



國立中山大學
National Sun Yat-sen University

Program in Interdisciplinary Studies
Sustainable Innovative Laboratory



Ch 12運用機器學習做分類 (classification) 預測及資料簡化

國立中山大學人文暨科技跨領域學士學位學程

楊政融老師

課程大綱

- 12-0 資料分類 (classification)
- 12-1 KNN (K 近鄰) 預測模型
- 12-2 邏輯斯 (Logistic) 迴歸模型
- 12-3 改善邏輯斯迴歸模型
- 12-4 主成分分析 (PCA)：減少需分析的變數



12-0 資料分類 (classification)

- 第11章介紹如何使用scikit-learn機器學習套件來做多元線性迴歸分析。
- 此套件目的並不是用來預測數值，而是要預測資料的**分類(classification)**。
- 舉例來說，scikit-learn內建一個資料集是搜集有關義大利某區三種葡萄品種Barolo、Grignolino、Barbera所釀的酒，總共178筆資料，內有13個自變數如右。

欄位	意義	欄位	意義
Alcohol	酒精	Nonflavanoid phenols	非黃酮類化合物
Malic acid	蘋果酸	Proanthocyanins	原花青素
Ash	灰分	Color intensity	色澤深度
Alcalinity of ash	灰分鹼度	Hue	色調
Magnesium	鎂	OD280/OD315 of diluted wines	葡萄酒稀釋後的吸光度比例
Total phenols	苯酚	Proline	脯胺酸
		Flavanoids	黃酮類化合物



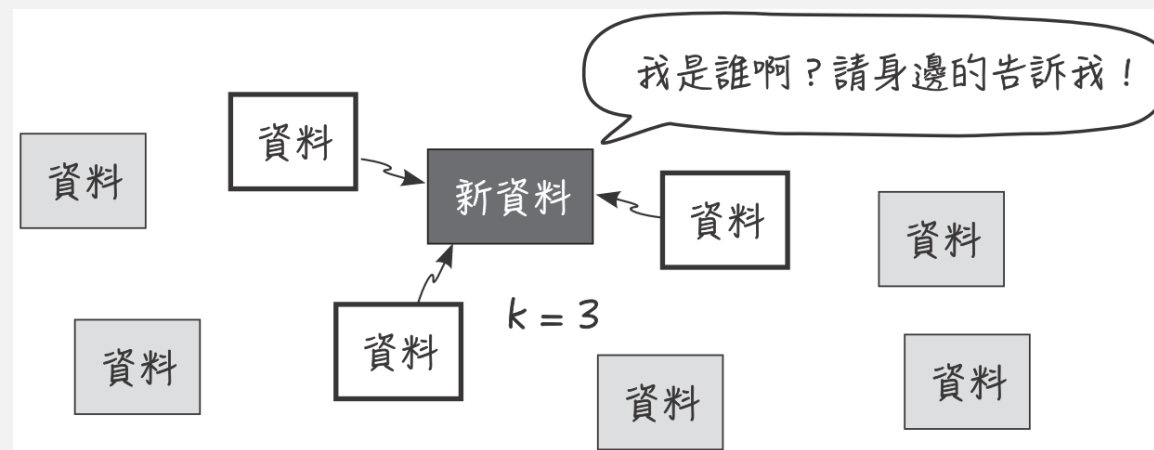
12-0 資料分類 (classification)

- 用機器學習來預測資料的分類
- 對分類演算法而言，目標變數又稱為**標籤(labels)**。
- 以葡萄酒品種這案例來說，三種葡萄酒的分類就是0、1、2。
- 我們想要做的事情就是利用這13種特徵資料來訓練機器學習模型。
- 利用這個模型就可以讓電腦看到一筆新資料時就能夠認定出這是用哪種葡萄釀的酒。
- scikit-learn套件提供了許多**分類(classification)**演算法或**分類器(classifier)**，以下將簡述最常用的幾種。



12-1 KNN (K 近鄰) 預測模型

- 首先介紹稱為KNN或K近鄰(K-nearest neighbors)，這是機器學習中原理最簡單且好用的一種分類器。
- 當KNN要判斷新資料的分類時，得先指定一個K值(鄰近數量)。
- 以這一題葡萄酒分類案例來說，我們假設是3，當我們把新的葡萄酒資料丟給KNN時，它就會從資料集中挑選3筆最相近的，再用它們來判斷新資料屬於哪一類。



12-1 KNN (K 近鄰) 預測模型

- 那KNN要如何知道這3瓶酒和我們測試的最接近呢？
- 原理就是用每瓶酒的13個自變數資料來比對計算。
- 然後選出3瓶最接近的，在看他們的標籤(label)標住的是什麼酒。
- 假如這3筆最接近的資料中，有兩筆是用Barolo葡萄所釀造的酒(分類0)，另一筆則為Barbera葡萄所釀造的酒(分類2)，那麼KNN就會判定新資料最有可能代表分類0。



12-1 KNN (K 近鄰) 預測模型

- 知識補充
- 在葡萄酒這案例中，葡萄酒分類是由人類是先標記好的，讓機器學習在訓練時當作參考。
- 就好比拿考古題及對應的答案來給學生練習，讓他們將來應付正式考試的能力。
- 因此KNN也被稱為是一種監督式學習(supervised learning)演算法。
- 這類機器學習類型的特質就是需要人類監督(提供標籤)方能完成訓練。



12-1 KNN (K 近鄰) 預測模型

- 匯入並分割資料集
- KNN模型的建立先從scikit-learn匯入葡萄酒資料集，並照11章的方式把80%分割成訓練集、20%當作測試集。

IN

```
from sklearn import datasets
from sklearn.model_selection import train_test_split

data, target = datasets.load_wine(return_X_y=True) ← 載入葡萄酒資料集
data_train, data_test, target_train, target_test = train_test_split(
    data, target, test_size=0.2, random_state=0) ←
```

用 `random_state` 參數來固定分割結果



12-1 KNN (K 近鄰) 預測模型

- 我們在第11章使用`train_test_split()`函式時，它會隨機分割資料集。
- 但你可用`random_state`參數來固定分割結果；只要對`random_state`指定同樣的數字，每次分割出來的資料集和預測結果就會相同。
- 本章範例都會採用設定`random_state`參數。



12-1 KNN (K 近鄰) 預測模型

- 建立 KNN 模型來預測分類
- 資料準備妥當後，就來建立KNN模型，並針對測試集做預測。

IN

```
from sklearn.neighbors import KNeighborsClassifier ← 載入 KNN 模型

knn = KNeighborsClassifier(n_neighbors=5) ← 設定訓練集
knn.fit(data_train, target_train)


predictions = knn.predict(data_test) } 預測目標值
print(predictions)
print(target_test) ← 真實目標值
```

建立 KNN 模型, 指定參數 `n_neighbors` (K 值或近鄰數量) 為 5



12-1 KNN (K 近鄰) 預測模型

- 建立 KNN 模型來預測分類
- 右側印出的結果可以看見大部分的值是相同的，但也有些有出入，接著會來學習判斷預測能力的好壞。
- 如果不指定n_neighbors參數，預設會是5。



[0	1	1	0	1	1	0	2	1	1	0	1	0	2	0	1	0	0	1	0	1	0	2	1	1	1	1	1	2	2	0	0	1	0	0	0]
[0	2	1	0	1	1	0	2	1	1	2	2	0	1	2	1	0	0	1	0	1	0	0	1	1	1	1	1	1	2	0	0	1	0	0	0]

預測目標值

真實目標值



12-1 KNN (K 近鄰) 預測模型

- 知識補充:懶惰學習法
- KNN有個比較特別之處，你會發覺它其實不需要訓練，只有在被要求預測時(呼叫predict())時才會從資料中挑出K個鄰近來判斷。
- 因此KNN也被稱為是懶惰學期法(lazy algorithm)。



12-1 KNN (K 近鄰) 預測模型

- 評估預測結果
- 和迴歸模型一樣，KNN模型的score()函式會傳回模型的預測能力分數，只不過以分類器來說，這個數值會是準確率(accuracy)，也就是預測正確的比率。

IN

```
print(knn.score(data_train, target_train).round(3))  
print(knn.score(data_test, target_test).round(3))
```



0.789
0.806

- 訓練集的預測準確率是78.9%，對測試集是80.6%，結果算是接近且不低。
- 若是訓練集分數特別高、測試集卻很低，就代表模型過度適配。



12-1 KNN (K 近鄰) 預測模型

- 若想了解模型中對每個分類的預測成效，下面的工具可用來產生報表：

IN

```
from sklearn.metrics import classification_report ← 載入報表工具  
  
print(classification_report(target_test, predictions))
```

填入真實目標值和預測目標值



12-1 KNN (K 近鄰) 預測模型

- 結果:



OUT

		precision	recall	f1-score	support	
分類標籤	{ 0	0.87	0.93	0.90	14	
	{ 1	0.88	0.88	0.88	16	
	{ 2	0.40	0.33	0.36	6	
accuracy				0.81	36	準確率
macro avg		0.71	0.71	0.71	36	直接平均
weighted avg		0.79	0.81	0.80	36	加權平均



12-1 KNN (K 近鄰) 預測模型

- 知識補充: 準確率/精準率/召回率
- 上面報表資料就是從前一節的預測值/真實值推算出來的。

																																		預測目標值			
[0	1	1	0	1	1	0	2	1	1	0	1	0	2	0	1	0	0	1	0	1	0	2	1	1	1	1	1	2	2	0	0	1	0	0	0]	
[0	2	1	0	1	1	0	2	1	1	2	2	0	1	2	1	0	0	1	0	1	0	0	1	1	1	1	1	1	2	0	0	1	0	0	0]	真實目標值
																																			預測值與真實值有出入之處		

- 可見36筆資料中有7筆預測錯誤、29筆正確，因此準確率為 $29/36=0.805$ (80.6%)



12-1 KNN (K 近鄰) 預測模型

- 知識補充: 準確率/精準率/召回率
- 報表中有precision、recall、f1-score、support四個欄位，又各自代表什麼意思？

	precision	recall	f1-score	support
0	0.87	0.93	0.90	14
1	0.88	0.88	0.88	16
2	0.40	0.33	0.36	6
accuracy			0.81	36
macro avg	0.71	0.71	0.71	36
weighted avg	0.79	0.81	0.80	36



12-1 KNN (K 近鄰) 預測模型

- 精確率(precision)和準確率(accuracy)的意義是不同
- 準確率代表整理資料預測正確的比例，以本例來說整體36筆資料有29筆正確被預測，因此算出0.81。
- 然而，資料中有三個分類，到底哪個葡萄品種分類被預測最準？
- 就可以利用精確率或召回率來看他們各別的預測效果。

	precision	recall	f1-score	support
0	0.87	0.93	0.90	14
1	0.88	0.88	0.88	16
2	0.40	0.33	0.36	6
accuracy			0.81	36
macro avg	0.71	0.71	0.71	36
weighted avg	0.79	0.81	0.80	36



12-1 KNN (K 近鄰) 預測模型

- 精確率(precision)和準確率(accuracy)的意義是不同

名稱	意義
precision (精準率)	當你預測資料為第 N 類時猜對的比例
recall (召回率)	在所有第 N 類的資料中, 正確被預測為第 N 類的比率
f1 score	精準率與召回率的調和平均數
support	資料分類為 N 的實際數量, 本例實際上第 0 類有 14 個、第 1 類有 16 個、第 2 類有 6 個



12-1 KNN (K 近鄰) 預測模型

- 以第0類(Barolo葡萄)為例，你能從他對應的support欄位看到。
- 它在測試集中佔了14筆資料。但K預測第0類的卻是15筆。
- 相對的，在這14筆資料裡，有一個第0類的卻被誤判為第2類。

預測值	[0 1 1 0 1 1 0 2 1 1 0 1 0 2 0 1 0 0 1 0 1 0 2 1 1 1 1 1 2 2 0 0 1 0 0 0]	15筆 "0"
真實值	[0 2 1 0 1 1 0 2 1 1 2 2 0 1 2 1 0 0 1 0 1 0 0 1 1 1 1 1 1 2 0 0 1 0 0 0]	14筆 "0"

- 這時我們就可以來看所謂的精準率跟召回率，這兩個指標能用不同角度檢視模型對某分類的預測效果。



12-1 KNN (K 近鄰) 預測模型

- 所以: 預測值 [0 1 1 0 1 1 0 2 1 1 0 1 0 2 0 1 0 0 1 0 1 0 2 1 1 1 1 1 2 2 0 0 1 0 0 0]
真實值 [0 2 1 0 1 1 0 2 1 1 2 2 0 1 2 1 0 0 1 0 1 0 0 1 1 1 1 1 1 2 0 0 1 0 0 0]

精準率 = 模型猜測資料為第N類時有猜對的比率

召回率 = 實際是第N類的資料中，有被正確猜出來的比率

- 模型猜測有15筆是第0類，但這裡面只有13筆猜對，故精準率為 $13/15=87\%$
- 屬於第0類的資料實際有14筆，但只有13個正確被猜出，召回率為 $13/14=93\%$

- 因此精準率著重的是『寧可漏抓一千, 也不錯抓一人』。
- 而召回率則著重『寧可錯抓一千, 也不漏抓一人』。



12-1 KNN (K 近鄰) 預測模型

- 如何找出最適合的 K 值？
- 由於 K 值(鄰近數)是使用者是先指定的，你可能得多試幾種不同的 K 值，看準確率、精準率和召回率等分數，才能判斷哪個 K 值可得到最佳結果。
- K 值設得太低或太高, 都有可能降低模型的預測能力。

	precision	recall	f1-score	support
0	0.87	0.93	0.90	14
1	0.88	0.88	0.88	16
2	0.40	0.33	0.36	6
accuracy			0.81	36
macro avg	0.71	0.71	0.71	36
weighted avg	0.79	0.81	0.80	36



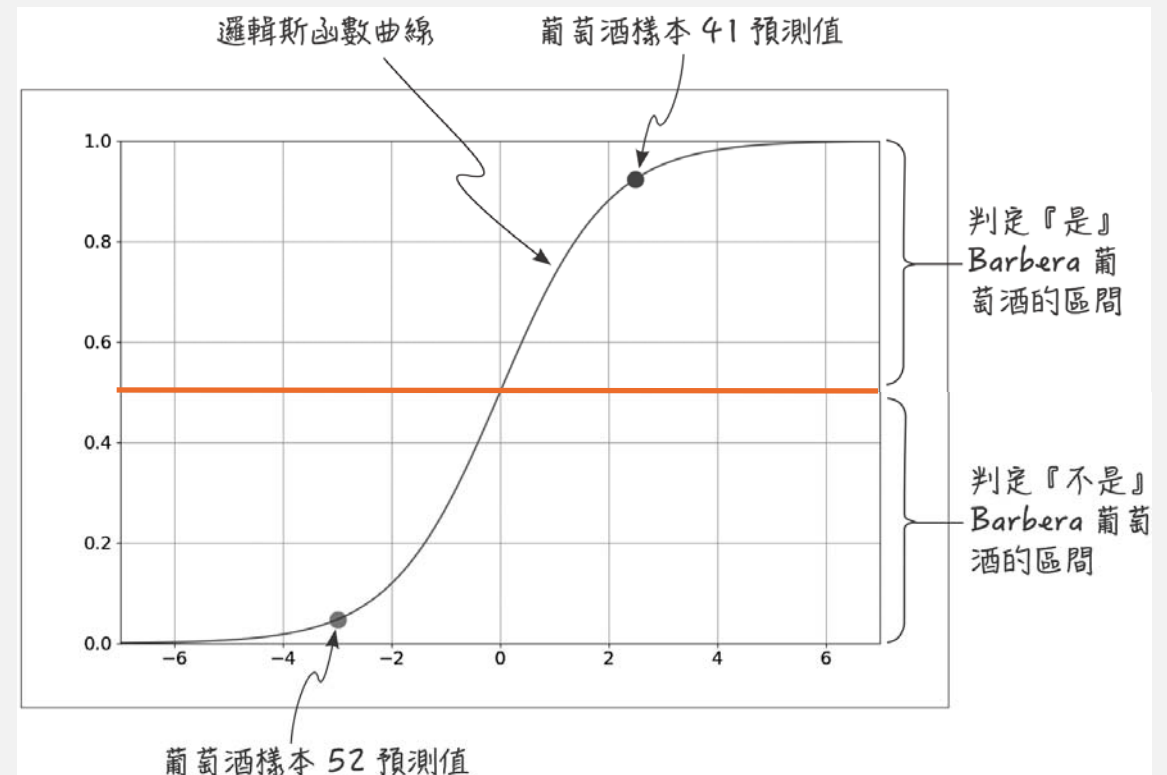
12-2 邏輯斯 (Logistic) 迴歸模型

- 邏輯斯迴歸 (Logistic Regression)：二分法分類器
- 此為第二種常見的分類器，目前已廣泛被各領域應用：
 - ✓ 預測病患是否有糖尿病、心臟病等疾病，或者所受創傷是否可能致命。
 - ✓ 預測機器的零件在一年後是否會故障、道路是否會發生土石流或崩塌。
 - ✓ 預測一個人是否會投票給某政黨、是否願意購買某商品或訂閱服務、是否會拖延繳房貸等等。
- 邏輯斯迴歸與線性迴歸相同會以自變數來算出目標值。
- 兩者差別在邏輯斯迴歸中預測值會介於0~1之間，它的值可當成出現某分類的機率，讓模型能用來判斷資料的分類。



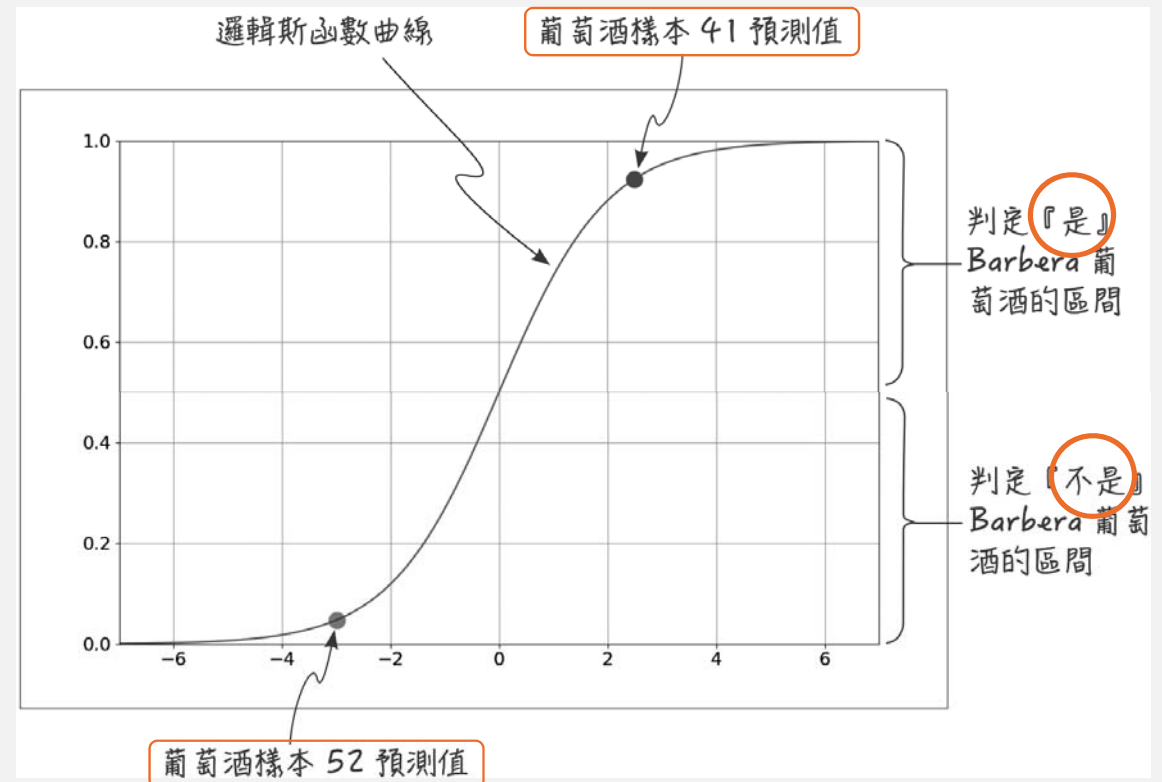
12-2 邏輯斯 (Logistic) 迴歸模型

- 邏輯斯迴歸如何分類
- KNN模型是直接計算自變數的差距來判斷新資料分類。
- 而邏輯斯迴歸模型會使用一個**邏輯斯函數 (logistic function)**，將資料的自變數丟入這函數來算出目標值(介於0到1)，目標值越接近1則越可能屬於該分類。
- 函數曲線如右圖。



12-2 邏輯斯 (Logistic) 迴歸模型

- 此迴歸是個『二分法』分類器，只能判定資料『是或不是』屬於某分類。
- 所以41號預測值在Y軸接近1，會判定是Barbera；
- 52號接近0就不是Barbera。



12-2 邏輯斯 (Logistic) 迴歸模型

- 邏輯斯很巧妙的利用邏輯斯函數來做二分法，只要算數大於或小於0.5即可。
- 以葡萄酒種類的案例來說，scikit-learn的邏輯斯迴歸模型會自動把資料拿去對每一種葡萄做預測，看在哪個分類得到的預測值最接近1，那就是判定為那類。



12-2 邏輯斯 (Logistic) 迴歸模型

- 訓練邏輯斯迴歸模型並預測資料
- 案例練習，匯入併分割葡萄酒品種資料集。

IN

```
data, target = datasets.load_wine(return_X_y=True)
```

```
data_train, data_test, target_train, target_test = train_  
test_split(data, target, test_size=0.2, random_state=0)
```

接下行

- 接著匯入邏輯斯迴歸模型，訓練後印出對測試集目標值的預測。



12-2 邏輯斯 (Logistic) 迴歸模型

- 程式碼:

IN

```
from sklearn.linear_model import LogisticRegression
```

```
log_model = LogisticRegression()
```

```
log_model.fit(data_train, target_train)
```

```
predictions = log_model.predict(data_test)
```

```
print(predictions)
```

```
print(target_test)
```

載入邏輯斯迴歸模型 (別和前一章的 *LinearRegression* 搞混)

預測目標值

建立邏輯斯迴歸模型物件



邏輯斯迴歸能對3個葡萄酒種類做預測

0	2	1	0	1	1	0	2	1	1	2	2	0	1	2	1	0	0	2	0	0	0	1	1	1	1	1	1	2	0	0	1	0	0	0	
0	2	1	0	1	1	0	2	1	1	2	2	0	1	2	1	0	0	1	0	1	0	0	1	1	1	1	1	1	2	0	0	1	0	0	0



國立中山大學

National Sun Yat-sen University



12-2 邏輯斯 (Logistic) 迴歸模型

- 知識補充: 有警告訊息?
- 在訓練邏輯斯迴歸模型時，可能會遇到螢幕出現如下面的警告訊息。
- 此訊息再說模型迭代(iterate)或訓練次數不夠，算出的還不是最佳解。此問題後續會探討。
- 迭代是指模型一次次的訓練過程(訓練一次即是迭代一次)。

```
C:\Users\admin\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

```
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression  
n_iter_i = _check_optimize_result(
```



12-2 邏輯斯 (Logistic) 迴歸模型

- 評估預測成效
- 印出模型的準確率、精準度及召回率指標，看預測成效如何。

IN

```
print(log_model.score(data_train, target_train).round(3))  
print(log_model.score(data_test, target_test).round(3))
```



0.986
0.917



12-2 邏輯斯 (Logistic) 迴歸模型

- 程式碼:

IN

```
from sklearn.metrics import classification_report  
print(classification_report(target_test, predictions))
```



12-2 邏輯斯 (Logistic) 迴歸模型

- 預測準確率明顯比KNN高，這表示邏輯斯迴歸更適合來預測葡萄品種的分類。

←對比→

	precision	recall	f1-score	support
0	0.93	0.93	0.93	14
1	0.93	0.88	0.90	16
2	0.86	1.00	0.92	6
accuracy			0.92	36
macro avg	0.91	0.93	0.92	36
weighted avg	0.92	0.92	0.92	36

邏輯斯迴歸

	precision	recall	f1-score	support
0	0.87	0.93	0.90	14
1	0.88	0.88	0.88	16
2	0.40	0.33	0.36	6
accuracy			0.81	36
macro avg	0.71	0.71	0.71	36
weighted avg	0.79	0.81	0.80	36

KNN分析結果



12-2 邏輯斯 (Logistic) 迴歸模型

- 挑選演算法: 天生我材必有用
- 葡萄酒品種案例不是在說KNN表現就一定比較差，若你嘗試練習第九章提到的花瓣資料集來說，使用KNN就能達成很高的預測準確率。
- 因此在實作過程時，可以試著套用不同的模型運算看看，才知道哪個模型最適合使用。



12-3 改善邏輯斯迴歸模型

- 增加迭代次數
- 此技巧可以改善邏輯斯迴歸模型的整體預測能力。
- 回顧前幾頁有提到編輯器警告迭代次數不足而找不到最佳解的狀況，這是因為內部預設的迭代次數只有100次。
- 解決辦法是以 `max_iter` 參數指定更大的最大迭代次數。

```
In [10]: log_model=LogisticRegression(max_iter=10000)
```

新增此資訊到原程式碼

- 重新執行程式應該就不會再出現警告訊息!



12-3 改善邏輯斯迴歸模型

- 若你想看模型實際迭代次數或詳細運算過程，可以將 `verbose` 參數設為 `True`。

IN

```
log_model = LogisticRegression(max_iter=10000, verbose=True)
```



12-3 改善邏輯斯迴歸模型

- 注意: 此訊息會出現在背景執行的終端機(啟動編輯器後會看到的黑色視窗)
- 重新訓練模型後看到只是的實際總迭代次數 (Tit, total iteration) 是 2638 次。

```
Jupyter Notebook (anaconda3)

At iterate 2400    f= 9.34497D+00    lproj gl= 3.09148D-01
At iterate 2450    f= 9.34464D+00    lproj gl= 1.10360D+00
At iterate 2500    f= 9.34452D+00    lproj gl= 1.98257D+00
At iterate 2550    f= 9.34416D+00    lproj gl= 1.77219D+00
At iterate 2600    f= 9.34360D+00    lproj gl= 7.61691D-01
    * * *

Tit  = total number of iterations
Tnf  = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip = number of BFGS updates skipped
Nact = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F    = final function value

    * * *

N  Tit  Tnf  Tnint  Skip  Nact  Projg  F
42 2638 2967    1    0    0  1.397D-02  9.343D+00
F = 9.34340453959338

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
[I 20:13:54.463 NotebookApp] Saving file at /Untitled21.ipynb
```



12-3 改善邏輯斯迴歸模型

- 而模型預測準確率也可能提高。

```
In [13]: print(log_model.score(data_train, target_train).round(3))  
         print(log_model.score(data_test, target_test).round(3))
```



0.993
0.972

←對比→

0.986
0.917

- 只要讓邏輯斯迴歸有足夠迭代次數，訓練出來的模型預測準確率就可以提高。



12-3 改善邏輯斯迴歸模型

- 資料標準化 (standardization)
 - 第二個增進模型效能的手段叫做資料標準化(亦稱為正規化)。
 - 標準化會把所有自變數欄位的資料按比例調整，使平均數都變成0、標準差為1。
 - 這樣的好處對模型是更容易處理並可能減少訓練時間和提高預測能力。
-
- 案例解釋:
 - 將下列兩筆範圍不同的資料進行標準化。

資料1	10	20	30	40	50
資料2	500	1000	1500	2000	2500



12-3 改善邏輯斯迴歸模型

- 標準化數值後如下:

資料1	-1.41421356	-0.70710678	0	0.70710678	1.41421356
資料2	-1.41421356	-0.70710678	0	0.70710678	1.41421356

- 兩筆資料變得一模一樣，接著來計算轉換過程的資料平均數與準差:

```
In [15]: import numpy as np
data_std=np.array([-1.41421356, -0.70710678, 0, 0.70710678, 1.41421356])

print(data_std.mean().round(3))
print(data_std.std().round(3))
```



0.0
1.0



12-3 改善邏輯斯迴歸模型

- 因此可以在分割資料集之前, 先對所有自變數資料做標準化。

IN

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
data_std = sc.fit_transform(data) ← 將自變數資料標準化

data_train, data_test, target_train, target_test = train_ 接下行
test_split(data_std, target, test_size=0.2, random_state=0)
```

使用標準化資料來分割資料集



12-3 改善邏輯斯迴歸模型

- 先將資料標準化再拿給train_test_split()分割，再訓練邏輯斯迴歸模型，最後再用score()評估成效。

```
In [20]: print(log_model.score(data_train, target_train).round(3))  
         print(log_model.score(data_test, target_test).round(3))
```



```
1.0  
1.0
```



12-3 改善邏輯斯迴歸模型

- 結果:
- 你會看到無論是訓練集還是測試集，準確率都提高到100%。
- 另外，若在建模時有加入verbose參數，可以發現僅要迭代26次，執行時間也從將近1秒降到0.1秒以下。

```
N      Tit      Tnf  Tnint  Skip  Nact      Projg      F
42      26      28      1      0      0      2.434D-04      1.102D+01
F =      11.0174758948706

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
[I 08:13:27.563 NotebookApp] Saving file at /Untitled23.ipynb
```



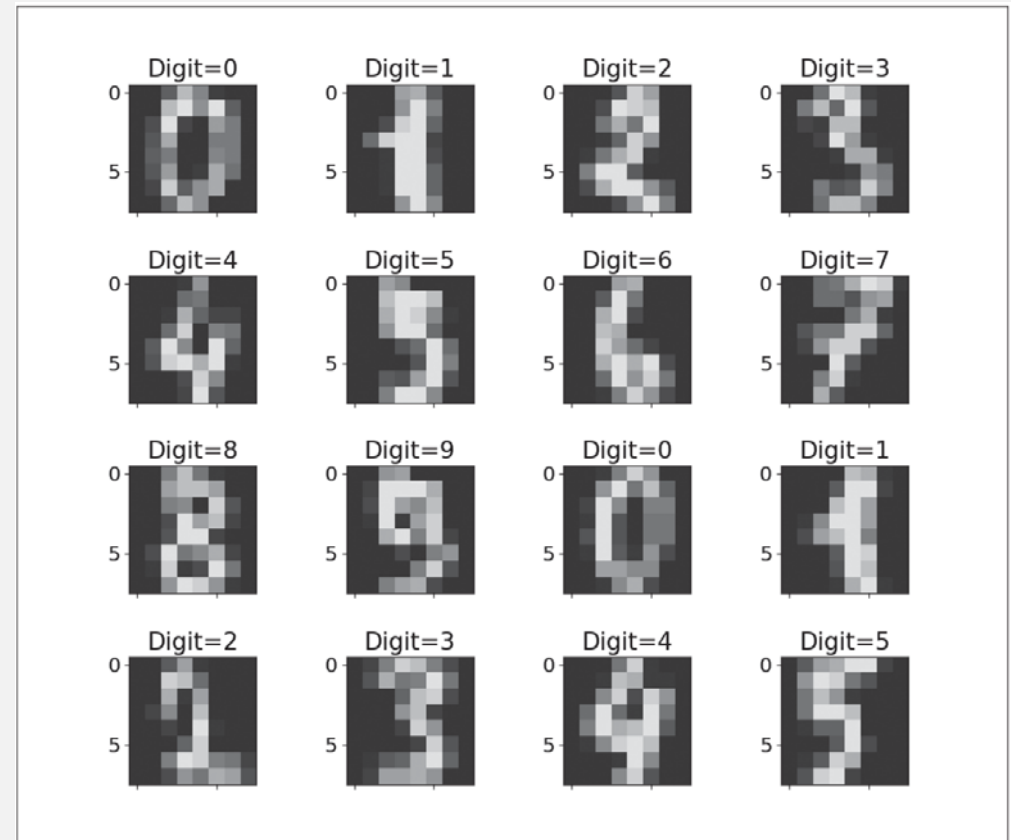
12-4 主成分分析 (PCA)：減少需分析的變數

- 前言:
- 不是所有機器學習演算法都是用來做迴歸或分類預測。
- 有一類演算法可以用來減少資料自變數(即feature，特徵值)的數量或複雜度，接下來要介紹主成分分析(principal component analysis)，簡稱PCA。
- 主成分分析就是分析自變數，看資料集中那些自變數的變異程度最明顯。
- 如果某變數(特徵值)的變異程度總是很小，它對目標值可能影響不大、甚至忽略，那這個自變數就不用納進來當分類或迴歸分析。
- 若是資料集的變數非常多且數據量也大，那主成分分析就可以大大減少要處理的資料量，讓模型更快被訓練完，而預測能力仍可維持在差不多的水準。



12-4 主成分分析 (PCA)：減少需分析的變數

- 特徵值很多的手寫數字圖片資料集
- 使用scikit-learn內建的手寫數字資料集。
- 如右，資料集當中每張圖片是由8x8像素的數值陣列組成，每個陣列元素的值代表圖片中一個像素的深淺度，其值在0~16之間。



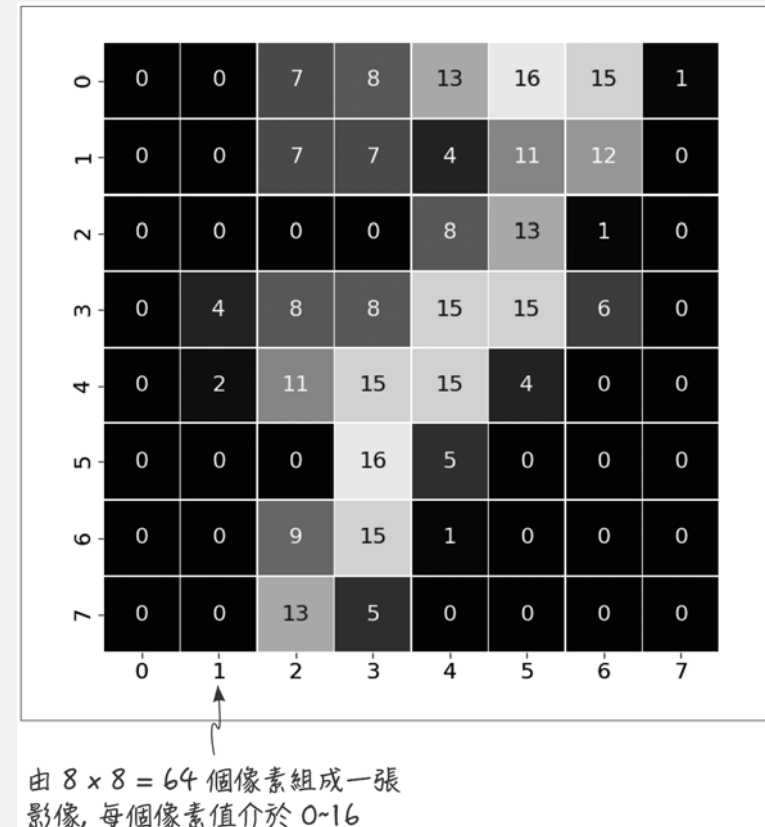
12-4 主成分分析 (PCA)：減少需分析的變數

- 知識補充:為何手寫數字的像素是0~16
- 一般說來，圖像的像素值範圍會是2的次方。
- 比如JPEG的紅藍綠三色都是介於0~255的數字(因此共256或 2^8 種顏色)。
- 然而，手寫數字資料集的像素值是0~16而不是更合理的0~15(2^4)?
- 這是因為當初這些數字的掃描影像經過處理後，剛好必須用17個數值才能儲存足夠的資訊。



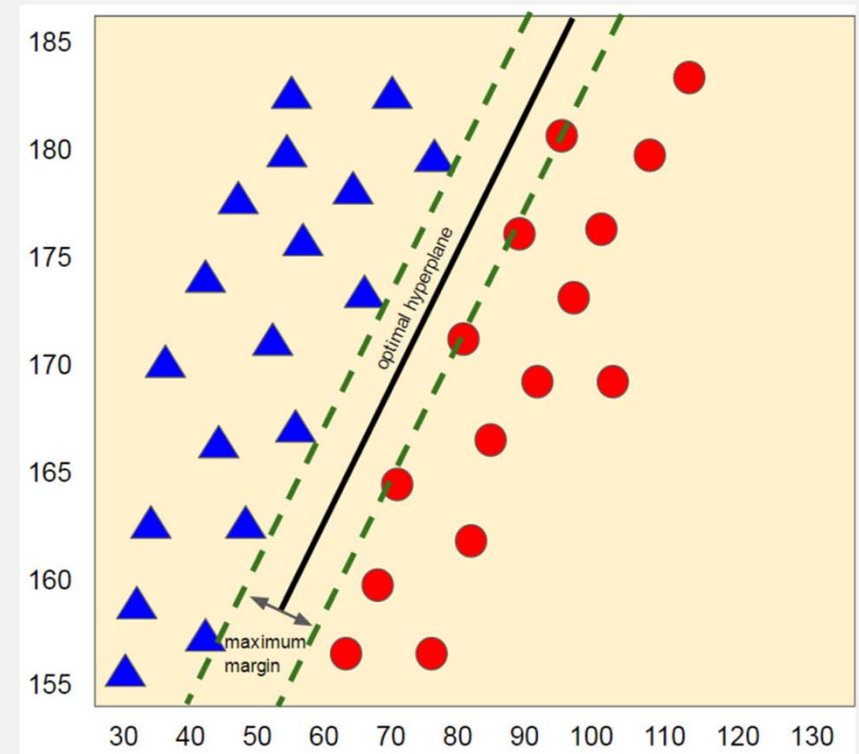
12-4 主成分分析 (PCA)：減少需分析的變數

- 右圖是其中一個數字7的原始資料：
- 可以看到每個元素會介於0到16的整數代表筆跡的深淺程度。
- 換言之，每張圖就有64個像數值，也就是64個變數或特徵值。
- 而每筆資料的目標值或標籤，當然就是該圖所表示的數字。



12-4 主成分分析 (PCA)：減少需分析的變數

- 用線性支援向量機建立預測模型
- 這是另一個廣泛被使用的模型，**支援向量機**(support vector machine, SVM)。
- 我們會用SVM來訓練經過PCA簡化之前和之後的資料來比較差異。
- SVM可用在多維的空間對資料做分類，它會在資料之間劃出一個分界區，好判定資料究竟是屬於標籤A或是B。
- 本案例探討線性支持向量機 (linear SVM)。



<https://ccw1986.blogspot.com/2016/03/svm.html>



12-4 主成分分析 (PCA)：減少需分析的變數

- 首先匯入手寫數字資料集、將之標準化，分割成訓練集與測試集，接著把資料給線性支援向量機訓練：

```
In [9]: from sklearn import datasets
        from sklearn.model_selection import train_test_split

        data,target=datasets.load_digits(return_X_y=True)
```

IN

```
sc = StandardScaler()
data_std = sc.fit_transform(data)

data_train, data_test, target_train, target_test = train_ 接下行
test_split(data_std, target, test_size=0.2, random_state=42)
```

- 接著把資料丟給線性支援向量機訓練，並找出預測準確率。



12-4 主成分分析 (PCA)：減少需分析的變數

- 程式碼:

IN

```
from sklearn.svm import LinearSVC ← 匯入模型

svc = LinearSVC(max_iter=10000) ← 建立模型, 設定最大迭代次數為一  
svc.fit(data_train, target_train) 萬次, 並在終端輸出訓練過程  
predictions = svc.predict(data_test)

print(svc.score(data_train, target_train).round(3))  
print(svc.score(data_test, target_test).round(3))
```



0.995
0.856

- 結果不錯，可以達到至少95%以上的準確率。



12-4 主成分分析 (PCA)：減少需分析的變數

- 檢視特徵值的變異程度
- 回頭來想想，PCA能將手寫數字集簡化到什麼程度呢？
- 你能否只用8X8圖片的一部分(也就是少於64個像素)就能準確預測手寫數字？
- 為了得知手寫數字集中所有特徵的變異程度，利用PCA來檢視每個變數的變動量所佔的比率：

IN

```
from sklearn.decomposition import PCA
```

```
pca = PCA()
```

```
pca.fit(data) ← 填入整個自變數資料
```

```
print(pca.explained_variance_ratio_.round(3))
```

顯示各變數所占的
變異度比率(四捨五
入到小數第3位)



12-4 主成分分析 (PCA)：減少需分析的變數

- 結果:

```
[0.149 0.136 0.118 0.084 0.058 0.049 0.043 0.037 0.034 0.031 0.024 0.023  
0.018 0.018 0.015 0.014 0.013 0.012 0.01 0.009 0.009 0.008 0.008 0.007  
0.007 0.006 0.006 0.005 0.005 0.004 0.004 0.004 0.003 0.003 0.003 0.003  
0.003 0.002 0.002 0.002 0.002 0.002 0.002 0.001 0.001 0.001 0.001 0.001  
0.001 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
0. 0. 0. 0. ]
```

- 上面64個數值由大排到小:0.149、0.136、0.117...這表示在所有變數中，變異最大的佔總變異量14.9%、第二占13.6%，第三個占11.7%...後面越來越小。
- 這些數值僅是依照大小排序，不代表真正的像素位置。
- 其實對預測模型來說真正在乎像素位置的只有人類。



12-4 主成分分析 (PCA)：減少需分析的變數

- 用 PCA 篩選特徵值
- 假設希望PCA在篩選變數後，總變異程度能保留原本資料的85%，可透過如下：

IN

```
pca = PCA(n_components=0.85) ← 設定保留 85% 變異程度  
pca.fit(data) ← 讓 PCA 篩選變數  
print(pca.explained_variance_ratio_.round(3)) ←
```



印出篩選後的變異程度比率

```
[0.149 0.136 0.118 0.084 0.058 0.049 0.043 0.037 0.034 0.031 0.024 0.023  
0.018 0.018 0.015 0.014 0.013]
```



12-4 主成分分析 (PCA)：減少需分析的變數

- 從結果看印出的數值只有17個，僅有原本1/3不到。
- 這表示我們活許只需要手寫數字圖片的1/3就可以做出跟眼前差不多的預測效果。
- 其實這一點也不意外，因為手寫數字圖片的周遭通常是空白，將這些空白像素去掉影響並不大。
- 如果你給n_components參數指定一個大於1的數字N，那麼PCA就會從所有變數中選出前N個變異程度最大的：

```
In [11]: pca=PCA(n_components=10)
```

← 選出前10個變異程度最大的變數



12-4 主成分分析 (PCA)：減少需分析的變數

- 這種減少變數或特徵數量的過程稱為降維(dimension reduction)。
- 也就是說我們把64維度的資料減少到17維度。
- 因此，若你想分析1600萬像素手機拍的照片(4600x3480)像素的彩色圖片，每個像素有紅綠藍三色，每張圖的變數將高達 $4600 \times 3480 \times 3 = 4800$ 萬個。
- 這時候能夠明顯減少數量就可以省下可觀的訓練時間。
- 由於PCA可判斷自變數的變異程度並篩選之，卻不需要知道每筆資料的分類為何。所以PCA 被稱為是非監督式學習 (unsupervised learning) 演算法。
- 但本節用來做預測的SVM得用到由人類準備好的標籤資料，還是屬於監督式學習演算法。



12-4 主成分分析 (PCA)：減少需分析的變數

- 拿簡化過的資料來訓練模型
- 來看看減化後的資料得到的預測準確率是多少

IN

```
pca = PCA(n_components=0.85) ← 只保留前 85% 變異度的自變數  
data_pca = pca.fit_transform(data) ← 使用 PCA 篩選資料 (注意  
這裡呼叫的函式名稱)  
  
sc = StandardScaler()  
data_pca_std = sc.fit_transform(data_pca) ← 對資料做標準化  
  
data_train, data_test, target_train, target_test = train_  
test_split(data_pca_std, target, test_size=0.2, random_state=0)
```



0.963
0.956



12-4 主成分分析 (PCA)：減少需分析的變數

- 總結:
- 為什麼PCA分析得在標準化之前進行? 因為資料標準化後，其變異程度就改變了，拿來做PCA自然失去篩選的依據。
- 從結果知道就算只用1/3不到的原始像素，SVM仍可達到95%以上的預測準確率，可見PCA的效益是值得的。
- 若在LinearSVC()加入verbose=Tuse參數，並觀看終端機內輸出的訓練過程訊息(黑色對話框)，會發現SVM迭代次數約在1200至1300次上下。但使用PCA後減少到500次上下。
- 資料簡化後，訓練時間也變成一半不到。



下課前，請帶走它~

- 能用來辨認或預測資料分類的機器學習演算法稱為**分類器(classifier)**。
- 本章介紹三種常用的分類器，包括K近鄰(K-nearest neighbors, KNN)、邏輯斯迴歸(Logistic Regression)、支援向量機(support vector machine, SVM)，它們屬於監督式學習(supervised learning)演算法，各自使用不同原理來預測資料分類。
- 用分類器對策是集做預測後，可以用模型的score()函式看齊預測準確率。
- 你也可用classification_report()來取得預測的**精準率(precision)**、**召回率(recall)**等數據。
- 有些模型可藉由增加訓練迭代次數來得到更加結果。



下課前，請帶走它~

- 資料標準化(standardization)也能減少模型訓練時間、進一步提高預測準確率。
- 主成分分析(principal component analysis, PCA)可以保留資料中變異程度最大的一些變數，藉此簡化資料(降維)。
- 資料簡化有助於減少模型訓練時間。

