機器學習

第 9 章 邏輯迴歸
(Logistic Regression)

講師：紀俊男

本章大綱

- 邏輯迴歸簡介
- 資料前處理
- 邏輯迴歸實作
- 評估模型好壞
- 卡方檢定降維法
- 使用「快樂版」實作
- 將結果視覺化
- 重點整理

邏輯迴歸簡介

Introduction of Logistic Regression

何謂「分類問題」

- 迴歸問題

年齡	月薪	購買額
19	19000	0
32	150000	15638
25	33000	0
47	25000	4752
45	26000	1244
46	28000	3677
32	18000	0
18	82000	0
47	49000	6221
48	41000	4576
45	22000	598
35	65000	0

連續數字

- 分類問題

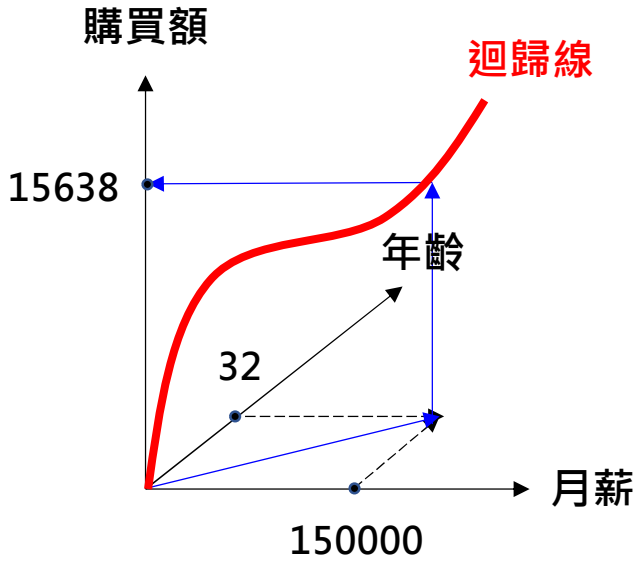
年齡	月薪	是否購買
19	19000	0
32	150000	1
25	33000	0
47	25000	1
45	26000	1
46	28000	1
32	18000	0
18	82000	0
47	49000	1
48	41000	1
45	22000	1
35	65000	0

離散數字

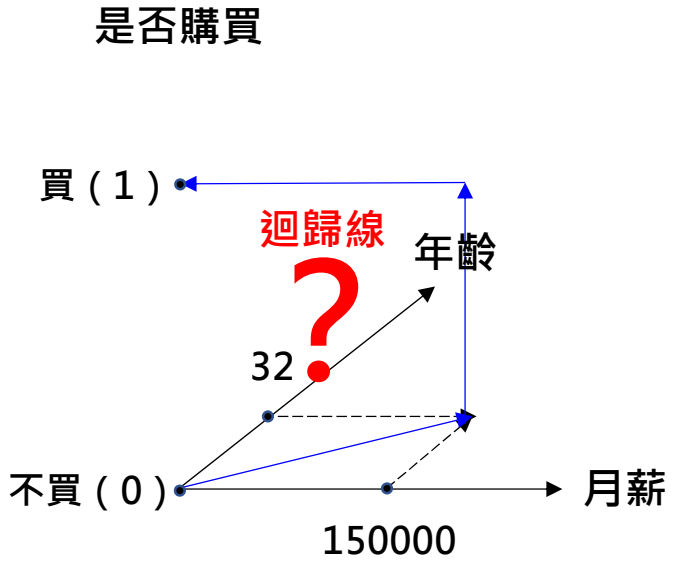
分類問題 = 應變數 Y 是離散數字的問題

「分類問題」的困難點

- 迴歸問題



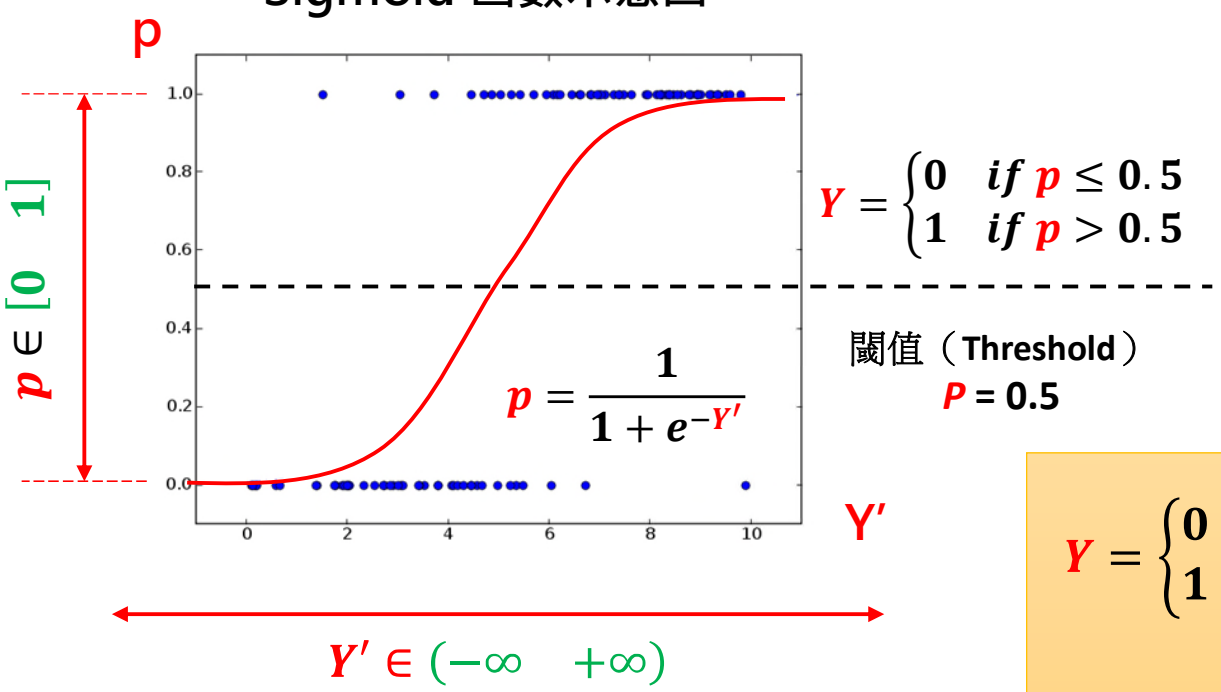
- 分類問題



困難點：擬合不出一條迴歸線，可以把連續數字，映射到離散數字！

解決方法：Sigmoid 函數

Sigmoid 函數示意圖



連續數字　　連續數字

$$Y' = c_0 + c_1X_1 \cdots c_nX_n$$

$$p = \frac{1}{1+e^{-Y'}} \rightarrow Y' = ln\left(\frac{p}{1-p}\right)$$

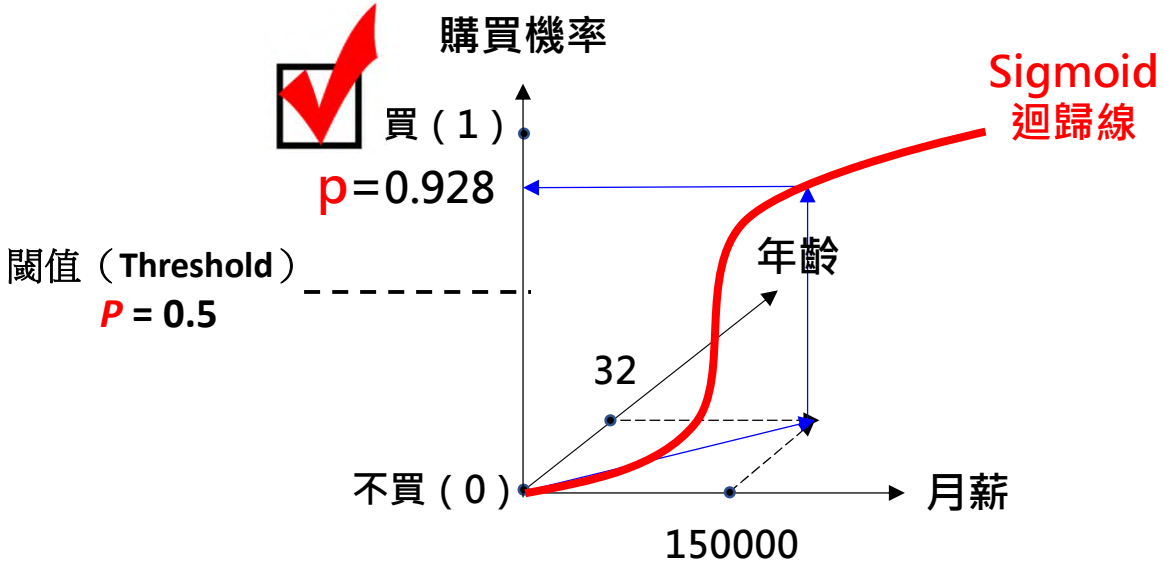
$$Y = \begin{cases} 0 & if\ p \le 0.5 \\ 1 & if\ p > 0.5 \end{cases} + ln\left(\frac{p}{1-p}\right) = c_0 + c_1X_1 \cdots c_nX_n$$

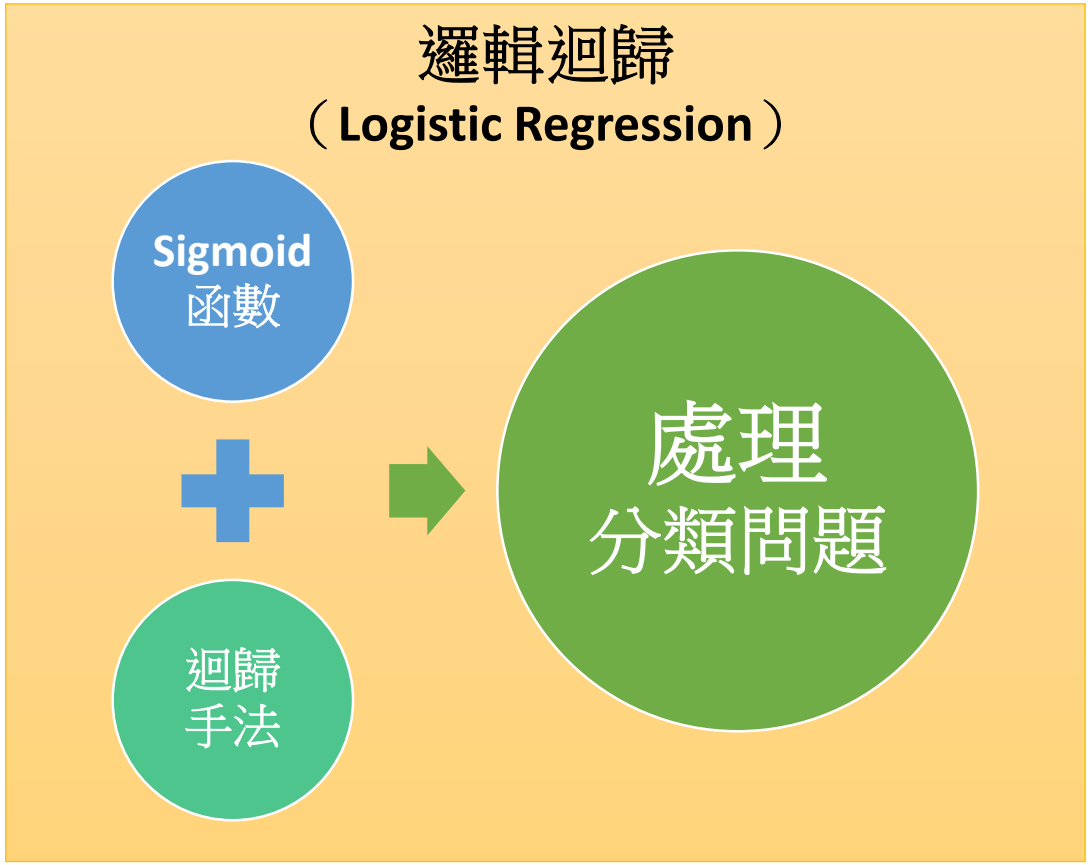
$$p \in [0 \quad 1]$$

邏輯迴歸數學原理

結論：何謂「邏輯迴歸」

Y = 1（買）





資料前處理

下載與瀏覽資料集

- 依照講師指示，**下載**並**瀏覽**資料集

Social_Network_Ads.csv

	A	B	C	D	E
1	User ID	Gender	Age	Salary	Purchased
2	15624510	Male	19	19000	0
3	15810944	Male	35	20000	0
4	15668575	Female	26	43000	0
5	15603246	Female	27	57000	0
6	15804002	Male	19	76000	0
7	15728773	Male	27	58000	0
8	15598044	Female	27	84000	0
9	15694829	Female	32	150000	1
10	15600575	Male	25	33000	0

目的：利用「性別、年齡、薪資」
--> 推算「是否購買」

資料前處理

撰寫程式碼

```python
import HappyML.preprocessor as pp

# Load Dataset
dataset = pp.dataset(file="Social_Network_Ads.csv")

# X, Y Decomposition
X, Y = pp.decomposition(dataset, x_columns=[1, 2, 3], y_columns=[4])

# Categorical Data Encoding & Remove Dummy Variable Trap
X = pp.onehot_encoder(X, columns=[0], remove_trap=True)

# Split Training & Testing set
X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y)

# Feature Scaling
X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```

資料前處理流程：

1. 載入資料
2. 切分自變數、應變數
3. 處理缺失資料

   （無缺失資料）
4. 類別資料數位化

   （+移除虛擬變數陷阱）
5. 切分訓練集、測試集
6. 特徵縮放

   （僅縮放 X 即可）

隨堂練習：資料前處理

- 請撰寫下列程式碼，並執行「資料前處理」流程：

```
1  import HappyML.preprocessor as pp
2
3  # Load Dataset
4  dataset = pp.dataset(file="Social_Network_Ads.csv")
5
6  # X, Y Decomposition
7  X, Y = pp.decomposition(dataset, x_columns=[1, 2, 3], y_columns=[4])
8
9  # Categorical Data Encoding & Remove Dummy Variable Trap
10 X = pp.onehot_encoder(X, columns=[0], remove_trap=True)
11
12 # Split Training & Testing set
13 X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y)
14
15 # Feature Scaling
16 X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```

邏輯迴歸實作

使用「標準函式庫」

使用標準函式庫訓練與預測

- 原始程式碼

引入所需的套件

```
1  from sklearn.linear_model import LogisticRegression
2  import time
3
4  # Model Creation
5  model = LogisticRegression(solver="lbfgs", random_state=int(time.time()))
6
7  # Training
8  model.fit(X_train, Y_train.values.ravel())
9
10 # Testing
11 Y_pred = model.predict(X_test)
```

收斂演算法

是否執行亂數抽樣

先將 Y_train 變成 NDArray ↓

← 將維度從二維的 (400, 1)，變成一維的 (400,)

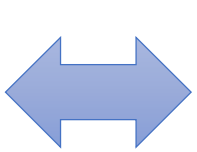
- 收斂演算法
  - liblinear：線性收斂法。小資料集較好用。僅能用於「二擇一」的應變數。受抽樣公平性影響較大。
  - lbfgs：Limited-memory Broyden–Fletcher–Goldfarb–Shanno。BFGS 法的簡化版。速度快。精度稍低。多擇一應變數可。
  - sag：Stochastic Average Gradient Descending Method。隨機平均梯度下降法。適用於大型資料集。必做特徵縮放。多擇一可。
  - newton-cg：Newton Conjugate Gradient。牛頓共軛梯度法。精度高。計算量大。多擇一可。

邏輯迴歸的訓練結果

- 執行結果

Y_test

	0
0	0
1	1
2	0
3	0
4	0
5	1
6	0
7	1
8	0
9	1
10	1



	0
0	0
1	1
2	0
3	1
4	0
5	0
6	1
7	1
8	0
9	1
10	0

Y_pred

隨堂練習：用標準函式庫訓練與測試

- 請撰寫下列程式碼，並執行之

```
1  from sklearn.linear_model import LogisticRegression
2  import time
3
4  # Model Creation
5  model = LogisticRegression(solver="lbfgs", random_state=int(time.time()))
6
7  # Training
8  model.fit(X_train, Y_train.values.ravel())
9
10 # Testing
11 Y_pred = model.predict(X_test)
```

- 請比對 Y_test 與 Y_pred 兩個陣列的內容，藉以檢測預測的正確性。

評估模型好壞

量測「分類演算法」效能時的問題

- 迴歸演算法
  - 使用「**殘差**」量測效能

真實值	預測值	偏差比例
-0.1318	-0.03942	70.10%
0.487943	0.294085	39.73%
0.684105	0.351052	48.68%
-2.17045	-1.54321	28.90%
-1.98485	-1.719	13.39%
-2.73623	-1.68287	38.50%
1.075907	1.203948	11.90%
-0.40964	-0.31774	22.43%
0.061201	-0.00798	113.04%
-0.52353	-0.44113	15.74%

可以鑒別出差異與大小

- 分類演算法
  - **無法**使用「**殘差**」量測效能

真實值	預測值	偏差比例
1	1	0.00%
1	1	0.00%
1	0	100.00%
0	0	0.00%
1	1	0.00%
0	0	0.00%
0	0	0.00%
1	0	100.00%
0	0	0.00%
0	0	0.00%

無法鑒別出差異與大小

「分類演算法」效能評估法

- 使用「**混淆矩陣（Confusion Matrix）**」

預測值

實際值	$\hat{Y}=0$	$\hat{Y}=1$	…	$\hat{Y}=n$
$Y=0$	個數	個數	…	個數
$Y=1$	個數	個數	…	個數
…	…	…	…	
$Y=n$	個數	個數	…	個數

預測正確個數

衍生的「分類效能指標」

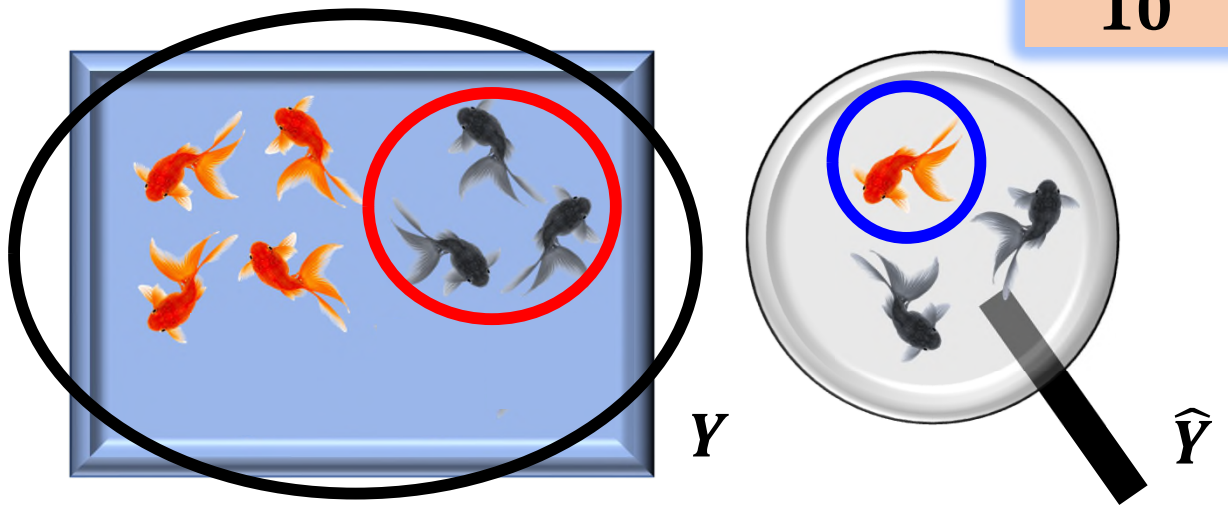
- 確度（準確率 Accuracy）

	$\hat{Y}=0$	$\hat{Y}=1$
$Y=0$	TN	FP
$Y=1$	FN	TP

$$Accuracy = \frac{TP + TN}{All}$$

$$\frac{\text{會買的、你有下廣告} + \text{不買的、你沒下廣告}}{\text{全體受眾}}$$

$$\frac{1+3}{10}$$



衍生的「分類效能指標」

- 廣度（召回率、查全率 Recall、敏感度 Sensitivity）

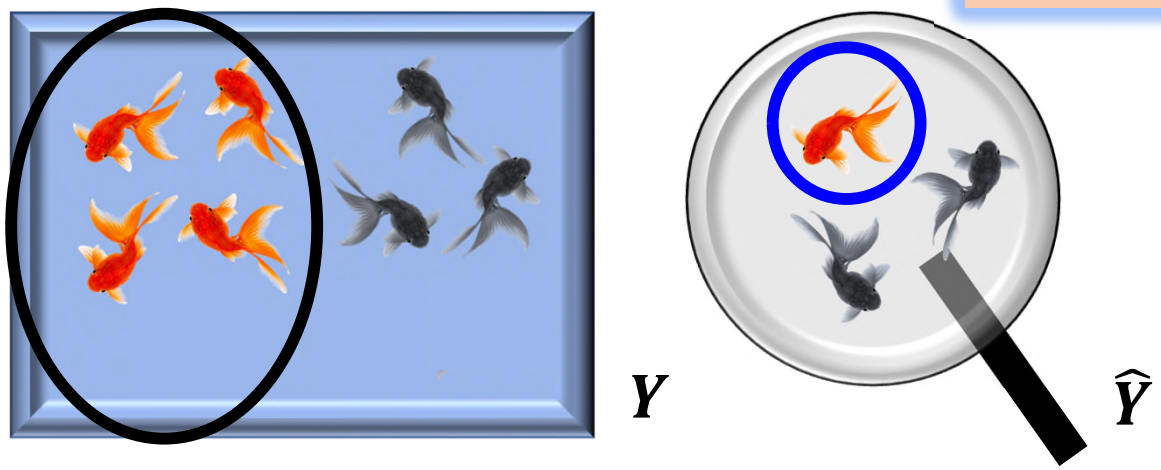
	$\hat{Y}=0$	$\hat{Y}=1$
$Y=0$	TN	FP
$Y=1$	FN	TP

$$Recall = \frac{TP}{TP + FN}$$

希望「廣度」提高的情況：
- 搜尋引擎、廣告投放
- 寧可錯殺一百，不能放過一個

$$\frac{\text{你下了廣告}}{\text{該下廣告的人}} = \text{該撈的、你撈了多少}$$

$$\frac{1}{5}$$



衍生的「分類效能指標」

- 精度（精確度 Precision）

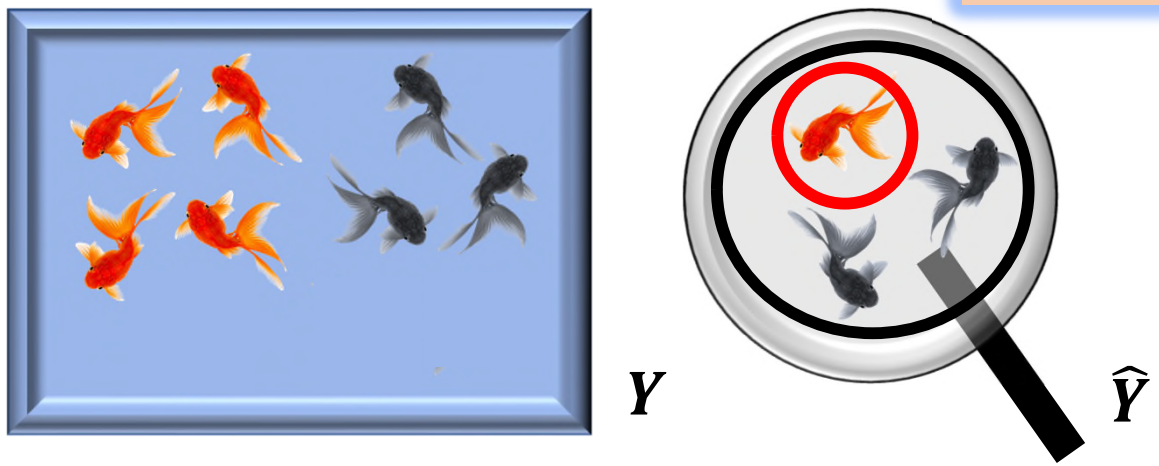
	$\hat{Y}=0$	$\hat{Y}=1$
$Y=0$	TN	FP
$Y=1$	FN	TP

$$Precision = \frac{TP}{TP + FP}$$

希望「精度」提高的情況：
- 癌症檢測、地震預報、金融詐欺
- 寧可不報，不能錯報

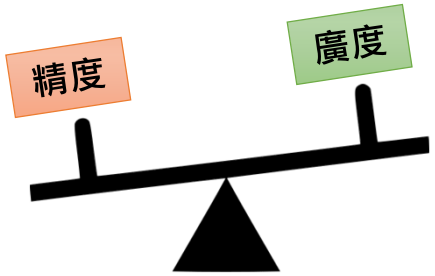
$$\frac{\text{最後真的買了}}{\text{你下了廣告}} = \text{撈到的、你撈對了多少}$$

$$\frac{1}{3}$$



衍生的「分類效能指標」

- F-score（廣度+精度的平衡指標）



$$F-Score = (1+\beta^2)\frac{\text{精度}\cdot\text{廣度}}{\beta^2\text{精度}+\text{廣度}}$$

$$= \frac{(1+\beta^2)}{\beta^2\text{精度}+\text{廣度}\Big/\text{精度}\cdot\text{廣度}}$$

$$= \frac{(1+\beta^2)}{\beta^2\frac{1}{\text{廣度}}+\frac{1}{\text{精度}}} = \text{廣度加權的「調和平均數」}$$

$$\left(\text{調和平均數} = \frac{2}{\frac{1}{A}+\frac{1}{B}}\right)$$

$$\begin{cases}\beta > 1 & \text{希望廣度比較重要}\\ \beta = 1 & \text{希望兩者同等重要}\\ \beta < 1 & \text{希望精度比較重要}\end{cases}$$

- 大部分的情況下，$\beta = 1$
- 當 $\beta = 1$ 時，稱為 **F1-Score**，略稱 **F-Score**

F-Score ↑ = 撈得到 & 撈得準

實作「分類演算法」效能評估

- 原始程式碼：

引入所需的套件

```
1  from sklearn.metrics import confusion_matrix
2  from sklearn.metrics import accuracy_score
3  from sklearn.metrics import precision_score
4  from sklearn.metrics import recall_score
5  from sklearn.metrics import fbeta_score
6
7  print("Confusion Matrix:\n", confusion_matrix(y_true=Y_test, y_pred=Y_pred))
8  print(f"Accuracy: {accuracy_score(Y_test, Y_pred):.2%}")
9  print(f"Recall: {recall_score(Y_test, Y_pred):.2%}")
10 print(f"Precision: {precision_score(Y_test, Y_pred):.2%}")
11 print(f"F1-score: {fbeta_score(Y_test, Y_pred, beta=1):.2%}")
```

[0,0] [[64 2] [0,1]
[1,0] [14 20]] [1,1]

混淆矩陣 →
確度 →
廣度 →
精度 →
F1-Score →

隨堂練習：「分類演算法」效能評估

- 請撰寫下列程式碼，並執行之：

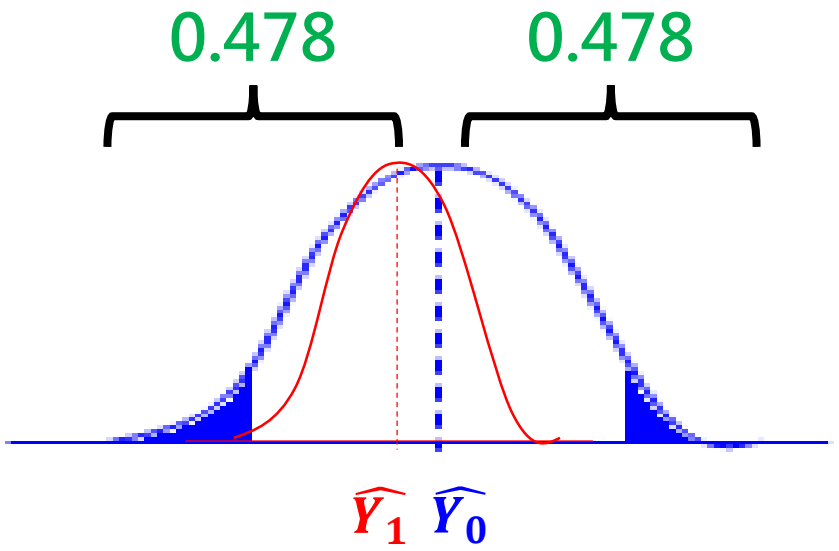
```
1  from sklearn.metrics import confusion_matrix
2  from sklearn.metrics import accuracy_score
3  from sklearn.metrics import precision_score
4  from sklearn.metrics import recall_score
5  from sklearn.metrics import fbeta_score
6
7  print("Confusion Matrix:\n", confusion_matrix(y_true=Y_test, y_pred=Y_pred))
8  print(f"Accuracy: {accuracy_score(Y_test, Y_pred):.2%}")
9  print(f"Recall: {recall_score(Y_test, Y_pred):.2%}")
10 print(f"Precision: {precision_score(Y_test, Y_pred):.2%}")
11 print(f"F1-score: {fbeta_score(Y_test, Y_pred, beta=1):.2%}")
```

- 針對以下主題討論：
  - 對一個**模型**而言（確度、精度），這個結果是好的嗎？
  - 對**廣告投放**這個應用而言（廣度），這個結果是好的嗎？

卡方檢定降維法

為何要學新的「降維演算法」？

- 迴歸專用的「反向淘汰法」



- 以連續函數圖形為基礎，下去計算 p-value
- 不適合分類演算法出現的離散數字

- 分類專用的「卡方檢定法」

	$\hat{Y}=0$	$\hat{Y}=1$
$Y=0$	64	2
$Y=1$	14	20

$$\chi^2=\sum_{i=0}^{1}\sum_{j=0}^{1}\frac{(O_{ij}-E_{ij})^2}{E_{ij}}$$

O_{ij}: 觀察值
E_{ij}: 期望值

- 以混淆矩陣為基礎，下去計算 p-value
- 適合分類演算法中，會出現的離散數字

用標準函式庫執行「卡方降維」

- (1) 找出每個特徵值的 p-value：

引入必要的套件 →

計算每個特徵的 p 值 →

```
1 # Feature Selection (wiht Standard Library)
2 from sklearn.feature_selection import SelectKBest
3 from sklearn.feature_selection import chi2
4
5 kbest = SelectKBest(score_func=chi2, k=1)
6 kbest = kbest.fit(X, Y)
7 print(f"The p-values of Feature Importance: {kbest.pvalues_}")
```

k=n：想保留 n 個特徵
k=1：特別數字，印出所有 p-value

The p-values of Feature Importance: [5.441e-001 4.043e-100 0.000e+000]

用標準函式庫執行「卡方降維」

- (2) 選擇顯著水準 α=0.05，決定保留哪些特徵值：

```
1  # Feature Selection (wiht Standard Library)
2  from sklearn.feature_selection import SelectKBest
3  from sklearn.feature_selection import chi2
4
5  kbest = SelectKBest(score_func=chi2, k= 2 )
6  kbest = kbest.fit(X, Y)
7  print(f"The p-values of Feature Importance: {kbest.pvalues_}")
8
9  X = kbest.transform(X)
```

K=2 之下，留下最佳的兩個欄位 →

X (K=3)

	0	1	2
0	1.0202	-0.246701	1.05227
1	-0.980196	-0.0532101	0.617807
2	1.0202	-1.79463	0.386093
3	1.0202	-0.923921	0.212308
4	-0.980196	-0.536938	1.8343
5	1.0202	-0.149956	-0.569726

X (K=2)

	0	1
0	-0.246701	1.05227
1	-0.0532101	0.617807
2	-1.79463	0.386093
3	-0.923921	0.212308
4	-0.536938	1.8343
5	-0.149956	-0.569726

在顯著水準 **α = 0.05** 之下　　[0.5441　0.000…04043　0.0000]

The p-values of Feature Importance: [**5.441e-001 4.043e-100 0.000e+000**]

用標準函式庫執行「卡方降維」

- (3) 重跑模型，比較效能

```
[[64  2]
 [14 20]]
```

K=3 的效能

Accuracy: 84.00% Recall: 58.82% Precision: 90.91% F1-Score: 71.43%

皆提昇一點點，
且運算量減低

```
[[64  4]
 [11 21]]
```

K=2 的效能

Accuracy: 85.00% Recall: 61.76% Precision: 91.30% F1-Score: 73.68%

注意！！

- 有時候**移除**不顯著的**特徵**，正確率反而會**下降**。
- 這會因為每次挑選的訓練集**樣本點不同**而不同。
- 只要特徵值變多變雜，你就能看出「卡方降維」的**好處**。
- 只要正確率下降不多，特徵值變少，也有**計算量下降**的好處！

隨堂練習：用標準函式庫實作

- 請撰寫**紅框部分**程式碼，並執行看看，您的模型效能是否有提昇？

```python
# In[] Preprocessing
import HappyML.preprocessor as pp

# Load Dataset
dataset = pp.dataset(file="Social_Network_Ads.csv")

# X, Y Decomposition
X, Y = pp.decomposition(dataset, x_columns=[1, 2, 3], y_columns=[4])

# Categorical Data Encoding & Remove Dummy Variable Trap
X = pp.onehot_encoder(X, columns=[0], remove_trap=True)

# Feature Selection (wiht Standard Library)
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

kbest = SelectKBest(score_func=chi2, k=2)
kbest = kbest.fit(X, Y)
print(f"The p-values of Feature Importance: {kbest.pvalues_}")

X = kbest.transform(X)

# Split Training & Testing set
X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y)

# Feature Scaling
X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```

AI

使用「快樂版」
實作

以快樂版重做「邏輯迴歸」

- 原始碼講解（1）：

/HappyML/regression.py

引入必要的套件

類別的成員變數

建構函數

regressor 成員變數的 getter

```
1  from sklearn.linear_model import LogisticRegression
2  import time
3
4  class LogisticRegressor:
5      __regressor = None
6      __solver = "lbfgs"
7
8      def __init__(self, solver="lbfgs"):
9          if solver not in ("liblinear", "lbfgs", "sag", "saga", "newton-cg"):
10             self.__solver = "lbfgs"
11         self.__regressor = LogisticRegression(solver=self.solver, random_state=int(time.time()))
12
13     @property
14     def regressor(self):
15         return self.__regressor
```

以快樂版重做「邏輯迴歸」

- 原始碼講解（２）：

/HappyML/regression.py

solver 成員變數的 getter

邏輯迴歸模型訓練

邏輯迴歸模型預測

```
17      @property
18      def solver(self):
19          return self.__solver
20
21      def fit(self, x_train, y_train):
22          if y_train.ndim > 1:
23              y_train = y_train.ravel()
24
25          self.regressor.fit(x_train, y_train)
26          return self
27
28      def predict(self, x_test):
29          return self.regressor.predict(x_test)
```

以快樂版重做「邏輯迴歸」

- 呼叫範例

套件引入

物件生成、
模型訓練＆預測

```
1  from HappyML.regression import LogisticRegressor
2
3  # Training & Predict
4  model = LogisticRegressor()
5  Y_pred = model.fit(X_train, Y_train).predict(X_test)
```

- 執行結果

真實值
Y_test

Index	Purchased
366	1
51	0
332	0
159	1
157	0
394	0
261	1
252	1
178	0

Index	Purchased
366	1
51	0
332	0
159	0
157	0
394	0
261	1
252	1
178	0

預測值
Y_pred

隨堂練習：以快樂版重做「邏輯迴歸」

- 請把前一個練習中，用「標準函式庫」實作、包含下列部分的程式碼註解掉：
  - 「卡方降維」相關程式碼
  - 「邏輯迴歸」訓練、預測相關程式碼
  - 「效能計算」相關程式碼
- 將下列程式碼寫到原先「邏輯迴歸」訓練＆預測的程式碼位置，並且在執行後、用肉眼比較 Y_test 與 Y_pred 看看：

```
1  from HappyML.regression import LogisticRegressor
2
3  # Training & Predict
4  model = LogisticRegressor()
5  Y_pred = model.fit(X_train, Y_train).predict(X_test)
```

以快樂版計算「分類模型效能」

- 原始碼解說：

/HappyML/performance.py

引入必要的套件

類別的成員變數

建構函數

混淆矩陣

確度（Accuracy）

廣度（Recall）

精度（Precision）

F-Score

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import fbeta_score

class ClassificationPerformance:
    __y_real = None
    __y_pred = None

    def __init__(self, y_real, y_pred):
        self.__y_real = y_real
        self.__y_pred = y_pred

    def confusion_matrix(self):
        return confusion_matrix(self.__y_real, self.__y_pred)

    def accuracy(self):
        return accuracy_score(self.__y_real, self.__y_pred)

    def recall(self):
        return recall_score(self.__y_real, self.__y_pred)

    def precision(self):
        return precision_score(self.__y_real, self.__y_pred)

    def f_score(self, beta=1):
        return fbeta_score(self.__y_real, self.__y_pred, beta)
```

以快樂版計算「分類模型效能」

- 呼叫範例

套件引入

製造效能計算物件

印出混淆矩陣、確度、廣度、精度、F-Score 等效能指標

```python
from HappyML.performance import ClassificationPerformance

pfm = ClassificationPerformance(Y_test, Y_pred)

print("Confusion Matrix:\n", pfm.confusion_matrix())
print(f"Accuracy: {pfm.accuracy():.2%}")
print(f"Recall: {pfm.recall():.2%}")
print(f"Precision: {pfm.precision():.2%}")
print(f"F1-score: {pfm.f_score():.2%}")
```

- 執行結果

```
Confusion Matrix:
 [[64  2]
 [13 21]]
Accuracy: 85.00%
Recall: 61.76%
Precision: 91.30%
F1-score: 73.68%
```

隨堂練習：以快樂版計算「分類模型效能」

- 請將下列程式碼，寫到先前「效能計算」的位置，並執行看看：

```python
from HappyML.performance import ClassificationPerformance

pfm = ClassificationPerformance(Y_test, Y_pred)

print("Confusion Matrix:\n", pfm.confusion_matrix())
print(f"Accuracy: {pfm.accuracy():.2%}")
print(f"Recall: {pfm.recall():.2%}")
print(f"Precision: {pfm.precision():.2%}")
print(f"F1-score: {pfm.f_score():.2%}")
```

以快樂版執行「卡方降維」

- 原始碼解說（1）：

/HappyML/preprocessor.py

引入必要的套件

類別的成員變數

建構函數

selector 成員變數的 getter

significance 成員變數的 getter

```python
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

class KBestSelector:
    __selector = None
    __significance = None
    __best_k = None
    __strategy = None

    def __init__(self, significance=0.05, best_k="auto"):
        self.significance = significance

        if type(best_k) is int:
            self.__strategy = "fixed"
            self.best_k = best_k
        else:
            self.__strategy = "auto"
            self.best_k = 1

        self.__selector = SelectKBest(score_func=chi2, k=self.best_k)

    @property
    def selector(self):
        return self.__selector

    @property
    def significance(self):
        return self.__significance
```

顯著水準

best_k 挑選模式

← 儲存顯著水準

← 儲存best_k模式

↓ 產生降維物件

以快樂版執行「卡方降維」

- 原始碼解說（２）：

significance 成員變數的 setter

best_k 成員變數的 getter & setter

```
30      @significance.setter
31      def significance(self, significance):
32          if significance > 0:
33              self.__significance = significance
34          else:
35              self.__significance = 0.05
36
37      @property
38      def best_k(self):
39          return self.__best_k
40
41      @best_k.setter
42      def best_k(self, best_k):
43          if best_k >= 1:
44              self.__best_k = best_k
45          else:
46              self.__best_k = 1
```

以快樂版執行「卡方降維」

- 原始碼解說（3）：

自變數 & 應變數　　是否顯示逐步內容　　p-value 是否排序

```
46      def fit(self, x_ary, y_ary, verbose=False, sort=False):
47          #Get the scores of every feature
48          kbest = SelectKBest(score_func=chi2, k="all")
49          kbest = kbest.fit(x_ary, y_ary)
50
51          # if auto, choose the best K features
52          if self.__strategy == "auto":
53              sig_ary = np.full(kbest.pvalues_.shape, self.significance)
54              feature_selected = np.less_equal(kbest.pvalues_, sig_ary)
55              self.best_k = np.count_nonzero(feature_selected == True)
56
57          # if verbose, show additional information
58          if verbose:
59              print("\nThe Significant Level: {}".format(self.significance))
60              p_values_dict = dict(zip(x_ary.columns, kbest.pvalues_))
61              print("\n--- The p-values of Feature Importance ---")
62
63              # if sorted, rearrange p-values in ascending order
64              if sort:
65                  name_pvalue = sorted(p_values_dict.items(), key=lambda kv: kv[1])
66              else:
67                  name_pvalue = [(k, v) for k, v in p_values_dict.items()]
```

計算各特徵的 p 值

best_k 交由程式決定的狀況

製作「顯著水準」的常數陣列 →
比大小：p 值陣列 <= 顯著水準陣列 →
計算 True（<=0.05）有幾個，以決定要留幾個特徵 →

如果需要逐步顯示內容

印出顯著水準
製作各特徵值 p-value 字典

是否遵照 p-value 排序

以快樂版執行「卡方降維」

- 原始碼解說（４）：

如果需要
逐步顯示內容

印出哪些特徵應保留，
p 值多少，及欄位名稱

印出最後保留多少個特徵值 →

用 best_k 計算哪些特徵要保留

刪除不保留的特徵

```
69              # Show each feature and its p-value
70              for k, v in name_pvalue:
                    sig_str = "TRUE  <" if v <= self.significance else "FALSE >"
                    sig_str += "{:.2f}".format(self.significance)
73                  print("{} {:.8e} ({})".format(sig_str, v, k))
74
75              # Show how many features have been selected
                print("\nNumber of Features Selected: {}".format(self.best_k))
77
78          # Start to select features
79          self.__selector = SelectKBest(score_func=chi2, k=self.best_k)
            self.__selector = self.__selector.fit(x_ary, y_ary)
81
82          return self
83
84
85      def transform(self, x_ary):
86          # indices=True will return an NDArray of integer for selected columns
87          cols_kept = self.selector.get_support(indices=True)
88          return x_ary.iloc[:, cols_kept]
```

以快樂版執行「卡方降維」

- 呼叫範例

在「資料前處理」處
以快樂版「降維」

```
1  import HappyML.preprocessor as pp
2
3  # Load Dataset
4  dataset = pp.dataset(file="Social_Network_Ads.csv")
5
6  # X, Y Decomposition
7  X, Y = pp.decomposition(dataset, x_columns=[1, 2, 3], y_columns=[4])
8
9  # Categorical Data Encoding & Remove Dummy Variable Trap
10 X = pp.onehot_encoder(X, columns=[0], remove_trap=True)
11
12 # Feature Selection (with HappyML)
13 from HappyML.preprocessor import KBestSelector
14 selector = KBestSelector()
15 X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)
16
17 # Split Training & Testing set
18 X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y)
19
20 # Feature Scaling
21 X_train, X_test = pp.feature_scaling(fit_ary=X_train, transform_arys=(X_train, X_test))
```

以快樂版執行「卡方降維」

- 執行結果

快樂版「卡方降維」
(verbose=True)
(sort=True)

```
The Significant Level: 0.05

--- The p-values of Feature Importance ---
TRUE  <0.05 0.00000000e+00 (EstimatedSalary)
TRUE  <0.05 4.04303193e-100 (Age)
FALSE >0.05 5.44126248e-01 (Gender_Male)

Number of Features Selected: 2
Confusion Matrix:
 [[54  5]
 [14 27]]
Accuracy: 81.00%
Recall: 78.69%
Precision: 81.89%
F1-score: 79.51%
```

隨堂練習：以快樂版執行「卡方降維」

- 請先看懂講師提供的下列類別程式碼：
  - HappyML.preprocessor.**KBestSelector**
- 於「**資料前處理**」程式區塊，輸入下列程式碼，並執行看看：

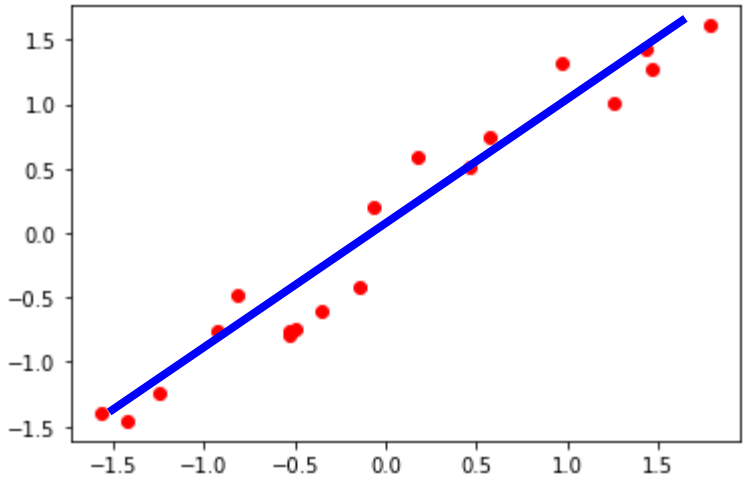
```
1  from HappyML.preprocessor import KBestSelector
2
3  selector = KBestSelector()
4  X = selector.fit(x_ary=X, y_ary=Y, verbose=True, sort=True).transform(x_ary=X)
```

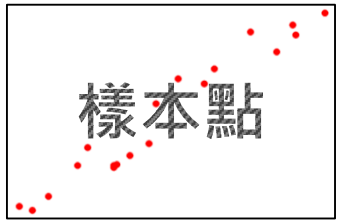
- 下次使用時，可以把 verbose=**False**（或乾脆拿掉）試試看

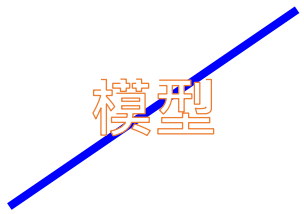
將結果視覺化

為何不能用「迴歸模型」的視覺化函數？

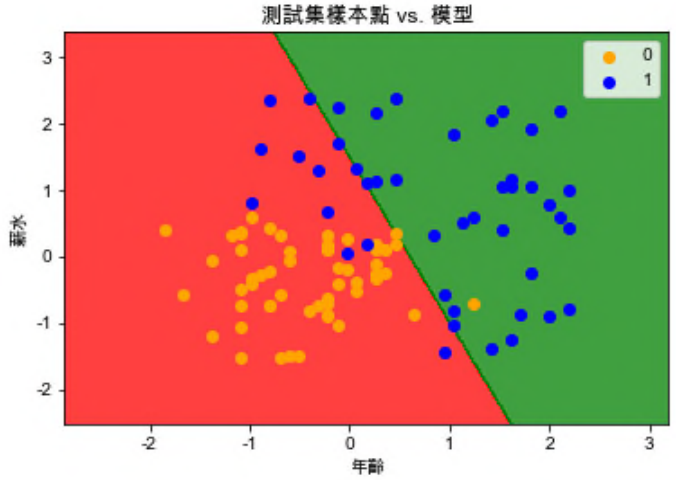
- 「迴歸模型」視覺化

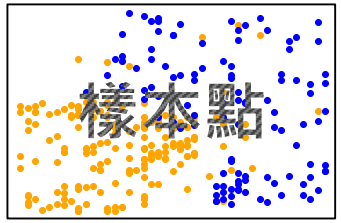


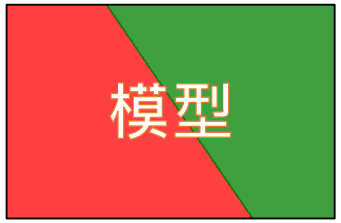




- 「分類模型」視覺化







分類模型視覺化的「快樂版程式碼」

- 原始碼解說

自變數、應變數 模型本身

指定前景色、與背景色

圖片標題、與字型名稱

```
1  def classify_result(x, y, classifier, fg_color=("orange", "blue"), bg_color=("red", "green"), title="", font='Arial Unicode MS'):
2      # Get the xlabel & ylabel first
3      xlabel = x.columns[0]
4      ylabel = x.columns[1]
5
6      # Prepare each dot of background
7      x = x.values
8      y = y.values
9      x_axis_range = np.arange(x[:, 0].min()-1, x[:, 0].max()+1, 0.01)
10     y_axis_range = np.arange(x[:, 1].min()-1, x[:, 1].max()+1, 0.01)
11     X_background, Y_background = np.meshgrid(x_axis_range, y_axis_range)
12
13     # Limit the range of drawing
14     plt.xlim(X_background.min(), X_background.max())
15     plt.ylim(Y_background.min(), Y_background.max())
16
17     # Draw the dots of background (as the predicting result)
18     Target_predict = pd.DataFrame(classifier.predict(pd.DataFrame(np.array([X_background.ravel(), Y_background.ravel()]).T))).values.reshape(X_background.shape)
19     plt.contourf(X_background, Y_background, Target_predict, alpha=0.75, cmap=ListedColormap(bg_color))
20
21     # Draw the sample data in dots
22     # Iterate all types of Targets (e.g. Y_real = 0, Y_real = 1, ...)
23     for y_real_index, y_real in enumerate(np.unique(y)):
24         row_selector = y.reshape(x.shape[0]) # y.ndim =2, we need 1D array to select rows of X
25         plt.scatter(x[row_selector == y_real, 0], x[row_selector == y_real, 1], c=[ListedColormap(fg_color)(y_real_index)], label=y_real)
26
27     # Set the Title & Label
28     # for showing Chinese characters
29     plt.rcParams['font.sans-serif']=[font]
30     plt.rcParams['axes.unicode_minus'] = False
31
32     plt.title(title)
33     plt.xlabel(xlabel)
34     plt.ylabel(ylabel)
35     plt.legend(loc="best")
36     plt.show()
```

← 取出 DataFrame 的欄位名稱，做為 X、Y 軸的軸線名稱

← X[0](年齡)：從最小值～最大值，每隔 0.01 打一點
X[1](薪資)：從最小值～最大值，每隔 0.01 打一點
← 把 X[0]、X[1] 兩個一維陣列，張開成滿滿的二維陣列，當背景點

← 限制作圖範圍，在剛剛張開、打滿背景的二維陣列點之範圍內

↓ 計算每個二維陣列點的 Y 值（背景點著色用）

↑ 每個背景點若模型預測為 0，著紅色。預測為 1，著綠色

↑ y_real 會迭代 Y 的所有可能答案（目前 = 0 & 1）
第一次迴圈，找所有樣本點預測為 0 的 X 值（含年齡、薪資），用橘色畫出。
第二次迴圈，找所有樣本點預測為 1 的 X 值（含年齡、薪資），用藍色畫出。

中文字型設定

圖形名稱
軸線標籤

分類模型視覺化的「快樂版程式碼」

- 呼叫範例

```
1  import HappyML.model_drawer as md
2
3  md.classify_result(x=X_train, y=Y_train, classifier=model.regressor,
4                     title="訓練集樣本點 vs. 模型", font="DFKai-sb")
5  md.classify_result(x=X_test, y=Y_test, classifier=model.regressor,
6                     title="測試集樣本點 vs. 模型", font="DFKai-sb")
```

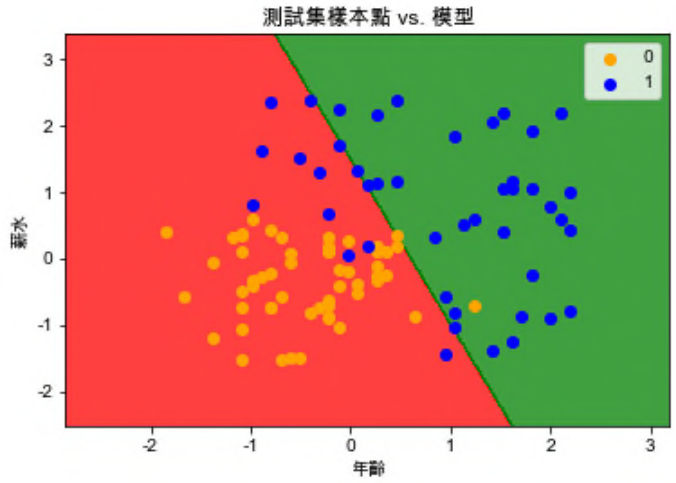
傳入 X, Y

傳入你做好的模型

圖形標題

指定字型

- 執行結果



使用時注意：

- **一定要降到二維：**
  因為只有兩個座標軸。
- **一定要做特徵縮放：**
  這樣底圖才會接近矩型。

隨堂練習：分類模型視覺化的「快樂版程式碼」

- 請先看懂講師提供的 **classify_result**() 函數。

- 撰寫下列程式碼，並執行看看：

```
1  import HappyML.model_drawer as md
2
3  md.classify_result(x=X_train, y=Y_train, classifier=model.regressor,
4                     title="訓練集樣本點 vs. 模型", font="DFKai-sb")
5  md.classify_result(x=X_test, y=Y_test, classifier=model.regressor,
6                     title="測試集樣本點 vs. 模型", font="DFKai-sb")
```

課後作業：乳癌資料處理&預測

- 說明：
  - sklearn 內有一份 569 筆的真實的乳癌檢測資料。載入方法如下：
    - from sklearn.datasets import load_breast_cancer
    - dataset = load_breast_cancer()
  - dataset 的欄位說明如下：
    - dataset.**DESCR**：文字檔。說明整個資料集的用法，與各欄位的意義。
    - dataset.**data**：自變數 X，共有 30 個特徵（Features）
    - dataset.**target**：應變數 Y，檢查結果是否為良性。0=惡性（有乳癌）、1=良性（沒有乳癌）。
    - dataset.**feature_names**：欄位名稱。供參考。可用可不用

- 要求：
  - 請用「**邏輯迴歸**」，訓練出一個模型。注意！要能挑選出「**夠顯著**」的特徵來！並**印出**你選擇了哪些特徵？（提示：使用預設參數 best_k= “auto” 即可）
  - 使用「**測試集**」的資料，印出「**混淆矩陣**」
  - 計算你模型的**確度、廣度、精度、F-Score**，並且把它印出來

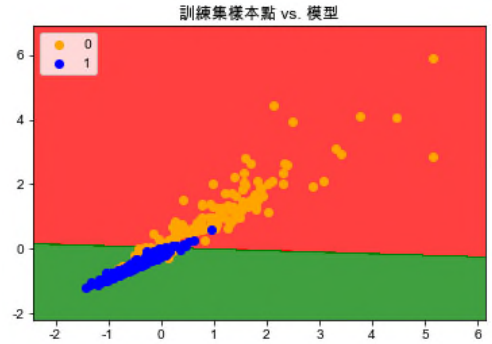
課後作業：乳癌資料處理＆預測

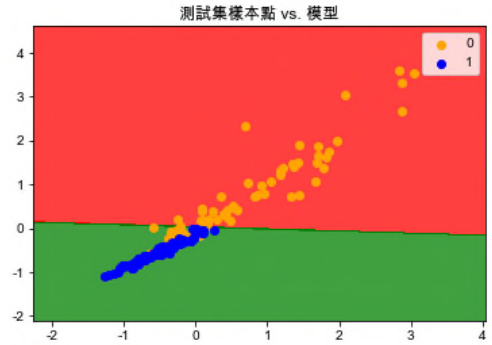
- 要求：
  - 用 KBestSelector 將特徵壓至 2 個，並另外訓練一個 Logistic Regressor。
  - 用這個「僅有兩個特徵」的資料集與模型，繪製出「訓練集」與「測試集」的分類結果。
- 輸出結果：

```
The Significant Level: 0.05

--- The p-values of Feature Importance ---
TRUE  <0.05 0.00000000e+00 (mean perimeter)
TRUE  <0.05 0.00000000e+00 (mean area)
TRUE  <0.05 0.00000000e+00 (area error)
TRUE  <0.05 0.00000000e+00 (worst perimeter)
TRUE  <0.05 0.00000000e+00 (worst area)
TRUE  <0.05 6.11324751e-109 (worst radius)
TRUE  <0.05 8.01397628e-60 (mean radius)
TRUE  <0.05 1.94877489e-56 (perimeter error)
TRUE  <0.05 7.89668299e-40 (worst texture)
TRUE  <0.05 3.32292194e-22 (mean texture)
TRUE  <0.05 3.25230064e-10 (worst concavity)
TRUE  <0.05 3.89553429e-09 (radius error)
TRUE  <0.05 9.00175712e-06 (mean concavity)
TRUE  <0.05 1.10836762e-05 (worst compactness)
TRUE  <0.05 2.40424384e-04 (worst concave points)
TRUE  <0.05 1.16563638e-03 (mean concave points)
TRUE  <0.05 2.01012999e-02 (mean compactness)
FALSE >0.05 2.54421307e-01 (worst symmetry)
FALSE >0.05 3.06726812e-01 (concavity error)
FALSE >0.05 4.33366115e-01 (compactness error)
FALSE >0.05 5.28452867e-01 (worst smoothness)
FALSE >0.05 5.80621137e-01 (concave points error)
FALSE >0.05 6.11926026e-01 (mean symmetry)
FALSE >0.05 6.30397277e-01 (worst fractal dimension)
FALSE >0.05 6.98631644e-01 (mean smoothness)
FALSE >0.05 9.21168192e-01 (texture error)
FALSE >0.05 9.36379753e-01 (fractal dimension error)
FALSE >0.05 9.54425121e-01 (smoothness error)
FALSE >0.05 9.92847410e-01 (symmetry error)
FALSE >0.05 9.93122221e-01 (mean fractal dimension)

Number of Features Selected: 17
Confusion Matrix:
 [[59  4]
 [ 0 80]]
Accuracy: 97.20%
Recall: 96.83%
Precision: 97.62%
F1-score: 97.14%
```





重點整理：邏輯迴歸

- 何時使用
  - 當**應變數 Y** 並非連續數字、而是「**類別資料**」時

- 擬合方程式
  - $ln\left(\frac{p}{1-p}\right) = c_0 + c_1X_1 \cdots c_nX_n$

- 相關套件
  - 邏輯迴歸：sklearn.linear_model.LogisticRegression 類別
  - 混淆矩陣：sklearn.metrics.confusion_matrix() 函數
  - 特徵選擇：sklearn.feature_selection.SelectKBest() 函數
  - 卡方檢定：sklearn.feature_selection.chi2 類別