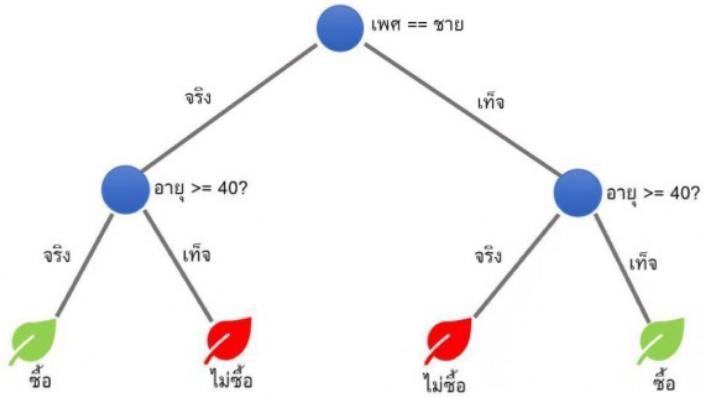


Decision Tree



Classification (discrete-ไม่ต่อเนื่องทางเวลา, continuous-ต่อเนื่องทางเวลา)
<ถ้าแบบ Regresion ก็จะใช้ Decis Tree อีกแบบหนึ่ง เราจะไม่พูดถึง 555 >

Data =

X_1	X_2	...	X_D	Y
x_1^1	x_1^2	...	x_1^D	y_1
x_2^1	x_2^2	...	x_2^D	y_2
x_3^1	x_3^2	...	x_3^D	y_3
:	:	⋮	⋮	⋮
x_N^1	x_N^2	...	x_N^D	y_N

X คือ ตัวแปรต้น (Feature)

Y คือ ตัวแปรตาม (Target)

N คือ จำนวนตัวอย่างทั้งหมด

Ex.

Data =

เพศ	อายุ	ชื่อคุณ ?
หญิง	40	ไม่ชือ
หญิง	50	ไม่ชือ
ชาย	20	ไม่ชือ
ชาย	40	ชือ
ชาย	50	ชือ
หญิง	20	ชือ

```

if เพศ == ชาย :
    if อายุ >= 40:
        ชื่อคุณ = ชือ
    else :
        ชื่อคุณ = ไม่ชือ
else :
    if อายุ >= 40:
        ชื่อคุณ = ไม่ชือ
    else :
        ชื่อคุณ = ชือ
  
```

$$X = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^D \\ x_2^1 & x_2^2 & \cdots & x_2^D \\ x_3^1 & x_3^2 & \cdots & x_3^D \\ \vdots & \vdots & \ddots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^D \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}$$

x_3^D คือ Feature ที่ D ของ Example ที่ 3

y_2 คือ Target ของ Example ที่ 2

- วัดความเป็นระเบียบข้อมูล (Gain) -> ก่อนจัด ลบ หลังจัด
 - จากผลรวมความแปรปรวนทุกกรณี (Gini)

$$Gain(parent, children) = Gini_{parent} - Gini_{children}$$

$$Gini_{parent} = 1 - \sum_{k \in K} P(k | parent)^2$$

$$\text{ Hosung } Gini_{children} = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

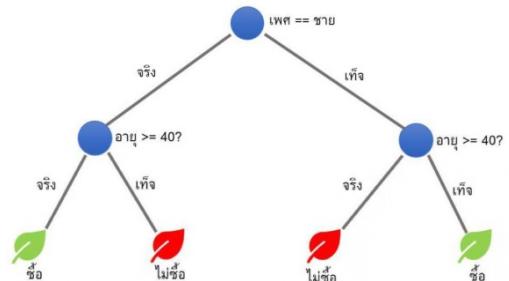
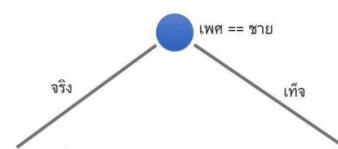
$$Weight_{true} = \frac{\#(true \cap children)}{\#children}$$

$$Weight_{false} = \frac{\#(false \cap children)}{\#children}$$

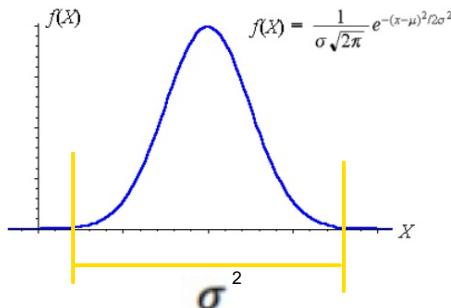
$$Gini_{true} = 1 - \sum_{k \in K} P(k | true)^2$$

$$Gini_{false} = 1 - \sum_{k \in K} P(k \mid false)^2$$

ผลลัพธ์ค่า Gain
ค่ามาก = ข้อมูลมีความระเบียบมาก
ค่าน้อย = ข้อมูลมีความเป็นระเบียบน้อย
0 = ข้อมูลไม่มีการเปลี่ยนแปลง



Normal Distribution



Standard Deviation

$$\sigma = \sqrt{\frac{\sum (x - \bar{x})^2}{n-1}}$$

$$\bar{X} = \frac{\text{Sum}}{n} = \mu$$

76	84	69	92	58
89	73	97	85	77

แบบ Gini

$$\sigma^2 = \sum_{i=1}^n p_i(x_i - \mu)^2$$

$$\mu = \sum_{i=1}^n p_i x_i = \text{ค่าเฉลี่ย}$$

$$X = \{1, 3, 3, 5, 5, 5, 9, 9\}$$

$$\frac{\text{Sum}}{n} = \mu$$

แบบที่เรารู้นเดียว

$$\mu = \frac{1 + 3 + 3 + 5 + 5 + 5 + 9 + 9}{8} = \frac{40}{8} = 5$$

ลองปรับมุมมอง

$$\mu = \frac{1}{8} + \frac{3+3}{8} + \frac{5+5+5}{8} + \frac{9+9}{8}$$

$$= \frac{1 \cdot 1}{8} + \frac{2 \cdot 3}{8} + \frac{3 \cdot 5}{8} + \frac{2 \cdot 9}{8}$$

$$= \frac{1}{8} \cdot 1 + \frac{2}{8} \cdot 3 + \frac{3}{8} \cdot 5 + \frac{2}{8} \cdot 9$$

$$= p_1 x_1 + p_2 x_2 + p_3 x_3 + p_4 x_4$$



$$\mu = \sum_{i=1}^4 p_i x_i$$

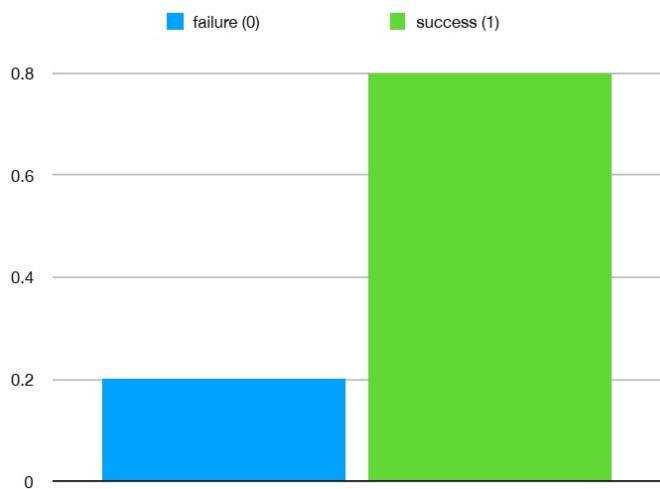
< หาความแปรปรวน (Variance ^2)แบบทั่วไป >

$$\sigma^2 = \sum_{i=1}^n p_i(x_i - \mu)^2$$

$$\begin{aligned} &= \frac{P_i}{N} \frac{(x_i - \mu)^2}{N} + \dots + \frac{(x_n - \mu)^2}{N} \\ &= P_i \left[(x_i - \mu)^2 + \dots + (x_n - \mu)^2 \right] \end{aligned}$$

ແບນ Gini $\sigma^2 = \sum_{i=1}^n p_i(x_i - \mu)^2$

ให้ p คือ ความน่าจะเป็นของ success
 $1 - p$ คือ ความน่าจะเป็นของ failure



จาก $\mu = \sum_{i=1}^n p_i x_i$

$\mu = (1 - p)0 + p(1) = p$

จาก $\sigma^2 = \sum_{i=1}^n p_i(x_i - \mu)^2$



$$\sigma^2 = (1 - p)(0 - p)^2 + p(1 - p)^2$$

$$= (1 - p)p^2 + p(1 - p)^2$$

$$= p^2 - p^3 + p(1 - 2p + p^2)$$

$$= p^2 - p^3 + p - 2p^2 + p^3$$

$$= p - p^2$$

$$= p(1 - p)$$

Pi -> False = 1-P
Pi -> True = P

Xi -> False = 0
Xi -> True = 1



EX (หาความแปรปรวน)

$$S = \{A, A, A, A, B, B, B, C, C, C, C, C\}$$

ความแปรปรวนที่จะได้ A หรือ ไม่ได้ A สามารถหาได้จาก

$$\sigma_A^2 = p_A(1 - p_A)$$

$$= \frac{4}{12}(1 - \frac{4}{12}) = 2/9$$

ความแปรปรวนที่จะได้ B หรือ ไม่ได้ B สามารถหาได้จาก

$$\sigma_B^2 = p_B(1 - p_B)$$

$$= \frac{3}{12}(1 - \frac{3}{12}) = 3/16$$

ความแปรปรวนที่จะได้ C หรือ ไม่ได้ C สามารถหาได้จาก

$$\sigma_C^2 = p_C(1 - p_C)$$

$$= \frac{5}{12}(1 - \frac{5}{12}) = 35/144$$

$$\sigma^2 = p(1 - p)$$

$$\sigma^2 = p(1 - p)$$

$$\sigma^2 = p(1 - p)$$

Gini Impurity คือการหาผลรวมความแปรปรวนของทุกกรณีที่เกิดขึ้น

$$\begin{aligned} Gini \text{ Impurity} &= \sigma_A^2 + \sigma_B^2 + \sigma_C^2 \\ &= p_A(1 - p_A) + p_B(1 - p_B) + p_C(1 - p_C) \\ &= p_A - p_A^2 + p_B - p_B^2 + p_C - p_C^2 \\ &= p_A + p_B + p_C - p_A^2 - p_B^2 - p_C^2 \\ &= 1 - \sum_{k \in \{A, B, C\}} p_k^2 \end{aligned}$$

ความน่าจะเป็นรวม
ทุกเหตุการณ์ = 1

จดจำให้อยู่ในสตอร์

$$Gini \text{ Impurity} = 1 - \sum_{k \in K} p_k^2$$

ตัวอย่างการคำนวณ *Gini Impurity*

ตัวอย่าง 1

$$S_1 = \{A, A, A, A, A, A, A, A, A\}$$

จาก $Gini = 1 - \sum_{k \in K} p_k^2$
 $Gini = 1 - 1^2$ มีแค่ A = $10/10 = 1$
 $= 0$ คือ ไม่มีการเปลี่ยนอะไรเลย

ตัวอย่าง 2

$$S_1 = \{A, A, A, A, A, B, B, B, B\}$$

จาก $Gini = 1 - \sum_{k \in K} p_k^2$
 $Gini = 1 - (\frac{1}{2})^2 - (\frac{1}{2})^2$
 $= 1 - \frac{1}{4} - \frac{1}{4}$
 $= 0.5$

- วัดความเป็นระเบียบข้อมูล (Gain) -> ก่อนจัด ลบ หลังจัด

- จากผลรวมความแปรปรวนทุกกรณี (Gini)

ก่อน

หลัง

$$Gain(parent, children) = Gini_{parent} - Gini_{children}$$

ผลลัพธ์ค่า Gain

ค่ามาก = ข้อมูลมีความระเบียบมาก

ค่าน้อย = ข้อมูลมีความเป็นระเบียบน้อย

0 = ข้อมูลไม่มีการเปลี่ยนแปลง

ก่อน

$$Gini_{parent} = 1 - \sum_{k \in K} P(k | parent)^2$$

$$Gini Impurity = 1 - \sum_{k \in K} p_k^2$$

หลัง

$$Gini_{children} = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

$$Weight_{true} = \frac{\#(true \cap children)}{\#children}$$

$$Weight_{false} = \frac{\#(false \cap children)}{\#children}$$

$$Gini_{true} = 1 - \sum_{k \in K} P(k | true)^2$$

$$Gini_{false} = 1 - \sum_{k \in K} P(k | false)^2$$

Ex.

Data =

เพศ	อายุ	ชื่อคุณ ?
หญิง	40	ไม่รู้
หญิง	50	ไม่รู้
ชาย	20	ไม่รู้
ชาย	40	รุ่ง
ชาย	50	รุ่ง
หญิง	20	รุ่ง

$$X = \begin{bmatrix} \text{หญิง} \\ \text{หญิง} \\ \text{ชาย} \\ \text{ชาย} \\ \text{ชาย} \\ \text{หญิง} \\ \text{หญิง} \end{bmatrix}, Y = \begin{bmatrix} 40 \\ 50 \\ 20 \\ 40 \\ 50 \\ 20 \end{bmatrix}$$

unique_value(Feature = เพศ) = [ชาย, หญิง]
 unique_value(Feature = อายุ) = [20, 40, 50]



question(Feature = เพศ) = [ชาย]

question(Feature = อายุ) = [40]

all_question = [ชาย, 40]

ด้วยจำนวนข้อมูลน้อย
ค่ากล่างมาใช้

* ในการที่ Feature ที่เป็น discrete และมี unique value 2 ค่า เราสามารถตั้ง
คำถามโดยใช้ unique value ค่าเดียวได้ (ทำแบบนี้เพื่อ optimize การคำนวณ)

* ในการที่ Feature ที่เป็น continuous เราสามารถตั้งคำถามโดยไม่ใช้ min และ
max ของ Feature นั้น ๆ ได้ (ทำแบบนี้เพื่อ optimize การคำนวณ)

* ในการที่ Feature เป็น discrete จะตั้งคำถามโดยการใช้เครื่องหมาย =

* ในการที่ Feature เป็น continuous จะตั้งคำถามโดยการใช้เครื่องหมาย \geq , \leq

$$Gain(parent, children) = Gini_{parent} - Gini_{children}$$

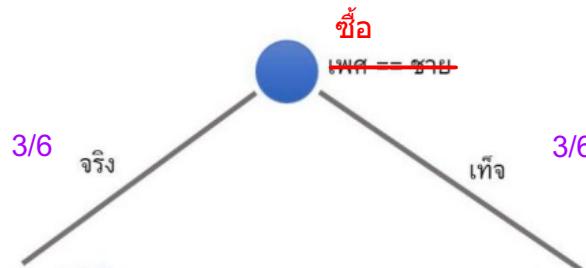
$$Gain(Training Set, \text{เพศ} = \text{ชาย}) = Gini_{Training Set} - Gini_{\text{เพศ}=\text{ชาย}}$$

สิ่งที่ต้องหา
ความเป็นระเบียบของข้อมูล
เทียบระหว่าง ก่อนจัด กับ หลังจัด

ก่อนจัด

$$\begin{aligned} Gini_{Training Set} &= 1 - [P(\text{ชื่อ})^2 + P(\text{ไม่ชื่อ})^2] \\ &= 1 - (3/6)^2 - (3/6)^2 \\ &= 1 - (1/2)^2 - (1/2)^2 = 0.5 \end{aligned}$$

$$Gini = 1 - \sum_{k \in K} P(k | parent)^2$$



เพศ	อายุ	ชื่อคุณ ?
หญิง	40	ไม่มีชื่อ
หญิง	50	ไม่มีชื่อ
ชาย	20	ไม่มีชื่อ
ชาย	40	มีชื่อ
ชาย	50	มีชื่อ
หญิง	20	มีชื่อ

หลังจัด

$$Gini(children) = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

$$Gini_{\text{เพศ=ชาย}} = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

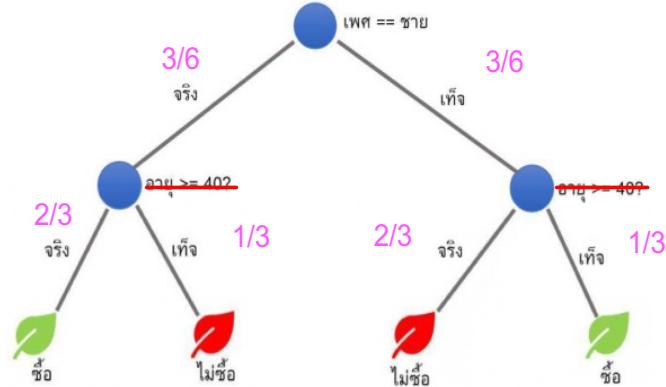
$$Weight_{true} = 3/6 \\ (\text{เพศชาย})$$

$$, \quad Weight_{false} = 3/6 \\ (\text{เพศหญิง})$$

$$Gini_{true} = 1 - [(2/3)^2 + (1/3)^2] = 4/9 \\ \text{เพศชายที่ซื้อ และไม่ซื้อ}$$

$$Gini_{false} = 1 - [(1/3)^2 + (2/3)^2] = 4/9 \\ \text{เพศไม่ใช่ชาย} \\ \text{ที่ซื้อและไม่ซื้อ}$$

$$Gini_{\text{เพศ=ชาย}} = (3/6)(4/9) + (3/6)(4/9) = 4/9 \\ = 0.444$$



$$Gini_{\text{Children}} = 0.444$$

เพศ	อายุ	ซื้อคอม ?
หญิง	40	ไม่ซื้อ
หญิง	50	ไม่ซื้อ
ชาย	20	ไม่ซื้อ
ชาย	40	ซื้อ
ชาย	50	ซื้อ
หญิง	20	ซื้อ

$$Gain(Training Set, \text{เพศ} = \text{ชาย}) = Gini_{\text{Training Set}} - Gini_{\text{เพศ=ชาย}}$$

$$\begin{aligned} \text{ความเป็นระเบียบ} &= 0.5 - (4/9) \\ \text{ของข้อมูล} &= 0.055 \end{aligned}$$

ครั้งถัดไปเราจะเอาค่าอายุแต่ละค่า มาคำนวนหาค่า Gain แล้วเลือกเอาค่าอายุที่ให้ค่า Gain มากที่สุด เป็นหนึดในการแยกกิ่ง เพศชาย/หญิง ไม่จำเป็นต้องใช้อายุที่เท่ากันก็ได้ ขึ้นกับค่า Gain

$$Gain(parent, children) = Gini_{parent} - Gini_{children}$$

$$Gain(Training\ Set, \text{เพศ} = \text{ชาย}) = Gini_{Training\ Set} - Gini_{\text{เพศ}=\text{ชาย}}$$

เพศชาย
อายุ >= 40

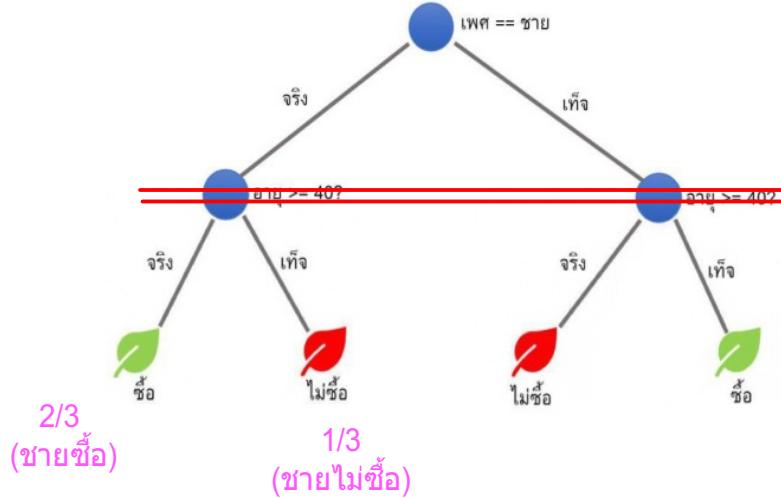
สิ่งที่ต้องหา ความเป็นระเบียบของข้อมูล เทียบระหว่าง ก่อนจัด กับ หลังจัด

ก่อนจัด

$$Gini_{Training\ Set} = 1 - [P(\text{ช้อ)}^2 + P(\text{ไม่ช้อ})^2]$$

เพศชาย = $1 - (2/3)^2 - (1/3)^2 = 0.44$

$$\text{Gini} = 1 - \sum_{k \in K} P(k | \text{parent})^2$$



ເພີ້ນ	ອາຍ	ຈົດຄອມ ?
ໜົມື້ງ	40	ໄນ້ຫົ້ວ້າ
ໜົມື້ງ	50	ໄນ້ຫົ້ວ້າ
ໜ້າຍ	20	ໄປ່ຫົ້ວ້າ
ໜ້າຍ	40	ຫຼື້ອ
ໜ້າຍ	50	ຫຼື້ອ
ໜົມື້ງ	20	ຫຼື້ອ

หลังจัด

$$Gini(\text{children}) = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

$$Gini_{\text{เพศ=ชาย}} = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

อายุ >= 40

$$Weight_{true} = 2/3 \quad (\text{อายุ}) \quad , \quad Weight_{false} = 1/3 \quad (\text{อายุไม่ถึง})$$

$$Gini_{true} = 1 - [(2/2)^2 + (0)^2] = 0$$

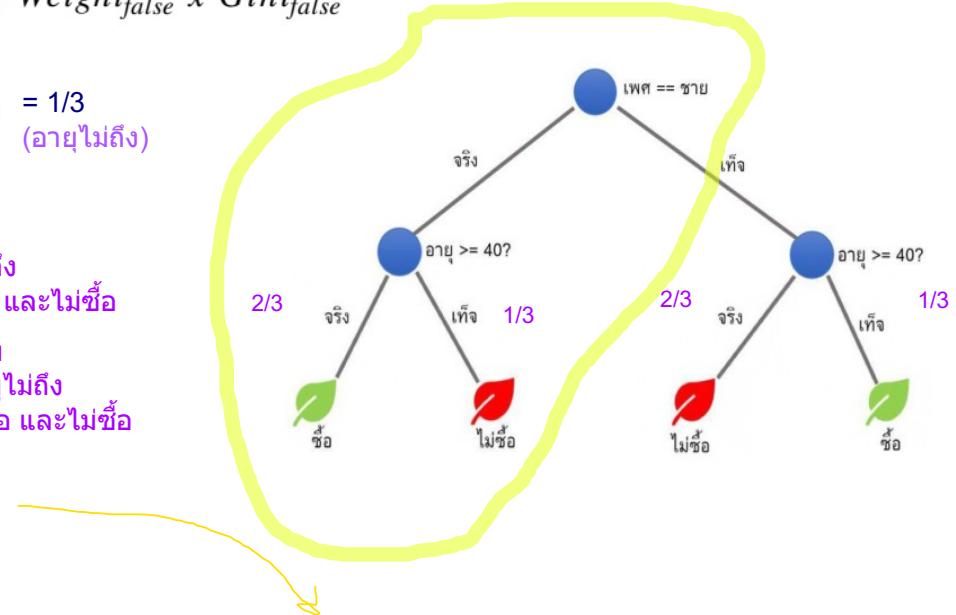
ชาย
อายุถึง
ที่ซื้อ และไม่ซื้อ

ชาย
อายุไม่ถึง
ที่ซื้อ และไม่ซื้อ

$$Gini_{false} = 1 - [(0)^2 + (1/1)^2] = 0$$

$$Gini_{\text{เพศ=ชาย}} = (2/3)(0) + (1/3)(0) = 0$$

เพศ	อายุ	ซื้อคอม ?
หญิง	40	ไม่ซื้อ
หญิง	50	ไม่ซื้อ
ชาย	20	ไม่ซื้อ
ชาย	40	ซื้อ
ชาย	50	ซื้อ
หญิง	20	ซื้อ



$$Gain(\text{Training Set}, \text{เพศ} = \text{ชาย}) = Gini_{\text{Training Set}} - Gini_{\text{เพศ=ชาย}}$$

ความเป็นระเบียบ
ของข้อมูล

$$= 0.44 - (0)$$

$$= 0.44$$

ค่าเท่ากับค่า Gain ก่อนจัด
= แบ่งได้สมบูรณ์แล้ว

ครั้งถัดไปเราจะเอาค่าอายุแต่ละค่า มาคำนวนหาค่า Gain
แล้วเลือกเอาค่าอายุที่ให้ค่า Gain มากที่สุด เป็นหนึ่งในการแยกกิ่ง
เพศชาย/หญิง ไม่จำเป็นต้องใช้อายุที่เท่ากันก็ได้ ขึ้นกับค่า Gain

$$Gain(parent, children) = Gini_{parent} - Gini_{children}$$

$$Gain(Training\ Set, \text{เพศ} = \text{ชาย}) = Gini_{Training\ Set} - Gini_{\text{เพศ}=\text{ชาย}}$$

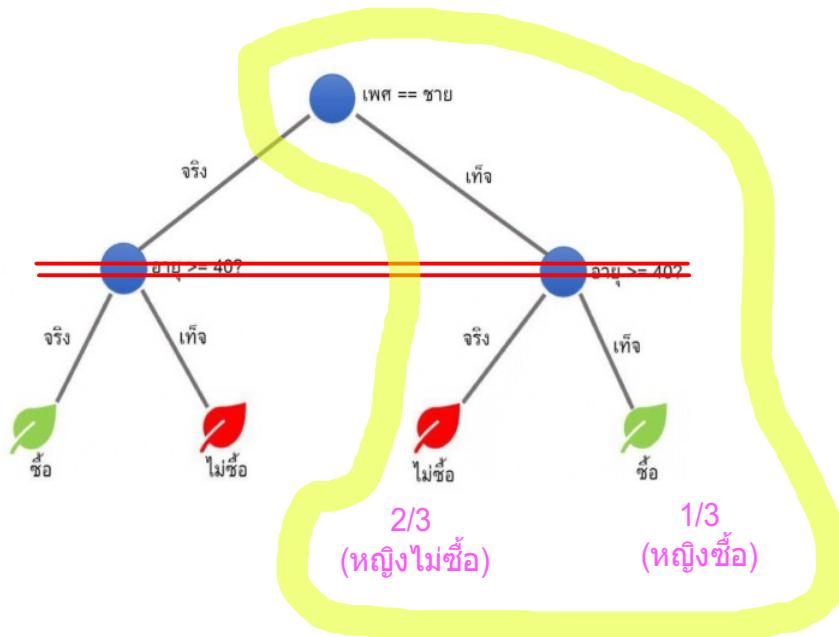
สิ่งที่ต้องหา ความเป็นระเบียนของข้อมูล เทียบระหว่าง ก่อนจัด กับ หลังจัด

ก่อนจัด

$$Gini_{Training\ Set} = 1 - [P(\text{ชื้อ})^2 + P(\text{ไม่ซื้อ})^2]$$

$$\text{เพศหญิง} = 1 - (1/3)^2 - (2/3)^2 = 0.44$$

$$\text{Gini} = 1 - \sum_{k \in K} P(k | \text{parent})^2$$



ເພີ້ນ	ວິເກາະ	ຈົ່ວຍຄວາມ?
ທະບຽງ	40	ໄປເຕັ້ນ
ທະບຽງ	50	ໄປເຕັ້ນ
ຫາຍ	20	ໄປເຕັ້ນ
ຫາຍ	40	ຖື້ນ
ຫາຍ	50	ຖື້ນ
ທະບຽງ	20	ຖື້ນ

หลังจัด

$$Gini(\text{children}) = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

$$Gini_{\text{เพศ}=\text{ชาย}} = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

(ญ) อายุ ≥ 40

$$Weight_{true} = 2/3 \quad (\text{อายุ}) \quad , \quad Weight_{false} = 1/3 \quad (\text{อายุไม่ถึง})$$

$$Gini_{true} = 1 - [(0)^2 + (2/2)^2] = 0$$

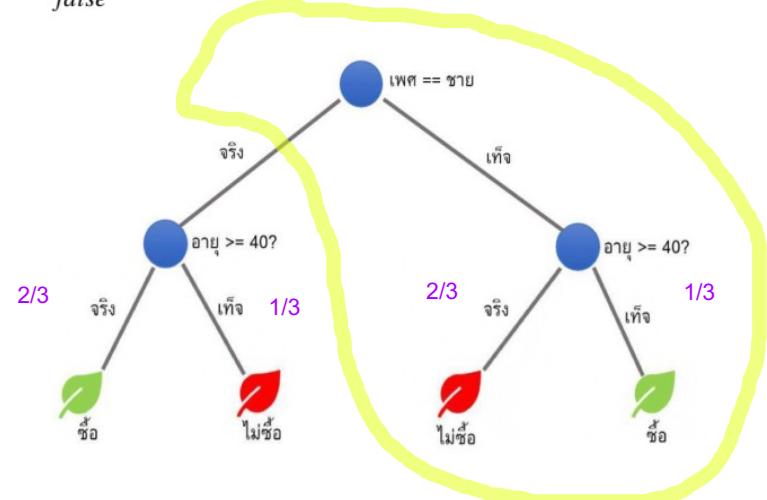
หญิง
อายุถึง
ที่ชื่อ และไม่ชื่อ

$$Gini_{false} = 1 - [(1/1)^2 + (0)^2] = 0$$

หญิง
อายุไม่ถึง
ที่ชื่อ และไม่ชื่อ

$$Gini_{\text{เพศ}=\text{ชาย}} = (2/3)(0) + (1/3)(0) = 0$$

เพศ	อายุ	ชื่อคุณ ?
หญิง	40	ไม่ชื่อ
หญิง	50	ไม่ชื่อ
ชาย	20	ไม่ชื่อ
ชาย	40	ชื่อ
ชาย	50	ชื่อ
หญิง	20	ชื่อ



$$Gain(\text{Training Set}, \text{เพศ} = \text{ชาย}) = Gini_{\text{Training Set}} - Gini_{\text{เพศ}=\text{ชาย}}$$

$$\begin{aligned} \text{ความเป็นระเบียบ} &= 0.44 - (0) \\ \text{ของข้อมูล} &= 0.44 \end{aligned}$$

ค่าเท่ากับค่า Gain ก่อนจัด
= แบ่งได้สมบูรณ์แล้ว

ครั้งถัดไปเราจะเอาค่าอายุแต่ละค่า มาคำนวนหาค่า Gain แล้วเลือกเอาค่าอายุที่ให้ค่า Gain มากที่สุด เป็นหนึ่นในการแยกกิ่ง เพศชาย/หญิง ไม่จำเป็นต้องใช้อายุที่เท่ากันก็ได้ ขึ้นกับค่า Gain

Decision Tree : Code

- I. หาคำถามที่แบ่งต้นไม้ได้ดีที่สุด
- II. ปลูกต้นไม้
- III. เรียนรู้
- IV. พยากรณ์
- V. วัดต้นไม้

I. หาคำถามที่แบ่งต้นไม้ได้ดีที่สุด

- I.1 DT_find_best_question(X, Y, Feature_Name, All_Class)
- I.2 DT_create_Question(X, Feature_Name)
- I.3 DT_compute_Gini(Y, All_Class)
- I.4 DT_find_filter(X, check_type, d, uv)
- I.5 DT_compute_Gini_True_False(Y_True, Y_False, All_Class)
- I.6 DT_compute_weight_true_false(filter_true, filter_false, N)
- I.7 DT_compute_Gini_Children(weight_true, Gini_True, weight_false, Gini_False)
- I.8 DT_compute_Gain(Gini_Parent, Gini_Children)

II. ปลูกต้นไม้

- II.1 DT_grow_tree(best, Y_True, Y_False, All_Class)

III. เรียนรู้

- III.1 DT_fit(X_Train, Y_Train, Feature_Name, All_Class, max_depth, max_majority, min_leaf)

IV. พยากรณ์

- IV.1 DT_predict(X_Test, tree)
- IV.2 DT_recursive_predict(x_test, tree)

V. วัดต้นไม้

- V.1 DT_print_tree(node, prev_id, spacing)

เพศ	อายุ	ซื้อคอม ?
หญิง	40	ไม่ซื้อ
หญิง	50	ไม่ซื้อ
ชาย	20	ไม่ซื้อ
ชาย	40	ซื้อ
ชาย	50	ซื้อ
หญิง	20	ซื้อ

$$X = \begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \\ \text{หญิง} & 20 \end{bmatrix} \quad Y = \begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ไม่ซื้อ} \\ \text{ไม่ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$$

Feature_Name = [เพศ, อายุ]

All_Class = [ซื้อ, ไม่ซื้อ]

I. หาคำถามที่แบ่งตันไม่ได้ดีที่สุด

I.1 DT_find_best_question(X, Y, Feature_Name, All_Class)

```
def DT_find_best_question(X, Y, Feature_Name, All_Class):
    max_Gain = -np.inf
    isComplete = False
    Gini_Parent = DT_compute_Gini(Y, All_Class)
    Question_Dict = DT_create_Question(X, Feature_Name)
    for d, fn in enumerate(Feature_Name):
        N = X.shape[0]
        if fn in Question_Dict:
            unique_value = Question_Dict[fn]['unique_value']
            check_type = Question_Dict[fn]['type_of_feature']
            for i, uv in enumerate(unique_value):
                filter_true, filter_false = DT_find_filter(X, check_type, d, uv)
                X_True = X[filter_true]; Y_True = Y[filter_true];
                X_False = X[filter_false]; Y_False = Y[filter_false];
                weight_true, weight_false = DT_compute_weight_true_false(filter_true, filter_false, N)
                Gini_True, Gini_False = DT_compute_Gini_True_False(Y_True, Y_False, All_Class)
                Gini_Children = DT_compute_Gini_Children(weight_true, Gini_True, weight_false, Gini_False)
                Gain = DT_compute_Gain(Gini_Parent, Gini_Children)
                if Gain >= max_Gain:
                    max_Gain = Gain
                    best = {}
                    best['fn'] = fn
                    best['findex'] = d
                    best['uv'] = uv
                    best['X_True'] = X_True
                    best['Y_True'] = Y_True
                    best['X_False'] = X_False
                    best['Y_False'] = Y_False
                if max_Gain == Gini_Parent:
                    isComplete = True
    return best, isComplete
```

I.1 DT_find_best_question(X, Y, Feature_Name, All_Class)

$$\begin{aligned} \max_Gain &= -\infty \\ \text{Gini_Parent} &= 0.5 \end{aligned}$$

$$\begin{aligned} N &= 6 \\ X_{\text{shape}} &= [6, 2] \end{aligned}$$

```
## fn = ลิสต์
unique_value = [ชาย] = Question_Dict[fn]['unique_value']
    = Question_Dict[เพศ]['unique_value']
check_type = discrete = Question_Dict[fn]['type_of_feature']
    = Question_Dict[เพศ]['type_of_feature']

index filter_true = [2], filter_false = [0]
[4] [5]

weight_true = 0.5, weight_false = 0.5
Gini_True = 0.444, Gini_False = 0.444
```

```
def DT_find_best_question(X, Y, Feature_Name, All_Class):
    max_Gain = -np.inf
    isComplete = False
    Gini_Parent = DT_compute_Gini(Y, All_Class) ← (3)
    Question_Dict = DT_create_Question(X, Feature_Name) ← (2)
    for d, fn in enumerate(Feature_Name):
        N = X.shape[0]
        if fn in Question_Dict:
            unique_value = Question_Dict[fn]['unique_value']
            check_type = Question_Dict[fn]['type_of_feature']
            for i, uv in enumerate(unique_value):
                filter_true, filter_false = DT_find_filter(X, check_type, d, uv) ← (4)
                X_True = X[filter_true]; Y_True = Y[filter_true];
                X_False = X[filter_false]; Y_False = Y[filter_false];
                weight_true, weight_false = DT_compute_weight_true_false(filter_true, filter_false, N)
                Gini_True, Gini_False = DT_compute_Gini_True_False(Y_True, Y_False, All_Class) ← (5)
                Gini_Children = DT_compute_Gini_Children(weight_true, Gini_True, weight_false, Gini_False)
                Gain = DT_compute_Gain(Gini_Parent, Gini_Children) ← (6)
                if Gain >= max_Gain:
                    max_Gain = Gain
                    best = {}
                    best['fn'] = fn
                    best['findex'] = d
                    best['uv'] = uv
                    best['X_True'] = X_True
                    best['Y_True'] = Y_True
                    best['X_False'] = X_False
                    best['Y_False'] = Y_False
                if max_Gain == Gini_Parent:
                    isComplete = True
                    return best, isComplete
    return best, isComplete
```

Question_Dict = {
 'เพศ' : {
 'type_of_feature' : 'discrete',
 'unique_value' : [ชาย]
 },
 'อายุ' : {
 'type_of_feature' : 'continuous',
 'unique_value' : [40]
 }
}

$X_{\text{True}} = \begin{bmatrix} \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \end{bmatrix}$ $Y_{\text{True}} = \begin{bmatrix} \text{สีเสื้อ} \\ \text{สีเสื้อ} \end{bmatrix}$
 $X_{\text{False}} = \begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{หญิง} & 20 \end{bmatrix}$ $Y_{\text{False}} = \begin{bmatrix} \text{ไม่เสื้อ} \\ \text{ไม่เสื้อ} \\ \text{สีเสื้อ} \end{bmatrix}$

Gini_Parent = 0.5, Gini_Children = 0.444

best = {
 'fn' : 'เพศ',
 'findex' : 0,
 'uv' : 'ชาย',
 'X_True' : $\begin{bmatrix} \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \end{bmatrix}$,
 'Y_True' : $\begin{bmatrix} \text{สีเสื้อ} \\ \text{สีเสื้อ} \\ \text{สีเสื้อ} \end{bmatrix}$,

'X_False' : $\begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{หญิง} & 20 \end{bmatrix}$,

'Y_False' : $\begin{bmatrix} \text{ไม่เสื้อ} \\ \text{ไม่เสื้อ} \\ \text{สีเสื้อ} \end{bmatrix}$,
 }

I.2 DT_create_Question(X, Feature_Name)

$$X = \begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \\ \text{หญิง} & 20 \end{bmatrix} \quad \text{Feature_Name} = [\text{เพศ}, \text{อายุ}]$$

d = 0, fn = เพศ

unique_value = [ชาย, หญิง]

Question_Dict = {
 เพศ : {
 'type_of_feature' : discrete,
 'unique_value' : [ชาย]
 }
}

```
def DT_create_Question(X, Feature_Name):  
    Question_Dict = {}  
    for d, fn in enumerate(Feature_Name):  
        unique_value = np.unique(X[:, d])  
        check_type = type(unique_value[0])  
        if check_type == str:  
            if len(unique_value) >= 2:  
                if len(unique_value) == 2:  
                    unique_value = unique_value[:1]  
                q = {}  
                q['type_of_feature'] = 'discrete'  
                q['unique_value'] = unique_value  
                Question_Dict[fn] = q  
        if (check_type == int) or (check_type == float):  
            unique_value = unique_value[1:-1]  
            if len(unique_value) != 0:  
                q = {}  
                q['type_of_feature'] = 'continuous'  
                q['unique_value'] = unique_value  
                Question_Dict[fn] = q  
    return Question_Dict
```

เลือก Indexแรก = ชาย
unique_value = [ชาย]

เราเลือกใช้ค่ากลาง(รองสุดท้าย)

d = 1, fn = อายุ

unique_value = [20, 40, 50]
unique_value = [40]

Question_Dict = {
 เพศ : {
 'type_of_feature' : discrete,
 'unique_value' : [ชาย]
 },
 อายุ : {
 'type_of_feature' : continuous,
 'unique_value' : [40]
 }
}

I.3 DT_compute_Gini(Y, All_Class)

$$X = \begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \\ \text{หญิง} & 20 \end{bmatrix} \quad Y = \begin{bmatrix} \text{ไม่เชื่อ} \\ \text{ไม่เชื่อ} \\ \text{ไม่เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \end{bmatrix} \quad All_Class = [\text{เชื่อ}, \text{ไม่เชื่อ}]$$

ก่อนจัด

$$\text{Gini} = 1 - \sum_{k \in K} P(k | parent)^2$$

```

n_class = 2
def DT_compute_Gini(Y, All_Class):
    n_class = len(All_Class)
    N = Y.shape[0]
    if N == 0:
        Gini = 0
    elif N != 0:
        p = np.zeros([1, n_class])
        for c, _class in enumerate(All_Class):
            p[0, c] = (Y == _class).sum() / N
        Gini = 1 - (p**2).sum()
    return Gini

```

$c = 0, \text{ class} = \text{เชื่อ}$

$$= \begin{bmatrix} \text{ไม่เชื่อ} \\ \text{ไม่เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \end{bmatrix} == \text{เชื่อ}.sum() = (\begin{bmatrix} \text{False} \\ \text{False} \\ \text{True} \\ \text{True} \\ \text{True} \\ \text{True} \end{bmatrix}).sum() = 3/6$$

$p = [0.5 \ 0]$

$c = 1, \text{ class} = \text{ไม่เชื่อ}$

$$= \begin{bmatrix} \text{ไม่เชื่อ} \\ \text{ไม่เชื่อ} \\ \text{ไม่เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \end{bmatrix} == \text{ไม่เชื่อ}.sum() = (\begin{bmatrix} \text{True} \\ \text{True} \\ \text{True} \\ \text{False} \\ \text{False} \\ \text{False} \end{bmatrix}).sum() = 3/6$$

$p = [0.5 \ 0.5]$

$\text{Gini} = 0.5 = 1 - (p**2).sum() = 1 - [0.25 \ 0.25].sum() = 1 - 0.5$

$$Y_{True} = \begin{bmatrix} \text{ไม่เชื่อ} \\ \text{เชื่อ} \\ \text{เชื่อ} \end{bmatrix} \quad Y_{False} = \begin{bmatrix} \text{ไม่เชื่อ} \\ \text{ไม่เชื่อ} \\ \text{เชื่อ} \end{bmatrix}$$

1.4 DT_find_filter(X, check_type, d, uv)

$$X = \begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \\ \text{หญิง} & 20 \end{bmatrix} \quad \begin{array}{l} \text{check_type} = \text{discrete} \\ d = 0 \\ uv = \text{ชาย} \end{array}$$

```
filter_true =
np.argwhere(X[:, 0] == ชาย).ravel()
filter_false =
np.argwhere(X[:, 0] != ชาย).ravel()

def DT_find_filter(X, check_type, d, uv):
    if check_type == 'discrete':
        filter_true = np.argwhere(X[:, d] == uv).ravel()
        filter_false = np.argwhere(X[:, d] != uv).ravel()
    elif check_type == 'continuous':
        filter_true = np.argwhere(X[:, d] >= uv).ravel()
        filter_false = np.argwhere(X[:, d] < uv).ravel()
    return filter_true, filter_false
```

$$\begin{array}{l} \text{np.argwhere}(\begin{bmatrix} \text{หญิง} \\ \text{หญิง} \\ \text{ชาย} \\ \text{ชาย} \\ \text{ชาย} \\ \text{หญิง} \end{bmatrix} \neq \text{ชาย}).ravel() = [0, 1, 2, 3, 4] \\ \text{np.argwhere}(\begin{bmatrix} \text{หญิง} \\ \text{หญิง} \\ \text{ชาย} \\ \text{ชาย} \\ \text{ชาย} \\ \text{หญิง} \end{bmatrix} = \text{ชาย}).ravel() = [5] \end{array}$$

Return

$$filter_true = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \quad filter_false = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}$$

I.5 DT_compute_Gini_True_False(Y_True, Y_False, All_Class)

$$Weight_{true} = \frac{\#(true \cap children)}{\#children}$$

$$Weight_{false} = \frac{\#(false \cap children)}{\#children}$$

$$filter_true = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \quad filter_false = \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix} \quad N = 6$$

count_true = 3

```
def DT_compute_weight_true_false(filter_true, filter_false, N):
    count_true = len(filter_true)
    count_false = N - count_true
    weight_true = count_true/N
    weight_false = count_false/N
    return weight_true, weight_false
```

weight_true = 0.5 = count_true/N = 3/6
 weight_false = 0.5 = count_true/N = 3/6

I.6 DT_compute_weight_true_false(filter_true, filter_false, N) All_Class = [ช้อ, ไม่ช้อ]

หลังจัด

$$Y_{True} = \begin{bmatrix} \text{ไม่ช้อ} \\ \text{ช้อ} \\ \text{ช้อ} \end{bmatrix}$$

$$Y_{False} = \begin{bmatrix} \text{ไม่ช้อ} \\ \text{ไม่ช้อ} \\ \text{ช้อ} \end{bmatrix}$$

$$Gain(parent, children) = Gini_{parent} - Gini_{children}$$

$$\text{Gini} = 1 - \sum_{k \in K} P(k | parent)^2$$

```
def DT_compute_Gini_True_False(Y_True, Y_False, All_Class):
    Gini_True = DT_compute_Gini(Y_True, All_Class) ← (I.3)
    Gini_False = DT_compute_Gini(Y_False, All_Class) ← (I.3)
    return Gini_True, Gini_False
```

I.3 DT_compute_Gini(Y, All_Class)

Return

$$\text{Gini_True} = 0.444, \quad \text{Gini_False} = 0.444$$

I.7 DT_compute_Gini_Children(weight_true, Gini_True, weight_false, Gini_False)

$$Gini(children) = Weight_{true} \times Gini_{true} + Weight_{false} \times Gini_{false}$$

จาก I.5

$$\text{weight_true} = 0.5 = \text{count_true}/N = 3/6$$

$$\text{weight_false} = 0.5 = \text{count_true}/N = 3/6$$

จาก I.6

$$\text{Gini_True} = 0.444$$

$$\text{Gini_False} = 0.444$$

```
def DT_compute_Gini_Children(weight_true, Gini_True, weight_false, Gini_False):
    Gini_Children = weight_true*Gini_True + weight_false*Gini_False
    return Gini_Children
```

Return

$$\text{Gini_Children} = 0.444$$

I.8 DT_compute_Gain(Gini_Parent, Gini_Children)

$$Gain(parent, children) = Gini_{parent} - Gini_{children}$$

Gini_Parent = 0.5, Gini_Children = 0.444

```
def DT_compute_Gain(Gini_Parent, Gini_Children):
    Gain = Gini_Parent - Gini_Children
    return Gain
```

Return

Gain = 0.055,

II. ปัญกดันไม้

II.1 DT_grow_tree(best, Y_True, Y_False, All_Class)

$$Y_{True} = \begin{bmatrix} \text{ไม่ชื่อ} \\ \text{ชื่อ} \\ \text{ชื่อ} \end{bmatrix}$$

$$Y_{False} = \begin{bmatrix} \text{ไม่ชื่อ} \\ \text{ชื่อ} \\ \text{ชื่อ} \end{bmatrix}$$

$$All_Class = [\text{ชื่อ}, \text{ไม่ชื่อ}]$$

```
def DT_grow_tree(best, Y_True, Y_False, All_Class):
    fn = best['fn']
    findex = best['findex']
    uv = best['uv']
    if type(uv) == str:
        type_of_feature = 'discrete'
    elif (type(uv) == float) or (type(uv) == int):
        type_of_feature = 'continuous'
```

fn = เพศ
finDEX = 0

uv = ชาย

```
Question = {}
Question['fn'] = fn
Question['findex'] = finDEX; Question['uv'] = uv
Question['type_of_feature'] = type_of_feature
```

Leaf_Node_True, Vote_True, Max_Vote_True, Max_Class_True = DT_Leaf(Y_True, All_Class) ← (II.2)

```
True_Branch = {}
True_Branch['Leaf_Node'] = Leaf_Node_True
True_Branch['Vote'] = Vote_True
True_Branch['Predict'] = Max_Class_True
True_Branch['Percent'] = Max_Vote_True
```

Leaf_Node_False, Vote_False, Max_Vote_False, Max_Class_False = DT_Leaf(Y_False, All_Class) ← (II.2)

```
False_Branch = {}
False_Branch['Leaf_Node'] = Leaf_Node_False
False_Branch['Vote'] = Vote_False
False_Branch['Predict'] = Max_Class_False
False_Branch['Percent'] = Max_Vote_False
```

```
Decision_Node = {}
Decision_Node['Question'] = Question
Decision_Node['True_Branch'] = True_Branch
Decision_Node['False_Branch'] = False_Branch
```

return Decision_Node, Max_Vote_True, Max_Vote_False

```
best = {
    'fn' : 'เพศ',
    'findex' : 0,
    'uv' : 'ชาย',
    'x' : [ชาย 20,
           ชาย 40,
           ชาย 50],
    'Y_True' : [ชาย,
                ชื่อ,
                ชื่อ],
    'X_False' : [หญิง 40,
                  หญิง 50,
                  หญิง 20],
    'Y_False' : [ไม่ชื่อ,
                  ไม่ชื่อ,
                  ชื่อ],
}
```

```
Question['type_of_feature'] = type_of_feature
```

```
Question = {'fn' : เพศ, 'findex' : 0, 'uv' : ชาย, 'type_of_feature' : 'discrete'}
```

$$\text{### } Y_{\text{True}} = \begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \end{bmatrix}, \quad All_{\text{Class}} = [\text{ซื้อ}, \text{ไม่ซื้อ}]$$

```
Leaf_Node_True, Vote_True, Max_Vote_True, Max_Class_True = DF_Leaf(Y_True, All_Class)
```

```
### Leaf_Node_True = {'ซื้อ' : 2, 'ไม่ซื้อ' : 1}
```

```
### Vote_True = {'ซื้อ' : 66.67, 'ไม่ซื้อ' : 33.33}
```

```
### Max_Vote_True = 66.67
```

```
### Max_Class_True = 'ซื้อ'
```

```
True_Branch = {}
```

```
True_Branch['Leaf_Node'] = Leaf_Node_True
```

```
True_Branch['Vote'] = Vote_True
```

```
True_Branch['Predict'] = Max_Class_True
```

```
True_Branch['Percent'] = Max_Vote_True
```

```
### 'True_Branch' : {
```

```
    'Leaf_Node' : {'ซื้อ' : 2, 'ไม่ซื้อ' : 1},  
    'Vote' : {'ซื้อ' : 66.67, 'ไม่ซื้อ' : 33.33},  
    'Predict' : 'ซื้อ',  
    'Percent' : 66.67
```

```
}
```

```
Question = {}  
Question['fn'] = fn  
Question['findex'] = findex; Question['uv'] = uv  
Question['type_of_feature'] = type_of_feature  
  
Leaf_Node_True, Vote_True, Max_Vote_True, Max_Class_True = DT_Leaf(Y_True, All_Class)  
True_Branch = {}  
True_Branch['Leaf_Node'] = Leaf_Node_True  
True_Branch['Vote'] = Vote_True  
True_Branch['Predict'] = Max_Class_True  
True_Branch['Percent'] = Max_Vote_True
```

```


$$Y\_True = \begin{bmatrix} \text{ไม่ชื่อ} \\ \text{ไม่ชื่อ} \\ \text{ชื่อ} \end{bmatrix}, \quad All\_Class = [\text{ชื่อ}, \text{ไม่ชื่อ}]$$

Leaf_Node_False, Vote_False, Max_Vote_False, Max_Class_False = DF_Leaf(Y_False, All_Class)
### Leaf_Node_False = {'ชื่อ' : 1, 'ไม่ชื่อ' : 2}
### Vote_False = {'ชื่อ' : 33.33, 'ไม่ชื่อ' : 66.67}
### Max_Vote_False = 66.67
### Max_Class_False = 'ไม่ชื่อ'

False_Branch = {}
False_Branch['Leaf_Node'] = Leaf_Node_False
False_Branch['Vote'] = Vote_False
False_Branch['Predict'] = Max_Class_False
False_Branch['Percent'] = Max_Vote_False
### 'False_Branch' : {
    'Leaf_Node' : {'ชื่อ' : 1, 'ไม่ชื่อ' : 2},
    'Vote' : {'ชื่อ' : 33.33, 'ไม่ชื่อ' : 66.67},
    'Predict' : 'ไม่ชื่อ',
    'Percent' : 66.67
}

Decision_Node = {}
Decision_Node['Question'] = Question
Decision_Node['True_Branch']= True_Branch
Decision_Node['False_Branch'] = False_Branch
### Decision_Node = {
    'Question' : {
        'fn' : 'เพศ',
        'findex' : 0,
        'uv' : 'ชาย',
        'type_of_feature' : 'discrete'
    },
    'True_Branch' : {
        'Leaf_Node' : {'ชื่อ' : 2, 'ไม่ชื่อ' : 1},
        'Vote' : {'ชื่อ' : 66.67, 'ไม่ชื่อ' : 33.33},
        'Predict' : 'ชื่อ',
        'Percent' : 66.67
    },
    'False_Branch' : {
        'Leaf_Node' : {'ชื่อ' : 1, 'ไม่ชื่อ' : 2},
        'Vote' : {'ชื่อ' : 33.33, 'ไม่ชื่อ' : 66.67},
        'Predict' : 'ไม่ชื่อ',
        'Percent' : 66.67
    }
}

Question = {}
Question['fn'] = fn
Question['findex'] = findex; Question['uv'] = uv
Question['type_of_feature'] = type_of_feature

Leaf_Node_True, Vote_True, Max_Vote_True, Max_Class_True = DT_Leaf(Y_True, All_Class) ←
True_Branch = {}
True_Branch['Leaf_Node'] = Leaf_Node_True
True_Branch['Vote'] = Vote_True
True_Branch['Predict'] = Max_Class_True
True_Branch['Percent'] = Max_Vote_True

Leaf_Node_False, Vote_False, Max_Vote_False, Max_Class_False = DT_Leaf(Y_False, All_Class)
False_Branch = {}
False_Branch['Leaf_Node'] = Leaf_Node_False
False_Branch['Vote'] = Vote_False
False_Branch['Predict'] = Max_Class_False
False_Branch['Percent'] = Max_Vote_False

Decision_Node = {}
Decision_Node[ 'Question' ] = Question
Decision_Node[ 'True_Branch' ] = True_Branch
Decision_Node[ 'False_Branch' ] = False_Branch

return Decision_Node, Max_Vote_True, Max_Vote_False

```

II.2 DT_Leaf(Y, All_Class)

$$Y = \begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \end{bmatrix} \quad All_Class = [\text{ซื้อ}, \text{ไม่ซื้อ}]$$

```
def DT_Leaf(Y, All_Class):
    N = Y.shape[0]
    leaf_node = {}
    vote = {}
    max_vote = -np.inf
    for c, _class in enumerate(All_Class):
        length = (Y == _class).sum()
        percent_vote = 100*length/N
        if max_vote < percent_vote:
            max_vote = percent_vote
            max_class = _class
        leaf_node[_class] = length
        vote[_class] = percent_vote
    return leaf_node, vote, max_vote, max_class
```

```
### All_Class = [ช้อ, ไม่ช้อ]
for c, _class in enumerate(All_Class):
```

```
c = 0, _class = ช้อ
```

$$\text{### } Y = \begin{bmatrix} \text{ไม่ช้อ} \\ \text{ช้อ} \\ \text{ช้อ} \end{bmatrix}$$

$$\text{length} = 2 = (Y == \text{_class}).\text{sum}() = (\begin{bmatrix} \text{ไม่ช้อ} \\ \text{ช้อ} \\ \text{ช้อ} \end{bmatrix} == \text{ช้อ}).\text{sum}() = \begin{bmatrix} \text{False} \\ \text{True} \\ \text{True} \end{bmatrix}.\text{sum}()$$

```
### N = 3
```

$$\text{percent_vote} = 66.67 = 100 * \text{length}/N = 100 * 2/3 = 200/3$$

```
### max_vote = -∞
```

```
if max_vote < percent_vote:
```

```
    ### percent_vote = 66.67
```

```
    max_vote = percent_vote
```

```
    ### max_vote = 66.67
```

```
    ### _class = ช้อ
```

```
    max_class = _class
```

```
    ### max_class = ช้อ
```

```
### length = 2
```

```
leaf_node[_class] = length
```

```
### leaf_node = {ช้อ : 2}
```

```
### percent_vote = 66.67
```

```
vote[_class] = percent_vote
```

```
### vote = {ช้อ : 66.67}
```

```
return leaf_node, vote, max_vote, max_class
```

```
def DT_Leaf(Y, All_Class):
    N = Y.shape[0]
    leaf_node = {}
    vote = {}
    max_vote = -np.inf
    for c, _class in enumerate(All_Class):
        length = (Y == _class).sum()
        percent_vote = 100 * length/N
        if max_vote < percent_vote:
            max_vote = percent_vote
            max_class = _class
        leaf_node[_class] = length
        vote[_class] = percent_vote
    return leaf_node, vote, max_vote, max_class
```

III. เรียนรู้

III.1 DT_fit(X_Train, Y_Train, Feature_Name, All_Class, max_depth, max_majority, min_leaf)

$$X_Train = \begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \\ \text{หญิง} & 20 \end{bmatrix} \quad Y_Train = \begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ไม่ซื้อ} \\ \text{ไม่ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$$

Feature_Name = [เพศ, อายุ]
All_Class = [ซื้อ, ไม่ซื้อ]

ขั้นตอนไม้ Loop

```
def DT_fit(X_Train, Y_Train, Feature_Name, All_Class, max_depth=np.inf, depth=1, max_majority=np.inf, min_leaf=-np.inf):
    best, isComplete = DT_find_best_question(X_Train, Y_Train, Feature_Name, All_Class) ← (I.1)
    Y_True = best['Y_True']
    Y_False = best['Y_False']
    Decision_Node, Max_Vote_True, Max_Vote_False = DT_grow_tree(best, Y_True, Y_False, All_Class) ← (II.1)

    if max_depth == depth:
        return Decision_Node
    if isComplete == True:
        return Decision_Node

    if Max_Vote_True < max_majority: (III.1)
        if len(Y_True) > min_leaf:
            X_True = best['X_True']
            Decision_Node['True_Branch']['Decision_Node'] = DT_fit(X_True, Y_True, Feature_Name, All_Class, max_depth=max_depth,
                                                               depth=depth+1, max_majority=max_majority, min_leaf=min_leaf)
    if Max_Vote_False < max_majority:
        if len(Y_False) > min_leaf: ← (III.1)
            X_False = best['X_False']
            Decision_Node['False_Branch']['Decision_Node'] = DT_fit(X_False, Y_False, Feature_Name, All_Class, max_depth=max_depth,
                                                               depth=depth+1, max_majority=max_majority, min_leaf=min_leaf)
    return Decision_Node
```

best, isComplete = DT_find_best_question(X_Train, Y_Train, Feature_Name, All_Class) (I.1)

isComplete = False

best = {

 'fn' : 'เพศ',

 'findex' : 0,

 'uv' : 'ชาย',

 'X_True' : $\begin{bmatrix} \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \end{bmatrix}$,

 'Y_True' : $\begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$,

 'X_False' : $\begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{หญิง} & 20 \end{bmatrix}$,

 'Y_False' : $\begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ไม่ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$

}

Y_True = $\begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$

Y_False = $\begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ไม่ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$

(II.1)

Decision_Node, Max_Vote_True, Max_Vote_False = DT_grow_tree(best, Y_True, Y_False, All_Class)

```
### Decision_Node = {
    'Question' : {
        'fn' : 'เพศ',
        'findex' : 0,
        'uv' : 'ชาย',
        'type_of_feature' : 'discrete'
    },
    'True_Branch' : {
        'Leaf_Node' : {'ซื้อ' : 2, 'ไม่ซื้อ' : 1},
        'Vote' : {'ซื้อ' : 66.67, 'ไม่ซื้อ' : 33.33},
        'Predict' : 'ซื้อ',
        'Percent' : 66.67
    },
    'False_Branch' : {
        'Leaf_Node' : {'ซื้อ' : 1, 'ไม่ซื้อ' : 2},
        'Vote' : {'ซื้อ' : 33.33, 'ไม่ซื้อ' : 66.67},
        'Predict' : 'ไม่ซื้อ',
        'Percent' : 66.67
    }
}
```

Max_Vote_True = 66.67, Max_Vote_False = 66.67

```
if max_depth == depth:  
    return Decision_Node
```

isComplete = False

```
if isComplete == True:  
    return Decision_Node
```

Max_Vote_True = 66.67

```
if Max_Vote_True < max_majority:
```

Y_True = $\begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$

```
if len(Y_True) > min_leaf:
```

X_True = $\begin{bmatrix} \text{ชาย} & 20 \\ \text{ชาย} & 40 \\ \text{ชาย} & 50 \end{bmatrix} = \text{best['X_True']}$

```
Decision_Node['True_Branch']['Decision_Node'] = DT_fit(X_True, Y_True, ...)
```

!!! Recursive !!!

Y_False = $\begin{bmatrix} \text{ไม่ซื้อ} \\ \text{ไม่ซื้อ} \\ \text{ซื้อ} \end{bmatrix}$

```
if len(Y_False) > min_leaf:
```

X_False = $\begin{bmatrix} \text{หญิง} & 40 \\ \text{หญิง} & 50 \\ \text{หญิง} & 20 \end{bmatrix} = \text{best['X_False']}$

```
Decision_Node['False_Branch']['Decision_Node'] = DT_fit(X_False, Y_False, ...)
```

!!! Recursive !!!

```
return Decision_Node
```

```
if Max_Vote_True < max_majority:  
    if len(Y_True) > min_leaf:  
        X_True = best['X_True']  
        Decision_Node['True_Branch']['Decision_Node'] = DT_fit(X_True, Y_True, Feature_Name, All_Class, max_depth=max_depth,  
        depth=depth+1, max_majority=max_majority, min_leaf=min_leaf)  
    if Max_Vote_False < max_majority:  
        if len(Y_False) > min_leaf:  
            X_False = best['X_False']  
            Decision_Node['False_Branch']['Decision_Node'] = DT_fit(X_False, Y_False, Feature_Name, All_Class, max_depth=max_depth,  
            depth=depth+1, max_majority=max_majority, min_leaf=min_leaf)  
return Decision_Node
```

IV. พยากรณ์ (Test)

IV.1 DT_predict(X_Test, tree)

```
def DT_predict(X_Test, tree):
    Yhat = []
    for x_test in X_Test:
        yhat = DT_recursive_predict(x_test, tree)
        Yhat.append(yhat)
    return np.array(Yhat)
```

(IV.2)

```
tree = Decision_Node =
{
    'Question': {
        'fn' : 'เพศ',
        'findex' : 0,
        'uv' : 'ชาย',
        'type_of_feature' : 'discrete'
    },
    'True_Branch': {
        'Leaf_Node' : {'ชื่อ' : 2, 'ไม่ชื่อ' : 1},
        'Vote' : {'ชื่อ' : 66.67, 'ไม่ชื่อ' : 33.33},
        'Predict' : 'ชื่อ',
        'Percent' : 66.67
    },
    'False_Branch' : {
        'Leaf_Node' : {'ชื่อ' : 1, 'ไม่ชื่อ' : 2},
        'Vote' : {'ชื่อ' : 33.33, 'ไม่ชื่อ' : 66.67},
        'Predict' : 'ไม่ชื่อ',
        'Percent' : 66.67
    }
}
```

Return

```
Yhat_Test = DT_predict(X_Test, tree)
print(Yhat_Test)
[['ไม่ชื่อ' '100.0']]
```

```
X_Test = np.array([['หญิง', 90]], dtype='object')
print(X_Test)
[['หญิง' 90]]
```

IV.2 DT_recursive_predict(x_test, tree)

```
tree = Decision_Node = {
    'Question': {
        'fn': 'เพศ',
        'findex': 0,
        'uv': 'ชาย',
        'type_of_feature': 'discrete'
    },
    'True_Branch': {
        'Leaf_Node': {'ชื่อ': 2, 'นามสกุล': 1},
        'Vote': {'ชื่อ': 66.67, 'นามสกุล': 33.33},
        'Predict': 'ชื่อ',
        'Percent': 66.67
    },
    'False_Branch': {
        'Leaf_Node': {'ชื่อ': 1, 'นามสกุล': 2},
        'Vote': {'ชื่อ': 33.33, 'นามสกุล': 66.67},
        'Predict': 'นามสกุล',
        'Percent': 66.67
    }
}
```

```
X_Test = np.array([[['หญิง', 90]], dtype='object')
print(X_Test)
```

```
[['หญิง' 90]]
```

```
Yhat_Test = DT_predict(X_Test, tree)
print(Yhat_Test)
```

```
[['ไม่ชอบ' '100.0']]
```

```
def DT_recursive_predict(x_test, tree):
    fn = tree['Question']['fn']
    findex = tree['Question']['findex']
    value = tree['Question']['uv']
    type_of_feature = tree['Question']['type_of_feature']

    if type_of_feature == 'discrete':
        if x_test[findex] == value:
            if 'Decision_Node' not in tree['True_Branch']:
                predict = tree['True_Branch']['Predict']
                percent = tree['True_Branch']['Percent']
                yhat = np.array([predict, percent])
                return yhat
            elif 'Decision_Node' in tree['True_Branch']:
                tree = tree['True_Branch']['Decision_Node']
                yhat = DT_recursive_predict(x_test, tree)
                return yhat
        elif x_test[findex] != value:
            if 'Decision_Node' not in tree['False_Branch']:
                predict = tree['False_Branch']['Predict']
                percent = tree['False_Branch']['Percent']
                yhat = np.array([predict, percent])
                return yhat
            elif 'Decision_Node' in tree['False_Branch']:
                tree = tree['False_Branch']['Decision_Node']
                yhat = DT_recursive_predict(x_test, tree)
                return yhat
    if type_of_feature == 'continuous':
        if x_test[findex] >= value:
            if 'Decision_Node' not in tree['True_Branch']:
                predict = tree['True_Branch']['Predict']
                percent = tree['True_Branch']['Percent']
                yhat = np.array([predict, percent])
                return yhat
            elif 'Decision_Node' in tree['True_Branch']:
                tree = tree['True_Branch']['Decision_Node']
                yhat = DT_recursive_predict(x_test, tree)
                return yhat
        elif x_test[findex] < value:
            if 'Decision_Node' not in tree['False_Branch']:
                predict = tree['False_Branch']['Predict']
                percent = tree['False_Branch']['Percent']
                yhat = np.array([predict, percent])
                return yhat
            elif 'Decision_Node' in tree['False_Branch']:
                tree = tree['False_Branch']['Decision_Node']
                yhat = DT_recursive_predict(x_test, tree)
                return yhat
```

Error เป็นเปอร์เซ็นต์

```
def find_error_classification(Y, Yhat):
    N = Y.shape[0]
    error = (100/N)*(Y != Yhat).sum()
    return error
```

```
X_Test = np.array([['หญิง', 90]], dtype='object')
Y_Test = np.array([['ไม่ซื้อ']], dtype='object')
print(X_Test)

[['หญิง' 90]]
```

```
Yhat_Test = DT_predict(X_Test, tree)
print(Yhat_Test)

[['ไม่ซื้อ' '100.0']]
```

```
error_Test = find_error_classification(Y_Test, Yhat_Test[:, 0:1])
print(error_Test)
```

```
0.0
```

V. جادต์น์ไม้

V.1 DT_print_tree(node, prev_id, spacing)

```
def DT_print_tree(node, prev_id, spacing=""):
    global curr_id
    curr_id += 1

    if 'Question' not in node:
        node_predictions = "{} : {}".format(node['Predict'], round(node['Percent'], 2))
        g.node(str(prev_id), label=str(node_predictions))
        return

    node_question = "{} {} {}".format(node['Question']['fn'], '=' if isinstance(node['Question']['uv'], str) else '>', node['Question']['uv'] if isinstance(node['Question']['uv'], str) else round(node['Question']['uv'], 6))
    g.node(str(prev_id), label=str(node_question))

    true_node = node['True_Branch']['Decision_Node'] if 'Decision_Node' in node['True_Branch'] else node['True_Branch']
    V.append((prev_id, curr_id))
    DT_print_tree(true_node, curr_id, spacing + "  ")

    false_node = node['False_Branch']['Decision_Node'] if 'Decision_Node' in node['False_Branch'] else node['False_Branch']
    V.append((prev_id, curr_id))
    DT_print_tree(false_node, curr_id, spacing + "  ")
```

```

g = Graph('G')
g.attr('node', shape='box', style='filled', color='darkseagreen1')
curr_id = 1
V = []
DT_print_tree(tree, curr_id)
g

```

