Correlation & Regression 相關與迴歸

賴志宏 Chi-Hung Lai created date: 2019.05.07, Last modified date: 2019.11.09

機器學習(machine learning)分為

- 監督式學習 (supervised learning)
- 非監督式學習 (unsupervised learning)
- 半監督式學習 (semisupervised learning)
- 增強學習 (reinforcement learning)

- [機器學習學習地圖 (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

監督式學習的問題基本上分成兩類

- 迴歸問題：預測連續的回應資料，一種數值資料，我們可以預測商店的營業額、學生的身高和體重等。 常用演算法有：線性迴歸、SVR等。
- 分類問題：預測可分類的回應資料，這是一些有限集合，我們可以分類成男與女、成功與失敗、癌症分成第1~4期等。 常用演算法有：Logistic迴歸、決策樹、K鄰近演算法、CART、樸素貝葉斯等。

非監督式學習的問題基本上分成三類，如下所示：

- 關聯：找出各種現象同時出現的機率，稱為購物籃分析（Market-basket Analysis）
  - 當顧客購買米時，78%可能會同時購買雞蛋。
  - 常用演算法有：Apriori演算法等。
- 分群：將樣本分成相似的群組 *這是資料如何組成的問題，可以幫助區分群出哪些喜歡同一類電影的觀眾。
  - 常用演算法有：K-means演算法等。
- 降維：減少資料集中變數的個數，但是仍然保留主要資訊而不失真，
  - 我們通常是使用特徵提取和選擇方法來實作。
  - 常用演算法有：主成分分析演算法等。

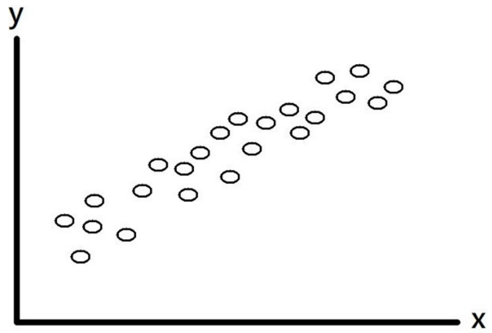
- Scikit-learn是scikits.learn的正式名稱，
- 一套支援Python 2和Python 3語言且完全免費的機器學習函數庫，
- 內建多種迴歸、分類和分群等機器學習演算法，
- 官方網址如下：[http://scikit-learn.org/stable/ (http://scikit-learn.org/stable/)](http://scikit-learn.org/stable/)

In [ ]:

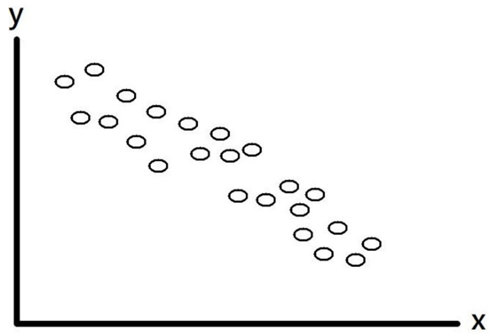
```

```

散佈圖的資料點可以幫助我們找出X和Y軸資料是正相關、負相關或無相關

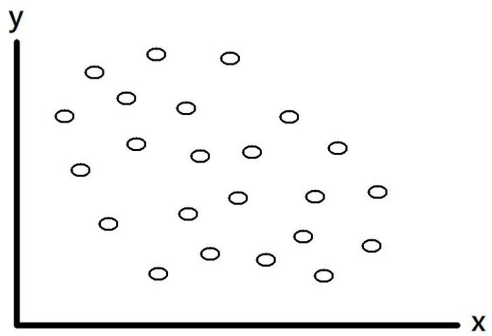
- 正相關（Positive Relation）：圖表顯示當一軸增加；同時另一軸也增加，資料排列成一條往右斜向上的直線， 例如：身高增加；體重也同時增加



- 負相關（Negative Relation）：圖表顯示當一軸增加；同時另一軸卻減少，資料排列成一條往右斜向下的直線， 例如：打手遊的時間增加；讀書的時間就會減少



- 無相關（No Relation）：圖表顯示的資料點十分分散，看不出有任何直線的趨勢，例如：學生身高和期中考成績



In [ ]:

```
# ch13_1_1 revised
# 手機使用時數和工作效率

import pandas as pd
%matplotlib inline  # pandas.plot() 不出現圖形的話，加上這行幾可
hours_phone_used = [0,0,0,1,1.3,1.5,2,2.2,2.6,3.2,4.1,4.4,4.4,5]
work_performance = [87,89,91,90,82,80,78,81,76,85,80,75,73,72]

df = pd.DataFrame({"hours_phone_used":hours_phone_used,
                   "work_performance":work_performance})

df.plot(kind="scatter", x="hours_phone_used", y="work_performance")

print(df)
```

共變數（Covariance）

- 用來測量2個隨機變數之間的關係，特別是指線性關係的強弱
- 變異數（Variance）可以告訴我們單一變數的離散程度
- 共變異數多了「共」，可以呈現2個變數一起的離散程度。

$$\text{共變異數} S_{xy} = \frac{(x_1 - \bar{x})(y_1 - \bar{y}) + (x_2 - \bar{x})(y_2 - \bar{y}) + \ldots + (x_n - \bar{x})(y_n - \bar{y})}{n}$$

- 共變異數的判斷原則如下：
  - 負相關：共變異數值小於0是負相關
  - 正相關：共變異數值大於0是正相關
  - 無相關：共變異數值約等於0，就是無相關

In [ ]:

```
# ch13_1_2

import numpy as np

hours_phone_used = [0,0,0,1,1.3,1.5,2,2.2,2.6,3.2,4.1,4.4,4.4,5]
work_performance = [87,89,91,90,82,80,78,81,76,85,80,75,73,72]

x = np.array(hours_phone_used)
y = np.array(work_performance)
n = len(x)
x_mean = x.mean()
y_mean = y.mean()
print("資料數:", n)
print("x平均:", x_mean)
print("y平均:", y_mean)

diff = (x-x_mean)*(y-y_mean)
print("x偏差*y偏差和:", diff.sum())
covar = diff.sum()/n
print("共變異數:", covar)
```

In [ ]:

```
import math
print(math.sqrt(15))
print(math.sqrt(3) * math.sqrt(5))
```

用共變數來看兩個變數之間的關係

- 限制：共變數的值和使用的單位有關，例如：體重和身高的關係， 身高的值使用公分，和使用公尺的值會不一樣
- 因此需要能夠有一個標準化的值，因此我們需要使用另一方式：以下介紹 "相關係數"

相關係數（Correlation Coefficient）

- 也稱為**皮爾森積差相關係數（Pearson Product Moment Correlation Coefficient）**
- 可以計算2個變數的線性相關性有多強
- 其值的範圍是-1~1之間
- 一種統計檢定方法，可以測量2個變數之間線性關係的強度和方向。
- 相關係數的公式是x和y的共變異數除以x和y的標準差
- 樣本的相關係數

$$\text{相關係數} r_{xy} = \frac{S_{xy}}{S_x S_y}$$

註：分子與分母共同除以n

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}$$

- 母體的相關係數，常用希臘小寫字母 ρ (rho) 作為代表符號

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

推導過程

母數的變異數與共變數的不偏估計數的N要減1

$$\rho = \frac{x\text{和}y\text{的共變異數}}{x\text{的標準差} \times y\text{的標準差}}$$

$$\text{共變異數(covariance)}: \operatorname{cov}(x, y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

$$\text{變異數(variance)}: \quad \operatorname{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)^2$$

$$\text{標準差(standard deviation)}: \quad std(x) = \sqrt{\operatorname{var}(x)}$$

$$\begin{aligned}
\rho &= \frac{\operatorname{cov}(x,y)}{std(x) \times std(y)} \\
&= \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\operatorname{var}(x)} \times \sqrt{\operatorname{var}(y)}} \\
&= \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)^2} \times \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \mu_y)^2}} \\
&= \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - \mu_x)^2} \times \sqrt{\sum_{i=1}^{n}(y_i - \mu_y)^2}} \\
&= \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - \mu_x)^2 \sum_{i=1}^{n}(y_i - \mu_y)^2}}
\end{aligned}$$

因果關係 vs. 相關性

- 相關性（Correlation）：量化相關性的值範圍在-1~1之間，即相關係數，我們可以使用相關係 數的值來測量2個變數的走勢是如何相關和其強度，
    - 例如：相關係數的值接近1，表示1個變數增加；另一個變數也增加，接近-1，表示1個變數增加 ；另一個變數減少。
- 因果關係（Causation）：一個變數真的影響另一個變數，也就是說，一個變數真的可以決定另 一個變數的值。

- 如果2個變數有因果關係，表示一定有相關性；
- 反之，有相關性，並不表示2個變數之間擁有因果關係

In [ ]:

```
# ch13_1_3  相關係數的算法
# pandas 的 corr()函式可以計算每一個欄位之間的相關係數

import numpy as np
import pandas as pd

hours_phone_used = [0,0,0,1,1.3,1.5,2,2.2,2.6,3.2,4.1,4.4,4.4,5]
work_performance = [87,89,91,90,82,80,78,81,76,85,80,75,73,72]

x = np.array(hours_phone_used)
y = np.array(work_performance)
n = len(x)
x_mean = x.mean()
y_mean = y.mean()

diff = (x-x_mean)*(y-y_mean)
covar = diff.sum()/n
print("共變異數:", covar)

corr = covar/(x.std()*y.std())
print("相關係數:", corr)

df = pd.DataFrame({"hours_phone_used":hours_phone_used,
                   "work_performance":work_performance})
print(df.corr())   # pandas 的 corr()函式可以計算每一個欄位之間的相關係數
df.corr().to_html("Ch13_1_3.html")
```

相關的統計檢定

In [ ]:

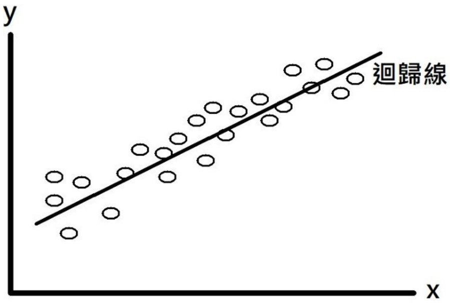
```
### SPSS 操作步驟
* 分析 / 相關 / 雙變數
* 輸入兩個變項
* 勾選所需的相關係數類型和其它設定 （顯著性訊號表示：當相關係數有統計意義時，以*表示）
* 勾選選項中的統計量 （平均數與標準差）

結果：看Person相關係數及p 值
```

In [ ]:

```
# 資料來自邱皓政 量化研究法二 p.15-38

# 由家庭人口數預測家庭開銷

X = np.array([3, 5, 4, 6, 2, 4, 5, 8, 7, 5])
y = np.array ([15000, 34000, 22000, 36300, 16000, 25000, 30000, 45000, 44000, 39000])
```

斜率與截距

- 迴歸線的斜率是正值：迴歸線往右斜向上的斜率是正值（見上述圖例），x和y的關係是正相關，x值增加；同時y值也會增加。
- 迴歸線的斜率是負值：迴歸線往右斜向下的斜率是負值，x和y的關係是負 相關，x值減少；同時y值也會減少。



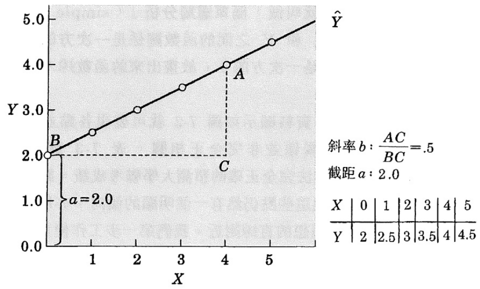


圖 8-1　斜線 Y 的斜率 (b) 和截距 (a)

簡單線性迴歸（Simple Linear Regression）

- 是一種最簡單的線性迴歸分析法，只有1個解釋變數，這條線可以使用數學的一次方程式來表示， 也就是2個變數之間關係的數學公式，如下所示：

$$迴歸方程式 y = a + bX$$

- 公式的變數y是反應變數（Response，或稱應變數），X是解釋變數（Explanatory，或稱自變數），a是截距（Intercept），b是迴歸係數(Regression coefficients)
- 當從訓練資料找出截距a和迴歸係數b的值後，就完成預測公式。我們只需使用新值X，即可透過公式 來預測y值。

In [ ]:

```

```

使用最小評方法(least square)求出迴歸線:

*　一條斜線，各點至此線之平行於　Y軸的距離的平方為為最小

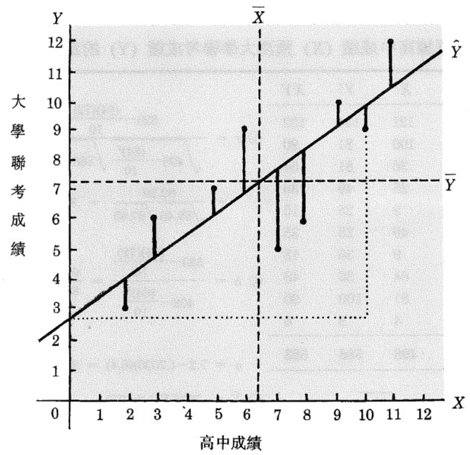


圖 8-3　由高中成績預測大學聯考成績的最適合線

根據 X變數預測 Y變數時，截距a和斜率b的值

$$b_{Y\cdot X} = \frac{\Sigma XY - \frac{\Sigma X \Sigma Y}{N}}{\Sigma X^2 - \frac{(\Sigma X)^2}{N}}$$

$$= \frac{\Sigma xy}{\Sigma x^2} = \frac{CP}{SS_X} = \frac{\frac{\Sigma xy}{N}}{\frac{\Sigma x^2}{N}} = \frac{C_{XY}}{S_X^2}$$

$$a_{Y\cdot X} = \overline{Y} - b_{Y\cdot X}\overline{X}$$

In [ ]:

```
# by Lai


import numpy as np
x = np.array([29, 28, 34, 31,
                         25, 29, 32, 31,
                         24, 33, 25, 31,
                         26, 30])
y = np.array([7.7, 6.2, 9.3, 8.4,
                        5.9, 6.4, 8.0, 7.5,
                        5.8, 9.1, 5.1, 7.3,
                        6.5, 8.4])

# 繪圖
import matplotlib.pyplot as plt
plt.plot(x, y, 'go')
plt.show()
```

In [ ]:

```
# by Lai
# 計算迴歸係數（斜率）、截距、並預測
# 自己算

import numpy as np
x = np.array([29, 28, 34, 31,
                         25, 29, 32, 31,
                         24, 33, 25, 31,
                         26, 30])
y = np.array([7.7, 6.2, 9.3, 8.4,
                        5.9, 6.4, 8.0, 7.5,
                        5.8, 9.1, 5.1, 7.3,
                        6.5, 8.4])
n = len(x)
x_mean = x.mean()
y_mean = y.mean()

diff = (x-x_mean)*(y-y_mean)
covar = diff.sum()/n
print("共變異數:", covar)

b= covar/(x.std()** 2)
print("斜率（迴歸係數）：", b)

intercept = y_mean - b * x_mean
print("截距 intercept:", intercept)

print("迴歸線：Y= {}X + {}".format(b, intercept) )

x1 = np.array([26, 30])
y_predict = b * x1 + intercept
print('\n[26, 30]的y_predict:', y_predict)

# 繪圖
import matplotlib.pyplot as plt
x_new = np.linspace(x.min(), x.max(), 100)
y_new  = b * x_new + intercept
# plt.plot(x, y, 'go', x_new, y_new)
plt.plot(x, y, 'go')
plt.plot(x_new, y_new)
plt.plot(x1, y_predict, 'ro')
plt.show()
```

In [ ]:

```
# by Lai
# 計算迴歸係數（斜率）、截距、並預測
# 自己算

import numpy as np
x = np.array([11,10, 6, 5, 3, 7, 3, 8, 9, 2])
y = np.array([12, 9, 9, 7, 5, 5, 6, 6, 10, 3])
n = len(x)
x_mean = x.mean()
y_mean = y.mean()

diff = (x-x_mean)*(y-y_mean)
covar = diff.sum()/n
print("共變異數:", covar)

b= covar/(x.std()** 2)
print("斜率（迴歸係數）：", b)

intercept = y_mean - b * x_mean
print("intercept:", intercept)

print("迴歸線：Y= {}X + {}".format(b, intercept) )

x = 4.0
y_predict = b * x + intercept
print('y_predict:', y_predict)
```

In [ ]:

```
# ch15_2_2
# 計算迴歸係數（斜率）、截距、並預測
# 使用當日氣溫來預測當日的業積
# use object: LinearRegression

import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

temperatures = np.array([29, 28, 34, 31,
                         25, 29, 32, 31,
                         24, 33, 25, 31,
                         26, 30])
drink_sales = np.array([7.7, 6.2, 9.3, 8.4,
                        5.9, 6.4, 8.0, 7.5,
                        5.8, 9.1, 5.1, 7.3,
                        6.5, 8.4])
X = pd.DataFrame(temperatures, columns=["Temperature"])
target = pd.DataFrame(drink_sales, columns=["Drink_Sales"])
y = target["Drink_Sales"]

lm = LinearRegression()
lm.fit(X, y)    # train predict model
print("迴歸係數（斜率）:", lm.coef_)
print("截距:", lm.intercept_ )
# 預測氣溫26, 30度的業績
new_temperatures = pd.DataFrame(np.array([26, 30]))
predicted_sales = lm.predict(new_temperatures)
print(predicted_sales)
```

In [ ]:

```
# ch15_2_2a
# 天氣預測營業額（千元）
# 繪出圖形

import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

temperatures = np.array([29, 28, 34, 31,
                         25, 29, 32, 31,
                         24, 33, 25, 31,
                         26, 30])
drink_sales = np.array([7.7, 6.2, 9.3, 8.4,
                        5.9, 6.4, 8.0, 7.5,
                        5.8, 9.1, 5.1, 7.3,
                        6.5, 8.4])
X = pd.DataFrame(temperatures, columns=["Temperature"])
target = pd.DataFrame(drink_sales, columns=["Drink_Sales"])
y = target["Drink_Sales"]
lm = LinearRegression()
lm.fit(X, y)
# 預測氣溫26, 30度的業績
new_temperatures = pd.DataFrame(np.array([26, 30]))
predicted_sales = lm.predict(new_temperatures)
print(predicted_sales)

plt.scatter(temperatures, drink_sales)  # 繪點
regression_sales = lm.predict(X)
plt.plot(temperatures, regression_sales, color="blue")
plt.plot(new_temperatures, predicted_sales,
         color="red", marker="o", markersize=10)
plt.show()
```

In [ ]:

```python
# ch15_2_2b revised
# 身高預測體重

import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

heights = np.array([147.9, 163.5, 159.8, 155.1,
                    163.3, 158.7, 172.0, 161.2,
                    153.9, 161.6])
weights = np.array([41.7, 60.2, 47.0, 53.2,
                    48.3, 55.2, 58.5, 49.0,
                    46.7, 52.5])
X = pd.DataFrame(heights, columns=["Height"])
target = pd.DataFrame(weights, columns=["Weight"])  # 資料型態為 DataFrame
y = target["Weight"]        # 資料型態為 Series, 也可以直接使用 y = pd.Series(weights)
lm = LinearRegression()
lm.fit(X, y)                 # 也可以直接使用 lm.fit(x, weights)
print("迴歸係數:", lm.coef_)
print("截距:", lm.intercept_ )

# 預測身高150, 160, 170的體重
new_heights = pd.DataFrame(np.array([150, 160, 170]))
predicted_weights = lm.predict(new_heights)
print(predicted_weights)

plt.scatter(heights, weights)  # 繪點
regression_weights = lm.predict(X)
plt.plot(heights, regression_weights, color="blue")
plt.plot(new_heights, predicted_weights,
         color="red", marker="o", markersize=10)
plt.show()
```

迴歸是否有達到統計意義 （統計學的議題）

SPSS 作法：

- 分析 / 迴歸方法 / 線性
  - 輸入自變項與依變項
  - 進入統計量勾選各種統計量（估計值、共變異數矩陣、描述性統計量、模式適合度）
  - 按確定

$$SS_t = \Sigma(Y_i - \bar{Y})^2 = \Sigma(Y' - \bar{Y})^2 + \Sigma(Y_i - Y')^2 = SS_{reg} + SS_e$$

$$F_{(p,N-p-1)} = \frac{MS_{reg}}{MS_e} = \frac{SS_{reg}/df_{reg}}{SS_e/df_e} = \frac{SS_{reg}/p}{SS_e/N-p-1} \quad (15\text{-}19)$$

P為自變項的數目

迴歸模型的變異數分析摘要表

變異來源	SS	df	MS	F
迴歸效果	SS_r	p	SS_r/df_r	MS_r/MS_e
誤差	SS_e	N-p-1	SS_e/df_e	
全體	SS_t	N-1		

	平方和	自由度	平均平方和	F 檢定	顯著性
迴歸	941242384.1	1	941242384.1	61.48	.000
殘差	122478615.9	8	15309827.0		
總和	1063721000	9			

迴歸模型的變異數分析摘要表可參考邱皓政，量化統計二，p.15-33

In [ ]:

```
# 資料來自邱皓政 量化研究法二 p.15-45
# 由家庭人口數預測家庭開銷
# 資料來自邱皓政 量化研究法二 p.15-45
# 由家庭人口數預測家庭開銷

import pandas as pd
import numpy as np

import statsmodels.api as sm

X = np.array([3, 5, 4, 6, 2, 4, 5, 8, 7, 5])
y = np.array ([15000, 34000, 22000, 36300, 16000, 25000, 30000, 45000, 44000, 39000])

# model = sm.OLS(y,X)
model = sm.OLS(y, sm.add_constant(X))   # statsmodels.OLS 不會假設迴歸模型有常數項，所以要自己加入常數（截距）
results = model.fit()
print("截距與斜率:", results.params) # 顯示截距與斜率
print()
print(results.summary())
import pandas as pd
import numpy as np

import statsmodels.api as sm

X = np.array([3, 5, 4, 6, 2, 4, 5, 8, 7, 5])
y = np.array ([15000, 34000, 22000, 36300, 16000, 25000, 30000, 45000, 44000, 39000])

# model = sm.OLS(y,X)
model = sm.OLS(y, sm.add_constant(X))   # statsmodels.OLS 不會假設迴歸模型有常數項，所以要自己加入常數（截距）
results = model.fit()
print("截距與斜率:", results.params) # 顯示截距與斜率
print()
print(results.summary())
```

結果解釋：

- R-squared: 0.885 表示家庭人口數可以解釋每月開銷的88.5%的變異量
- F(1, 8) = 61.48, p=5.04e-05 < .05 顯示迴歸模型具有統計意義 (註：迴歸模型有統計意義之後，再看每個自變項的係數是否有統計意義，自變項有可能不只一個)
- 係數估計的結果指出：斜率為 5706.9204，截距為 2666.0900，其 t = 7.841 , p value <.05， 表示人口數對家庭開銷有預測效益（需拒絕虛無假設：迴歸係數= 0）
- 迴歸模型的變異數分析摘要表可參考邱皓政，量化統計二，p.15-33

線性複迴歸 （Linear Multiple Regression):或稱多元迴歸

* 多個解釋變數（自變數），一個反應變數（依變數）

In [ ]:

```python
# ch15_3_1
# 腰圍和身高預測體重

import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

waist_heights = np.array([[67,160], [68,165], [70,167],
                          [65,170], [80,165], [85,167],
                          [78,178], [79,182], [95,175],
                          [89,172]])
weights = np.array([50, 60, 65, 65,
                    70, 75, 80, 85,
                    90, 81])
X = pd.DataFrame(waist_heights, columns=["Waist", "Height"])
target = pd.DataFrame(weights, columns=["Weight"])
y = target["Weight"]
lm = LinearRegression()
lm.fit(X, y)
print("迴歸係數:", lm.coef_)
print("截距:", lm.intercept_ )

# 預測腰圍和身高[66,164],[82,172]的體重
new_waist_heights = pd.DataFrame(np.array([[66, 164],
                                           [82, 172]]))
predicted_weights = lm.predict(new_waist_heights)
print(predicted_weights)

print(type(target))
print(target)
print("\ny:", y, sep='\n')
print(type(y))
```

In [ ]:

```python
# ch15_3_1a
# 使用店面面積和車站距離來預測單月營業額

import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

area_dists = np.array([[10,80], [8,0], [8,200],
                       [5,200], [7,300], [8,230],
                       [7,40], [9,0], [6,330],
                       [9,180]])
sales = np.array([46.9, 36.6, 37.1, 20.8,
                  24.6, 29.7, 36.6, 43.6,
                  19.8, 36.4])
X = pd.DataFrame(area_dists, columns=["Area", "Distance"])
target = pd.DataFrame(sales, columns=["Sales"])
y = target["Sales"]
lm = LinearRegression()
lm.fit(X, y)
print("迴歸係數:", lm.coef_)
print("截距:", lm.intercept_ )

# 預測腰面積和距離[10,100]的營業額
new_area_dists = pd.DataFrame(np.array([[10, 100]]))
predicted_sales = lm.predict(new_area_dists)
print(predicted_sales)
```

實戰練習：波斯頓房價預測

In [ ]:

```
# ch15_3_2

from sklearn import datasets

boston = datasets.load_boston()      # 載入其它資料庫，如鳶尾花的方式：datasets.load_iris(), load_diabetes()
# data type is dictionaryz

print("keys:\n", boston.keys())
print("\ndata shape:\n", boston.data.shape)
print("\nfield name in data:\n", boston.feature_names)
print("\nDescription:", boston.DESCR)
```

In [12]:

```
# ch15_3_2a modified

import pandas as pd
from sklearn import datasets

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
print(X.head())

target = pd.DataFrame(boston.target, columns=["MEDV"])
print('\n','target:', '\n', target.head(), sep='\n')
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33


target:


   MEDV
0  24.0
1  21.6
2  34.7
3  33.4
4  36.2
```

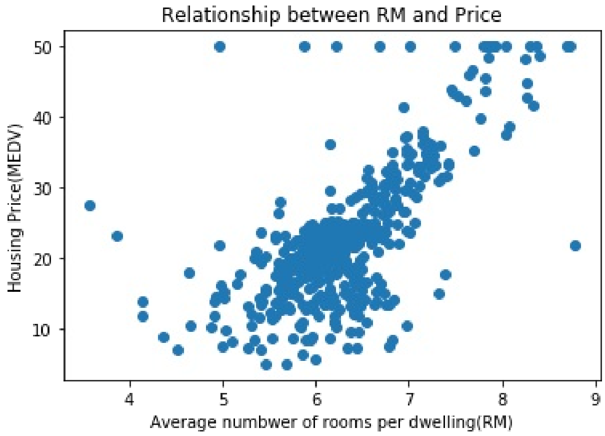
In [18]:

```python
# ch15_3_2b modified

import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target, columns=["MEDV"])
y = target["MEDV"]

lm = LinearRegression()
lm.fit(X, y)
print("迴歸係數:", lm.coef_)
print("截距:", lm.intercept_ )

coef = pd.DataFrame(boston.feature_names, columns=["features"])
coef["estimatedCoefficients"] = lm.coef_
print("\n迴歸係數:", coef, sep = '\n')

# The bigest regression coefficient is "RM"
```

```
迴歸係數: [-1.08011358e-01  4.64204584e-02  2.05586264e-02  2.68673382e+00
 -1.77666112e+01  3.80986521e+00  6.92224640e-04 -1.47556685e+00
  3.06049479e-01 -1.23345939e-02 -9.52747232e-01  9.31168327e-03
 -5.24758378e-01]
截距: 36.459488385089855

迴歸係數:
   features  estimatedCoefficients
0      CRIM              -0.108011
1        ZN               0.046420
2     INDUS               0.020559
3      CHAS               2.686734
4       NOX             -17.766611
5        RM               3.809865
6       AGE               0.000692
7       DIS              -1.475567
8       RAD               0.306049
9       TAX              -0.012335
10  PTRATIO              -0.952747
11        B               0.009312
12    LSTAT              -0.524758
```

In [20]:

```
# ch15_3_2b modified
# Picture of "Relationship between RM and Price"

import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target, columns=["MEDV"])
y = target["MEDV"]

lm = LinearRegression()
lm.fit(X, y)

plt.scatter(X.RM, y)
plt.xlabel("Average numbwer of rooms per dwelling(RM)")
plt.ylabel("Housing Price(MEDV)")
plt.title("Relationship between RM and Price")
plt.show()
```



In [17]:

```
# ch15_3_2c
# depic "Price vs Predicted Price"

import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target, columns=["MEDV"])
y = target["MEDV"]

lm = LinearRegression()
lm.fit(X, y)

predicted_price = lm.predict(X)
print(predicted_price[0:5])

plt.scatter(y, predicted_price)
plt.xlabel("Price")
plt.ylabel("Predicted Price")
plt.title("Price vs Predicted Price")
plt.show()
```

```
[30.00384338 25.02556238 30.56759672 28.60703649 27.94352423]
```

Training dataset & test dataset

XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size=0.33, random_state=5)

- test_size = 0.33
  - training data:67%, test data: 33%
- random_state=5
  - random seed number

In [22]:

```
# ch15_3_3
# Price vs Predicted Price in test dataset

import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target, columns=["MEDV"])
y = target["MEDV"]

XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size=0.33,
                                                random_state=5)
lm = LinearRegression()
lm.fit(XTrain, yTrain)

pred_test = lm.predict(XTest)

plt.scatter(yTest, pred_test)
plt.xlabel("Price")
plt.ylabel("Predicted Price")
plt.title("Price vs Predicted Price")
plt.show()
```

Performance of Prediction in Regression

- MSE(Mean Squared Error)
  - 預測時誤差的平方和的平均數
  - (y-predicted_price) ** 2
  - smaller is better
- R-squared (Cofficient of Determination) 決定係數：告訴我們資料是如何符合迴歸線
  - value: 0 ~ 1
  - biger is better
  - LinerRegression_object.score()

In [23]:

```python
# ch15_3_3a

import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target, columns=["MEDV"])
y = target["MEDV"]

XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size=0.33,
                                                random_state=5)
lm = LinearRegression()
lm.fit(XTrain, yTrain)

pred_train = lm.predict(XTrain)
pred_test = lm.predict(XTest)

MSE_train = np.mean((yTrain-pred_train)**2)
MSE_test = np.mean((yTest-pred_test)**2)
print("訓練資料的MSE:", MSE_train)
print("測試資料的MSE:", MSE_test)

print("訓練資料的R-squared:", lm.score(XTrain, yTrain))
print("測試資料的R-squared:", lm.score(XTest, yTest))
```

```
訓練資料的MSE: 19.54675847353467
測試資料的MSE: 28.530458765974686
訓練資料的R-squared: 0.7551332741779998
測試資料的R-squared: 0.6956551656111596
```

In [24]:

```python
# ch15_3_3b

import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LinearRegression

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target, columns=["MEDV"])
y = target["MEDV"]

lm = LinearRegression()
lm.fit(X, y)

predicted_price = lm.predict(X)
print(predicted_price[0:5])

MSE = np.mean((y-predicted_price)**2)
print("MSE:", MSE)
print("R-squared:", lm.score(X, y))
```

```
[30.00384338 25.02556238 30.56759672 28.60703649 27.94352423]
MSE: 21.894831181729213
R-squared: 0.7406426641094095
```

In [ ]:

```python
# ch15_3_4
# plot of residual 殘差圖 to highlight outliers (異常值)
# residual = y - predicted_y

import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

boston = datasets.load_boston()

X = pd.DataFrame(boston.data, columns=boston.feature_names)
target = pd.DataFrame(boston.target, columns=["MEDV"])
y = target["MEDV"]

XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size=0.33,
                                                random_state=5)
lm = LinearRegression()
lm.fit(XTrain, yTrain)

pred_train = lm.predict(XTrain)
pred_test = lm.predict(XTest)

plt.scatter(pred_train, pred_train-yTrain,
            c="b", s=40, alpha=0.5, label="Training Data")
plt.scatter(pred_test, pred_test-yTest,
            c="r", s=40, label="Test Data")
plt.hlines(y=0, xmin=0, xmax=50)
plt.title("Residual Plot")
plt.ylabel("Residual Value")
plt.legend()
plt.show()
```

邏輯迴歸 Logistic Regression

- 主要應用是二元性資料，例如：男或女、成功或失敗、真或假等
- Logistic迴歸和線性迴歸不同，它是在解決分類問題。
- Logistic迴歸的作法和線性迴歸相同，只不過其結果需要使用logistic函數 或稱sigmoid函數（即S函數）轉換成0~1之間的機率

In [ ]:

```python
# ch15_4_1

import numpy as np
import matplotlib.pyplot as plt

t = np.arange(-6, 6, 0.1)
S = 1/(1+(np.e**(-t)))

plt.plot(t, S)
plt.title("sigmoid function")
plt.show()
```

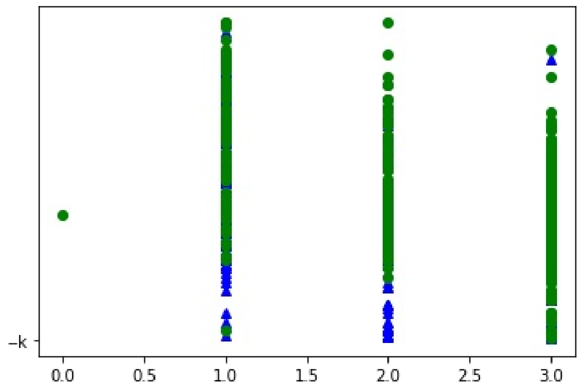
In [ ]:

```python
# ch15_4_2

import numpy as np
import matplotlib.pyplot as plt

t = np.arange(-6, 6, 0.1)
S = 1/(1+(np.e**(-t)))

plt.plot(t, S)
plt.title("sigmoid function")
plt.show()
```

In [34]:

```
# 區分預測值後，繪出圖形， Lai

import pandas as pd
import numpy as np
from sklearn import preprocessing, linear_model

import matplotlib.pyplot as plt


titanic = pd.read_csv("data/titanic.csv")
print(titanic.info())
# 將年齡的空值填入年齡的中位數
age_median = np.nanmedian(titanic["Age"])
new_age = np.where(titanic["Age"].isnull(),
                   age_median, titanic["Age"])
titanic["Age"] = new_age
# 轉換欄位值成為數值
label_encoder = preprocessing.LabelEncoder()
titanic["PClass"] = label_encoder.fit_transform(titanic["PClass"])

X = pd.DataFrame([titanic["PClass"],
                  titanic["SexCode"],
                  titanic["Age"]]).T
y = titanic["Survived"]

alive = titanic[titanic["Survived"] == 1]
dead = titanic[titanic["Survived"] == 0]
print(titanic["Survived"][:5])

# red dashes, blue squares and green triangles
plt.plot(alive["PClass"] , alive["Age"], "b^", "--k")
plt.plot(dead["PClass"] , dead["Age"], "go", "--k")
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1313 entries, 0 to 1312
Data columns (total 7 columns):
PassengerId    1313 non-null int64
Name           1313 non-null object
PClass         1313 non-null object
Age            756 non-null float64
Sex            1313 non-null object
Survived       1313 non-null int64
SexCode        1313 non-null int64
dtypes: float64(1), int64(3), object(3)
memory usage: 71.9+ KB
None
0    1
1    0
2    0
3    0
4    1
Name: Survived, dtype: int64
```



In [ ]:

```
# ch15_4_2a

import pandas as pd
import numpy as np
from sklearn import preprocessing, linear_model

titanic = pd.read_csv("titanic.csv")
print(titanic.info())
# 將年齡的空值填入年齡的中位數
age_median = np.nanmedian(titanic["Age"])
new_age = np.where(titanic["Age"].isnull(),
                   age_median, titanic["Age"])
titanic["Age"] = new_age
# 轉換欄位值成為數值
label_encoder = preprocessing.LabelEncoder()
encoded_class = label_encoder.fit_transform(titanic["PClass"])

X = pd.DataFrame([encoded_class,
                  titanic["SexCode"],
                  titanic["Age"]]).T
y = titanic["Survived"]

logistic = linear_model.LogisticRegression()
logistic.fit(X, y)

preds = logistic.predict(X)
print(pd.crosstab(preds, titanic["Survived"]))
pd.crosstab(preds, titanic["Survived"]).to_html("Ch15_4_2a.html")

print((804+265)/(804+185+59+265))
print(logistic.score(X, y))
```

In [ ]:

```
# ch15_4_2b

import pandas as pd
from sklearn import preprocessing, linear_model

titanic = pd.read_csv("titanic.csv")
print(titanic.info())
# 轉換欄位值成為數值
label_encoder = preprocessing.LabelEncoder()
encoded_class = label_encoder.fit_transform(titanic["PClass"])

X = pd.DataFrame([encoded_class,
                  titanic["SexCode"]]).T
y = titanic["Survived"]

logistic = linear_model.LogisticRegression()
logistic.fit(X, y)
print("迴歸係數:", logistic.coef_)
print("截距:", logistic.intercept_ )

preds = logistic.predict(X)
print(pd.crosstab(preds, titanic["Survived"]))
pd.crosstab(preds, titanic["Survived"]).to_html("Ch15_4_2b.html")

print((840+228)/(840+222+23+228))
print(logistic.score(X, y))
```

In [ ]:

```

```

參考資料:

- 陳允傑（2018）。Python資料科學與人工智慧應用實務 ch.13, 15
- 邱皓政 量化研究法二