Machine Learning in Computational Biology

HW3 report

109350008 張詠哲

目錄

建模訓練紀錄

Regression

- **資料切分**

  也是採 train : test = 7:3 的方式切分資料。

- **特徵初選(統計方法)**

  利用 pearson correlation 去篩選出有相關性的特徵，為了減少維度，但又不想去太多的特徵，因此選出相關係數絕對值>0.03 的特徵。總共 92 個

```
plt.figure(figsize=(8,60))
correlation_matrix = df.corr().loc[:,['label']]

ori_column = df.columns
column_and_corr = pd.DataFrame([ori_column, correlation_matrix['label']]).T
column_and_corr = column_and_corr.rename(columns = {0:'features', 1:'pearson'})
print(column_and_corr)
```

```
    features   pearson
0         d1  0.005184
1         d2 -0.011968
2         d3  0.007046
3         d4 -0.032338
4         d5 -0.005653
..       ...       ...
512     c513  0.001803
513     c514  0.014264
514     c515 -0.001792
515     c516  -0.00125
516    label       1.0

[517 rows x 2 columns]
<Figure size 800x6000 with 0 Axes>
```

```
prefeature = []

for i in column_and_corr.iloc:
    if(abs(i['pearson']) >= 0.03 and abs(i['pearson']) != 1): prefeature.append(i['features'])

print(prefeature)
print(len(prefeature))

train_predata = train_df.loc[:, prefeature]
test_predata = test_df.loc[:, prefeature]
```

- **模型訓練過程**

  接下來我總共做了兩個實驗，每個實驗都包含 4 種模型，包含 SVR、ElasticNet、

XGBoostRegressor(XGBR)、GradientBoostRegressor(GBR)，分別是指使用統計方法挑出的特徵(作為 baseline)以及有經過特徵挑選過的，特徵挑選的部分我使用了 RFECV(CV = 5、scoring = R2、step = 1)，但是 ElasticNet 所得出的結果都不好，因此對此我也在多做了 2 種不同的嘗試，多做了使用統計+embedd 以及用原特徵+embedd 的方式。而 2 個實驗找超參數的方式若是參數較多是採用 RandomSearchCV，若是參數少則採用類似 GridSearch 的方式。

- **Statistic only**

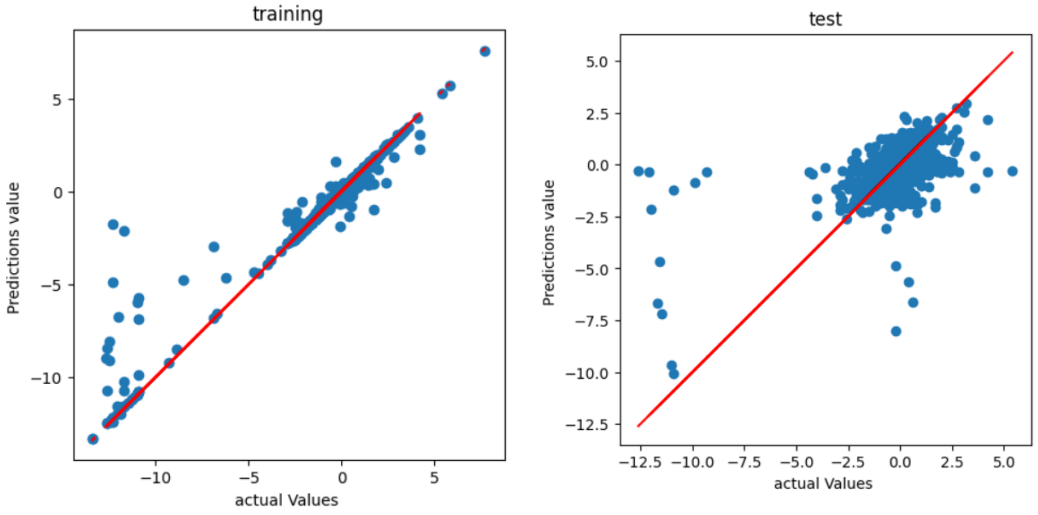
1. SVR

```
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 = SVR()
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': 64}
```

結果:

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	10.569732	0.113600	0.102565	0.320258	0.947116	0.946211
Test	12.279824	0.497205	0.831861	0.912064	0.317727	0.306042



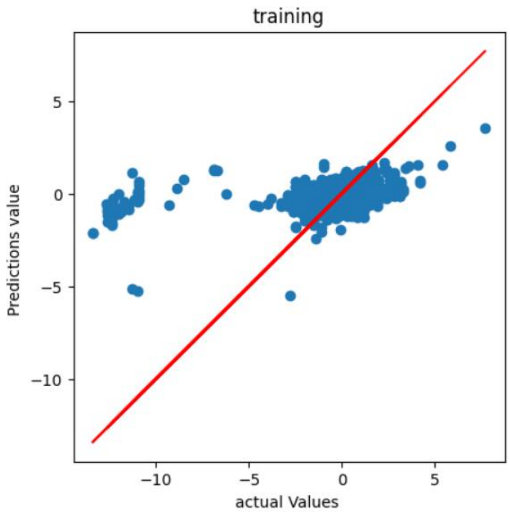
2. ElasticNet

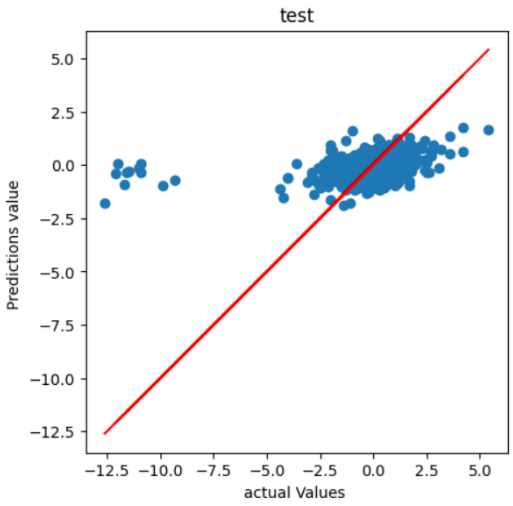
```
l1_ratio = np.linspace(0.001, 1, 50)
best_score = 0
op_al = 0
op_l1 = 0

for j in l1_ratio:
    elastic_cv = linear_model.ElasticNetCV(cv= 5, random_state = 0, l1_ratio = j)
    elastic_cv.fit(x_pretrain_std, y_train)
    score = elastic_cv.score(x_pretrain_std, y_train)

    if(score > best_score):
        op_al = elastic_cv.alpha_
        op_l1 = j
        best_score = score

# print(elastic_cv.alpha_)
# print(elastic_cv.l1_ratio_)
print(op_al)
print(op_l1)
```

所選出的最佳 alpha = 0.01637、L1 = 1。

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	12.465616	0.638956	1.715917	1.309930	0.115255	0.100103
Test	12.047188	0.596509	1.081804	1.040098	0.112729	0.097534





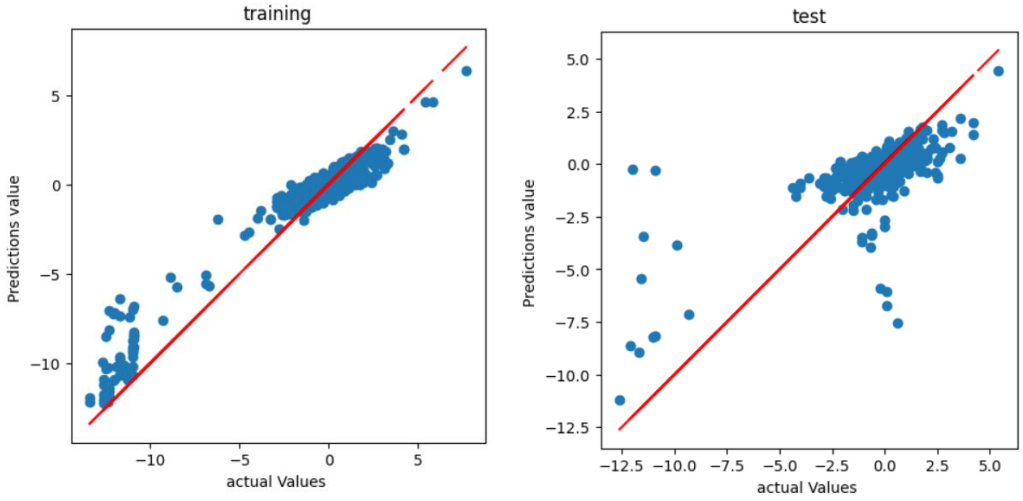
3. XGBR

```
param = {
        'eta':np.linspace(0.01, 0.2, 10),
        'gamma':np.linspace(0.001, 30, 40),
        'max_depth':range(3, 11)
}

model = XGBRegressor(objective='reg:squarederror')
best_para = Randomized_gridsearch(model, param, x_pretrain_std, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'max_depth': 10, 'gamma': 6.923846153846154, 'eta': 0.052222222222222225}
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	5.287173	0.402533	0.313040	0.559500	0.838593	0.835829
Test	11.739712	0.498022	0.671104	0.819209	0.449576	0.440149



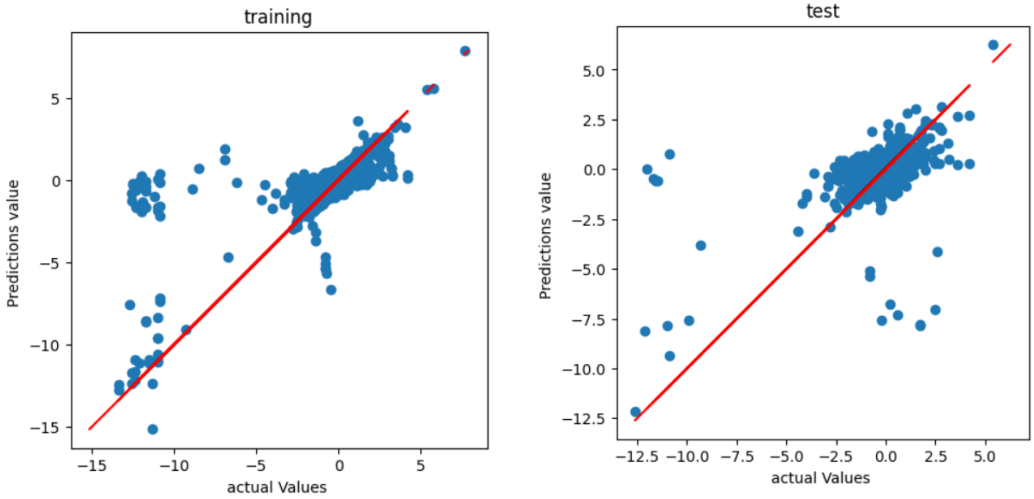
4. GBR

```
param = {
        'loss':['squared_error', 'absolute_error', 'huber', 'quantile'],
        'learning_rate':np.linspace(0.01, 2, 10),
        'n_estimators': range(1, 100, 10)
}

model = GradientBoostingRegressor()
best_para = Randomized_gridsearch(model, param, x_pretrain_std, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 71, 'loss': 'huber', 'learning_rate': 0.6733333333333333}
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	12.314288	0.385380	0.906666	0.95219	0.532513	0.524507
Test	12.003048	0.484862	0.883920	0.94017	0.275029	0.262613



◆ **Feature selection**

1. SVR

由於 SVR 當 kernel 為 rbf 的時候沒有 coef 或是 feature importence 這些 attribute，因此改使用 permutation importance 作為替代。而選出的特徵為特徵重要度 > 總特徵重要度平均值 * 0.75。共 32 個

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

perm = permutation_importance(permut_model, x_pretrain_std, y_train, n_repeats = 5, scoring = "r2", random_state=0)
```

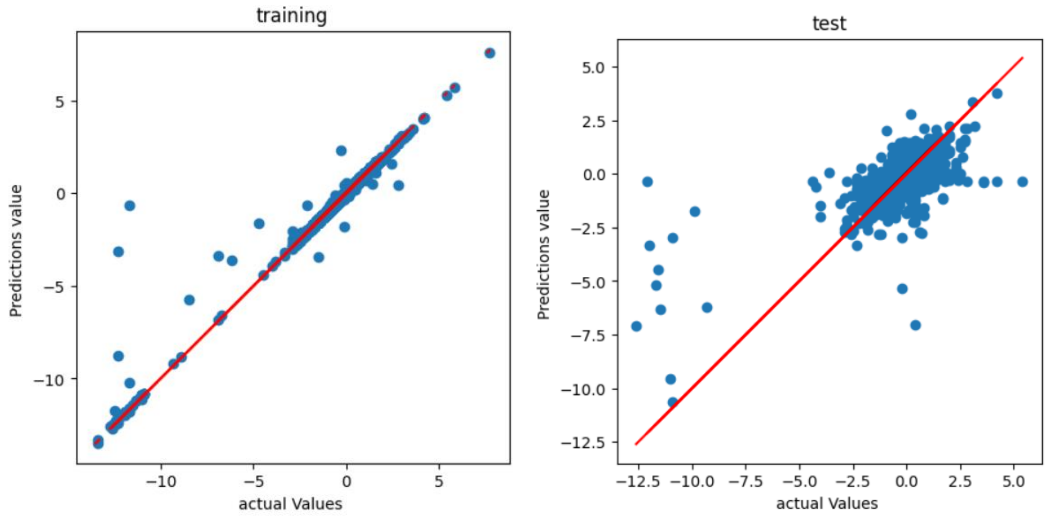
```
important_features=[]
for i in perm.importances_mean.argsort()[::-1]:
    if perm.importances_mean[i] > perm.importances_mean.mean() * 0.75:
        important_features.append(x_pretrain_std.columns[i])
```

```
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 = SVR()
best_para = Randomized_gridsearch(model, param_grid, svr_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'kernel': 'rbf', 'gamma': 0.03125, 'C': 128}
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	11.069954	0.099536	0.060557	0.246082	0.968776	0.968424
Test	11.759248	0.427039	0.626034	0.791223	0.486541	0.480745



2. ElasticNet

ElasticNet 的部分我實驗了 3 種結果都不好。原先以為是因為 RFECV 的問題，因此改採用 embedd 的方式，但結果也是不好，想說有可能是因為一開始統計方式所挑掉的特徵可能有對於 ElasticNet 有益的特徵，又想說 ElasticNet 建模速度快，因此又實驗了利用全部特徵再加上 embedd 的方式。但結果還是不好，推測有可能是 ElasticNet 不是適合用來做這個題目。(或是我做不好)

RFECV

```
elastic_cv = linear_model.ElasticNetCV(cv= 5, random_state = 0, l1_ratio = 0.1)
elastic_cv.fit(x_pretrain_std, y_train)

print(elastic_cv.alpha_)
```

```
0.15274812528953316
```

```
en = elastic_model = linear_model.ElasticNet(alpha=0.15274812528953316, l1_ratio=0.1)
n_feature, selected = rfecv(en, x_pretrain_std, y_train)
```

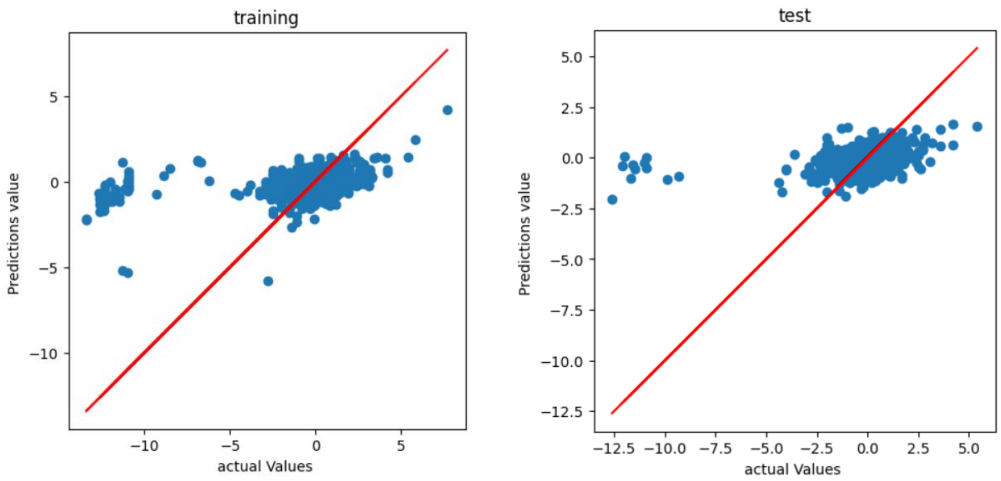
```
Optimal number of features : 45
```

```
elastic_cv = linear_model.ElasticNetCV(alphas = np.linspace(0.001, 1, 200), cv= 5, random_state = 0, l1_ratio = np.linspace(0.001, 1, 100))
elastic_cv.fit(en_x_train_std, y_train)
```

找出的最佳超參數為: alpha = 0.1817、L1 = 0.001

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	12.461932	0.646600	1.714584	1.309421	0.115942	0.108600
Test	12.052008	0.603042	1.083056	1.040700	0.111702	0.104326



embedded

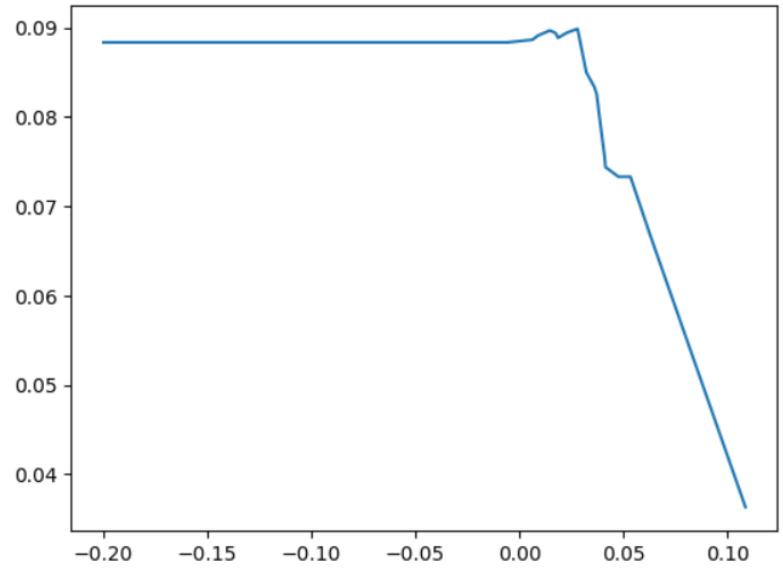
```
elastic_cv = linear_model.ElasticNetCV(cv= 5, random_state = 0)
elastic_cv.fit(x_pretrain_std, y_train)

print(elastic_cv.alpha_)
```

```
0.032757361594298505
```

```
elastic_model = linear_model.ElasticNet(alpha=0.032757361594298505)
elastic_model.fit(x_pretrain_std, y_train)

threshold = []
for i in elastic_model.coef_:
    if(i != 0): threshold.append(i)

preselected = np.array(elastic_model.coef_[:] != 0)
en_train_predata = train_predata.loc[:, preselected]
en_test_predata = test_predata.loc[:, preselected]
threshold.sort()
```

```
best_threshold = fs_embedded(elastic_model, threshold, en_x_pretrain_std, y_train)
```

```
0.02804560665335097
0.0898475240533797
```

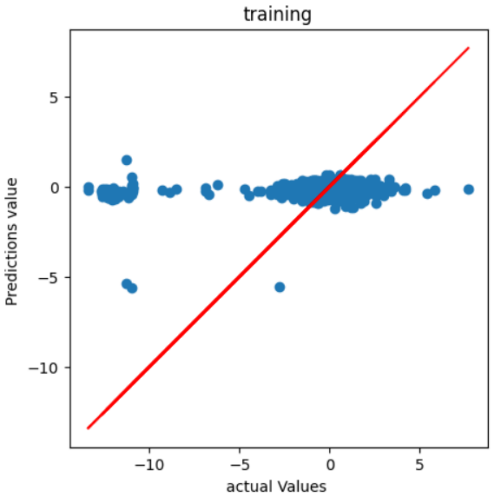


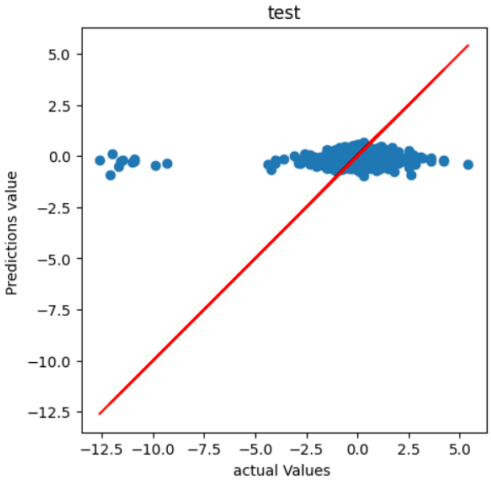

```
l1_ratio = np.linspace(0.001, 1, 200)
best_score = 0
op_al = 0
op_l1 = 0

for j in l1_ratio:
    elastic_cv = linear_model.ElasticNetCV(cv= 5, random_state = 0, l1_ratio = j)
    elastic_cv.fit(en_x_train_std, y_train)

    if(elastic_cv.score(en_x_train_std, y_train) > best_score):
        op_al = elastic_cv.alpha_
        op_l1 = j
        best_score = elastic_cv.score(en_x_train_std, y_train)

# print(elastic_cv.alpha_)
# print(elastic_cv.l1_ratio_)
print(op_al)
print(op_l1)
```

找出的最佳超參數為: alpha = 0.00886、L1 = 1.0

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	13.412039	0.680154	1.879298	1.370875	0.031013	0.028881
Test	12.406036	0.636402	1.205265	1.097846	0.011469	0.009293





origion

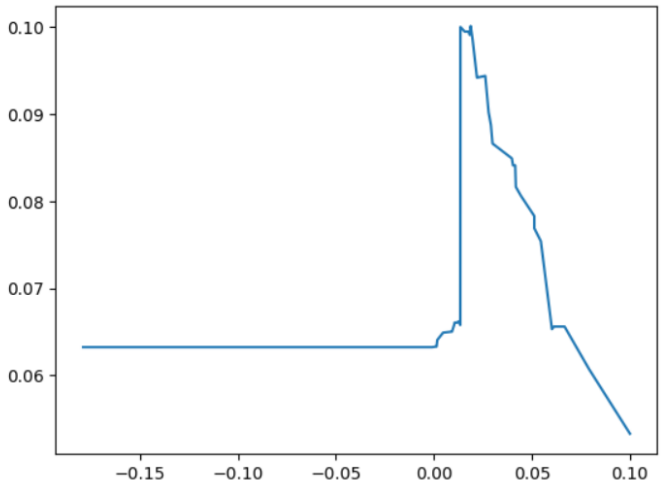
```
elastic_cv = linear_model.ElasticNetCV(cv= 5, random_state = 0)
elastic_cv.fit(X_train_Std, y_train)

print(elastic_cv.alpha_)
```

```
0.037663005643817846
```

```
elastic_model = linear_model.ElasticNet(alpha = 0.037663005643817846)
elastic_model.fit(X_train_Std, y_train)

threshold = []
for i in elastic_model.coef_:
    if(i != 0): threshold.append(i)

preselected = np.array(elastic_model.coef_[:] != 0)
en_train_predata = X_train.loc[:, preselected]
en_test_predata = X_test.loc[:, preselected]
threshold.sort()
```

```
best_threshold = fs_embedded(elastic_model, threshold, en_x_pretrain_std, y_train)
```

```
0.01876718925875847
0.10008039816438385
```

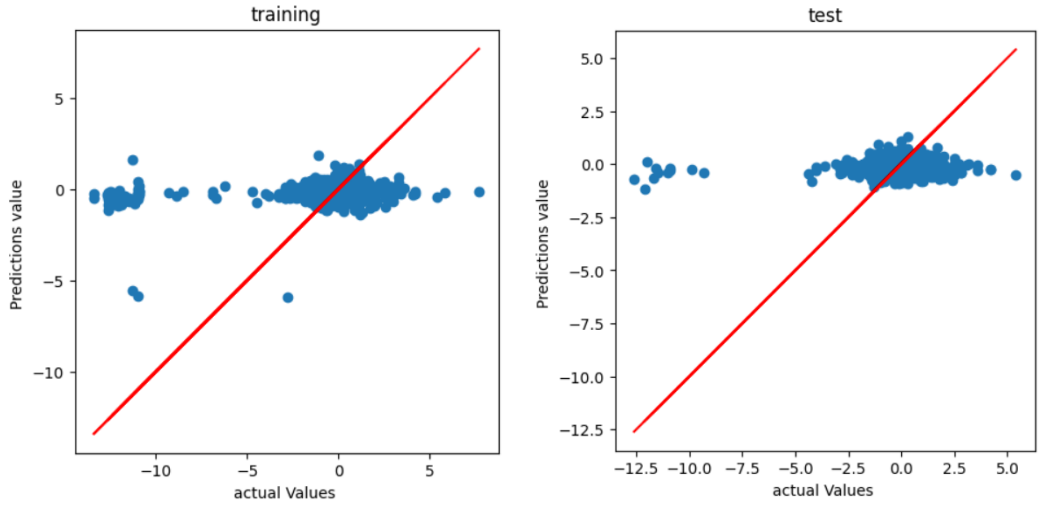


```
l1_ratio = np.linspace(0.0001, 1, 200)
best_score = 0
alpha = 0
best_l1_ratio = 0

for i in l1_ratio:
    elastic_cv = linear_model.ElasticNetCV(cv= 5, random_state = 0, l1_ratio = i)
    elastic_cv.fit(en_x_train_std, y_train)
    score = elastic_cv.score(en_x_train_std, y_train)

    if(score > best_score):
        alpha = elastic_cv.alpha_
        best_l1_ratio = i
        best_score = score

print(alpha)
print(best_l1_ratio)
```

找出的最佳超參數為: alpha = 0.00333、L1 = 1.0

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	13.150894	0.683562	1.862345	1.364678	0.039754	0.035872
Test	12.102527	0.646143	1.215415	1.102459	0.003144	-0.000886



結果反而更差....

3. XGBR

```
xgbr = XGBRegressor(objective='reg:squarederror')
n_feature, selected = rfecv(xgbr, x_pretrain_std, y_train)
```

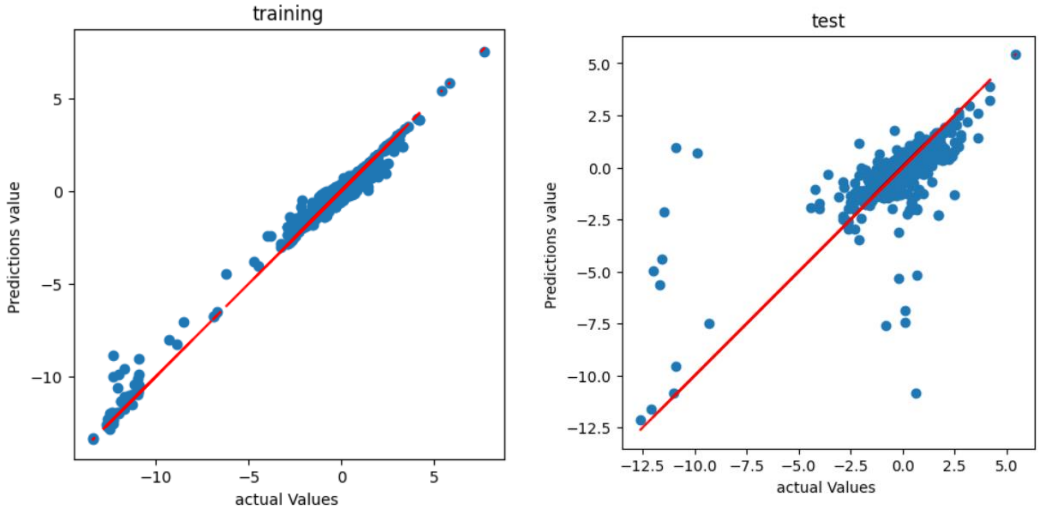
```
Optimal number of features : 14
```

```
param = {
        'eta':np.linspace(0.01, 0.2, 10),
        'gamma':np.linspace(0.001, 30, 40),
        'max_depth':range(3, 11)
}

xgbr = XGBRegressor(objective='reg:squarederror')
best_para = Randomized_gridsearch(xgbr, param, xgbr_train_std, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'max_depth': 9, 'gamma': 0.001, 'eta': 0.1366666666666667}
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	3.419318	0.163887	0.057592	0.239984	0.970305	0.970228
Test	11.846824	0.353341	0.590619	0.768517	0.515588	0.514343



4. GBR

```
gbr = GradientBoostingRegressor()
n_feature, selected = rfecv(gbr, x_pretrain_std, y_train)
```

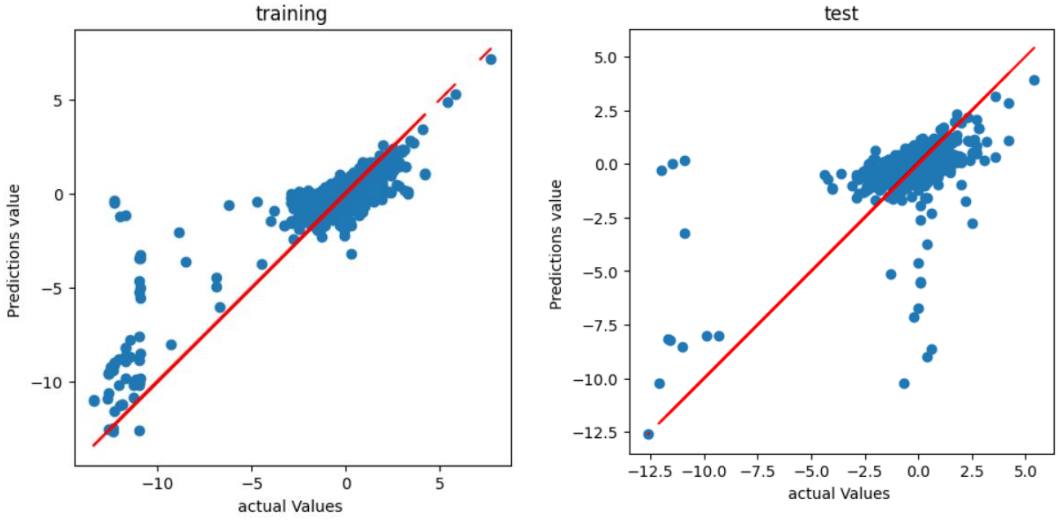
```
Optimal number of features : 37
```

```
param = {
        'loss':['squared_error', 'absolute_error', 'huber', 'quantile'],
        'learning_rate':np.linspace(0.01, 2, 10),
        'n_estimators': range(1, 100, 10)
}

gbr = GradientBoostingRegressor()
best_para = Randomized_gridsearch(gbr, param, gbr_train_std, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 41, 'loss': 'squared_error', 'learning_rate': 0.45222222222222225}
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	11.941781	0.474332	0.555250	0.745151	0.713707	0.711755
Test	11.724342	0.535945	0.855899	0.925148	0.298011	0.293225



Survival

- **資料切分**

  由於樣本數少，因此使用 ShuffleSplit 保留每個類別的樣本百分比，接著把沒有記錄到復發的病人(-1)去掉,方便統計。

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

- **特徵初選(統計方法)**

  先跑 ranksum test 算出 p-value，為了減少特徵數以防建模時間太久且也怕篩出的特徵太適合復發，以至於對於存活的預測不好，因此選用 p-value < 0.01 的特徵。選出的特徵共 366 個

```
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)
```

- **模型訓練過程**

  模型的部分，我總共實驗了 2 種模型，分別為 CoxPH 以及 ComponentwiseGradientBoostSurvival (CGBS)，由於直接使用 CoxPH 若是只使用統計方法挑出的特徵去找超參數會建模很久，因此只直接實驗有做特徵挑選的。特徵挑選我是採用模型本身的 coef，當作 threshold，藉由把 threshold 排序去一個一個把特徵消掉建模找出表現最好的特徵數。

◆ **Feature selection**

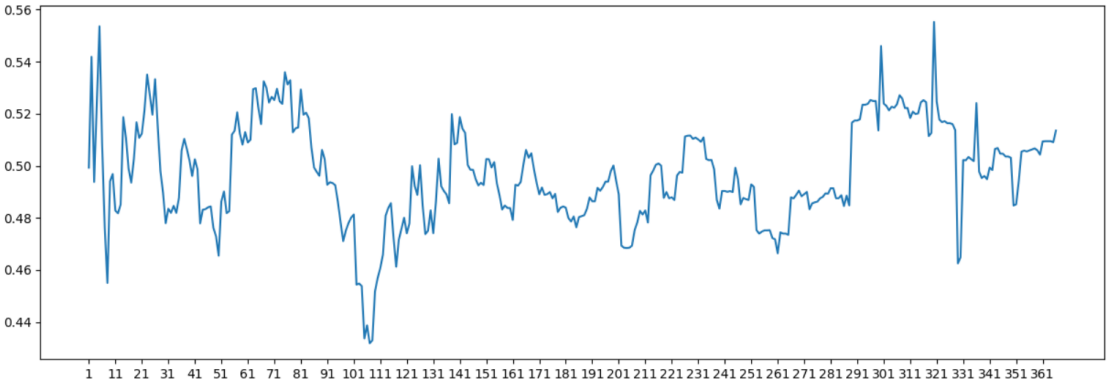
1. **CoxPH**

   由各特徵的表現圖可看出再 coef >= -3.94 時所得出的表現最好，共 320 個。

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

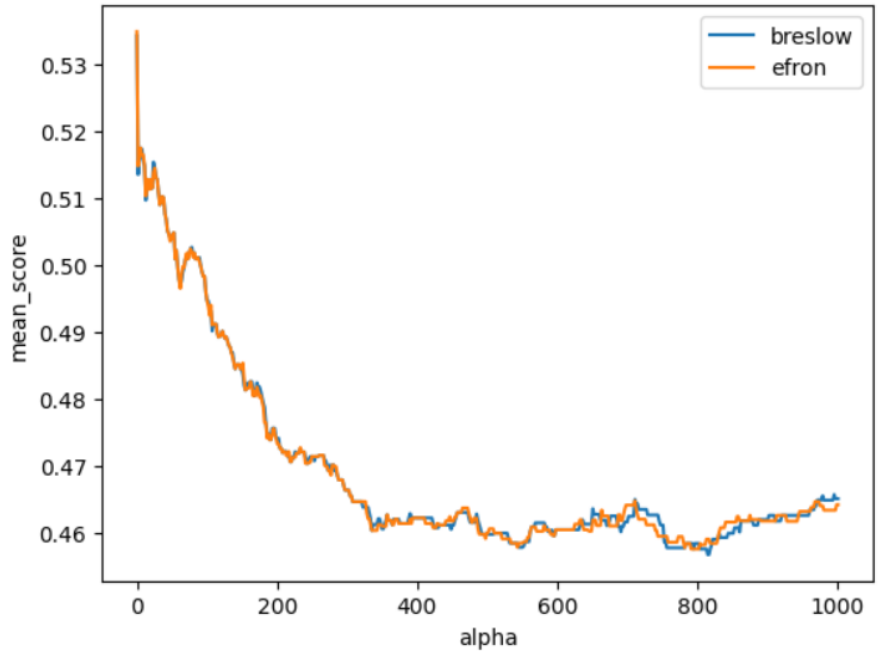
```
threshold = coxPH.coef_
threshold.sort()
```

```
best_threshold:  -3.943563151376996
best_score:  0.5553025554076294
```



以下為超參數挑選的表現圖。最後選出的超參數為: tie = efron、alpha = 0.002。

```
efron
0.002
0.53488134412984
```



結果

	C-index	C-ipcw
training	1.000000	1.000000
test	0.463842	0.434973

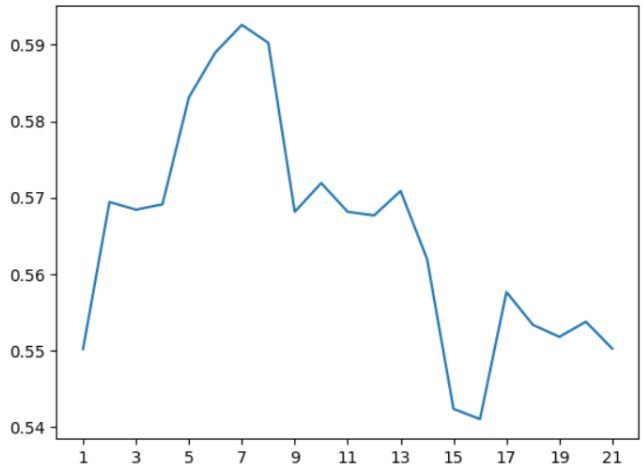
2. CGBS

```
#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()
```

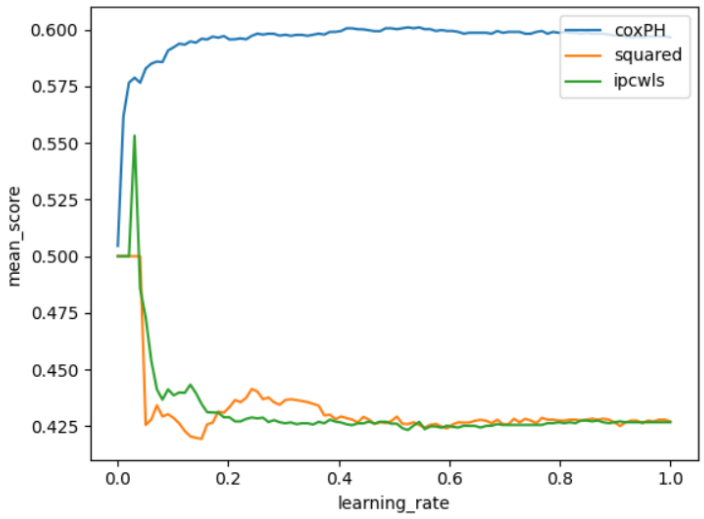
由各特徵的表現圖可看出再 coef >= 0.065 時所得出的表現最好，共 7 個。

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



以下為特徵挑選的表現圖。最後選出的超參數為: loss = coxph 、 learning rate = 0.5253。

```
loss:  coxph
learning rate:  0.5253
best score:  0.6010049890249411
```



結果

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

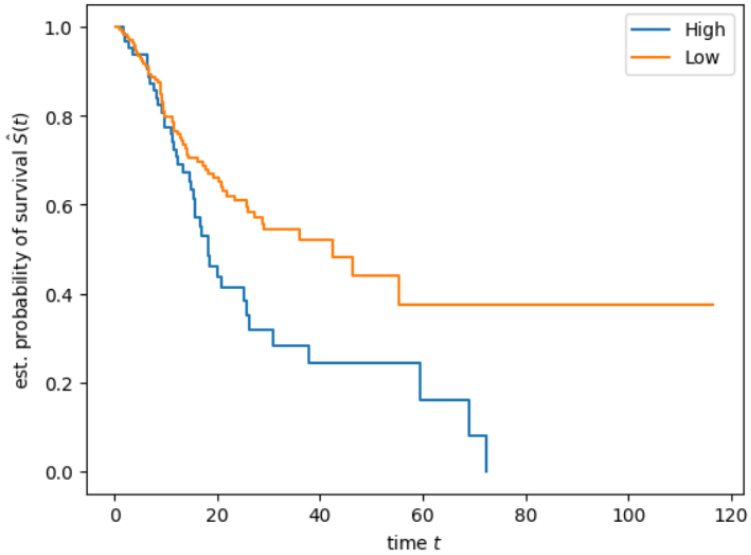
◆ KM-plot & log-rank test

由綜合表現下來，CGBS 相較於 CoxPH 較沒有 overfitting 的現象，表現較好特徵樹也少很多。因此選擇 CGBS 所選出的特徵去做 KM plot 以及 log-rank test。利用 permutation importance 做特徵排序後，看前 4 個表現好的。

	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

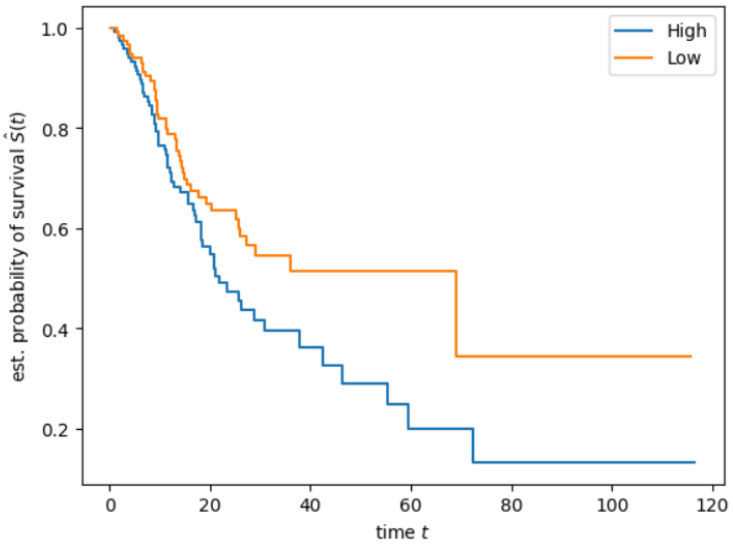
```
km_logrank('ENSG00000254622.1', x_km, y_struct)
```

```
<ipython-input-8-85c4425bbf24>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  gene.loc[i] = "High"
          test_statistic         p    -log2(p)
High Low       13.016528  0.000309  11.661256
```



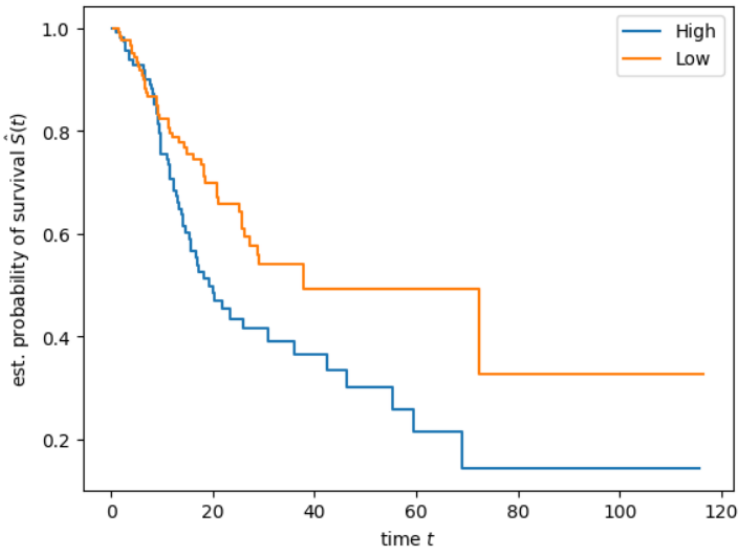
```
km_logrank('ENSG00000237975.7', x_km, y_struct)
```

```
<ipython-input-8-85c4425bbf24>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  gene.loc[i] = "High"
         test_statistic         p  -log2(p)
High Low        6.225743  0.012591  6.311508
```



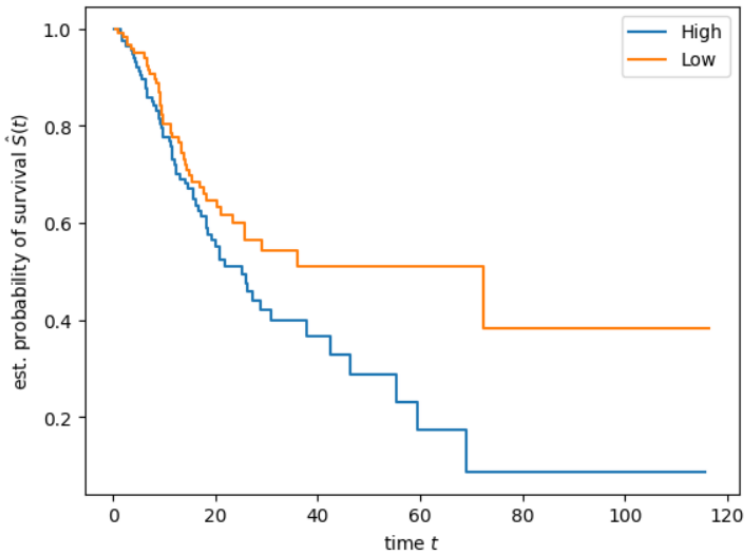

```
km_logrank('ENSG00000248469.1', x_km, y_struct)
```

```
<ipython-input-8-85c4425bbf24>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  gene.loc[i] = "Low"
         test_statistic         p  -log2(p)
High Low        4.840857  0.027793  5.169131
```



```
km_logrank('ENSG00000223466.2', x_km, y_struct)
```

```
<ipython-input-8-85c4425bbf24>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
  gene.loc[i] = "High"
          test_statistic         p  -log2(p)
High Low         5.00974  0.025205  5.310141
```



結論

Regression

以下各個模型的 test 資料及的 R2，以及 MSE 的比較表格(Elastic 採表現最好的那次):

R2	**SVR**	**ElasticNet**	**XGBR**	**GBR**
Statistic only (92 feature)	0.230	0.112	0.449	0.275
進一步特徵挑選	0.486	0.111	0.515	0.298
MSE				
Statistic only (92 feature)	0.938	1.08	0.671	0.883
進一步特徵挑選	0.626	1.08	0.590	0.855
進一步挑選的特徵數	32	RFECV: 45 Sta + Emb: 36 Ori + Emb: 58	14	37

由上述表現圖可看出，XGBR 在這次的題目中不論是 R2 或是 MSE 都表現得相較於其他模型好，且所選出的特徵也較少，最後所選的模型也是 XGBR。而其餘模型的表現大概是 SVR > GBR > ElasticNet。但就所有模型來看，除了 ElasticNet 再 train 以及 test 表現都不好外，其餘的模型都有挺嚴重的 overtraining 情形。且由預測 vs 實際的圖中看出在實際值 < 0 的預測都不是很好。我在想是因為一開始再做統計初篩的時候可能

做得不好，應該有更好的選擇方式，用 pearson 所選出來的特徵似乎不適合這個題目。

Survival

以下為個模型的 train 以及 test 的 C-index。

	CoxPH	**CGBS**
Train	1.0	0.612
test	0.463	0.647

由上面的表格來看，CoxPH 有明顯 overtraining 的現象，而 CGBS 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 的表現不如 CoxPH，但是 test 可以也可以表現得不錯，泛化能力較好。而我認為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也是造成 overtraining 或是表現不好的一個很大原因。而相較於 CoxPH，CGBS 表現較穩定且較好，因此最後選擇這個模型。

而為了看這些 lncRNA 和存活的相關程度，是先將基因分成高表現以及低表現再做了 KM plot 還有 logrank test。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。

上傳模型名稱以及紀錄

- 模型名稱: regression.joblib 、 survival.joblib

- 對應訓練紀錄:
  詳細的訓練紀錄:
  - Regression: XGBR with feature selection
  - Survuval: CGBS

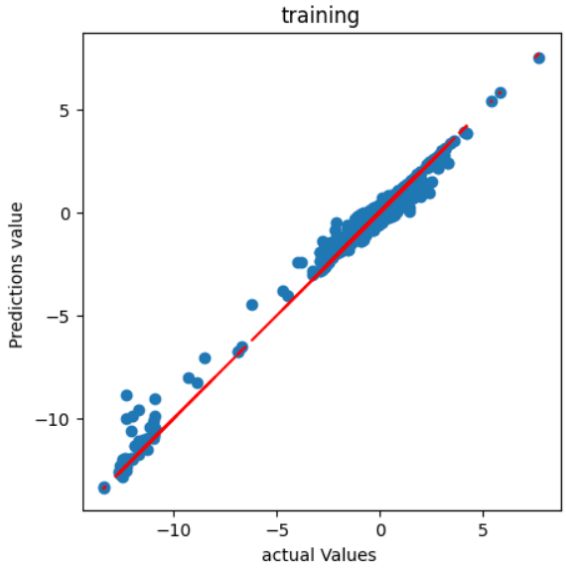
加分

這次多實驗了 3 個模型: XGBoostRegressor、GradientBoostRegressor、ComponentwiseGradientBoostSurvival。

而以下是 prediction vs. label 的程式碼

```
def actual_vs_predict(tr_act, tr_pre, te_act, te_pre):
    #training
    plt.figure(figsize=(5, 5))
    plt.scatter(tr_act, tr_pre)
    plt.plot([tr_act, tr_pre],[tr_act, tr_pre], 'r-')
    plt.xlabel('actual Values ')
    plt.ylabel('Predictions value ')
    plt.axis('equal')
    plt.axis('square')
    plt.title('training')

    #test
    plt.figure(figsize=(5, 5))
    plt.scatter(te_act, te_pre)
    plt.plot([te_act, te_pre],[te_act, te_pre], 'r-')
    plt.xlabel('actual Values ')
    plt.ylabel('Predictions value ')
    plt.axis('equal')
    plt.axis('square')
    plt.title('test')
```

得出的結果會長得像這樣:



Package & function

以下為一些 package 以及自訂的 function，其餘像是一些挑超參數的程式碼或是做特徵挑選的程式碼因為較占空間因此就放在 2 個 ipynb 檔中，不特別呈現。

Regression

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE, RFECV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectFromModel
from sklearn import preprocessing
import math
from sklearn.inspection import permutation_importance
#-----------------model-----------------------
from sklearn.svm import SVR
from sklearn import linear_model
from xgboost.sklearn import XGBRegressor
from sklearn.ensemble import GradientBoostingRegressor
#---------------------------------------------
from sklearn.metrics import explained_variance_score, mean_absolute_error, mean_squa
```

```
def rfecv(model, x_train, y_train):
    r = RFECV(estimator = model, step = 1, cv = 5, scoring = 'r2').fit(x_tra
    print("Optimal number of features : %d" % r.n_features_)
    print("Support : %s" % r.support_)
    print("Ranking : %s" % r.ranking_)

    return r.n_features_, r.support_
```

```
def fs_embedded(model, threshold, x_train, y_train):
    score = []
    best_threshold = 0
    best_score = 0
    for i in threshold:
        x_embedded = SelectFromModel(model, threshold = i).fit_transform(x_train, y
        mean_score = cross_val_score(model, x_embedded, y_train, cv = 5, scoring
        score.append(mean_score)
        if(mean_score > best_score):
            best_score = mean_score
            best_threshold = i

    print(best_threshold)
    print(best_score)
    plt.plot(threshold, score)
    plt.show()

    return best_threshold
```

```
def regression_model_evaluation_result(train_X, train_Y, test_X, test_Y, model):
    # Training and Prediction
    model.fit(train_X, train_Y)
    train_pred = model.predict(train_X)


    # Evaluation: Training
    size, var_num = train_X.shape
    train_R2 = r2_score(train_Y, train_pred)
    train_adj_R2 = 1-(1-train_R2)*(size-1)/(size-var_num-1)
    train_MAE = mean_absolute_error(train_Y, train_pred)
    train_MSE = mean_squared_error(train_Y, train_pred)
    train_RMSE = (train_MSE ** 0.5)
    train_Max_error = max_error(train_Y, train_pred)
    svr_train = pd.DataFrame(data = {"actual values":train_Y, "predicted values":train_pred})

    # Prediction: Test
    test_pred = model.predict(test_X)

    # Evaluation: Test
    test_R2 = r2_score(test_Y, test_pred)
    test_adj_R2 = 1-(1-test_R2)*(size-1)/(size-var_num-1)
    test_MAE = mean_absolute_error(test_Y, test_pred)
    test_MSE = mean_squared_error(test_Y, test_pred)
    test_RMSE = (test_MSE ** 0.5)
    test_Max_error = max_error(test_Y, test_pred)
    svr_test = pd.DataFrame(data = {"actual values":test_Y, "predicted values":test_pred})

    # Summary
    result = [{"Max error":train_Max_error, "MAE":train_MAE, "MSE":train_MSE, "RMSE":train_RMSE, "R2":train_R2, "Adjusted R2": train_adj_R2},
                        {"Max error":test_Max_error, "MAE":test_MAE, "MSE":test_MSE, "RMSE":test_RMSE, "R2":test_R2, "Adjusted R2": test_adj_R2}]

    summary = pd.DataFrame(result)
    summary.index = ["Training", "Test"]
```

```
def actual_vs_predict(tr_act, tr_pre, te_act, te_pre):
    #training
    plt.figure(figsize=(5, 5))
    plt.scatter(tr_act, tr_pre)
    plt.plot([tr_act, tr_pre],[tr_act, tr_pre], 'r-')
    plt.xlabel('actual Values ')
    plt.ylabel('Predictions value ')
    plt.axis('equal')
    plt.axis('square')
    plt.title('training')

    #test
    plt.figure(figsize=(5, 5))
    plt.scatter(te_act, te_pre)
    plt.plot([te_act, te_pre],[te_act, te_pre], 'r-')
    plt.xlabel('actual Values ')
    plt.ylabel('Predictions value ')
    plt.axis('equal')
    plt.axis('square')
    plt.title('test')
```

```
def Randomized_gridsearch(model, param_grid_, x_train, y_train_):
    optimal_params = RandomizedSearchCV(
            model,
            param_grid_,
            cv = 5,
            scoring = 'r2',
            verbose = 1,
            n_jobs = -1,
        )

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

Survival

```
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import ranksums
from sklearn.feature_selection import RFECV
#-----------------model for survival---------------------------------
from sksurv.linear_model import CoxPHSurvivalAnalysis
from sksurv.ensemble import GradientBoostingSurvivalAnalysis
from sksurv.ensemble import ComponentwiseGradientBoostingSurvivalAnalysis
from sksurv.ensemble import RandomSurvivalForest
#--------------------------------------------------------------------
from sklearn.inspection import permutation_importance
import statistics
from sksurv.nonparametric import kaplan_meier_estimator
from lifelines.statistics import pairwise_logrank_test
from sklearn.metrics import make_scorer
from sksurv.metrics import concordance_index_censored,concordance_index_ipcw
from sklearn.preprocessing import StandardScaler
```

```
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)
```

```
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 沒有支援 c-index 的 cross validation，所以就自己做了一個

```
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
```

```
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
```

```
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")
```