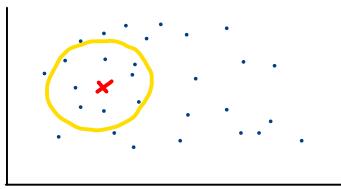
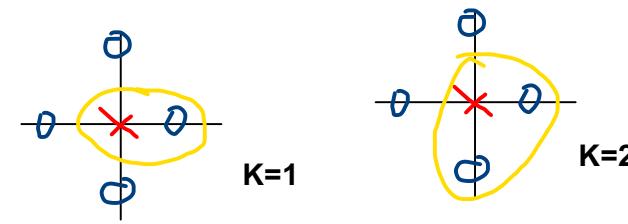


K-Nearest Neighbor (KNN) ->Classification/Regression

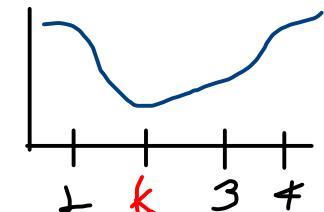


Model คณิตฯ จัดกลุ่มตามความคล้าย ลักษณะที่ใช้จำแนก
เทียบกับจุด(K) -> ความใกล้ของจุดมาร์ก(X) ตามจำนวน K

ระยะห่างแกน x,y ต่างกันมาก
ต้องทำ normalization



ใช้ K ตามใจชอบ
แล้วเลือกหา K ที่ Error
ดีสุด



min-max normalization

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

$$X1 = \begin{bmatrix} x_1^1 \\ x_2^1 \\ \vdots \\ x_N^1 \end{bmatrix}, X2 = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \vdots \\ x_N^2 \end{bmatrix}, \dots, XD = \begin{bmatrix} x_1^D \\ x_2^D \\ \vdots \\ x_N^D \end{bmatrix}$$

$$X = [X1 \quad X2 \quad \dots \quad XD]$$

$$X1_Norm = \begin{bmatrix} \frac{x_1^1 - X1.\min()}{X1.\max() - X1.\min()} \\ \frac{x_2^1 - X1.\min()}{X1.\max() - X1.\min()} \\ \vdots \\ \frac{x_N^1 - X1.\min()}{X1.\max() - X1.\min()} \end{bmatrix} = \frac{X1 - X1.\min()}{X1.\max() - X1.\min()}$$

$$X2_Norm = \begin{bmatrix} \frac{x_1^2 - X2.\min()}{X2.\max() - X2.\min()} \\ \frac{x_2^2 - X2.\min()}{X2.\max() - X2.\min()} \\ \vdots \\ \frac{x_N^2 - X2.\min()}{X2.\max() - X2.\min()} \end{bmatrix} = \frac{X2 - X2.\min()}{X2.\max() - X2.\min()}$$

•
•

$$XD_Norm = \begin{bmatrix} \frac{x_1^D - XD.\min()}{XD.\max() - XD.\min()} \\ \frac{x_2^D - XD.\min()}{XD.\max() - XD.\min()} \\ \vdots \\ \frac{x_N^D - XD.\min()}{XD.\max() - XD.\min()} \end{bmatrix} = \frac{XD - XD.\min()}{XD.\max() - XD.\min()}$$

X1	X2	Class
1	0	Blue
-3	0	Blue
0	-4	Blue
0	2	Orange
5	0	Orange
-2	0	Blue
0	3	Orange

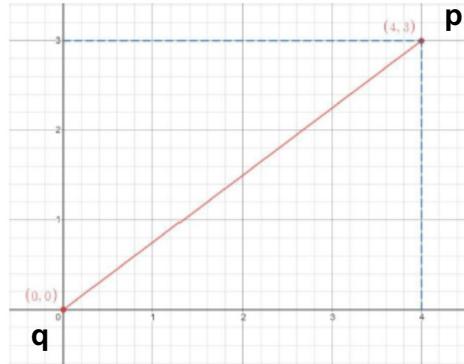


- Normalize Feature ของ Training Set และ Validation Set
- วัดระยะห่างระหว่างข้อมูลแต่ละตัวใน Validation Set กับข้อมูลทุกตัวใน Training Set => Distance Function (Euclidean Distance)
- เรียงลำดับระยะจากขั้นตอนที่ 2 จากน้อยไปมาก
- หาผลลัพธ์จากขั้นตอนที่ 3 ที่ใกล้ที่สุด K ค่า และ Majority Vote
- หาก K ที่ทำให้ Error บน Validation Set ต่ำสุด

2. วัดระยะห่างระหว่างข้อมูลแต่ละตัวใน Validation Set กับข้อมูลทุกตัวใน Training Set => Distance Function (Euclidean Distance)

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_D - q_D)^2}$$

$$= \sqrt{\sum_{i=1}^D (p_i - q_i)^2}$$

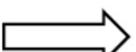


$$\begin{aligned}
 d(p, q) &= \sqrt{(4 - 0)^2 + (3 - 0)^2} \\
 &= \sqrt{4^2 + 3^2} = \sqrt{16 + 9} \\
 &= \sqrt{25} = 5
 \end{aligned}$$

EX

$$\begin{bmatrix} 50 & 1.60 \\ 90 & 1.80 \\ 70 & 1.70 \end{bmatrix}$$

X_Valid

X_Train

$$\begin{bmatrix} 60 & 1.50 \\ 70 & 1.80 \\ 80 & 2.00 \\ 90 & 1.90 \\ 100 & 1.70 \\ 50 & 1.60 \end{bmatrix}$$

X_Train

X_Valid_Norm

$$\begin{bmatrix} 0 & 0.2 \\ 0.8 & 0.6 \\ 0.4 & 0.4 \end{bmatrix}$$

X_Train_Norm

$$\begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix}$$

$$X = \begin{bmatrix} 60 & 1.50 \\ 70 & 1.80 \\ 80 & 2.00 \\ 90 & 1.90 \\ 100 & 1.70 \\ 50 & 1.60 \end{bmatrix}$$

X_Train

$$X1_Norm = \begin{bmatrix} 60 \\ 70 \\ 80 \\ 90 \\ 100 \\ 50 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.8 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 60 - 50 \\ 100 - 50 \\ 70 - 50 \\ 100 - 50 \\ 80 - 50 \\ 100 - 50 \end{bmatrix}$$

$$= \frac{X1 - X1.\min()}{X1.\max() - X1.\min()}$$

$$= \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix}$$

X_Train_Norm

$$X2_Norm = \begin{bmatrix} 1.5 \\ 1.8 \\ 2.0 \\ 1.9 \\ 1.7 \\ 1.6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.6 \\ 1 \\ 0.8 \\ 0.4 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 1.5 - 1.5 \\ 2.0 - 1.5 \\ 1.8 - 1.5 \\ 2.0 - 1.5 \\ 2.0 - 1.5 \\ 2.0 - 1.5 \end{bmatrix}$$

$$= \frac{X2 - X2.\min()}{X2.\max() - X2.\min()}$$

จัดรูปแบบด้วยประกอบ การหา min-max normalization
เพื่อให้ง่ายต่อการเขียน code python

$$X_{Norm} = \begin{bmatrix} 60 - 50 & 1.5 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 70 - 50 & 1.8 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 80 - 50 & 2.0 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 90 - 50 & 1.9 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 100 - 50 & 1.7 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 50 - 50 & 1.6 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 100 - 50 & 2.0 - 1.5 \\ 1 & 1 \\ 100 - 50 & 2.0 - 1.5 \\ 1 & 1 \\ 100 - 50 & 2.0 - 1.5 \\ 1 & 1 \\ 100 - 50 & 2.0 - 1.5 \\ 1 & 1 \\ 100 - 50 & 2.0 - 1.5 \end{bmatrix} * \left(\begin{bmatrix} 60 & 1.50 \\ 70 & 1.80 \\ 80 & 2.00 \\ 90 & 1.90 \\ 100 & 1.70 \\ 50 & 1.60 \end{bmatrix} - \begin{bmatrix} 50 & 1.50 \\ 50 & 1.50 \\ 50 & 1.50 \\ 50 & 1.50 \\ 50 & 1.50 \\ 50 & 1.50 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 60 - 50 & 1.5 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 70 - 50 & 1.8 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 80 - 50 & 2.0 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 90 - 50 & 1.9 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 100 - 50 & 1.7 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 50 - 50 & 1.6 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \end{bmatrix} = \left[\frac{1}{100 - 50} \quad \frac{1}{2.0 - 1.5} \right] * \left(\begin{bmatrix} 60 & 1.5 \\ 70 & 1.8 \\ 80 & 2.0 \\ 90 & 1.9 \\ 100 & 1.7 \\ 50 & 1.6 \end{bmatrix} - [50 \quad 1.5] \right)$$

$$X_{Norm} = \begin{bmatrix} 60 - 50 & 1.5 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 70 - 50 & 1.8 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 80 - 50 & 2.0 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 90 - 50 & 1.9 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 100 - 50 & 1.7 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 50 - 50 & 1.6 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \end{bmatrix} = \left(\begin{bmatrix} 60 & 1.5 \\ 70 & 1.8 \\ 80 & 2.0 \\ 90 & 1.9 \\ 100 & 1.7 \\ 50 & 1.6 \end{bmatrix} - [50 \quad 1.5] \right) / [100 - 50 \quad 2.0 - 1.5]$$

$$\begin{bmatrix} 60 - 50 & 1.5 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 70 - 50 & 1.8 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 80 - 50 & 2.0 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 90 - 50 & 1.9 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 100 - 50 & 1.7 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \\ 50 - 50 & 1.6 - 1.5 \\ 100 - 50 & 2.0 - 1.5 \end{bmatrix} = \left(\begin{bmatrix} 60 & 1.5 \\ 70 & 1.8 \\ 80 & 2.0 \\ 90 & 1.9 \\ 100 & 1.7 \\ 50 & 1.6 \end{bmatrix} - [50 \quad 1.5] \right) / ([100 \quad 2.0] - [50 \quad 1.5])$$

$$X = \begin{bmatrix} 60 & 1.50 \\ 70 & 1.80 \\ 80 & 2.00 \\ 90 & 1.90 \\ 100 & 1.70 \\ 50 & 1.60 \end{bmatrix}$$

$$_max = [100 \quad 2.0]$$

$$_min = [50 \quad 1.5]$$

```
def minmaxNorm(Data, _min, _max):
    Data_Norm = (Data - _min)/(_max - _min)
    return Data_Norm
```

$$X_{Norm} = (X - _min)/(_max - _min)$$

หาค่าน้อยสุดของแต่ละ column

```
def min4norm(Data):
    _min = Data.min(axis=0, keepdims = True)
    return _min
```

รักษามิติเดิม

$$X = \begin{bmatrix} 60 & 1.50 \\ 70 & 1.80 \\ 80 & 2.00 \\ 90 & 1.90 \\ 100 & 1.70 \\ 50 & 1.60 \end{bmatrix} \implies _min = [50 \quad 1.50]$$

หาค่ามากสุดของแต่ละ column

```
def max4norm(Data):
    _max = Data.max(axis=0, keepdims = True)
    return _max
```

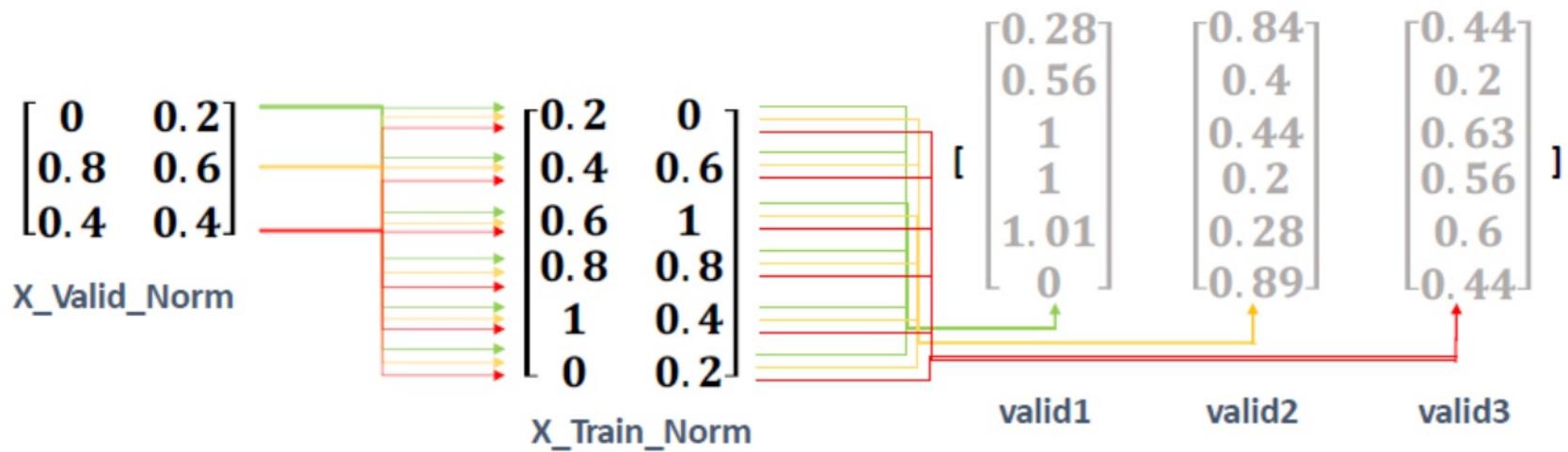
$$X = \begin{bmatrix} 60 & 1.50 \\ 70 & 1.80 \\ 80 & 2.00 \\ 90 & 1.90 \\ 100 & 1.70 \\ 50 & 1.60 \end{bmatrix} \implies _max = [100 \quad 2.00]$$

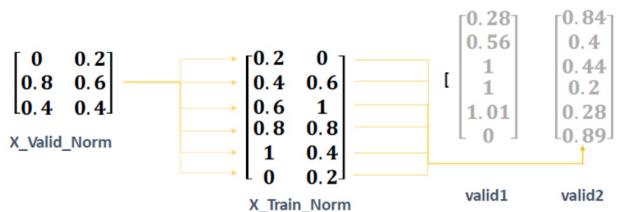
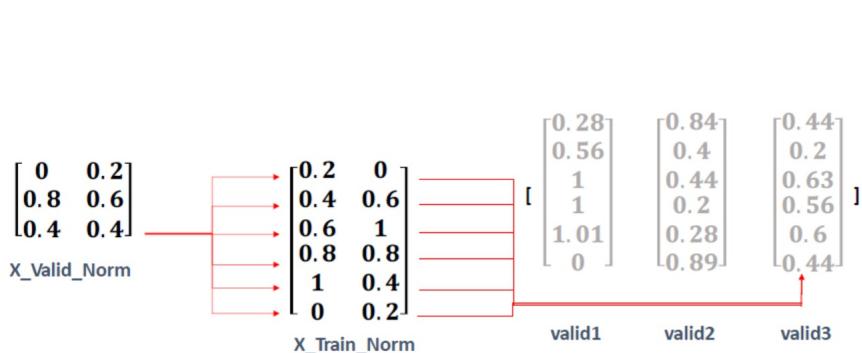
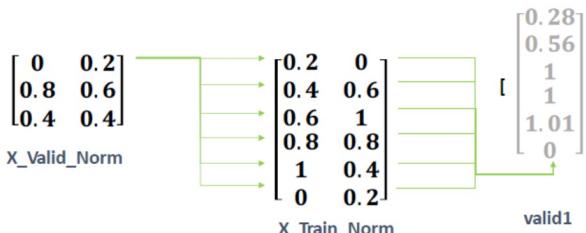
X_Valid_Norm ก็ทำแบบเดียวกันเนอะ

2. วัดระยะห่างระหว่างข้อมูลแต่ละตัวใน Validation Set กับข้อมูลทุกตัวใน Training Set => Distance Function (Euclidean Distance)

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_D - q_D)^2}$$

$$= \sqrt{\sum_{i=1}^D (p_i - q_i)^2}$$





Calculation of distances between x_{valid1} and X_{Train_Norm} :

$$\begin{aligned} [0 & 0.2] &\rightarrow \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix} = \sqrt{(0.2 - 0)^2 + (0 - 0.2)^2} = 0.28 \\ x_{valid1} &\rightarrow \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix} = \sqrt{(0.4 - 0)^2 + (0 - 0.2)^2} = 0.56 \\ &\rightarrow \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix} = \sqrt{(0.6 - 0)^2 + (1 - 0.2)^2} = 1 \\ &\rightarrow \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix} = \sqrt{(0.8 - 0)^2 + (0.8 - 0.2)^2} = 1 \\ &\rightarrow \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix} = \sqrt{(1 - 0)^2 + (0.4 - 0.2)^2} = 1.01 \\ &\rightarrow \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0.6 \\ 0.6 & 1 \\ 0.8 & 0.8 \\ 1 & 0.4 \\ 0 & 0.2 \end{bmatrix} = \sqrt{(0 - 0)^2 + (0.2 - 0.2)^2} = 0 \end{aligned}$$

Result: $\begin{bmatrix} 0.28 \\ 0.56 \\ 1 \\ 1 \\ 1.01 \\ 0 \end{bmatrix}$ → valid1

```
def KNN_find_distance_each_data(X_Train, X_Valid):
    all_distance = []
    for x_valid in X_Valid:
        distance = KNN_find_distance(X_Train, x_valid)
        all_distance.append(distance)
    return all_distance
```

Red arrow pointing from the first code block to the second, indicating the transition from calculating distances for all validation points to calculating the distance for a single validation point.

```
def KNN_find_distance(X_Train, x_valid):
    distance2 = ((X_Train - x_valid)**2).sum(axis=1)
    distance = np.sqrt(distance2)
    return distance
```

3. เรียงลำดับระยะจากขั้น ตอนที่2 จากน้อยไปมาก

$\begin{bmatrix} 0.28 \\ 0.56 \\ 1 \\ 1 \\ 1.01 \\ 0 \end{bmatrix} \begin{bmatrix} thin \\ thin \\ thin \\ fat \\ fat \\ thin \end{bmatrix}$	$\begin{bmatrix} 0.84 \\ 0.4 \\ 0.44 \\ 0.2 \\ 0.28 \\ 0.89 \end{bmatrix} \begin{bmatrix} thin \\ thin \\ thin \\ fat \\ fat \\ thin \end{bmatrix}$	$\begin{bmatrix} 0.44 \\ 0.2 \\ 0.63 \\ 0.56 \\ 0.6 \\ 0.44 \end{bmatrix} \begin{bmatrix} thin \\ thin \\ thin \\ fat \\ fat \\ thin \end{bmatrix}$	\rightarrow	$\begin{bmatrix} 0 \\ 0.28 \\ 0.56 \\ 1 \\ 1 \\ 1.01 \end{bmatrix} \begin{bmatrix} thin \\ thin \\ thin \\ thin \\ fat \\ fat \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ 0.28 \\ 0.4 \\ 0.44 \\ 0.56 \\ 0.6 \\ 0.89 \end{bmatrix} \begin{bmatrix} fat \\ fat \\ thin \\ thin \\ thin \\ fat \\ thin \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ 0.44 \\ 0.44 \\ 0.56 \\ 0.6 \\ 0.63 \end{bmatrix} \begin{bmatrix} thin \\ thin \\ thin \\ fat \\ fat \\ thin \end{bmatrix}$
valid1	valid2	valid3		valid1	valid2	valid3

```
def KNN_find_sorted_target_each_data(Y_Train, all_distance):
    all_sorted_target = []
    for distance in all_distance:
        sorted_target = KNN_sort_target_by_distance(Y_Train, distance)
        all_sorted_target.append(sorted_target)
    return all_sorted_target
```



```
def KNN_sort_target_by_distance(Y_Train, distance):
    sorted_target = Y_Train[distance.argsort()]
    return sorted_target
```

ได้ Index
ที่เรียงลำดับจากน้อยไปมาก

argsort

0 1 2 3 4

```
xxx = np.array([7, 9, 1, 4, 5])
```

```
xxx
```

```
array([7, 9, 1, 4, 5])
```

```
yyy = xxx.argsort()
```

```
yyy
```

```
array([2, 3, 4, 0, 1])
```

```
zzz = np.array(['zero', 'one', 'two', 'three', 'four'])
```

เรียงตามตัวอักษร

```
zzz[yyy]
```

```
array(['two', 'three', 'four', 'zero', 'one'], dtype='<U5')
```

3. เรียงลำดับระยะจากขั้น ตอนที่ 2 จากน้อยไปมาก ตามจำนวน K

(กรณีกำหนดให้ K = 3)

	K	3	Real Class
valid1	thin thin thin thin fat fat	fat fat thin thin thin thin	thin thin thin thin fat thin
valid2	thin thin thin fat thin thin	thin thin thin thin thin thin	fat thin thin thin thin thin
valid3	thin thin thin thin thin thin	thin thin thin thin thin thin	thin thin thin thin thin thin

The diagram illustrates the iterative process of sorting validation data based on the nearest neighbor classification. It shows three stages of sorting:

- Stage 1:** Initial data with K=3. Validation samples valid1, valid2, and valid3 are shown with their respective distances and class labels.
- Stage 2:** After one iteration of sorting, valid1 is correctly classified as thin, valid2 as fat, and valid3 as thin. The sorted data is shown with red boxes highlighting the changes.
- Stage 3:** After another iteration, all validation samples (valid1, valid2, valid3) are correctly classified according to their nearest neighbors.

```
def KNNC_find_class_each_data(all_sorted_target, K):
    all_class = []
    for sorted_target in all_sorted_target:
        _class = KNNC_find_class(sorted_target, K)
        all_class.append(_class)
    all_class = np.array(all_class).reshape(-1, 1) คำนวณในล้อห่วงว่ามีกี่ row
    return all_class
```

numpy unique

The diagram shows the final step of sorting validation data using the `KNNC_find_class` function. The validation samples valid1, valid2, and valid3 are sorted into their final classes (thin, fat, thin) based on the KNNC algorithm's output.

```
def KNNC_find_class(sorted_target, K):
    unique, count_unique = np.unique(sorted_target[:, :], return_counts = True)
    _class = unique[count_unique.argmax()]
    return _class
```

numpy unique

```
aaa = np.array(['bat', 'cat', 'cat', 'ant', 'ant', 'cat'])
```

เรียงตามตัวอักษร
(หรือเลขน้อยไปมาก)

```
unique, count_unique = np.unique(aaa, return_counts = True)
```

ไม่เอาข้า

```
unique
```

```
array(['ant', 'bat', 'cat'], dtype='<U3')
```

มากไปน้อย

```
count_unique
```

```
array([2, 1, 3])
```

```
count_unique.argmax()
```

```
2
```

```
unique[count_unique.argmax()]
```

```
'cat'
```

ลำดับ index
ค่าที่มากสุด
หรือข้ากันมากสุด

4. หาผลลัพธ์จากขั้นตอนที่ 3 ที่ใกล้ที่สุด K ค่า และ Majority Vote เพื่อดูว่าจะใช้ K จำนวนเท่าไหร่ เทียบค่า Error

K = 1

0	<i>thin</i>	0.2	<i>fat</i>	0.2	<i>thin</i>
0.28	<i>thin</i>	0.28	<i>fat</i>	0.44	<i>thin</i>
0.56	<i>thin</i>	0.4	<i>thin</i>	0.44	<i>thin</i>
1	<i>thin</i>	0.44	<i>thin</i>	0.56	<i>fat</i>
1	<i>fat</i>	0.84	<i>thin</i>	0.6	<i>fat</i>
1.01	<i>fat</i>	0.89	<i>thin</i>	0.63	<i>thin</i>

K = 3

0	thin	0.2	<i>fat</i>	0.2	<i>thin</i>
0.28	<i>thin</i>	0.28	<i>fat</i>	0.44	<i>thin</i>
0.56	<i>thin</i>	0.4	<i>thin</i>	0.44	<i>thin</i>
1	<i>thin</i>	0.44	<i>thin</i>	0.56	<i>fat</i>
1	<i>fat</i>	0.84	<i>thin</i>	0.6	<i>fat</i>
1.01	<i>fat</i>	0.89	<i>thin</i>	0.63	<i>thin</i>

K = 5

0	<i>thin</i>	0.2	<i>fat</i>	0.2	<i>thin</i>
0.28	<i>thin</i>	0.28	<i>fat</i>	0.44	<i>thin</i>
0.56	<i>thin</i>	0.4	<i>thin</i>	0.44	<i>thin</i>
1	<i>thin</i>	0.44	<i>thin</i>	0.56	<i>fat</i>
1	<i>fat</i>	0.84	<i>thin</i>	0.6	<i>fat</i>
1.01	<i>fat</i>	0.89	<i>thin</i>	0.63	<i>thin</i>

K = 2

0	<i>thin</i>	0.2	<i>fat</i>	0.2	<i>thin</i>
0.28	<i>thin</i>	0.28	<i>fat</i>	0.44	<i>thin</i>
0.56	<i>thin</i>	0.4	<i>thin</i>	0.44	<i>thin</i>
1	<i>thin</i>	0.44	<i>thin</i>	0.56	<i>fat</i>
1	<i>fat</i>	0.84	<i>thin</i>	0.6	<i>fat</i>
1.01	<i>fat</i>	0.89	<i>thin</i>	0.63	<i>thin</i>

K = 4

	thin	fat	thin	fat
thin	0.2	0.28	0.4	0.44
fat	0.28	0.56	0.44	0.56
thin	0.4	0.44	0.56	0.6
fat	0.44	0.56	0.6	0.63
thin	0.56	0.6	0.63	0.7
fat	0.6	0.63	0.7	0.7
thin	0.7	0.7	0.7	0.7
fat	0.7	0.7	0.7	0.7

K	1	2	3	4	5	Real Class
valid1	thin	thin				thin
valid2	fat	fat				fat
valid3	thin	thin				thin

K	1	2	3	4	5	Real Class
valid1	thin	thin	thin	thin		thin
valid2	fat	fat	fat	fat		fat
valid3	thin	thin	thin	thin		thin

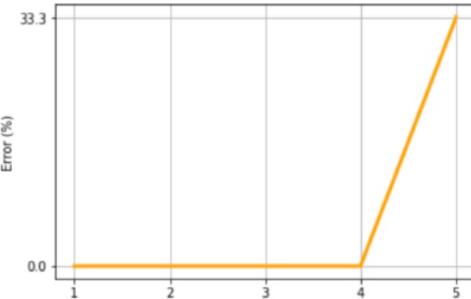
5. หา K ที่ทำให้ Error บน Validation Set ต่ำสุด

K	1	2	3	4	5	Real Class
valid1	thin	thin	thin	thin	thin	thin
valid2	fat	fat	fat	fat	thin	fat
valid3	thin	thin	thin	thin	thin	thin



$$(1/3) * 100\%$$

K	Error
K = 1	0%
K = 2	0%
K = 3	0%
K = 4	0%
K = 5	33%



```
def KNNC_find_error_each_K(Y_Valid, all_sorted_target, min_K, max_K):
    K_list = [K for K in range(min_K, max_K + 1)]
    error_list = []
    for K in K_list:
        K_all_class = KNNC_find_class_each_data(all_sorted_target, K)
        K_error = find_error_classification(Y_Valid, K_all_class)
        error_list.append(K_error)
    return K_list, error_list
```



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

ไม่เท่ากัน = True

```
def KNN_find_best_K(K_list, error_list):
    K_list = np.array(K_list)
    error_list = np.array(error_list)
    plt.plot(K_list, error_list)
    plt.xlabel('K')
    plt.ylabel('Error')
    sorted_K = K_list[error_list.argsort()]
    error_list.sort()
    best_K = sorted_K[0]
    print(best_K)
    print()
    L = len(K_list)
    for l in range(L):
        print('K :', sorted_K[l], ', error :', error_list[l])
    return best_K
```

```
Y = np.array([['ant'], ['ant'], ['ant']])
Yhat = np.array([['cat'], ['ant'], ['ant']])
```

```
Y
```

```
array([['ant'],
       ['ant'],
       ['ant']], dtype='<U3')
```

```
Y.shape[0]
```

```
3
```

```
Y
```

```
array([['ant'],
       ['ant'],
       ['ant']], dtype='<U3')
```

```
Yhat
```

```
array([['cat'],
       ['ant'],
       ['ant']], dtype='<U3')
```

```
Y != Yhat
```

```
array([[ True],
       [False],
       [False]])
```

```
(Y != Yhat).sum()
```

```
1
```

6. เรียนรู้

```
def KNNC_fit(X_Train, Y_Train, X_Valid, Y_Valid, min_K, max_K):
    _min = min4norm(X_Train)
    _max = max4norm(X_Train)
    X_Train_Norm = minmaxNorm(X_Train, _min, _max) (1)
    X_Valid_Norm = minmaxNorm(X_Valid, _min, _max) (2)
    all_distance = KNN_find_distance_each_data(X_Train_Norm, X_Valid_Norm) (3)
    all_sorted_target = KNN_find_sorted_target_each_data(Y_Train, all_distance) (4)
    K_list, error_list = KNNC_find_error_each_K(Y_Valid, all_sorted_target, min_K, max_K)
    best_K = KNN_find_best_K(K_list, error_list) (5)
    return best_K
```

7. พยากรณ์

```
def KNNC_predict(X_Train, Y_Train, X_Test, best_K):
    _min = min4norm(X_Train)
    _max = max4norm(X_Train)
    X_Train_Norm = minmaxNorm(X_Train, _min, _max) (1)
    X_Test_Norm = minmaxNorm(X_Test, _min, _max)
    #all_distance = KNN_find_distance_each_data(X_Train, X_Test) (2)
    all_distance = KNN_find_distance_each_data(X_Train_Norm, X_Test_Norm) (3)
    all_sorted_target = KNN_find_sorted_target_each_data(Y_Train, all_distance)
    Yhat_Test = KNNC_find_class_each_data(all_sorted_target, best_K) (5)
    return Yhat_Test
```

Regression

1. Normalize Feature ของ Training Set และ Validation Set
2. วัดระยะห่างระหว่างข้อมูลแต่ละตัวใน Validation Set กับข้อมูลทุกตัวใน Training Set => Distance Function (Euclidean Distance)
3. เรียงลำดับระยะจากขั้นตอนที่ 2 จากน้อยไปมาก
4. หาผลลัพธ์จากขั้นตอนที่ 3 ที่ใกล้ที่สุด K ค่า และ Majority Vote => ต่างจาก Classification
5. หา K ที่ทำให้ Error บน Validation Set ต่ำสุด

4. หาผลลัพธ์จากขั้นตอนที่ 3 ที่ใกล้ที่สุด K ค่า และ Majority Vote => ต่างจาก Classification

$$MAE = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{Y}_i|$$

```
def find_MAE(Y, Yhat):  
    N = Y.shape[0]  
    MAE = (np.abs(Y - Yhat)).sum() / N  
    return MAE
```

6. เรียนรู้

```

def KNNR_fit(X_Train, Y_Train, X_Valid, Y_Valid, min_K, max_K):
    _min = min4norm(X_Train)
    _max = max4norm(X_Train)
    X_Train_Norm = minmaxNorm(X_Train, _min, _max)
    X_Valid_Norm = minmaxNorm(X_Valid, _min, _max)
    all_distance = KNN_find_distance_each_data(X_Train_Norm, X_Valid_Norm)
    all_sorted_target = KNN_find_sorted_target_each_data(Y_Train, all_distance) 3. ให้ Index
    K_list, error_list = KNNR_find_error_each_K(Y_Valid, all_sorted_target, min_K, max_K) ที่เรียงลำดับจากน้อยไปมาก
    best_K = KNN_find_best_K(K_list, error_list) 5. หา K ที่ทำให้ Error บน
    return best_K

```

ขั้นตอน 4
นำผลลัพธ์จากขั้นตอนที่ 3 ที่ได้มาที่สุด K ค่า และ
Majority Vote

2. วัดระยะห่างระหว่างข้อมูลแต่ละตัวใน Validation Set กับข้อมูลทุกตัวใน Training Set

3. ให้ Index ที่เรียงลำดับจากน้อยไปมาก

5. หา K ที่ทำให้ Error บน Validation Set ต่ำสุด

```

def KNNR_find_error_each_K(Y_Valid, all_sorted_target, min_K, max_K):
    K_list = [K for K in range(min_K, max_K + 1)]
    error_list = []
    for K in K_list:
        K_all_value = KNNR_find_value_each_data(all_sorted_target, K)
        K_error = find_MAE(Y_Valid, K_all_value)
        error_list.append(K_error)
    return K_list, error_list

```

K : 5 ,	error : 0.2793283445042106
K : 6 ,	error : 0.2796154639730383
K : 4 ,	error : 0.2870354959898069
K : 1 ,	error : 0.2906886155919456
K : 3 ,	error : 0.2913682342595428
K : 7 ,	error : 0.29161310829704473
K : 2 ,	error : 0.2940527745937604
K : 9 ,	error : 0.30425812935574215
K : 8 ,	error : 0.3079550031723539
K : 10 ,	error : 0.315570319950900734
K : 11 ,	error : 0.32320543335057766
K : 12 ,	error : 0.3240056570867888

```

def find_MAE(Y, Yhat):
    N = Y.shape[0]
    MAE = (np.abs(Y - Yhat)).sum() / N
    return MAE

```

```

def KNNR_find_value_each_data(all_sorted_target, K):
    all_value = []
    for sorted_target in all_sorted_target:
        value = KNNR_find_value(sorted_target, K)
        all_value.append(value)
    all_value = np.array(all_value).reshape(-1, 1)
    return all_value

```

```

def KNNR_find_value(sorted_target, K):
    value = sorted_target[:K, :].mean()
    return value

```

ค่าเฉลี่ย

	Weight	Height	Target
0	66.17	185.21	19.289713
1	71.27	168.91	24.980524
2	73.17	179.83	22.625011
3	81.74	171.76	27.707434
4	92.97	172.89	31.102301
5	59.45	168.20	21.013901

7. พยากรณ์

```
def KNNR_predict(X_Train, Y_Train, X_Test, best_K):
    _min = min4norm(X_Train)
    _max = max4norm(X_Train)
    X_Train_Norm = minmaxNorm(X_Train, _min, _max)      (1)
    X_Test_Norm = minmaxNorm(X_Test, _min, _max)
    #all_distance = KNN_find_distance_each_data(X_Train, X_Test)
    all_distance = KNN_find_distance_each_data(X_Train_Norm, X_Test_Norm)      (2)
    all_sorted_target = KNN_find_sorted_target_each_data(Y_Train, all_distance)
    Yhat_Test = KNNR_find_value_each_data(all_sorted_target, best_K)           (3)
    return Yhat_Test
```

```
def KNNR_find_value_each_data(all_sorted_target, K):
    all_value = []
    for sorted_target in all_sorted_target:
        value = KNNR_find_value(sorted_target, K)
        all_value.append(value)
    all_value = np.array(all_value).reshape(-1, 1)
    return all_value
```

```
def KNNR_find_value(sorted_target, K):
    value = sorted_target[:K, :].mean()
    return value
```

ค่าเฉลี่ย