**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

Contents

[**สารบัญ 1**](#_Toc56025143)

[**1. การเตรียมข้อมูล 2**](#_Toc56025144)

[**1.1 คำอธิบายข้อมูล 2**](#_Toc56025145)

[**กระบวนการทำงานขั้นตอนวิธีเชิงพันธุกรรม 2**](#_Toc56025146)

[**1.2 ภาพรวมของระบบ 2**](#_Toc56025147)

[**1.3 การทำงานของระบบ 3**](#_Toc56025148)

[**สำหรับการทำงานของระบบนั้น แบ่งขั้นตอนการทำงานทั้งหมดเป็น 6 ขั้นตอน ดังนี้ 3**](#_Toc56025149)

[**2. การทดลอง 4**](#_Toc56025150)

[**2.1 การเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการฝึกสอนของแต่ละ folds 5**](#_Toc56025151)

[**2.2 การเปลี่ยนแปลงค่า MSE โดยเฉลี่ยการทดสอบของแต่ละ folds 6**](#_Toc56025152)

[**2.3 การเปลี่ยนแปลงโครโมโซมที่ดีที่สุดของแต่ละ folds 6**](#_Toc56025153)

[**2.4 เปอร์เซ็นต์ความถูกต้องโดยเฉลี่ยของทั้ง 10 folds 8**](#_Toc56025154)

[**3. สรุปผลการทดลอง 9**](#_Toc56025155)

[**4. โปรแกรม 10**](#_Toc56025156)

# การเตรียมข้อมูล

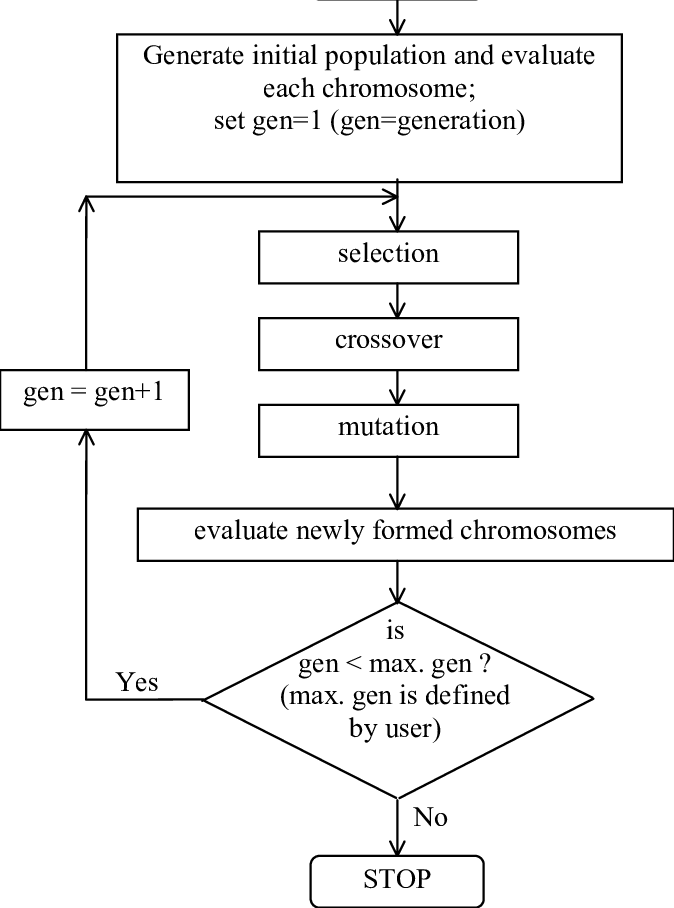
## **คำอธิบายข้อมูล**

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

## **ภาพรวมของระบบ**



**รูปที่ 1** แผนผังการทำงานของระบบ

## **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

# 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) ดังตางรางที่ 1

**ตารางที่ 1** แสดงพารามิเตอร์ในแต่ละโครงสร้าง

|  |  |
| --- | --- |
| **พารามิเตอร์** | **จำนวน** |
| Neural weights (genes) | ~U (-2,2) |
| Number of populations | 100 |
| Number of generations | 1000 |
| Objective function |  |

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

## **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 เป็นต้นไปพบว่า แนวโน้มค่า MSE ลดลงอย่างฉับพลันอย่างเห็นได้ชัด และจากโครงสร้างที่ 3 พบว่ามีผลลัพธ์ที่แย่ที่สุดในแต่ละโครงสร้าง

## **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 เป็นต้นไปพบว่า แนวโน้มค่า MSE ลดลงอย่างฉับพลันอย่างเห็นได้ชัด และจากโครงสร้างที่ 3 พบว่ามีผลลัพธ์ที่แย่ที่สุดในแต่ละโครงสร้าง

## **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

## **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

# สรุปผลการทดลอง

จาการทดลองในหัวข้อที่ 3 จะพบว่า โครงสร้างที่ดีที่สุดที่เป็นอันดับหนึ่งและอันดับสอง คือ โครงสร้างที่ 1 และ โครงสร้างที่ 4 โดยพบว่าผลลัพธ์โดยเฉลี่ยทั้ง 10 folds โครงสร้างที่ 1 จะทำได้ดีกว่า โครงสร้างที่ 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

# 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. Input = np.genfromtxt('data/wdbc\_input.csv', delimiter=',')
6. Output = np.genfromtxt('data/wdbc\_output.csv', delimiter=',')
7. def **cross\_validations\_split**(shape,folds):
8. fold\_size = int(shape \* folds/100)
9. k = 0
10. index = []
11. for i in **range**(1,folds+1):
12. if i < folds:
13. index.append([k,i\*fold\_size])
14. else:
15. index.append([k,shape])
16. k = i\*fold\_size
17. return index
18. class **GA**(object):
19. def **\_\_init\_\_**(self, hiddenSize, inputSize, outputSize):
20. *# initiate layers*
21. self.inputSize = inputSize
22. self.outputSize = outputSize
23. self.hiddenSize = hiddenSize
25. layers = [self.inputSize] + self.hiddenSize + [self.outputSize]
26. *# initiate genes*
27. genes = []
28. for i in **range**(**len**(layers)-1):
29. g = np.random.uniform(-2,2,(layers[i], layers[i+1]))
30. genes.append(g)
31. self.genes = genes
32. self.layers = layers
33. def **feedForward**(self, X):
34. Output\_node = X
35. for i, g in **enumerate**(self.genes):
37. v = np.dot(Output\_node, g)
38. Output\_node = self.sigmoid(v)
39. return Output\_node
40. def **sigmoid**(self, s, deriv=False):
41. if (deriv == True):
42. return s \* (1-s)
43. return 1/(1 + np.exp(-s))
44. def **object\_funct**(self, X, Y):
45. self.acc = 0
46. seed = randint(1, 100\*100)
47. np.random.seed(seed)
48. np.random.shuffle(X)
49. np.random.seed(seed)
50. np.random.shuffle(Y)
51. sum\_err = 0
52. for j, **input** in **enumerate**(X):
53. target = Y[j]
54. output = self.feedForward(**input**)
55. if **abs**(output-target) < 0.5:
56. self.acc += 1
57. sum\_err += self.\_mse(target, output)
58. self.fx = 1/math.log(1+sum\_err/**len**(X))
60. return self.fx,self.acc\*100/**len**(X)
61. def **\_mse**(self, target, output):
62. return np.average(**abs**(target - output)\*\*2)
63. class **selection**(object):
64. def **\_\_init\_\_**(self, chromosomes):
65. *# initiate fitness*
66. self.chromosomes = chromosomes
67. self.Max = 1.2
68. self.Min = 2-self.Max
70. *# initiate probability*
71. def **prob**(self):
72. P = []
73. N = **len**(chromosomes)
74. Max = self.Max
75. Min = self.Min
76. for r,\_ in **enumerate**(self.chromosomes):
77. p = (Min + (Max - Min) \* ((r-1)/(N-1)))/N
78. P.append(p)
79. return P
80. def **expect\_values**(self):
81. ni = []
82. P = self.prob()
84. for p in P:
85. ni.append(p\***len**(self.chromosomes))
86. return ni
87. *# Stochastic universal sampling*
88. def **Stochastic\_sampling**(self):
89. ni = self.expect\_values()
90. answers = []
91. index = []
92. ptr = np.random.uniform(0,1,1)[0]
93. sum\_pi = 0
94. for i in **range**(**len**(self.chromosomes)):
95. sum\_pi += ni[i]
96. while sum\_pi > ptr :
97. index.append(i)
98. ptr+=1
99. for i in index:
100. *#print(self.chromosomes[i])*
101. answers.append(self.chromosomes[i][1])
103. return answers
104. def **Crossover\_chromosomes**(mating\_pool,no\_chromosomes):
105. new\_generation = []
106. Hidden,Input,Output = mating\_pool[0].hiddenSize,mating\_pool[0].inputSize,mating\_pool[0].outputSize
108. t = random.randint(int(**len**(mating\_pool)\*0.8), no\_chromosomes)
109. for i in **range**(t):
111. *# random cuple*
112. couple = random.sample(mating\_pool, 2)
113. *# crossover*
114. child = GA(Hidden,Input,Output)
115. for i, g in **enumerate**(child.genes):
116. for j in **range**(g.shape[1]):
117. k = random.randint(0,1)
118. g[:,j] = couple[k].genes[i][:,j].copy()
120. new\_generation.append(child)
121. *# if size of new\_gen != mating\_pool*
122. while **len**(new\_generation) < no\_chromosomes:
123. child = random.choice(mating\_pool)
124. new\_generation.append(child)
125. return new\_generation
126. def **Mutation**(chromosomes):
127. next\_generation = chromosomes.copy()
128. for c in next\_generation:
129. for i, g in **enumerate**(c.genes):
130. for j in **range**(g.shape[1]):
131. k = random.randint(0, 4) *# 25% for mutate*
132. if k == 1: *# is mutate*
133. mutate = np.random.uniform(-1,1,(g.shape[0]))
134. g[:,j] += mutate
136. return next\_generation
137. chromosomes = []
138. no\_generations = 100
139. no\_chromosomes = 100
140. *# initiate chromosomes P(0)*
141. for i in **range**(no\_chromosomes):
142. c = GA([10,5,5,3],Input.shape[1],Output.shape[1])
143. chromosomes.append(c)
145. *# initiate best objective function*
146. best\_chromosome = [float('-inf'),chromosomes[0]]
147. obj\_per\_fold = []
148. best\_per\_fold = []
149. test\_per\_fold = []
150. acc\_per\_fold = []
151. for a,b in cross\_validations\_split(Input.shape[0],10):
152. x\_train = np.concatenate((Input[:a],Input[b+1:]))
153. y\_train = np.concatenate((Output[:a],Output[b+1:]))
154. x\_test = Input[a:b,:]
155. y\_test = Output[a:b]
156. for t in **range**(1,no\_generations+1):
157. *# evaluation objective function*
158. ranks = []
159. mean\_obj = []
160. *# train neural network with GA*
161. for c in chromosomes:
162. fx,\_ = c.object\_funct(x\_train, y\_train)
163. mean\_obj.append(fx)
164. ranks.append([fx,c])
165. if fx > best\_chromosome[0]:
166. **print**("-- update best fitenss ",**round**(fx,6))
167. best\_chromosome[0] = fx
168. best\_chromosome[1] = c
169. *# Linear ranks selection*
170. ranks = **sorted**(ranks, key = lambda x: (x[0]))
171. mating\_pool = selection(ranks).Stochastic\_sampling()
172. *# Crossover*
173. c\_chromosomes = Crossover\_chromosomes(mating\_pool,no\_chromosomes)
174. *# Mutation*
175. new\_generations = Mutation(c\_chromosomes)
176. *# P(t) = P(t+1)*
177. chromosomes = new\_generations.copy()
178. **print**("---- average fitness",t," : ", np.mean(mean\_obj))
180. mean\_test = []
181. acc\_test = []
182. for c in chromosomes:
183. fx,acc = c.object\_funct(x\_test, y\_test)
184. mean\_test.append(fx)
185. acc\_test.append(acc)
186. **print**("\*\* test fitness : ",np.mean(mean\_test))
187. **print**("\*\* aac test fitness : ",np.mean(acc\_test))
188. test\_per\_fold.append(np.mean(mean\_test))
189. acc\_per\_fold.append(np.mean(acc\_test))
190. obj\_per\_fold.append(np.mean(mean\_obj))
191. best\_per\_fold.append(best\_chromosome[0])