碩管一甲 M09218001 周彥廷
數據分析與應用-作業 2：caret 分類器實作 (PYTHON 版)

請見網路大學中的"FattyLiver.csv"檔案，表中每列代表一位受檢者，"是否有脂肪肝"欄位代表該受檢者是否有患有脂肪肝，其他欄位則為體檢項目，試對該分類問題進行建模，建模過程至少須包含幾個元素：

(1)敘述欄位

(2)敘述建模流程圖

(3)將數據切成訓練與測試樣本，比例為 7:3、進行 5-fold CV

(4)盡可能地找出你的最佳分類器，並寫下該分類器的數學原理

(5)紀錄實驗結果，包含最佳超參數、測試樣本混淆矩陣、測試樣本預測正確率、測試樣本 precision、recall、重要變數列表等

```
# 載入所需模組
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy import mean
from numpy import std
import sklearn
import scipy
from sklearn.utils import shuffle
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV,
RandomizedSearchCV
from sklearn.model_selection import validation_curve
from sklearn.metrics import confusion_matrix, classification_report
from matplotlib import pyplot
from sklearn.datasets import make_blobs
from sklearn.datasets import make_classification
from sklearn import metrics
```

# import model
from sklearn.svm import SVC
from sklearn import svm
from sklearn.linear_model import LogisticRegression

I.資料前處理

# 1.讀取資料與去清洗資料為 Tidy data
data = pd.read_csv('C:/Users/MCUT/Desktop/FattyLiver.csv', header=0, sep=',',
encoding='Big5')
df = data.dropna()
# 查看資料
print(df.head(10))
print("資料數量", df.shape)
print("資料欄位", df.columns)
print(type(df))

df - DataFrame

Index	是否有脂肪肝	年齡	性別	BMI	收縮壓	舒張壓	脈搏	抽菸喝酒檳榔	腰圍	白血球	紅血球	血色素
0	NO	2	0	23.07	98.35	67.1	88.06	0	73.26	6.04	4.57	8.88
1	NO	2	0	20.72	103.23	72.77	94.56	0	67.94	4.69	5.53	12.06
2	NO	2	0	20.24	115.18	59.49	71.69	0	69.84	6.56	4.46	13.21
3	NO	3	0	21.25	98.2	62.03	68.74	0	72.33	3.82	4.16	12.64
4	YES	3	1	25.16	109.37	64.61	89.36	2	85.59	8.4	4.91	15.2
5	NO	2	0	20.48	93.38	61.89	74.11	0	62.34	6.44	4.36	13.34
6	NO	2	0	22.06	105.58	65.43	96.94	0	78.74	6.74	4.42	12.59
7	NO	2	1	21.9	104.62	69.32	66.02	1	77.46	6.57	5.62	17
8	NO	2	0	19.46	100.21	64.19	77.18	0	63.35	7.25	4.7	12.44
9	YES	2	0	29.73	102.03	60.18	61.6	0	84.39	8.53	4.59	13.36

df	DataFrame	(1877, 42)

```
資料欄位 Index(['是否有脂肪肝', '年齡', '性別', 'BMI', '收縮壓', '舒張壓', '脈搏', '抽菸喝酒檳榔', '腰圍', '白血球',
       '紅血球', '血色素', '血中紅血球百分比', '紅血球平均容積', '紅血球色素', '紅血球色素濃度', '紅血球分佈變異數',
       '血小板', '嗜中性球', '淋巴球', '單核球', '嗜伊紅性球', '嗜鹼性球', '膽固醇總量', '尿素氮', '三酸甘油脂',
       '飯前血糖', '肌酸酐', '丙胺酸丙酮酸轉胺酵素', '麩胺轉酸脢', '促甲狀腺激素', '麩胺酸草酸轉胺酵素', '鹼性磷酸酵素',
       '白蛋白', '總膽紅素', '直接膽紅素', '尿酸', '總蛋白', '高密度脂蛋白膽固醇', '乳酸脫氫脢', '尿酸鹼度',
       '尿液外物'],
      dtype='object')

In [7]: print(type(df))
<class 'pandas.core.frame.DataFrame'>
```

# DATA preprocess 使用 MinMaxScaler() 作 Feature Transforms
MMSS = MinMaxScaler().fit(df.iloc[:,1:])
df.iloc[:,1:] = MMSS.transform(df.iloc[:,1:])
df = shuffle(df)

df - DataFrame

Index	是否有脂肪肝	年齡	性別	BMI	收縮壓	舒張壓	脈搏	抽菸喝酒檳榔	腰圍	白血球
1504	NO	0.25	1	0.269461	0.345195	0.332205	0.288031	0	0.305882	0.283232
789	YES	0.25	1	0.410512	0.520231	0.456987	0.246252	0	0.553581	0.494353
175	NO	0.25	1	0.347638	0.328938	0.239956	0.316458	0.333333	0.332353	0.402259
769	YES	0.25	1	0.369261	0.652908	0.575	0.364955	0	0.43798	0.518679
1613	NO	0.25	1	0.158017	0.417811	0.435699	0.536905	0	0.203708	0.274544
1514	NO	0	1	0.340652	0.322977	0.165721	0.233555	0	0.35422	0.295395
287	NO	0.5	1	0.23686	0.180816	0.240611	0.396576	0	0.323146	0.414422
1770	YES	0.5	1	0.444112	0.458092	0.253166	0.312935	0	0.411637	0.419635
1569	YES	0	1	0.257152	0.283869	0.236463	0.262472	0.333333	0.181586	0.242398

```
#2.設置特徵值和目標值
X = df.iloc[:, 1:]   #特徵值 其他指標
Y = df.iloc[:, 0]   #目標值 是否為脂肪肝
# Dummy variable
labelencoder = LabelEncoder()
y=pd.DataFrame(Y)
y['是否有脂肪肝'] = labelencoder.fit_transform(y['是否有脂肪肝'])
print(y.head(10)) #使用 LabelEncoder 將 YES/NO 轉換成 1 與 0

# 3.splitting the dataset into the Training set and Test set
#將數據切成訓練與測試樣本，比例為 7:3
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,
random_state=1)
print("資料數量", X_train.shape)
print("資料數量", X_test.shape)
print("資料數量", y_train.shape)
print("資料數量", y_test.shape)
```

X_train	DataFrame	(1313, 41)	Column names: 酒檳榔, 腰圍,
X_test	DataFrame	(564, 41)	Column names: 酒檳榔, 腰圍,

y_train	Series	(1313,)	Series object
y_test	Series	(564,)	Series object

```
# 4.Data 5 K-fold cross validation
kfold= KFold(n_splits=5, random_state=1, shuffle=True)
```

II.建模與調整最佳超參數

#5.GridSearchCV 尋找 SVM 最佳參數和最佳效能

```
parameters = {'gamma': [0.001, 0.01, 0.1, 1, 10,100], 'C':[0.001, 0.01, 0.1, 1,10,100]}
gs = GridSearchCV(svm.SVC(), parameters, refit = True, cv = kfold, verbose = 1,
n_jobs = -1)
gs.fit(X_train,y_train) #Run fit with all sets of parameters.
print('最佳參數: ',gs.best_params_)
print('最佳效能: ', gs.best_score_)
```

#6.得到最佳參數並用於建立 svm 模型使用

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.4s
最佳參數:  {'C': 100, 'gamma': 0.01}
最佳效能:  0.8179490900647257
```

#7.建模並套入找出的最佳參數

```
svm_model = svm.SVC(kernel='linear', C=100, gamma=0.01).fit(X_train, y_train)
svm_scores = cross_val_score(svm_model, X_train, y_train, cv=kfold)
print(svm_scores)
print("SVM_model Accuracy: %0.2f (+/- %0.2f)" % (svm_scores.mean(),
svm_scores.std() * 2))
```

```
[0.84030418 0.82889734 0.78326996 0.75572519 0.82061069]
SVM_model Accuracy: 0.81 (+/- 0.06)
```

III.紀錄實驗結果(混淆矩陣與重要變數列表)

```
pred_svm = svm_model.predict(X_test)
```

#8.建立混淆矩陣

```
confusion_matrix(y_test, pred_svm)
```

```
array([[232,  43],
       [ 64, 225]], dtype=int64)
```

#列出 tn, fp, fn, tp 四個指標

```
tn, fp, fn, tp = confusion_matrix(y_test, pred_svm).ravel()
print(tn,fp, fn, tp)
```

232, 43, 64, 225

#9.顯示測試樣本預測正確率、測試樣本 precision、recall

```
print(classification_report(y_test, pred_svm))
```

```
In [21]: print(classification_report(y_test, pred_svm))
              precision    recall  f1-score   support

          NO       0.78      0.84      0.81       275
         YES       0.84      0.78      0.81       289

    accuracy                           0.81       564
   macro avg       0.81      0.81      0.81       564
weighted avg       0.81      0.81      0.81       564
```

#10.重要變數列表

```
def f_importances(coef, names):
    imp = coef
    imp,names = zip(*sorted(zip(imp,names)))
    plt.barh(range(len(names)), imp, align='center')
    plt.yticks(range(len(names)), names)
    plt.show()
```

#列出重要變數欄位名稱

```
df_columns = df.columns
df_columns = df_columns[1:]
print(df_columns)
```

#特徵欄位

```
features_names = df_columns
```

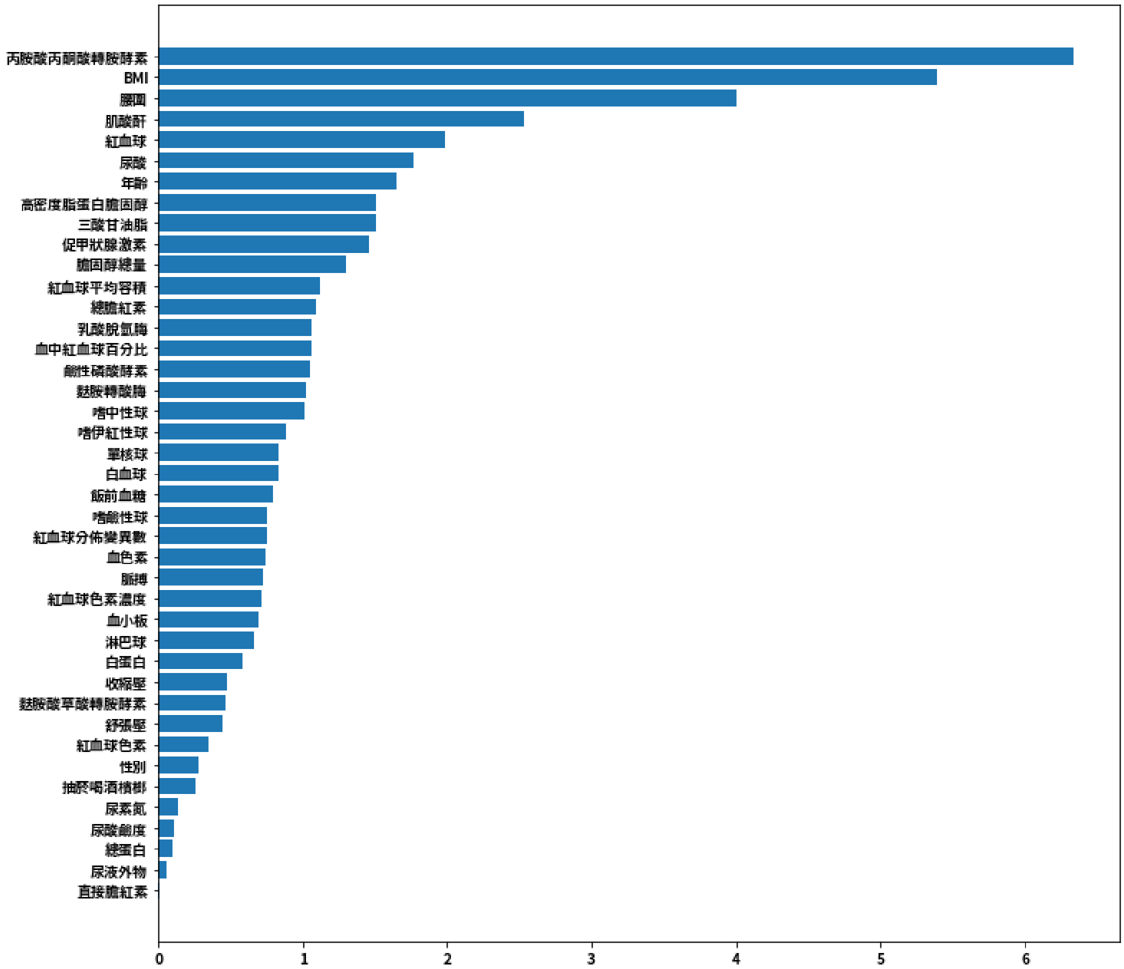
```
Index(['年齡', '性別', 'BMI', '收縮壓', '舒張壓', '脈搏', '抽菸喝酒檳榔', '腰圍', '白血球', '紅血球',
       '血色素', '血中紅血球百分比', '紅血球平均容積', '紅血球色素', '紅血球色素濃度', '紅血球分佈變異數', '血小板',
       '嗜中性球', '淋巴球', '單核球', '嗜伊紅性球', '嗜鹼性球', '膽固醇總量', '尿素氮', '三酸甘油脂', '飯前血糖',
       '肌酸酐', '丙胺酸丙酮酸轉胺酵素', '麩胺轉酸脢', '促甲狀腺激素', '麩胺酸草酸轉胺酵素', '鹼性磷酸酵素', '白蛋白',
       '總膽紅素', '直接膽紅素', '尿酸', '總蛋白', '高密度脂蛋白膽固醇', '乳酸脫氫脢', '尿酸鹼度', '尿液外物'],
      dtype='object')
```

#下載安裝並使用字型_台北黑體

```
plt.rcParams['font.sans-serif'] = ['Taipei Sans TC Beta']
plt.figure(figsize=(12,12))
```

#繪製重要變數列表

f_importances(abs(svm_model.coef_[0]), features_names)



#結語：初步利用 GridSearchCV 方式對 SVM model 進行參數調整，得到預測樣本正確率 81%，後續將嘗試使用其他演算法與調參提升預測樣本正確率。

LogisticRegression
#設置超參數範圍

```
tuned_parameters=[{'penalty':['l1','l2'], 'C':[0.01,0.05,0.1,0.5,1,5,10,50,100],
                   'solver':['liblinear'], 'multi_class':['ovr']}, {'penalty':['l2'],
                  'C':[0.01,0.05,0.1,0.5,1,5,10,50,100],    'solver':['lbfgs'],
                  'multi_class':['ovr','multinomial']}]
logistic=GridSearchCV(LogisticRegression(tol=1e-6),tuned_parameters,cv=kfold)
logistic.fit(X_train,y_train)
print('Best parameters set found:', logistic.best_params_)
```

#得到最佳參數設定如下

```
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(
Best parameters set found: {'C': 5, 'multi_class': 'ovr', 'penalty': 'l2', 'solver': 'liblinear'}
```

LR_model=LogisticRegression(C=5, penalty='l2', multi_class = 'ovr', tol=0.0001).fit(X_train,y_train)

LR_scores = cross_val_score(LR_model, X_train, y_train, cv=kfold)

print(LR_scores)

```
In [99]: print(LR_scores)
[0.84790875 0.82889734 0.78707224 0.75954198 0.81679389]
```

print("logistic_model Accuracy: %0.2f (+/- %0.2f)" % (LR_scores.mean(), LR_scores.std() * 2))

```
In [100]: print("logistic_model Accuracy: %0.2f (+/- %0.2f)"
logistic_model Accuracy: 0.81 (+/- 0.06)
```

#評估模型

pred_LR = LR_model.predict(X_test)

#建立混淆矩陣

confusion_matrix(y_test, pred_LR)

print(classification_report(y_test, pred_LR))

```
array([[232,  43],
       [ 61, 228]], dtype=int64)

In [104]: print(classification_report(y_test, pred_LR))
              precision    recall  f1-score   support

          NO       0.79      0.84      0.82       275
         YES       0.84      0.79      0.81       289

    accuracy                           0.82       564
   macro avg       0.82      0.82      0.82       564
weighted avg       0.82      0.82      0.82       564
```

#結語：使用 LogisticRegression 經調整參數後使測試樣本正確率提升至 0.82。

Logistic Regression(邏輯斯回歸)簡介與數學原理

Logistic Regression 是一個平滑的曲線，當 $w_0*x_0+w_1*x_1+\cdots+w_n*x_n$ 越大時判斷成 A 類的機率越大，越小時判斷成 A 類的機率越小。由於是二元分類，如果判斷成 A 類的機率越小，B 類的機率越大(判斷成 B 類的機率 ＝1 - 判斷成 A 的機率)，如圖 1 所示。

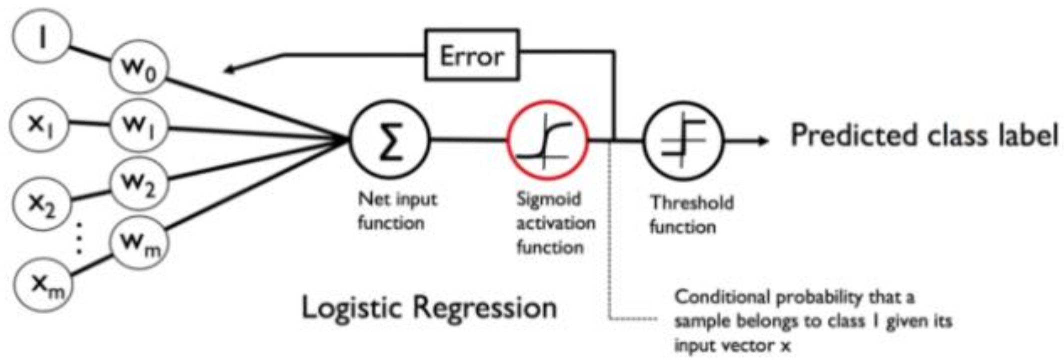


圖 1. LogisticRegression 運作原理示意圖

數學原理：

假設將二元分類的 A 類以+1 表示、B 類以-1 表示，現在將 A 類改以+1 表示、B 類以 0 表示。我們想要找到一組 w，能夠將下方的式子變成最大值，那組 w 就是我們要找的線(z = w*x)。下方的式子是希望當 y=1 的時候 Ø(z) 越靠近 1（判斷成 A 類的機率越大），由於 1-y 是 0 所以右邊的項會是 1，當 y=0 時左邊這項會是 1 右邊這項希望 Ø(z) 越靠近 0 越好（判斷成 B 類的機率越大），將 0 與 1 分為兩個區塊如圖 3 所示。

$$\prod_{i=1}^{n}\left(\phi\left(z^{(i)}\right)\right)^{y^{(i)}}\left(1-\phi\left(z^{(i)}\right)\right)^{1-y^{(i)}}$$

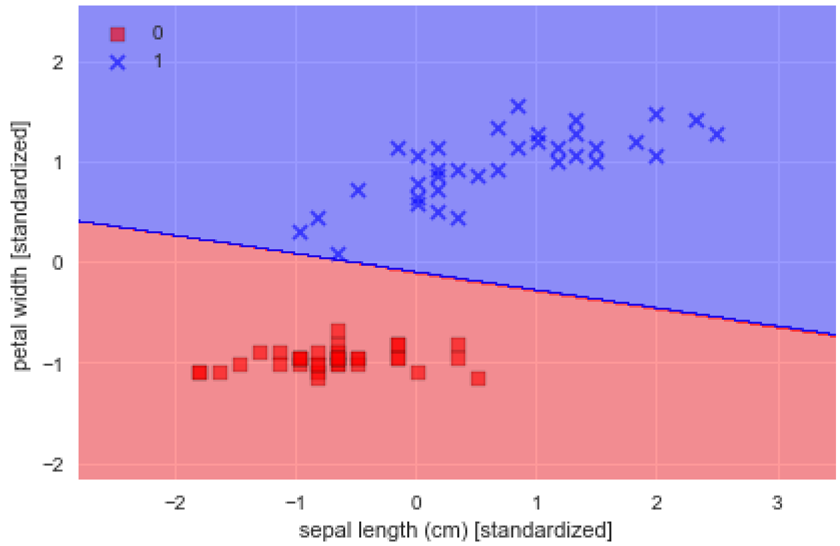


圖 3. 滿足數學式並分類 0 與 1 示意圖

參考資料來源：

[資料分析&機器學習] 第 3.3 講：

線性分類-邏輯斯回歸(Logistic Regression)介紹 https://yehjames.medium.com/

附錄：簡易調整 xgboost 參數並評估績效

```
import xgboost as xgb
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
model2 = xgb.XGBClassifier()

pipeline = Pipeline([
    ('standard_scaler', StandardScaler()),
    ('pca', PCA()),
    ('model', model2)])

param_grid = {
    'pca__n_components': [5, 10, 15, 20, 25, 30],
    'model__max_depth': [2, 3, 5, 7, 10],
    'model__n_estimators': [10, 100, 500],}
grid = GridSearchCV(pipeline, param_grid, cv=kfold, n_jobs=-1, scoring='roc_auc')
grid.fit(X_train, y_train)
print('最佳參數:', grid.best_params_)
```

```
XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic'
'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
最佳參數: {'model__max_depth': 2, 'model__n_estimators': 100, 'pca__n_components': 20}
```

```
xgb_model = xgb.XGBClassifier(model__max_depth=2,
model__n_estimators=100, pca__n_components=20).fit(X_train, y_train)
xgb_scores = cross_val_score(xgb_model, X_train, y_train, cv=kfold)
print(xgb_scores)
print("logistic_model Accuracy: %0.2f (+/- %0.2f)" % (LR_scores.mean(),
xgb_scores.std() * 2))
#評估模型並建立混淆矩陣
pred_xgb = xgb_model.predict(X_test)
confusion_matrix(y_test, pred_xgb)
print(classification_report(y_test, pred_xgb))
```

```
In [161]: print(classification_report(y_test, pred_xgb))
              precision    recall  f1-score   support

          NO       0.76      0.81      0.78       275
         YES       0.80      0.75      0.78       289

    accuracy                           0.78       564
   macro avg       0.78      0.78      0.78       564
weighted avg       0.78      0.78      0.78       564
```