Train Multilayer Perceptron

with

Genetic Algorithms

จัดทำโดย

นายปัณณวิชญ์ พันธ์วงศ์

600610752

เสนอ

รศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของวิชา

CPE 261456 (Introduction to Computational Intelligence)

ภาคเรียนที่ 1 ปีการศึกษา 2563

มหาวิทยาลัยเชียงใหม่

สารบัญ

1. การเตรียมข้อมูล

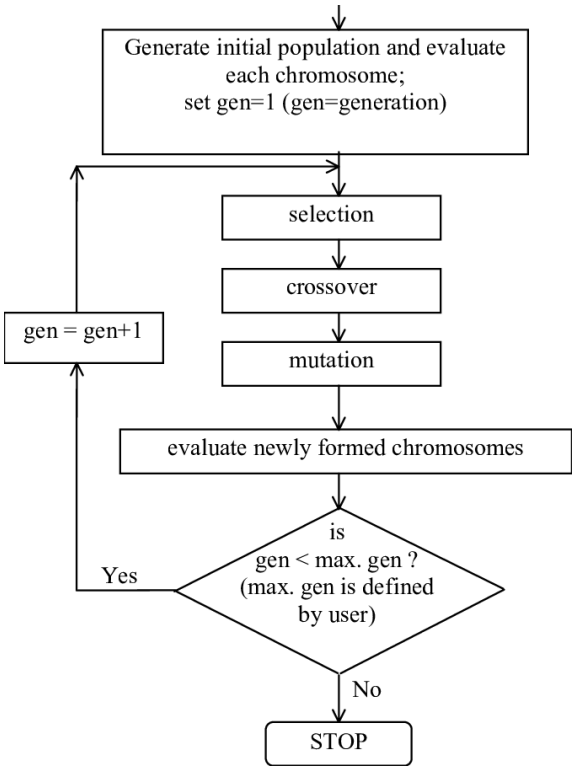
1.1 คำอธิบายข้อมูล

ข้อมูลไฟล์ wdbc.data เป็นข้อมูล Wisconsin Diagnostic Breast Cancer จาก UCI Machine learning Repository) โดยที่ data set นี้ มี 2 classes และ 30 features ซึ่งในแต่ละ sample จะมีทั้งหมด 32 ค่า

โดยผู้จัดทำได้ทำการเตรียมข้อมูลเนื่องจากข้อมูลอินพุตจะประกอบไปด้วย 30 features โดยผู้จัดทำได้ทำการ Min-max normalization ให้อยู่ในช่วง (0,1] และเนื่องจากข้อมูลแบ่งเป็น 2 classes คือ M และ B ซึ่งผู้จัดทำได้ทำการเอาต์พุตของระบบให้อยู่รูปแบบตัวเลขจำนวนเต็ม โดยให้ M มีค่าคือ 1 และ B มีค่าคือ 0

กระบวนการทำงานขั้นตอนวิธีเชิงพันธุกรรม

1.2 ภาพรวมของระบบ



รูปที่ 1 แผนผังการทำงานของระบบ

1.3 การทำงานของระบบ

สำหรับการทำงานของระบบนั้น แบ่งขั้นตอนการทำงานทั้งหมดเป็น 6 ขั้นตอน ดังนี้

1. กำหนดจำนวนประชากรของโครโมโซม(N)ในแต่ละเจนเนอเรชั่น(Generation) และจำนวนเจนเนอเรชั่นทั้งหมด(t_max)
2. ประเมิน(Evaluate) แต่ละโครโมโซม โดยจะจำโครโมโซมที่มีค่าความเหมาะสม (Fitness value) ที่ดีที่สุดไว้
3. ทำการคัดเลือก(Selection) โดยใช้วิธีการ การเลือกอันดับเชิงเส้น(Linear ranks selection) ซึ่งใช้ Stochastic universal sampling ในการเลือกโครโมโซมเมื่อเรียงอันดับแล้ว ซึ่งผลลัพธ์ที่ได้จากการคัดเลือกจะถูกส่งไปยัง บ่อผสมพันธุ์(Mating pool)
4. นำโครโมโซมจาก บ่อผสมพันธุ์(Mating pool) มาทำการครอสโอเวอร์(Crossover) โดยจำนวนครั้งในการครอสโอเวอร์ หรือจำนวนลูกที่จะได้นั้นจะอยู่ที่ 80-100% ของจำนวนประชากรทั้งหมดของโครโมโซมที่กำหนดใน ข้อ 1. ซึ่งหากจำนวนลูกที่ได้ไม่ครบตามจำนวนที่กำหนด จะทำการสุ่มเติมจำนวนลูกให้ครบด้วยโครโมโซมที่อยู่ใน บ่อผสมพันธุ์(Mating pool)
5. นำโครโมโซมลูกที่ได้จากข้อ 4. มาสุ่มการเกิดการกลายพันธุ์(Mutation) โดยทำการสุ่มการกลายพันธุ์ในแต่ละโครโมโซมโดยแต่ละ Neural node มีโอกาสที่จะเกิดการกลายพันธุ์(Mutation) อยู่ที่ 25% โดยสุ่มค่าให้อยู่ในช่วง ~U(-1,1) โดยหลักการการครอสโอเวอร์(Crossover) และการกลายพันธุ์(Mutation) นั้นจะอ้างอิงงานวิจัยของ David J. Montana and Lawrence Davis 1989
6. ผลลัพธ์โรมโมโซมที่เกิดการกลายพันธุ์จาก ข้อ 5. จะถูกตั้งค่าให้เป็นชุดของประชากรโครโมโซมในรุ่นถัดไป

2. การทดลอง

ในแต่ละการทดลองผู้จัดทำได้ทำการปรับเปลี่ยนจำนวน Hidden layers และ จำนวน Nodes โดยผู้จัดทำทำการทดลองทั้งหมด 4 โครงสร้าง แต่ละโครงสร้างได้ทำการสุ่ม weight หรือ gene อยู่ช่วงของ ~U(-2,2) ,จำนวนประชากรในแต่ละรุ่นให้มีค่าเท่ากับ 100 ,จำนวนรุ่นที่จะทำการฝึกสอนมีค่าเท่ากับ 1000 และ ฟังก์ชันวัตถุประสงค์(Objective function) ใช้ Mean square error(MSE) ในการหาค่าความเหมาะสม(Fitness value) ดังตารางที่ 1

ตารางที่ 1 แสดงพารามิเตอร์ในแต่ละโครงสร้าง

พารามิเตอร์	**จำนวน**
Neural weights (genes)	~U (-2,2)
Number of populations	100
Number of generations	1000
Objective function	$f(x) = \frac{1}{\log(1 + MSE)}$

โดยโครงสร้างของนิวรอลเน็ตเวิร์ค(Neural network) ในแต่ละการทำปรับเปรี่ยนแบบสุ่ม โดยวิธีการฝึกสอนได้ใช้ 10-folds cross-validation และได้สุ่มโครงสร้างดังตารางที่ 2

ตารางที่ 2 โครงสร้างนิวรอลเน็ตเวิร์คของแต่ละการทดลอง

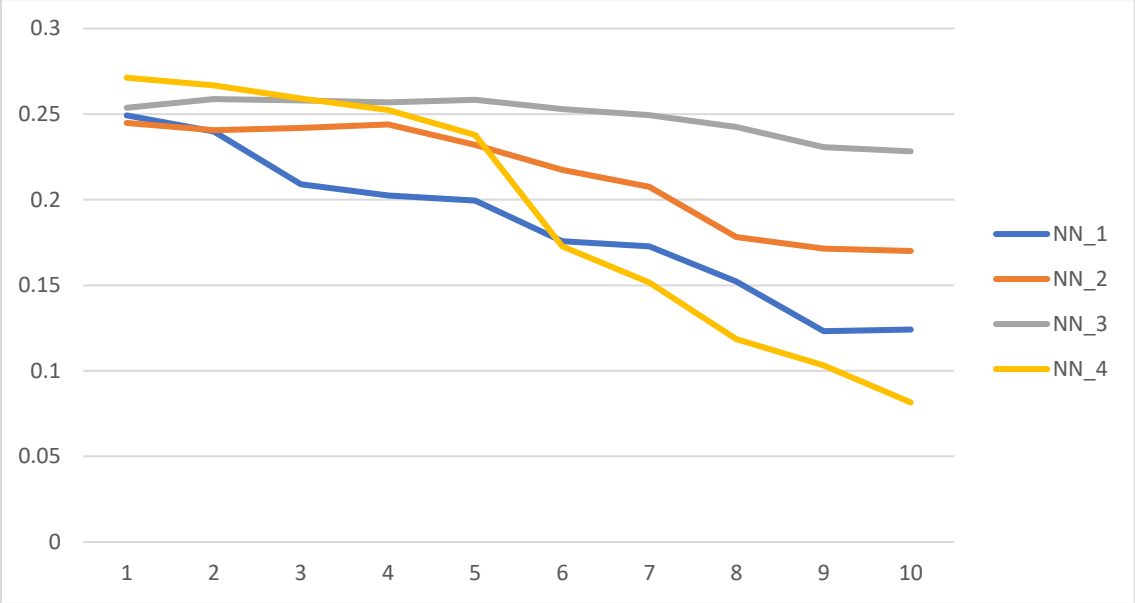
การทดลอง	**โครงสร้าง**
1	30-25-15-5-2-1
2	30-30-18-9-4-2-1
3	30-40-50-30-20-5-1
4	30-10-5-5-3-1

โดยผลการทดลองจะสรุปผลในรูปแบบดังนี้

1. กราฟที่การเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการฝึกสอนของแต่ละ folds
2. กราฟที่การเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการทดสอบของแต่ละ folds
3. กราฟการเปลี่ยนแปลงโครโมโซมที่ดีที่สุดของแต่ละ folds
4. แผนภูมิแท่งที่ที่จะแสดงผลลัพธ์เปอร์เซ็นต์ความถูกต้องโดยเฉลี่ยของทั้ง 10 folds

2.1 การเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการฝึกสอนของแต่ละ folds

กราฟจะแสดงแนวโน้มการเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการฝึกสอนในแต่ละ folds ซึ่งเป็นการหาค่าเฉลี่ยของทุกโครโมโซมในรุ่นนั้นๆ โดย 1 fold จะหมายถึงการฝึกสอน 100 รุ่น

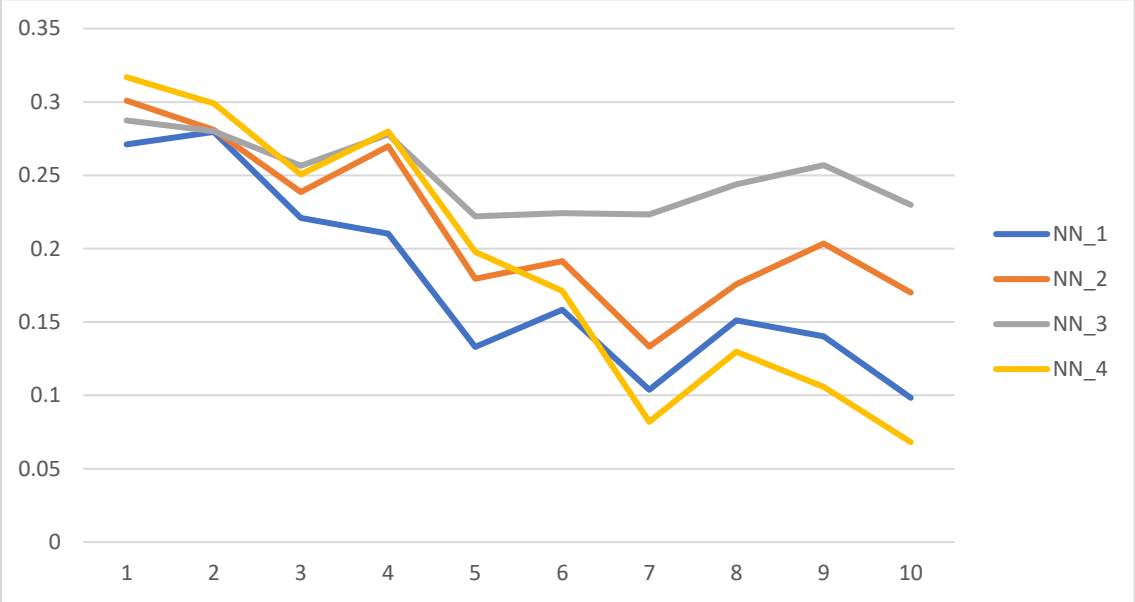


รูปที่ 2 กราฟแสดงแนวโน้มค่า MSE ของการฝึกสอน

จากกราฟจะเห็นได้ว่าค่าเฉลี่ยในการฝึกสอนของการดโครงสร้างที่ 1 และโครงสร้างที่ 4 แนวโน้มของการค่า MSE ที่ลดลงที่มากกว่าโครงสร้างอื่นๆ โดยในช่วงต้นๆ ของ folds การฝึกสอนของโครงสร้างที่ 4 มีแนวโน้มค่า MSE มากที่สุด แต่ตั้งแต่ fold ที่ 6 เป็นต้นไปพบว่าแนวโน้มค่า MSE ลดลงอย่างฉับพลันอย่างเห็นได้ชัด และจากโครงสร้างที่ 3 พบว่ามีผลลัพธ์ที่แย่ที่สุดในแต่ละโครงสร้าง

2.2 การเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการทดสอบของแต่ละ folds

กราฟจะแสดงแนวโน้มการเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการทดสอบในแต่ละ folds ซึ่งเป็นการหาค่าเฉลี่ยของทุกโครโมโซมในรุ่นนั้นๆ โดย 1 fold จะหมายถึงการฝึกสอน 100 รุ่น

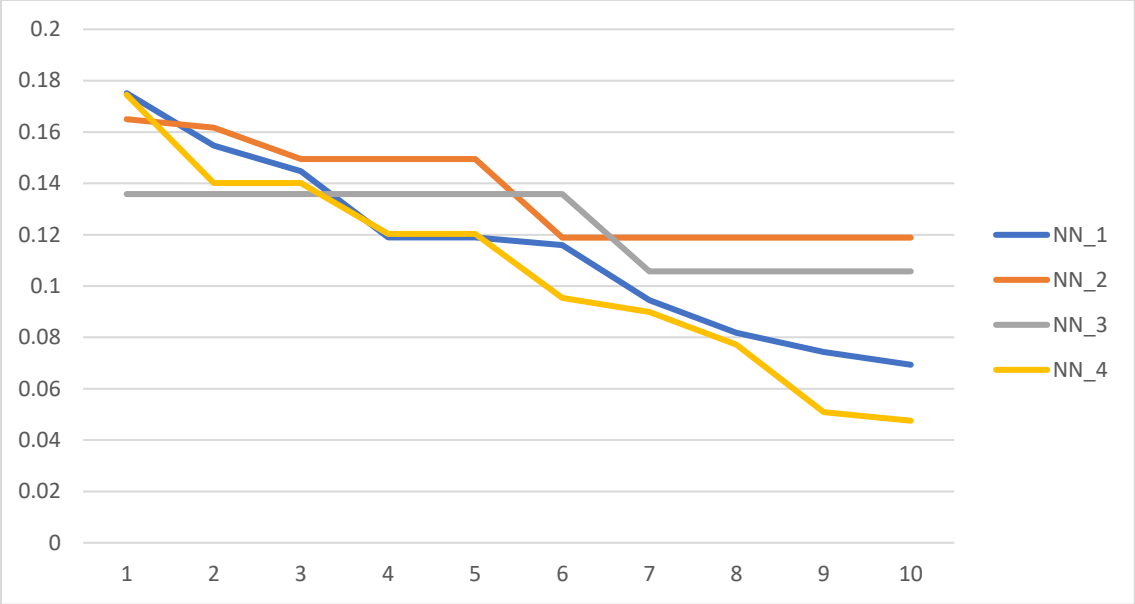


รูปที่ 3 กราฟแสดงแนวโน้มค่า MSE ของการฝึกสอน

จากกราฟจะเห็นได้ว่าค่าเฉลี่ยในการทดสอบของโครงสร้างที่ 1 และโครงสร้างที่ 4 แนวโน้มของการค่า MSE ที่ลดลงที่มากกว่าโครงสร้างอื่นๆ โดยในช่วงต้นๆ ของ folds การทดสอบของโครงสร้างที่ 4 มีแนวโน้มค่า MSE มากที่สุด แต่ตั้งแต่ fold ที่ 5 เป็นต้นไปพบว่าแนวโน้มค่า MSE ลดลงอย่างฉับพลันอย่างเห็นได้ชัด และจากโครงสร้างที่ 3 พบว่ามีผลลัพธ์ที่แย่ที่สุดในแต่ละโครงสร้าง

2.3 การเปลี่ยนแปลงโครโมโซมที่ดีที่สุดของแต่ละ folds

กราฟจะแสดงแนวโน้มการเปลี่ยนแปลงโครโมโซมที่ดีสุดในโครงสร้างนั้นๆ โดยแสดงเป็นแนวโน้มของแต่ละ fold ว่าโครโมโซมที่ดีที่สุดมีค่า MSE เป็นอย่างไร โดยใช้ชุดข้อมูลทดสอบเป็นข้อมูลในการหาค่าความเหมาะสม(Fitness value)

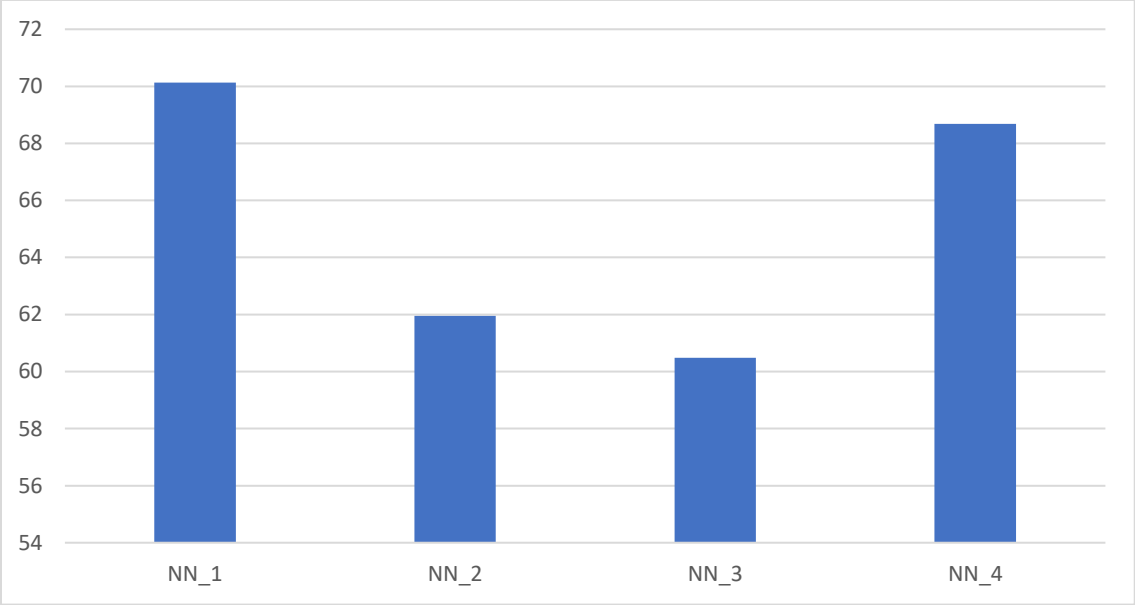


รูปที่ 4 แนวโน้มการเปลี่ยนแปลงโครโมโซมที่ดีที่สุดอ้างอิงค่า MSE

จากกราฟจะเห็นได้ว่าโครมที่ดีที่สุดของโครงสร้างที่ 4 มีแนวโน้มให้ค่า MSE ลดลงอย่างต่อเนื่องเป็นอันดับหนึ่ง และมีโครงสร้างที่ 3 มีแนวโน้มลดลงเป็นอันดับ 2 แต่ในทางกลับกันพบว่าโครงสร้างที่ 2 มีโครโมโซมที่ดีที่สุดมีค่า MSE มากที่สุดในแต่ละโครงสร้าง

2.4 เปอร์เซ็นต์ความถูกต้องโดยเฉลี่ยของทั้ง 10 folds

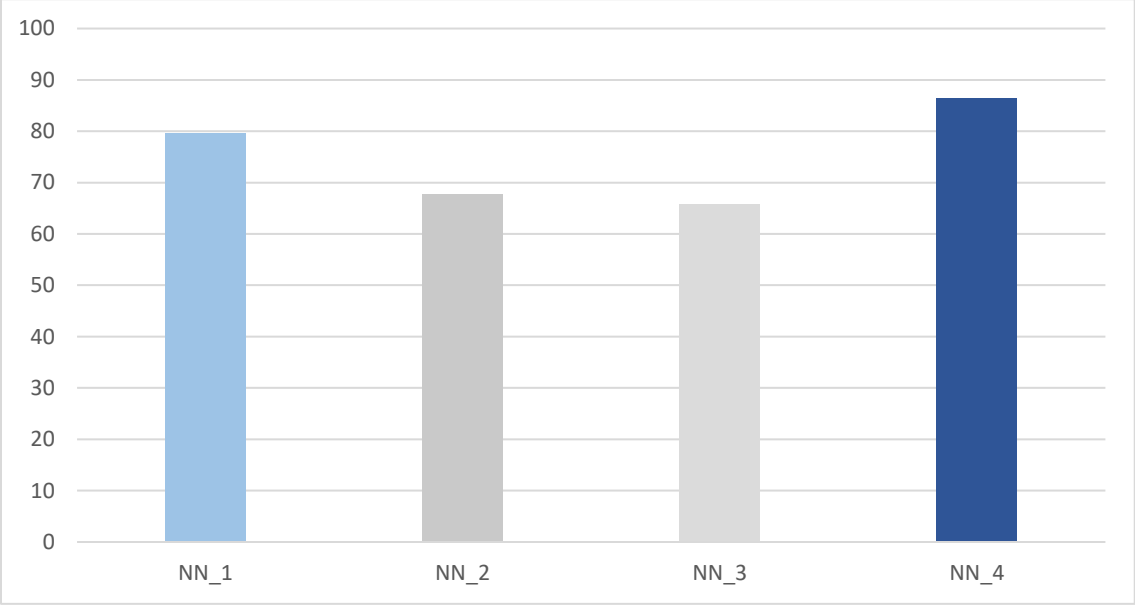
นำเสนอในรูปแบบแผนภูมิแท่งโดยเป็นเปอร์เซ็นต์ความถูกต้องในการทดสอบโดยเฉลี่ยทั้ง 10 folds ซึ่งพบว่าโครงสร้างที่ 1 มีเปอร์เซ็นต์ความถูกต้องโดยเฉลี่ยมากที่สุด โดยมีความเป็นยำถึง 70% และอันดับรองลงเป็นโครงสร้างที่ 4 มีความถูกต้องอยู่ที่ 68.67% แต่ในทางกลับกันโครงสร้างที่ 2 และโครงสร้างที่ 3 มีความถูกต้องเพียงแค่ 62% และ 60.5% ตามลำดับ



รูปที่ 5 แผนภูมิแท่งแสดงเปอร์เซ็นต์ความถูกต้องโดยเฉลี่ย

3. สรุปผลการทดลอง

จากการทดลองในหัวข้อที่ 3 จะพบว่า โครงสร้างที่ดีที่สุดที่เป็นอันดับหนึ่งและอันดับสอง คือ โครงสร้างที่ 1 และ โครงสร้างที่ 4 โดยพบว่าผลลัพธ์โดยเฉลี่ยทั้ง 10 folds โครงสร้างที่ 1 จะทำได้ดีกว่าโครงสร้างที่ 4 แต่ถึงอย่างไร ก็เป็นเพียงแค่แนวโน้มของประชากรโครโมโซมทั้งกลุ่ม ซึ่งเป้าหมายในการฝึกสอนด้วยวิธีเชิงพันธุกรรมนั้นจะต้องนำโครมโมโซมที่ดีที่สุดเป็นผลลัพธ์สุดท้ายซึ่งพบว่า เมื่อนำโครมโมโซมที่ดีที่สุดในแต่ละโครงสร้างมาใช้ทดสอบกับข้อมูลทดสอบในทุกๆ folds พบว่า โครงสร้างที่ 4 มีผลลัพธ์เปอร์เซ็นต์ความแม่นยำที่ดีที่สุด ซึ่งแม่นยำถึง 86.4% รองลงมาคือ โครงสร้างที่ 1 มีความแม่นยำ 79.5% และ อันดับสามและอันดับที่สี่ คือโครงสร้าง 2 และ ที่ 3 ตามลำดับ



รูปที่ 6 แผนภูมิแท่งแสดงเปอร์เซ็นต์ความแม่นยำของโครโมโซมที่ดีที่สุด

4. โปรแกรม

ผู้จัดการทำได้การ Preprocessing ในไฟล์ csv. ก่อนทำการนำเข้าเขียนโปรแกรม ซึ่งโปรแกรมที่ในการทดลองมี Source code ดังนี้

```
1. import numpy as np
2. import random
3. import math
4. from random import randint
5.
6. Input = np.genfromtxt('data/wdbc_input.csv', delimiter=',')
7. Output = np.genfromtxt('data/wdbc_output.csv', delimiter=',')
8.
9. def cross_validations_split(shape,folds):
10.     fold_size = int(shape * folds/100)
11.     k = 0
12.     index = []
13.     for i in range(1,folds+1):
14.         if i < folds:
15.             index.append([k,i*fold_size])
16.         else:
17.             index.append([k,shape])
18.         k = i*fold_size
19.     return index
20.
21.class GA(object):
22.     def __init__(self, hiddenSize, inputSize, outputSize):
23.
24.         # initiate layers
25.         self.inputSize = inputSize
26.         self.outputSize = outputSize
27.         self.hiddenSize = hiddenSize
28.
29.         layers = [self.inputSize] + self.hiddenSize + [self.outputSize]
30.         # initiate genes
31.         genes = []
32.         for i in range(len(layers)-1):
33.             g = np.random.uniform(-2,2,(layers[i], layers[i+1]))
34.             genes.append(g)
35.         self.genes = genes
36.         self.layers = layers
37.
38.     def feedForward(self, X):
```

```
        Output_node = X
        for i, g in enumerate(self.genes):

            v = np.dot(Output_node, g)
            Output_node = self.sigmoid(v)

        return Output_node

    def sigmoid(self, s, deriv=False):
        if (deriv == True):
            return s * (1-s)
        return 1/(1 + np.exp(-s))

    def object_funct(self, X, Y):
        self.acc = 0
        seed = randint(1, 100*100)

        np.random.seed(seed)
        np.random.shuffle(X)

        np.random.seed(seed)
        np.random.shuffle(Y)

        sum_err = 0
        for j, input in enumerate(X):
            target = Y[j]
            output = self.feedForward(input)

            if abs(output-target) < 0.5:
                    self.acc += 1

            sum_err += self._mse(target, output)

        self.fx = 1/math.log(1+sum_err/len(X))

        return self.fx,self.acc*100/len(X)

    def _mse(self, target, output):
        return np.average(abs(target - output)**2)

class selection(object):
    def __init__(self, chromosomes):

        # initiate fitness
```

```
        self.chromosomes = chromosomes
        self.Max = 1.2
        self.Min = 2-self.Max

    # initiate probability
    def prob(self):

        P = []
        N = len(chromosomes)
        Max = self.Max
        Min = self.Min

        for r,_ in enumerate(self.chromosomes):
            p = (Min + (Max - Min) * ((r-1)/(N-1)))/N
            P.append(p)

        return P

          def expect_values(self):

              ni = []
              P = self.prob()

              for p in P:
                  ni.append(p*len(self.chromosomes))

              return ni

          # Stochastic universal sampling
          def Stochastic_sampling(self):

              ni = self.expect_values()
              answers = []
              index = []
              ptr = np.random.uniform(0,1,1)[0]
              sum_pi = 0

              for i in range(len(self.chromosomes)):
                  sum_pi += ni[i]
                  while sum_pi > ptr :
                      index.append(i)
                      ptr+=1
              for i in index:
                  #print(self.chromosomes[i])
                  answers.append(self.chromosomes[i][1])
```

```
128.
129.             return answers
130.
131.
132.     def Crossover_chromosomes(mating_pool,no_chromosomes):
133.
134.         new_generation = []
135.         Hidden,Input,Output = mating_pool[0].hiddenSize,mating_pool[0].i
    nputSize,mating_pool[0].outputSize
136.
137.         t = random.randint(int(len(mating_pool)*0.8), no_chromosomes)
138.
139.         for i in range(t):
140.
141.             # random cuple
142.             couple = random.sample(mating_pool, 2)
143.
144.             # crossover
145.             child = GA(Hidden,Input,Output)
146.             for i, g in enumerate(child.genes):
147.                 for j in range(g.shape[1]):
148.                     k = random.randint(0,1)
149.                     g[:,j] = couple[k].genes[i][:,j].copy()
150.
151.             new_generation.append(child)
152.
153.         # if size of new_gen != mating_pool
154.         while len(new_generation) < no_chromosomes:
155.             child = random.choice(mating_pool)
156.             new_generation.append(child)
157.
158.         return new_generation
159.
160.     def Mutation(chromosomes):
161.         next_generation = chromosomes.copy()
162.         for c in next_generation:
163.             for i, g in enumerate(c.genes):
164.                 for j in range(g.shape[1]):
165.                     k = random.randint(0, 4) # 25% for mutate
166.                     if k == 1: # is mutate
167.                         mutate = np.random.uniform(-1,1,(g.shape[0]))
168.                         g[:,j] += mutate
169.
```

```
        return next_generation


chromosomes = []
no_generations = 100
no_chromosomes = 100

# initiate chromosomes P(0)
for i in range(no_chromosomes):
    c = GA([10,5,5,3],Input.shape[1],Output.shape[1])
    chromosomes.append(c)

# initiate best objective function
best_chromosome = [float('-inf'),chromosomes[0]]

obj_per_fold = []
best_per_fold = []

test_per_fold = []
acc_per_fold = []
for a,b in cross_validations_split(Input.shape[0],10):
    x_train = np.concatenate((Input[:a],Input[b+1:]))
    y_train = np.concatenate((Output[:a],Output[b+1:]))
    x_test = Input[a:b,:]
    y_test = Output[a:b]

    for t in range(1,no_generations+1):

        # evaluation objective function
        ranks = []
        mean_obj = []
        # train neural network with GA
        for c in chromosomes:

            fx,_ = c.object_funct(x_train, y_train)
            mean_obj.append(fx)
            ranks.append([fx,c])

            if fx > best_chromosome[0]:
                print("-- update best fitenss ",round(fx,6))
                best_chromosome[0] = fx
                best_chromosome[1] = c
```

```
212.
213.             # Linear ranks selection
214.             ranks = sorted(ranks, key = lambda x: (x[0]))
215.             mating_pool = selection(ranks).Stochastic_sampling()
216.
217.             # Crossover
218.             c_chromosomes = Crossover_chromosomes(mating_pool,no_chromos
    omes)
219.
220.             # Mutation
221.             new_generations = Mutation(c_chromosomes)
222.
223.             # P(t) = P(t+1)
224.             chromosomes = new_generations.copy()
225.
226.             print("---- average fitness",t," : ", np.mean(mean_obj))
227.
228.         mean_test = []
229.         acc_test = []
230.         for c in chromosomes:
231.             fx,acc = c.object_funct(x_test, y_test)
232.             mean_test.append(fx)
233.             acc_test.append(acc)
234.
235.         print("** test fitness : ",np.mean(mean_test))
236.         print("** aac test fitness : ",np.mean(acc_test))
237.
238.         test_per_fold.append(np.mean(mean_test))
239.         acc_per_fold.append(np.mean(acc_test))
240.         obj_per_fold.append(np.mean(mean_obj))
241.         best_per_fold.append(best_chromosome[0])
```