# Datamining Homework : Classification

นาย ดรันภพ เป็งคำตา

580610642

**รูปแบบของการทดลอง**

เป็นการทดลองเพื่อทดสอบการทำงาน และประสิทธิภาพของการทำ Classification ของ algorithm 2 แบบ คือ Neural network และ Decision tree ซึ่งทั้งสองจะทดสอบความแม่นยำด้วย การใช้ K-fold validation

**รูปแบบของข้อมูลที่ใช้ทดสอบ**

เป็นชุดข้อมูลของนักศึกษาจำนวน 395 คน ประกอบด้วย attribute ทั้งหมด 33 อย่าง ซึ่งประกอบด้วยข้อมูลลักษณะต่าง ๆดังนี้ เป็นattributeที่เป็นไปได้ 2 ค่า (binary), เป็นจำนวนเต็มในช่วง ๆ หนึ่ง (numeric) และเป็น ข้อความต่าง ๆ กันจำนวนหนึ่ง (nominal) โดยattribute ที่ต้องการจำแนกคือ G3 เป็นข้อมูลของเกรดที่นักศึกษาคนนั้นได้เมื่อจบคอสนั้น ๆ ซึ่งเป็นค่าที่เป็นไปได้ในช่วง [0, 20]

ดังนั้นจึงจะให้ผลของการทดสอบเป็นการจำแนกคลาสผลลัพธ์ 21 แบบ แต่ละคลาสคลาสจะเป็นตัวแทนของค่าตั้งแต่ 0 – 20

**การ Preprocess ข้อมูล**

เนื่องจากข้อมูลที่ได้มา พบว่าเป็นข้อมูลที่ไม่มี missing value จึงไม่จำเป็นที่จะต้องจัดการกับปัญหานี้ ในขั้นตอนนี้ได้ทำการ normalize ข้อมูลทั้งหมดให้อยู่อยู่ในช่วง [0, 1] อย่างเท่าเทียมกัน ยกเว้นเว้น attribute **‘G3’** เนื่องจากเป็น output class ที่ต้องการทำการจำแนก และ เนื่องจากพบว่า attribute ทั้งหมด มีข้อมูลอยู่สามแบบ จังได้ออกแบบ function สำหรับการ normalize ไว้เพื่อความสะดวกในการใช้งาน

* Function สำหรับ normalize ข้อมูล

def normalize(input):

input['school'] = input['school'].apply(lambda x:binary('GP', x))

input['sex'] = input['sex'].apply(lambda x:binary('F', x))

input['age'] = input['age'].apply(lambda x:scaling(15, 22, x))

input['address'] = input['address'].apply(lambda x:binary('U', x))

input['famsize'] = input['famsize'].apply(lambda x:binary('LE3', x))

input['Pstatus'] = input['Pstatus'].apply(lambda x:binary('T', x))

input['Medu'] = input['Medu'].apply(lambda x:scaling(0, 4, x))

input['Fedu'] = input['Fedu'].apply(lambda x:scaling(0, 4, x))

input['Mjob'] = input['Mjob'].apply(lambda x:nominal(['teacher', 'health', 'services', 'at\_home', 'other'], x))

input['Fjob'] = input['Fjob'].apply(lambda x:nominal(['teacher', 'health', 'services', 'at\_home', 'other'], x))

input['reason'] = input['reason'].apply(lambda x:nominal(['home', 'reputation', 'course', 'other'], x))

input['guardian'] = input['guardian'].apply(lambda x:nominal(['mother', 'father', 'other'], x))

input['traveltime'] = input['traveltime'].apply(lambda x:scaling(1, 4, x))

input['studytime'] = input['studytime'].apply(lambda x:scaling(1, 4, x))

input['failures'] = abs(input['failures'].apply(lambda x:scaling(1, 4, x)))

input['schoolsup'] = input['schoolsup'].apply(lambda x:binary('yes', x))

input['famsup'] = input['famsup'].apply(lambda x:binary('yes', x))

input['paid'] = input['paid'].apply(lambda x:binary('yes', x))

input['activities'] = input['activities'].apply(lambda x:binary('yes', x))

input['nursery'] = input['nursery'].apply(lambda x:binary('yes', x))

input['higher'] = input['higher'].apply(lambda x:binary('yes', x))

input['internet'] = input['internet'].apply(lambda x:binary('yes', x))

input['romantic'] = input['romantic'].apply(lambda x:binary('yes', x))

input['famrel'] = input['famrel'].apply(lambda x:scaling(1, 5, x))

input['freetime'] = input['freetime'].apply(lambda x:scaling(1, 5, x))

input['goout'] = input['goout'].apply(lambda x:scaling(1, 5, x))

input['Dalc'] = input['Dalc'].apply(lambda x:scaling(1, 5, x))

input['Walc'] = input['Walc'].apply(lambda x:scaling(1, 5, x))

input['health'] = input['health'].apply(lambda x:scaling(1, 5, x))

input['absences'] = input['absences'].apply(lambda x:scaling(0, 93, x))

input['G1'] = input['G1'].apply(lambda x:scaling(0, 20, x))

input['G2'] = input['G2'].apply(lambda x:scaling(0, 20, x))

* Function สำหรับ normalize ข้อมูลที่เป็นช่วงค่า (numeric)

def scaling(min, max, x):

return (x - min)/(max - min)

* Function สำหรับ normalize ข้อมูลที่เป็นสองค่า (binary)

def binary(a, x):

return 1 if x == a else 0

* Function สำหรับ normalize ข้อมูลที่เป็นชุดของข้อความ (nominal)

def nominal(p\_list, x):

for n, i in enumerate(p\_list):

if i == x:

return scaling(0, len(p\_list)-1, n)

# การทำ K – fold validation

เป็นการทำการเทรนโมเดล และทดสอบข้อมูล ทั้งหมด k รอบ แล้วนำค่าความแม่นยำของทุกรอบมาเฉลี่ยกัน โดยในทีนี้ได้ใช้ค่า k=10 โดยในแต่ละ fold ได้ทำการแบ่งข้อมูลออกเป็น ข้อมูลสำหรับการ เทรนโมเดล 356 ตัว และ สำหรับการทดสอบ 39 ตัว โดยทำการเลือกข้อมูลโดยวิธีการสุ่ม โดยรับประกันว่าจะไม่มีข้อมูลที่ซ้ำกันถูกหยิบออกมา

# Neural network

เป็นการออกแบบโค้ดการทำงานตามหลักการของ OOP ซึ่งจะเลียนแบบโครงสร้างการทำงานตาม หลักการทำงานของ neuron network กล่วคือ มีการสร้างคลาส Neuron Network ซึ่งประกอบด้วยคลาสย่อยคือ คลาสของ Layer และในคลาสของ Layer ยังประกอบด้วยคลาสของ Node ดังนั้นจะทำให้โค้ดที่ได้มีการทำงานที่มีระเบียบและโครงสร้างการทำงานที่ชัดเจน สามารถเพิ่มลดจำนวน Layer และ Node ได้อย่างง่ายดาย ตามหลักการทำงานของ OOP

**การประกาศ และการเพิ่ม layer ของ neural network**

ตอนทดสอบได้ทำการทดลองเพิ่มและลด เลเยอร์ลงหลายครั้งแต่กลับไม่ได้ผลการทดลองที่น่าพอใจนัก จึงจะขอกล่าวในบทสรุปถัดไป

nn = neuralNetwork.NeuralNetwork(len(i[0])-1, 0.01) # 0.01 Learning rate

nn.addHidden(21) # Hidden layers

nn.addHidden(18)

nn.addHidden(15)

nn.addHidden(13)

nn.addHidden(10)

nn.addHidden(21) # output layer

**การ Normalize และการ เทรน**

จะเห็นได้ว่ามีการตั้งจำนวนการเทรนไว้ที่ 20 รอบ เนื่องจากการเทรนแต่ละรอบกินระยะว่าประมาณ 20 วินาที จึงไม่ได้สามารถที่จะเพิ่มจำนวนรอบได้มากนักเนื่องจากข้อจำกัดของเวลา

input.normalize(i[0]) # training set

input.normalize(i[1]) # test set

for r in range(20): # epoch

for j in range(len(i[0])): #feed each row

inp = list(i[0].iloc[j][:-1])

expect = i[0].iloc[j][-1:]

nn.train(inp, expect)

**การ ทดสอบของ neural network**

ทดสอบด้วยการตรวจสอบว่าคลาสที่ออกได้ เท่ากับผลลัพธ์หรือไม่ หากถูกต้อง จะเป็น 1 หากไม่จะเป็น 0 เลย ไม่ได้มีการประเมินความใกล้เคียง เช่น หากผลได้ 9 แต่ที่ถูกต้องคือ 10 ก็จะยังถูกนับว่าเป็น 0

for j in range(len(i[1])): #feed test

inp = list(i[1].iloc[j][:-1])

expect = i[1].iloc[j][-1:]

nn.setInput(inp)

nn.allProcess()

o = nn.classify()

if o == int(expect):

success += 1

**ผลการการทดสอบ**

|  |  |  |  |
| --- | --- | --- | --- |
| fold | accuracy | fold | accuracy |
| 1 | 0.23076923076923078 | **6** | 0.10256410256410256 |
| 2 | 0.10256410256410256 | **7** | 0.1282051282051282 |
| 3 | 0.15384615384615385 | **8** | 0.10256410256410256 |
| 4 | 0.1794871794871795 | **9** | 0.23076923076923078 |
| 5 | 0.10256410256410256 | **10** | 0.2564102564102564 |
| Average accuracy | | **0.158974358974358935** | |

**สรุปผลการทดสอบของ neural network**

จากการทดสอบเห็นได้ว่า ค่าความถูกต้องที่ได้นั้นน้อยมาก แสดงให้เห็นถึงความผิดพลาดอย่างชัดเจน คาดว่าความผิดพลาดเกิดมาจากการที่กำหนดจำนวนรอบในการเทรนน้อยเกินไป และไม่ได้มีการ optimization การคำนวณที่ดี จึงส่งผลให้เกิดความล่าช้าในการคำนวณและทำให้รอบการเทรนน้อยกว่าที่ควรจะเป็น

ทั้งยังไม่ได้ทดสอบกับโครงสร้าง neural network ที่หลากหลายมากพอ กล่าวคือ ไม่ได้ทดสอบกับ layer ที่หลากหลายขึ้น และ จำนวนnode ในแต่ละเลเยอต่าง ๆ กันไป

ซึ่งเป็นความผิดพลาดทองผู้จัดทำที่ไม่ได้ทำให้ละเอียดและรอบคอบมากกว่านี้

# Decision tree

# เป็นการสร้าง tree สำหรับการ classify ข้อมูลตาม algorithm ID4.5 โดยในการสร้าง tree นี้ได้ตั้งสมมุติฐานว่า ในแต่ละชั้นของ tree จะแบ่ง class ของ เป้าหมายให้ออกเป็น 2 คลาส โดยจะใช้การแบ่งจากจุดกึ่งกลางของคลาสที่เป็นไปได้ ยกตัวอย่างเช่น root node จะมีคลาสเป็นเป็นได้ 21 คลาส โดยจะทำการแบ่งเป็นสองคลาส คือ 0-9 และ 10-20 และใช้ ID4.5 เลือก attribute ในการแบ่ง และจะทำไปเรื่อย ๆ จนสุด

**โครงสร้างของ class Decision tree**

class DecisionTree:

class Node:

def \_\_init\_\_(self, target):

self.target = target

self.child = []

self.attr = ''

self.avaiable\_attr = []

self.theshold = 0

def classify(self, input):

if len(self.target) == 1:

return self.target[0]

else:

o = input[self.attr]

if o <= self.theshold:

return self.child[0].classify(input)

else:

return self.child[1].classify(input)

# END Node

def \_\_init\_\_(self, target):

self.tree = []

# END DecisionTree

**Function สำหรับสร้าง decision tree**

def buildTree(data, root, attr, count=1 ):

root\_node = root

if len(root\_node.target) <= 1:

pass

else:

m\_p = len(root\_node.target) / 2

l\_t = root\_node.target[:math.floor(m\_p)]

r\_t = root\_node.target[math.floor(m\_p):]

d\_l = data.loc[data['G3'] <= l\_t[-1:][0]]

d\_r = data.loc[data['G3'] > l\_t[-1:][0]]

d\_ls = len(d\_l)

d\_rs = len(d\_r)

info = getInfo(d\_ls, d\_rs)

attr\_info = []

g\_info = []

for i in root\_node.avaiable\_attr:

x, y = attrInfoGain(i, 'G3', data, d\_l, d\_r)

attr\_info.append(x)

g\_info.append(y)

gain = np.full(len(attr\_info), info) - attr\_info

max = [0,0]

for n, i in enumerate(gain):

tmp = i/g\_info[n]

if tmp > max[0]:

max = [tmp, n]

root\_node.attr = root\_node.avaiable\_attr.pop(max[1])

root\_node.theshold = findMid(data, root\_node.attr)

if count == 0:

l\_node = DecisionTree.Node(l\_t)

r\_node = DecisionTree.Node(r\_t)

else:

l\_node = DecisionTree.Node(root\_node.target)

r\_node = DecisionTree.Node(root\_node.target)

l\_node.avaiable\_attr = cp.copy(root\_node.avaiable\_attr)

r\_node.avaiable\_attr = cp.copy(root\_node.avaiable\_attr)

root\_node.child.append(l\_node)

root\_node.child.append(r\_node)

buildTree(data, l\_node, 'G3')

buildTree(data, r\_node, 'G3')

**ผลการทำงานของ Decision tree**

|  |  |  |  |
| --- | --- | --- | --- |
| fold | accuracy | fold | accuracy |
| 1 | 0.10256410256410256 | **6** | 0.10256410256410256 |
| 2 | 0.1794871794871795 | **7** | 0.1282051282051282 |
| 3 | 0.15384615384615385 | **8** | 0.2564102564102564 |
| 4 | 0.07692307692307693 | **9** | 0.20512820512820512 |
| 5 | 0.07692307692307693 | **10** | 0.05128205128205128 |
| Average accuracy | | **0.110256410256410229** | |

# สรุปผลการทดลอง

เห็นได้ว่าผลที่ได้จาก Decision tree มีความแม่นยำค่อนข้างต่ำ ถึงต่ำมาก เพราะจากสมมุติฐานที่ตั้งขึ้นมา จะทำให้ tree มีความสูงได้เพียง เท่า ซึ่งจะได้ใช้ attribute ในการclassify เพียงเท่าความสูงเท่านั้น จึงส่งผลให้ ความแม่นยำต่ำถึงขนาดนี้

เมื่อนำมาเปรียบเทียบกับ neural network จะเห็นว่า neural network มีความแม่นยำที่มากกว่าเพียงเล็กน้อยและยังถูกจัดว่า ต่ำ อยู่ดี จึงได้ข้อสรุปที่ว่า

* ควรให้เวลาในการเทรน neural network มากกว่านี้
* ควรศึกษา algorithm ของ decision มาใหม่ เพราะว่าสมมุติฐานที่ตนตั้งขึ้นมาหละหลวมเกินไป
* ควรเริ่มทำงานให้ไวกว่านี้ เพราะงานที่ได้นอกจากไม่มีเวลาให้พอแล้ว ยังไม่มีเวลาในการศึกษา decision tree ให้เข้าใจยิ่งขึ้น
* งานนี้ทำตามความเข้าใจของตนและความรู้ที่รวบรวมมาได้ ผลลัพธ์ความแม่นยำจึงเป็นดังที่เห็น