Group 1

心臟病發生原因預測模型之建構與分類

指導教授：邱銘傳教授

109034553 莊婉琦
109034570 林溥鈞
109034402 陳光源

01 研究主題說明

02 資料前處理

03 預測模型設定

04 分析與模型效度驗證

05 結論與未來展望

01

研究主題說明

研究動機、目的

- 心臟疾病常為國人的前十大死因之一
- 突發性、無預警，若發病常造成直接的死亡
- 在靜態或休息狀態下毫無症狀的，需要更進一步的檢查或追蹤才能及早發現及早治療及預防

藉由Kaggle上心臟資料庫裡面所記錄的相關身理特徵，找出導致心臟疾病的相關因子及可能原因，並建立預測模型

5W1H

What
分類罹患心臟病的身理特徵

Where
醫院心臟科門診

Who
具有相關症狀的患者、心臟科醫生

Why
心臟病需透過詳細的檢查才能被發現，
常常一發作就導致死亡

When
心臟科醫生問診時

How
機器學習、資料分析、決策樹分析

資料欄位

欄位	定義	資料型別
age	年齡	連續型
sex	性別	0 = 女生 1 = 男生
cp	胸部疼痛類型	1 = 典型心絞痛 2 = 非典型心絞痛 3 = 非心絞痛 4 = 無症狀的
trestbps	靜息血壓	連續型 (unit: mmHg)
chol	膽固醇	連續型 (unit: mg/dl)
fbs	空腹血糖	1 = True (> 120mg/dl) 2 = False (<= 120mg/dl)
restecg	靜息心電圖測量	0 = 正常 1 = ST-T波異常 2 = 可能或確定左心室肥大
thalach	達到的最大心率	連續型

欄位	定義	資料型別
exang	運動誘發心絞痛	1 = Yes 2 = No
oldpeak	運動引起的S-T下降相較於休息	連續型 & float
slope	最高S-T段的斜率	1 = 向上 2 = 平坦 3 = 向下
ca	主要血管	0 - 3
thal	地中海貧血 (說明: 一種先天性貧血)	3 = 正常 6 = 固定缺陷 7 = 可逆缺陷
target	心臟病	0 = No 1 = Yes

02

資料前處理

1. Data Analysis

2. Feature Selection

3. Standardization

4. Encoding

Data Analysis - importing

- 將csv檔以dataframe 導入
- .head() 檢查前幾項數據欄位
- .shape() 查看整筆數據欄位與行數
- .groupby('target') 檢查有無心臟病分布數量
- .info() 檢查有沒有缺值(Non-Null)與資料類別(Dtype)

```
df = pd.read_csv('heart.csv')
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
[5] print('Heart Disease data set dimension: {}'.format(df.shape))
    df.groupby('target').size()
```

```
Heart Disease data set dimension: (303, 14)
target
0    138
1    165
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

➔ 發現資料無缺失

➔ 有303筆資料與14個欄位

➔ 資料類別除了一欄為浮點數其他為整數型但有連續與離散之分

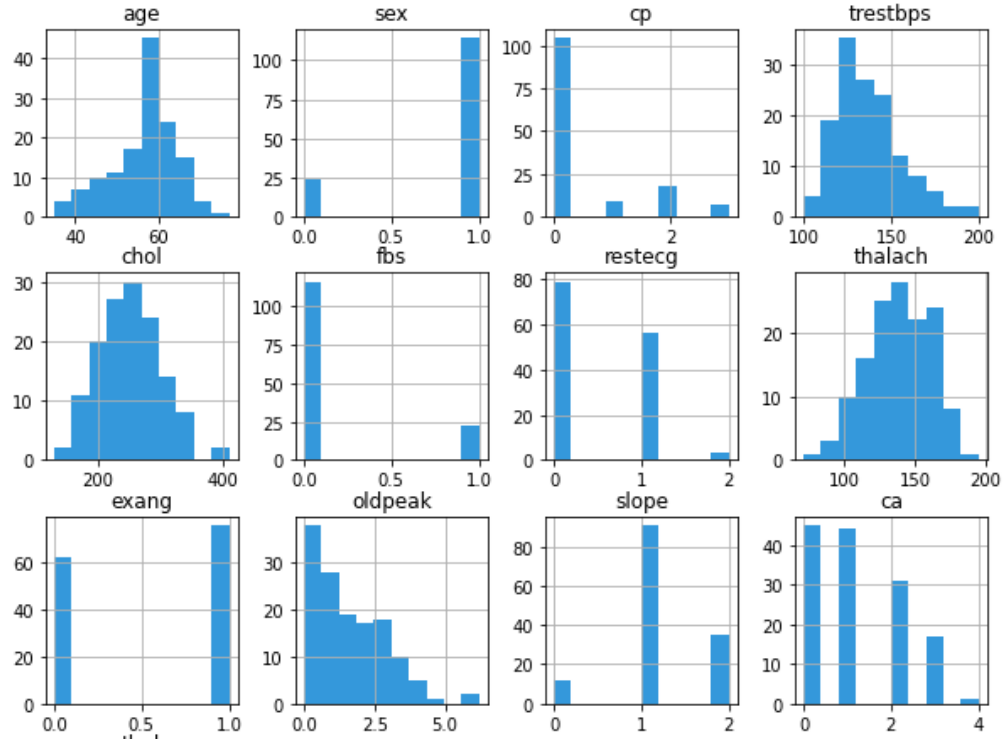
→ 需要標準化

➔ 無心臟病 138 筆

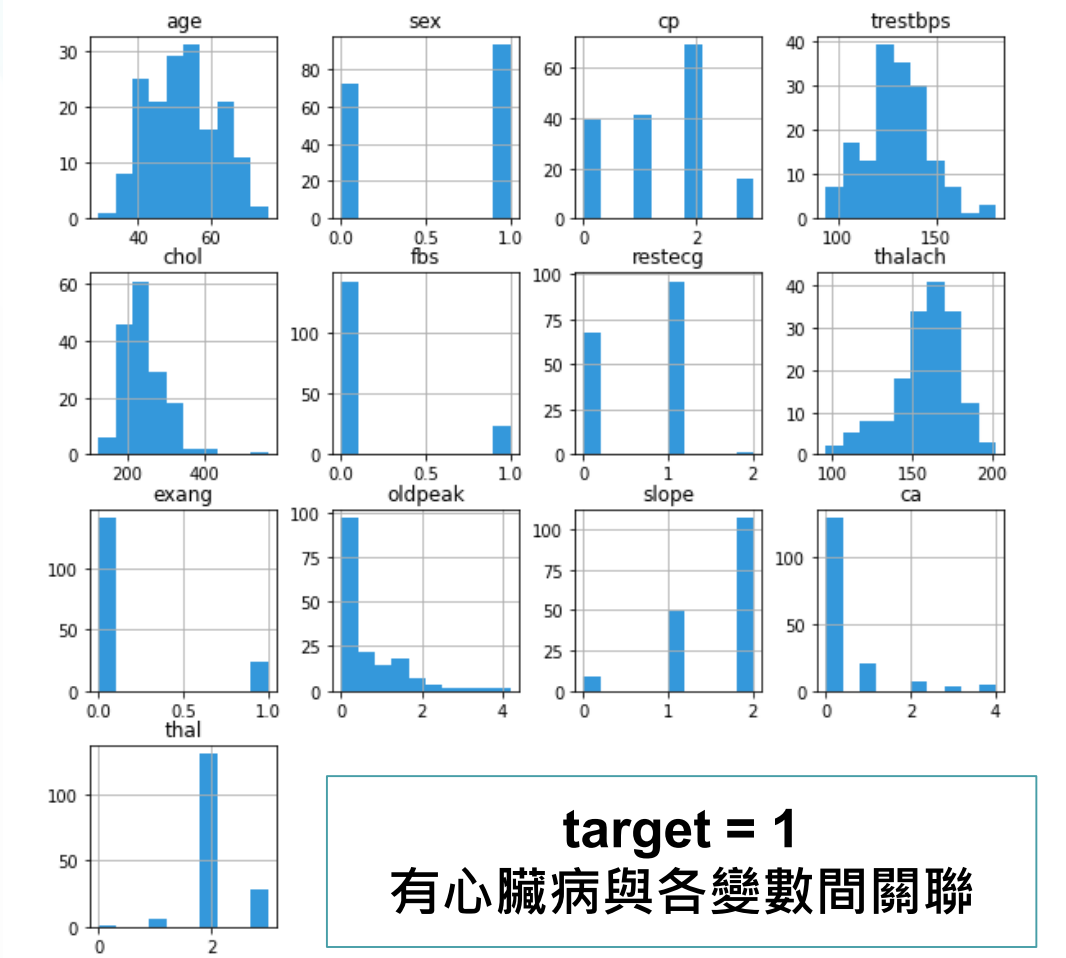
➔ 有心臟病 165 筆

Data Analysis - visulizing

利用pandas的groupby()視覺化工具，
查看各欄位在有無心臟病時的資料分佈



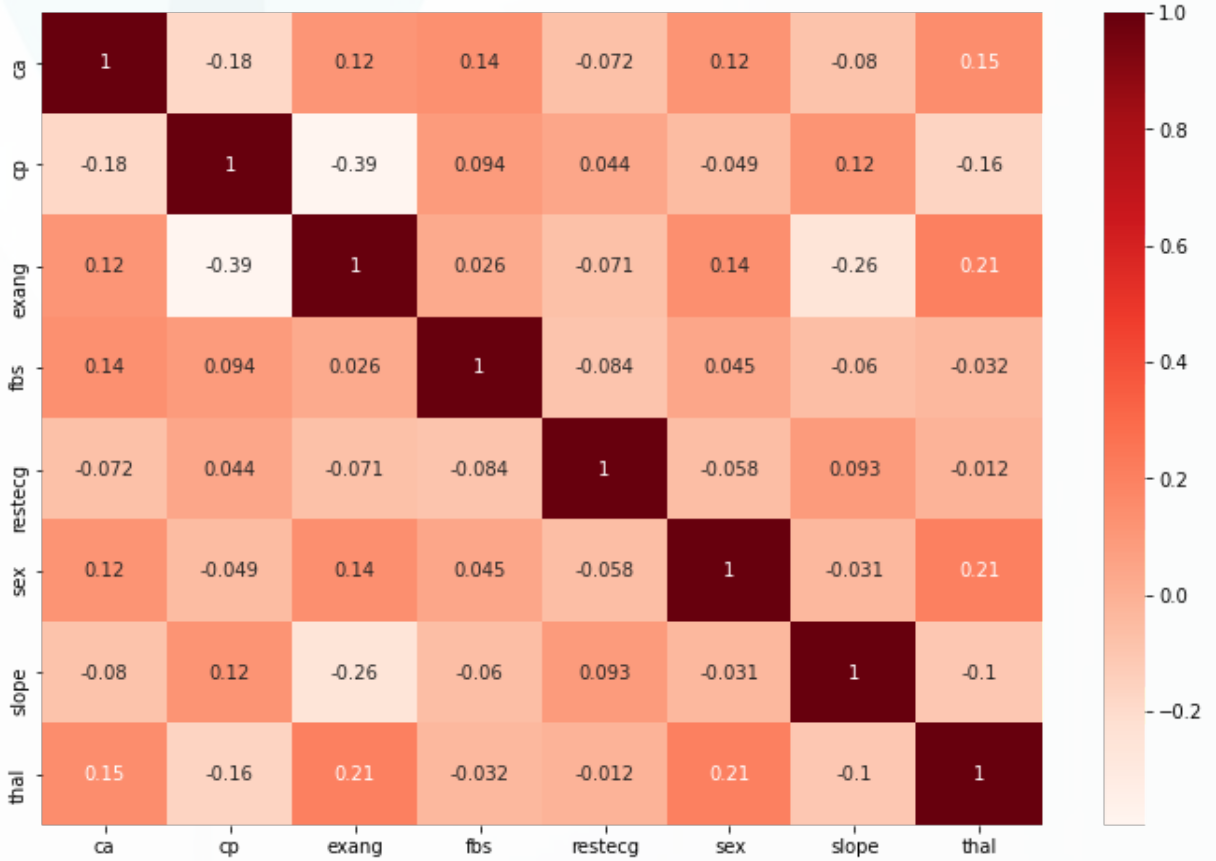
target = 0
無心臟病與各變數間關聯



target = 1
有心臟病與各變數間關聯

Data Analysis - correlation & Feature Selection

- ➔ **發現與target有正相關的欄位:**
  - **cp(0.43)**: 有發生過心絞痛的人較易有心臟病
  - **thalach(0.42)**: 心率越高較亦有心臟病
  - **slope(0.35)**: 心電圖最高S-T段的斜率越大越亦有心臟病
- ➔ **其他:**
  - cp 0.3 talch
  - talach 0.39 slope
  - oldpeak 0.29 exang
  - age 0.28 ca
  - age 0.28 trestbps



Data Analysis & Feature Selection

```
df[['cp', 'target']].groupby(['cp'],  as_index=False).mean().sort_values(by='target', ascending=False)
```

```
df[['thal', 'target']].groupby(['thal'], as_index=False).mean().sort_values(by='target', ascending=False)
```

```
df[['slope', 'target']].groupby(['slope'],  as_index=False).mean().sort_values(by='target', ascending=False)
```

	cp	target	
1	1	0.820000	非典型心絞痛
2	2	0.793103	非心絞痛
3	3	0.695652	無症狀的
0	0	0.272727	典型心絞痛

	thal	target	
2	2	0.783133	可逆缺陷
0	0	0.500000	正常
1	1	0.333333	固定缺陷
3	3	0.239316	其他

	slope	target	
2	2	0.753521	向下
0	0	0.428571	向上
1	1	0.350000	平坦

Encoding, Standardizing

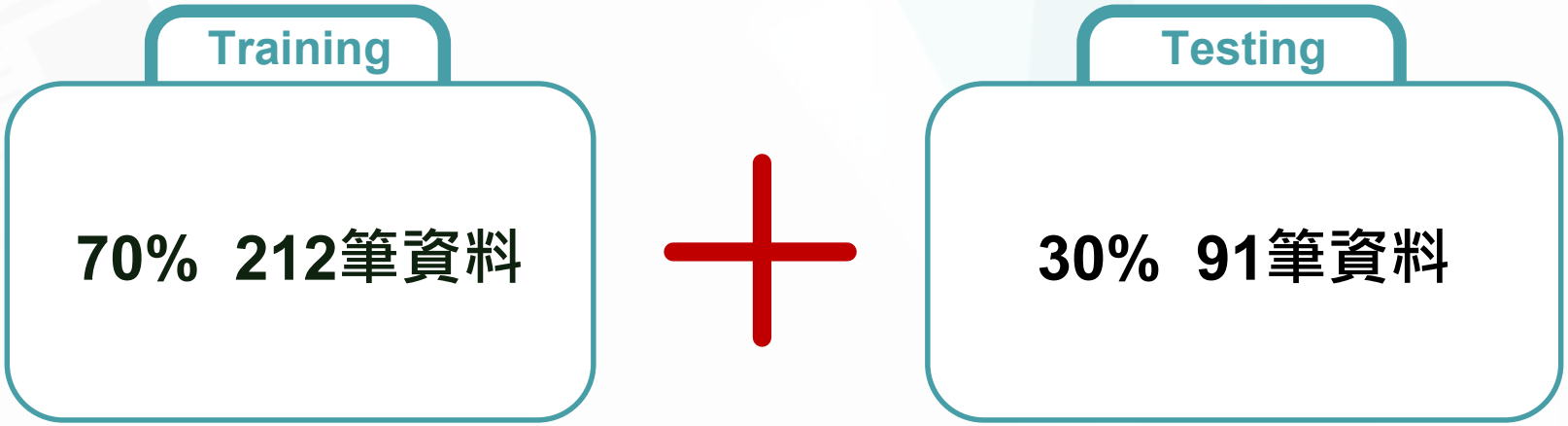
Encoding離散型欄位

```
# encoding
col_dis = ['ca', 'cp', 'exang', 'fbs', 'restecg', 'sex', 'slope', 'thal']
df = pd.get_dummies(df, columns=col_dis, drop_first=True)
```

標準化資料集

```
from sklearn.preprocessing import StandardScaler
df_std = pd.DataFrame(df, columns=df.columns)
X_std = df_std.drop(['target'], axis=1)
y_std = df_std['target']
df_std = StandardScaler().fit_transform(df_std)
```

資料前處理 - 區分特徵值欄位與目標欄位




```
# define feature & target
X = df.loc[:, df.columns != 'target']
y = df['target']
# Spliting dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
```

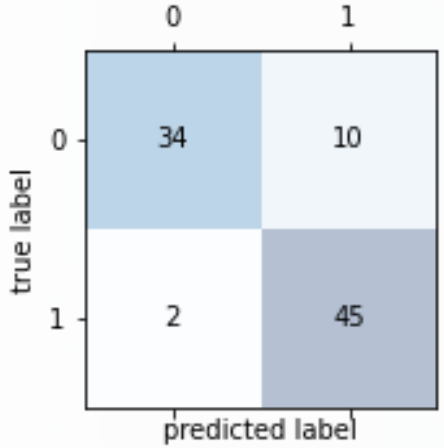
預測模型設定

Decision Tree
決策樹演算法

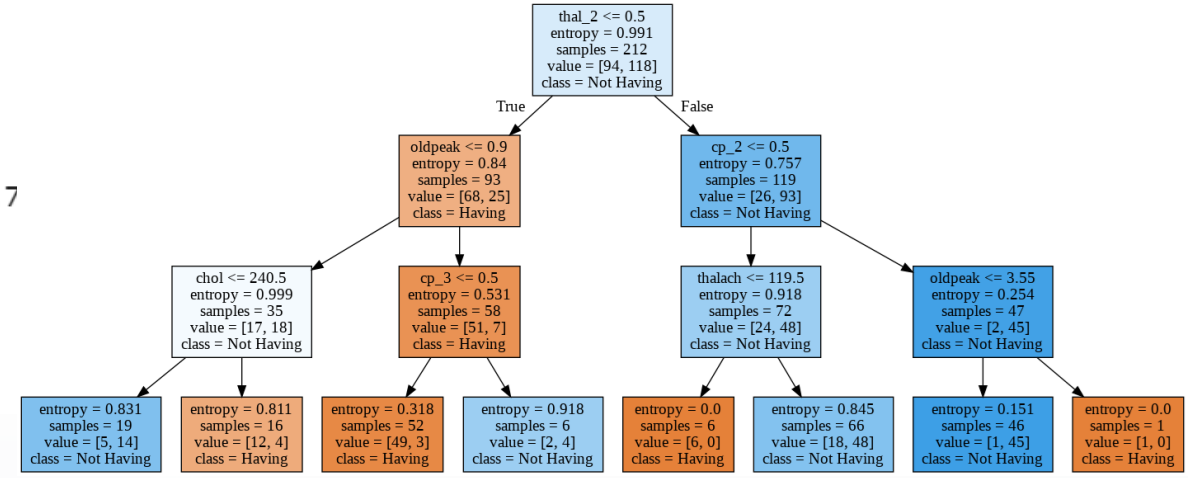
決策樹

利用 分枝準則選擇 **entropy** 、樹深度 **max_depth為 3** 與 **random_state 為 101**來訓練模型

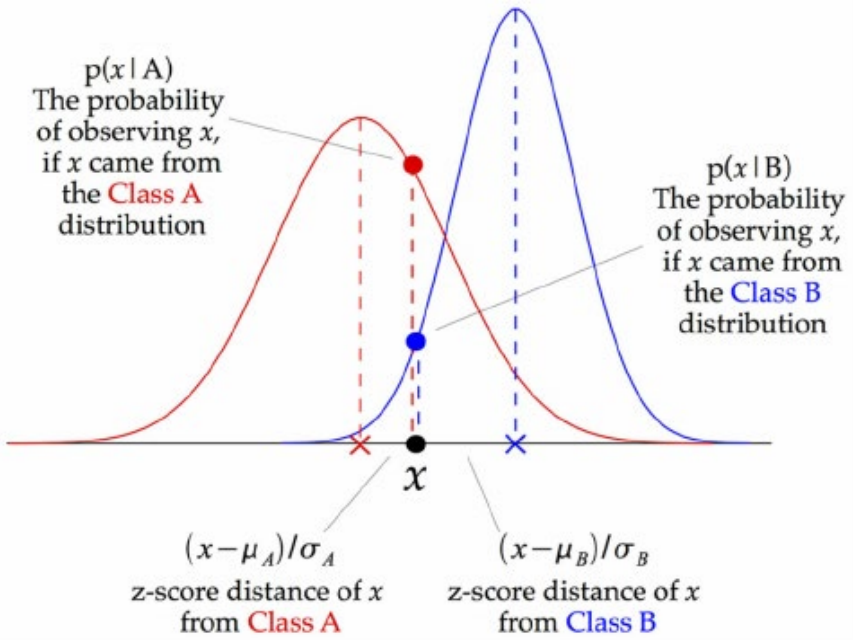
```
決策樹 BEFORE Standardization

Accuracy on trianing set:  84.43%
Accuracy on testing set: 82.42%
Scores of 5-fold cross validation:  [0.81967213 0.81967213 0.70491803 0.7        0.8        ]
Average score from 5-fold cross validation: 76.89%
Confusion Matrix:
 [[33 11]
 [ 5 42]]
```




```
決策樹 AFTER Standardization

Accuracy on trianing set: 84.43%
Accuracy on testing set: 82.42%
Scores of 5-fold cross validation:  [0.81967213 0.81967213 0.70491803 0.7
Average accuracy from 5-fold cross validation: 76.89%
Confusion Matrix:
 [[33 11]
 [ 5 42]]
```

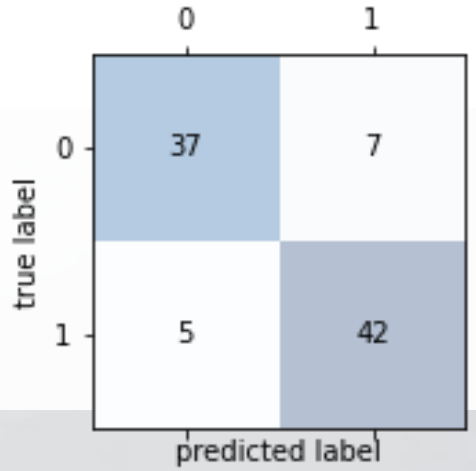


高斯貝氏




```
貝氏 BEFORE Standardization

Accuracy on trianing set: 82.55%
Accuracy on testing set: 86.81%
Scores of 5-fold cross validation:  [0.80327869 0.8852459  0.80327869 0.76666667 0.73333333]
Average accuracy from 5-fold cross validation: 79.84%
Confusion Matrix:
 [[37  7]
 [ 5 42]]
```

```
貝氏 AFTER Standardization

Accuracy on trianing set: 82.55%
Accuracy on testing set: 86.81%
Scores of 5-fold cross validation:  [0.80327869 0.8852459  0.80327869 0.76666667 0.73333333]
Average accuracy from 5-fold cross validation: 79.84%
Confusion Matrix:
 [[37  7]
 [ 5 42]]
```

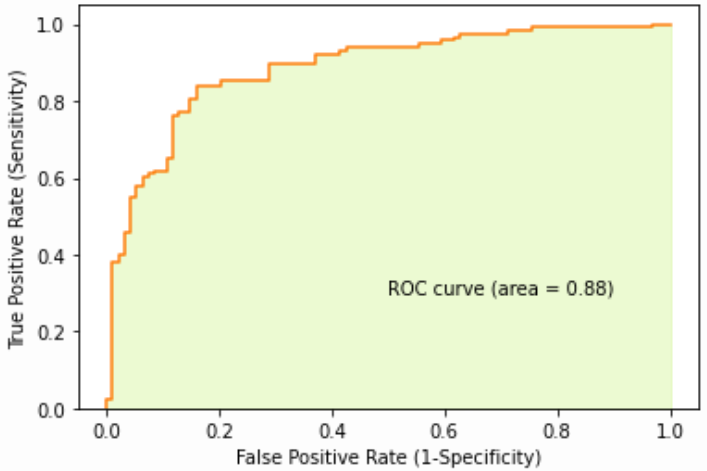


ROC Curve & AUC

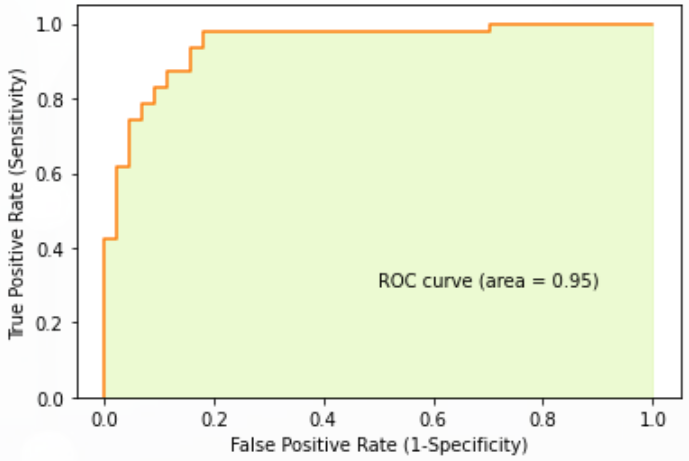
```
transformed = pd.DataFrame(StandardScaler().fit_transform(X_train))

pipeline = make_pipeline(StandardScaler(), GaussianNB())
pipeline.fit(X_train, y_train)
```

```
transformed = pd.DataFrame(StandardScaler().fit_transform(X_test))

pipeline = make_pipeline(StandardScaler(), GaussianNB())
pipeline.fit(X_test, y_test)
```

Receiver Operating Characteristic Plot for training



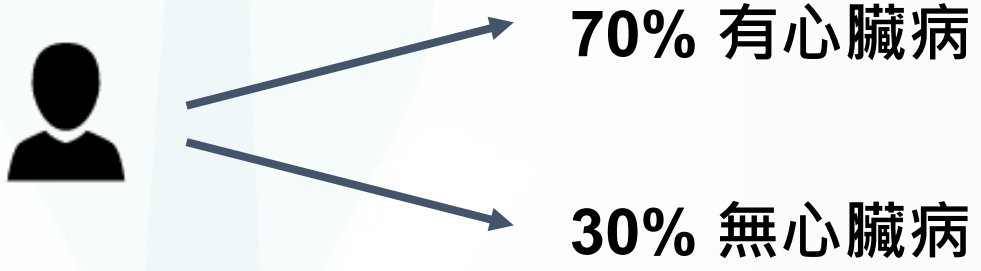
Receiver Operating Characteristic Plot for testing

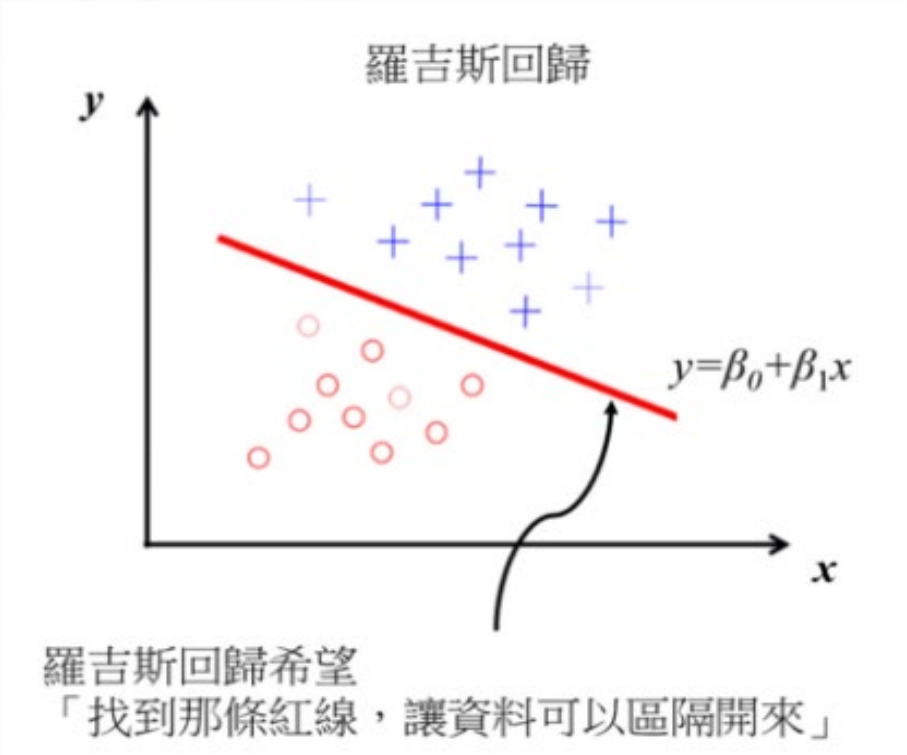


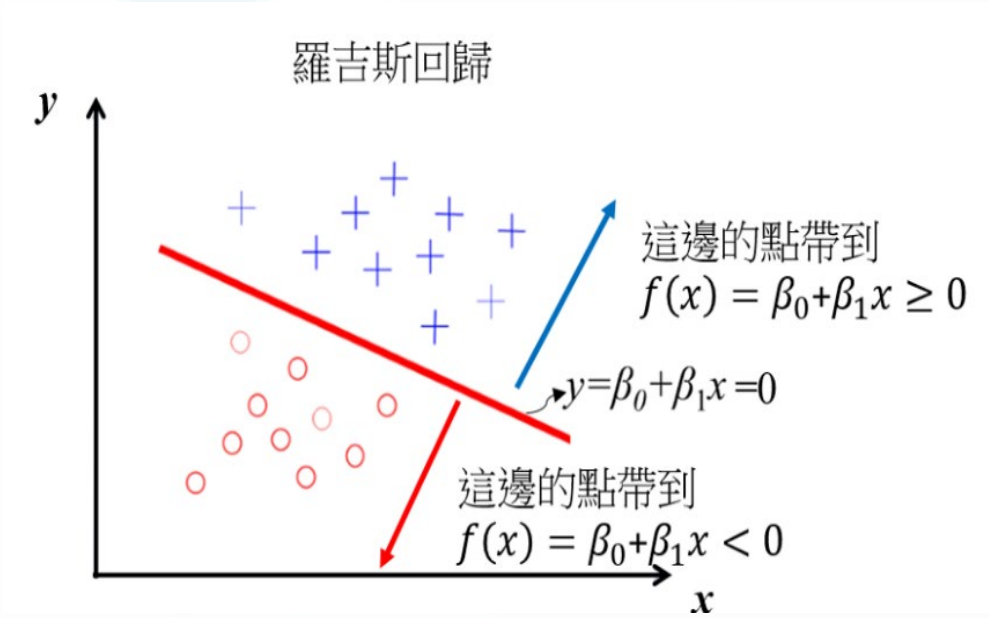
Logistic Regression
羅吉斯迴歸

Logistic Regression

- 使用迴歸的概念來分類
- 可以計算分類的機率







Logistic Model

```
from  sklearn.linear_model    import  LogisticRegression
lr=LogisticRegression(    C=50.0,
                          class_weight='balanced',
                          max_iter=100,
                          solver='liblinear',
                          tol=0.01   )
```

C：C參數與懲罰系數成反比，C值越小，則正規化效果越強

class_weight 權重參數 ：調節樣本比例

Solver 優化算法： liblinear ,sag , lbfgs , newton-cg

max_iter：迭代次數

tol：殘差收斂條件，即迭代的連續兩次之間殘差小于tol就停止

參數值	優化方式
liblinear	坐標軸下降來迭代，優化損失函數
lbfgs	利用損失函數二階導數矩陣來迭代，優化損失函數
newton-cg	lbfgs的另一種方法，也是利用損失函數二階導數矩陣來迭代
sag	隨機平均梯度下降，每次迭代僅用一部份樣本計算梯度 適合大樣本

超參數調整

超參數 C	結果	決策
c = 1	0.769	
c = 50	**0.791**	✔☐
c = 100	**0.791**	✔☐

超參數 max_ iter	結果	決策
100	**0.791**	✔☐
500	0.791	
1000	0.791	
50	0.791	

超參數 solver	結果	決策
liblinear	**0.791**	✔☐
sag	0.782	
lbfgs	**0.791**	✔☐
newton-cg	0.773	

超參數 tol	結果	決策
0.0001	**0.791**	✔☐
0.001	0.791	
0.01	0.780	
0.1	0.769	

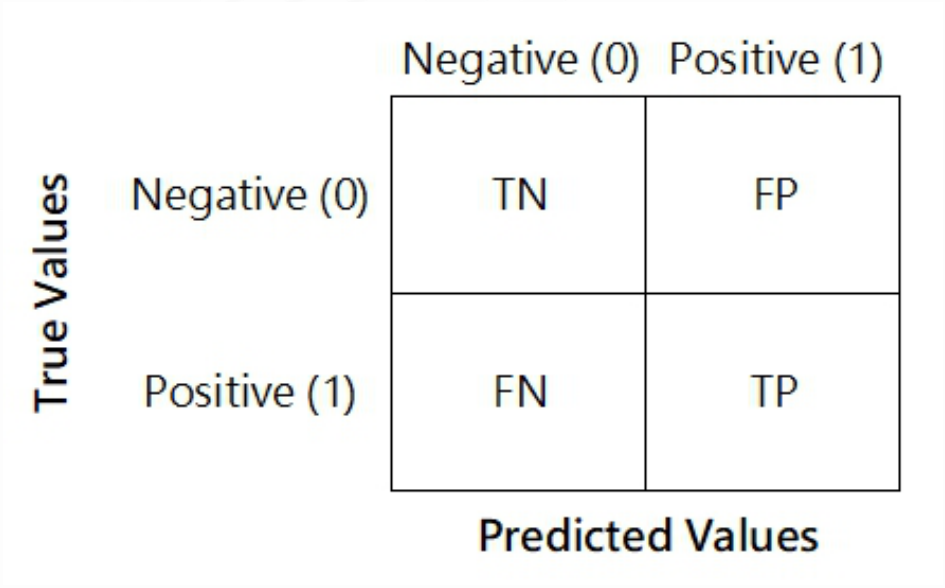
超參數調整

```
[204] from sklearn.model_selection import GridSearchCV
      import warnings
      warnings.filterwarnings("ignore")

      #  参数设置
      params = {'C':[    1,   50,   100],
                        'max_iter':[50,   100,500,   1000],
                        'class_weight':['balanced',   None],
                        'tol':[0.0001,0.001,0.01,0.1],
                        'solver':['liblinear','sag','lbfgs','newton-cg']
                       }
      lr = LogisticRegression()
      clf = GridSearchCV(lr,  param_grid=params,  cv=10)
      clf.fit(x_train_nor,y_train)
```

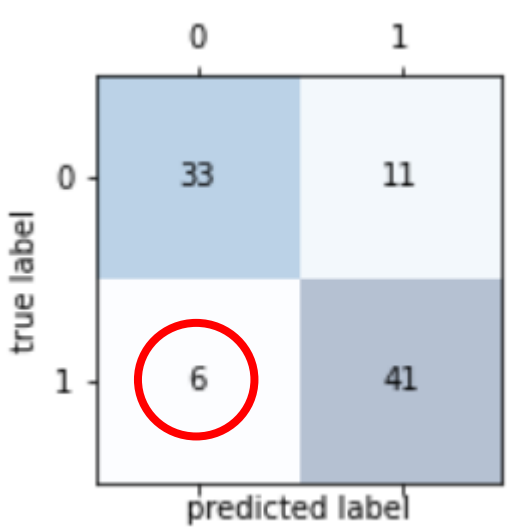
```
train_accuracy: 0.8113207547169812

test_accuracy: 0.7912087912087912

[[-0.31948254 -0.42793986  0.85542465  0.164883    0.10222755 -0.87524607]]
[0.24768291]
```

超參數	數值	結果	決策
c	1	**0.791**	✔☐
max_iter	100		
tol	0.0001		
solver	lbfgs		

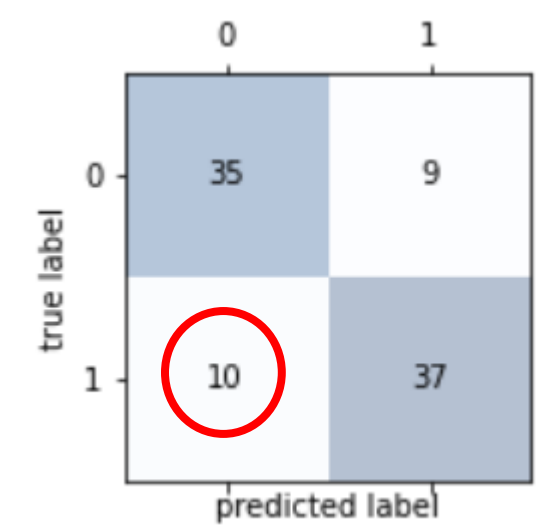
Confusion matrix 混淆矩陣



GridSearchCV



手動調整

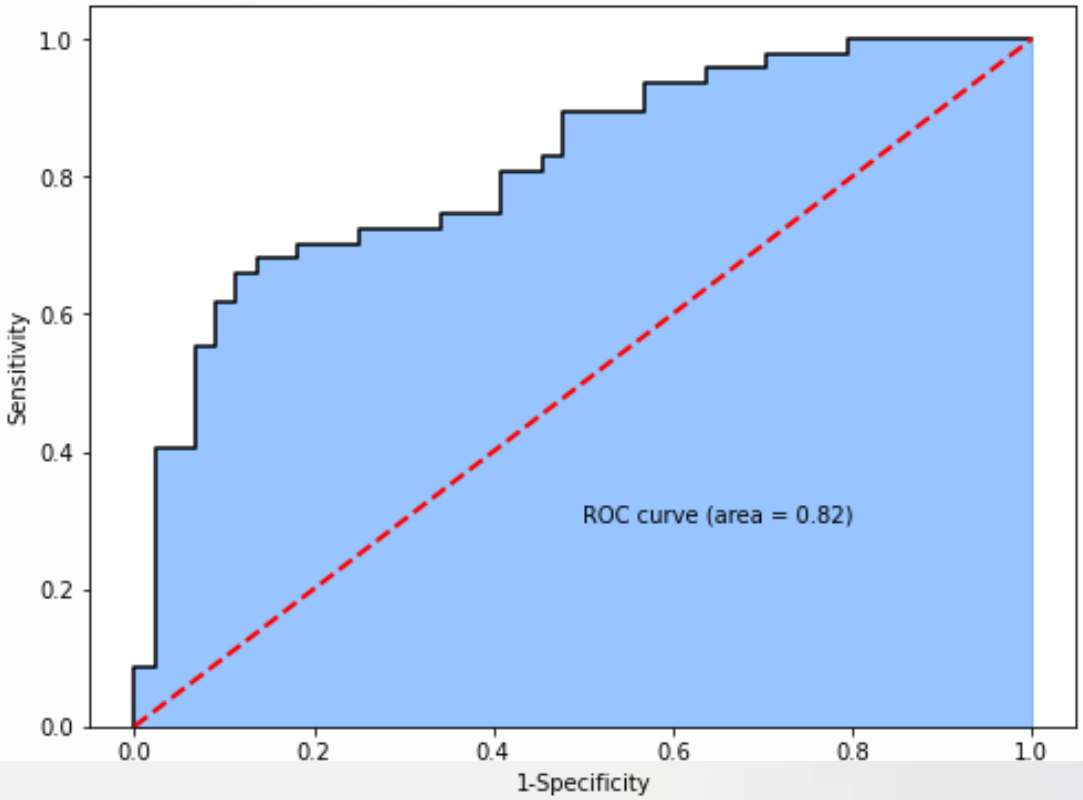


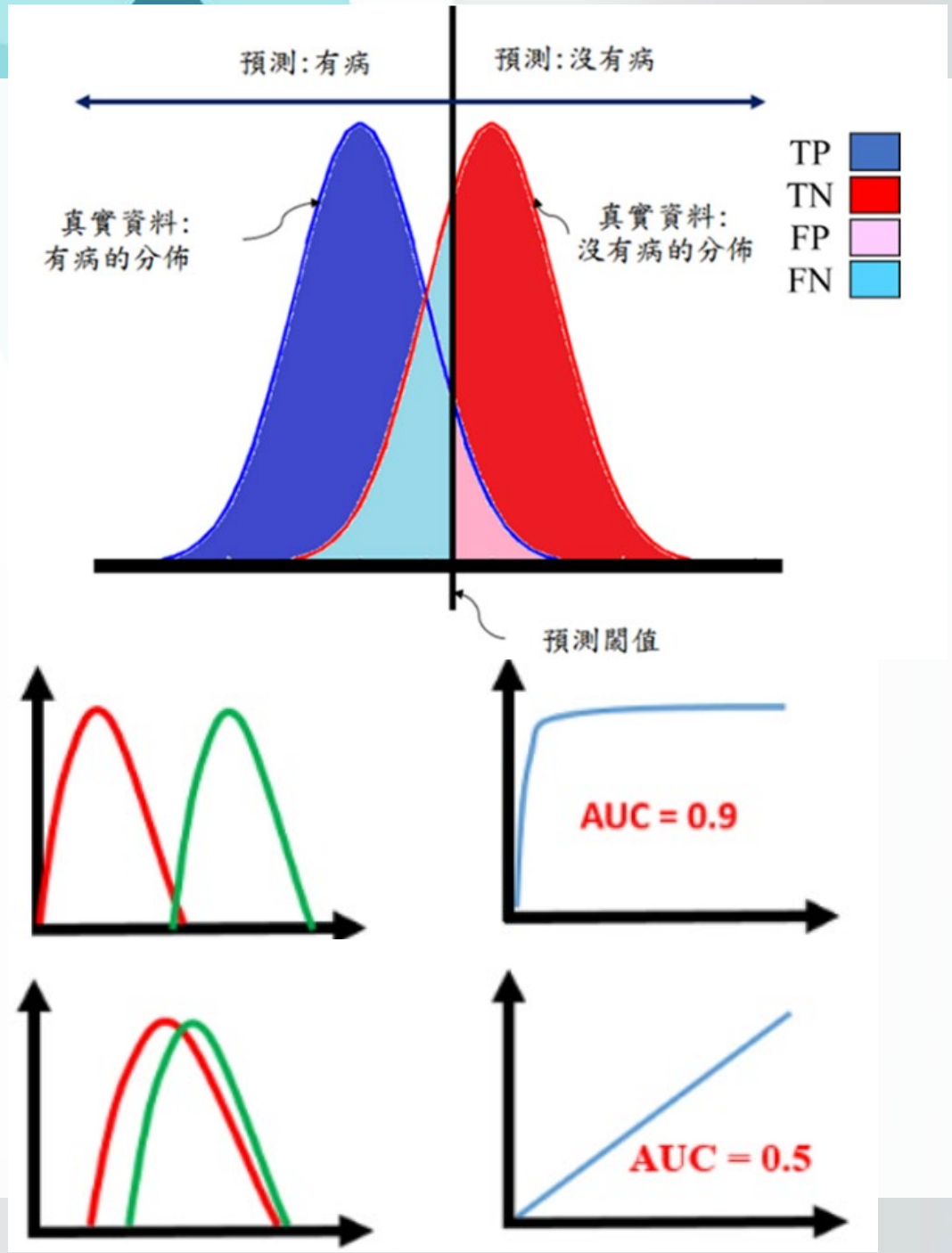
把有心臟病的分類為沒心臟病

Confusion matrix 混淆矩陣

$$\text{Recall} = \text{or Sensitivity} = \frac{TP}{TP + FN} \quad \text{(真陽率)}$$

$$\text{Specificity} = \frac{TN}{TN + FP} \quad \text{(假陽率)}$$





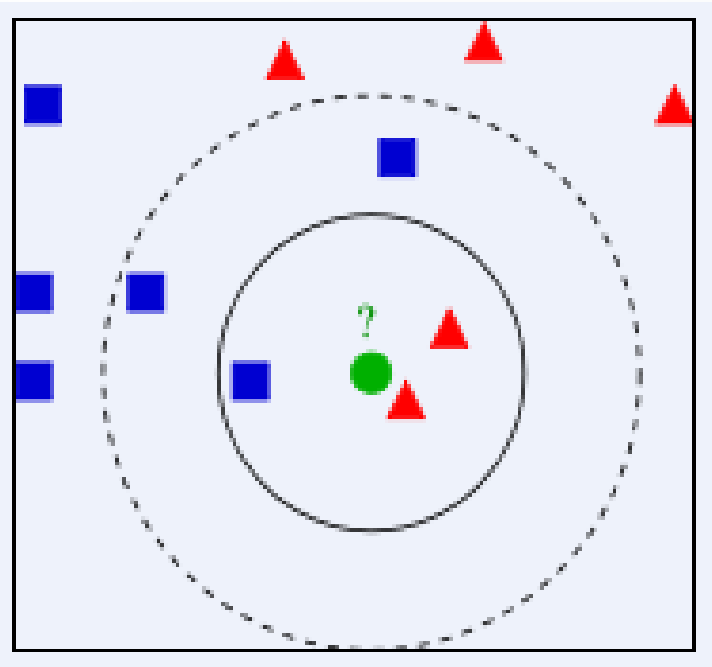
KNN (k nearest neighbor)
近鄰演算法

模型建立

KNN演算法原理

計算測試樣本與每個訓練樣本的距離，選擇這k個最相似資料中出現最多的分類作為新資料的分類類別

```
#建立模型
from  sklearn.neighbors  import  KNeighborsClassifier
def  knnprediction(k,  x_train,  y_train,  x_test):
        knn  =  KNeighborsClassifier(n_neighbors  =  k)
        knn.fit(x_train,  y_train)
        pred  =  knn.predict(x_test)
        return  pred

knn_pred  =  knnprediction(12,  x_train,  y_train,  x_test)
```



- k為KNN的鄰居數
- 找到最佳的k值
- 建立模型並預測

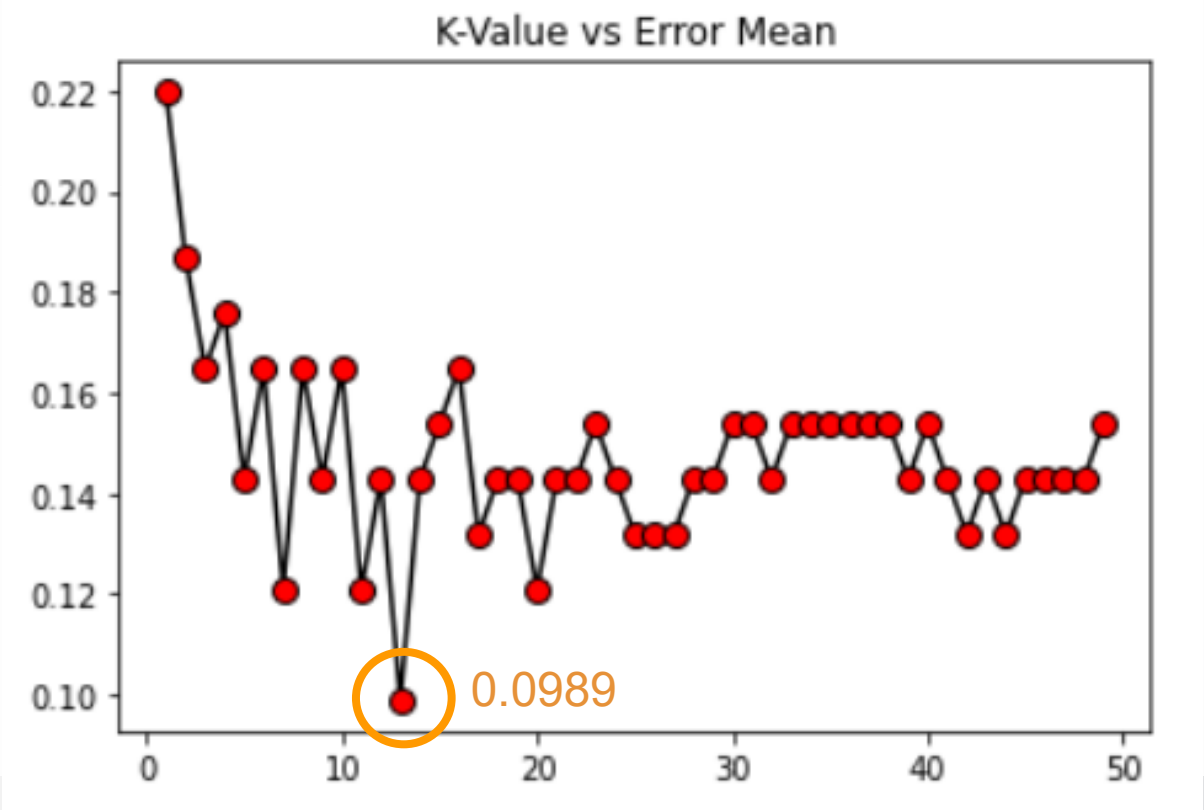
參數調整

```
#利用for迴圈尋找k值，使error_meam為最小
error_mean  =  []
for  k  in  range(1,  50):
        pred_i  =  knnprediction(k,  x_train,  y_train,  x_test)
        error_mean.append(np.mean(pred_i  !=  y_test))

for  i  in  range(0,49):
    if  error_mean[i]  ==  min(error_mean):
        print(i)

print(error_mean[12])
```

```
12
0.0989010989010989
```

參數調整

```
plt.plot(range(1,50),  error_mean,  color='black',  marker='o',  markerfacecolor='r',  markersize=8)
plt.title('K-Value  vs  Error  Mean')
plt.show()
```



模型準確度

```
from  sklearn.metrics  import  classification_report
def  evaluate(prediction,y_test):
        result  =  classification_report(y_test,prediction,output_dict=True)
        accuracy  =  result['accuracy']
        performance_data=  {'accuracy':(accuracy)}
        return  performance_data

knn_pr  =  evaluate(knn_pred,y_test)
print(knn_pr)
```

```
{'accuracy': 0.8571428571428571}
```

04

分析與模型效度驗證

Model 評估模型效度

Model	train_accuracy	test_accuracy
Decision Tree	0.826	0.868
Naive Bayes	0.844	0.824
Logistic Regression	0.811	0.791
KNN	0.844	0.857

Decision Tree Generaliztion

```
決策樹 AFTER Standardization

Accuracy on trianing set: 84.43%
Accuracy on testing set: 82.42%
Scores of 5-fold cross validation:  [0.81967213 0.81967213 0.70491803 0.7        0.8        ]
Average accuracy from 5-fold cross validation: 76.89%
Confusion Matrix:
 [[33 11]
 [ 5 42]]
```

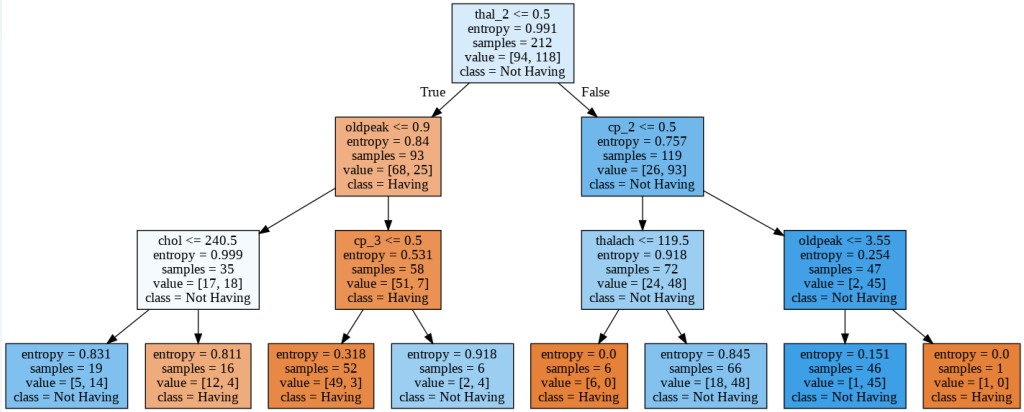
05

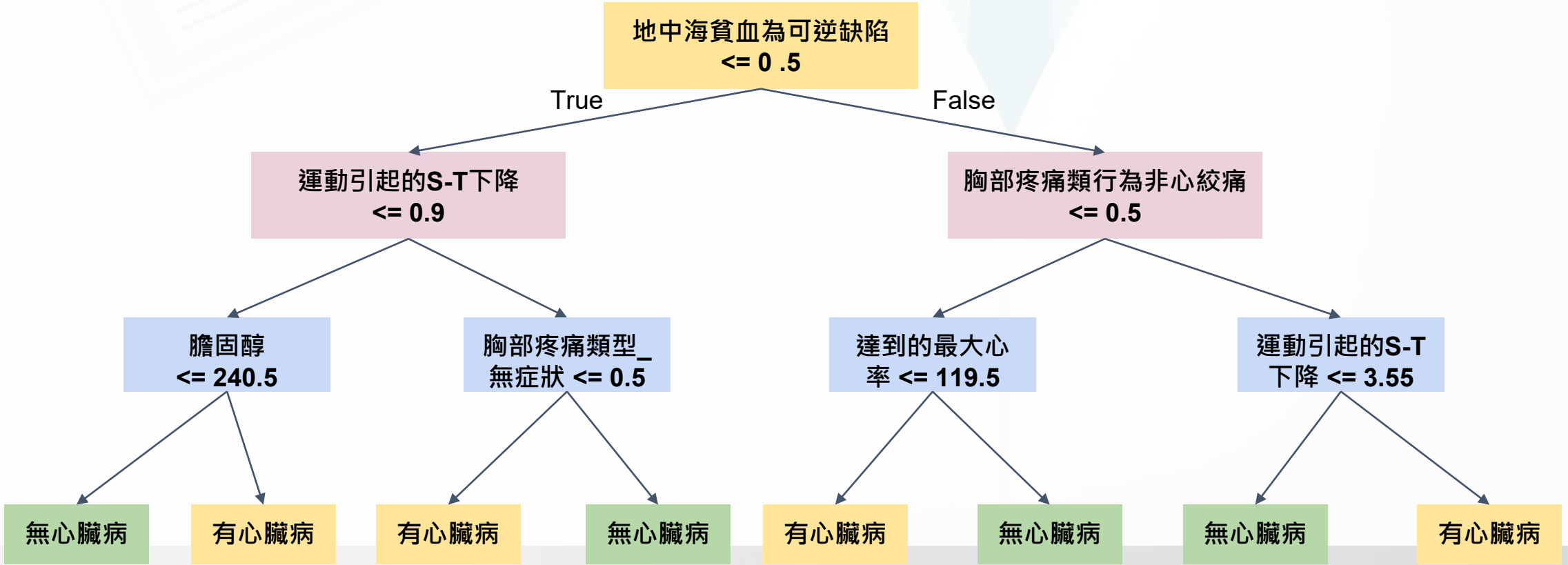
結論與未來展望

結論

4種模型都具有良好的預測能力，其中推薦使用

Decision Tree ，其準確率高、直觀好理解





未來展望

- 輔助醫師判斷預測是否有罹患心臟病，讓病患可以提前做預防
- 運用在更多的疾病預測分類
- 可在院方的網站或APP設置心臟病的查詢功能，讓民眾能自行輸入個人的資料檢測

Thank you for your listening!