建模訓練紀錄&實驗過程&程式碼:

- **Package 以及會用到的函式:**

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score,confusion_matrix
from scipy.stats import chi2_contingency
from scipy.stats import mannwhitneyu
import math
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import RFE, RFECV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier

from joblib import dump, load
```

RFECV 可將初步篩完的特徵做進一步挑選特徵

```python
def rfe_f_select(model, x_train, y_train):
    rfecv = RFECV(estimator = model, step = 1, cv = 5, scoring='accuracy').fit(x_train, y_train)
    print("Optimal number of features : %d" % rfecv.n_features_)
    print("Support is %s" % rfecv.support_)
    print("Ranking of features : %s" % rfecv.ranking_)

    return rfecv.n_features_, rfecv.support_
```

為了可以找出最佳化的特徵數量，因此 step = 1，一個一個迭代來找出最佳特徵數且 cv = 5

Grid search

```
def grid_search(model, param_grid_, x_train, y_train_):
    optimal_params = GridSearchCV(
            model,
            param_grid,
            cv = 5,
            scoring = 'accuracy',
            verbose = 1,
            n_jobs = -1,
        )

    optimal_params.fit(x_train, y_train_)
    print(optimal_params.best_params_)
```

ROC and AUC

```
def ROC_and_AUC(label, score, name):
    fpr, tpr , _ = roc_curve(label, score)
    roc_auc = auc(fpr, tpr)
    name = name + '_auc = ' + str(roc_auc.round(3))
    lw = 2
    plt.plot(fpr, tpr, lw = lw, label = name)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve')
    plt.legend(loc = "lower right")
```

Model evaluation result

```
def model_evaluation_result(X_train_Std, y_train, X_test_Std, y_test, model, roctrainname, roctestname):

    model.fit(X_train_Std, y_train)

    train_pred = model.predict(X_train_Std)
    score_train = model.decision_function(X_train_Std)
    train_acc = accuracy_score(y_train, train_pred)
    tn, fp, fn, tp = confusion_matrix(y_train, train_pred).ravel()
    train_specificity = tn / (tn+fp)
    train_sensitivity = tp / (tp+fn)
    train_PPV = tp / (tp+fp)
    train_NPV = tn / (fn+tn)
    train_MCC = ((tp*tn)-(fp*fn))/math.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
    ROC_and_AUC(y_train, score_train, roctrainname)

    test_pred = model.predict(X_test_Std)
    score_test = model.decision_function(X_test_Std)
    test_acc = accuracy_score(y_test, test_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, test_pred).ravel()
    test_specificity = tn / (tn+fp)
    test_sensitivity = tp / (tp+fn)
    test_PPV = tp / (tp+fp)
    test_NPV = tn / (fn+tn)
    test_MCC = ((tp*tn)-(fp*fn))/math.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
    ROC_and_AUC(y_test, score_test, roctestname)
    result = [{'Accuracy': train_acc,'Specificity': train_specificity, 'Sensitivity': train_sensitivity, 'MCC': train_MCC, 'PPV': train_PPV, 'NPV': train_NPV},
              {'Accuracy': test_acc,'Specificity': test_specificity, 'Sensitivity': test_sensitivity, 'MCC': test_MCC, 'PPV': test_PPV, 'NPV': test_NPV}]
    result_df = pd.DataFrame(result)
    result_df.index = ['Training set', 'Test_set']
    result_df = result_df.round(3)
    print(result_df)
    return result_df, score_train, train_pred, score_test, test_pred, model
```

為了給使用機率去預測類別的(=助教給的 knn_model_evaluation_result):

```
def proba_model_evaluation_result(X_train_Std, y_train, X_test_Std, y_test, model, roctrainname, roctestname):

    model.fit(X_train_Std, y_train)

    train_pred = model.predict(X_train_Std)
    score_train = model.predict_proba(X_train_Std)
    train_acc = accuracy_score(y_train, train_pred)
    tn, fp, fn, tp = confusion_matrix(y_train, train_pred).ravel()
    train_specificity = tn / (tn+fp)
    train_sensitivity = tp / (tp+fn)
    train_PPV = tp / (tp+fp)
    train_NPV = tn / (fn+tn)
    train_MCC = ((tp*tn)-(fp*fn))/math.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
    ROC_and_AUC(y_train, score_train[:, 1], roctrainname)

    test_pred = model.predict(X_test_Std)
    score_test = model.predict_proba(X_test_Std)
    test_acc = accuracy_score(y_test, test_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, test_pred).ravel()
    test_specificity = tn / (tn+fp)
    test_sensitivity = tp / (tp+fn)
    test_PPV = tp / (tp+fp)
    test_NPV = tn / (fn+tn)
    test_MCC = ((tp*tn)-(fp*fn))/math.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
    ROC_and_AUC(y_test, score_test[:, 1], roctestname)
    result = [{'Accuracy': train_acc,'Specificity': train_specificity, 'Sensitivity': train_sensitivity, 'MCC': train_MCC, 'PPV': train_PPV, 'NPV': train_NPV},
              {'Accuracy': test_acc,'Specificity': test_specificity, 'Sensitivity': test_sensitivity, 'MCC': test_MCC, 'PPV': test_PPV, 'NPV': test_NPV}]
    result_df = pd.DataFrame(result)
    result_df.index = ['Training set', 'Test_set']
    result_df = result_df.round(3)
    print(result_df)
    return result_df, score_train, train_pred, score_test, test_pred, model
```

confusion matrix

```
[27] def confusion_matrix_scorer(clf, X, y):
         y_pred = clf.predict(X)
         cm = confusion_matrix(y, y_pred)
         return {'tn': cm[0, 0], 'fp': cm[0, 1],'fn': cm[1, 0], 'tp': cm[1, 1]}
```

```
def print_cm(model, x_train, y_train):
    cv_results = cross_validate(model, x_train, y_train, cv = 3, scoring = confusion_matrix_scorer)
    tn = round(cv_results['test_tn'].mean())
    fp = round(cv_results['test_fp'].mean())
    fn = round(cv_results['test_fn'].mean())
    tp = round(cv_results['test_tp'].mean())
    accuracy = (tp+tn)/(tn+fp+fn+tp)
    precision = tp/(tp+fp)
    sensitivity = tp/(tp+fn)
    specificity = tn/(tn+fp)
    MCC = ((tp*tn)-(fp*fn))/math.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
    print(cv_results['test_tn'], round(cv_results['test_tn'].mean()))
    print(cv_results['test_fp'], round(cv_results['test_fp'].mean()))
    print(cv_results['test_fn'], round(cv_results['test_fn'].mean()))
    print(cv_results['test_tp'], round(cv_results['test_tp'].mean()))
    print('accuracy:', accuracy)
    print('precision:', precision)
    print('sensitivity:', sensitivity)
    print('specificity:', specificity)
    print('MCC:', MCC)
```

- **Data split**

  我是採 train : test = 7:3

```
train_df, test_df = train_test_split(df, train_size = 0.7, random_state = 0)
```

- **特徵預選:**

先用統計方法計算 p-value 以找出和 label 有顯著差異的特徵

前 40 個使用卡方統計

```
#chi-square
cat_df = train_df.iloc[:,0:39]
cat_df = pd.concat([cat_df, train_df['label']], axis = 1)

cat_p_value = []
cat_df = cat_df[cat_df['label'] >= 0]
titles = list(cat_df.columns)

for i in titles:
        table = pd.crosstab(cat_df['label'],cat_df[i])
        chi2, p, dof, expected = chi2_contingency(table)
        cat_p_value.append(p)
d1_d39_p_value = pd.DataFrame([titles, cat_p_value]).T
d1_d39_p_value = d1_d39_p_value.rename(columns = {0:'features', 1:'p_value'})
```

後 50 個使用 Mann–Whitney U test

```
#Mann-Whitney U test
cont_df = train_df.iloc[:,39:90]
cont_df = pd.concat([cont_df, train_df['label']], axis = 1)
cont_df = cont_df[cont_df['label'] >= 0]
titles = list(cont_df.columns)

c_p_value = []
for i in titles:
        value1=[]
        value0=[]
        my_col = cont_df[[i,'label']]
        for j in range(0, my_col.shape[0]):
                if (str(my_col.iloc[j,1]) == '1'):
                        value1.append(my_col.iloc[j,0])
                elif (str(my_col.iloc[j,1]) == '-1'):
                        continue
                else:
                        value0.append(my_col.iloc[j,0])
        result = mannwhitneyu(value0, value1, alternative= 'two-sided')
        c_p_value.append(result[1])
c40_c90_p_value = pd.DataFrame([titles, c_p_value]).T
c40_c90_p_value = c40_c90_p_value.rename(columns = {0:'features', 1:'p_value'})
#print(c40_c90_p_value)
#print(d1_d39_p_value)
```

之後合併把 p-value 小於等於 0.05 的挑出來表示這些特徵和 label 有顯著差異

```
train_features = []
for i in d1_d39_p_value.iloc:
    if(i['p_value'] <= 0.05 and i['features'] != 'label'): train_features.append(i['features'])
for i in c40_c90_p_value.iloc:
    if(i['p_value'] <= 0.05 and i['features'] != 'label'): train_features.append(i['features'])

train_predata = train_df.loc[:,train_features]
test_predata = test_df.loc[:, train_features]
print(train_predata)
```

其選出來的特徵有 58 個

```
        d1     d2    d3  d4   d5   d6    d7  d8   d9  d11  ...   c73     c74  \
3318  1.70  1.800  1250   1  1.0  0.5  40.0   1  2.5    0  ...  12.6  2.9185
2701  1.60  1.680  1000   2  1.0  1.0  40.0   1  2.5  500  ...  10.6  1.1365
5862  1.60  1.700     0   2  2.0  1.0  50.0   1  2.5    0  ...  11.3  7.9100
3595  1.64  1.735   250   1  1.0  0.5  40.0   1  3.0  250  ...  10.6  7.8165
2377  1.60  1.680  1000   2  1.0  1.0  40.0   1  2.5  500  ...  11.4  1.1365
...    ...    ...   ...  ..  ...  ...   ...  ..  ...  ...  ...   ...     ...
4931  1.60  1.700     0   2  1.0  1.5  40.0   1  3.0    0  ...  10.3  5.1200
3264  1.70  1.800  1250   1  1.0  0.5  40.0   1  2.5    0  ...   9.2  2.9185
1653  1.63  1.720   750   2  1.0  1.0  50.0   1  2.5  500  ...  12.2  7.6560
2607  1.68  1.780  1000   2  0.0  0.5  40.0   1  2.5  500  ...  10.7  2.9185
2732  1.60  1.670  1000   2  1.0  1.0  40.0   1  2.5  500  ...  11.0  1.1365

       c75    c78   c80     c82    c83    c84    c89     c90
3318  6.00  143.0  33.0   621.6   51.0  260.0  280.0  1.1500
2701  6.22   70.0  40.0   532.2  223.0  257.0  280.0  1.5625
5862  6.90  132.0  24.0   341.4  572.0  146.0  150.0  0.5000
3595  5.50  101.0  35.0    24.0  338.0  184.0  280.0  1.2750
2377  6.40   60.0  40.0  1145.6  189.0  218.0  280.0  1.6750
...    ...    ...   ...     ...    ...    ...    ...     ...
4931  5.20  209.0  35.0    10.2  696.0  320.0  280.0  0.9500
3264  6.20  123.0  33.0   621.6  147.0  222.0  280.0  1.1500
1653  6.00   97.0  37.0   233.4  559.0  212.0  280.0  1.4375
2607  5.50  120.0  33.0   147.4  698.0  175.0  280.0  1.8500
2732  6.10   64.0  40.0   532.2  223.0  257.0  280.0  1.5625

[4946 rows x 58 columns]
```

接著把這些 feature data 標準化

```
y_train = train_df.iloc[:,-1]
y_test = test_df.iloc[:,-1]

scaler_Std = preprocessing.StandardScaler().fit(train_predata)

scaler_MinMax = preprocessing.MinMaxScaler().fit(train_predata)
x_pretrain_Std = scaler_Std.transform(train_predata)
x_pretest_Std = scaler_Std.transform(test_predata)
x_pretrain_MinMax = scaler_MinMax.transform(train_predata)
x_pretest_MinMax = scaler_MinMax.transform(test_predata)

print(x_pretrain_Std)
```

- **模型訓練過程:**

1. SVC

A. Feature selection:

使用 RFECV 做進一步的特徵挑選

```python
svc = SVC(kernel = 'linear')
n_feature, selected = rfe_f_select(svc, x_pretrain_Std, y_train)
```

```
Optimal number of features : 43
Support is [ True False  True  True  True  True  True  True  True False False False
  True  True False  True  True  True  True  True  True False  True  True
  True  True  True False  True False False False  True  True  True  True
  True  True False  True False False False  True  True  True  True  True
  True  True  True  True  True  True  True False  True  True]
Ranking of features : [ 1 15  1  1  1  1  1  1  1  5  9  3  1  1 16  1  1  1  1  1  1 12  1  1
  1  1  1  6  1  7 11 10  1  1  1  1  1  1  8  1  2  4 14  1  1  1  1  1
  1  1  1  1  1  1  1 13  1  1]
```

取出選完的特徵:

```python
svc_train_data = train_predata.loc[:, selected]
svc_test_data = test_predata.loc[:, selected]

print(svc_train_data.columns)
print(svc_train_data)
```

```
Index(['d1', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'd15', 'd20', 'd22',
       'd23', 'd24', 'd27', 'd28', 'd29', 'd31', 'd32', 'd33', 'd34', 'd37',
       'c43', 'c50', 'c51', 'c53', 'c55', 'c56', 'c59', 'c61', 'c66', 'c67',
       'c70', 'c71', 'c72', 'c73', 'c74', 'c75', 'c78', 'c80', 'c82', 'c83',
       'c89', 'c90'],
      dtype='object')
        d1    d3  d4   d5   d6    d7  d8   d9   d15    d20  ...   c72   c73  \
3318  1.70  1250   1  1.0  0.5  40.0   1  2.5  17.0  16.00  ...   9.5  12.6
2701  1.60  1000   2  1.0  1.0  40.0   1  2.5  16.0  16.00  ...  10.1  10.6
5862  1.60     0   2  2.0  1.0  50.0   1  2.5  16.0  16.95  ...   9.3  11.3
3595  1.64   250   1  1.0  0.5  40.0   1  3.0  16.0  17.00  ...   9.3  10.6
2377  1.60  1000   2  1.0  1.0  40.0   1  2.5  16.0  16.00  ...  10.1  11.4
...    ...   ...  ..  ...  ...   ...  ..  ...   ...    ...  ...   ...   ...
4931  1.60     0   2  1.0  1.5  40.0   1  3.0  16.0  16.00  ...  10.9  10.3
3264  1.70  1250   1  1.0  0.5  40.0   1  2.5  17.0  15.00  ...   9.2   9.2
1653  1.63   750   2  1.0  1.0  50.0   1  2.5  16.0  16.00  ...   9.5  12.2
2607  1.68  1000   2  0.0  0.5  40.0   1  2.5  16.0  16.00  ...   8.2  10.7
2732  1.60  1000   2  1.0  1.0  40.0   1  2.5  16.0  16.00  ...  10.7  11.0

         c74   c75    c78   c80     c82    c83    c89     c90
3318  2.9185  6.00  143.0  33.0   621.6   51.0  280.0  1.1500
2701  1.1365  6.22   70.0  40.0   532.2  223.0  280.0  1.5625
5862  7.9100  6.90  132.0  24.0   341.4  572.0  150.0  0.5000
3595  7.8165  5.50  101.0  35.0    24.0  338.0  280.0  1.2750
2377  1.1365  6.40   60.0  40.0  1145.6  189.0  280.0  1.6750
```

取出的特徵為:

['d1', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8','d9','d15', 'd20', 'd22', 'd23','d24', 'd27', 'd28', 'd29', 'd31', 'd32', 'd33', 'd34', 'd37', 'c43', 'c50', 'c51', 'c53', 'c55','c56', 'c59', 'c61', 'c66', 'c67', 'c70', 'c71', 'c72', 'c73', 'c74', 'c75', 'c78', 'c80', 'c82', 'c83', 'c89', 'c90']

共 43 個

B. 特徵資料標準化

取完要用的特徵之後把他們標準化

```python
svc_scaler_Std = preprocessing.StandardScaler().fit(svc_train_data)

svc_scaler_MinMax = preprocessing.MinMaxScaler().fit(svc_train_data)
svc_x_train_Std = svc_scaler_Std.transform(svc_train_data)
svc_x_test_Std = svc_scaler_Std.transform(svc_test_data)
#svc_x_train_MinMax = svc_scaler_MinMax.transform(svc_train_data)
#svc_x_test_MinMax = svc_scaler_MinMax.transform(svc_test_data)

print(svc_x_train_Std)
```

```
[[ 1.07689771  0.98260277 -0.5702343  ... -1.58739438  0.25256772
  -0.33489454]
 [-0.72008145  0.48048961  0.50501498 ... -0.81229314  0.25256772
   0.09048478]
 [-0.72008145 -1.52796304  0.50501498 ...  0.76044135 -5.01003395
  -1.00518923]
 ...
 [-0.1809877  -0.02162356  0.50501498 ...  0.70185812  0.25256772
  -0.03841805]
 [ 0.71750188  0.48048961  0.50501498 ...  1.32824808  0.25256772
   0.38696127]
 [-0.72008145  0.48048961  0.50501498 ... -0.81229314  0.25256772
   0.09048478]]
```

C. 用 grid search 找最佳超參數

```python
param_grid = {
              'kernel': ['linear', 'rbf'],
              'C': [1, 4, 10, 12, 14, 16],
              'gamma': [0.001, 0.01, 0.1, 1, 10]
              }

model = SVC()
grid_search(model, param_grid, svc_x_train_Std, y_train)
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
{'C': 12, 'gamma': 1, 'kernel': 'rbf'}
```

由 grid search 跑出的結果可知最好的參數為: C = 12, gamma = 1, kernel = rbf

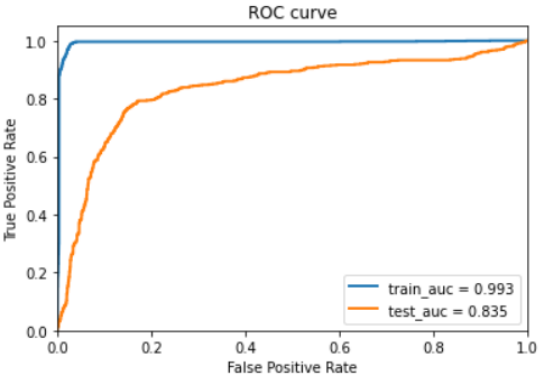
D. Training Model

```python
svc_model = SVC(kernel = 'rbf', C = 12, gamma = 1)
svc_model.fit(svc_x_train_Std, y_train)
plt.figure()
df_svc, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = model_evaluation_result(svc_x_train_Std, y_train, svc_x_test_Std, y_test, svc_model, 'train', 'test')
plt.show()
print_cm(svc_model, svc_x_train_Std, y_train)
```

其訓練結果如下:

ROC_AUC

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.974	0.981	0.952	0.933	0.949	0.983
Test_set	0.830	0.895	0.651	0.557	0.692	0.876

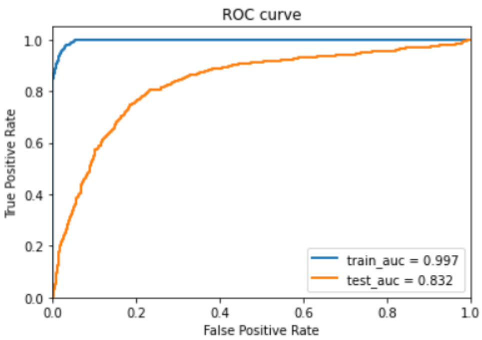


Confusion matrix:

```
[668 660 661 655 655] 660
[57 64 63 69 69] 64
[ 89  83  99  95 110] 95
[176 182 166 170 155] 170
accuracy: 0.839231547017189
precision: 0.7264957264957265
sensitivity: 0.6415094339622641
specificity: 0.9116022099447514
MCC: 0.5763995282854502
```

*助教範例的結果:

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.971	0.981	0.946	0.927	0.947	0.98
Test_set	0.807	0.881	0.605	0.496	0.647	0.86




```
[1084 1069 1073] 1075
[123 138 134] 132
[195 179 177] 184
[247 263 264] 258
accuracy: 0.8083687083080655
precision: 0.6615384615384615
sensitivity: 0.583710407239819
specificity: 0.8906379453189727
MCC: 0.494443870870684
```

同樣是用 SVC 但經過特徵挑選後 specificity、sensitivity、MCC、PPV、NPV 以及 test 的 AUC 都上升了一點點，且用得特徵數也較少。

2. KNN

A. Feature selection

因為 KNN 沒有 logic 也沒有 feature_importances，因此似乎只能使用 filter 方法挑特徵，因此直接使用特徵預選所選出來的 58 個特徵:

['d1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'd11', 'd13', 'd14', 'd15', 'd20', 'd21', 'd22', 'd23', 'd24', 'd27', 'd28', 'd29', 'd30', 'd31', 'd32', 'd33', 'd34', 'd37', 'd39', 'c43', 'c45', 'c46', 'c49', 'c50', 'c51', 'c53', 'c55', 'c56', 'c59', 'c60', 'c61', 'c62', 'c64', 'c65', 'c66', 'c67', 'c70', 'c71', 'c72', 'c73', 'c74', 'c75', 'c78', 'c80', 'c82', 'c83', 'c84', 'c89', 'c90']

B. 用迴圈找最佳超參數以及比較好的前處理資料

先跑使用標準化(Std)的資料

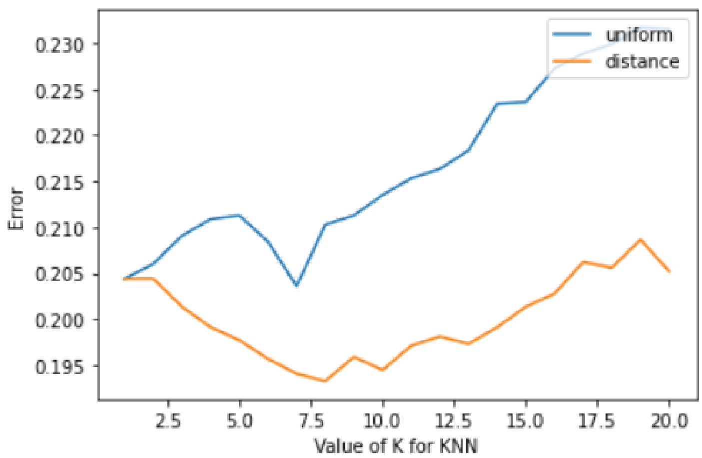
```
k_range = range(1, 21)
uni_k_error = []
dis_k_error = []

for method in ["uniform", "distance"]:
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k, weights = method)
        scores = cross_val_score(knn, x_pretrain_Std, y_train, cv=5, scoring='accuracy')

        if(method == 'uniform'): uni_k_error.append(1 - scores.mean())
        else: dis_k_error.append(1 - scores.mean())

print(uni_k_error)
print(dis_k_error)
plt.plot()
unifrom, = plt.plot(k_range, uni_k_error, label = 'uniform')
distance, = plt.plot(k_range, dis_k_error, label = 'distance')
plt.xlabel('Value of K for KNN')
plt.ylabel('Error')
plt.legend(handles = [unifrom, distance], loc='upper right')
plt.show()
```

```
[0.20441033183196988, 0.2060248593110069, 0.20905720501271563,
[0.20441033183196988, 0.20441033183196988, 0.20137553492457427,
```



由圖表可看出選 distance 且 n_neighbor = 8 時的 score 誤差較小,約為 0.193

再跑使用 MinMax 的資料

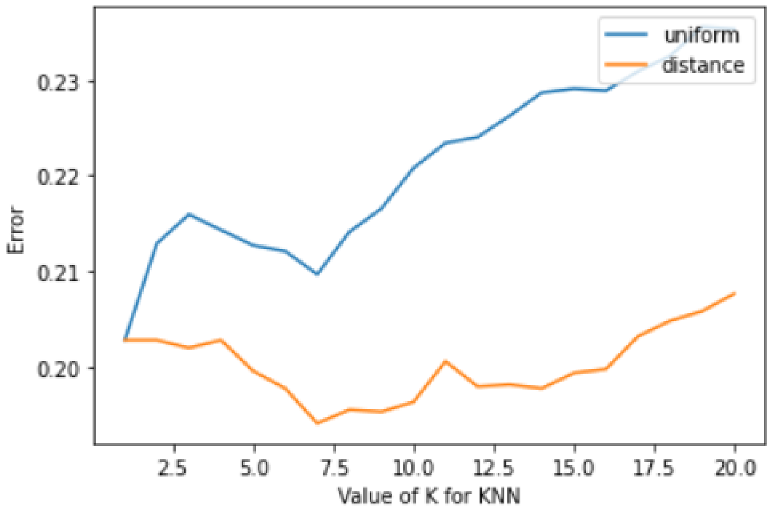
```
k_range = range(1, 21)
uni_k_error = []
dis_k_error = []

for method in ["uniform", "distance"]:
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k, weights = method)
        scores = cross_val_score(knn, x_pretrain_MinMax, y_train, cv=5, scoring='accuracy')

        if(method == 'uniform'): uni_k_error.append(1 - scores.mean())
        else: dis_k_error.append(1 - scores.mean())

print(uni_k_error)
print(dis_k_error)
plt.plot()
unifrom, = plt.plot(k_range, uni_k_error, label = 'uniform')
distance, = plt.plot(k_range, dis_k_error, label = 'distance')
plt.xlabel('Value of K for KNN')
plt.ylabel('Error')
plt.legend(handles = [unifrom, distance], loc='upper right')
plt.show()
```

```
[0.20279253607868364, 0.21289844859106744, 0.21593161136133832,
[0.20279253607868364, 0.20279253607868364, 0.20198200406491607,
```



由圖表可看出選 distance 且 n_neighbor = 7 時的 score 誤差最小，其誤差約為 0.194
使用 Std 的誤差較小,因此選用 Std 的資料

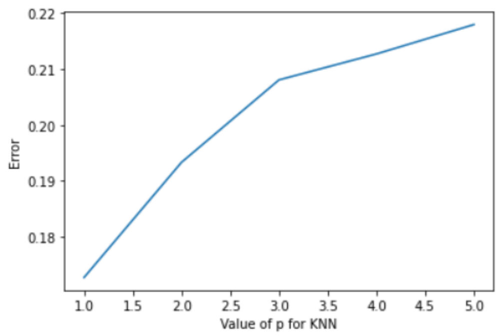
因為使用 distance(有考慮距離權重)，因此要找最佳的距離算法 p。 同樣使用迴圈找最佳 p 值:

```
p_range = range(1, 6)
p_error = []

for p in p_range:
    knn = KNeighborsClassifier(n_neighbors= 8, weights = 'distance', p = p)

    scores = cross_val_score(knn, x_pretrain_Std, y_train, cv=5, scoring='accuracy')
    p_error.append(1 - scores.mean())


plt.plot(p_range, p_error)
plt.xlabel('Value of p for KNN')
plt.ylabel('Error')
plt.show()
```



由圖表可知當 p = 1 (曼哈頓距離)時, score 誤差最小

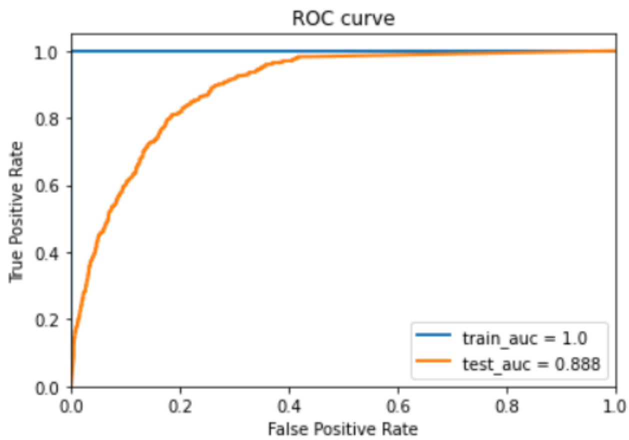
C. Training model:

```
knn_model = KNeighborsClassifier(n_neighbors= 8, weights = 'distance', p = 1)
knn_model.fit(x_pretrain_Std, y_train)
plt.figure()
df_knn, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = proba_model_evaluation_result(x_pretrain_Std, y_train, x_pretest_Std, y_test, knn_model, 'train', 'test')
plt.show()
print_cm(knn_model, x_pretrain_Std, y_train)
```

代入上面所得的超參數: n_neighbor =8, weights=distance, p=1。並以 Std 做為訓練集, 其結果為:

ROC_AUC:

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	1.000	1.000	1.000	1.000	1.000	1.000
Test_set	0.819	0.878	0.656	0.535	0.661	0.876



Confusion matrix:

```
[656 639 640 646 653] 647
[69 85 84 78 71] 77
[88 94 94 95 96] 93
[177 171 171 170 169] 172
accuracy: 0.8281092012133469
precision: 0.6907630522088354
sensitivity: 0.6490566037735849
specificity: 0.893646408839779
MCC: 0.5537821066397217
```

3. RandomForest

A. feature selection

利用 RFEVC 進一步的特徵挑選

```
rf = RandomForestClassifier()
n_feature, selected = rfe_f_select(rf, x_pretrain_Std, y_train)
```

```
Optimal number of features : 3
Support is [False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False  True  True False False False False False False False False False
  True False False False False False False False False False]
Ranking of features : [21  8 27 39 32 36 37 44 35 41 56 45 48 33 28 25 47 55 54 49 46 38 43 40
 42 53 50 52  2  4 11 22 30 34 31 26 24  1  1  3  6  7 15 13 18 20  5 12
  1 19  9 17 29 10 16 14 51 23]
```

挑出選擇的特徵

```
rf_train_data = train_predata.loc[:, selected]
rf_test_data = test_predata.loc[:, selected]

print(rf_train_data.columns)
print(rf_train_data)
```

```
Index(['c59', 'c60', 'c73'], dtype='object')
       c59    c60   c73
3318  41.0  218.0  12.6
2701  32.1  116.0  10.6
5862  32.9  190.0  11.3
3595  33.6  263.0  10.6
2377  36.4  299.0  11.4
...    ...    ...   ...
4931  33.1  236.0  10.3
3264  33.2  256.0   9.2
1653  36.7  262.0  12.2
2607  32.0  283.0  10.7
2732  32.8   91.0  11.0

[4946 rows x 3 columns]
```

所選擇的是 ['c59', 'c60', 'c73'] 共三個

B. 特徵資料標準化

```
rf_scaler_Std = preprocessing.StandardScaler().fit(rf_train_data)

rf_scaler_MinMax = preprocessing.MinMaxScaler().fit(rf_train_data)
rf_x_train_Std = rf_scaler_Std.transform(rf_train_data)
rf_x_test_Std = rf_scaler_Std.transform(rf_test_data)

print(rf_x_train_Std)
```

C. 用 grid search 找最佳超參數

第一次

```
param_grid = {
        'max_depth': [10, 20, 30, 40],
        'min_samples_leaf': [1, 2, 4],
        'min_samples_split': [2, 4, 8],
        'n_estimators': [100, 150, 200, 250]
}

model = RandomForestClassifier()
grid_search(model, param_grid, rf_x_train_Std, y_train)
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
{'max_depth': 40, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 250}
```

因為 max_depth 和 n_estimators 都剛好在邊界，因此再做一次。

第二次

```
param_grid = {
        'max_depth': [40, 50, 60, 70],
        'criterion': ['entropy', 'gini'],
        'min_samples_leaf': [1, 2, 4],
        'min_samples_split': [2, 4, 8],
        'n_estimators': [250, 300, 350, 400]
}

model = RandomForestClassifier()
grid_search(model, param_grid, rf_x_train_Std, y_train)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
{'criterion': 'gini', 'max_depth': 70, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 350}
```

跑出來的結果為 criterion=gini, max_depth=70, min_samples_leaf=1, min_samples_split=2, n_estimators=350，但 max_depth 還是在邊界上，因此再做第三次

第三次

```
param_grid = {
        'max_depth': [70, 80, 90],
        'min_samples_leaf': [1],
        'min_samples_split': [2],
        'n_estimators': [300, 350, 400]
}

model = RandomForestClassifier()
grid_search(model, param_grid, rf_x_train_Std, y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
{'max_depth': 80, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 350}
```

得出來的最佳 max_depth=80

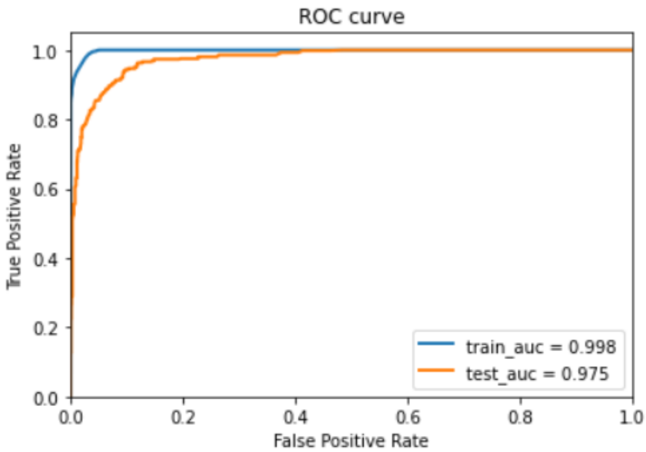
D. Training Model

```
rf_model = RandomForestClassifier(max_depth= 80, min_samples_leaf=1, min_samples_split=2, n_estimators=350)
rf_model.fit(rf_x_train_Std, y_train)
plt.figure()
df_rf, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = proba_model_evaluation_result(rf_x_train_Std, y_train, rf_x_test_Std, y_test, rf_model, 'train', 'test')
plt.show()
print_cm(rf_model, rf_x_train_Std, y_train)
```

結果如下:

ROC_AUC

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.975	0.983	0.952	0.936	0.955	0.982
Test_set	0.925	0.942	0.878	0.809	0.845	0.955



Confusion matrix

```
[689 692 684 681 688] 687
[36 32 40 43 36] 37
[35 43 51 46 47] 44
[230 222 214 219 218] 221
accuracy: 0.9180990899898888
precision: 0.8565891472868217
sensitivity: 0.8339622641509434
specificity: 0.9488950276243094
MCC: 0.7895984364684694
```

4. Xgboot

A. feature selection

一樣使用 RFECV

```
xgb = XGBClassifier()
n_feature, selected = rfe_f_select(xgb, x_pretrain_Std, y_train)
```

```
Optimal number of features : 28
Support is [ True  True  True False False  True  True  True False False False False
 False False False  True False False  True False False False False False
  True False False False False False False  True  True False False  True
 False  True  True  True  True  True  True  True  True  True  True  True
  True  True  True False False  True False  True False False]
Ranking of features : [ 1  1  1  6 10  1  1  1 23 25 18 17 15 19 20  1  3 13  1 27 28 26 24 22
  1 14 12 29  8  9 11  1  1 16  5  1  7  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  2 21  1  4  1 30 31]
```

接著取出特徵

```
xgb_train_data = train_predata.loc[:, selected]
xgb_test_data = test_predata.loc[:, selected]

print(xgb_train_data.columns)
print(xgb_train_data)
print(xgb_test_data)
```

所選擇的是['d1', 'd2', 'd3', 'd6', 'd7', 'd8', 'd22', 'd27','d33', 'c49', 'c50', 'c55', 'c59', 'c60', 'c61','c62','c64','c65','c66', 'c67', 'c70', 'c71', 'c72', 'c73', 'c74', 'c75','c82', 'c84']共 28 個

B. 特徵資料標準化

```
xgb_scaler_Std = preprocessing.StandardScaler().fit(xgb_train_data)

xgb_scaler_MinMax = preprocessing.MinMaxScaler().fit(xgb_train_data)
xgb_x_train_Std = xgb_scaler_Std.transform(xgb_train_data)
xgb_x_test_Std = xgb_scaler_Std.transform(xgb_test_data)

print(xgb_x_train_Std)
```

C. 用 RandomSearchCV 找超參數

```
random_grid = {
        'n_estimators':[100, 200, 300, 400],
        'max_depth': [20, 30, 40, 50, 60],
        'learning_rate': [0.01, 0.1, 0.2, 0.3, 0.4],
        'gamma': [0.1, 1, 10]
}

model = XGBClassifier()
model_RSCV = RandomizedSearchCV(estimator = model, param_distributions=random_grid, n_iter=100, cv=5, verbose=1, n_jobs=-1)

model_RSCV.fit(xgb_x_train_Std,y_train)
print(model_RSCV.best_params_)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
{'n_estimators': 100, 'max_depth': 20, 'learning_rate': 0.4, 'gamma': 0.1}
```

因為 Xgboot 用 grid search 算得比較久，因此改為使用 RandomSearchCV，快

了一點。其找出來的超參數為: n_estimator=100, max_depth=20, learning_rate=0.4, gamma=0.1

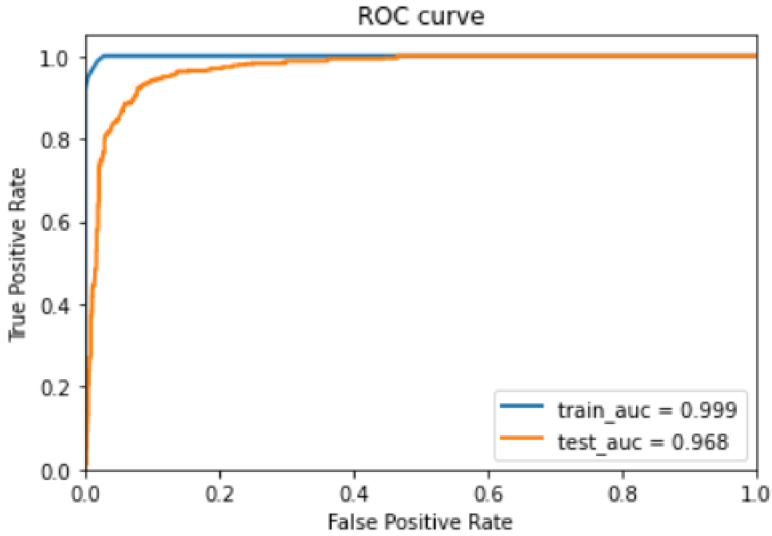
D. Training Model

```
xgb_model = XGBClassifier(n_estimators=100, max_depth=20, learning_rate=0.4, gamma=0.1)
xgb_model.fit(xgb_x_train_Std, y_train)
plt.figure()
df_xgb, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = proba_model_evaluation_result(xgb_x_train_Std, y_train, xgb_x_test_Std, y_test, xgb_model, 'train', 'test')
plt.show()
print_cm(xgb_model, xgb_x_train_Std, y_train)
```

其結果為:

ROC_AUC

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.983	0.989	0.968	0.958	0.970	0.988
Test_set	0.925	0.945	0.869	0.808	0.851	0.952



Confusion matrix

```
[690 699 684 690 689] 690
[35 25 40 34 35] 34
[30 41 38 43 40] 38
[235 224 227 222 225] 227
accuracy: 0.9271991911021233
precision: 0.8697318007662835
sensitivity: 0.8566037735849057
specificity: 0.9530386740331491
MCC: 0.8135786548404282
```

```python
random_grid = {
        'n_estimators':[100, 200, 300, 400],
        'max_depth': [20, 30, 40, 50, 60],
        'learning_rate': [0.01, 0.1, 0.2, 0.3, 0.4],
        'gamma': [0.1, 1, 10]
}

model = XGBClassifier()
model_RSCV = RandomizedSearchCV(estimator = model, param_distributions=random_grid, n_iter=100, cv=5, verbose=1, n_jobs=-1)

model_RSCV.fit(xgb_x_train_Std,y_train)
print(model_RSCV.best_params_)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
{'n_estimators': 100, 'max_depth': 20, 'learning_rate': 0.4, 'gamma': 0.1}
```

上傳的模型名稱以及其對應的訓練紀錄:

模型名稱: finalmodel.joblib

```python
dump(rf_model, 'finalmodel.joblib')
```

對應的訓練紀錄: 在上方的建模過程中的 RandomForest

結論:

這次建了 4 個模型，以結果來說 RandomForset 以及 Xgboot 的 test_set 評估表現高於 SVM 以及 KNN， 而 RandomForset 和 Xgboot 的指標各有勝過對方的地方，但是兩者 accuracy 一樣，就特徵數以及 test_AUC 來看 RandomForest 的表現比 Xgboot 好一點點，因此選擇 Randomforest。