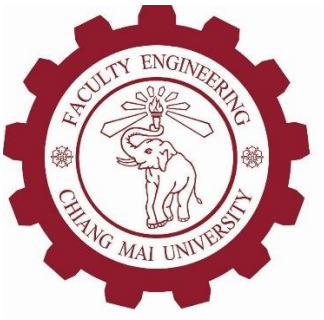


รายงานการทดลอง

Computer Assignment 3

261456 (Introduction to Computational Intelligence)

โดย

ปิยะนันท์ ปิยะวรรณ์โณ 650610845

เสนอ

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

ภาคเรียนที่ 1 ปีการศึกษา 2567

มหาวิทยาลัยเชียงใหม่

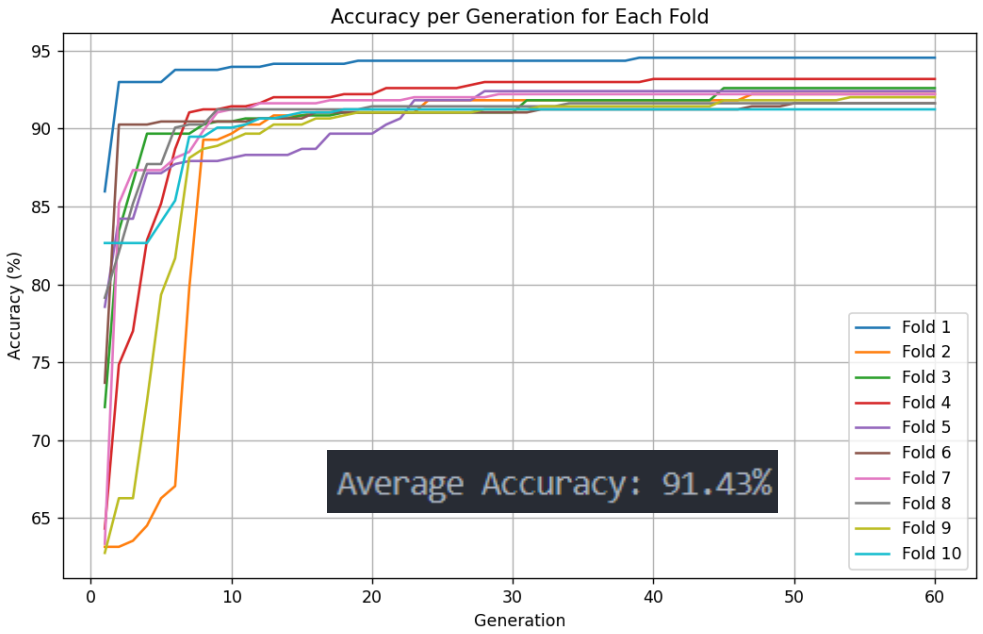
การทดลอง Train Multi Layer Perceptron โดยใช้ Genetic Algorithms ด้วย WDBC

วิธีการทำงานของโปรแกรม

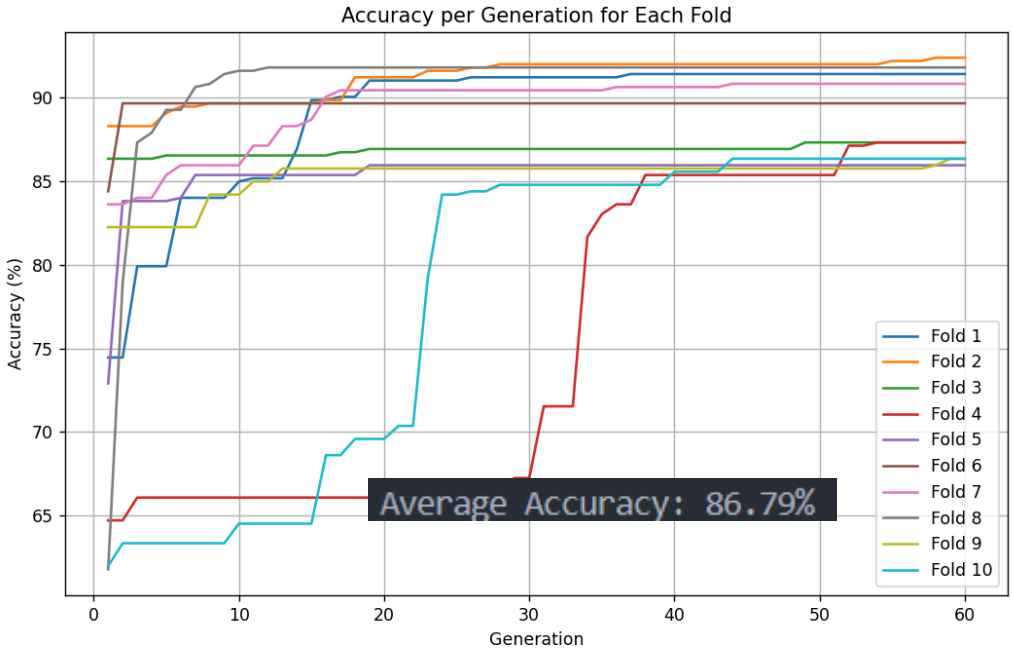
1. นำข้อมูลเข้าจากไฟล์ WDBC
2. ทำ 10 % Cross validation โดยการแบ่งข้อมูลเป็น 10 ชุด
3. สุ่ม weight ของข้อมูลแต่ละชุด
4. นำขอมูลแต่ละชุดไป Feed forward และหาค่า fitness ของ chromosome แต่ละตัว
5. เลือก chromosome ที่มีค่า fitness มากที่สุด 20% แรก
6. นำ 20% ที่เลือกไป generation ถัดไป
7. นำ 20% มา crossover กันให้มี chromosome ตามที่กำหนด
8. ทำซ้ำจนครบ 10 ชุด แล้วหาค่า Average Accuracy
9. Plot graph ของค่า accurate ของแต่ละ gens ของแต่ละ fold

ผลการทำงาน

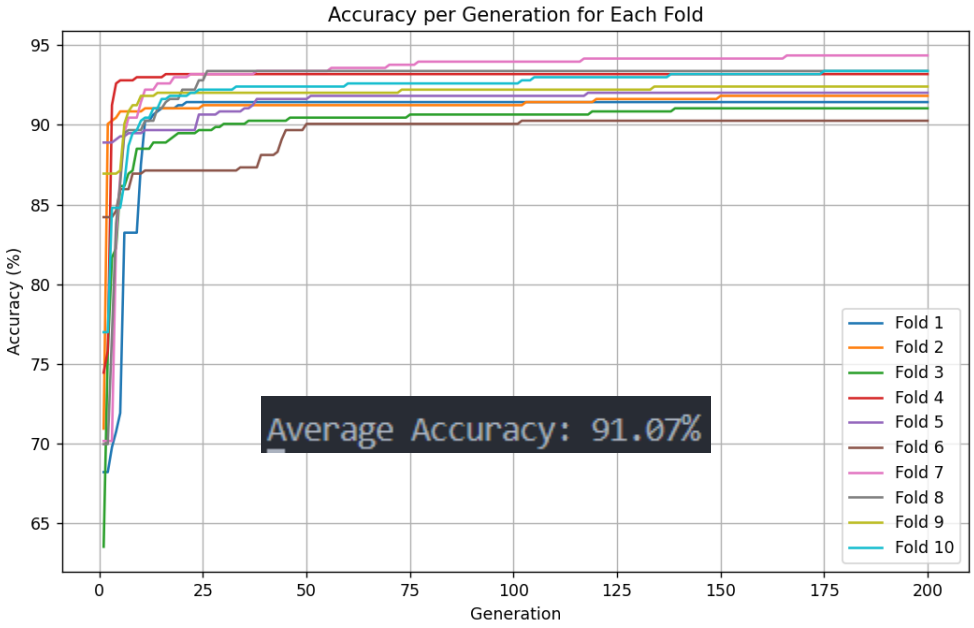
1. ใช้ Hidden layers = 10 , Population = 50 , Generation = 60 , Mutation rate = 0.1



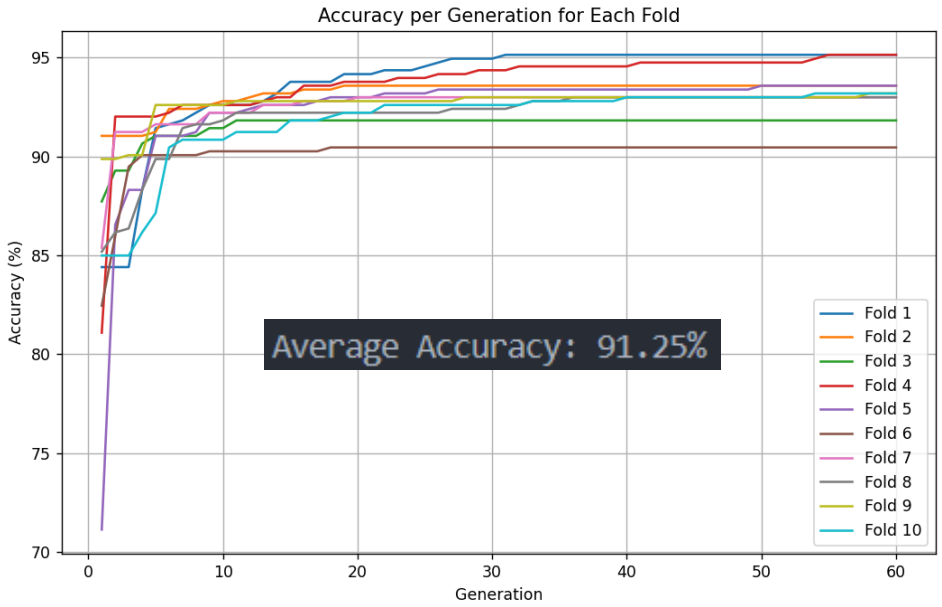
2. ใช้ Hidden layers = 10 , Population = 50 , Generation = 60 , Mutation rate = 0.005



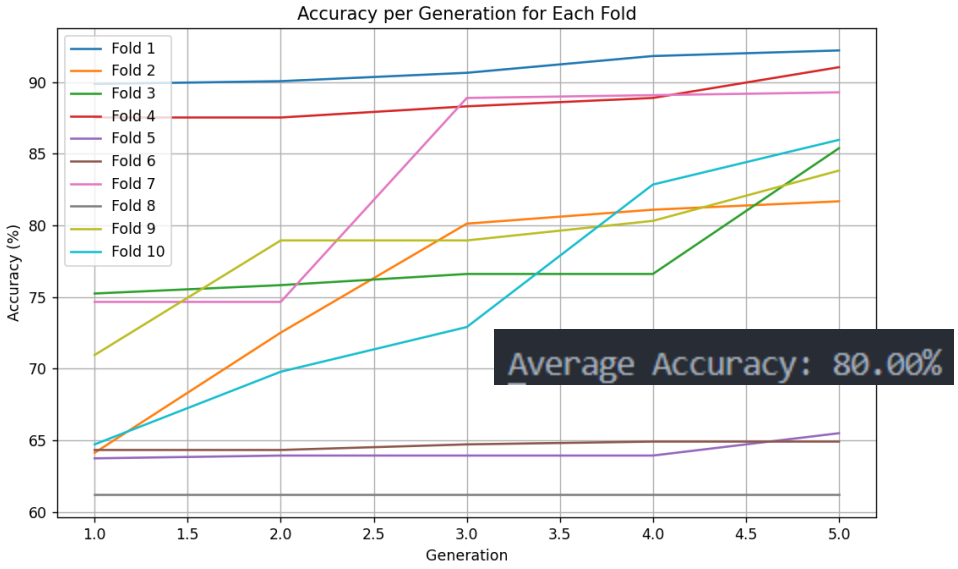
3. ใช้ Hidden layers = 10 , Population = 60 , Generation = 200 , Mutation rate = 0.1



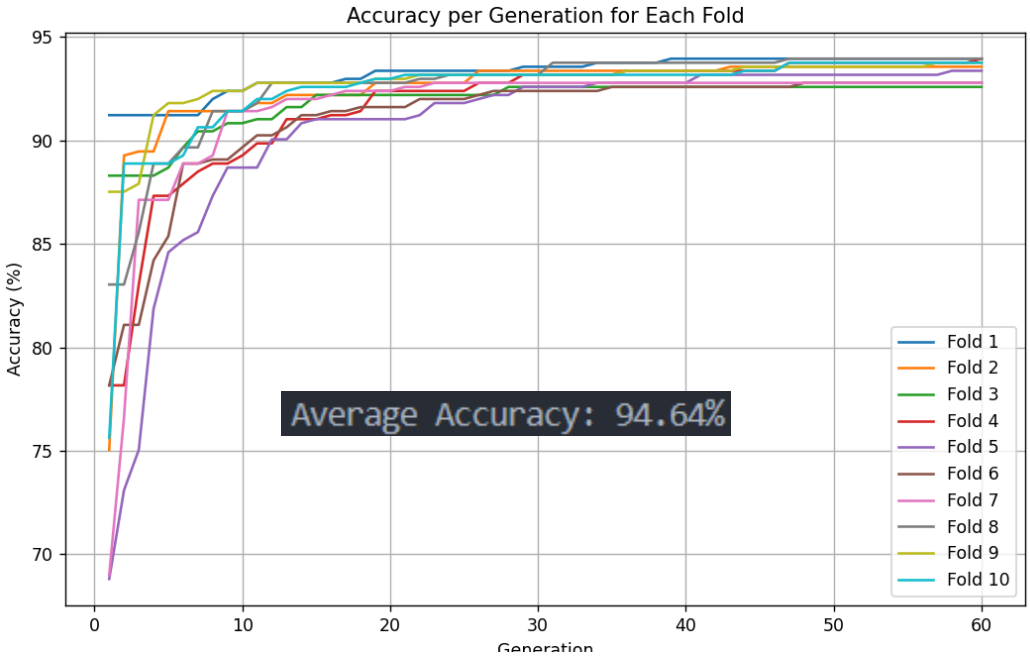
4. ใช้ Hidden layers = 10 , Population = 150 , Generation = 60 , Mutation rate = 0.1



5. ใช้ Hidden layers = 10 , Population = 15 , Generation = 5 , Mutation rate = 0.1



6. ใช้ Hidden layers = 10 , Population = 50 , Generation = 60 , Mutation rate = 0.1



สรุปผลการทำงาน

จากการทดลองทำ MLP โดยใช้ Genetic Algorithms จะเห็นว่าเมื่อ input ค่าที่เหมาะสม โปรแกรมจะมีค่า Accurate มากกว่า 90% แต่ถ้า Input ค่าใดค่าหนึ่งมากหรือน้อยเกินไป ค่า Accurate จะลดลง ทั้งนี้ไม่สามารถบอกได้ว่า input ค่าใดคือค่าที่จะทำให้ค่า Accurate มากที่สุด

โปรแกรม

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


def Load_data(file = 'wdbc.txt'):
    data_output = []
    data_input = []


    with open(file,'r') as file:
        for line in file :
            data = line.strip().split(',')

            if data[1] ==  'M':
                data_output.append(1)
            elif data[1] == "B":
                data_output.append(0)

            data_input.append(data[2:])

    return data_output , data_input

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

def Init_population(population_size,input_size,hidden_size,output_size):
    length = input_size * hidden_size + hidden_size * output_size

    return np.random.uniform(-1,1,(population_size,length))
```

```python
def Init_Weight(chromosome,input_size,hidden_size,output_size):
    input_weight = chromosome[:input_size*hidden_size].reshape((input_size,hidden_size))
    hidden_weight = chromosome[input_size*hidden_size:(input_size+hidden_size)*(hidden_size+output_size)].reshape((hidden_size,output_size))

    return input_weight , hidden_weight

def Feed_forward(inputs,input_weight,hidden_weigh):
    hidden_input = np.dot(inputs,input_weight)
    hidden_output = Sigmoid(hidden_input)

    final_input = np.dot(hidden_output,hidden_weigh)
    final_output = Sigmoid(final_input)

    return final_output
```

```python
def Fitness_function(inputs,indv,chromosome,input_size,hidden_size,output_size):
    input_weight , hidden_weigth = Init_Weight(chromosome,input_size,hidden_size,output_size)
    output = Feed_forward(inputs,input_weight,hidden_weigth)
    fitness = np.sum(np.argmax(output,axis=1) == np.argmax(indv,axis=1)) / len(indv)

    return fitness

def Crossing_over(P1,P2):
    Crossing_site = np.random.randint(len(P1))
    child = np.concatenate((P1[:Crossing_site],P2[Crossing_site:]))

    return child

def Mutate(chromosome,rate):
    P = np.random.rand(len(chromosome)) < rate
    chromosome[P] += np.random.uniform(-0.1,0.1,np.sum(P))

    return chromosome
```

```python
def Genetic_Algorithm(inputs, indv, input_size, hidden_size, output_size, population_size, generation, rate):
    population = Init_population(population_size, input_size, hidden_size, output_size)
    accuracy_per_generation = []

    for gen in range(generation):
        fitness_array = []

        for chromosome in population:
            fitness = Fitness_function(inputs, indv, chromosome, input_size, hidden_size, output_size)
            fitness_array.append(fitness)

        best_index = np.argmax(fitness_array)
        best_chromosome = population[best_index]
        best_fitness = fitness_array[best_index]

        # Record the best fitness (accuracy) for this generation
        accuracy_per_generation.append(best_fitness * 100)

        print(f"Generation {gen + 1}, Accuracy: {best_fitness * 100:.2f}%")

        # Select top 20% of population for crossover
        select_index = np.argsort(fitness_array)[-int(0.2 * population_size):]
        select_population = population[select_index]

        new_population = []
        for _ in range(population_size - len(select_population)):
            P1 = select_population[np.random.randint(len(select_population))]
            P2 = select_population[np.random.randint(len(select_population))]
            child = Mutate(Crossing_over(P1, P2), rate)
            new_population.append(child)

        population = np.vstack((select_population, new_population))

    return best_chromosome, accuracy_per_generation
```

```python
def Cross_Validaion(inputs, indv, input_size, hidden_size, output_size, population_size, generation, rate, k=10):
    fold_size = len(inputs) // k
    fitness_array = []
    best_hidden_size = []
    best_chromosomes = []
    all_accuracies = []  # Store accuracies for all folds

    for fold in range(k):
        start_index = fold * fold_size
        end_index = (fold + 1) * fold_size

        Validation_input = inputs[start_index:end_index]
        Validation_indv = indv[start_index:end_index]

        train_input = np.concatenate([inputs[:start_index], inputs[end_index:]])
        train_indv = np.concatenate([indv[:start_index], indv[end_index:]])

        best_chromosome, accuracies = Genetic_Algorithm(
            train_input, train_indv, input_size, hidden_size[fold], output_size,
            population_size, generation, rate
        )

        all_accuracies.append(accuracies)  # Collect accuracies for this fold
```

```python
        all_accuracies.append(accuracies)  # Collect accuracies for this fold

        fitness = Fitness_function(
            Validation_input, Validation_indv, best_chromosome,
            input_size, hidden_size[fold], output_size
        )
        fitness_array.append(fitness)
        best_hidden_size.append(hidden_size[fold])
        best_chromosomes.append(best_chromosome)

        print(f"Fold {fold + 1} Average Accuracy: {fitness * 100:.2f}%")

    mean_accuracy = np.mean(fitness_array)
    print(f"\nAverage Accuracy: {mean_accuracy * 100:.2f}%")

    # Plot accuracies per generation for each fold
    plt.figure(figsize=(10, 6))
    for fold_idx, accuracies in enumerate(all_accuracies):
        plt.plot(range(1, generation + 1), accuracies, label=f'Fold {fold_idx + 1}')
    plt.xlabel('Generation')
    plt.ylabel('Accuracy (%)')
    plt.title('Accuracy per Generation for Each Fold')
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
150  def hidden_size(x):
151      return [x,x,x,x,x,x,x,x,x,x]
152
153
154  data_output , data_input = Load_data()
155
156  data_input = np.array(data_input,dtype=float)
157  data_output = np.array(data_output)
158
159  indv = np.zeros((len(data_output),2))
160  indv[np.arange(len(data_output)),data_output] = 1
161
162  input_size = len(data_input[0])
163  hidden_size = hidden_size(100)
164  output_size = len(np.unique(data_output))
165  population_size = 50
166  generation = 60
167  Mutate_rate = 0.1
168
169  Cross_Validaion(data_input,indv,input_size,hidden_size,output_size,population_size,generation,Mutate_rate)
```

SPHSTR/Computer_Intelligence_Genetic_Algorithm