

# Introduction to Scikit-learn

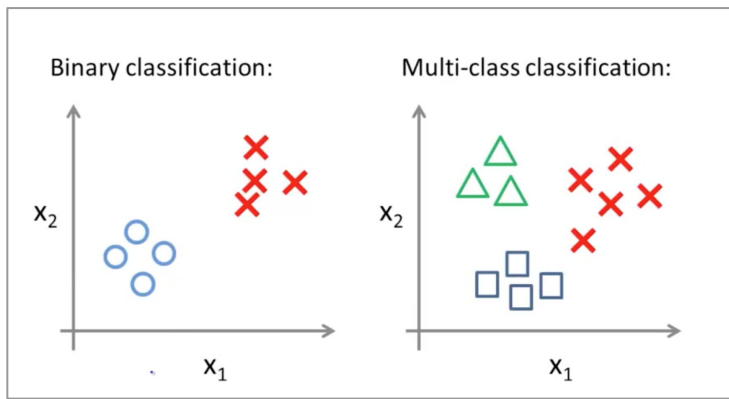
## through K-Nearest Neighbors

DR. SETHAVIDH GERTPHOL

# Classification

**Classification** คือ การจำแนกหมวดหมู่ของข้อมูลตามผลเฉลย (label) โดยใช้โมเดลที่ผ่านการฝึก

- **Feature Value** = Numeric / Categorical
- **LABEL** = Categorical Output Value
- **Example**
  - **Features:** อายุ, เพศ, รายได้, อาชีพ
  - **Label:** เป็นผู้ซื้อ, ไม่เป็นผู้ซื้อ
- **ประเภทของ Classification**
  - Binary Classification
  - Multi-class Classification
  - Multi-label Classification



Source:

[https://raw.githubusercontent.com/ritchieng/machine-learning-stanford/master/w3\\_logistic\\_regression\\_regularization/multiclass\\_classification.png](https://raw.githubusercontent.com/ritchieng/machine-learning-stanford/master/w3_logistic_regression_regularization/multiclass_classification.png)

- Building a model from training set
- Model **creates a decision surface** through training data
- Making **prediction** by checking **which side of the decision surface** that a new data point lies
- Decision surface can be **linear or curved** depending on models

# Classification Algorithms



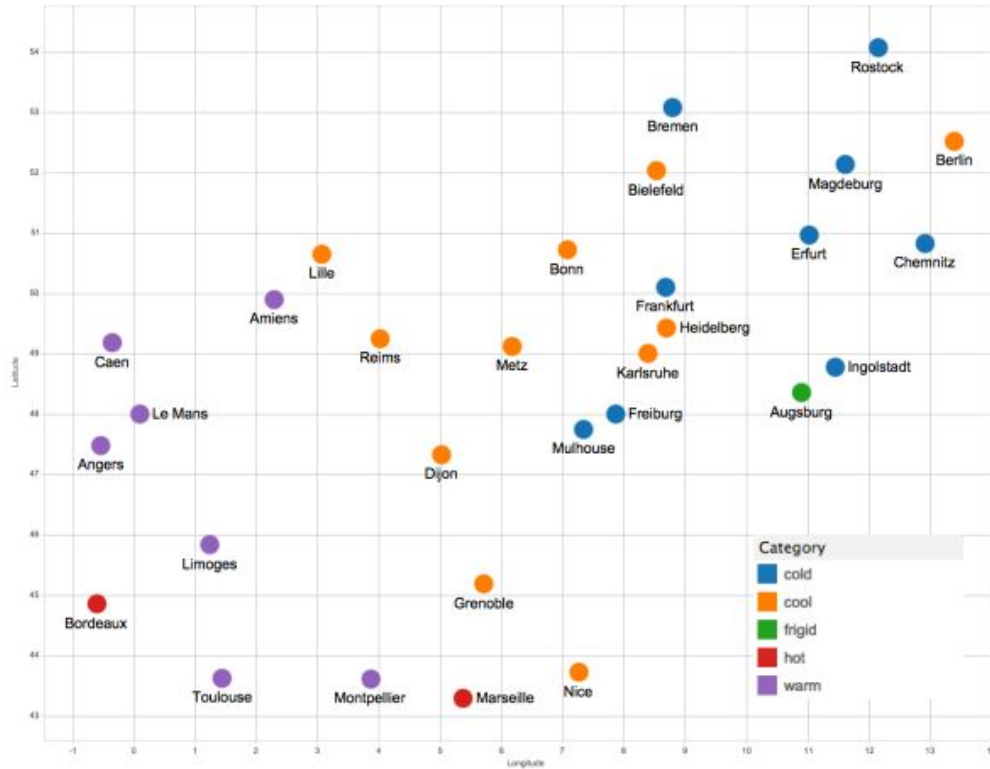
# K-nearest Neighbors (KNN)

- เริ่มต้น Train โมเดลด้วยการจำข้อมูลของ Training ทั้งหมด
- ข้อมูล Train แต่ละตัวอย่างประกอบด้วย feature:  $x_1, x_2, \dots, x_n$  เมื่อ  $n$  คือจำนวน feature ทั้งหมด และ  $l$  คือ label (class) ของตัวอย่างนั้น
- เมื่อมีตัวอย่างใหม่ (Test) เข้ามา โมเดลจะคำนวณหาตัวอย่างเก่าที่ใกล้กับข้อมูล Test ที่สุด  $k$  ตัว ( $k$  คือพารามิเตอร์ที่ผู้สร้างโมเดลปรับได้ ปกติคือ 5)
- วิธีการคำนวณหาระยะห่างของจุดข้อมูล
  - Euclidean distance  $d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$
  - Manhattan distance  $d(x, y) = |x_1 - y_1| + |x_2 - y_2|$
- จากนั้นดูว่า  $k$  ตัวอย่างเก่าที่ใกล้ที่สุดมี label อะไรมากที่สุด  
จะเป็น label ของ Test

# K-nearest Neighbors Example

- City Temperatures Prediction – France and Germany
- **Features:** longitude, latitude
- **Distance:** Euclidean method
- **Labels:** frigid, cold, cool, warm, hot
- Training Example
  - Nice (7.27, 43.72) **cool**
  - Toulouse (1.45, 43.62) **warm**
  - Frankfurt (8.68, 50.1) **cold**

# K-nearest Neighbors Training



# K-nearest Neighbor Test

- New City: Paris Lat: 48, Long: 2

- **K = 5**

- **Closest Cities and Labels**

Amiens                  Warm

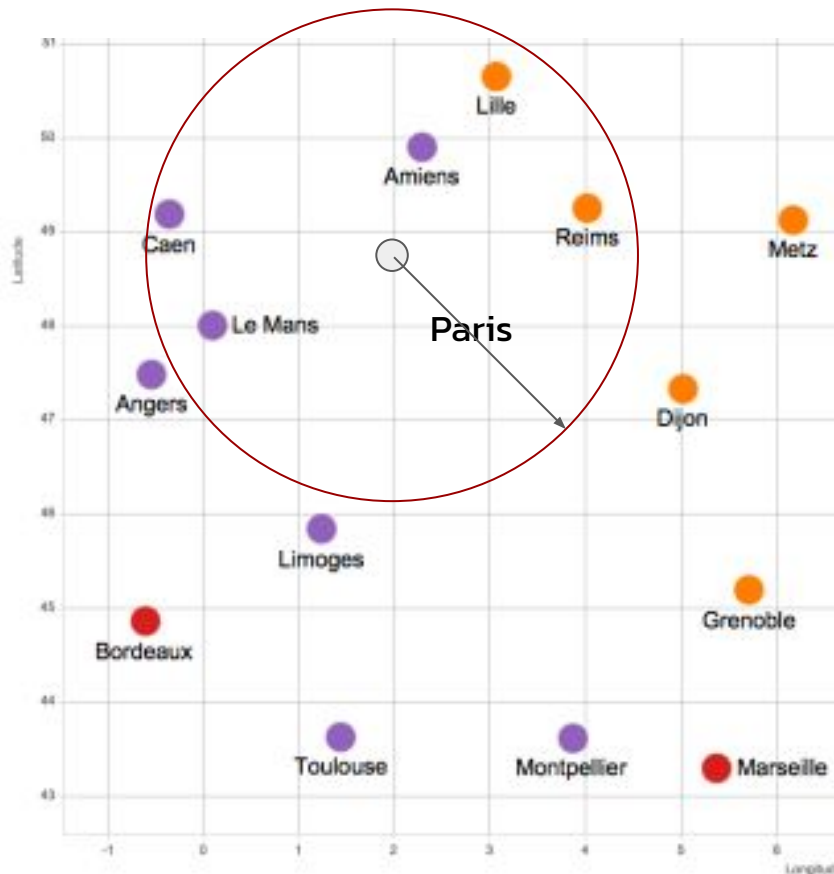
Le Mans                Warm

Caen                    Warm

Reims                   Cool

Lille                    Cool

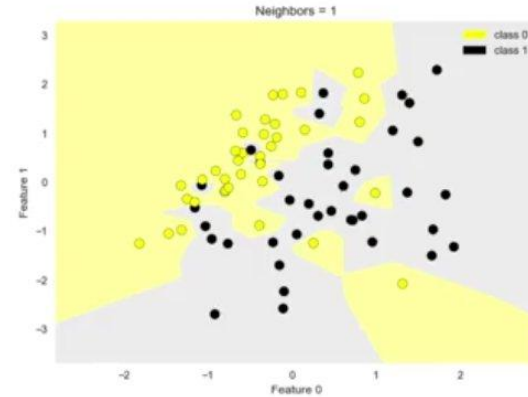
- **Paris is labeled Warm**



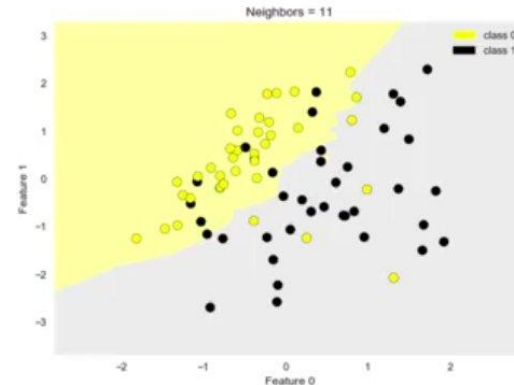
## ค่า k

- ค่า k ที่ต่ำ (เช่น  $k=1$ ) จะทำให้โมเดล overfit
  - เพราะ sensitive ต่อความแปรปรวนของข้อมูล
  - พิจารณาแค่ข้อมูลตัวเดียวที่อยู่ใกล้ ถ้าตัวอย่างใกล้สุดเพี้ยนมี label ไม่เหมือนตัวอื่นรอบๆ จะทำให้โมเดลเข้าใจผิดได้
- ค่า k ที่สูง (เช่น  $k = 50$ ) จะทำให้โมเดล underfit
  - เพราะดูภาพรวมมากเกินไป

## Nearest neighbors classification ( $k=1$ )



## Nearest neighbors classification ( $k=11$ )





# ประเด็นอื่นของ k-Nearest Neighbors

- อาจต้องทำการ rescale ค่าของ feature ที่มี range ต่างกันมาก
  - เงินเดือน: 6000 - 200000 ความแตกต่างของเงินเดือนที่ 20 บาทนั้นน้อยมาก
  - อายุ: 18-82 ความแตกต่างของอายุที่ 20 ปีนั้นสูงมาก
  - Rescale ให้ทั้งเงินเดือนและอายุอยู่ในช่วง [0, 1] ทั้งคู่
- นำ feature ที่เป็น category มาสร้างโมเดลยาก
  - ต้องเปลี่ยน categorical feature ให้เป็น numerical และให้ค่า distance ด้วย
  - สามารถใช้ one-hot encoding ช่วยได้

One-hot encoding

Color		Red	Yellow	Green
Red		1	0	0
Red		1	0	0
Yellow		0	1	0
Green		0	0	1
Yellow		0	0	1

<https://naadispeaks.wordpress.com/2018/04/09/one-hot-encoding-in-practice/>

# ข้อดีข้อเสียของ kNN

## ข้อดี

- เข้าใจง่าย
- ปรับใช้กับ Multi-class/Multi-label หรือ Regression ได้ง่าย

## ข้อเสีย

- ถ้า feature หรือจำนวนแถวเยอะๆ จะใช้เวลาคำนวณนาน
- โมเดลมีขนาดใหญ่เพราะต้องจำข้อมูลชุด train ทั้งหมด

# Using Pandas and Scikit-learn

- import pandas เพื่ออ่านข้อมูลเข้าเป็นตาราง (DataFrame)
  - ใช้ method read\_csv() เพื่ออ่านข้อมูลจากไฟล์ CSV



```
import pandas as pd
```

```
df = pd.read_csv("../drive/MyDrive/Datasets/01-census-income.csv")  
df
```



	age	workclass	weight	education	edu num	marital status	occupation
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial

## Select label column

- สังเกตว่า label ไม่ได้เป็นตาราง เป็นแค่คอลัมน์เดียว โครงสร้างข้อมูลแบบนี้ใน pandas เรียกว่า Series

หมายเหตุ: ตอนนี้จะ  
แสดงกระบวนการที่  
ยังไม่ถูกต้องก่อน

```
[4] y = df[ 'label' ]  
y
```

เลือกคอลัมน์เดียว  
โดยใช้ชื่อคอลัมน์ระหว่าง [ ]

```
0      No  
1      No  
2      Yes  
3      Yes  
4      Yes
```

...

```
33869   No  
33870   No  
33871   No  
33872   No  
33873   Yes
```

```
Name: label, Length: 33874, dtype: object
```

## Select input columns

- เลือกเฉพาะคอลัมน์ที่มีค่าเป็นตัวเลข ยกเว้น weight ที่ไม่ทราบว่าจะเป็นอะไร

หมายเหตุ: ตอนนี้จะ  
แสดงกระบวนการที่  
ยังไม่ถูกต้องก่อน

```
[5] df.columns
```

```
Index(['age', 'workclass', 'weight', 'education', 'edu num', 'marital status',  
       'occupation', 'relationship', 'race', 'sex', 'captial-gain',  
       'capital-loss', 'hours-per-week', 'native country', 'label'],  
      dtype='object')
```

เลือกคอลัมน์ใน  
ตารางใหม่โดยใช้  
ชื่อคอลัมน์ใน list

```
▶ X = df[ ['edu num', 'captial-gain', 'capital-loss', 'hours-per-week'] ]  
X
```

```
↗
```

	edu num	captial-gain	capital-loss	hours-per-week
0	13	2174	0	40
1	13	0	0	13
2	9	0	0	40
3	7	0	0	40

# Import and train K-nearest neighbors model

- Import model แล้วสร้าง k-nearest neighbors object โดยระบุจำนวน neighbors ที่จะใช้ (ค่า k)
- สร้างโมเดลโดยใช้ method fit() และส่ง feature dataframe กับ label เป็น argument

```
[5] from sklearn.neighbors import KNeighborsClassifier
```



```
knn = KNeighborsClassifier(15)  
knn.fit(X,y)
```



```
KNeighborsClassifier(n_neighbors=15)
```

หมายเหตุ: ตอนนี้จะ  
แสดงกระบวนการที่  
ยังไม่ถูกต้องก่อน

# Prediction

- ใช้ method `predict()` เพื่อทำนาย label ของข้อมูลใหม่
- ต้องใส่ **ตาราง** เป็น argument ของ `predict`

```
[24] knn.predict( [ [13,10000,0,40], [6, 0, 0, 10] ] )
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning:  
"X does not have valid feature names, but"  
array(['Yes', 'No'], dtype=object)
```

Sklearn เตือนว่าตาราง  
ที่ใส่ไม่มีชื่อคอลัมน์ แต่ก็  
predict ให้

# สร้างตารางของข้อมูลใหม่อย่างถูกต้อง

- สร้างตารางเป็น DataFrame และใส่ชื่อคอลัมน์เป็น keyword argument

ใช้ DataFrame  
class ใน pandas

ชื่อคอลัมน์ของตาราง  
ใหม่เหมือนกับชื่อคอลัมน์ของตาราง feature

```
[25] new_data = pd.DataFrame( [ [13,10000,0,40], [6, 0, 0, 10] ], columns=X.columns )  
new_data
```

	edu num	captial-gain	capital-loss	hours-per-week
0	13	10000	0	40
1	6	0	0	10



```
[26] knn.predict(new_data)
```

```
array(['Yes', 'No'], dtype=object)
```



# ค่า Accuracy ของโมเดล

- ใช้ method score() ในการคำนวณค่า accuracy โดยใส่ตาราง feature กับ label ผลเลย
- วิธีนี้ผิดเพราะอะไร
- ข้อมูลที่ใช้สร้างโมเดล (fit) เป็นชุดเดียวกันกับข้อมูลทดสอบหาค่า accuracy (score)
- ผู้สร้างโมเดลปรับจูนโมเดลจนเข้ากับข้อมูลได้ แต่อาจไม่ดีต่อข้อมูลในอนาคต

หมายเหตุ: ตอนนี้จะ  
แสดงกระบวนการที่  
ยังไม่ถูกต้องก่อน



```
knn.score(X,y)
```



```
0.8146661156048887
```

```
[91] knn_tune = KNeighborsClassifier(81)  
      knn_tune.fit(X,y)  
      knn_tune.score(X,y)
```

```
0.8325264214441754
```

จูนค่า k ไปเรื่อย ๆ จน  
ได้ accuracy ที่สูง

# ลองเอาข้อมูลที่โมเดลไม่เคยเห็นมาทดสอบ

```
[24] unseen_data = pd.read_csv("./drive/MyDrive/Datasets/02-future-census.csv")
```

```
[25] unseen_X = unseen_data[ ['edu num', 'captial-gain', 'capital-loss', 'hours-per-week' ] ]  
unseen_y = unseen_data[ 'label' ]
```

```
[92] knn_tune.score(unseen_X, unseen_y)
```

ทดสอบโมเดลที่จูน  
แล้วด้วยข้อมูลอนาคต

0.8273955773955773

Accuracy ตก  
ไป 0.5%

# วิธีที่ถูกต้อง

- แบ่งชุดข้อมูลเป็นสองชุดก่อนคือชุด train กับชุด test
- ใช้ function `train_test_split` จาก `model_selection` submodule ช่วยแบ่ง โดยใส่ตาราง feature และคอลัมน์ label
- จะได้ข้อมูลที่ split แล้วเป็น `X_train`, `X_test`, `y_train`, `y_test` ตามลำดับ

```
[12] from sklearn.model_selection import train_test_split
```

```
[55] X_train, X_test, y_train, y_test = train_test_split(X, y)
```

# สร้างโมเดลและทดสอบอย่างถูกต้อง

- สร้างโมเดลด้วยชุด train และทดสอบด้วยชุด test

```
[76] knn_new = KNeighborsClassifier(25)
      knn_new.fit(X_train, y_train)
      knn_new.score(X_test, y_test)
```

สร้างโมเดลด้วยชุด train

ทดสอบโมเดลด้วยชุด test

```
0.8337466052662652
```

```
[79] knn_new.score(unseen_X, unseen_y)
```

ทดสอบโมเดลที่จูน  
แล้วด้วยข้อมูลอนาคต

```
0.831081081081081
```

Accuracy ลดลง  
ไป 0.27%

## K-fold cross validation

- ใช้ฟังก์ชัน `cross_validate_score()` ใน submodule `model_selection` เพื่อคำนวณค่า accuracy ของแต่ละรอบในการทดสอบ k ครั้ง
- สร้าง object ที่เป็นโมเดลที่ยังไม่ได้ `fit()` เพื่อส่งเป็น argument ให้ฟังก์ชัน
- ไม่ต้องแบ่งข้อมูลเป็นชุด train กับ test ก่อน เพราะฟังก์ชันจะแบ่งให้เองในแต่ละรอบ

```
[21] from sklearn.model_selection import cross_val_score
```

```
[26] knn9_cv = KNeighborsClassifier(9)
```

```
[27] cross_val_score(knn9_cv, X, y, cv=5)
```

```
array([0.83232472, 0.83527675, 0.81874539, 0.8295203 , 0.82285208])
```

cv คือจำนวนรอบที่จะทำ  
k-fold cross validation

ค่า accuracy  
ของแต่ละรอบ

## การใช้ k-fold cross validation

```
[34] knn9_cv = KNeighborsClassifier(9)
      knn17_cv = KNeighborsClassifier(17)

[35] result_knn9 = cross_val_score(knn9_cv, X, y, cv=5)
      result_knn17 = cross_val_score(knn17_cv, X, y, cv=5)

[36] result_knn9

array([0.83232472, 0.83527675, 0.81874539, 0.8295203 , 0.82285208])

[37] result_knn17

array([0.83394834, 0.83439114, 0.8295203 , 0.83557196, 0.82846177])

[38] result_knn9.mean(), result_knn17.mean()

(0.8277438480319258, 0.8323786999783195)
```

เฉลี่ยแล้ว  
neighbors = 17  
ได้ผลดีกว่า

- คำนวณค่าเฉลี่ยความแม่นยำของกระบวนการสร้างโมเดล ป้องกันการตัดสินใจผิดพลาดเนื่องจาก**ความบังเอิญ** ในการแบ่งชุด train/test
- เปรียบเทียบความแม่นยำของการสร้างโมเดลสองวิธี โดยดูจากค่าเฉลี่ยความแม่นยำ

## การใช้ k-fold cross validation

- เมื่อได้กระบวนการที่คิดว่าดีที่สุดแล้ว จึง train โมเดลด้วยกระบวนการนั้น และด้วยข้อมูลที่มีทั้งหมด

```
[39] knn17_final_model = KNeighborsClassifier(17)  
knn17_final_model.fit(X, y)
```

```
KNeighborsClassifier(n_neighbors=17)
```

Model ตัวสุดท้าย  
จะ Train กับข้อมูล  
ทั้งหมด

```
[40] knn17_final_model.score(unseen_X, unseen_y)
```

```
0.8306715806715806
```

# Hyperparameter tuning

- ใน sklearn มีฟังก์ชันที่ใช้หาค่า hyperparameter ที่ดีที่สุดหลายฟังก์ชัน
- **GridSearchCV()** (อยู่ในโมดูลย่อย model\_selection) จะไล่ปรับค่า hyperparameter ตามที่กำหนดทุก combination แล้วระบุค่าที่ดีที่สุดให้
- **สร้างโมเดลที่ยังไม่ได้กำหนด hyperparameter** แล้วส่งเป็น argument
- ค่า hyperparameter ส่งเป็น **dictionary** โดยมี **key** เป็นชื่อ **keyword argument** ที่จะปรับ และ **value** เป็นลิสต์ของค่าที่จะปรับ
- ระบุจำนวน fold ที่จะใช้ด้วย (cv)



## GridSearchCV()

```
[42] knn_search = KNeighborsClassifier()
```

สร้าง model  
object ที่ยังไม่  
ได้ fit()

ชื่อและค่า  
hyperparameter  
ที่จะปรับ

```
[43] grid_search = GridSearchCV(knn_search  
                                , param_grid={'n_neighbors': (9,11,13,15,17,19,21,23,25)}  
                                , cv=5)
```

```
grid_search.fit(X, y)
```

fit() ด้วยข้อมูล  
ทั้งหมดที่ไม่ได้ split

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
              param_grid={'n_neighbors': (9, 11, 13, 15, 17, 19, 21, 23, 25)})
```

```
[45] grid_search.best_params_ , grid_search.best_score_
```

hyperparameter ที่ดี  
ที่สุด และค่า  
accuracy เก็บอยู่ใน  
ตัวแปรสองตัวนี้

```
({'n_neighbors': 25}, 0.8342975278822762)
```

## ผลลัพธ์ของ GridSearchCV()

- ผลลัพธ์ของค่า accuracy ของ hyperparameter แต่ละชุดอยู่ในตัวแปรชื่อ cv\_results\_
- GridSearchCV() จะ train โมเดลด้วย hyperparameter ที่ดีที่สุดให้ด้วย สามารถเรียกใช้ผ่าน predict() และ score() ได้ตามปกติ

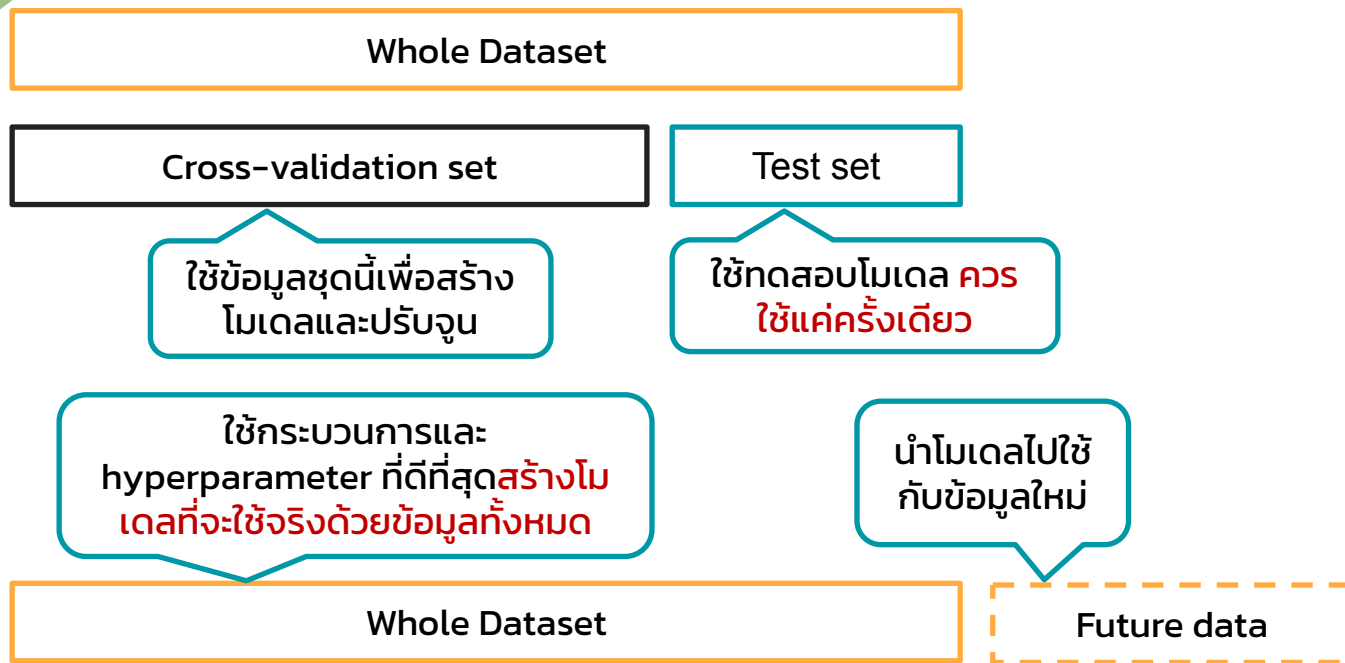
```
[47] grid_search.predict(new_data)

array(['Yes', 'No'], dtype=object)

[48] grid_search.score(unseen_X, unseen_y)

0.8348689598689598
```

# สรุปกระบวนการสร้าง และทดสอบโมเดล



# Data Preprocessing

- สังเกตว่าค่าของบางคอลัมน์ต่างกันมาก เช่น "edu num" กับ "captial-gain"
- feature ที่มีค่ามากจะมี**น้ำหนักมากกว่า** feature ที่มีค่าน้อยเวลาคำนวณร่วมกัน
  - เช่นใน KNN ตอนคำนวณ distance
- ควรปรับค่าตัวเลขให้อยู่ในช่วงเดียวกัน
- Sklearn มีการปรับค่าหลายวิธี เช่น Standardization และ Scaling



	edu num	captial-gain	capital-loss	hours-per-week
0	13	2174	0	40
1	13	0	0	13
2	9	0	0	40
3	7	0	0	40



# Standardization

- เป็นการปรับค่าโดยนำข้อมูลมาลบค่าเฉลี่ยและหารด้วยส่วนเบี่ยงเบนมาตรฐาน
- ชุดข้อมูลที่ปรับแล้วจะมีค่าเฉลี่ยประมาณ 0 และส่วนเบี่ยงเบนมาตรฐานประมาณ 1
- ใช้ `StandardScaler()` ใน `sklearn`
  - ใช้ method `fit()` แล้วใส่ตาราง X ที่ต้องการปรับสเกลเป็น argument
  - `StandardScaler()` จะเรียนรู้ค่าเฉลี่ยและส่วนเบี่ยงเบนมาตรฐานของแต่ละคอลัมน์
  - เมื่อต้องการปรับค่าให้ใช้ method `transform()` ก่อนนำข้อมูลที่ปรับค่าแล้วไปใช้
- เมื่อต้องการทดสอบหรือทำนายก็ต้องใช้ `StandardScaler()` ตัวเดิมปรับค่าของข้อมูล

# StandardScaler()

```
[36] from sklearn.preprocessing import StandardScaler
```

import

```
[37] std_scaler = StandardScaler()
```

สร้าง  
**StandardScaler()**  
object

```
[38] std_scaler.fit(X)
```

**fit()** ในที่นี้เราใช้ทั้ง  
ตาราง X โดยไม่ split  
ก่อน เพื่อจะนำไปทำ  
cross-validation ต่อ

```
[51] X
```

	edu num	capitlal-gain	capital-loss	hours-per-week
--	---------	---------------	--------------	----------------

0	13	2174	0	40
1	14	0	0	40
2	13	5178	0	40

## ผลลัพธ์จาก Scaler

ค่า mean และ standard deviation ของแต่ละคอลัมน์จะเก็บไว้ใน attribute ชื่อ **mean\_** และ **scale\_** ตามลำดับ

```
[42] std_scaler.mean_, std_scaler.scale_
```

```
(array([ 10.06671784, 1051.42141465,  87.46982937,  40.41955482]),  
 array([2.57076281e+00, 7.20456775e+03, 4.01533636e+02, 1.23810551e+01]))
```

```
[47] pd.DataFrame(zip(std_scaler.mean_, std_scaler.scale_), index=X.columns, columns=['mean', 'std']).T
```

	edu num	captial-gain	capital-loss	hours-per-week
mean	10.066718	1051.421415	87.469829	40.419555
std	2.570763	7204.567748	401.533636	12.381055



ใช้ **transform()** เพื่อ  
ปรับค่า ด้วย mean\_  
และ scale\_ ที่เรียนรู้ไป

## transform()

```
[49] X_transformed = std_scaler.transform(X)
      X_std_scaled = pd.DataFrame(X_transformed, columns=X.columns)
      X_std_scaled
```

	edu num	captial-gain	capital-loss	hours-per-week
0	1.141016	0.155815	-0.217839	-0.033887
1	1.530006	-0.145938	-0.217839	-0.033887
2	1.141016	0.572773	-0.217839	-0.033887

ตารางใหม่จะมีค่าเฉลี่ย  
ของแต่ละคอลัมน์ใกล้ 0  
และส่วนเบี่ยงเบน  
มาตรฐานใกล้ 1

```
[53] X_std_scaled.mean()
```

```
edu num          2.920260e-16
captial-gain      4.695563e-16
capital-loss      5.269200e-16
hours-per-week   -3.587495e-16
dtype: float64
```

```
[54] X_std_scaled.std(ddof=0)
```

```
edu num          1.0
captial-gain      1.0
capital-loss      1.0
hours-per-week    1.0
dtype: float64
```



# สร้างโมเดลและปรับ hyperparameter

```
[55] grid_search_std_scaled = GridSearchCV(knn_search  
      , param_grid={'n_neighbors': (9,11,13,15,17,19,21,23,25)}  
      , cv=5)
```

```
grid_search_std_scaled.fit(X_std_scaled, y)
```

ทำ Grid Search ด้วย  
hyperparameters ชุดเดิม แต่ fit()  
ด้วย X ที่ transform แล้ว

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
      param_grid={'n_neighbors': (9, 11, 13, 15, 17, 19, 21, 23, 25)})
```

```
[56] grid_search_std_scaled.best_params_, grid_search_std_scaled.best_score_  
  
({'n_neighbors': 13}, 0.8209539578832459)
```

ได้ค่า k ที่ดีที่สุดต่างไปจากครั้งก่อน

## ทดสอบด้วยข้อมูลใหม่

```
▶ unseen_X_std_scaled = pd.DataFrame(std_scaler.transform(unseen_X), columns=unseen_X.columns)  
unseen_X_std_scaled
```

```
↳
```

	edu num	capitail-gain	capital-loss	hours-per-week
0	-0.414942	-0.145938	-0.217839	-0.033887
1	1.918995	-0.145938	-0.217839	0.369956
2	-1.581911	-0.145938	-0.217839	-2.295407

อย่าลืม transform() ค่า X ชุดใหม่  
ด้วย scaler ตัวเดิม

```
[58] grid_search_std_scaled.score(unseen_X_std_scaled, unseen_y)
```

```
0.8193079443079443
```

ทดสอบด้วย X ชุดใหม่  
ที่ scale แล้ว

- **MinMaxScaler()** จะปรับข้อมูลทั้งหมดให้อยู่ในช่วง  $[0,1]$ 
  - ลบข้อมูลด้วยค่าน้อยสุด แล้วหารด้วยพิสัย
  - ค่าน้อยสุดจะปรับเป็น 0, ค่ามากที่สุดเป็น 1
- **MaxAbsScaler()** จะปรับข้อมูลทั้งหมดให้อยู่ในช่วง  $[-1, 1]$ 
  - หารข้อมูลด้วยค่ามากที่สุดของค่า absolute ของข้อมูล
- มี method `fit()`, `transform()`, และ `fit_transform()` เหมือน `StandardScaler()`

## MinMaxScaler

```
[48] from sklearn.preprocessing import MinMaxScaler, MaxAbsScaler
```

import

```
[49] mms = MinMaxScaler()  
X_mms = mms.fit_transform(X)  
grid_search_mms_scaled = GridSearchCV(knn_search  
                                     , param_grid={'n_neighbors': (9,11,13,15,17,19,21,23,25)}  
                                     , cv=5)  
grid_search_mms_scaled.fit(X_mms, y)  
grid_search_mms_scaled.best_params_, grid_search_mms_scaled.best_score_  
  
({'n_neighbors': 25}, 0.8203931071374487)
```

fit\_transform() จะ  
เรียนรู้พร้อมทั้งเปลี่ยน  
ค่าในตาราง X ให้เลย

กระบวนการทั้งหมด  
เหมือนกับตอนใช้  
StandardScaler

```
[50] grid_search_mms_scaled.score(mms.transform(unseen_X), unseen_y)
```

0.8203316953316954

ทดสอบด้วย X ชุดใหม่  
ที่ scale แล้ว

## MaxAbsScaler

```
[58] mas = MaxAbsScaler()
X_mas = mas.fit_transform(X)
grid_search_mas_scaled = GridSearchCV(knn_search
                                     , param_grid={'n_neighbors': (9,11,13,15,17,19,21,23,25)}
                                     , cv=5)
grid_search_mas_scaled.fit(X_mas, y)
grid_search_mas_scaled.best_params_, grid_search_mms_scaled.best_score_

({'n_neighbors': 25}, 0.8203931071374487)

[60] grid_search_mas_scaled.score(mas.transform(unseen_X), unseen_y)

0.8202293202293203
```

## Scaler attributes

```
[65] mms.data_min_, mms.data_range_  
  
(array([1., 0., 0., 1.]),  
 array([1.5000e+01, 9.9999e+04, 4.3560e+03, 9.8000e+01]))
```

MinMaxScaler: **data\_min\_** แสดงค่าต่ำสุด  
**data\_range\_** แสดงพิสัยของคอลัมน์

```
[68] X_mms.min(axis=0), X_mms.max(axis=0)  
  
(array([0., 0., 0., 0.]), array([1., 1., 1., 1.]))
```

ค่า min, max ของ  
ตารางที่ปรับแล้ว

```
[62] mas.max_abs_  
  
array([1.6000e+01, 9.9999e+04, 4.3560e+03, 9.9000e+01])
```

MaxAbsScaler: **max\_abs\_** แสดงค่าที่มาก  
ที่สุดหลังหาค่า absolute แล้ว

```
[69] X_mas.min(axis=0), X_mas.max(axis=0)  
  
(array([0.0625      , 0.          , 0.          , 0.01010101]),  
 array([1., 1., 1., 1.]))
```

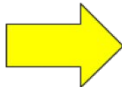
ค่า min, max ของ  
ตารางที่ปรับแล้ว

# การปรับค่าหมวดหมู่

- หมวดหมู่ที่เป็นตัวอักษรไม่สามารถนำไปคำนวณได้ ต้องเปลี่ยนเป็นตัวเลขก่อน
- ถ้าเป็นหมวดหมู่แบบ**มีลำดับ** (Ordinal)
  - สามารถแทนค่าตัวเลขจากน้อยไปมากตามลำดับได้ แต่ค่าควรเป็นเท่าไรจะต้องพิจารณาเอง
- ถ้าเป็นหมวดหมู่แบบ**ไม่มีลำดับ** (Nominal) และ**มีแค่สองค่า** (เช่น ชาย/หญิง)
  - สามารถแทนเป็น 0/1 ได้
- อย่างไรก็ดี ควรปรับหมวดหมู่ไม่ว่ารูปแบบใดด้วยวิธี one-hot encoding

# One-Hot encoding

- เทคนิคที่ใช้ปรับหมวดหมู่  
แบบ Nominal เป็นตัวเลข
- สร้างคอลัมน์ใหม่จำนวน  
เท่ากับค่าของหมวดหมู่
  - ใส่ค่า 1 ในคอลัมน์ที่ตรงกับค่าของแถวนั้น
  - ใส่ 0 ในคอลัมน์อื่น
  - 1 หมายถึงแถวนี้มีค่าของคอลัมน์นี้



Color		Red	Yellow	Green
Red		1	0	0
Red		1	0	0
Yellow		0	1	0
Green		0	0	1
Yellow		0	0	1

<https://naadispeaks.wordpress.com/2018/04/09/one-hot-encoding-in-practice/>



# OneHotEncoder

```
[57] from sklearn.preprocessing import OneHotEncoder
```

import

```
[58] ohe = OneHotEncoder()  
      marital_1hot = ohe.fit_transform(df[ ['marital status'] ] )  
      marital_1hot
```

fit() ด้วยคอลัมน์ที่เป็น category

```
<33874x7 sparse matrix of type '<class 'numpy.float64'>'  
with 33874 stored elements in Compressed Sparse Row format>
```

ตารางที่ transform แล้วจะเป็น  
ประเภท sparse matrix (มี 0 เป็น  
จำนวนมาก)

# OneHotEncoder

```
[59] marital_1hot.toarray()

array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 1., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.]])
```

ใช้ method  
toarray() เพื่อ  
แสดง sparse  
matrix ให้เป็น  
ndarray

```
[60] ohe.categories_
```

```
[array([' Divorced', ' Married-AF-spouse', ' Married-civ-spouse',
       ' Married-spouse-absent', ' Never-married', ' Separated',
       ' Widowed'], dtype=object)]
```

ชื่อคอลัมน์เดิมจะเก็บไว้ใน  
attribute ชื่อ **categories\_**

marital status

0 Never-married

1 Married-civ-spouse

2 Married-civ-spouse

3 Married-civ-spouse

4 Married-civ-spouse

... ..

33869 Never-married

33870 Widowed

33871 Married-civ-spouse

33872 Divorced

33873 Married-civ-spouse

## inverse\_transform

```
[72] ohe.inverse_transform(marital_1hot[0:5])
```

ใช้ inverse\_transform เพื่อ  
แปลงค่าที่ one-hot แล้ว  
กลับให้เป็นหมวดหมู่เดิม

```
y([' Never-married',  
  ' Married-civ-spouse',  
  ' Married-civ-spouse',  
  ' Married-civ-spouse',  
  ' Married-civ-spouse'], dtype=object)
```

- OneHotEncoder จะทำ one-hot encoding ให้กับทุกคอลัมน์ของตาราง
- ตารางที่มีคอลัมน์ที่เป็นหมวดหมู่กับตัวเลขปนกันจะต้องใช้วิธีที่ซับซ้อนขึ้น
  - เลือกแต่คอลัมน์ที่เป็นหมวดหมู่มาทำ one-hot encoding แล้วนำตารางที่ transform แล้วมาต่อกับตารางที่มีแต่คอลัมน์ที่เป็นตัวเลข
  - ใช้ make\_column\_transformer เพื่อระบุว่าจะปรับคอลัมน์ยังไง

## make\_column\_transformer

import จาก  
**sklearn.compose**

```
[62] from sklearn.compose import make_column_transformer
```

ระบุ tuple ของ  
(**transformer**, ชื่อคอลัมน์)  
ทุกคู่ที่จะใช้ คั่นด้วย ,  
(**comma**)

สร้าง  
transformer

```
[63] transformer = make_column_transformer(  
    ( OneHotEncoder(), ['marital status', 'sex'] ),  
    ( MinMaxScaler(), ['age', 'edu num', 'captial-gain',  
                        'capital-loss', 'hours-per-week'] )  
)
```

```
[61] X_with_cat = df[ ['age', 'edu num', 'marital status', 'sex',  
                      'captial-gain', 'capital-loss', 'hours-per-week'] ]  
X_with_cat.head()
```

	age	edu num	marital status	sex	captial-gain	capital-loss	hours-per-week
0	39	13	Never-married	Male	2174	0	40
1	37	14	Married-civ-spouse	Female	0	0	40
2	42	13	Married-civ-spouse	Male	5178	0	40

## make\_column\_transformer

```
[64] X_transformed = transformer.fit_transform(X_with_cat)  
X_transformed
```

```
array([[0.          , 0.          , 0.          , ..., 0.02174022, 0.          ,  
        0.39795918],  
       [0.          , 0.          , 1.          , ..., 0.          , 0.          ,  
        0.39795918],  
       [0.          , 0.          , 1.          , ..., 0.05178052, 0.          ,  
        0.39795918],
```

fit ด้วยตารางที่มีคอลัมน์ตามที่กำหนด แล้ว transform

```
[65] X_transformed.min(axis=0), X_transformed.max(axis=0)
```

```
(array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),  
 array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]))
```

ลองเช็คค่า min, max ของตารางที่ transform แล้ว

# ใช้ตารางที่ one-hot และ MinMaxScaler แล้ว

```
[66] grid_search_transformed = GridSearchCV(knn_search
      , param_grid={'n_neighbors': (9,11,13,15,17,19,21,23,25)}
      , cv=5)
grid_search_transformed.fit(X_transformed, y)
grid_search_transformed.best_params_, grid_search_transformed.best_score_

({'n_neighbors': 19}, 0.839788625273321)
```

อย่าลืมเลือกคอลัมน์ของข้อมูลชุด  
test ให้ตรงกับชุด train

```
[67] unseen_X_with_cat = unseen_data[ ['age', 'edu num', 'marital status', 'sex'
      , 'captial-gain', 'capital-loss', 'hours-per-week'] ]
```

อย่าลืม transform ข้อมูลด้วย  
transformer ตัวที่เรียนมาแล้ว

```
[68] grid_search_transformed.score(transformer.transform(unseen_X_with_cat), unseen_y)
```

0.842035217035217



```
[88] unseen_y.value_counts()
```

```
No      7395  
Yes     2373  
Name: label, dtype: int64
```

## Other metrics

import มาตรวัดที่ต้องการจาก  
sklearn.metrics

```
[79] from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
[81] predicted_y = grid_search_transformed.predict(transformer.transform(unseen_X_with_cat))  
accuracy = accuracy_score(unseen_y, predicted_y)  
precision = precision_score(unseen_y, predicted_y, pos_label="Yes")  
recall = recall_score(unseen_y, predicted_y, pos_label="Yes")  
f1 = f1_score(unseen_y, predicted_y, pos_label="Yes")  
accuracy, precision, recall, f1
```

ใส่ y ที่เป็นข้อเท็จจริง  
ก่อน ตามด้วย y ที่  
ทำนาย

ระบุ class ที่ต้องการวัด precision,  
recall, f1 ด้วย option **pos\_label**

```
(0.842035217035217, 0.7214514407684098, 0.5697429414243573, 0.6366847186249117)
```

# Confusion matrix

```
[91] from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

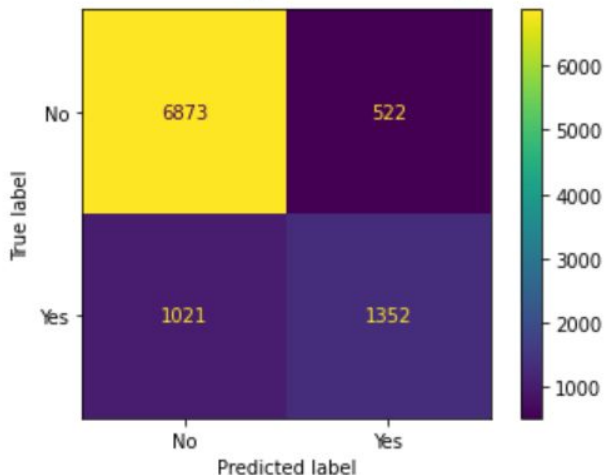
```
[92] confusion_matrix(unseen_y, predicted_y)
```

```
array([[6873,  522],  
       [1021, 1352]])
```

ใส่  $y$  ที่เป็นข้อเท็จจริง  
ก่อน ตามด้วย  $y$  ที่  
ทำนาย

```
[93] ConfusionMatrixDisplay.from_predictions(unseen_y, predicted_y)
```

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7f2c>



ใช้ method  
**from\_predictions** ใน  
**ConfusionMatrixDisplay**  
เพื่อสร้างแผนภาพ



## GridSearchCV for other metrics

- สามารถส่ง parameter ชื่อ scoring ให้กับ GridSearchCV เพื่อให้หา hyperparameters ที่ได้คะแนนตามที่เราต้องการมากที่สุดได้ (แทน accuracy)
- ควรเปลี่ยน class ให้เป็น 0,1 ก่อน โดยให้ class ที่เราสนใจมีค่าเป็น 1

```
[108] label = y.replace({'Yes':1, 'No':0})  
label
```

0	0
1	0
2	1
3	1
4	1

	..
33869	0
33870	0
33871	0
33872	0
33873	1

Name: label, Length: 33874, dtype: int64

เปลี่ยน Yes เป็น 1 และ No เป็น 0 ด้วย method replace ของ pandas

```
[109] unseen_label = unseen_y.replace({'Yes':1, 'No':0})
```

## GridSearchCV for recall

```
[111] grid_search_recall = GridSearchCV(knn_search
    , param_grid={'n_neighbors': (3,5,7,9,11,13,15,17,19,21,23,25)}
    , cv=5
    , scoring='recall')
grid_search_recall.fit(X_transformed, label)
grid_search_recall.best_params_, grid_search_recall.best_score_

({ 'n_neighbors': 3}, 0.5734395747286442)
```

เปลี่ยน scoring  
ให้เป็น recall

ทำนาย (อย่าลืม  
transform)

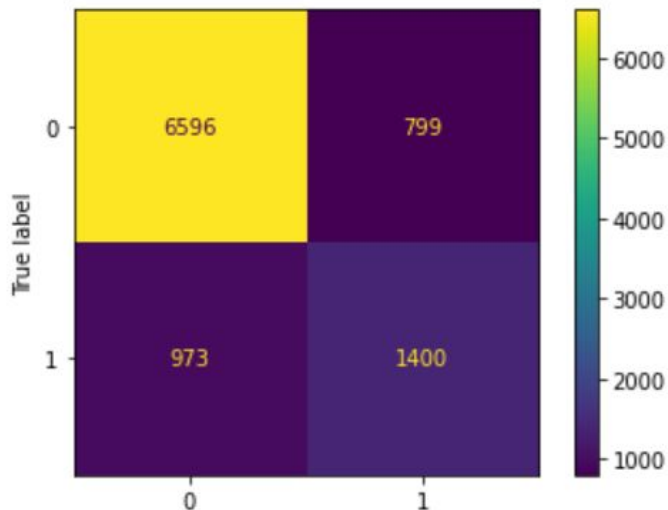
```
[112] predicted_label = grid_search_recall.predict(transformer.transform(unseen_X_with_cat))
```

```
[114] recall_score(unseen_label, predicted_label)
```

```
0.5899705014749262
```

```
[113] ConfusionMatrixDisplay.from_predictions(unseen_label, predicted_label)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f2cb5d5b8d0>
```



## KNN Fuzzy Classification

- Machine Learning algorithm หลายตัวมีความสามารถในการระบุความเชื่อมั่นในการทำนาย (ความน่าจะเป็น) ได้ด้วย เช่น KNN
- Algorithm ทั้งหมดที่ทำได้จะสามารถใช้ method ชื่อ **predict\_proba()** ในการระบุความน่าจะเป็นของแต่ละ class
- ระวัง ควรใช้ลำดับของ class จาก attribute `classes_`

```
grid_search_transformed.classes_  
array(['No', 'Yes'], dtype=object)
```

ใช้ลำดับ  
ของ class

```
grid_search_transformed.predict_proba(  
    transformer.transform(unseen_X_with_cat)  
)
```

```
array([[0.57894737, 0.42105263],  
       [0.47368421, 0.52631579],  
       [1.          , 0.          ],  
       ...,  
       [0.78947368, 0.21052632],  
       [0.84210526, 0.15789474],  
       [0.94736842, 0.05263158]])
```

ใช้  
**predict\_proba**

ความน่าจะเป็น  
ของ  
class No

ความน่าจะเป็น  
ของ class Yes

# สรุปกระบวนการสร้างโมเดล

1. เลือกและ/หรือสร้างคอลัมน์ที่จะนำมาเป็น feature ที่ใช้ในการสร้างโมเดล
2. ใช้ `make_column_transformer` เพื่อ
  - ปรับคอลัมน์หมวดหมู่ด้วย `OneHotEncoder`
  - ปรับคอลัมน์ตัวเลขด้วย `StandardScaler`, `MinMaxScaler`, หรือ `MaxAbsScaler`
3. ปรับ class ให้เป็น 0, 1 โดยให้ class ที่เราสนใจเป็น 1
4. ใช้ `GridSearchCV` (หรือตัวอื่น) ในการหา hyperparameters ที่ดีที่สุด กำหนด scoring ตามมาตรวัดที่ต้องการ
  - ถ้าผลไม่ดีเท่าที่ต้องการ กลับไปทำข้อ 1 ใหม่
5. ทดสอบโมเดลกับข้อมูลที่ไม่เคยเห็นแค่ครั้งเดียว
6. สร้างโมเดลด้วยคอลัมน์และ hyperparameters ที่ได้โดยใช้ข้อมูลทั้งหมด แล้วนำออกใช้งาน