Regression

- **Package & function**

```
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
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
import math
#-----------------model-----------------------
from sklearn.svm import SVR
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from xgboost.sklearn import XGBRegressor
from sklearn.ensemble import GradientBoostingRegressor
#---------------------------------------------
from sklearn.metrics import explained_variance_score, mean_absolute_error, mean_squared_error, max_error, r2_score
```

```
def rfecv(model, x_train, y_train):
    r = RFECV(estimator = model, step = 1, cv = 3, scoring = 'neg_mean_squared_error').fit(x_train, y_train)
    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_
```

```
[151] def grid_search(model, param_grid, x_train, y_train):
          optimal_paramas = GridSearchCV(
                  model,
                  param_grid,
                  cv = 3,
                  scoring = 'neg_mean_squared_error',
                  verbose = 1,
                  n_jobs = -1
          )
          optimal_paramas.fit(x_train, y_train)
          print(optimal_paramas.best_params_)
```

```python
def randomsearchCV(model, param, x_train, y_train):
    optimal_paramas = RandomizedSearchCV(
            model,
            param,
            cv = 3,
            scoring = 'neg_mean_squared_error',
            verbose = 1,
            n_iter = 50,
            n_jobs = -1
    )
    optimal_paramas.fit(x_train, y_train)
    print(optimal_paramas.best_params_)
```

評分方式是採最好的 MSE

*雖然 randomizedsearchCV 不一定歷遍每一種參數組合，但輸入連續變數時會將其當作一個分布進行採樣這是 grid search 做不到的，且可以透過變更搜尋次數(n_iter)控制計算量，速度也較快

```python
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_train)
        mean_score = cross_val_score(model, x_embedded, y_train, cv = 5).mean()
        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()
```

- **Data splitting**

```python
[56] train_df, test_df = train_test_split(df, test_size = 0.3, random_state = 0)
```

切分資料的方式為 train : test = 7 : 3

- **特徵預選**

利用 pearson 相關係數先把較不相關的特徵去除

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

     # annot = True 讓我們可以把數字標進每個格子裡
     sns.heatmap(data=correlation_matrix, square = True, annot = True)

     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)
```

為了保留大部分的特徵，因此選擇絕對值大於 0.005 的特徵留下來

```
prefeature = []

for i in column_and_corr.iloc:
    if(abs(i['pearson']) >= 0.005 and abs(i['pearson']) != 1): prefeature.append(i['features'])

print(prefeature)
print(len(prefeature))
```

```
['d1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'd12', 'd13', 'd14', 'd15', 'd17', 'd18', 'd20',
68
```

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

print(train_predata)
print(test_predata)
```

留下的特徵數為 ['d1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'd12', 'd13', 'd14', 'd15', 'd17', 'd18', 'd20', 'd21', 'd22', 'd27', 'd28', 'd29', 'd31', 'd34', 'd35', 'd36', 'd37', 'd38', 'd39', 'c1', 'c2', 'c3', 'c4', 'c6', 'c7', 'c8', 'c9', 'c10', 'c12', 'c13', 'c14', 'c15', 'c16', 'c17', 'c18', 'c19', 'c20', 'c21', 'c22', 'c23', 'c25', 'c28', 'c29', 'c30', 'c31', 'c32', 'c33', 'c34', 'c35', 'c36', 'c37', 'c38', 'c39', 'c43', 'c44', 'c45', 'c46', 'c49', 'c51']
共 68 個

將這些 feature data 標準化用於之後的特徵挑選

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

scaler_std = preprocessing.StandardScaler().fit(train_predata)
x_pretrain_std = scaler_std.transform(train_predata)
x_pretest_std = scaler_std.transform(test_predata)
```

- **Training model**

SVR

A. Feature selection

先利用 RFECV 進一步把特徵選出

```
svr = SVR(kernel = 'linear')
n_feature, selected = rfecv(svr, x_pretrain_std, y_train)
```

```
Optimal number of features : 47
Support : [False  True False False  True False  True  True False False  True  True
  True  True  True  True  True  True  True  True False  True  True  True
  True False  True  True  True  True  True  True  True False False  True
  True False  True  True  True  True  True  True False  True  True False
  True False  True  True  True False False  True  True False  True  True
  True False False  True False False  True  True]
Ranking : [22  1 21 11  1 20  1  1 14 17  1  1  1  1  1  1  1  1  1  1  3  1  1  1
  1 19  1  1  1  1  1  1  1  6 10  1  1 12  1  1  1  1  1  1  5  1  1  7
  1  4  1  1  1  9 13  1  1 18  1  1  1 15  8  1  2 16  1  1]
```

```
svr_train_data = train_predata.loc[:, selected]
svr_test_data = test_predata.loc[:, selected]

print(svr_train_data)
```

選出來的特徵為: ['d2', 'd5', 'd7', 'd8', 'd13', 'd14','d15', 'd17', 'd18', 'd20', 'd21', 'd22', 'd27', 'd28', 'd31', 'd34', 'd35', 'd36', 'd38', 'd39', 'c1', 'c2', 'c3', 'c4', 'c6', c9', 'c10', 'c13', 'c14', 'c15', 'c16', 'c17', 'c18', 'c20', 'c21', 'c23', 'c28', 'c29', 'c30', 'c33', 'c34', 'c36', 'c37', 'c38', 'c44', 'c49', 'c51']

共 47 個

B. 資料標準化

將要選出的特徵標準化

```
[36] svr_scaler_std = preprocessing.StandardScaler().fit(svr_train_data)
     svr_x_train_std = svr_scaler_std.transform(svr_train_data)
     svr_x_test_std = svr_scaler_std.transform(svr_test_data)
```

C. 利用 randomized searchCV 找最佳超參數

```
[18] param = {
             'kernel': ['rbf'],
             'C': [14, 16, 18, 20, 22],
             'gamma': [0.01, 0.1, 0.125, 0.15, 1]
     }

     model = SVR()
     randomsearchCV(model, param, svr_x_train_std, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'kernel': 'rbf', 'gamma': 0.01, 'C': 22}
```

結果顯示為 C = 22 、 gamma = '0.01'，因為 C 在邊界，因此再做一次，且在更細找 gamma

```
param = {
        'kernel': ['rbf'],
        'C': [20, 22, 24, 25, 26],
        'gamma': [0.01, 0.015, 0.025, 0.05, 0.1]
}

model = SVR()
randomsearchCV(model, param, svr_x_train_std, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
{'kernel': 'rbf', 'gamma': 0.015, 'C': 26}
```

這次 C = 26 、gamma = 0.015， C 還是在邊界，所以再做一次

```
param = {
        'kernel': ['rbf'],
        'C': range(26, 36, 2),
        'gamma': [0.015]
}

model = SVR()
randomsearchCV(model, param, svr_x_train_std, y_train)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
/usr/local/lib/python3.7/dist-packages/sklearn/model_select
  UserWarning,
{'kernel': 'rbf', 'gamma': 0.015, 'C': 32}
```

最後採的結果為: kernel = rbf, C = 32, gamma = 0.015

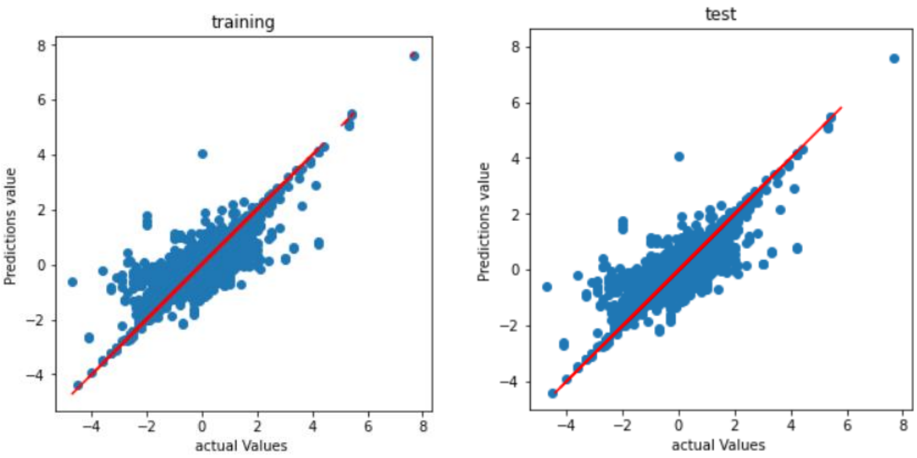
D. Training model

```
svr_model=SVR(kernel = 'rbf',C = 32, gamma=0.015)
df_svr, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = regression_model_evaluation_result(svr_x_train_std, y_train, svr_x_test_std, y_test, svr_model)
```

其結果為:

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	4.094848	0.251255	0.169342	0.411512	0.745184	0.743506
Test	4.724600	0.376903	0.304595	0.551901	0.561360	0.558472

實際值對預測值的分布圖



ElasticNet Regression

ElasticNet 我做了很多次，但結果都很不好，以下是我主要的 3 次訓練

A. 第一次

我先用 RFECV 做特徵挑選

```
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_)
```

```
0.009241791597637293
```

```
en = elastic_model = linear_model.ElasticNet(alpha=0.009241791597637293,l1_ratio=0.1)
n_feature, selected = rfecv(en, x_pretrain_std, y_train)
```

```
Optimal number of features : 41
Support : [ True False False False False  True False False False False  True  True
  True  True  True False  True  True False  True  True  True  True  True
 False False  True False  True False  True False  True False False  True
  True  True  True  True  True  True  True  True False  True  True False
 False False  True  True  True  True  True  True  True  True False  True
 False  True  True  True False False False False]
Ranking : [ 1  6 13 23 18  1 22  5 21 16  1  1  1  1  1  9  1  1 14  1  1  1  1  1
 28  2  1 26  1 25  1 12  1  4 27  1  1  1  1  1  1  1  1  1 11  1  1 20
  7 17  1  1  1  1  1  1  1  1  8  1  3  1  1  1 10 19 24 15]
```

選出的特徵為:

['d1','d6','d13','d14','d15','d17','d18','d21','d22','d28', 'd29', 'd31', 'd34', 'd35', 'd38', 'c1', 'c3','c6','c9', 'c10', 'c12','c13', 'c14', 'c15', 'c16','c17','c18','c20', 'c21','c28,'c29','c30', 'c31', 'c32', 'c33', 'c34', 'c35', 'c37', 'c39', 'c43', 'c44']

共 41 個

接著把資料取出並標準化

```
en_train_data = train_predata.loc[:, selected]
en_test_data = test_predata.loc[:, selected]

print(en_train_data)
```

```
en_scaler_std = preprocessing.StandardScaler().fit(en_train_data)
en_x_train_std = en_scaler_std.transform(en_train_data)
en_x_test_std = en_scaler_std.transform(en_test_data)
```

找適合的 alpha 以及 l1_ratio

```
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)

print(elastic_cv.alpha_)
print(elastic_cv.l1_ratio_)
```

```
0.006020100502512563
0.001
```

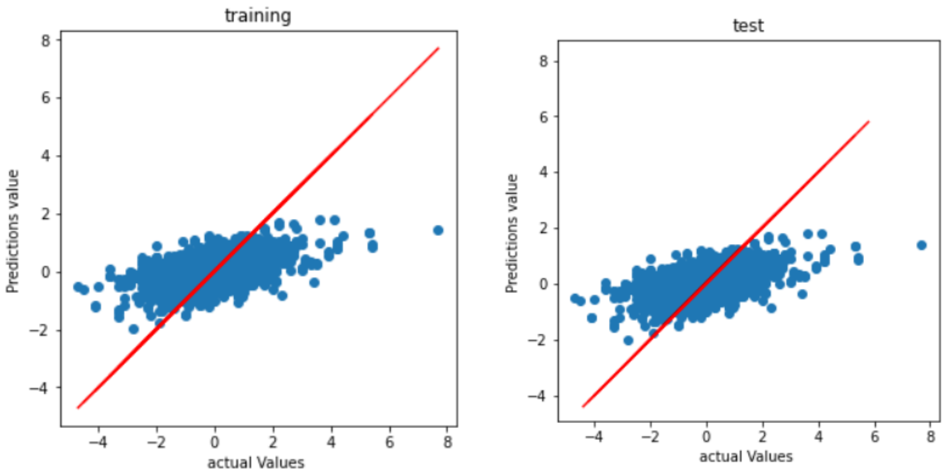
Alpha = 0.006020100502512563 以及 l1_ratio = 0.001

接著訓練 model

```
elastic_model = linear_model.ElasticNet(alpha=0.006020100502512563,l1_ratio=0.001)
df_svr, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = regression_model_evaluation_result(en_x_train_std, y_train, en_x_test_std, y_test, elastic_model)
actual_vs_predict(y_train, tr_predlabel, y_test, te_predlabel)
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	6.275080	0.530115	0.519546	0.720795	0.218219	0.213733
Test	4.919666	0.543145	0.542386	0.736469	0.218921	0.214440



由 R2 結果可知擬合效果非常的差(< 0.5)，且誤差也頗大，因此我想說使用 embedded 的方式來做 feature selection 看看

B. 第二次

先找大約的 alpha 值

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

print(elastic_cv.alpha_)
```

```
0.003713782536160237
```

Alpha = 0.003713782536160237

接著把 coef 為 0 的特徵去除，並且標準化

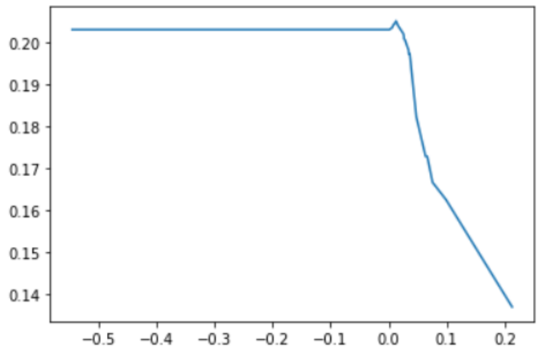
```
elastic_model = linear_model.ElasticNet(alpha=0.003713782536160237)
elastic_model.fit(x_pretrain_std, y_train)

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()

print(en_train_predata)
```

```
en_scaler_std = preprocessing.StandardScaler().fit(en_train_predata)
en_x_pretrain_std = en_scaler_std.transform(en_train_predata)
en_x_pretest_std = en_scaler_std.transform(en_test_predata)
```

畫學習曲線，更進一步找出最終分數最高時的 coef 閥值

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

```
0.012663678039826843
0.20527094606474652
```



可知 threshold = 0.012663678039826843，有最高的分數

取出資料，再標準化作為我們的訓練 data

```
selected = np.array(elastic_model.coef_[:] >= 0.012663678039826843)

# print(preselected)
# print(selected)

en_train_data = train_predata.loc[:, selected]
en_test_data = test_predata.loc[:, selected]

print(en_train_data)
print(en_train_data.columns)

en_scaler_std = preprocessing.StandardScaler().fit(en_train_data)
en_x_train_std = en_scaler_std.transform(en_train_data)
en_x_test_std = en_scaler_std.transform(en_test_data)
```

接著更細的找出 alpha 以及 l1_ratio

```
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)
```

```
0.056382243314452726
0.006020100502512563
```

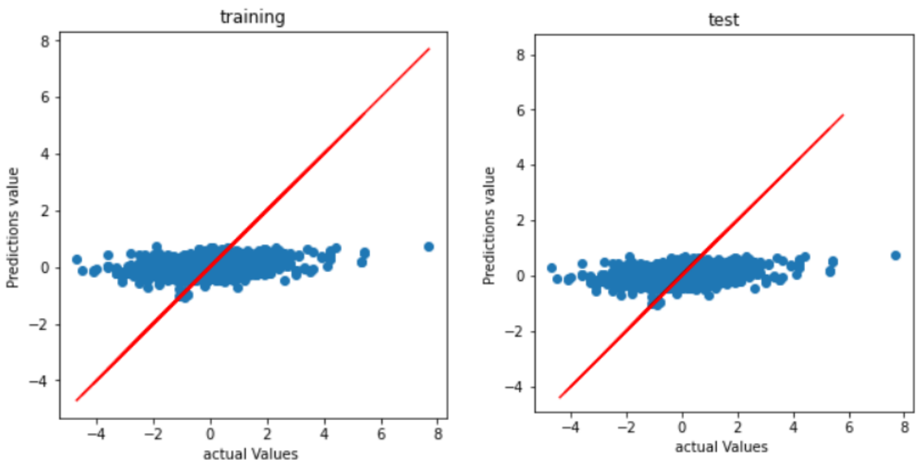
Alpha = 0.056382243314452726、l1_ratio = 0.006020100502512563

訓練模型

```
elastic_model = linear_model.ElasticNet(alpha=0.056382243314452726,l1_ratio=0.006020100502512563)
df_svr, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = regression_model_evaluation_result(en_x_train_std, y_train, en_x_test_std, y_test, elastic_model)
actual_vs_predict(y_train, tr_predlabel, y_test, te_predlabel)
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	6.971826	0.575437	0.625859	0.791113	0.058245	0.055880
Test	5.247022	0.583615	0.648949	0.805574	0.065462	0.063116



結果更差了…。 因此我想是不是我在做 corr 時刪除太多的特徵，以至於把一些可能會有用的特徵也去除了，所以我用了全部的特徵再去做 embedded 試看看

C. 第三次

其過程和第 2 次一樣

找大約的 alpha

```
] X_train = train_df.iloc[:,0:-1]
  X_test = test_df.iloc[:,0:-1]

  scaler_Std = preprocessing.StandardScaler().fit(X_train)
  X_train_Std = scaler_Std.transform(X_train)
  X_test_Std = scaler_Std.transform(X_test)
```

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

  print(elastic_cv.alpha_)

  0.0018483583195274581
```

接著把 coef 為 0 的去除

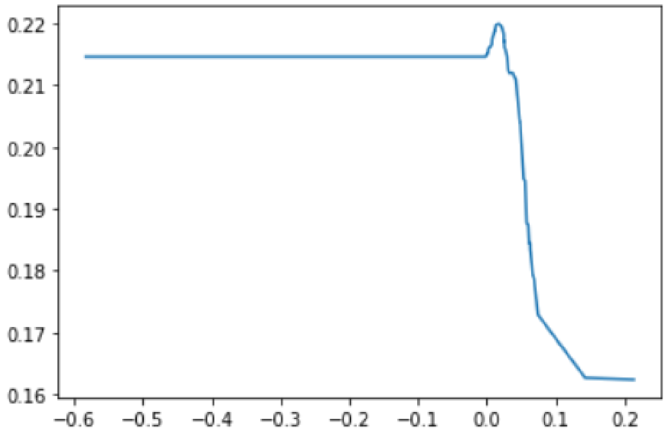
```
elastic_model = linear_model.ElasticNet(alpha=0.0018483583195274581)
elastic_model.fit(X_train_Std, y_train)

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()

print(en_train_predata)
```

```
en_scaler_std = preprocessing.StandardScaler().fit(en_train_predata)
en_x_pretrain_std = en_scaler_std.transform(en_train_predata)
en_x_pretest_std = en_scaler_std.transform(en_test_predata)

print(en_x_pretrain_std)
```

畫學習曲線以進一步找特徵

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

```
0.017962027494725835
0.2199295314466923
```



接著取出並標準化，作為訓練 data

```
selected = np.array(elastic_model.coef_[:] >= 0.017962027494725835)
en_train_data = X_train.loc[:, selected]
en_test_data = X_test.loc[:, selected]

en_scaler_std = preprocessing.StandardScaler().fit(en_train_data)
en_x_train_std = en_scaler_std.transform(en_train_data)
en_x_test_std = en_scaler_std.transform(en_test_data)

print(en_train_data.columns)
```

```
Index(['d7', 'd9', 'd11', 'd13', 'd18', 'd22', 'd23', 'd24', 'd28', 'd29',
       'd32', 'd33', 'd35', 'd37', 'd38', 'c3', 'c10', 'c14', 'c17', 'c20',
       'c21', 'c22', 'c27', 'c31', 'c35', 'c37', 'c41', 'c43', 'c50'],
      dtype='object')
```

找 alpha 以及 l1_ratio

```
l1_ratio = np.linspace(0.0001, 1, 200)
best_R2 = 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)

    if(elastic_cv.score(en_x_train_std, y_train) > best_R2):
        alpha = elastic_cv.alpha_
        best_l1_ratio = i
        best_R2 = elastic_cv.score(en_x_train_std, y_train)

print(alpha)
print(best_l1_ratio)
```

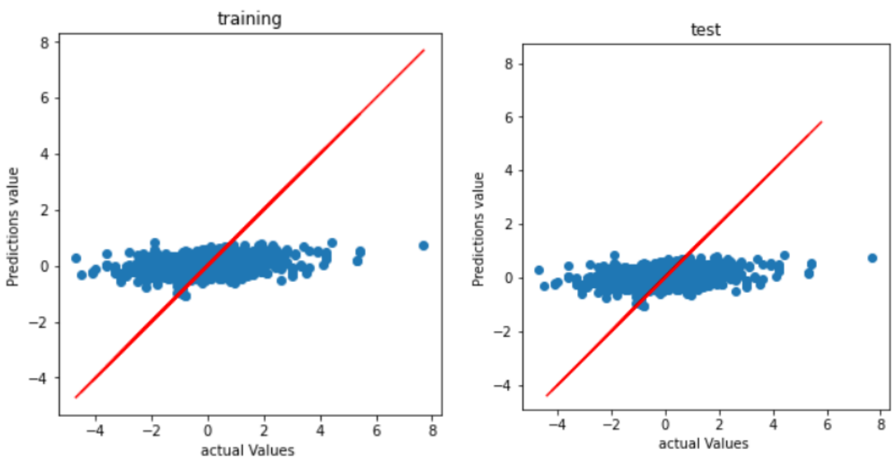
```
0.05760737700131587
0.00512462311557789
```

訓練模型

```
elastic_model = linear_model.ElasticNet(alpha=0.05760737700131587,l1_ratio=0.00512462311557789)
df_svr, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = regression_model_evaluation_result(en_x_train_std, y_train, en_x_test_std, y_test, elastic_model)
actual_vs_predict(y_train, tr_predlabel, y_test, te_predlabel)
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	6.951352	0.573209	0.620954	0.788006	0.065627	0.061841
Test	5.171857	0.585134	0.651122	0.806921	0.062333	0.058534



看起來並沒有比較好… 還是非常的差，我實在想不出原因。

Linear Regression

A. Feature selection

利用 RFECV 做進一步的特徵挑選，接著取出資料

```
lr = LinearRegression()
n_feature, selected = rfecv(lr, x_pretrain_std, y_train)
```

```
Optimal number of features : 45
Support : [ True False False False False False False  True False False  True  True
  True  True  True False  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True False  True False False  True
  True  True  True  True  True  True  True  True False  True  True False
 False False  True  True False  True  True  True  True False False  True
  True False  True  True False False  True False]
Ranking : [ 1  5 13 17 23  2 20  1 18 12  1  1  1  1  1 15  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1 14  1  6 19  1  1  1  1  1  1  1  1  1 16  1  1 22
 11 24  1  1  9  1  1  1  1  7 10  1  1  4  1  1  8 21  1  3]
```

```
lr_train_data = train_predata.loc[:, selected]
lr_test_data = test_predata.loc[:, selected]
```

B. 特徵資料標準化

```
lr_scaler_std = preprocessing.StandardScaler().fit(lr_train_data)
lr_x_train_std = lr_scaler_std.transform(lr_train_data)
lr_x_test_std = lr_scaler_std.transform(lr_test_data)
```

C. 選擇超參數

因為建模較快且參數也較少，因此使用 grid search

```
param = {
        'fit_intercept': ['True', 'False']
}

model = LinearRegression()
grid_search(model, param, lr_x_train_std, y_train)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
{'fit_intercept': 'True'}
```

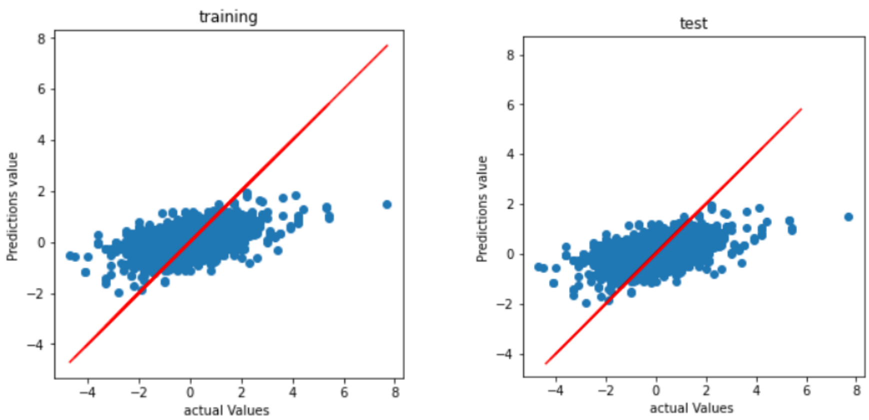
得出得結果為 fit_intercept = True

D. Training model

```
lr_model=LinearRegression(fit_intercept = True)
df_svr, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = regression_model_evaluation_result(lr_x_train_std, y_train, lr_x_test_std, y_test, lr_model)
actual_vs_predict(y_train, tr_predlabel, y_test, te_predlabel)
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	6.202603	0.530613	0.521481	0.722136	0.215307	0.210363
Test	4.929116	0.544234	0.545206	0.738381	0.214860	0.209913



XGBoost Regressor

A. Feature selection

使用 RFECV 做特徵挑選後取出

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

```
Optimal number of features : 49
Support : [ True  True False  True False  True  True  True  True False  True  True
 False False False False False  True False  True  True  True False
 False False False False  True  True  True  True  True False False  True
  True  True  True False False  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True False  True]
Ranking : [ 1  1  6  1  5  1  1  1  1  9  1  1 14 16 18 17  2  1 11  1  1  1  1  7
  3  8  4 12  1  1  1  1  1 20 19  1  1  1  1 15 13  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 10  1]
```

得出的特徵為['d1', 'd2', 'd4', 'd6', 'd7', 'd8', 'd9', 'd13', 'd14', 'd22', 'd28','d29', 'd31', 'd34', 'c1', 'c2', 'c3', 'c4','c6','c9', 'c10', 'c12','c13','c16','c17','c18','c19','c20','c21','c22','c23','c25','c28','c29','c30' ,'c31','c32', 'c33', 'c34', 'c35', 'c36', 'c37','c38', 'c39', 'c43', 'c44', 'c45', 'c46', 'c51']

共 49 個

B. 特徵資料標準化

```
xgb_scaler_std = preprocessing.StandardScaler().fit(xgb_train_data)
xgb_x_train_std = xgb_scaler_std.transform(xgb_train_data)
xgb_x_test_std = xgb_scaler_std.transform(xgb_test_data)
```

C. 選擇超參數

因為超參數較多，因此使用 randomsearchCV

```
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')
randomsearchCV(xgbr, param, xgb_x_train_std, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
{'max_depth': 9, 'gamma': 0.7702051282051282, 'eta': 0.01}
```

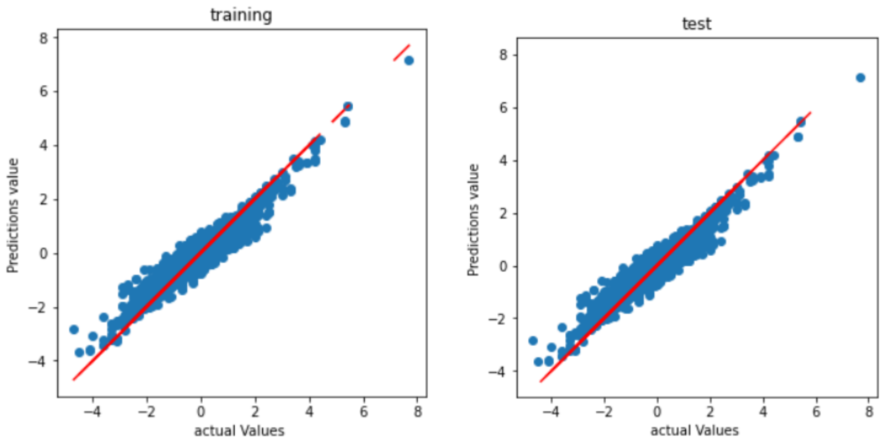
結果為 max_depth = 9, gamma = 0.7702051282051282, eta = 0.01

D. Training model

```
xgbr_model=XGBRegressor(max_depth = 9, gamma = 0.7702051282051282, eta = 0.01, objective='reg:squarederror')
df_svr, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = regression_model_evaluation_result(xgb_x_train_std, y_train, xgb_x_test_std, y_test, xgbr_model)
actual_vs_predict(y_train, tr_predlabel, y_test, te_predlabel)
```

結果

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	1.868687	0.229118	0.088433	0.297377	0.866931	0.866018
Test	6.378789	0.343489	0.243542	0.493500	0.649280	0.646873



GradientBoostingRegressor

A. Feature selection

利用 RFECV 挑特徵並取出

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

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

```
gbr_train_data = train_predata.loc[:, selected]
gbr_test_data = test_predata.loc[:, selected]
```

選擇的是['d22', 'c1', 'c2', 'c3', 'c6', 'c16', 'c18', c19', 'c20', 'c21', 'c22','c25', 'c30', 'c31', 'c32', 'c33', 'c34', 'c35', 'c37', 'c39', 'c43','c44', 'c45', 'c46', 'c49']

共 25 個

B. 特徵資料標準化

```
gbr_scaler_std = preprocessing.StandardScaler().fit(gbr_train_data)
gbr_x_train_std = gbr_scaler_std.transform(gbr_train_data)
gbr_x_test_std = gbr_scaler_std.transform(gbr_test_data)
```

C. 選擇超參數

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

gbr = GradientBoostingRegressor()
randomsearchCV(gbr, param, xgb_x_train_std, y_train)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
{'n_estimators': 71, 'loss': 'huber', 'learning_rate': 0.45222222222222225}
```

得出的結果為

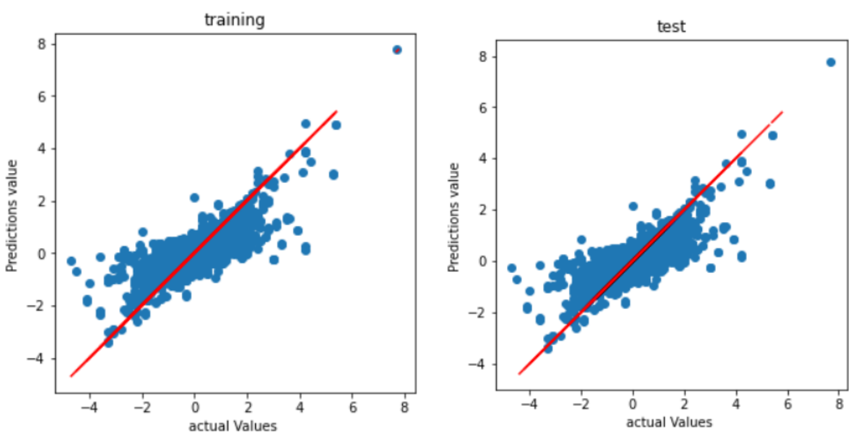
n_estimators = 71, loss = 'huber', learning_rate = 0.45222222222222
225

D. Training model

```
gbr_model=GradientBoostingRegressor(n_estimators = 71, loss = 'huber', learning_rate = 0.45222222222222225)
df_svr, tr_predsc, tr_predlabel, te_predsc, te_predlabel, model = regression_model_evaluation_result(gbr_x_train_std, y_train, gbr_x_test_std, y_test, gbr_model)
actual_vs_predict(y_train, tr_predlabel, y_test, te_predlabel)
```

結果為

	Max error	MAE	MSE	RMSE	R2	Adjusted R2
Training	4.426517	0.378671	0.282507	0.531514	0.574901	0.573417
Test	4.768897	0.433931	0.367432	0.606162	0.470869	0.469022



Survival

- **Package & function**

```python
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest
from sksurv.ensemble import RandomSurvivalForest
from sksurv.linear_model import CoxnetSurvivalAnalysis
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 scipy.stats import ranksums
from sklearn.feature_selection import RFECV
#-----------------model for regression----------------------------------
from sklearn.svm import SVR
from xgboost.sklearn import XGBRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import SGDRegressor
#-----------------model for survival------------------------------------
from sksurv.linear_model import CoxPHSurvivalAnalysis
#-----------------------------------------------------------------------
import eli5
from eli5.sklearn import PermutationImportance
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
```

RFECV

```python
def rfecv(model, x_train, y_train):
    r = RFECV(estimator = model, step = 8, cv = 3, scoring = 'neg_mean_squared_error').fit(x_train, y_train)
    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_
```

C_indicator：印出 training 以及 test 的 C-index、C-ipcw

```python
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)
```

regre_peform: 傳入 regression model 以及預測存活的 model 來看存活指標

```
def  regre_peform(regre_model,  sur_model,  x_train,  y_train_struct,  ystatus_train,  ytime_train,  x_test,  y_test_struct,  ystatus_test,  ytime_test):

    x_pretrain  =  x_train.values
    n_feature,  selected  =  rfecv(regre_model,  x_pretrain,  ytime_train)

    trfeature  =  x_train.loc[:,  selected]
    tefeature  =  x_test.loc[:,  selected]

    C_indicator(sur_model,  trfeature,  y_train_struct,  ystatus_train,  ytime_train,  tefeature,  y_test_struct,  ystatus_test,  ytime_test)

    return  trfeature.columns,  cox.coef_
```

Km_logrank: 印 KM plot 以及 log_rank

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

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

    for  expression  in  ("High  expression","Low  expression"):
        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")
```

My_score_val: cross_val_score 但是用 c-index

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

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

    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)
            #  print(i)
            #  print(remaining_data.shape)
            #  print(remaining_targets.shape)
            model.fit(remaining_data,  remaining_targets)
            record.append(model.score(val_data,  val_targets))

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

    return  cvs
```

自訂的評分(for c-index)

```
def  c_index_scoring(model,  x_train,  y_train):
    model.fit(x_train,  y_train)
    return  model.score(x_train,  y_train)
```

- **Data splitting**

因為樣本數少，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

```
x = raw_data.iloc[:,0:-3]
y_recur = raw_data.iloc[:,-1]
y_time = raw_data.iloc[:,-2]
y_status = raw_data.iloc[:,-3]
y_label = raw_data.iloc[:,-3:-1]
```

```
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]
```

```
[327] df_ranksum = pd.concat([X_train,y_train], axis = 1)
      filter = df_ranksum["recurrence"] != -1 #
      df = df_ranksum[filter]
```

- **Feature selection**

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 0.01

```
titles = list(df.iloc[:,0:-1].columns)
drop = list(df.iloc[:,-1])
df = df
p01_name=[]
c=0
for i in titles:
    #print(i)
    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:
        c=c+1
        p01_name.append(i)
print(c)
print(p01_name)
```

```
844
['ENSG00000082929.8', 'ENSG00000132832.10', 'ENSG00000166
```

接著取出資料

```
x_train = X_train[p01_name]
x_test = X_test[p01_name]

# y_train, y_test 為訓練和推論使用的正式答案(狀態+時間)
ytime_train = y_time[train_index]
ytime_test = y_time[test_index]
ystatus_train = y_status[train_index]
ystatus_test = y_status[test_index]
y_train = y_label.loc[train_index]
y_test = y_label.loc[test_index]
# print(x_train)
# print(ytime_train)
```

```
y_train_struct = y_train.to_records(index=False).astype([('Status', 'bool'), ('Survival', 'float64')])
y_test_struct = y_test.to_records(index=False).astype([('Status', 'bool'), ('Survival', 'float64')])
```

再來我試用了 4 種回歸模型來找出對於存活時間較相關的特徵並且再用這些特徵去跑 coxPH 來看看最後的 C-index 以及 C-ipcw

1. XGBoostRegressor

```
xgbr = XGBRegressor(objective='reg:squarederror')
cox = CoxPHSurvivalAnalysis(alpha = 0.1)

xgbr_selected, xgb_coef = regre_peform(xgbr, cox, x_train, y_train_struct, ystatus_train, ytime_train, x_test, y_test_struct, ystatus_test, ytime_test)
```

```
Optimal number of features : 244
```

	C-index	C-ipcw
training	0.982821	0.987557
test	0.560000	0.538840

2. GrandientBoostingRegressor

```
gbr = GradientBoostingRegressor()
cox = CoxPHSurvivalAnalysis(alpha = 0.00001)

gbr_selected, gbr_coef = regre_peform(gbr, cox, x_train, y_train_struct, ystatus_train, ytime_train, x_test, y_test_struct, ystatus_test, ytime_test)
```

```
Optimal number of features : 52
```

	C-index	C-ipcw
training	0.735477	0.760533
test	0.568136	0.556663

3. SGDregressor

```
sgd = SGDRegressor()
cox = CoxPHSurvivalAnalysis(alpha = 0.00001)

sgd_selected, sgd_coef = regre_peform(sgd, cox, x_train, y_train_struct, ystatus_train, ytime_train, x_test, y_test_struct, ystatus_test, ytime_test)
```

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

	C-index	C-ipcw
training	0.542626	0.560032
test	0.533898	0.544025

4. SVR

```
svr = SVR(kernel = 'linear')
cox = CoxPHSurvivalAnalysis(alpha = 0.00001)

svr_selected, svr_coef = regre_peform(svr, cox, x_train, y_train_struct, ystatus_train, ytime_train, x_test, y_test_struct, ystatus_test, ytime_test)
```

```
Optimal number of features : 4
```

	C-index	C-ipcw
training	0.547794	0.536619
test	0.558644	0.561524

由這 4 種跑出的 train 以及 test 的 C-index、C-ipcw 可得知 XGBoostRegressor 以及 GrandientBoostingRegressor 篩出來的特徵在跑 CoxPH 時表現比較好。而 XGBoostRegressor 的 train 好於 GrandientBoostingRegressor, 但是其 test 的表現較弱, 因此可能是 overfitting, 所以選擇 GrandientBoostingRegressor 選出的特徵, 共 52 個

```
coxPH_x_train = x_train.loc[:, gbr_selected]
coxPH_x_test = x_test.loc[:, gbr_selected]
```

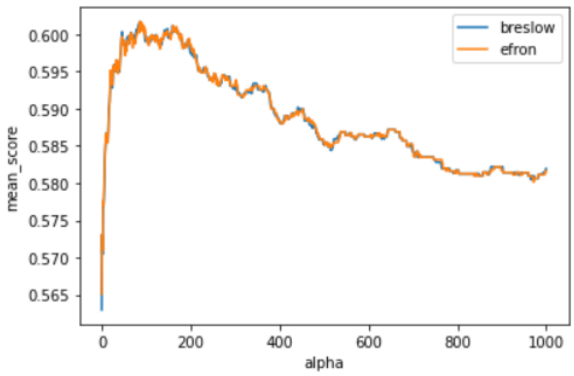
- **超參數挑選**

試各種 alpha 值以及看哪種 ties 的表現比較好

```python
alpha = np.linspace(0.00001, 1000, 1000)
bre_score = []
efr_score = []
best_score = 0
best_alpha = 0
best_tie = ""

for t in ["breslow", "efron"]:
    for i in alpha:
        cox = CoxPHSurvivalAnalysis(alpha = i, ties = t, n_iter = 200)
        now_score = my_cross_val(cox, coxPH_x_train, y_train_struct, 5)
        if(t == 'breslow'): bre_score.append(now_score)
        else: efr_score.append(now_score)

        if(now_score > best_score):
            best_tie = t
            best_score = now_score
            best_alpha = i

print(best_tie)
print(best_alpha)
print(best_score)

plt.plot()
breslow, = plt.plot(alpha, bre_score, label = 'breslow')
efron, = plt.plot(alpha, efr_score, label = 'efron')
plt.xlabel('alpha')
plt.ylabel('mean_score')
plt.legend(handles = [breslow, efron], loc='upper right')
plt.show()
```

最後得出的結果為

```
breslow
86.08609522522522
0.6016652760032903
```



Ties = Breslow 、 alpha = 86.0860952252522

- **Training model**

```python
cox = CoxPHSurvivalAnalysis(alpha = 86.08609522522522, ties = 'breslow', n_iter = 200)
C_indicator(cox, coxPH_x_train, y_train_struct, ystatus_train, ytime_train, coxPH_x_test, y_test_struct, ystatus_test, ytime_test)
```

結果

```
          C-index    C-ipcw
training  0.667403  0.689030
test      0.595593  0.581848
```

test 表現有比沒有調參數時好一點點

- **KM plot & logrank test**

先利用 PermutationImportance 把 feature importance 顯示出來

```
cox = CoxPHSurvivalAnalysis(alpha = 86.08609522522522, ties = 'breslow', n_iter = 200).fit(coxPH_x_train, y_train_struct)
importance = PermutationImportance(cox, random_state=1).fit(coxPH_x_test, y_test_struct)
eli5.show_weights(importance, feature_names = coxPH_x_test.columns.tolist())
```

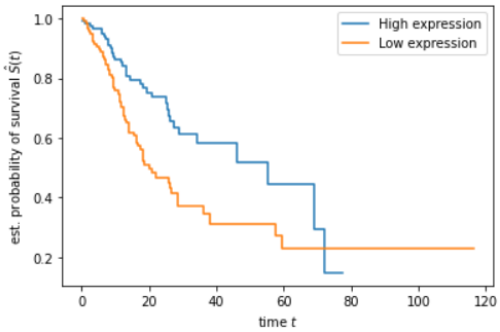
Weight	Feature
0.1031 ± 0.0427	ENSG00000229953.1
0.0328 ± 0.0507	ENSG00000230479.1
0.0193 ± 0.0105	ENSG00000263588.1
0.0188 ± 0.0129	ENSG00000272405.1
0.0167 ± 0.0092	ENSG00000232079.7
0.0165 ± 0.0141	ENSG00000251129.2
0.0145 ± 0.0118	ENSG00000244137.1
0.0127 ± 0.0057	ENSG00000233818.1
0.0103 ± 0.0103	ENSG00000256001.2
0.0081 ± 0.0060	ENSG00000259436.1
0.0074 ± 0.0052	ENSG00000231437.3
0.0063 ± 0.0077	ENSG00000272970.3
0.0058 ± 0.0057	ENSG00000246430.7
0.0056 ± 0.0050	ENSG00000256124.6
0.0045 ± 0.0124	ENSG00000255910.2
0.0042 ± 0.0134	ENSG00000272625.1
0.0042 ± 0.0029	ENSG00000288045.1
0.0038 ± 0.0054	ENSG00000258654.1
0.0035 ± 0.0029	ENSG00000275894.1
0.0034 ± 0.0038	ENSG00000285653.1
… 32 more …	

我選擇了前面 5 個來看他們的 KM plot 以及 logrank test

1. ENSG00000229953.1

```
km_logrank('ENSG00000229953.1', X_train, y_train_struct)
```

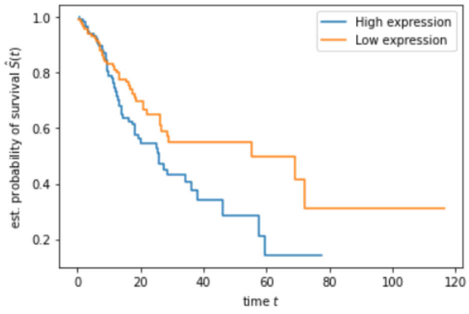
```
                                     test_statistic        p   -log2(p)
High expression Low expression            12.272727  0.00046  11.087252
```



2. ENSG00000230479.1

```
km_logrank('ENSG00000230479.1', X_train, y_train_struct)
```

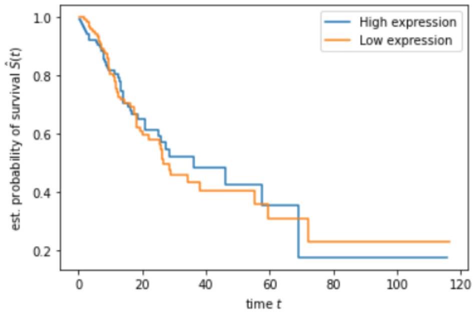
```
                                test_statistic         p  -log2(p)
High expression Low expression        4.86532  0.027402   5.18959
```



3. ENSG00000263588.1

```
km_logrank('ENSG00000263588.1', X_train, y_train_struct)
```

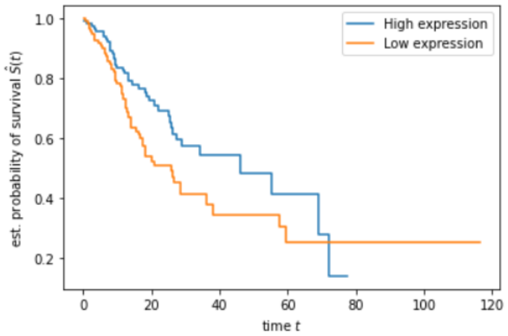
```
                                test_statistic         p  -log2(p)
High expression Low expression       2.845118  0.091652  3.447692
```



4. ENSG00000272405.1

```
km_logrank('ENSG00000272405.1', X_train, y_train_struct)
```

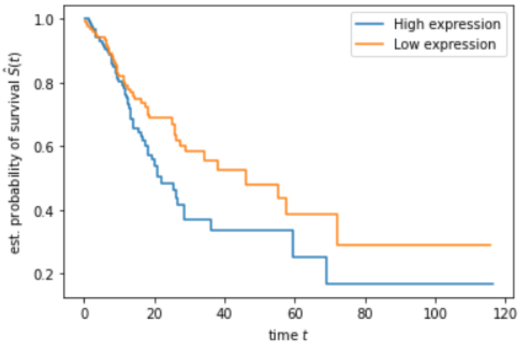
```
                                test_statistic         p  -log2(p)
High expression Low expression       6.077441  0.013692  6.190534
```



5. ENSG00000232079.7

```
km_logrank('ENSG00000232079.7', X_train, y_train_struct)
```

		test_statistic	p	-log2(p)
High expression	Low expression	1.363636	0.242908	2.041517



由上述結果看出 ENSG00000229953.1、ENSG00000230479.1、ENSG00000272405.1 的高表現與低表現在存活曲線上存在顯著差異 (test_statistic > 3.841)

最終模型

Regression:

```
from joblib import dump, load
```

```
dump(xgbr_model, 'regression.joblib')
```

```
['regression.joblib']
```

```
test_model = load('regression.joblib')
```

其對應的訓練紀錄在 regression 中的 XGBboostRegression

COX:

```
from joblib import dump, load
```

```
dump(cox, 'cox.joblib')
```

```
['cox.joblib']
```

其對應的訓練紀錄在上方的 survival 中

結論

Regression:

由最後的表現可看出 XGBoost Regressor 中 training 以及 testing 的 MAE、MSE、R2、adjust-R2 都比其他的模型來的好，且最後得出來的 actual vs predict 的也較為接近對角線，因此最後選擇 XGBoost Regressor 所訓練出來的模型。其中 elasticNet 不知道為什麼都做不好

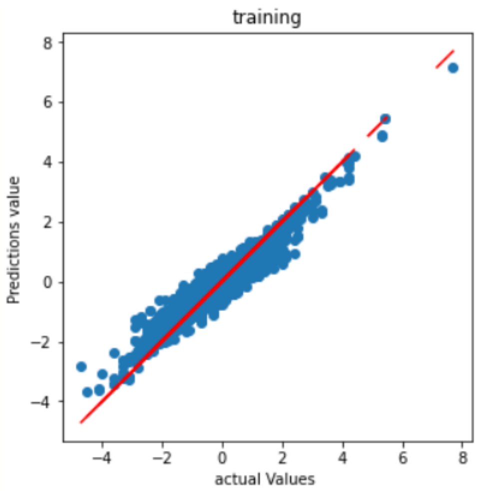
COX:

這次先利用 p-value 找出和復發相關的 lncRNA，再使用了各種回歸的模型挑特徵，再去跑 coxPH。 雖然就 training 以及 test 的 C-index、C-ipcw 還是有待加強，但其 C-index 接近 0.6 還算是有一些些預測能力。最後由 logrank 看出 ENSG00000229953.1、ENSG00000230479.1、ENSG00000272405.1 的高表現與低表現在存活曲線上存在顯著差異

加分題

```
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(tr_act, tr_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')
```

結果會顯示出



實作問題

1. 我有嘗試用 cross_val_score 以及自訂評分標準來做 CoxPH 的特徵篩選，但都會跳出一堆 Runtimewarning 且很常最後會顯示 search direction contains NaN or infinite values 而終止。 之後我想說可能是函式本身的問題，因此自己手刻了一個 cross_val_score，結果發現也會出現相同狀況。 上網查得到的解法是把 alpha 調小，或是換一種模型。 雖然 alpha 調小後有比較少發生，但還是偶爾會這樣。

```
def embedded(model, threshold, x_train, y_train):
    score = []
    best_threshold = threshold[0]
    best_score = 0
    scoring_for_cindex = make_scorer(c_index_scoring)
    for i in threshold:
        x_embedded = SelectFromModel(model, threshold = i).fit_transform(x_train, y_train)
        model.fit(x_embedded, y_train)
        # now_score = model.score(x_embedded, y_train)
        # score.append(now_score)
        # if(now_score > best_score):
        #     best_score = now_score
        #     best_threshold = i
        mean_score = cross_val_score(model, x_embedded, y_train, cv = 5, scoring = scoring_for_cindex).mean()
        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()
```

```
def c_index_scoring(model, x_train, y_train):
    model.fit(x_train, y_train)
    return model.score(x_train, y_train)
```

```
embedded(cox, threshold, x_train, y_train_struct)
```

```
/usr/local/lib/python3.7/dist-packages/sksurv/linear_model/coxph.py:174: RuntimeWarning: overflow encountered in exp
  risk_set += numpy.exp(xw[k])
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:680: ConvergenceWarning: Optimization d
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:774: UserWarning: Scoring failed. The s
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line 761, in _score
    scores = scorer(estimator, X_test, y_test)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_scorer.py", line 103, in __call__
    score = scorer._score(cached_call, estimator, *args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_scorer.py", line 264, in _score
    return self._sign * self._score_func(y_true, y_pred, **self._kwargs)
TypeError: c_index_scoring() missing 1 required positional argument: 'y_train'

  UserWarning
```

自訂的 cross_val_score

```
def embedded(model, threshold, x_train, y_train):
    score = []
    best_threshold = threshold[0]
    best_score = 0

    for i in threshold:
        x_embedded = SelectFromModel(model, threshold = i).fit_transform(x_train, y_train)
        #model.fit(x_embedded, y_train)
        # now_score = model.score(x_embedded, y_train)
        # score.append(now_score)
        # if(now_score > best_score):
        #     best_score = now_score
        #     best_threshold = i
        mean_score = my_cross_val(model, x_embedded, y_train, 5)
        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()
```

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

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

    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)

        # print(i)
        # print(remaining_data.shape)
        # print(remaining_targets.shape)
        model.fit(remaining_data, remaining_targets)
        record.append(model.score(val_data, val_targets))

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

    return cvs
```

```
embedded(cox, threshold, x_train, y_train_struct)
```

```
search direction contains NaN or infinite values
```

```
/usr/local/lib/python3.7/dist-packages/sksurv/linear_model/coxph.py:174: RuntimeWarning: overflow encountered in exp
  risk_set += numpy.exp(xw[k])
/usr/local/lib/python3.7/dist-packages/sksurv/linear_model/coxph.py:171: RuntimeWarning: overflow encountered in exp
  risk_set2 += numpy.exp(xw[k])
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: ConvergenceWarning: Optimization did not converge: Maximum number of iterations has been exceeded.
/usr/local/lib/python3.7/dist-packages/sksurv/linear_model/coxph.py:174: RuntimeWarning: overflow encountered in exp
  risk_set += numpy.exp(xw[k])
/usr/local/lib/python3.7/dist-packages/sksurv/linear_model/coxph.py:171: RuntimeWarning: overflow encountered in exp
  risk_set2 += numpy.exp(xw[k])
```

2. 我在做 CoxPH 時，若把篩選特徵的 p-value 調到 0.01 後，建模時會顯示 search direction contains NaN or infinite values，但我分別檢查了 x_train、y_train_struct 以及 x_test、y_test_struct 都沒有看到有空值或是無限大的值

```python
titles = list(df.iloc[:,0:-1].columns)
drop = list(df.iloc[:,-1])
df = df
p01_name=[]
c=0
for i in titles:
        #print(i)
        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:
            c=c+1
            p01_name.append(i)
print(c)
print(p01_name)
```

```
844
['ENSG00000082929.8', 'ENSG00000132832.10', 'ENSG0000016
```

```python
cox = CoxPHSurvivalAnalysis()
C_indicator(cox, x_train, y_train_struct, ystatus_train, ytime_train, x_test, y_test_struct, ystatus_test, ytime_test)
```

```
frames
/usr/local/lib/python3.7/dist-packages/sksurv/linear_model/coxph.py in fit(self, X, y)
    435
    436             if not numpy.all(numpy.isfinite(delta)):
--> 437                 raise ValueError("search direction contains NaN or infinite values")
    438
    439             w_new = w - delta

ValueError: search direction contains NaN or infinite values
```