Machine Learning in Computational Biology

HW2 report

109350008　張詠哲

這次實驗了 4 種模型分別為 SVM、KNN、RandomForest、XGBoost，而以下為記錄我資料切分、特徵挑選、參數最佳化的實驗過程，也會比較只用統計方法所選出來的特徵以及再更進一步用 model 去選特徵的表現結果。至於一些用到的 package 以及自訂函式，因為占版面因此會放在報告的最後面。

目錄

建模訓練紀錄

- **資料切分**

下載完資料後先看看 label 的分布情況，發現 0:1 的比例為 5178:1889(約為 2.74:1)，有一點點不平均，因此我是採用 StratifiedShuffleSplit 去按照 0 跟 1 的比例切分資料，以防 label 不平均造成 train 以及 test 的不平衡。並且 train 以及 test 的比例為 7:3。

```
rd = pd.read_csv("/content/drive/MyDrive/BinaryClassifier_2weeks_data_v2.csv")
```

```
from collections import Counter
label_count = Counter(rd["label"])
print("label count:", label_count)
```

```
label count: Counter({0: 5178, 1: 1889})
```

```
x = rd.iloc[:, 0:-1]
y = rd.iloc[:, -1]

sss = StratifiedShuffleSplit(n_splits=1,test_size=0.3,random_state=0)
for train_index, test_index in sss.split(x, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = x.iloc[train_index,:], x.iloc[test_index,:]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [2426 3567 2003 ... 4767 4405 2960] TEST: [1833 3223 6953 ... 7003 3553 2313]
```

- **特徵初選(統計方法)**

因為此資料集中含有類別以及連續型變數，且這兩種所使用的統計方法皆不一樣，因此需要先將這兩種資料分別提取出來去做統計顯著性的分析。

```
c_idx = []
d_idx = []
idx = 0
for i in rd.columns:
    if(i[0] == 'd'): d_idx.append(idx)
    elif(i[0] == 'c'): c_idx.append(idx)
    idx += 1
```

```
d = x_train.iloc[:, d_idx]
c = x_train.iloc[:, c_idx]
```

這裡類別型變數所使用的是 Chi-square 的方式，而連續型是使用 Mann–Whitney U test，接著把這兩者所選出來的特徵合併並從原始資料中提取出這些特徵的 data。總共選出來的特徵共 **170 個**。

```
d_s = pd.concat([d, y_train], axis = 1)
d_prefeature = []
p_value = []
d_s = d_s[d_s['label'] >= 0]
columns = list(d_s.columns)
for i in columns:
        table = pd.crosstab(d_s['label'], d_s[i])
        chi2, p, dof, expected = chi2_contingency(table)
        p_value.append(p)
        if p < 0.05:
            d_prefeature.append(i)

d_p_value = pd.DataFrame([columns, p_value]).T
d_p_value = d_p_value.rename(columns = {0:'features', 1:'p_value'})
print(d_p_value)
print(d_prefeature[0:-1])
```

```
c_s = pd.concat([c, y_train], axis = 1)

c_s = c_s[c_s['label'] >= 0]
columns = list(c_s.columns)
c_prefeature = []
p_value = []
for i in columns:
        value1=[]
        value0=[]
        my_col = c_s[[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')
        p_value.append(result[1])

        if result[1] < 0.05:
            c_prefeature.append(i)

c_p_value = pd.DataFrame([columns, p_value]).T
c_p_value = c_p_value.rename(columns = {0:'features', 1:'p_value'})
print(c_p_value)
print(c_prefeature[0:-1])
```

```
prefeature = d_prefeature[0:-1] + c_prefeature[0:-1]
x_pretrain = x_train.loc[:, prefeature]
x_pretest = x_test.loc[:, prefeature]
y_train = y_train
y_test = y_test
```

- **模型訓練過程**

在上面用統計方法選完特徵後，在要進模型訓練前先把資料都標準化，使模型可以有更好的表現。

```
x_pretrain_std = std(x_pretrain)
x_pretest_std = std(x_pretest)
```

```
x_pretrain_std = pd.DataFrame(x_pretrain_std)
x_pretest_std = pd.DataFrame(x_pretest_std)

x_pretrain_std.columns = prefeature
x_pretrain_std.index = train_index
x_pretest_std.columns = prefeature
x_pretest_std.index = test_index
```

接著以下 4 種模型的訓練過程(由於有些參數太多要試且資料集較大建模需要的時間比較多，因此改使用 Randomized grid search):

1. **SVC**

- **Statistic only**

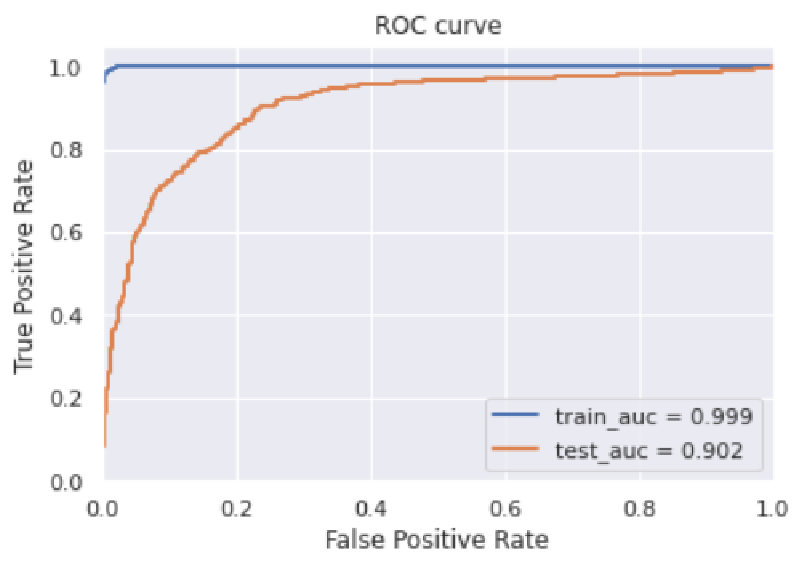
先看看若是只用統計方法的所選出來的特徵表現如何。
找出最佳超參數後進行訓練:

```
param_grid = {
              'kernel': ["rbf"],
              'C': [256, 128, 64, 32, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125],
              'gamma': [256, 128, 64, 32, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125]
              }

model = SVC()
best_para = Randomized_gridsearch(model, param_grid, x_pretrain_std, y_train)
```

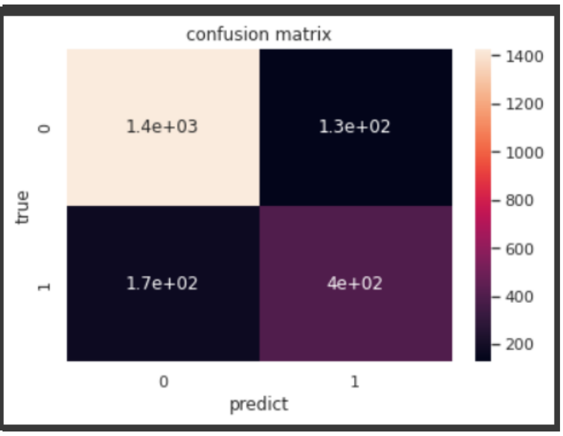
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'kernel': 'rbf', 'gamma': 0.015625, 'C': 16}
```

- 結果:

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.992	0.995	0.983	0.978	0.986	0.994
Test_set	0.860	0.918	0.700	0.635	0.758	0.894




```
TN_result:  [658 663 651 662 655]  TN_mean:  658
FP_result:  [67 62 74 63 69]  FP_result:  67
FN_result:  [87 75 81 64 77]  FN_result:  77
TP_result:  [178 189 183 200 188]  TP_result:  188
accuracy: 0.8545454545454545
precision: 0.7372549019607844
sensitivity: 0.7094339622641509
specificity: 0.9075862068965517
MCC: 0.6247086810062852
```



- **With Feature selection**

因為 SVC 在 kernel 為 rbf 時沒有 coef 或是 feature importance 可以去使用 embbed 或是 waper 的方法。sklearn 有一個可以為這種情況產出特徵重要程度的 package 叫 permutation_importance，這個函式對於特徵重要程度的計算取決於該特徵被隨機重排後，模型表現的下降程度。以下以及 KNN 都會用這個方式去做特徵挑選的方式。

先使用 permutation_importance 做進一步的特徵挑選(把不為 0 的去掉)，總共選出來的有 166 個特徵

```
permut_model = SVC(kernel = "rbf").fit(x_pretrain_std, y_train)

perm = permutation_importance(permut_model, x_pretrain_std, y_train, n_repeats = 5, scoring = "accuracy", random_state=0)

important_features=[]
for i in perm.importances_mean.argsort()[::-1]:
    if perm.importances_mean[i] != 0:
        important_features.append(x_pretrain_std.columns[i])

print("n_feature: ", len(important_features))
print(important_features)
```

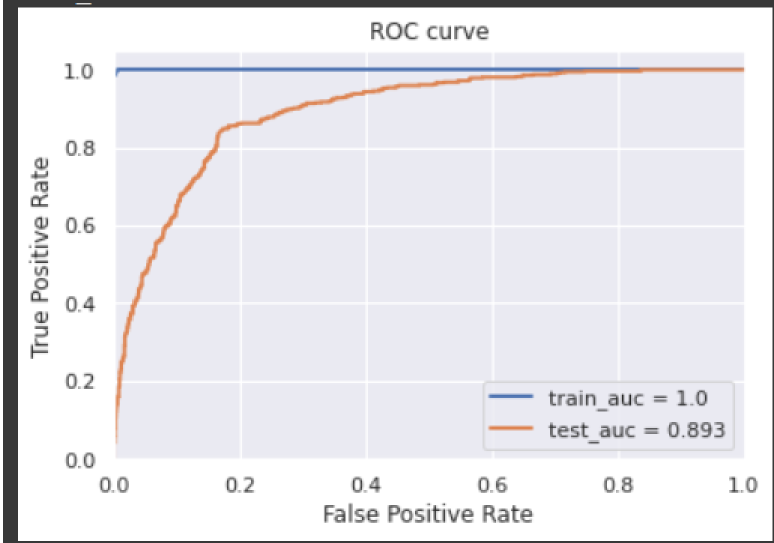
接著找最佳超參數後進行訓練。

```
param_grid = {
            'kernel': ["rbf"],
            'C': [256, 128, 64, 32, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125],
            'gamma': [256, 128, 64, 32, 16, 8, 4, 2, 1, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125]
            }

model = SVC()
best_para = Randomized_gridsearch(model, param_grid, svc_train, y_train)
```

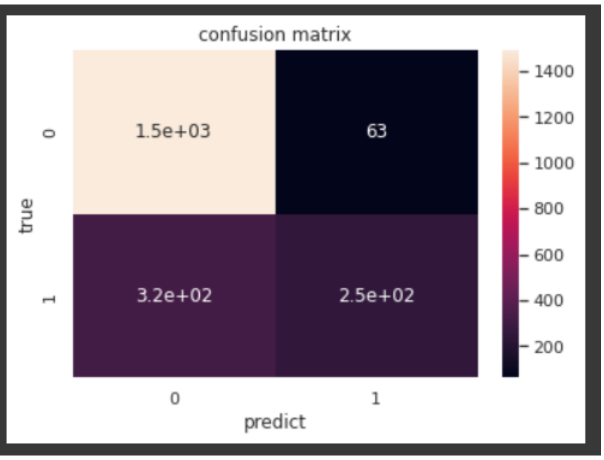
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'kernel': 'rbf', 'gamma': 0.125, 'C': 2}
```

- 結果:

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.997	0.998	0.992	0.991	0.995	0.997
Test_set	0.820	0.959	0.439	0.498	0.798	0.824



```
TN_result:  [700 699 690 690 691]  TN_mean:  694
FP_result:  [25 26 35 35 33]  FP_result:  31
FN_result:  [160 152 162 134 151]  FN_result:  152
TP_result:  [105 112 102 130 114]  TP_result:  113
accuracy: 0.815151515151515152
precision: 0.7847222222222222
sensitivity: 0.42641509433962266
specificity: 0.9572413793103448
MCC: 0.48180138420044727
```



2. KNN

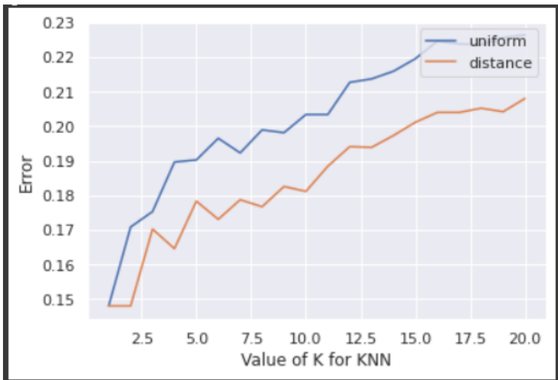
- **Statistic only**

先使用統計方法所挑出的特徵去做超參數挑選後訓練。

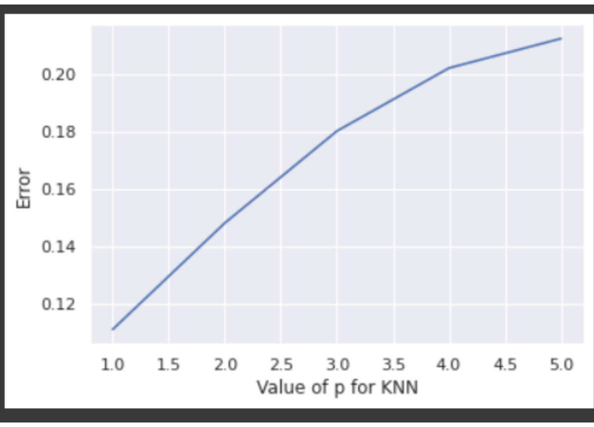
找出最佳的 k 值、method，由跑出的圖可以看出當 k 值越來越大時誤差也會越大，且當 method 為 distance 時的誤差較 uniform 小。由於 method 為 distance 所以也要選 p 值作為算距離的方式。最後最好的 p 值為 1 時誤差最小。

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

ubest_k = 0
ubest_error = 1
dbest_k = 0
dbest_error = 1
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())
            if ( (1 - scores.mean()) < ubest_error):
                ubest_k = k
                ubest_error = 1 - scores.mean()
        else:
            dis_k_error.append(1 - scores.mean())
            if ( (1 - scores.mean()) < dbest_error):
                dbest_k = k
                dbest_error = 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()

print("u_k: ", ubest_k)
print("d_k: ", dbest_k)
```

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

for p in p_range:
    knn = KNeighborsClassifier(n_neighbors= 1, 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()
```

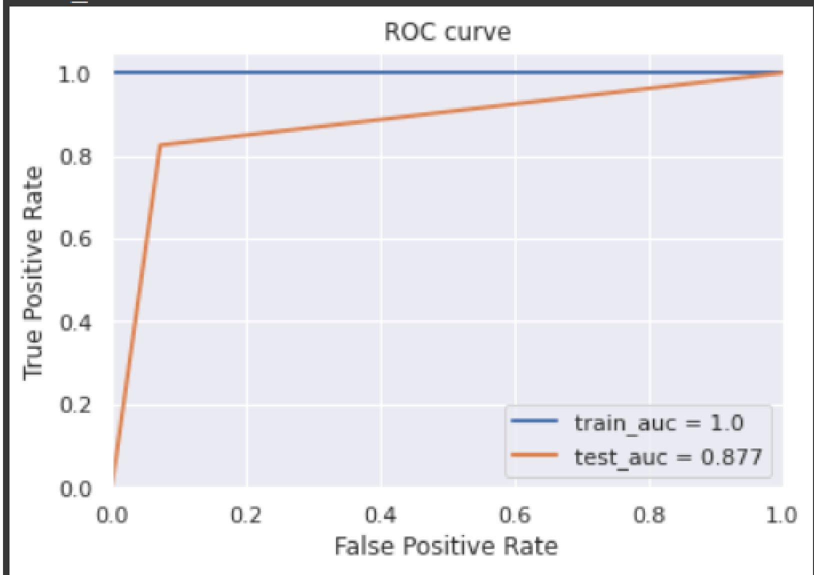



```
u_k:  1
d_k:  1
```

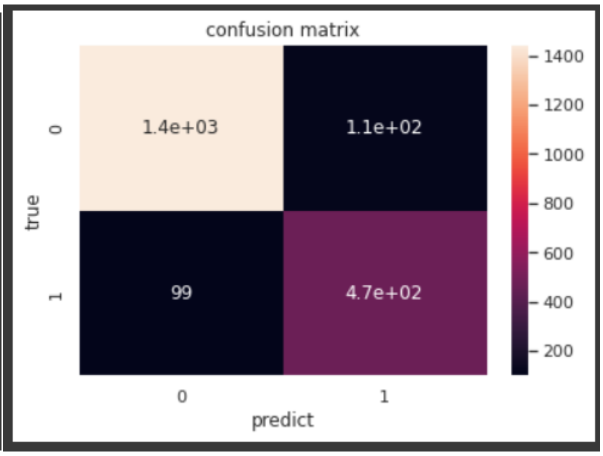


- 結果:

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	1.000	1.000	1.000	1.000	1.000	1.000
Test_set	0.901	0.928	0.825	0.748	0.807	0.936



```
TN_result:  [670 663 671 676 662]  TN_mean:  668
FP_result:  [55 62 54 49 62]  FP_result:  56
FN_result:  [67 50 55 46 48]  FN_result:  53
TP_result:  [198 214 209 218 217]  TP_result:  211
accuracy: 0.8896761133603239
precision: 0.7902621722846442
sensitivity: 0.7992424242424242
specificity: 0.9226519337016574
MCC: 0.7193191642776955
```

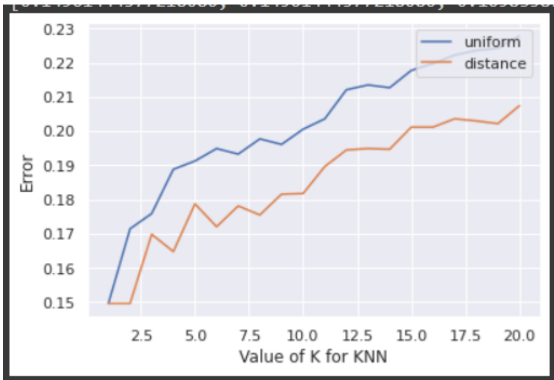


- **With Feature selection**

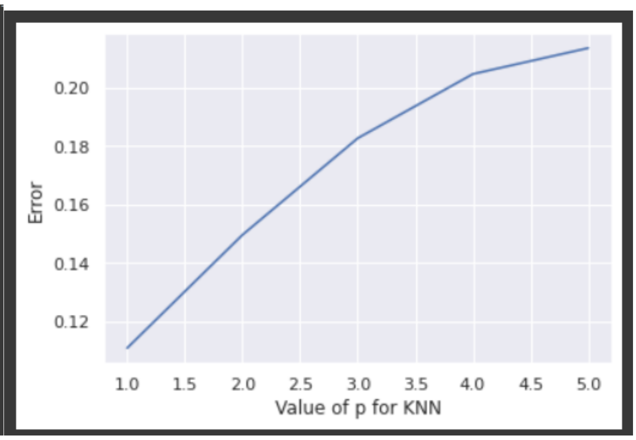
由於 KNN 也沒有 coef 或是 feature importance 相關的 attribute，因此這次也是使用 permutation_importance 來做更進一步的特徵篩選(permutation_importance 相關的解釋在 SVC 的特徵挑選有講述)，所選出來的特徵為 136 個。

```
permut_model = KNeighborsClassifier().fit(x_pretrain_std, y_train)

perm = permutation_importance(permut_model, x_pretrain_std, y_train, n_repeats = 5, scoring = "accuracy", random_state=0)

important_features=[]
for i in perm.importances_mean.argsort()[::-1]:
    if perm.importances_mean[i] != 0:
        important_features.append(x_pretrain_std.columns[i])

print("n_feature: ", len(important_features))
print(important_features)
```

接著比照上面的方式找出的最佳超參數為 distance、k = 1、p = 1。

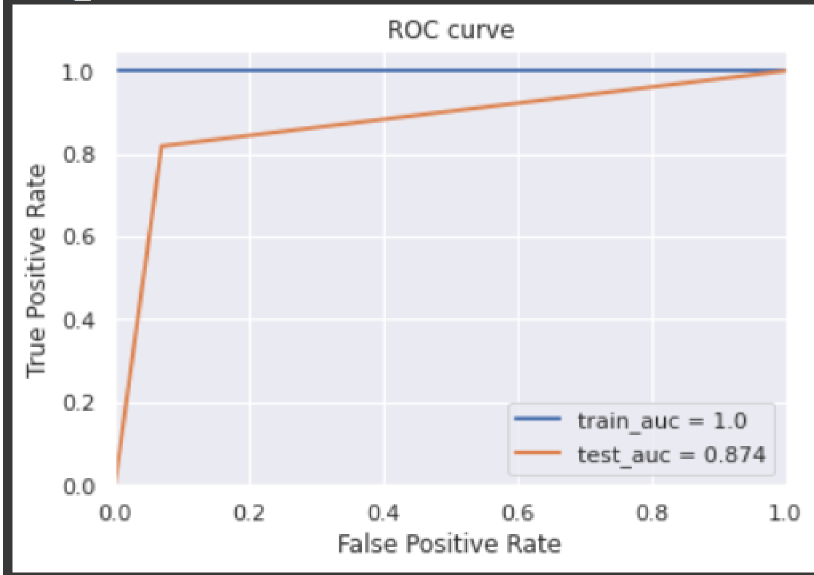



```
u_k:  1
d_k:  1
```

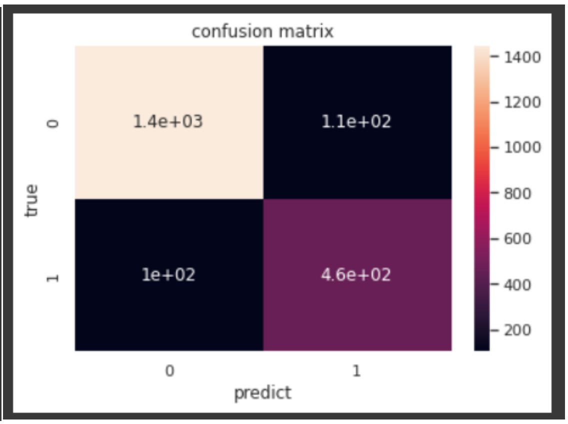


- 結果:

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	1.000	1.000	1.000	1.000	1.000	1.000
Test_set	0.901	0.931	0.818	0.747	0.811	0.934




```
TN_result:  [670 663 671 676 662]  TN_mean:  668
FP_result:  [55 62 54 49 62]  FP_result:  56
FN_result:  [67 50 55 46 48]  FN_result:  53
TP_result:  [198 214 209 218 217]  TP_result:  211
accuracy: 0.8896761133603239
precision: 0.7902621722846442
sensitivity: 0.7992424242424242
specificity: 0.9226519337016574
MCC: 0.7193191642776955
```



3. RandomForest

- **Statistic only**

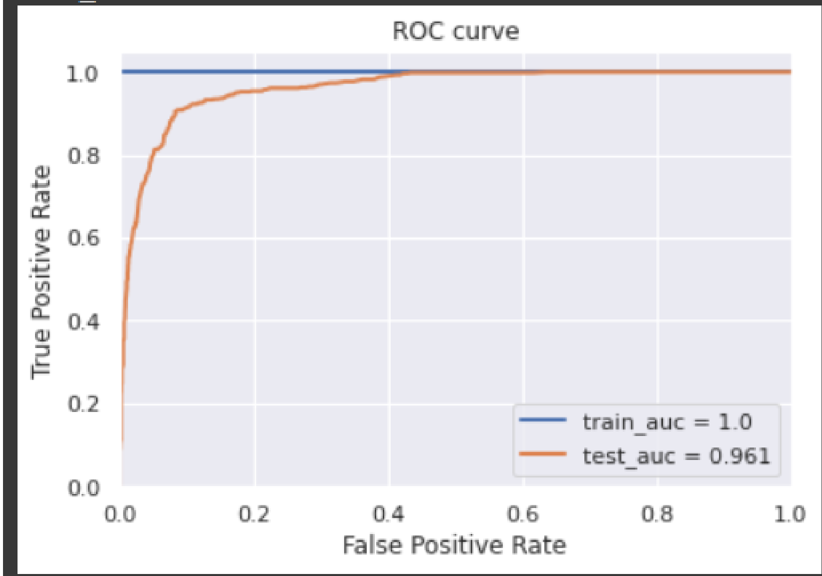
跑完超參數挑選後進去訓練

```
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()
best_para = Randomized_gridsearch(model, param_grid, x_pretrain_std, y_train)
```

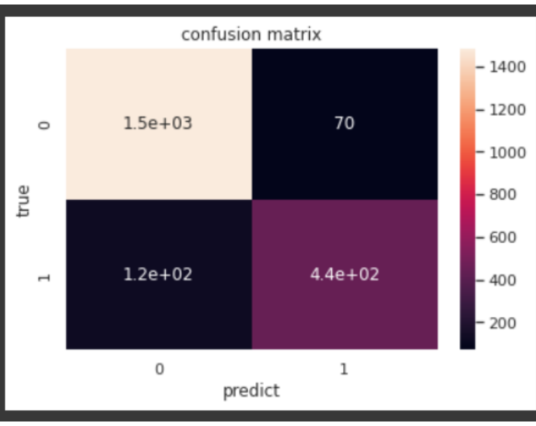
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 50, 'criterion': 'gini'}
```

- 結果

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	1.000	1.000	1.000	1.000	1.000	1.000
Test_set	0.909	0.955	0.781	0.761	0.864	0.923




```
TN_result:  [673 690 682 690 686]  TN_mean:  684
FP_result:  [52 35 43 35 38]  FP_result:  41
FN_result:  [65 48 56 50 64]  FN_result:  57
TP_result:  [200 216 208 214 201]  TP_result:  208
accuracy: 0.901010101010101
precision: 0.8353413654618473
sensitivity: 0.7849056603773585
specificity: 0.9434482758620689
MCC: 0.7432341123584031
```



- **With feature selection**

這次是使用 RFECV 去做進一步的特徵挑選。RFE 是將所有特徵灌入模型進行訓練後，刪除 step 個超參數的特徵，緊接著用剩餘特徵重新訓練一個新模型後再刪除 step 個特徵，如此反覆循環直到剩餘特徵數目等於我們的期望數字，而 RFECV 就是 RFE 結合 cross validation。

最後選出的特徵共 17 個

```
rf = RandomForestClassifier()
n_feature, selected = rfe(rf, x_pretrain_std, y_train)
```

```
Optimal number of features : 17
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 False False  True False False False False False False False False
 False False False  True False False False False False False False False
 False False False False False False False False False False False False
 False False  True False False False False False False False False False
 False  True False False False False False False False False False False
 False  True  True False False False False False False False False False
  True  True  True  True  True  True  True False False False  True  True
  True 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 False False False False False False False False False False
 False False False False False False False False False False False False
 False False]
Ranking of features : [103  90 113 109 112 117 140 129 127 150 133 145 124 122  52 130 142 121
  92 108 115 131 132 123  81  76  36  37  71  86  82 110 120 118 119  89
  96  35  24   1  31  23  34  28  63  55  51  32  70  20  45   1  73  64
  50 106  56  67  61  80 134  88  59  14   8  40  84 111 107  98 116  83
  54  16   1  15   3   7   9  29  25  30  48   2  22   1  60  46  33  97
  41  62  38  78 138  79  43   1   1  10  19  91 102 114  99 125  93  57
   1   1   1   1   1   1   1   6   4  12   1   1   1  66  39  18  94  49
  26  13  42 136  69 146 154 148 153  68  44  87 104 143 141 149 147   5
  17   1  47  53 139  85 101  74 144 151  27  65 135  95 152 105 126 128
 137  77  21  58  11  75  72 100]
```

接著做超參數挑選，因為第一次有超參數剛好選在範圍的邊界，因此又固定其他參數，在將在範圍邊界的超參數做擴展再做一次 Randomized gird search (min_sample_leaf 因為已經不能再小了所以就不調了)

```
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()
best_para = Randomized_gridsearch(model, param_grid, rf_train, y_train)
```

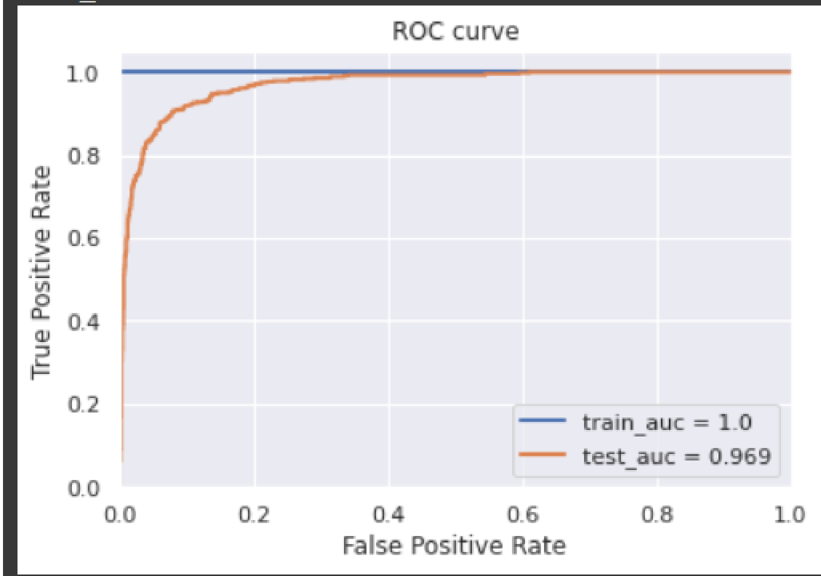
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 400, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_depth': 40, 'criterion': 'gini'}
```

```
param_grid = {
        'max_depth': [10, 20, 30, 40, 50],
        'criterion': ['gini'],
        'min_samples_leaf': [1],
        'min_samples_split': [4],
        'n_estimators': [350, 400, 500, 600]
}

model = RandomForestClassifier()
best_para = Randomized_gridsearch(model, param_grid, rf_train, y_train)
```

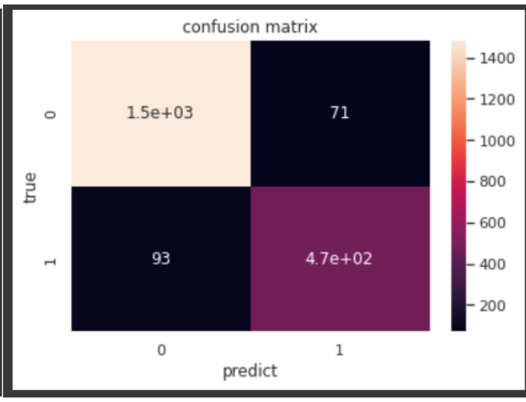
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 500, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_depth': 30, 'criterion': 'gini'}
```

- 結果

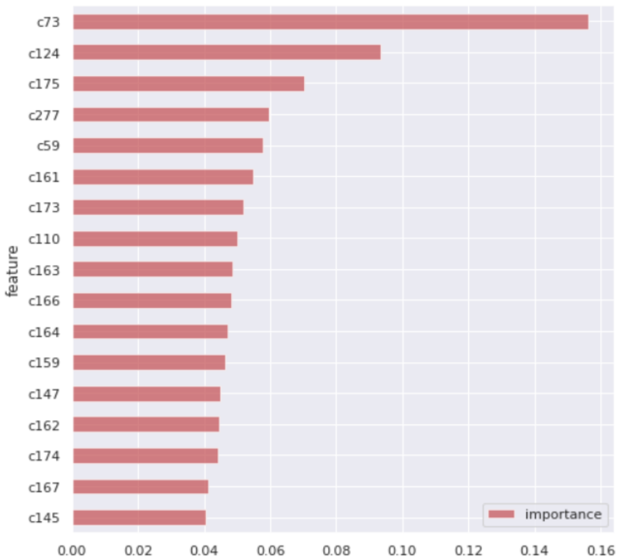
	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	1.000	1.000	1.000	1.0	1.00	1.000
Test_set	0.923	0.954	0.836	0.8	0.87	0.941




```
TN_result:  [688 694 684 692 694]  TN_mean:  690
FP_result:  [37 31 41 33 30]  FP_result:  34
FN_result:  [53 52 47 44 55]  FN_result:  50
TP_result:  [212 212 217 220 210]  TP_result:  214
accuracy: 0.9149797570850202
precision: 0.8629032258064516
sensitivity: 0.8106060606060606
specificity: 0.9530386740331491
MCC: 0.7793291266755875
```



最後也有用 RandomForest 的 feature importance 做特徵重要度排序



4. XGBoost

- **Statistic only**

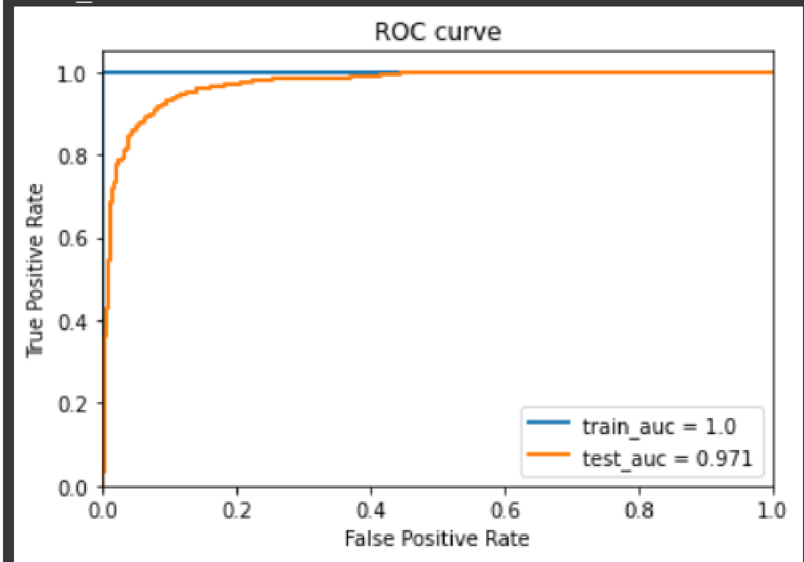
跑完超參數挑選後進去訓練

```
param_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()
best_para = Randomized_gridsearch(model, param_grid, x_pretrain_std, y_train)
```

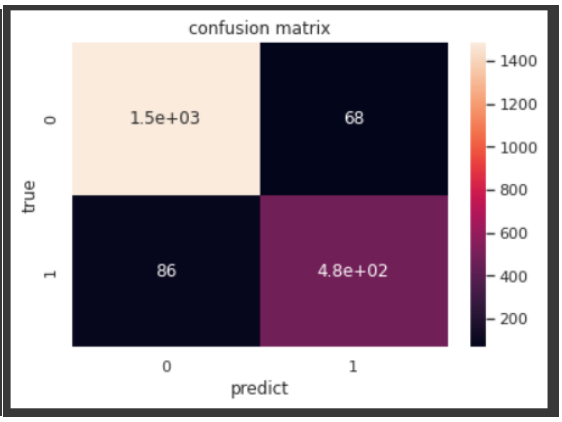
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 200, 'max_depth': 20, 'learning_rate': 0.3, 'gamma': 0.1}
```

- 結果

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	1.000	1.000	1.000	1.000	1.000	1.000
Test_set	0.927	0.956	0.848	0.813	0.876	0.945




```
TN_result: [685 690 685 692 693] TN_mean: 689
FP_result: [40 35 40 33 31] FP_result: 36
FN_result: [42 45 39 28 37] FN_result: 38
TP_result: [223 219 225 236 228] TP_result: 226
accuracy: 0.9251769464105156
precision: 0.8625954198473282
sensitivity: 0.8560606060606061
specificity: 0.9503448275862069
MCC: 0.8083632495739865
```



- **With feature selection**

和 RandomForest 一樣是使用 RFECV 做特徵挑選。最後挑出的共 77 個

```
xgb = XGBClassifier()
n_feature, selected = rfe(xgb, x_pretrain_std, y_train)
```

```
Optimal number of features : 77
Support is [False  True False False False  True False False False False  True False
 False False  True False False False False  True False False  True False
 False  True False False False False False False False False False  True
 False  True  True  True  True  True  True  True False  True  True  True
  True  True  True  True False  True  True False  True False  True False
 False False  True  True False  True False False False False False  True
  True  True  True  True  True  True  True  True False  True False False
 False  True  True  True False False False False False False False  True
  True False  True  True  True False False False False False False  True
 False  True  True  True  True  True  True  True  True  True  True  True
  True  True False False False False False  True False False  True False
 False False False  True  True  True False False False False False  True
  True  True False  True False  True False False False False  True  True
 False  True False False False False False  True  True False False  True
 False False]
Ranking of features : [55  1 38 28 16  1 59 44 50 75  1 79 81 82  1 41 89 24 67  1 70 73  1 47
 42  1 34 27 18 37 21 56 39 46 57  1 54  1  1  1  1  1  1  1 20  1  1  1
  1  1  1  1 43  1  1 58  1 30  1 74 45 40  1  1  4  1 92 64 94 63 72  1
  1  1  1  1  1  1  1  1 14  1 12  3 10  1  1  1  2 87 22  7 31 62 80  1
  1  6  1  1  1 78 48 26 76 51 52  1 11  1  1  1  1  1  1  1  1  1  1  1
  1  1 23 15 68 25  5  1 84 83  1 60 69 86 93  1  1  1 19 85 61 88 91  1
  1  1 32  1 90  1 17 29 77 71  1  1 36  1 65 49 53 13 66  1  1  8  9  1
 35 33]
```

接著做超參數挑選，因為第一次有超參數剛好選在範圍的邊界，因此又固定其他參數，在將在範圍邊界的超參數做擴展再做一次 Randomized gird search。

```
param_grid = {
        'n_estimators':[200, 300, 400, 500, 550],
        'max_depth': [5, 10, 20, 30, 40],
        'learning_rate': [0.0001, 0.005, 0.01, 0.1, 0.2, 0.3, 0.4],
        'gamma': [0.001, 0.05, 0.1, 1, 10]
}

model = XGBClassifier()
best_para = Randomized_gridsearch(model, param_grid, xgb_train, y_train)
```

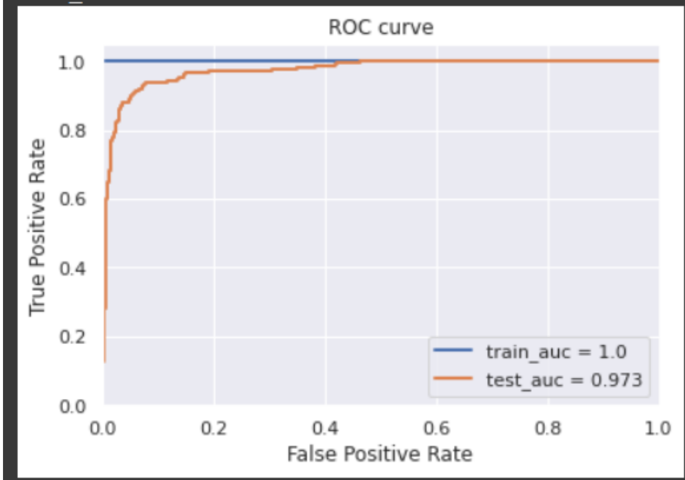
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 400, 'max_depth': 20, 'learning_rate': 0.2, 'gamma': 0.001}
```

```
param_grid = {
        'n_estimators':[400],
        'max_depth': [20],
        'learning_rate': [0.2],
        'gamma': [0.0001, 0.0005, 0.0008, 0.001, 0.05,]
}

model = XGBClassifier()
best_para = Randomized_gridsearch(model, param_grid, xgb_train, y_train)
```

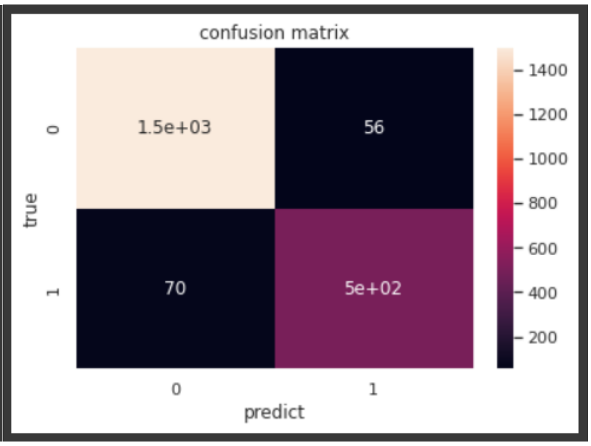
```
{'n_estimators': 400, 'max_depth': 20, 'learning_rate': 0.2, 'gamma': 0.001}
```

- 結果

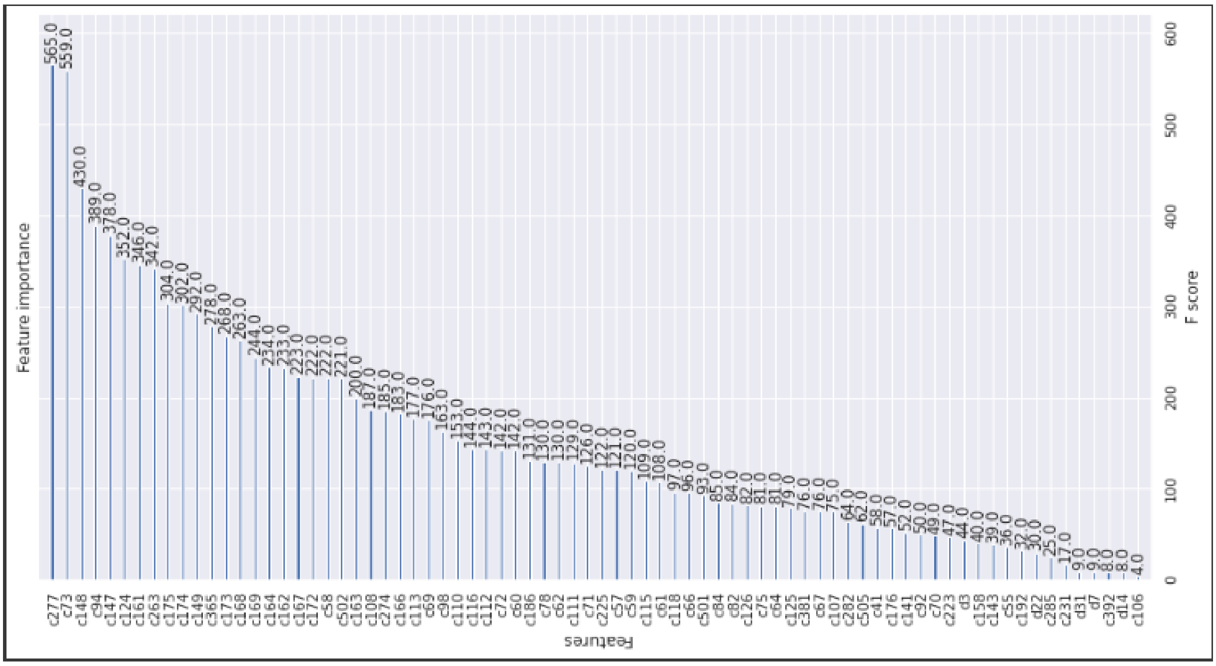
	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	1.000	1.000	1.000	1.000	1.000	1.000
Test_set	0.941	0.964	0.877	0.847	0.899	0.955




```
TN_result:  [697 692 689 694 698]  TN_mean:  694
FP_result:  [28 33 36 31 26]  FP_result:  31
FN_result:  [32 38 35 33 42]  FN_result:  36
TP_result:  [233 226 229 231 223]  TP_result:  228
accuracy: 0.9322548028311426
precision: 0.8803088803088803
sensitivity: 0.8636363636363636
specificity: 0.9572413793103448
MCC: 0.8259202895231342
```



最後也有用 XGBoost 的 feature importance 做特徵重要度排序



結論

由 4 種模型的實驗中所得到的 test AUC 結果:

	SVC	KNN	XGB	RF
Statistic only (170 feature)	0.902	0.877	0.971	0.961
進一步特徵挑選	0.893	0.874	0.973	0.969
進一步挑選的特徵數	166	136	77	17

在 SVC 以及 KNN 中由於是使用 permutation_importance 來做特徵挑選，且表現均比只有使用統計方法挑出來的還要不好一些，因此我認為可能原因是 permutation_importance 本身就不適合使用在高度多重共線性的數據中，因此表現會不如只使用統計方法來挑得好。而就 XGB 以及 RF 來看在有做特徵挑選後的特徵數不僅降低了，表現(test AUC、MCC、accuracy)也都有上升，因此我認為若是本身模型有 coef 或是 feature importance 的 attribute 可以在統計方法挑完後再做一次對於模型的特徵挑選，也許可以提升模型表現，且特徵也會使用得更少。

而就 test AUC、MCC、accuracy 最後結果也不意外的以 ensemble model 獲勝，而其中 XGB 更是勝過 RF，但是這是因為 SVC 以及 KNN 都只有使用一層的關係，若是可以使用更多的 SVC 以及 KNN 作為 base model 去做 ensemble model 也許結果又會不一樣。最後所選的最終模型為 XGBoost。

上傳模型名稱以及紀錄

- **模型名稱**: 109350008_HW2
- **對應訓練紀錄**:
  詳細的訓練紀錄在 **XGBoost 的 with feature selection** 中
  簡略的模型
  xgb_model = XGBClassifier(n_estimators = 400, max_depth = 20, learning_rate = 0.2, gamma = 0.001).fit(xgb_train, y_train)

加分

透過改變 threshold 的值，影響模型的預測結果，進而影響混淆矩陣（confusion matrix）中的 TP、FP、TN、FN 等數值，進而計算出不同的指標，如 Sensitivity、Specificity 等。

以下為示範 code:

```
model = XGBClassifier(**best_para).fit(xgb_train, y_train)
y_pred_prob = model.predict_proba(xgb_test)[:, 1]

thresholds = 0.8

y_pred = (y_pred_prob >= threshold).astype(int)
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
```

Package & function

```
#----------------| normal |----------------
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot

#--------------| processsing |--------------
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn import preprocessing
from scipy.stats import chi2_contingency
from scipy.stats import mannwhitneyu
import math

#----------------| model |----------------
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier

#-------| hyperparameter & feature selecting |-------
from sklearn.model_selection import RandomizedSearchCV
from sklearn.feature_selection import RFE, RFECV
from sklearn.inspection import permutation_importance
#--------------| evaluation |--------------
from xgboost import plot_importance
import seaborn as sns
from sklearn.metrics import accuracy_score,confusion_matrix
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
```

```
def rfe(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_
```

```python
def Randomized_gridsearch(model, param_grid_, x_train, y_train_):
    optimal_params = RandomizedSearchCV(
            model,
            param_grid_,
            cv = 5,
            scoring = 'accuracy',
            verbose = 1,
            n_jobs = -1,
        )

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

```python
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")
```

```python
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
```

```python
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]}
```

```python
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
```

```python
def pointer(model, x_train, y_train):
    cv_results = cross_validate(model, x_train, y_train, cv = 5, 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("TN_result: ", cv_results['test_tn'], " TN_mean: ", round(cv_results['test_tn'].mean()))
    print("FP_result: ",cv_results['test_fp']," FP_result: ", round(cv_results['test_fp'].mean()))
    print("FN_result: ",cv_results['test_fn']," FN_result: ", round(cv_results['test_fn'].mean()))
    print("TP_result: ",cv_results['test_tp']," TP_result: ", round(cv_results['test_tp'].mean()))
    print('accuracy:', accuracy)
    print('precision:', precision)
    print('sensitivity:', sensitivity)
    print('specificity:', specificity)
    print('MCC:', MCC)
```

```python
def confusion_matrix_(actual, predict):
    sns.set()
    f,ax = plt.subplots()
    cm = confusion_matrix(actual, predict)
    sns.heatmap(cm, annot=True, ax = ax)

    ax.set_title('confusion matrix')
    ax.set_xlabel('predict')
    ax.set_ylabel('true')
```

```
def  std(x_train):
    Std  =  preprocessing.StandardScaler().fit(x_train)
    x_train_Std  =  Std.transform(x_train)

    return  x_train_Std
```