於血液透析前施打 EPO，預測 4 週後是否符合 HGB 值為 11~12g/dL 及 HGB 變化量為-0.5~0.5 之間

學生: 張詠哲　　授課教授:何信瑩

一. 摘要 Abstract

血液透析是一種治療腎功能不全的方法，它通過機器來過濾血液中的廢物和多餘的液體，以維持身體的正常功能。而紅血球生成素(EPO)可以刺激骨髓產生紅血球，從而提高血液中的血紅蛋白(Hb)水平。

在透析病人中，由於腎臟功能受損，通常會導致貧血。因此，為了提高透析病人的 Hb 水平，醫生會透過施打 EPO 來刺激紅血球生成。然而 EPO 價格高昂，過量的 EPO 使用也可能導致一些副作用，如高血壓、血栓等，因此掌握好 EPO 的施打劑量不僅是經濟價值考量也有安全考量。所以醫生需要追蹤透析病人的 Hb 水平，以確保病人的貧血得到適當的治療，同時也要使 EPO 可以使用得精準。

本篇研究目的是想要透過一些機器學習的方法來預測腎臟病患者治療貧血的雙目標：

1. HGB 值維持在 11~12 g/dL (血紅蛋白濃度充足)
2. HGB 值和上次相比變化為 -0.5~0.5 g/dL (變化小)

所採用的方式是先利用統計配上基因演算法或是 RFECV 的方式找出較好的一組特徵，其中使用的分類模型包括: Random Forest(RF)、SVM、XGBoost 等等。其中因為 RF 和 XGBoost，SVM 的部分也會使用 Bagging 的方式做成集成模型(estimator = 100)來評估。最後再評估哪個模型在表現最好的前提下所用的特徵最少。

最後的結果為統計方法 + RFECV + RF 的結果最好。test auc = 0.975、test MCC = 0.814、test acc = 0.939

因為有些程式碼比較大，因此怕占版面，所以這篇報告只會呈現較小的程式碼，整個程式碼會附在修課心得後面的 github 連結裡面。

二. 引言 Introduction

血液透析（Hemodialysis）是將血液抽出體外，經過血液透析機的滲透膜，清除血液中的新陳代謝廢物和雜質後，再將已淨化的血液輸送回體內。血液透析可用於腎衰竭、腎臟病患或血液中毒等身體無法自行排出毒害物質的情況。

而 EPO 紅血球生成素是一種 Hormone，可以控制紅血球生成，或紅血球的產生，大部份的紅血球生成素是經由腎臟製造，而腎衰竭或是腎臟病的患者會因為腎功能不完全，腎臟中製造紅血球生成素的細胞受損，導致其不能正常製造紅血球生成素，因此容易產生貧血。而透過施打 EPO 可以刺激骨髓產生紅血球，從而提高血液中的血紅蛋白

(Hb)水平，來達到預防貧血的效果。但是 EPO 的價格很高，且過量的 EPO 使用也會導致紅血球太多、Hb 濃度太高，導致一些副作用，例如高血壓、血栓等，有可能會因此合併中風之類的風險。而施打不夠又會導致貧血沒有得到很好的改善。一般施打 EPO 的患者大約會在 2~4 周觀察一次 Hb 的狀況，因此若是可以在施打前預測患者 4 周後的 Hb 情況，就可以看說這次 EPO 大概要施打多少。

而機器學習是人工智慧(Artificial Intelligence, AI)的一個分支，用於檢驗數據，透過過往數據與資料找尋其中規則，建立系統以幫助預測未來發生事件的機率。近年來，AI 應用於醫療行業的發展熱度不斷提升，致力於預測疾病風險、發病率等。機器學習即是訓練電腦從資料中學習，透過演算法找出大量資料中的關聯性，並隨著經驗累積而進行改善，最後得以分析出最佳決策或預測。

因此本篇的研究目標是想要使用一些機器學習以及統計的方式去預測腎臟病患者在施打 EPO 後，4 周的 1)HGB 值維持在 11~12 g/dL (血紅蛋白濃度充足)以及 2)HGB 值和上次相比變化為 -0.5~0.5 g/dL (變化小)。

三. 資料集 Dataset

此次的資料集如下:

- 樣本數: 7084
- 欄位:
  - 離散型: 41 個(d1~d39、d499、d500)
  - 連續型: 477 個(c40~c498、c501~c516)
- Label: ΔHGB (符合 label 為 1，不符合 label 為 0)

  -> 即每個樣本 28 天後的 HGB 值減去此樣本當次 HGB 值，若此樣本的 28 天後找不到 HGB 值，則再往後 7 天內找看看有沒有，如果還是沒有則將此樣本丟掉不用，也就是每個樣本的 35 天後仍找不到 HGB 值則丟掉不用。

四. 方法 Method

先看看資料 label 的分布狀態，發現到 0 和 1 的分布有些不平衡，0 : 1 ≈ 3.31 : 1，因此為了保留每個類別的樣本百分比，採用 StratifiedShuffleSplit 並且以 7:3 的方式切分，接著特徵選擇方面分為 3 個實驗:

baseline: 不調超參數，也不做特徵挑選

實驗一: 單純用統計方式(離散型用 Chi–Square test、連續型用 Mann Whitney U test)，把 p-value < 0.05 的特徵挑出來

實驗二: 統計方式 + Simple genetic algorithm(SGA)

實驗三: 統計方法 + RFECV

而模型方面會從 sklearn 中選出幾個分類用的模型，例如:Random Forest、XGBoost、SVC 之類的作為使用模型，其中因為 RF 和 XGBoost 皆為集成模型，因此為了公平比

較，SVC 的部分也會使用 Bagging 的方式做成集成模型(estimator = 100)來當作比較模型(但因為 SVC 本身複雜度大($O(N^3)$) 因此在做 SGA、RFECV 或是 BO 時只會單純使用一個 SVC 來做挑選) 。而超參數挑選會使用 Bayesian Optimization (BO)，其迭代次數設定為 300。SGA、RFECV、BO 的評分標準設定為 5-fold cross validation、scoring = roc_auc。之後再從這些實驗中找出表現最好以及使用特徵最小的特徵(考量的標準是由 Test 的 AUC、Accuracy、MCC，以及所使用 feature number 決定，但主要還是以 AUC 為評斷標準)。最後順便從表現最好的模型中找出各個特徵的重要程度。

因為 RFECV 以及 BO 是上課中沒有提到的因此以下為 2 者的簡單介紹:

1. RFECV 是 Recursive Feature Elimination + Cross validation。他會依據模型提供的 coef 以及 feature importance 的 attribute 來遞迴地從資料集中刪除特徵。其中會使用 cross validation 來評估每個特徵子集的性能。以下是 RFECV 的流程圖

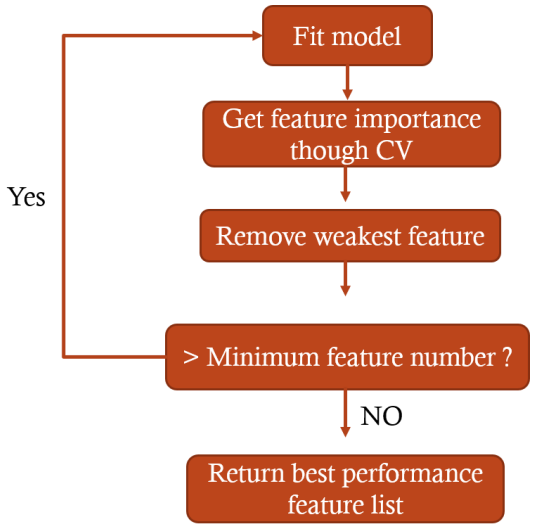


Figure 1. RFECV 演算法流程

2. Bayesian Optimization (BO) 貝葉斯優化是一種 black box 優化算法，用於求解表達式未知的函數的極值問題。它會根據一組採樣點處的函數值預測出任意點處函數值的概率分佈，這通過 Gaussian Process Regression 以及 Acquisition Function (AF)來實現。用於計算每一個點值得探索的程度，求 Acquisition Function 的極值從來確定下一個採樣點。最後返回這組採樣點的極值作為函數的極值。

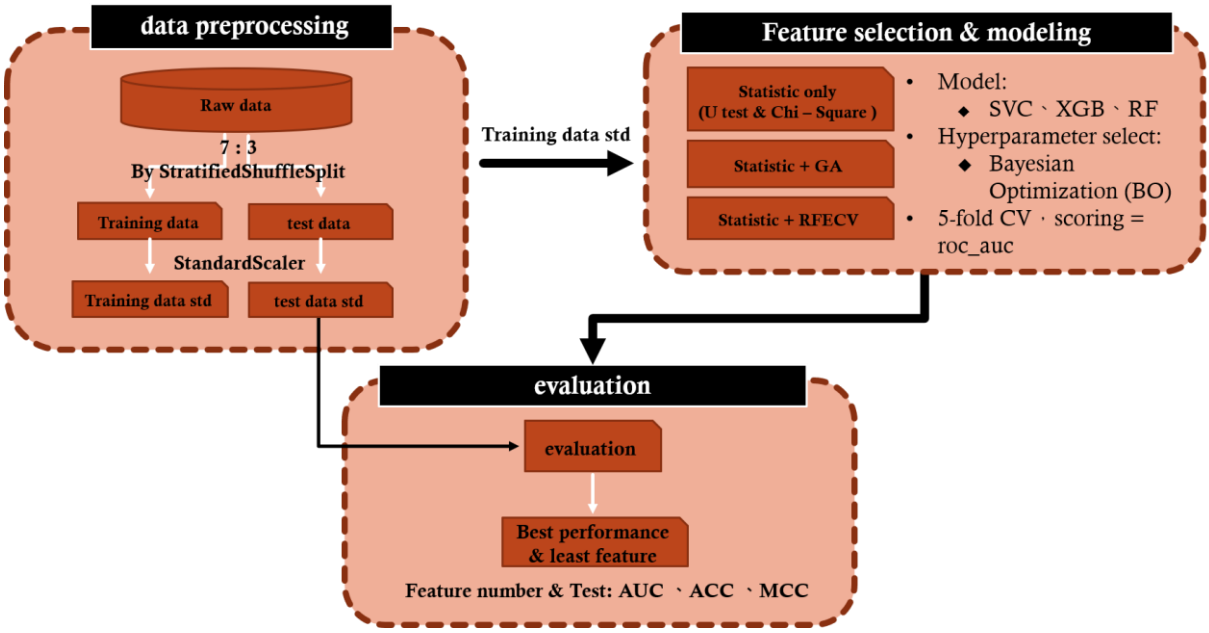


Figure 2. 方法流程圖

五. 結果 Result

5.1 Baseline

Baseline 的部分是不做超參數挑選以及特徵挑選，但是也是會把資料標準化。

```
feature  =  x_train.columns
x_rdt_std  =  std(x_train)
x_rds_std  =  std(x_test)
```

```
x_rdt_std  =  pd.DataFrame(x_rdt_std)
x_rds_std  =  pd.DataFrame(x_rds_std)

x_rdt_std.columns  =  feature
x_rds_std.columns  =  feature
```

Figure 3. baseline 擷取資料示意圖

而以下是跑完 3 個模型的指標以及圖。

5.1.1 SVC

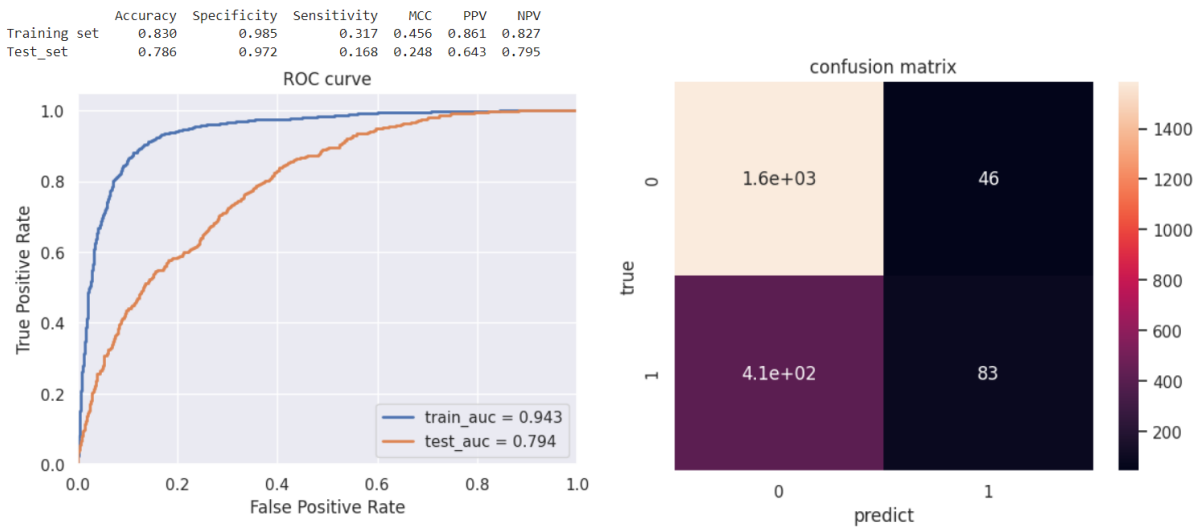


Figure 4. baseline SVC result

5.1.2 XGB

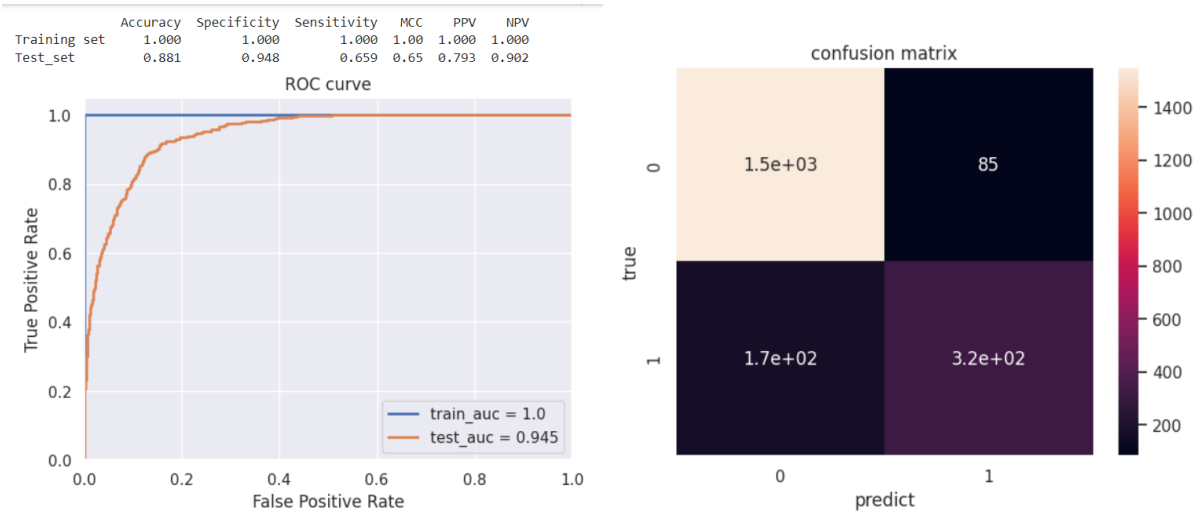


Figure 5. baseline XGB result

5.1.3 RF

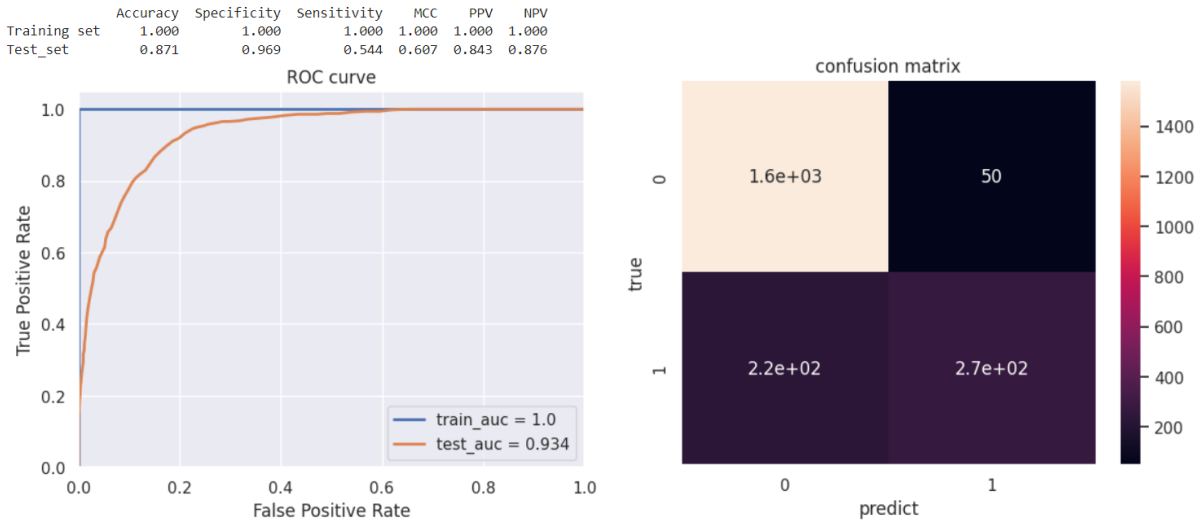


Figure 6. baseline RF result

可以看到說除了 SVC 的 test AUC = 0.794 比較低之外，其餘 XGB 和 RF 都有道 0.94 左右，還不錯。但是應該可以再做更好。

5.2 實驗一: Statistic only

連續型特徵用 Mann Whitney U test 挑出 p-value < 0.05 的特徵，離散型特徵用 Chi-Square test 挑出 p-value < 0.05 的特徵，挑出來共 166 個。之後利用 StandardScaler 標準化做成標準化後的訓練集以及測試集。

```
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])
```

```
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
```

Figure 7. 統計及標準化程式碼

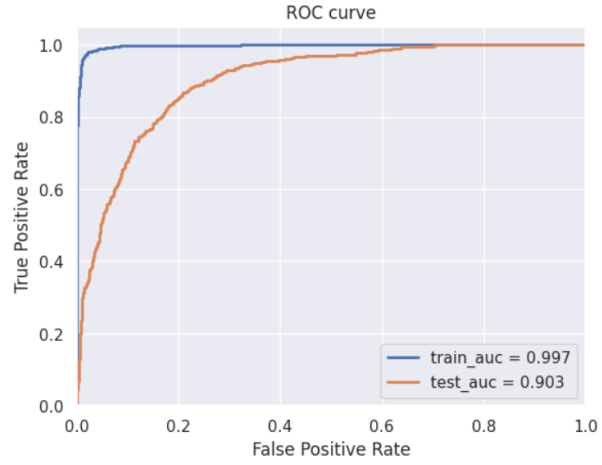
而以下是跑完 3 個模型的指標以及圖。

5.2.1 SVC

```
svc_optimizer.max
```

```
{'target': 0.8935633696390705,
 'params': {'C': 32.13162009624968, 'gamma': 0.0078125}}
```

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.982	0.991	0.953	0.950	0.970	0.986
Test_set	0.859	0.918	0.663	0.595	0.709	0.900



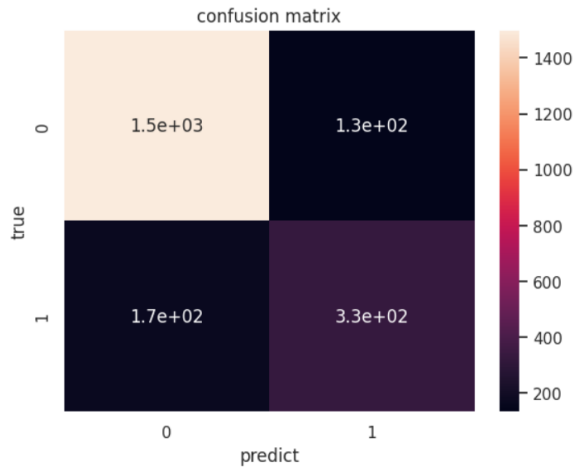


Figure 8. Statistic only SVC with BO result

5.2.2 XGB

```
xgb_optimizer.max
```

```
{'target': 0.9621020228553245,
 'params': {'gamma': 0.6894903888318535,
  'learning_rate': 0.2749218496878777,
  'max_depth': 47.86651040794286,
  'n_estimators': 124.90230343161514}}
```

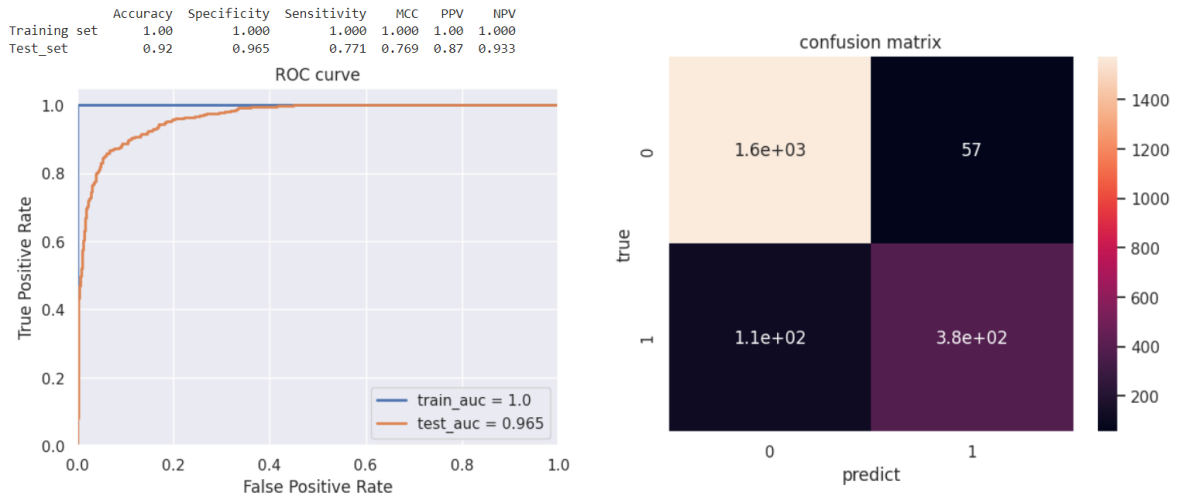


Figure 9. Statistic only XGB with BO result

5.2.3 RF

```
rf_optimizer.max
```

```
{'target': 0.955897171283573,
 'params': {'max_depth': 48.70429606621174,
  'min_samples_leaf': 1.0,
  'min_samples_split': 2.0,
  'n_estimators': 419.4386305002588}}
```

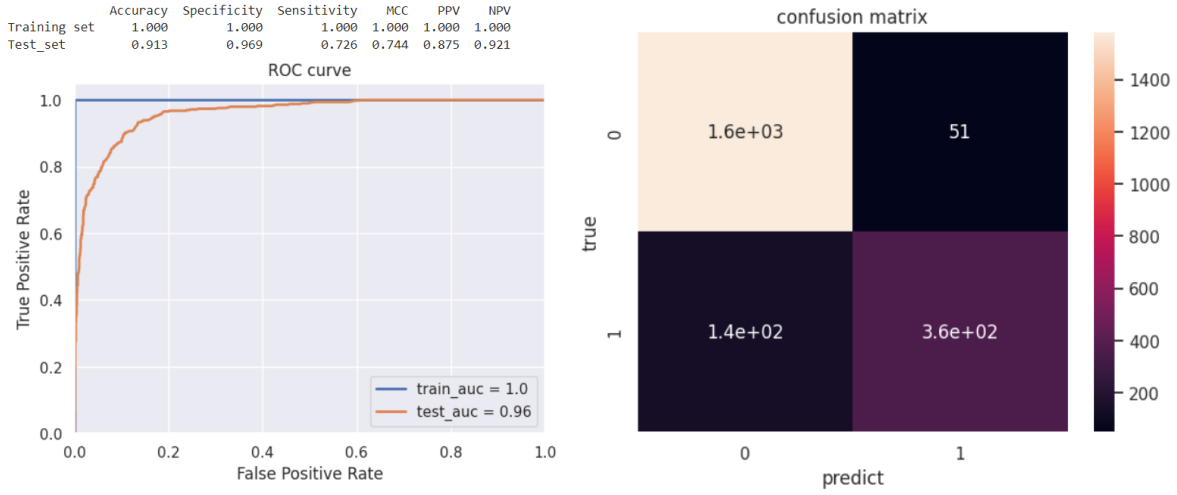


Figure 10. Statistic only RF with BO result

由上面的結果圖可以看出，經過統計方法挑出一些和 label 相關的特徵後，不僅使特徵數下降，所有指標也都有提升。SVC 也來到了 test AUC 為 0.9 左右，相較於 baseline 的 0.79 上升很多，XGB 以及 RF 也都有明顯提升，到了 0.96 左右。

5.3 實驗二: Statistic + SGA

接著是實驗統計方式初篩完後，再使用基因演算法進一步挑選特徵。這部分是利用實驗一所產生的特徵再去跑基因演算法。基因演算法的子代數是設定 50、交配率 =0.5、突變率 =0.3。

而以下是跑完 3 個模型的指標以及圖。

5.3.1 SVC

因為若是用 bagging SVC 去跑的話會非常久，超過 colab 使用時間，因此這部分只使用一個 SVC 去做 GA，最後評估時才會再利用挑出來的特徵做 bagging。最後是跑出了 26 個特徵。

```
svc_optimizer.max
```

```
{'target': 0.931722684810088,
 'params': {'C': 53.340672217626185, 'gamma': 1.469174997511715}}
```

	Accuracy	Specificity	Sensitivity	MCC	PPV	NPV
Training set	0.998	0.999	0.994	0.994	0.997	0.998
Test_set	0.924	0.968	0.781	0.781	0.879	0.936

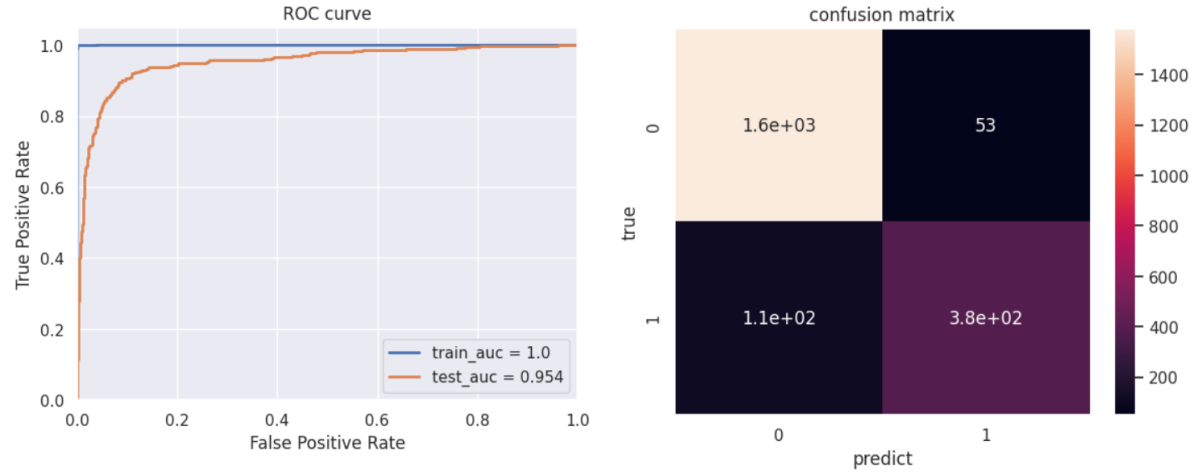


Figure 11. Statistic + GA SVC with BO result

5.3.2 XGB

特徵數跑出了 51 個，而 BO 挑出的超參數結果: gamma = 0.01562 、 learning rate = 0.1328、 max_depth = 60 、 n_estimator = 398。

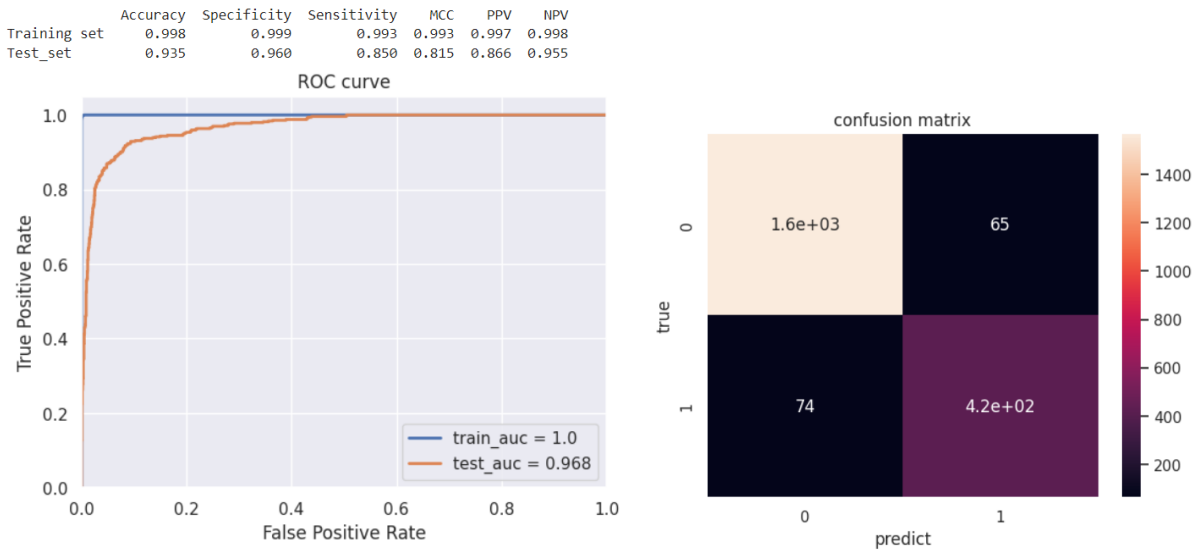


Figure 12. Statistic + GA XGB with BO result

5.3.3 RF

特徵數跑出了 35 個。

```
rf_optimizer.max
```

```
{'target': 0.9606646929517744,
 'params': {'max_depth': 59.07218871119701,
  'min_samples_leaf': 1.0,
  'min_samples_split': 2.0,
  'n_estimators': 256.77836666489344}}
```

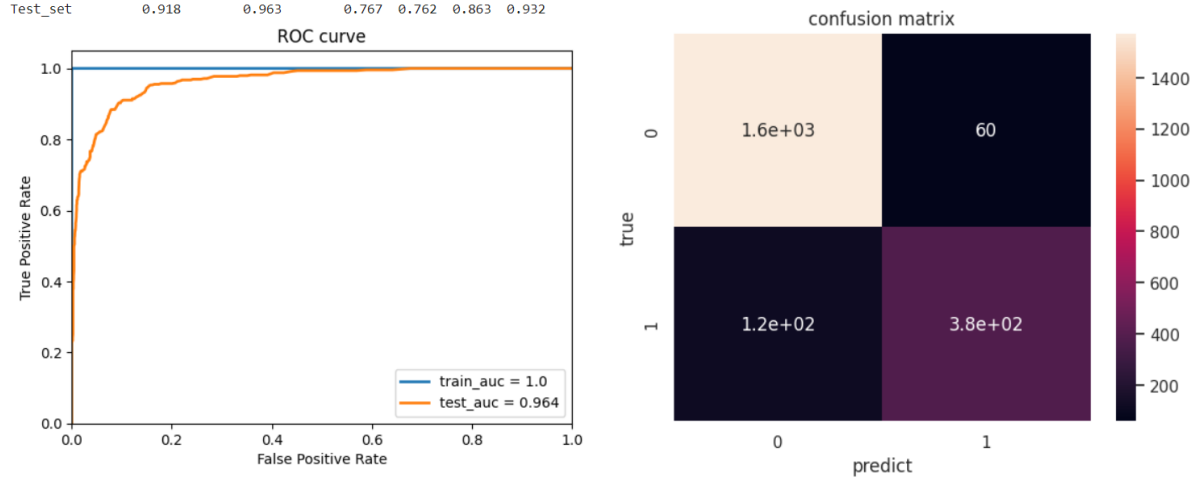


Figure 13. Statistic + GA RF with BO result

5.4 實驗三: Statistic + RFECV

這部分和 GA 相同，也是利用實驗一挑出的特徵再使用 RFECV。

```
def rfe(model, x_train, y_train):
    rfecv = RFECV(estimator = model, step = 1, cv = 5, scoring='roc_auc').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_
```

Figure 14. RFECV 程式碼

5.4.1 SVC

因為 RFECV 是需要傳入帶有 coef 或是 feature importance attribute 的模型，但是當 SVC 的 kernel = rbf 時，是沒有這 2 的 attribute。因此替代方法是利用 permutation importance 的 package 算出 SVC 特徵的特徵重要度後，接著把重要度為 0 的去掉，再自己實作一個 RFECV。而 **Figure 16** 是順手做了依照重要度剃除特徵後每個特徵數下的 5-fold CV AUC 表現圖。最後挑出的特徵數為 25 個 、 BO 所挑出的超參數: C = 218.3 、 gamma = 12.79。

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

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

```
score = []
best_threshold = 0
best_score = 0

for i in threshold.iloc[:,0]:
    temp_feature = svc_prefeature.loc[:, importance[importance["importances_mean"] >= i].index]
    temp_svc = cuSVC(kernel = "rbf").fit(temp_feature, y_train)
    mean_score = cross_val_score(temp_svc, temp_feature, y_train, cv = 5, scoring = "roc_auc").mean()
    score.append(mean_score)
    if(mean_score > best_score):
        best_score = mean_score
        best_threshold = i

max = len(threshold)
score.reverse()
print("best_threshold: ", best_threshold)
print("best_score: ", best_score)
plt.figure(figsize = (8, 5))
plt.plot(range(1, max+1), score)
plt.xticks(range(1, max+1, 9))
plt.show()
```

Figure 15. 實作類似 RFECV 的程式碼

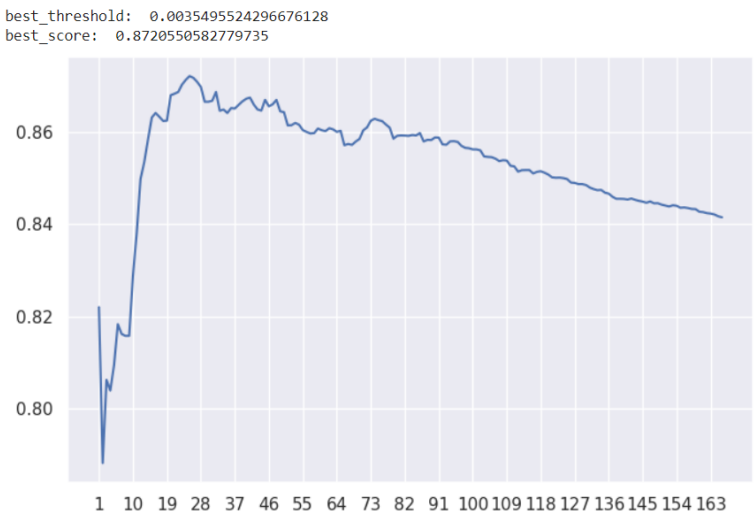


Figure 16. SVC 個特徵數的 5-fold CV AUC 表現圖

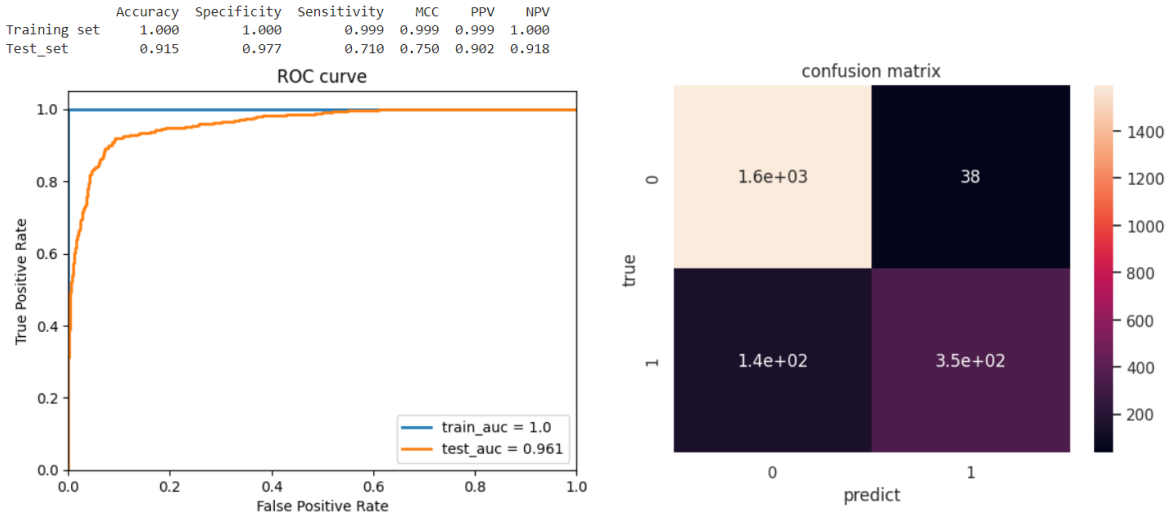


Figure 17. Statistic + RFECV SVC result

5.4.2 XGB

特徵數跑出了 70 個，BO 挑出的超參數結果: gamma = 0.03434 、 learning rate = 0.1396、 max_depth = 23 、 n_estimator = 323。

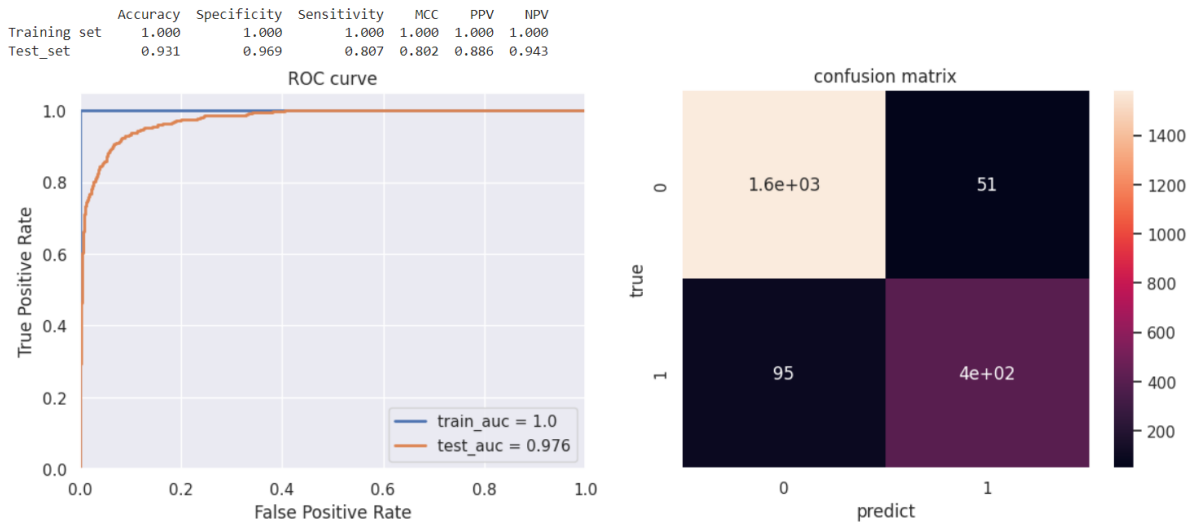


Figure 18. Statistic + RFECV XGB result

5.4.3 RF

特徵數跑出了 15 個，BO 挑出的超參數結果: max_depth = 39、min_samples_leaf = 1 、 min_sample_ split = 2 、 n_estimator = 406。

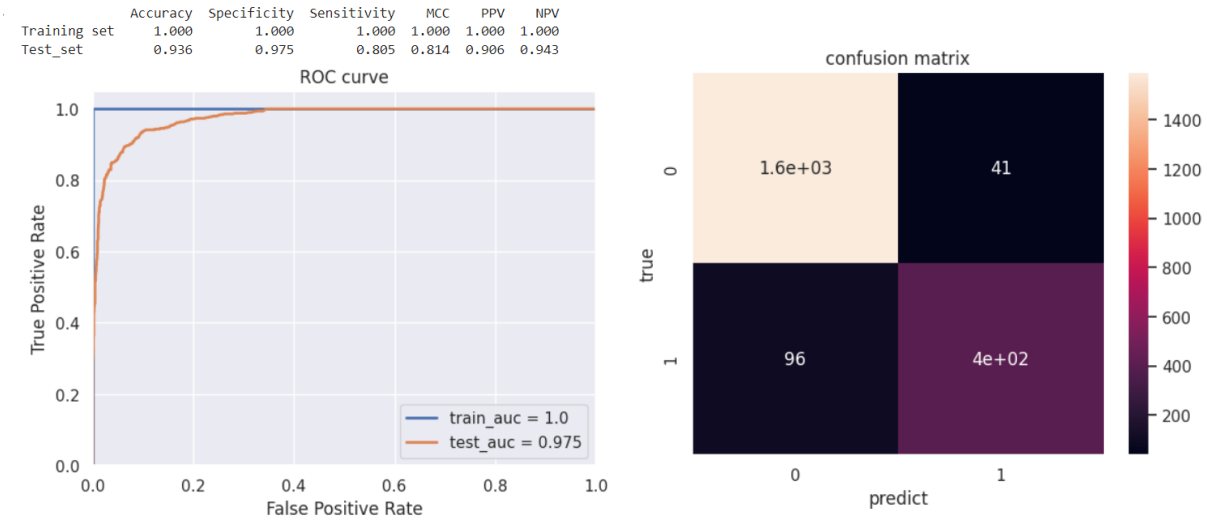


Figure 19. Statistic + RFECV RF result

5.5 Result Overview

由於上面結果圖的各指標比較不明顯，因此以下為統整上面的指標以及特徵數總表。但因為主要會考量的指標只有 AUC、accuracy、MCC，因此只有統整這 3 個以及特徵數。圖表上面顏色較淺的為 training set 的結果，下面橘色的是 test 的結果。

SVC (bagging · base estimator 100)	Feature number	Accuracy	AUC	MCC
baseline	516	0.830	0.943	0.456
Statistic only	166	0.982	0.997	0.950
Statistic + GA	26	0.998	1.0	0.994
Statistic + RFECV	25	1.0	1.0	0.999
baseline	516	0.786	0.794	0.248
Statistic only	166	0.859	0.903	0.595
Statistic + GA	26	0.924	0.954	0.781
Statistic + RFECV	25	0.915	0.961	0.750

表格 1. SVC result overview

XGB	Feature number	Accuracy	AUC	MCC
baseline	516	1.0	1.0	1.0
Statistic only	166	1.0	1.0	1.0
Statistic + GA	51	0.998	1.0	0.993
Statistic + RFECV	70	1.0	1.0	1.0
baseline	516	0.881	0.945	0.650
Statistic only	166	0.920	0.965	0.769
Statistic + GA	51	0.935	0.968	0.866
Statistic + RFECV	70	0.931	0.976	0.802

表格 2. XGB result overview

RF	Feature number	Accuracy	AUC	MCC
baseline	516	1.0	1.0	1.0
Statistic only	166	1.0	1.0	1.0
Statistic + GA	35	1.0	1.0	0.999
Statistic + RFECV	15	1.0	1.0	1.0
baseline	516	0.871	0.934	0.607
Statistic only	166	0.913	0.960	0.744
Statistic + GA	35	0.918	0.964	0.762
Statistic + RFECV	15	0.936	0.975	0.814

表格 3. RF result overview

六. 討論 Discussion

由結果的 3 個總整表可看出，隨著特徵挑選過後各模型的表現都有提升。其中提升幅度最大的是 SVC，從原本 baseline 的 test AUC 為 0.794 提升到 statistic + RFECV 的 0.961。而 XGB 以及 RF 也從 baseline 的 0.94 左右提升到 0.97 多，且特徵數都有減少。雖然就 training set 以及 test set 的 accuracy 以及 MCC 這兩個指標相差較大(大約差了 7% 以及 20%)看下來有一些 overtraining 的現象，但是 AUC 相差大約只有 1.5%左右，我認為是因為我在做挑超參數以及特徵挑選時，BO、GA、RFECV 所用的 scoring 都為 AUC 為主造成的，所以我覺得是可以接受的且可理解的結果。

表格 4 為各模型最好的 test AUC 結果以及其挑選特徵的方式:

首先可以看到 test AUC 最好的結果都為統計方法 + RFECV 挑出來的，但是這不代表 RFECV 就勝過 GA。我認為可能的原因是因為我的 GA 子代數設定的不夠，只有迭代 50 次而已，而且 SGA 不會像 RFECV 一樣會把不重要的特徵去掉，它比較隨機一點，單純把表現好的特徵集留下來繼續演化。也許迭代數再多一些也會收斂到最佳解，只是它有可能會收斂到 local optimization。但是 GA 相較於 RFECV 對於大型特徵空間和多目標優化問題就擁有較強的搜索能力和靈活性 (可以自己調突變率或是交配率)。而 RFECV 若是特徵間有較強的共線性則容易 overtraining。因此若是可以結合兩種特徵挑選的特性也許可以有更好的表現。

最後比較可看出即便 SVC 也同樣使用 bagging 做成集成模型，但表現依舊略輸 RF 以及 XGB，我認為有可能是因為我 SVC 在 bagging 時只固定 base estimator 為 100 個(再多會跑很久)，因此才造成 SVC 的結果略輸 RF 以及 XGB，也許把 estimator 的數量也當作超參數挑選會造成不一樣的結果。但就目前的結果表現為XGB ≈ RF ＞ SVC。 XGB 的 AUC 略勝 RF 0.1%，但是 RF 的 accuracy 以及 MCC 表現都好於 XGB。

	Feature selection	Feature num	Accuracy	AUC	MCC
SVC (bagging，base estimator 100)	Statistic + RFECV	25	0.915	0.961	0.750
XGB	Statistic + RFECV	70	0.931	0.976	0.802
RF	Statistic + RFECV	15	0.936	0.975	0.814

表格 4. Test score (best auc of every model)

因此我再對這兩個模型 (RF & XGB)做一次 test set 的 boostrap resample，即針對 test set 做可放回的重複抽樣，直到 sample 數和原始的 test set 一樣多，接著測表現，去模擬面對其他未知資料集時的表現。這樣迭代 10000 次，大致得出這兩個模型指標的平均，中位數以及 95%信心區間，如表格 5。由表格 5 可看出兩個模型的 AUC 表現一樣，但是 RF 的 accuracy 、 MCC 都略高於 XGB。因此也許這表示在面對未知的數據集時這個 RF 模型表現會略好於這次訓練的 XGB 模型。

Mean / (95% CI)	Feature num	Accuracy	AUC	MCC
XGB	70	0.931 / (0.921, 0.943)	0.975 / (0.969, 0.981)	0.802 / (0.774, 0.835)
RF	15	0.939 / (0.929, 0.949)	0.975 / (0.969, 0.981)	0.814 / (0.794, 0.852)

表格 5. Test set boostrap resampling score (iter = 10000 times)

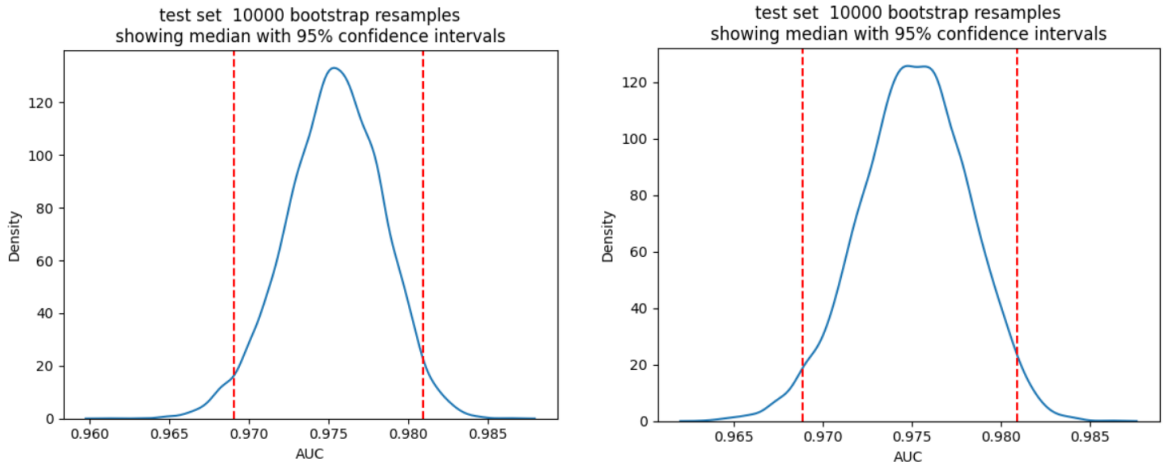


Figure 20. RF(left) & XGB(right) boostrap resample AUC result

最後我也看了一下 RF 選出來的 15 個特徵的重要度，可以看到只有連續型的特徵被選出來，沒有半個離散型的特徵，而其中最重要的特徵是 c59。

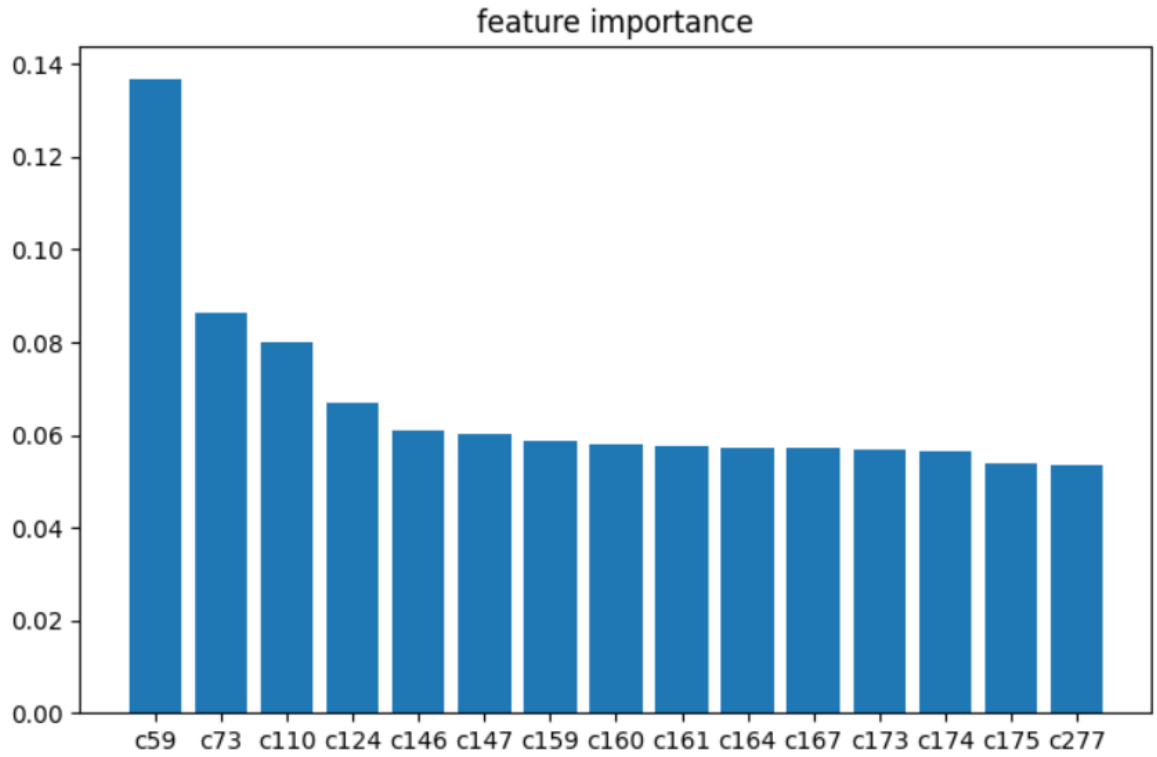


Figure 21. Feature importance of RF's features

七. 參考資料 Reference

https://deap.readthedocs.io/en/master/
https://github.com/bayesian-optimization/BayesianOptimization
https://scikit-learn.org/stable/