Identification of the Recurrence-Related lncRNAs to Predict Survival Proportion in Gastric Cancer with TCGA dataset

學生: 張詠哲 指導教授: 何信瑩

一. 摘要 Abstract

胃癌(gastric cancer)是目前第五大常見癌症,其造成原因除了和先天上的基因遺傳有關,也和後天的作息有很大的關係,像是飲食、吸菸、肥胖。其五年存活率僅有 3 成, 且死亡率為所有癌症的第二高,而存活率低常常伴隨著轉移至其他器官有關,而且手術後兩年內復發率也高達 6 至 7 成。

由於胃癌的高死亡率以及高復發率,因此本文研究目的是想要透過 lncRN 來找到一組復發相關生物標誌物來預測胃癌患者存活率,也能透過這組生物標誌物來預防胃癌復發的發生。

我利用的方法是先利用 ranksum-test 將 p-value 小於 0.01 的特徵(lncRNA)挑出。接下來我嘗試了兩種不同的方式再細挑出更少的特徵且可以使 c-index 表現不錯,實驗一是利用存活模型的 coef 或是 feature importance,實驗二是利用回歸模型挑特徵再去建存活模型。使用的存活模型有:CoxPH、GradientBoostingSurvival(GBS)、ComponentwiseGradientBoostingSurvival(CGBS)、RandomSurvivalForest(RSF),而回歸模型使用了:LassoCV、XGBRegression、GradientBoostingRegression。再將 cross validation 分數最高的挑出做超參數挑選再利用 test 來看有沒有 overfitting。最後再以最終模型所挑出的 feature 做 KM 圖以及 logrank test。延伸的部分則有再另外加了一些臨床欄位去看可不可以增加準確率,也有用做了病患間的存活機率函數。

結果顯示皆為實驗一的表現最好,其中 CGBS 在 cross validation 的 c-index 為 0.592,雖然不是實驗一中最好的,但是在 test 的 c-index 為 0.647,沒有表現出 overfitting 的現象,因此選擇它以及其選擇的特徵 7 個特徵,依照 feature importance 排序分別為:ENSG00000254622.1、ENSG00000237975.7、ENSG00000248469.1、ENSG00000223466.2、ENSG00000262061.6、ENSG00000224758.1、ENSG00000229628.1,而在對存活的 logrank test 的 p-value 中,前 4 個較有顯著的差異,依序為:0.000309、0.012591、0.027793、0.025205。

但是在助教給的 independent test 的資料集中預測效果卻不是很好,c-index 以及 c-ipcw 分別為 0.515 以及 0.479。

二. 引言 Introduction

胃癌（gastric cancer, GC）是發生在胃部黏膜的癌症。是全世界中排名第四個最普遍被診斷的癌症,而且是所有癌症死亡率排名第二高,手術後兩年內復發率也高達6至7成。早期的症狀包括熱,上腹疼痛、噁心及食慾不振。症狀與消化性潰瘍類似,是導致延誤就

醫與高死亡率的原因,其痛感是一種咬合性疼痛並且制酸劑不能解除痛感,然而還是有較少數胃癌痛感會受進食些許影響並且制酸劑能稍微緩解,導致診斷更難,目前胃鏡是唯一有效直接的診斷方法。而最常見的輔助原因是因為感染幽門螺桿菌(Helicobacter pylori),將近六成的胃部腫瘤患者都檢查出感染這種細菌,某些種的幽門螺旋桿菌較為其他種來的危險。晚期的症狀包括體重減輕、皮膚及眼白泛黃、嘔吐、難以吞嚥、便血等症狀, 且晚期的 GC 患者在復發以及轉移的預後很差,容易切除後復發或是轉移。治療前的評估可能有助於識別高復發的 GC 患者,也許可以為此來制定標靶治療以降低死亡率以及復發率, 因此預後指標對於 GC 患者很重要。

lncRNA 已被證明在 GC 中失調,通過與表達異常的 lncRNA 通過與 DNA、RNA、蛋白質互相交互作用參與 GC 的發生和發展,從而影響其患者的生存率,因此 lncRNA 也許可以作為預後的生物標誌物(biomarker)。因此本文是想利用 TCGA(The Cancer Genome Altas)的資料集並且利用統計、機器學習(machine learning)、存活分析等方法去找一組和復發有相關的 lncRNA 可以用來預測患者的存活率, 也能透過這組 lncRNA 來預防胃癌復發的發生。之後再進一步的使用一些臨床資料來做到個人化的模型也可以預測患者的個人存活機率曲線。

三. 資料集 Dataset

資料集有兩個部分, 分為臨床資料以及 lncRNA 加上復發以及存活的資料,詳細資料如下:

1. RNA-seq expression data(TPM-scaled)
    a. 樣本數：247
    b. lncRNA 數量(特徵數)：16304
    c. label :
        I. Overall Survival Status:
            i. 0 為存活 : 142
            ii. 1 為死亡: 105
        II. Overall Survival time

2. Clinical data
    a. 樣本數：440
    b. 復發欄位:
        I. DiseaseFree: 212(label 為 0)
        II. Recurred/Progressed: 46(label 為 1)
        III. 未紀錄:103(label 為-1)

四. 方法 Method

資料集的切分上因為 sample 數量很少且為了保留每個類別的樣本百分比使用 StratifiedShuffleSplit 以 7:3 的方式切成 training 和 test。

首先為了挑出和復發相關的 lncRNA,利用了 ranksum test 挑出了 p-value < 0.01 的 lncRNA,共 366 個,如**圖一**。

接著為了找出更少的 lncRNA 使其在存活預測上也有不錯的表現,我做了 2 個實驗來做更細的挑選:

1. 利用存活模型本身的 coef 或是 feature importance 排序後從低到高一次去掉一個特徵,用迭代的方式把特徵去掉後做 3 fold cross validation 找出 c-index 最高時的特徵數並記錄下來。
2. 因為 C-index 是把所有的病患隨機兩兩配對,如果生存時間較長的病患,其根據模型預測生存時間長於另一位生存時間較短的病患;或是生存機率高於另一位生存機率較低的病患,則稱之為預測結果與實際結果相一致,最後再把一致配對的樣本數除以有效配對的樣本所得出來。因此我打算再利用機器學習的方式如 Lasso、XGBRegression、GradientBoostingRegression 找出和預測存活時間相關的特徵以縮小範圍,再利用這些去跑存活分析的模型,找出最適合的模型以及最少的特徵數。

最後選出 cross validation 分數最高以及其對應的特徵數作為預選的模型及特徵去做超參數挑選,接著去做 test。若是在 test 表現出 overfitting 的現象則重回上面找表現次好的模型以及特徵數。

找出表現不錯且沒有 overfitting 的模型及對應特徵後,最後再利用特徵的重要程度排序,將較重要的特徵做 logrank test 並且做 KM 圖。上述方法的流程圖如**圖二**

```
titles = list(df.iloc[:,0:-1].columns)
drop = list(df.iloc[:,-1])
df = df
p01_name=[]
d=0
for i in titles:
        value1=[]
        value0=[]
        my_col = df[[i,"recurrence"]]


        for j in range(0,my_col.shape[0]):
                if (str(my_col.iloc[j,1]) == "1"):
                        value1.append(my_col.iloc[j,0])
                else:
                        value0.append(my_col.iloc[j,0])
        value0 = np.array(value0)
        value1 = np.array(value1)
        ttt = ranksums(value0,value1)
        if ttt.pvalue<0.01:
            d=d+1
            p01_name.append(i)
print(d)
print(p01_name)
```

```
366
['ENSG00000162913.10', 'ENSG00000176349.11', 'ENSG000001
```

圖一 ranksum test 結果

圖二 方法流程簡圖

延伸的部分則再另外加上了其他臨床資料的欄位未來做個人化預測,且也利用挑出來的模型做看看個人存活機率的預測。

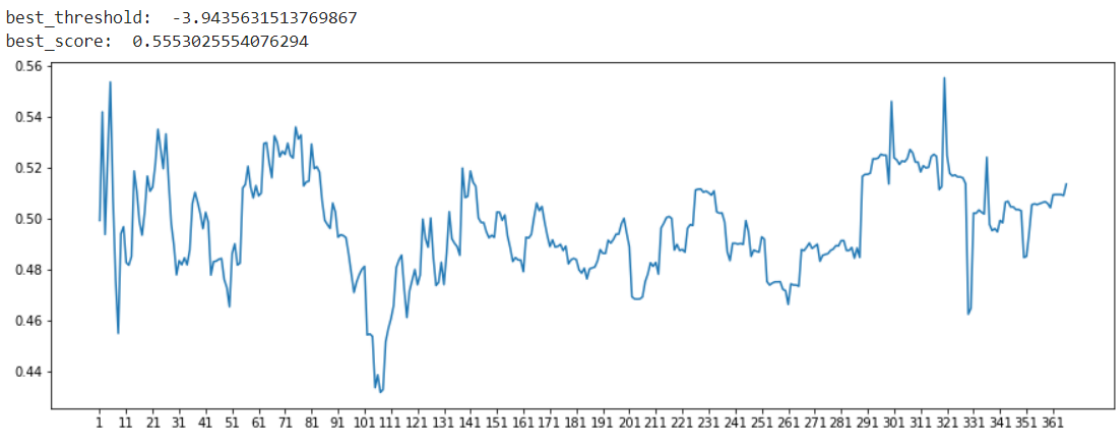
五. 結果 Result

5.1 實驗一: 利用模型的 coef 或是 feature importance

實驗一各項的 3-fold cross validation 的 c-index vs 特徵數折線圖以及所有的準確度和特徵數比較圖如下(所有較大的程式碼會附在修課心得後面以防占版面),其中 best threshold 是紀錄 coef 或是 feature importance 只要大於 best threshold 就可以得到最好 c-index 對應的特徵數。

5.1.1 coxPH

會把 alpha 設為 0.01 是因為不調 alpha 的話容易跑出 search direction contains NaN or infinite values 而終止,上網查到的解法是說把 alpha 調小就比較不會報錯,因此才將 alpha 設為 0.01。由**圖三**來看不同特徵數下的 c-index 波動很大,但是有一點點呈現出 M 型分布的感覺。而最後找出的特徵數為 320 個, training 以及 3-fold cross validation 的 c-index 為 1 和 0.555**(圖四)**。



圖三 CoxPH 的 3 fold cross validation vs 特徵數折線圖

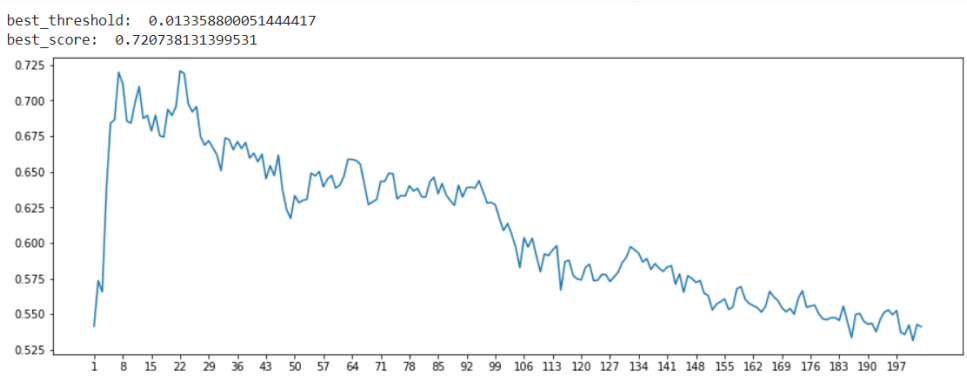
```
coxPH = CoxPHSurvivalAnalysis(alpha = 0.01)
train_mcv(coxPH, coxph_x_train_p01, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 1.0
cross validation score: 0.5553025554076294
```

圖四 CoxPH 的 training 和 3-cross validation 的 c-index

5.1.2 GBS:

把一些 feature importance 為 0 的特徵去掉後共有 202 個,由**圖五**來看雖然不同特徵數下的 c-index 都有一些小波動,但是隨著特徵數的上升,大趨勢有明顯的往下,推測有可能是因為雖然這些特徵有一些些關係,但是對於準確度來說還是不夠相關反而形成躁聲,而最後找出的特徵數為 22 個, training 以及 3-fold cross validation 的 c-index 為 0.961 和 0.678**(圖六)**



圖五 GBS 的 3 fold cross validation vs 特徵數折線圖

```
gbs = GradientBoostingSurvivalAnalysis()
train_mcv(gbs, gbs_x_train_p01, y_train_struct, ystatus_train, ytime_train, 3)
```

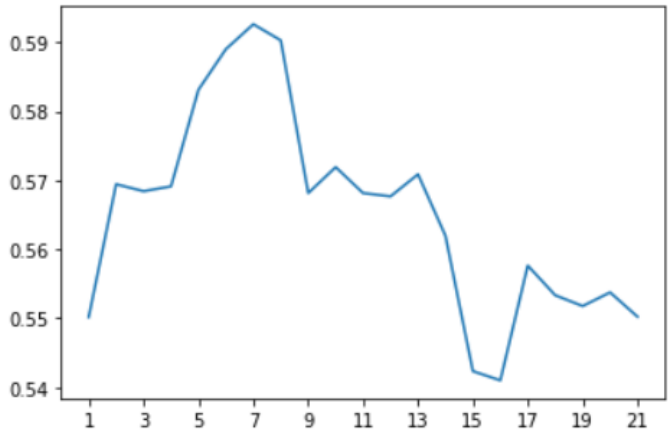
```
train cindex: 0.9610655737704918
cross validation score: 0.6789406696742772
```

圖六 GBS 的 training 和 3-cross validation 的 c-index

5.1.3 CGBS:

把一些 coef 為 0 的特徵去掉後共有 21 個,而最後找出的特徵數為 7 個**(圖七)**, training 以及 3-fold cross validation 的 c-index 為 0.612 和 0.592**(圖八)**

```
best_threshold:  0.0651708187969308
best_score:  0.5925983526466508
```



圖七 CGBS 的 3 fold cross validation vs 特徵數折線圖

```
cgbs = ComponentwiseGradientBoostingSurvivalAnalysis()
train_mcv(cgbs, cgbs_x_train_p01, y_train_struct, ystatus_train, ytime_train, 3)
```

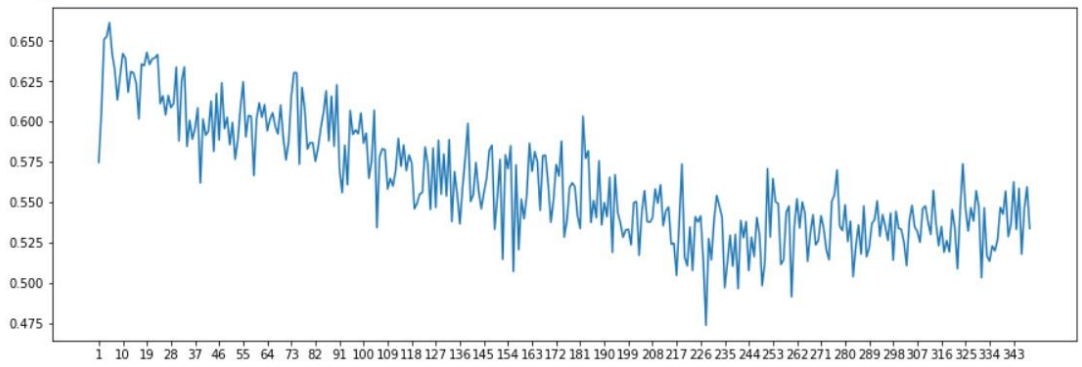
```
train cindex: 0.6129781420765027
cross validation score: 0.5925983526466508
```

圖八 CGBS 的 training 和 3-cross validation 的 c-index

5.1.4 RSF:

因為 RandomForestSurvival 本身是沒有實作 coef 或是 feature importance 的 attribute, 因此利用 sklearn 中的 permutation importance 去計算出每個特徵的 feature importance。接著把為 0 的特徵去掉後共有 347 個。由**圖九**來看也是和 GBS 一樣,雖然有波動,但是隨著特徵數上升其 c-index 有明顯往下的趨勢,而最後找出的特徵數為 5 個**(圖九)**, training 以及 3-fold cross validation 的 c-index 為 0.778 和 0.653**(圖十)**,而

```
best_threshold:  0.0025956284153004993
best_score:  0.6614619680545855
```



圖九 RSF 的 3 fold cross validation vs 特徵數折線圖

```
rsf = RandomSurvivalForest()
train_mcv(rsf, rsf_x_train_p01, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.7782786885245901
cross validation score: 0.6533290746522906
```

圖十 RSF 的 training 和 3-cross validation 的 c-index

5.2 實驗二: 先用回歸模型找出和存活時間相關的特徵再去跑存活模型

利用 RFECV(cv = 3, scoring = r2)或是 LassoCV(cv = 3)找出和存活時間相關的特徵後再去跑存活特徵,所使用的回歸模型包含: GradientBoostingRegression、LassoCV、XGBRegression。而為了比較,因此存活模型也是使用和實驗一相同的模型。

5.2.1 GradientBoostingRegression:

由 GBR 挑選出來的特徵有 225 個(**圖十一**),而由存活模型跑出來的結果為 coxPH 的 cross validation 結果最好為 0.531(**圖十二**)

```
gbr = GradientBoostingRegressor()
n_feature, gbr_selected = rfecv_reg(gbr, x_train_p01, ytime_train, 3)
```

```
Optimal number of features : 225
```

圖十一 GBR 跑出的特徵數

```
cox = CoxPHSurvivalAnalysis(alpha = 0.001)
train_mcv(cox, gbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 1.0
cross validation score: 0.5313230965785278
```

```
gbs = GradientBoostingSurvivalAnalysis()
train_mcv(gbs, gbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.9778688524590164
cross validation score: 0.4661191873481353
```

```
cgbs = ComponentwiseGradientBoostingSurvivalAnalysis()
train_mcv(cgbs, gbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.6856557377049181
cross validation score: 0.5111691737827104
```

```
rsf = RandomSurvivalForest()
train_mcv(rsf, gbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.8677595628415301
cross validation score: 0.5257983401757469
```

圖十二 GBR+各 survival model 的結果

5.2.2 XGBR:

由 XGBR 挑選出來的特徵只有 1 個(**圖十三**),我大概試驗了 5 次跑出來的結果都只有 1 個,雖然覺得不太可能,但還是依照結果去跑存活模型,其結果為 GBS 的 cross validation 結果最好為 0.540(**圖十四**)

```
xgbr = XGBRegressor(objective = 'reg:squarederror')
n_feature, xgbr_selected = rfecv_reg(xgbr, x_train_p01, ytime_train, 3)
```

```
Optimal number of features : 1
```

圖十三 XGBR 跑出的特徵

```
cox = CoxPHSurvivalAnalysis(alpha = 0.001)
train_mcv(cox, xgbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.4939207650273224
cross validation score: 0.4641194049256091
```

```
gbs = GradientBoostingSurvivalAnalysis()
train_mcv(gbs, xgbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.7504781420765028
cross validation score: 0.5408473713921442
```

```
cgbs = ComponentwiseGradientBoostingSurvivalAnalysis()
train_mcv(cgbs, xgbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.4939207650273224
cross validation score: 0.4641194049256091
```

```
rsf = RandomSurvivalForest(n_estimators=70, min_samples_split=15, min_samples_leaf=10, n_jobs=-1,)
train_mcv(rsf, xgbr_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.6209699453551912
cross validation score: 0.5300161591075078
```

圖十四 XGBR+各 survival model 的結果

5.2.3 LassoCV

由 LassoCV 把 coef 為 0 的去掉之後所挑選出來的特徵有 3 個(**圖十五**),而跑存活模型結果為 RSF 的 cross validation 結果最好為 0.550(**圖十六**)

```
lassocv = LassoCV(cv = 3).fit(x_train_p01, ytime_train)
```

```
lassocv_feature = x_train_p01.loc[:, lassocv.coef_[:] != 0]
lassocv_tefeature = x_test_p01.loc[:, lassocv.coef_[:] != 0]
```

圖十五 LassoCV 跑出的特徵

```
cox = CoxPHSurvivalAnalysis(alpha = 0.001)
train_mcv(cox, lassocv_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.5147540983606558
cross validation score: 0.4910955357480453
```

```
gbs = GradientBoostingSurvivalAnalysis()
train_mcv(gbs, lassocv_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.8625683060109289
cross validation score: 0.5001377670050394
```

```
cgbs = ComponentwiseGradientBoostingSurvivalAnalysis()
train_mcv(cgbs, lassocv_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.4830601092896175
cross validation score: 0.47908898719675974
```

```
rsf = RandomSurvivalForest(n_estimators=70, min_samples_split=15, min_samples_leaf=10, n_jobs=-1,)
train_mcv(rsf, lassocv_feature, y_train_struct, ystatus_train, ytime_train, 3)
```

```
train cindex: 0.7157786885245901
cross validation score: 0.5502590001121055
```

圖十六 LassoCV+各 survival model 的結果

就實驗一(**表一**)以及實驗二(**表二**)的結果來看,以 cross validation 來說實驗一的結果較實驗二好,其中實驗一依照 3-fold cv 的 c-index 表現排序為: GBS、RSF、CGBS、coxPH, 分別為 0.678、0.653、0.592、0.555,推測有可能是因為我在做 regression 時都用默認的超參數,沒有特別挑選,因此選出來的特徵沒有和存活時間非常契合。但依照實驗出來的結果最後還是選擇了實驗一各模型以及所挑出的特徵來做超參數的挑選以及 test 測試有沒有 overfitting,若是有的話則回頭找次好的模型。依序為:GBS、RSF、CGBS、coxPH。

	coxPH	GBS	CGBS	RSF
Train	1	0.961	0.612	0.778
3-fold cv	0.555	0.678	0.592	0.653
features	320	22	7	5

表一 實驗一結果表

	GBR				XGBR				Lasso			
	coxPH	GBS	CGBS	RSF	coxPH	GBS	CGBS	RSF	coxPH	GBS	CGBS	RSF
Train	1	0.977	0.685	0.867	0.493	0.750	0.493	0.620	0.514	0.862	0.483	0.715
3-fold cv	0.531	0.466	0.511	0.525	0.464	0.540	0.464	0.530	0.491	0.500	0.479	0.550
features	225	225	225	225	1	1	1	1	3	3	3	3

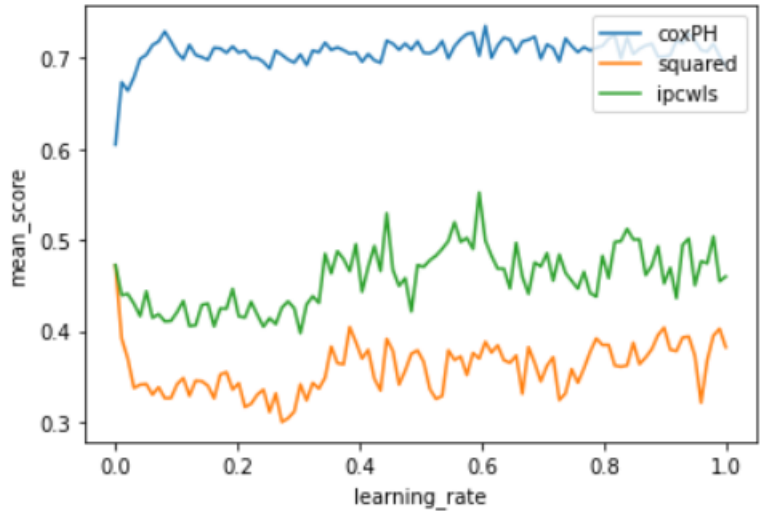
表二 實驗二結果表

5.3 挑超參數和 test

5.3.1 GBS:

挑選了 2 個參數做調整,分別是 loss 和 learning rate,其學習曲線如**圖十七**,由圖中可看出 loss 選 coxPH 的表現相較於其餘兩者有很大的區別,但是 learning rate 方面在各 loss 下則差異不大,而最後選出最好的 learning 為 0.6061。但是挑完超參數後再做 test 後發現表現不好有 overfitting 的現象(**圖十八**),因此往下做次好的 model 測試。

```
loss:  coxph
learning rate:  0.6061
best score:  0.735596355390882
```



圖十七 GBS 挑超參數的學習曲線

	C-index	C-ipcw
training	0.998087	0.997867
test	0.536158	0.415401

圖十八 GBS 挑完超參數後測 test 的結果

5.3.2 RSF

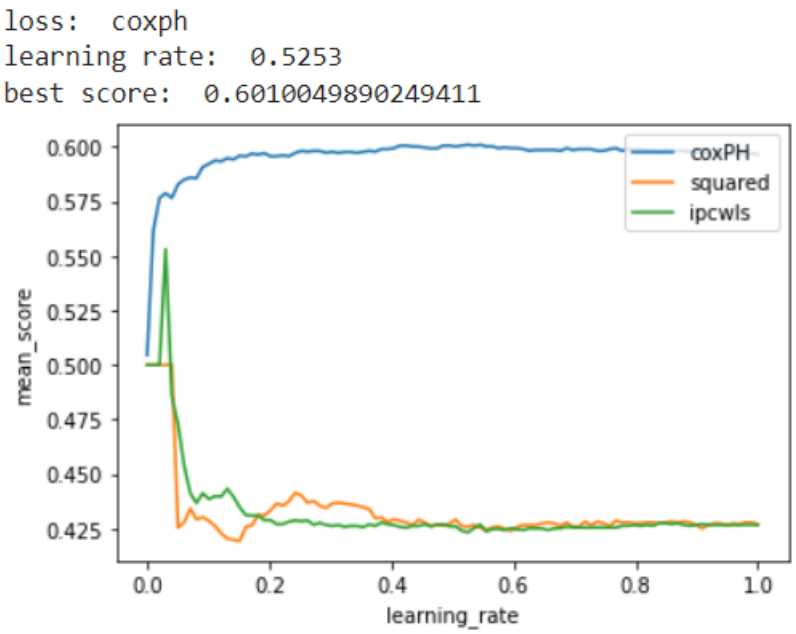
而 RSF 因為建模時間較長為了不要挑完之後還是 overfitting,因此我有先手動條一些參數使其不要那麼容易 overfitting 像是把 n_estimator 調低(=70), min_sample_split (=15) 以及 min_sample_split (=10)調高,若是做 test 的結果也不錯,再仔細的挑超參數看能不能提升表現。而測試的時候發現 RSF 跑出的結果會有一些蠻大的浮動,因此為了看他的平均表現,我跑了 50 次再去算平均值,結果如**圖十九**。很顯然的表現還是不好,有 overfitting 的現象。因此繼續往下做次好的 model。

	C-index_avg	C-ipcw_avg
training	0.774109	0.760500
test	0.578598	0.445527

圖十九 RSF 50 次的平均值

5.3.3 CGBS

CGBS 選的超參數和 GBS 相同,皆為 loss 以及 learning rate,其學習曲線如**圖二十**,圖中可看出 loss 為 coxPH 時的表現相較於其餘兩者好很多,而 learning rate 的波動相較於 GBS 較平緩很多,而最後選出的 learning rate 為 0.5253。在挑完超參數後測 test 發現表現還不錯(**圖二十一**),因此最後也選擇了這個模型以及所挑出的 7 個特徵,分別為 ENSG00000254622.1、ENSG00000237975.7、ENSG00000248469.1、ENSG00000223466.2、ENSG00000262061.6、ENSG00000224758.1、ENSG00000229628.1



圖二十 CGBS 挑超參數的學習曲線

	C-index	C-ipcw
training	0.612842	0.610431
test	0.647188	0.639316

圖二十一 GBS 挑完超參數後測 test 的結果

六. 討論 Discussion

經由實驗一以及實驗二所呈現出來的結果,實驗一(用 survival 挑特徵)的表現比實驗二(regression+survival)好, 推測有可能是因為我在做 regression 時都用默認的超參數,沒有特別挑選,因此選出來的特徵沒有和存活時間非常契合。但依照實驗出來的結果最後還是選擇了實驗一各模型以及所挑出的特徵。

而 test 的 3 個模型結果如**表三**,由表中可看出,雖然 GBS 和 RSF 在 training 以及 3-fold cross validation 表現不錯,但是在 test 時卻不如 CGBS,有明顯 overtraining 的現象。我認為可能是資料本身樣本數很少,且在拆分成 training 以及 test 後兩者的資料分布差異較

大,而 GBS 以及 RSF 相較於 CGBS 較為複雜,因此在樣本少以及資料分布差異較大的情況下前兩者都可以把 train 以及用 train 來做的 cross validation 做不錯,但是容易 overfitting,而 CGBS 雖然和 GBS 和 RSF 同樣都是集成式學習,但是最終模型還是線性模型,類似於 Lasso penalized Cox model,因此雖然在 train 以及 cross validation 表現不如 GBS 以及 RSF,但是 test 可以也可以表現得不錯,泛化能力較好。

	GBS	✓ CGBS	RSF
Train	0.998	0.612	0.774
test	0.536	0.647	0.578

表三 test 結果整理

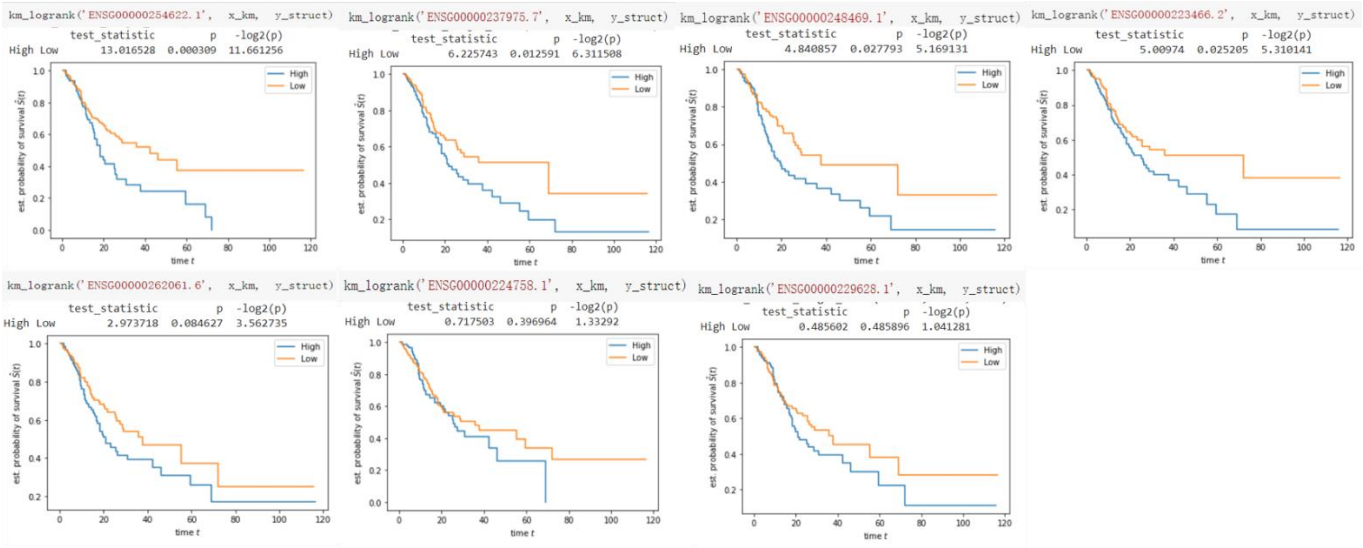
接著為了看這些 lncRNA 和存活的相關程度,先用 CGBS 本身的 feature importance 把他們排序(**圖二十二**),接著分成高表現以及低表現再做了 KM plot 還有 logrank test(**圖二十三**)。依照 feature importance 排序分別為: ENSG00000254622.1、ENSG00000237975.7、ENSG00000248469.1、ENSG00000223466.2、ENSG00000262061.6、ENSG00000224758.1、ENSG00000229628.1,而在對存活的 logrank test 的 p-value 中,前 4 個(ENSG00000254622.1、ENSG00000237975.7、ENSG00000248469.1、ENSG00000223466.2)較有顯著的差異,依序為:0.000309、0.012591、0.027793、0.025205。顯示說這 4 個 lncRNA 不僅對復發有相關,在對存活方面也有一定的相關性。

```
cgbs = ComponentwiseGradientBoostingSurvivalAnalysis(loss = 'coxph', learning_rate = 0.5253).fit(cgbs_x_train_p01, y_train_struct)
```

```
feature_importance = pd.DataFrame({
        "feature importance": cgbs.feature_importances_[1:]
})

feature_importance.index=cgbs_x_train_p01.columns
feature_importance.sort_values(by="feature importance", ascending=False)
```

	feature importance
ENSG00000254622.1	1.0
ENSG00000237975.7	2.0
ENSG00000248469.1	3.0
ENSG00000223466.2	5.0
ENSG00000262061.6	6.0
ENSG00000224758.1	11.0
ENSG00000229628.1	12.0

圖二十二 CGBS 的特徵重要度排序



圖二十三 CGBS 挑出的特徵做的 KM plot 以及 logrank test

延伸

增加了一些臨床欄位如年齡、MSIsensor Score 以及 Fraction Genome Altered(**表四**), 但因為加進去後會導致有一些缺值,且為了不要再減少樣本數,因此選擇了利用 KNN 補值,結果表現是有上升一點點(**圖二十四**)。

1 to 25 of 440 entries Filter

Diagnosis Age	MSIsensor Score	Fraction Genome Altered
70.0	0.13	0.523
51.0	0.0	0.1752
51.0	28.57	0.0876
62.0	0.33	0.206
52.0	13.82	0.2297
74.0	1.4	0.4391
51.0	0.04	0.0004
60.0	0.13	0.4807
55.0	0.0	0.23
70.0	33.44	0.0613
56.0	0.04	0.0006
53.0	0.0	0.0534
72.0	0.0	0.4263
66.0	22.15	0.0517
80.0	0.0	0.0235
76.0	0.06	0.0988
80.0	11.0	0.0
79.0	27.08	0.4109
51.0	0.21	0.3089
43.0	0.04	0.1568
78.0	22.22	0.0585
73.0	27.53	0.1165
65.0	0.1	0.9083
58.0	0.25	0.1489
66.0	44.14	0.0612

表四 CGBS 挑出的特徵做的 KM plot 以及 logrank test

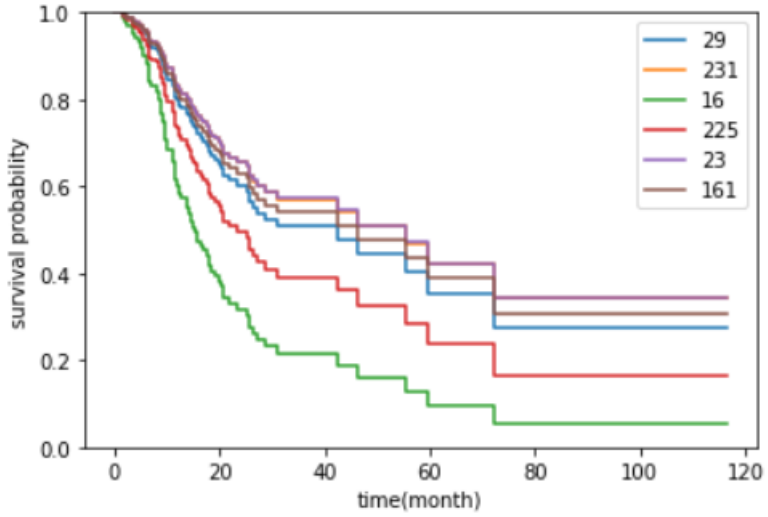
```
from  sklearn.impute  import  KNNImputer

imputer  =  KNNImputer(n_neighbors  =  5)
x_psl_train_all  =  imputer.fit_transform(x_psl_train_all)
```

```
cgbs_model = ComponentwiseGradientBoostingSurvivalAnalysis(loss = 'coxph', learning_rate = 0.5253)
C_indicator(cgbs_model, x_psl_train_all, y_train_struct, ystatus_train, ytime_train, x_psl_test_all, y_test_struct, ystatus_test, ytime_test)
```

	C-index	C-ipcw
training	0.611612	0.616795
test	0.650840	0.635750

圖二十四 KNN 補值 & 加臨床欄位後的結果

我也有試著利用模型來畫患者的存活機率圖,使用的患者資料是從 train 的資列集裡取 6 個人出來,由**圖二十五**可看出只有編號 225 的患者和其他患者所預測的的相對存活機率和相對實際存活時間(**表五**)不同。但也許是因為是用訓練集的關係錯誤率才比較小,若是用測試集或是助教給的獨立測試集則可能錯誤率會比較大。



圖二十五 挑出患者的存活機率預測

	ENSG00000223466.2	ENSG00000224758.1	ENSG00000229628.1	ENSG00000237975.7	ENSG00000248469.1	ENSG00000254622.1	ENSG00000262061.6	Overall Survival (Months)
29	0.0837	0.0776	0.0000	0.0939	0.0309	0.0397	0.2053	6.739652
231	0.0000	0.0000	0.0000	0.0295	0.0000	0.0000	0.1075	15.221751
16	0.2036	0.0000	0.2757	0.3460	0.0000	0.0000	2.0767	5.030082
225	0.0350	0.0000	0.1105	0.4590	0.0000	0.0000	0.2573	28.733932
23	0.0238	0.0000	0.0000	0.0080	0.0264	0.0000	0.0175	19.955946
161	0.0000	0.0000	0.0000	0.0367	0.0000	0.0423	0.3498	11.309465

表五 挑出患者的實際存活情況

助教給的獨立測試集結果

其結果如**圖二十六**,c-index 以及 c-ipcw 分別為 0.515 以及 0.479,相較於 test 下降了非常多。也許是因為雖然這些特徵和復發和存活都有關,但是還不足以把存活預測得很好。或是我的 train 以及 test 和 independent test 的資料分布差異也很大,因此在 train 以及 test 表現不錯,但是獨立測試卻不好。

```
[13] ind_model = load('final_model.joblib')

[14] ind_feature_ = pd.read_csv('final_features.csv')

[15] select = []
     for i in ind_feature_:
         select.append(i)

[16] ind = pd.read_csv('stad(Ind.).csv')

[17] ind_status = ind.iloc[:, -2]
     ind_time = ind.iloc[:, -1]
     ind_y = ind.iloc[:, -2:]

[18] y_ind_struct = ind_y.to_records(index=False).astype([('Status', 'bool'), ('Survival', 'float64')])

[19] ind_feature = ind.loc[:, select]

[20] ind_pre = ind_model.predict(ind_feature)
     ind_cindex, ind_iccd, ind_idcd, ind_iti_risk, ind_iti_time = concordance_index_censored(ind_status > 0, ind_time, ind_pre)
     ind_cipcw, ind_pccd, ind_pdcd, ind_pti_risk, ind_pti_time = concordance_index_ipcw(y_ind_struct, y_ind_struct, ind_pre)

[21] print("ind-test c-index:", ind_cindex)
     print("ind-test c-ipcw:", ind_cipcw)

     ind-test c-index: 0.5157766990291263
     ind-test c-ipcw: 0.4792037710876706
```

圖二十六 獨立測試集結果

七. 參考資料 References

https://zh.m.wikipedia.org/zh-hant/%E8%83%83%E7%99%8C
https://scikit-survival.readthedocs.io/en/stable/user_guide/boosting.html
https://scikit-survival.readthedocs.io/en/stable/user_guide/boosting.html
https://www.yongxi-stat.com/c-index/

八. 修課心得 Course Review

這堂課應該把所有機器學習會碰到的問題都有講述過一遍,諸如二分類、多分類、回歸,更甚至於存活的問題,也講了基因演算法。由於本身對於資訊領域同樣也有很大的興趣,對於 AI 方面更是勝於其他諸如網路、系統、軟體、資安...等等。但礙於本身實力不足,雖然老師在台上口沫橫飛,講了很多的算法以及背後的數學理論(有很好心的講簡單一點,以免我們聽不懂)等等,但我幾乎都還是似懂非懂(幾乎是不太懂),問題也很多以至於

也不知道要怎麼問。只好自己找了很多資料,像是一些國內外的網站如 IT 幫幫忙、知乎、CSDN 博客、sklearn 或是 sksurvival 自己的官網以及一些個人的 blog 等等。而在書籍方面我也有看了一點 C. Bishop, Pattern Recognition and Machine Learning 還有 Hands-On Machine Learning with Scikit-Learn and TensorFlow。發現到其實 AI 方面不論是機器學習或是深度學習都是基於數學原理的實作,每個演算法的背後都是深厚的數學理論。也讓這學期修的線性代數有一點起到作用,像是 PCA 其實就是以 SVD(singular value decomposition)為根基去實作,降維、特徵提取也都是需要線性代數的觀念。還有老師在講到特徵數太多而樣本數太少會導致 underdetermined 也是讓我瞬間聽懂。

但是上課的部分也從老師身上學到了很多大方向觀念像是 garbege-in-garbege-out、no-free-lunch,因此我也在前處理的時候想盡量把噪聲丟掉以保留最有用的一組特徵出來,還有算法上盡量多嘗試希望可以找到合適的模型來讓評估指標更好。也很喜歡老師在介紹演算法的時候會用比喻的方式去簡化複雜的理論,像是 SVM 就是盡量找一條楚河漢界可以分開兩邊資料,可是為了不要 overfitting 或是 underfitting 因此可以調整 C 來決定誤差的容忍程度。也知道了其實 regression model 也是可以做 classification,只要決定 threshold 就好。同樣理解了雖然都是算 p-value,但是不同的資料型態對於要用哪一種算法也是有講究,像是離散型用卡方,連續型用 Mann–Whitney U test。

作業方面也是花了很多的時間,幾乎都要花個 3、4 天去完成,期間也是看過了好幾次日出,以及看了好幾次大家的 debug 好朋友—stack overflow。而 finalproject 更是花了 2、3 個禮拜,雖然似乎結果預測的都不是非常好。但這也讓我了解到在處理真實世界資料的時候,結果不可能跟 iris 資料集一樣怎麼測都 0.9 以上,是需要經過好幾次的試錯才可以得到還可以看的結果。還有作業也讓我複習了很久沒有用的 python,也學習了一些資料處理的 package,諸如 pandas、numpy。雖然期間都有使用 C++做一些作業,所以突然要打程式的時候不是至於完全生疏,但是要突然轉換語言也是有些不習慣,用了才發現 python 真好用……。也非常感謝彥成助教以及采瑩學姊,彥成助教在每個作業都有提供 sample code 讓我在作業的時間減少了很多,也比較知道要從何下手;采瑩學姊給了我很多的知識支援。

這學期在各方面都下了很多的功夫,也有一些實作經驗。但對於這方面還是菜鳥的我還是只能做一些粗糙的推測以及不到位的分析。因此之後也會盡量去精進自己的實力,像是修資工系的機器學習或是深度學習與實務之類的課。也為了不要讓自己只是淪為所謂的套膜仔,會盡量去理解背後的原理以及算法。

*以下是一些上面沒有展示的 code:

做 train 以及 test 的函式

```
def C_indicator(model, x_train, y_train, y_trstatus, y_trtime, x_test, y_test, y_testatus, y_tetime):

    #modeling
    model.fit(x_train, y_train)

    #train
    train_pre = model.predict(x_train)
    tr_c_index, tr_iccd, tr_idcd, tr_iti_risk, tr_iti_time = concordance_index_censored(y_trstatus > 0, y_trtime, train_pre)
    tr_c_ipcw, tr_pccd, tr_pdcd, tr_pti_risk, tr_pti_time = concordance_index_ipcw(y_train, y_train, train_pre)

    #test
    test_pre = model.predict(x_test)
    te_c_index, te_iccd, te_idcd, te_iti_risk, te_iti_time = concordance_index_censored(y_testatus > 0, y_tetime, test_pre)
    te_c_ipcw, te_pccd, te_pdcd, te_pti_risk, te_pti_time = concordance_index_ipcw(y_train, y_test, test_pre)

    #combine
    result = {
              "C-index":[tr_c_index, te_c_index],
              "C-ipcw":[tr_c_ipcw, te_c_ipcw]
    }

    indicator = pd.DataFrame(result)
    indicator.index = ['training', 'test']
    indicator.round(3)
    print(indicator)
```

regression+RFECV

```
def rfecv_reg(model, x_train, y_train, step):
    r = RFECV(estimator = model, step = step, cv = 3, scoring = 'r2').fit(x_train, y_train)
    print("Optimal number of features : %d" % r.n_features_)

    return r.n_features_, r.support_
```

用來看 train 以及 cross validation 的 c-index

```
def train_mcv(suv_model, x_train, y_train, ystatus_train, ytime_train, cv):

    train_model = suv_model.fit(x_train, y_train)
    train_pre = train_model.predict(x_train)
    tr_c_index, tr_iccd, tr_idcd, tr_iti_risk, tr_iti_time = concordance_index_censored(ystatus_train > 0, ytime_train, train_pre)

    val_score = my_cross_val(suv_model, x_train, y_train, ystatus_train, ytime_train, cv)

    print('train cindex:', tr_c_index)
    print('cross validation score:', val_score)
```

因為 sklearn 的 cross validation 沒有支援 c-index,且想自己實做一個,因此就自己做了 cross validation

```
def my_cross_val(model, x_train, y_train, ystatus_train, ytime_train, cv):

    cvs = 0
    record = []
    num_val_samples = len(x_train)//cv
    cols = x_train.columns

    for i in range(cv):

            val_data = x_train[i*num_val_samples : (i+1)*num_val_samples]
            val_targets = y_train[i*num_val_samples : (i+1)*num_val_samples]

            remaining_data = np.concatenate(
                                            [x_train[: i*num_val_samples],
                                            x_train[(i+1)*num_val_samples :]],
                                            axis = 0)
            remaining_targets = np.concatenate(
                                            [y_train[: i*num_val_samples],
                                            y_train[(i+1)*num_val_samples :]],
                                            axis = 0)

            val_ystatus = ystatus_train[i*num_val_samples : (i+1)*num_val_samples]
            val_ytime = ytime_train[i*num_val_samples : (i+1)*num_val_samples]

            remaining = pd.DataFrame(remaining_data, columns=cols)
            val = pd.DataFrame(val_data, columns=cols)

            now_score = mcs_c_index_scoring(model, remaining, remaining_targets, val_data, val_ystatus, val_ytime)
            record.append(now_score)

    for i in record:
        cvs += (i/cv)

    return cvs
```

配合 my_cross_val 的 scoring

```
def mcs_c_index_scoring(model, x_train, y_train, x_val, ystatus_val, ytime_val):
    model.fit(x_train, y_train)
    pre = model.predict(x_val)
    a, b, c, d, e = concordance_index_censored(ystatus_val>0, ytime_val, pre)

    return a
```

同時做 logrank 以及 KM plot

```
def km_logrank(inter, X_train, y_train_struct):

    gene = X_train[inter]
    median = statistics.median(gene)
    for i in X_train.index:
        if gene.loc[i] > median:
            gene.loc[i] = "High"
        else:
            gene.loc[i] = "Low"

    for expression in ("High","Low"):
        mask_treat = gene == expression
        time_treat, survival_prob_treat = kaplan_meier_estimator(y_train_struct["Status"][mask_treat],y_train_struct["Survival"][mask_treat])
        plt.step(time_treat, survival_prob_treat, where="post",label=expression)

    log_rank = pairwise_logrank_test(y_train_struct["Status"], gene, y_train_struct["Survival"])
    print(log_rank.summary)

    plt.ylabel("est. probability of survival $\hat{S}(t)$")
    plt.xlabel("time $t$")
    plt.legend(loc="best")
```

實驗一:

coxPH

```
coxPH = CoxPHSurvivalAnalysis(alpha = 0.01).fit(x_train_p01, y_train_struct)
```

```
threshold = coxPH.coef_
threshold.sort()
```

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

for i in threshold:
    temp_feature = x_train_p01.loc[:, coxPH.coef_[:] >= i]
    temp_coxph = CoxPHSurvivalAnalysis(alpha = 0.01).fit(temp_feature, y_train_struct)
    now_score = my_cross_val(temp_coxph, temp_feature, y_train_struct, ystatus_train, ytime_train, 3)
    score.append(now_score)
    if(now_score > best_score and now_score < 0.9):
        best_score = now_score
        best_threshold = i

max = len(threshold)
score.reverse()
print('best_threshold: ', best_threshold)
print("best_score: ", best_score)
plt.figure(figsize=(15,5))
plt.plot(range(1, max+1), score)
plt.xticks(range(1, max+1, 10))
plt.show()
```

GBS

```
gbs = GradientBoostingSurvivalAnalysis().fit(x_train_p01, y_train_struct)
```

```
threshold = gbs.feature_importances_
threshold.sort()
```

```
threshold = threshold[threshold[:] != 0]
```

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

for i in threshold:
    temp_feature = x_train_p01.loc[:, gbs.feature_importances_[:] >= i]
    temp_gbs = GradientBoostingSurvivalAnalysis(n_estimators = 70).fit(temp_feature, y_train_struct)
    now_score = my_cross_val(temp_gbs, temp_feature, y_train_struct, ystatus_train, ytime_train, 5)
    score.append(now_score)
    if(now_score > best_score):
        best_score = now_score
        best_threshold = i

max = len(threshold)
score.reverse()
print("best_threshold: ", best_threshold)
print("best_score: ", best_score)
plt.figure(figsize = (15, 5))
plt.plot(range(1, max+1), score)
plt.xticks(range(1, max+1, 7))
plt.show()
```

CGBS:

```
#去0
cgbs = ComponentwiseGradientBoostingSurvivalAnalysis().fit(x_train_p01, y_train_struct)
cgbs_prefeature = x_train_p01.loc[:, cgbs.coef_[1:] != 0]
cgbs_test_prefeature = x_test_p01.loc[:, cgbs.coef_[1:] != 0]
```

```
#取coef
cgbs = ComponentwiseGradientBoostingSurvivalAnalysis().fit(cgbs_prefeature, y_train_struct)
threshold = cgbs.coef_[1:]
threshold.sort()
```

*cgbs 因為他的 coef_前面會多存一個 intercept,因此從 index 從 1 開始

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

for i in threshold:
    temp_feature = cgbs_prefeature.loc[:, cgbs.coef_[1:] >= i]
    temp_cgbs = ComponentwiseGradientBoostingSurvivalAnalysis().fit(temp_feature, y_train_struct)
    now_score = my_cross_val(temp_cgbs, temp_feature, y_train_struct, ystatus_train, ytime_train, 3)
    score.append(now_score)
    if(now_score > best_score):
        best_score = now_score
        best_threshold = i

max = len(threshold)
score.reverse()
print('best_threshold: ', best_threshold)
print("best_score: ", best_score)
plt.plot(range(1, max+1), score)
plt.xticks(range(1, max+1, 2))
plt.show()
```

RSF:

```
rsf = RandomSurvivalForest().fit(x_train_p01, y_train_struct)
result = permutation_importance(rsf, x_train_p01, y_train_struct)
```

```
importance = pd.DataFrame({
        "importances_mean": result['importances_mean']
})

importance.index=x_train_p01.columns
```

```
rsf_prefeature = x_train_p01.loc[:, importance[importance["importances_mean"] != 0].index]
rsf_test_prefeature = x_test_p01.loc[:, importance[importance["importances_mean"] != 0].index]
```

```
threshold = importance[importance["importances_mean"] != 0].sort_values(by="importances_mean", ascending=True)
importance = importance[importance["importances_mean"] != 0]
```

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

for i in threshold.iloc[:,0]:
    temp_feature = rsf_prefeature.loc[:, importance[importance["importances_mean"] >= i].index]
    temp_rsf = RandomSurvivalForest().fit(temp_feature, y_train_struct)
    now_score = my_cross_val(temp_rsf, temp_feature, y_train_struct, ystatus_train, ytime_train, 3)
    score.append(now_score)
    if(now_score > best_score):
        best_score = now_score
        best_threshold = i

max = len(threshold)
score.reverse()
print("best_threshold: ", best_threshold)
print("best_score: ", best_score)
plt.figure(figsize = (15, 5))
plt.plot(range(1, max+1), score)
plt.xticks(range(1, max+1, 9))
plt.show()
```

挑超參數

GBS

```
learning_rate = np.linspace(0.0001, 1, 100)
cox_score = []
squ_score = []
ipcw_score = []
best_score = 0
best_rate = 0
best_loss = ""

for l in ['coxph', 'squared', 'ipcwls']:
    for i in learning_rate:
        gbs = GradientBoostingSurvivalAnalysis(loss = l, learning_rate = i)
        now_score = my_cross_val(gbs, gbs_x_train_p01, y_train_struct, ystatus_train, ytime_train, 5)
        if(l == 'coxph'): cox_score.append(now_score)
        elif(l == 'squared'): squ_score.append(now_score)
        else: ipcw_score.append(now_score)

        if(now_score > best_score):
            best_loss = l
            best_score = now_score
            best_rate = i

print('loss: ', best_loss)
print('learning rate: ', best_rate)
print('best score: ', best_score)

plt.plot()
coxph, = plt.plot(learning_rate, cox_score, label = 'coxPH')
squared, = plt.plot(learning_rate, squ_score, label = 'squared')
ipcwls, = plt.plot(learning_rate, ipcw_score, label = 'ipcwls')
plt.xlabel('learning_rate')
plt.ylabel('mean_score')
plt.legend(handles = [coxph, squared, ipcwls], loc='upper right')
plt.show()
```

RSF

```
total_cindex_train = 0
total_ipcw_train = 0
total_cindex_test = 0
total_ipcw_test = 0
iter = 50

for i in range(0, iter):
    rsf = RandomSurvivalForest(n_estimators=70, min_samples_split=15, min_samples_leaf=10, n_jobs=-1,).fit(rsf_x_train_p01, y_train_struct)

    #train
    train_pre = rsf.predict(rsf_x_train_p01)
    tr_c_index, tr_iccd, tr_idcd, tr_iti_risk, tr_iti_time = concordance_index_censored(ystatus_train > 0, ytime_train, train_pre)
    tr_c_ipcw, tr_pccd, tr_pdcd, tr_pti_risk, tr_pti_time = concordance_index_ipcw(y_train_struct, y_train_struct, train_pre)

    #test
    test_pre = rsf.predict(rsf_x_test_p01)
    te_c_index, te_iccd, te_idcd, te_iti_risk, te_iti_time = concordance_index_censored(ystatus_test > 0, ytime_test, test_pre)
    te_c_ipcw, te_pccd, te_pdcd, te_pti_risk, te_pti_time = concordance_index_ipcw(y_train_struct, y_test_struct, test_pre)


    total_cindex_train = total_cindex_train + tr_c_index
    total_ipcw_train = total_ipcw_train + tr_c_ipcw
    total_cindex_test = total_cindex_test + te_c_index
    total_ipcw_test = total_ipcw_test + te_c_ipcw

result = {
       "C-index_avg":[total_cindex_train/iter, total_cindex_test/iter],
       "C-ipcw_avg":[total_ipcw_train/iter, total_ipcw_test/iter]
    }

indicator = pd.DataFrame(result)
indicator.index = ['training', 'test']
indicator.round(3)
print(indicator)
```

CGBS

```
learning_rate = np.linspace(0.0001, 1, 100)
cox_score = []
squ_score = []
ipcw_score = []
best_score = 0
best_rate = 0
best_loss = ""

for l in ['coxph', 'squared', 'ipcwls']:
    for i in learning_rate:
        cgbs = ComponentwiseGradientBoostingSurvivalAnalysis(loss = l, learning_rate = i)
        now_score = my_cross_val(cgbs, cgbs_x_train_p01, y_train_struct, ystatus_train, ytime_train, 3)
        if(l == 'coxph'): cox_score.append(now_score)
        elif(l == 'squared'): squ_score.append(now_score)
        else: ipcw_score.append(now_score)

        if(now_score > best_score):
            best_loss = l
            best_score = now_score
            best_rate = i

print('loss: ', best_loss)
print('learning rate: ', best_rate)
print('best score: ', best_score)

plt.plot()
coxph, = plt.plot(learning_rate, cox_score, label = 'coxPH')
squared, = plt.plot(learning_rate, squ_score, label = 'squared')
ipcwls, = plt.plot(learning_rate, ipcw_score, label = 'ipcwls')
plt.xlabel('learning_rate')
plt.ylabel('mean_score')
plt.legend(handles = [coxph, squared, ipcwls], loc='upper right')
plt.show()
```