
# 資料預處理
# Data Preprocessing

Sui-Hua Ho, PhD
Data Science, SCU

# 資料預處理步驟

| | | |
|---|---|---|
| 0 | Importing the Libraries | 導入函式庫 |
| | Importing Data Sets | 匯入資料集 |
| 1 | Missing Data | 缺失資料處理 |
| 2 | Categorical Data | 分類資料編碼 |
| 3 | Splitting the Dataset into the Training set and Test set | 將資料分成訓練集和測試集 |
| 4 | Feature Scaling | 特徵縮放 |

# 0 導入函式庫

# 匯入函式庫

- 語法：import 函式庫(模組) as 縮寫

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
```

# 匯入資料集

# 匯入資料集 (1/3)

- **Pandas 常見的讀檔方法**
  - **可歸類為三種：**
    - 讀取 csv 檔案的 pd.read_csv (“檔案路徑名稱.csv”)
    - 讀取 txt 檔案的 pd.read_table (“檔案路徑名稱.txt”)
    - 讀取 xlsx 檔案的 pd.read_excel (“檔案路徑名稱.xlsx”)
  - **將檔案路徑輸入括號中、加上雙引號、載入檔案**

# 匯入資料集 (2/3)

- 使用範例：
  - df = pd.read_csv (“C:/test/data.csv”)
    df
- 在這一段 code 中我們做兩件事：
  - 載入資料：pd.read_csv(“C:/test/data.csv”)載入後即是DataFrame格式資料
  - 呼喚變數：呼喚變數「df」叫出資料，再進行任何後續處理

# 匯入資料集 (3/3)

- 需注意以下兩點：
  - 路徑名稱中若有「\」出現，需要將它替換為「/」或是「\」，否則會找不到檔案路徑，也可以在引號前面加上「r」，表示將全部「\」反轉為「/」，如 pd.read_csv ( r“C:\test\data.csv” )
  - Pandas 讀檔預設支援「utf-8」格式，建議先將檔案轉換至「utf-8」格式，否則可能會讀取失敗。
- 若欲讀取的資料集與目前正在編輯的Notebook檔案置於同一資料夾內，使用pd.read_csv讀取時，僅需輸入資料集檔案名稱

In [2]:

```
dataset = pd.read_csv('Data.csv')
dataset #此為Pandas的二維資料資料結構dataframe
# print(type(dataset))
```

Out[2]:

| | Country | Age | Salary | Purchased |
|---|---|---|---|---|
| 0 | France | 44.0 | 72000.0 | No |
| 1 | Spain | 27.0 | 48000.0 | Yes |
| 2 | Germany | 30.0 | 54000.0 | No |
| 3 | Spain | 38.0 | 61000.0 | No |
| 4 | Germany | 40.0 | NaN | Yes |
| 5 | France | 35.0 | 58000.0 | Yes |
| 6 | Spain | NaN | 52000.0 | No |
| 7 | France | 48.0 | 79000.0 | Yes |
| 8 | Germany | 50.0 | 83000.0 | No |
| 9 | France | 37.0 | 67000.0 | Yes |

# 將資料集分割成特徵與標籤

# 將資料集分割成特徵與標籤 (1/2)


- 輸入X是具有特徵欄和樣本列的矩陣，而輸出y是具有樣本元素的列向量
- 因為遵循在數學和相關領域中慣例，使用大寫字母命名矩陣和使用小寫字母向量命名的慣例，X必須為大寫字母，而y必須為小寫字母

# 將資料集分割成特徵與標籤 (2/2)


- 在進行資料預處理和建立機器學習模型時，Numpy陣列(array)是最方便的工作格式，因此將資料集拆分成自變數(特徵集)X與應變數y時，會以.value方法dataframe將轉換為array

In [3]:

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [4]:
```
x
```

Out[4]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [5]:
```
y
```

Out[5]:
```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

# 1 缺失資料處理

In [6]:

```
from sklearn.impute import SimpleImputer #從 函式庫.模組 匯入類別
imputer = SimpleImputer(missing_values=np.nan, strategy='mean') #創建一個SimpleImputer類的物件
imputer.fit(X[:, 1:3]) #計算替換數值(平均值)[替代資料集範圍]
X[:, 1:3] = imputer.transform(X[:, 1:3]) #以計算後之平均數取代原數據集中的缺失值[替代資料集範圍]
X
```

Out[6]:

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

# 2 分類資料編碼

# 虛擬編碼(Dummy Encoding)

| Country |
|---|
| France |
| Spain |
| Germany |
| Spain |
| Germany |
| France |
| Spain |
| France |
| Germany |
| France |

| France | Spain | Germany |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |
| 1 | 0 | 0 |

# 編碼自變數

In [7]:

```
from sklearn.compose import ColumnTransformer #從 函式庫.模組 匯入類
from sklearn.preprocessing import OneHotEncoder #從 函式庫.模組 匯入類

#創建一個ColumnTransformer類的物件
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
X = np.array(ct.fit_transform(X)) #以編碼取代原數據集中的分類資料
```

# 編碼自變數

```
In [8]: x
```

```
Out[8]: array([[1.0, 0.0, 0.0, 44.0, 72000.0],
               [0.0, 0.0, 1.0, 27.0, 48000.0],
               [0.0, 1.0, 0.0, 30.0, 54000.0],
               [0.0, 0.0, 1.0, 38.0, 61000.0],
               [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
               [1.0, 0.0, 0.0, 35.0, 58000.0],
               [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
               [1.0, 0.0, 0.0, 48.0, 79000.0],
               [0.0, 1.0, 0.0, 50.0, 83000.0],
               [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

# 編碼應變數

| Country | Age | Salary | Purchased |
|---|---|---|---|
| France | 44 | 72000 | No |
| Spain | 27 | 48000 | Yes |
| Germany | 30 | 54000 | No |
| Spain | 38 | 61000 | No |
| Germany | 40 | | Yes |
| France | 35 | 58000 | Yes |
| Spain | | 52000 | No |
| France | 48 | 79000 | Yes |
| Germany | 50 | 83000 | No |
| France | 37 | 67000 | Yes |

# 編碼應變數

In [9]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

# 編碼應變數

In [10]:
```
y
```

Out[10]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

# 3 將資料集分成訓練集與測試集

In [11]:

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = None)

# random_state：可以為整數、RandomState例項或None，預設為None，若為None時，每次生成的資料都是隨機，
# 可能不一樣若為整數時，每次生成的資料都相同
```

In [12]:
```
X_train
```

Out[12]:
```
array([[0.0, 0.0, 1.0, 27.0, 48000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 83000.0],
       [1.0, 0.0, 0.0, 44.0, 72000.0],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 1.0, 0.0, 30.0, 54000.0],
       [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
       [0.0, 0.0, 1.0, 38.77777777777778, 52000.0]], dtype=object)
```

In [13]:
```
X_test
```

Out[13]:
```
array([[0.0, 0.0, 1.0, 38.0, 61000.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

In [14]:
```
y_train
```

Out[14]:
```
array([1, 1, 0, 0, 1, 0, 1, 0])
```

In [15]:
```
y_test
```

Out[15]:
```
array([0, 1])
```

# 4 特徵縮放

# 為什麼需要特徵縮放？

- 提升模型的收斂速度
- 提高模型的精準度

# 提升模型的收斂速度

- 基於梯度下降的演算法
  - 像線性回歸、邏輯回歸、神經網路等使用梯度下降作為優化技術的機器學習演算法，若能將資料標準化，則能減少梯度下降法的收斂時間

# 提升模型的收斂速度

- 梯度下降的公式：$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$
  - 公式中的特徵值X的存在將影響梯度下降的步長，特徵範圍的不同將導致每個特徵的步長不同
  - 為了確保梯度下降平穩地走向最小值，並且梯度下降的步長對所有特徵都以相同的速度更新，我們在將資料送入模型之前對其進行縮放
  - 擁有相似比例的特徵可以說明梯度下降更快地收斂到最小值

# 提高模型的精準度

- 基於距離的算法
  - 像KNN、K-means和SVM這樣的距離演算法受特徵範圍的影響最大。這是因為上述演算法使用資料點之間的距離來確定其相似性，當特徵間的範圍差距過大時，將可能導致分析的結果失真
  - 擁有相似比例的特徵可以說明梯度下降更快地收斂到最小值

# 基於樹的算法


- 另一方面，基於樹的演算法對特徵的規模相當不敏感
- 決策樹只是根據單一的特徵來分割一個節點，這種分割方式不受其他特徵的影響，因此，採用次類型演算法前，基本來說不需針對資料集進行特徵縮放

# 特徵縮放方法

| 標準化 (Standardization) | 正規化 (Normalization) |
| --- | --- |
| $x_{stand} = \frac{x - mean(x)}{StandardDeviation(x)}$ | $x_{norm} = \frac{x - min(x)}{max(x) - min(x)}$ |

# 資料的標準化 (1/6)

- 假設我們擷取前3筆資料計算資料點間特徵縮放前的歐基里德距離：

| No. | Country | Age | Salary | Purchased |
|---|---|---|---|---|
| 1 | France | 44 | 72000 | No |
| 2 | Spain | 27 | 48000 | Yes |
| 3 | Germany | 30 | 54000 | No |

$$d(1,2) = \sqrt{(44-27)^2 + (72000-48000)^2} = 24000.01$$
$$\cong \sqrt{(72000-48000)^2} = 24000$$

$$d(2,3) = \sqrt{(27-30)^2 + (48000-54000)^2} = 6000.001$$
$$\cong \sqrt{(48000-54000)^2} = 6000$$

# 資料的標準化 (2/6)

- 由上述計算結果可以發現這些距離幾乎都被值最大的特徵所影響
- 為避免這種狀況，則可以對資料作「標準化(Standardization)」，將每個特徵的重要程度變成一致

$$x_{stand} = \frac{x - mean(x)}{StandardDeviation(x)}$$

# 資料的標準化 (4/6)

- 假設我們擷取前3筆資料計算資料點間特徵縮放後的歐基里德距離：

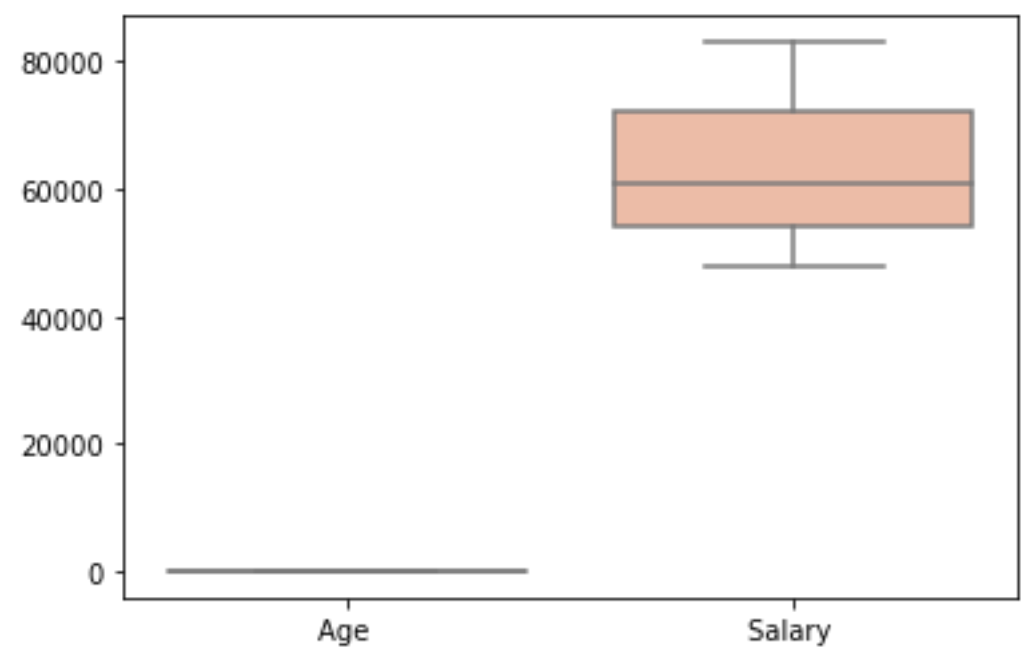
| No. | Country | Age | Salary | Purchased |
|---|---|---|---|---|
| 1 | France | 0.720 | 0.711 | No |
| 2 | Spain | -1.624 | -1.364 | Yes |
| 3 | Germany | -1.210 | -0.846 | No |

$$d(1,2) = \sqrt{(0.720 + 1.624)^2 + (0.711 + 1.364)^2} = 3.130$$
$$\neq \sqrt{(0.711 + 1.364)^2} = 2.075$$

$$d(2,3) = \sqrt{(-1.624 + 1.210)^2 + (-1.364 + 0.846)^2} = 0.664$$
$$\neq \sqrt{(-1.364 + 0.846)^2} = 0.519$$

# 資料的標準化 (5/6)

Before Feature Scaling




After Feature Scaling

# 資料的標準化 (5/6)

- 比較資料標準化前後的特徵距離計算結果，可發現經過標準化後得到的特徵距離不會受值特別大的特徵影響太多

[16]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:]) #以訓練集的資料計算平均數與標準差後將每一筆資料標準化，並取代原測試集內資料
X_test[:, 3:] = sc.transform(X_test[:, 3:]) #同樣以訓練集的縮方標準將每一筆資料標準化，並取代原測試集內資料
```

In [17]:
```
X_train
```

Out[17]:
```
array([[0.0, 0.0, 1.0, -1.5799974195763156, -1.291577832432],
       [1.0, 0.0, 0.0, 1.1627765165883106, 1.255066798299647],
       [0.0, 1.0, 0.0, 1.4239930819373225, 1.5836661054908274],
       [1.0, 0.0, 0.0, 0.6403433858902865, 0.6800180107150816],
       [1.0, 0.0, 0.0, -0.5351311581802676, -0.4700795644540 4925],
       [0.0, 1.0, 0.0, -1.1881725715527975, -0.7986788716452295],
       [0.0, 1.0, 0.0, 0.11791025519226246, 0.004563879266544702],
       [0.0, 0.0, 1.0, -0.04172209029880032, -0.9629785252408196]],
      dtype=object)
```

In [18]:
```
X_test
```

Out[18]:
```
array([[0.0, 0.0, 1.0, -0.14330631015674955, -0.22363008406066406],
       [1.0, 0.0, 0.0, -0.2739145928312555, 0.26926887672610633]],
      dtype=object)
```