รายงาน

เรื่อง

Program สำหรับการ Train Multilayer Perceptron โดยใช้ Particle Swarm Optimization (PSO)

สำหรับการทำ prediction Benzene concentration โดยเป็นการ predict 5 วันล่วงหน้า และ 10 วันล่วงหน้า

โดย

น.ส. ภัทรจาริน ผดุงกิจเจริญ รหัสนักศึกษา 650610851

เสนอ

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา CPE 261456

Introduction to Computational Intelligence

สาขาวิชาวิศวกรรมหุ่นยนต์และปัญญาประดิษฐ์

ภาคเรียนที่ 1 ปีการศึกษา 2567

มหาวิทยาลัยเชียงใหม่

1.ลักษณะการทำงานของระบบ

1.1 วัตถุประสงค์

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

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

ในการวัด Error ให้ใช้ Mean Absolute Error (MAE)

1.2 ขั้นตอนการทำงาน

1.2.1 การโหลดและการจัดเตรียมข้อมูล

ข้อมูลในโค้ดได้ถูกโหลดจากไฟล์ Excel (AirQualityUCI.xlsx) โดยเลือกเฉพาะคอลัมน์ที่เกี่ยวข้องกับการทำนายค่าความเข้มข้นของเบนซีน (Benzene) ได้แก่:

Input: PT08.S1(CO), PT08.S2(NMHC), PT08.S3(NOx), PT08.S4(NO2), PT08.S5(O3), T, RH, AH

Output: C6H6(GT)

หลังจากเลือกคอลัมน์ที่ต้องการ จะทำการลบแถวที่มีค่าหายไป (NaN) จากนั้นกำหนดตัวแปร X สำหรับค่า input และตัวแปร y_5days สำหรับการทำนายค่าล่วงหน้า 5 วัน โดยการเลื่อนค่าของ y ลง 5 แถว และลบแถวสุดท้าย 5 แถวออก เพราะไม่มีข้อมูลการทำนาย

1.2.2 การออกแบบ Multilayer Perceptron (MLP)

โมเดล MLP ถูกสร้างขึ้นโดยมีโครงสร้างดังนี้:

จำนวนชั้นซ่อน (hidden layers): 1 ชั้น # กำหนดโครงสร้างของ hidden layers และจำนวนโหนดตามต้องการ

จำนวนโหนด (nodes): 5 โหนดในชั้นซ่อน # กำหนดโครงสร้างของ hidden layers และจำนวนโหนดตามต้องการ

ฟังก์ชัน MLP_forward ทำหน้าที่คำนวณผลลัพธ์ของโมเดลผ่านกระบวนการ Forward Propagation โดยใช้ tanh เป็นฟังก์ชัน Activation

1.2.3 การใช้ Particle Swarm Optimization (PSO)

ฟังก์ชัน PSO_optimize ทำหน้าที่ปรับค่า weights ของ MLP โดยใช้ PSO ซึ่งมีการกำหนดให้เริ่มต้นด้วยจำนวนอนุภาค (particles) 10 อนุภาค และจำนวนรอบการทำซ้ำสูงสุด (max_iter) 20 รอบ PSO นี้ทำการอัปเดตตำแหน่ง (position) และความเร็ว (velocity) ของแต่ละอนุภาคในแต่ละรอบการทำซ้ำเพื่อหาค่า weights ที่ดีที่สุดตามค่า Mean Absolute Error (MAE)

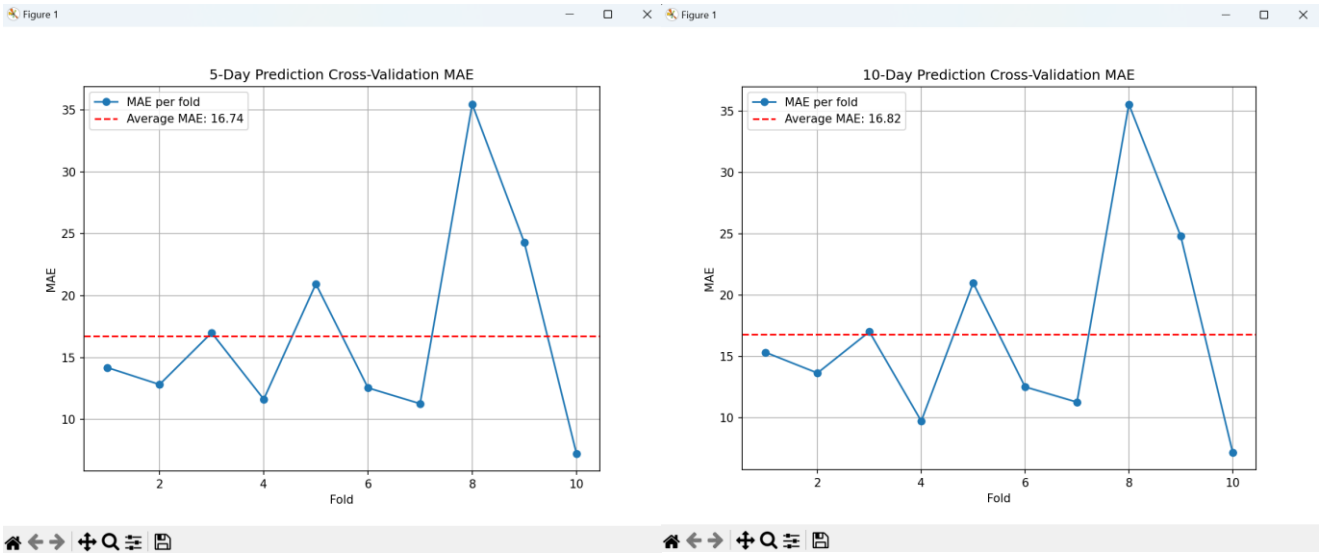
1.2.4 การใช้ Cross-Validation

ใช้ Cross-Validation แบบ 10% เพื่อตรวจสอบความสามารถของโมเดล โดยแบ่งข้อมูลเป็น 10 fold และประเมินค่า MAE ในแต่ละ fold จากนั้นทำการคำนวณค่า MAE เฉลี่ยสำหรับการพยากรณ์ล่วงหน้า 5 วันและ 10 วัน

2. Simulation ของระบบ ผลการทดลอง และวิเคราะห์

2.1 การตั้งค่าการทดลอง และผลการทดลอง

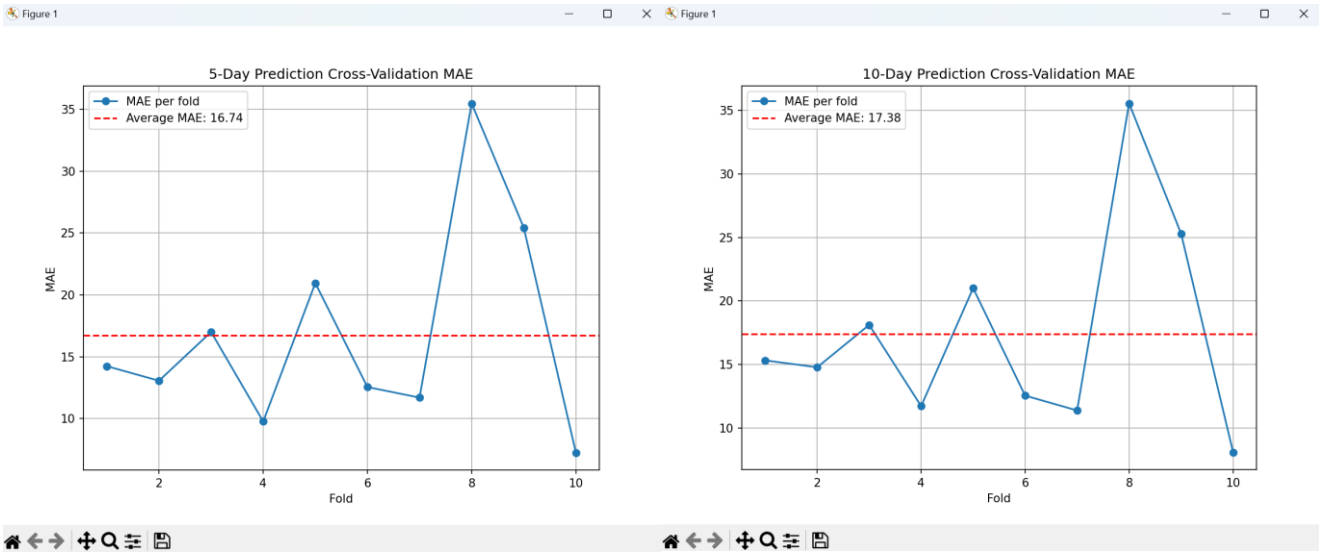
layers = [8, 5, 1] : 1 hidden layer และ 5 nodes



Mean Error for 5-day prediction: 16.743041842589946

Mean Error for 10-day prediction: 16.822933193691387

layers = [8, 20, 10, 5, 1] : 3 hidden layers และมีจำนวน nodes ที่แตกต่างกันในแต่ละชั้น



Mean Error for 5-day prediction: 16.740647211983855

Mean Error for 10-day prediction: 17.37856994772832

จากการทดลองจะเห็นได้ว่าค่า MAE เฉลี่ยที่ได้มีความแตกต่างกันตามโครงสร้างของโมเดล โดยโครงสร้างที่มีจำนวน hidden layers มากกว่าอาจมีผลลัพธ์ที่ดีกว่าในบางกรณี แต่ก็มีความเสี่ยงที่จะเกิด overfitting ได้เช่นกัน

2.2 การวิเคราะห์ผลการทดลอง

ค่า MAE เฉลี่ยที่ลดลงอาจแสดงถึงประสิทธิภาพที่สูงขึ้นของโมเดลในการทำนายค่าความเข้มข้นของเบนซีน แต่ควรพิจารณาด้วยว่าโครงสร้างโมเดลที่ซับซ้อนมากขึ้นจะใช้เวลาคำนวณมากขึ้นเช่นกัน ดังนั้น การเลือกโครงสร้างของ hidden layers ควรพิจารณาจากความสมดุลระหว่างความแม่นยำและความเร็วในการคำนวณ

สำหรับการทำนายล่วงหน้า 10 วัน ค่า MAE เฉลี่ยที่เพิ่มขึ้นอาจแสดงถึงความท้าทายที่เพิ่มขึ้นในการทำนายค่าล่วงหน้าไกลขึ้น เนื่องจากมีความไม่แน่นอนในข้อมูลที่สูงขึ้น

2.3 สรุปผลการทดลอง

โมเดล Multilayer Perceptron (MLP) ที่ได้รับการปรับค่า weights ผ่าน Particle Swarm Optimization (PSO) นั้นสามารถใช้ทำนายค่าความเข้มข้นของเบนซีนล่วงหน้าได้อย่างมีประสิทธิภาพในระดับหนึ่ง โดยโครงสร้างที่เหมาะสมของโมเดลจะขึ้นอยู่กับช่วงเวลาที่ต้องการทำนาย (5 วันหรือ 10 วัน)

3.โปรแกรม

GITHUB : https://github.com/Pattharajrin/261456_CI_Assignment4_Y3-1.git

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# โหลดชุดข้อมูลจากไฟล์ Excel
file_path = 'AirQualityUCI.xlsx'
data = pd.read_excel(file_path)

# เลือกเฉพาะคอลัมน์ที่เกี่ยวข้อง (input: 3,6,8,10,11,12,13,14; output: 5)
selected_columns = ['PT08.S1(CO)', 'PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH', 'C6H6(GT)']
df_selected = data[selected_columns]

# จัดการค่าที่ขาดหายไป (NaNs) โดยการลบแถวที่มีค่าว่าง
df_selected = df_selected.dropna()

# กำหนดตัวแปร input (X) และ output (y) สำหรับการทำนายล่วงหน้า 5 วัน
X = df_selected[['PT08.S1(CO)', 'PT08.S2(NMHC)', 'PT08.S3(NOx)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH']].values
y_5days = df_selected['C6H6(GT)'].values

# เลื่อนค่าของ output สำหรับการทำนายล่วงหน้า 5 วัน
y_5days = np.roll(y_5days, -5)

# ลบแถวสุดท้าย 5 แถวออก เนื่องจากไม่มีข้อมูลทำนาย
X = X[:-5]
y_5days = y_5days[:-5]

# กำหนดตัวแปร input (X) และ output (y) สำหรับการทำนายล่วงหน้า 10 วัน
y_10days = np.roll(y_5days, -5)

# ลบแถวสุดท้ายอีก 5 แถวสำหรับการทำนายล่วงหน้า 10 วัน
y_10days = y_10days[:-5]
X_10days = X[:-5]

# ฟังก์ชันสำหรับการคำนวณผลลัพธ์ใน MLP ผ่านกระบวนการ Forward Propagation
def MLP_forward(X, weights, layers):
    input_layer = X
    for layer_weights in weights:
        input_layer = np.dot(input_layer, layer_weights)  # คำนวณผลลัพธ์ของแต่ละ layer
        input_layer = np.tanh(input_layer)  # ใช้ tanh เป็นฟังก์ชัน Activation
    return input_layer
```

```python
# ฟังก์ชัน Particle Swarm Optimization (PSO) สำหรับปรับค่า weights ของโมเดล
def PSO_optimize(X_train, y_train, layers, num_particles=10, max_iter=20):
    num_inputs = X_train.shape[1]
    swarm = []

    # เริ่มต้นด้วยการสุ่ม weights สำหรับแต่ละ particle
    for _ in range(num_particles):
        particle = {
            'position': [np.random.randn(num_inputs, layers[0])] + \
                        [np.random.randn(layers[i], layers[i + 1]) for i in range(len(layers) - 1)],
            'velocity': [np.random.randn(num_inputs, layers[0])] + \
                        [np.random.randn(layers[i], layers[i + 1]) for i in range(len(layers) - 1)],
            'best_position': None,  # ตำแหน่งที่ดีที่สุดที่ particle นี้เคยเจอ
            'best_error': float('inf'),  # ค่า error ที่น้อยที่สุดที่ particle นี้เคยเจอ
        }
        swarm.append(particle)

    # กำหนดตำแหน่งที่ดีที่สุด (global best position) สำหรับทุก particle
    global_best_position = None
    global_best_error = float('inf')

    # เริ่มลูปการทำงานของ PSO
    for iteration in range(max_iter):
        for particle in swarm:
            # คำนวณผลลัพธ์จาก MLP ด้วย weights ของ particle นี้
            predictions = MLP_forward(X_train, particle['position'], layers)
            error = mean_absolute_error(y_train, predictions)  # คำนวณ MAE

            # อัปเดตตำแหน่งที่ดีที่สุดของ particle ถ้าค่า error ดีกว่าเดิม
            if error < particle['best_error']:
                particle['best_error'] = error
                particle['best_position'] = particle['position']

            # อัปเดต global best position ถ้า error ของ particle นี้ดีกว่าค่า global best
            if error < global_best_error:
                global_best_error = error
                global_best_position = particle['position']

        # อัปเดต velocity และตำแหน่งของแต่ละ particle
        for particle in swarm:
            for i in range(len(particle['position'])):
                inertia = 0.5 * particle['velocity'][i]  # โมเมนตัมเพื่อรักษาทิศทางการเคลื่อนที่เดิม
                cognitive = 1.5 * np.random.rand() * (particle['best_position'][i] - particle['position'][i])  # การเรียนรู้จากตำแหน่งที่ดีที่สุดของตัวเอง
                social = 1.5 * np.random.rand() * (global_best_position[i] - particle['position'][i])  # การเรียนรู้จากตำแหน่งที่ดีที่สุดของกลุ่ม
                particle['velocity'][i] = inertia + cognitive + social  # อัปเดต velocity
                particle['position'][i] += particle['velocity'][i]  # อัปเดตตำแหน่ง

        print(f"Iteration {iteration + 1}/{max_iter}, Best Error: {global_best_error}")

    return global_best_position  # return ค่า weights ที่ดีที่สุดที่หาได้

# ฟังก์ชันคำนวณ Mean Absolute Error (MAE)
def mean_absolute_error(y_true, y_pred):
    return np.mean(np.abs(y_true - y_pred.flatten()))
```

```python
# ฟังก์ชัน Cross-Validation แบบ 10% และแสดงกราฟ MAE
def cross_validate(X, y, layers, num_folds=10, title="Cross-Validation MAE"):
    fold_size = len(X) // num_folds
    errors = []
    best_weights = None
    lowest_error = float('inf')

    for i in range(num_folds):
        start_test = i * fold_size
        end_test = start_test + fold_size

        # แบ่งชุดข้อมูลทดสอบและฝึกตาม fold ที่กำหนด
        X_test = X[start_test:end_test]
        y_test = y[start_test:end_test]
        X_train = np.concatenate((X[:start_test], X[end_test:]), axis=0)
        y_train = np.concatenate((y[:start_test], y[end_test:]), axis=0)

        # เรียกใช้งาน PSO เพื่อหา weights ที่ดีที่สุดสำหรับ fold นี้
        current_weights = PSO_optimize(X_train, y_train, layers)

        # คำนวณ error โดยใช้ weights ที่หาได้
        predictions = MLP_forward(X_test, current_weights, layers)
        error = mean_absolute_error(y_test, predictions)
        errors.append(error)

        # ตรวจสอบว่า weights ที่ได้มีค่า error ต่ำสุดหรือไม่
        if error < lowest_error:
            lowest_error = error
            best_weights = current_weights

        print(f"Fold {i + 1}/{num_folds}, MAE: {error}")

    mean_error = np.mean(errors)  # คำนวณค่า MAE เฉลี่ย

    # แสดงกราฟ MAE สำหรับแต่ละ fold
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, num_folds + 1), errors, marker='o', label='MAE per fold')
    plt.axhline(y=mean_error, color='r', linestyle='--', label=f'Average MAE: {mean_error:.2f}')
    plt.title(f'{title}')
    plt.xlabel('Fold')
    plt.ylabel('MAE')
    plt.legend()
    plt.grid()  # เพิ่มเส้นกริดให้กราฟ
    plt.show()

    return mean_error, best_weights  # return ค่า MAE เฉลี่ยและ weights ที่ดีที่สุด

# ใช้ฟังก์ชันตัวอย่าง: ฝึกโมเดลและประเมินผลสำหรับการทำนาย 5 วัน
layers = [8, 5, 1]  # กำหนดโครงสร้างของ hidden layers และจำนวนโหนดตามต้องการ
mean_error_5days, best_weights_5days = cross_validate(X, y_5days, layers, title="5-Day Prediction Cross-Validation MAE")

# ใช้ฟังก์ชันตัวอย่าง: ฝึกโมเดลและประเมินผลสำหรับการทำนาย 10 วัน
mean_error_10days, best_weights_10days = cross_validate(X_10days, y_10days, layers, title="10-Day Prediction Cross-Validation MAE")

# แสดงค่า MAE เฉลี่ยของการทำนาย 5 วัน และ 10 วัน
print(f"Mean Error for 5-day prediction: {mean_error_5days}")
print(f"Mean Error for 10-day prediction: {mean_error_10days}")
```