes = 1
num_particles = 10
num_dimensions = (input_nodes * hidden_nodes
num_iterations = 20
##---------------------------------------
```




```
##---------------------------------------
input_nodes = input_data.shape[1]
hidden_nodes = 5
output_nodes = 1
num_particles = 20
num_dimensions = (input_nodes * hidden_nodes
num_iterations = 25
##---------------------------------------
```

3.วิเคราะห์และสรุปผล

จากการทดลองและเรียนรู้การใช้ PSO ในการ Train MLP แล้วพบว่า การใช้ PSO เป็นเครื่องมือในการค้นหาค่าน้ำหนักที่ดีที่สุดสำหรับ MLP ทำให้เราสามารถค้นพบค่าน้ำหนักที่ให้ผลลัพธ์ที่ดีที่สุดต่อ MAE ในการทำนายค่า Benzene concentration ได้ การแบ่งข้อมูลเป็น folds และทำการทดสอบแบบ Cross Validation ช่วยในการประเมินประสิทธิภาพของโมเดลที่สร้างให้ดีขึ้น โดยลดความเสี่ยงของการ overfitting หรือ underfitting ของโมเดล การปรับค่าของ c1, c2, และ w ใน PSO และ hidden layers, hidden nodes, จำนวน iterations ใน MLP จะช่วยในการเพิ่มประสิทธิภาพของการทำนายข้อมูล

แต่โดยรวม PSO เป็นสิ่งที่ค่อนข้างซับซ้อนเนื่องจากต้องตรวจสอบค่ามากมาย เช่น particles, velocities, best_positions, และ global_best_position และข้อที่เห็นได้ชัดที่สุดสำหรับ PSO คือการใช้ทรัพยากรณ์ในการประมวณผลค่อนข้างมากจึงทำให้ใช้เวลานานกว่าที่จะได้ผลลัพธ์ออกมา

ภาคผนวก

psonn > _

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix


def data_read(file_path):
    data = []
    with open(file_path, 'r') as file:
        lines = file.readlines()
        for line in lines:
            line = line.strip().split(',')
            data.append(line)
    return data

source_data = 'airQ.txt'
data = data_read(source_data)
data = np.array([line[0].split('\t') for line in data])
data = data.astype(float)
#data = data // 2000
#print(data[0][1])
k=0.1
output_data = data[:, 2]
input_data = np.delete(data,(1,2,4,6), axis=1)
fold_bes_mae = []
fold_test_mae =[]

for fold in range(1,3):
    print(fold)
    point1 = (input_data.shape[0])*k
    point2 = (output_data.shape[0])*k
    point1 = int(point1)
    point2 = int(point2)
    input_data_train = input_data[point1:,:]
    input_data_test = input_data[:point1,:]
    output_data_train =output_data[point2:]
    output_data_test = output_data[:point2]

    if fold == 2:
        input_data_train = input_data[:((input_data.shape[0])-point1),:]
        input_data_test = input_data[((input_data.shape[0])-point1):,:]
        output_data_train =output_data[:((output_data.shape[0])-point1)]
        output_data_test = output_data[((output_data.shape[0])-point1):]

    ##-------------------------------------------------------
    input_nodes = input_data.shape[1]
    hidden_nodes = 5
    output_nodes = 1
    num_particles = 20
    num_dimensions = (input_nodes * hidden_nodes) + (hidden_nodes * output_nodes)
    num_iterations = 25
    ##-------------------------------------------------------
```

psonn > optimize_mlp

```python
    num_dimensions = (input_nodes * hidden_nodes) + (hidden_nodes * output_nodes)
    num_iterations = 25
    ##------------------------------------------------------------
    def find_mae(weights,input_data_train, output_data_train):

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

        weights_input_hidden = weights[:input_nodes * hidden_nodes].reshape((input_nodes, hidden_nodes))
        weights_hidden_output = weights[input_nodes * hidden_nodes:].reshape((hidden_nodes, output_nodes))

        # Implement the forward pass of the MLP
        hidden_layer_input = np.dot(input_data_train, weights_input_hidden)
        hidden_layer_output = activation(hidden_layer_input)
        output_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
        output_layer_output = output_layer_input  # No activation function for output

        # Calculate MAE on the validation data
        absolute_errors = np.abs(output_layer_output - output_data_train)
        mae = np.mean(absolute_errors)

        return mae

    # Define a function for PSO optimization
    def optimize_mlp(input_data_train, output_data_train):
        c1 = 1.5
        c2 = 1.5
        w = 0.7
        def initialize_particles(num_particles, num_dimensions):
            return np.random.uniform(-1, 1, (num_particles, num_dimensions))

        def update_particles(particles, velocities, best_positions, global_best_position, c1, c2, w):
            for i in range(particles.shape[0]):  # ลูปตามจำนวนพาทิเคิล
                # Generate random values for r1 and r2
                r1 = np.random.rand()
                r2 = np.random.rand()

                # Check if global_best_position is not None
                if global_best_position is not None:
                    # Update velocity using the PSO equation
                    cognitive_velocity = c1 * r1 * (best_positions[i] - particles[i])
                    social_velocity = c2 * r2 * (global_best_position - particles[i])
                    velocities[i] = w * velocities[i] + cognitive_velocity + social_velocity

                    # Update particle position using the updated velocity
                    particles[i] = particles[i] + velocities[i]

                    # Apply bounds to particle position
                    particles[i] = np.maximum(particles[i], lb)
                    particles[i] = np.minimum(particles[i], ub)

        lb = np.full(num_dimensions, -1)  # Lower bound for weights
        ub = np.full(num_dimensions, 1)   # Upper bound for weights

        particles = initialize_particles(num_particles, num_dimensions)
        velocities = np.zeros((num_particles, num_dimensions))
        best_positions = particles.copy()
        best_mae = np.full(num_particles, float('inf'))
```

```
                    particles[i] = np.maximum(particles[i], lb)
                    particles[i] = np.minimum(particles[i], ub)

        lb = np.full(num_dimensions, -1)  # Lower bound for weights
        ub = np.full(num_dimensions, 1)   # Upper bound for weights

        particles = initialize_particles(num_particles, num_dimensions)
        velocities = np.zeros((num_particles, num_dimensions))
        best_positions = particles.copy()
        best_mae = np.full(num_particles, float('inf'))
        global_best_position = None
        global_best_mae = float('inf')

        for iteration in range(num_iterations):
            update_particles(particles, velocities, best_positions, global_best_position, c1, c2, w)
            for i in range(num_particles):
                mae = find_mae(particles[i],input_data_train, output_data_train)
                if mae < best_mae[i]:
                    best_mae[i] = mae
                    best_positions[i] = particles[i]
                if mae < global_best_mae:
                    global_best_mae = mae
                    global_best_position = particles[i]

        return global_best_mae, global_best_position


    best_mae, best_weights = optimize_mlp(input_data_train, output_data_train)
    #print("Best Mean Absolute Error:", best_mae)
    #print("Best Position:", best_weights)
    test_case_mae = find_mae(best_weights,input_data_test,output_data_test)
    #print("Test Case Mae = ",test_case_mae)
    fold_bes_mae.append(best_mae)
    fold_test_mae.append(test_case_mae)

print("best mae 2 fold : ",fold_bes_mae)
print("test mae 2 fold : ",fold_test_mae)


fold_names = ['Fold 1', 'Fold 2']
num_folds = len(fold_names)
bar_width = 0.35
index = range(num_folds)
plt.bar(index, fold_bes_mae, bar_width, label='Train', color='b')
plt.bar([i + bar_width for i in index], fold_test_mae, bar_width, label='Test', color='r')
plt.xlabel('Folds')
plt.xticks([i + bar_width / 2 for i in index], fold_names)
plt.ylabel('Mean Absolute Error (MAE)')
plt.legend(loc='upper right')
plt.show()
```