

# Final Report

109550137 資工 13 徐敏芝

github : [https://github.com/MinChihHsu/NYCU\\_ML\\_Final.git](https://github.com/MinChihHsu/NYCU_ML_Final.git)

model : [https://drive.google.com/drive/folders/15H1Gmzj7jPi61YhHlPpySuUBJeLezLcK?usp=share\\_link](https://drive.google.com/drive/folders/15H1Gmzj7jPi61YhHlPpySuUBJeLezLcK?usp=share_link)

## Environment details

- Environment  
Google Colab
- Python Version  
Colab 上的 python 版本為 3.8.16

```
[1] !python --version
```

```
Python 3.8.16
```

## Brief Introduce

我使用的是 Logistic Regression 方法搭配 Feature Engineering  
因為開始做之前，課後討論時就有聽到同學用 LR 做出不錯的結果，初步嘗試也得到 0.58792，覺得是可行的方向，因此才用這個方法繼續做下去

## Methodology

- Feature Engineering  
我們擁有的 data 總共有 25 個 features，而太多特徵對於訓練模型並不一定是好事，甚至有可能因為不必要的資訊太多而降低 performance，因此我先分析 features 並選擇和要求的 failure 相關度高的，同時增加一些重要 feature

### [最終採用的 features]

```
X_columns = [ # 0.59184
    'loading',
    'attribute_0',
    'attribute_1',

    'm3_missing',
    'm5_missing',
    'area',
    'missing(3*5)',
    #'measurement(3*5)',

    'measurement_0 / loading',
    'measurement_1 / loading',
    'measurement_2 / loading',
    'measurement_3_to_16_mean / loading',
    'measurement_17 / loading',
]
```

1. **id** (捨去)  
index, 每個產品都不一樣, 對於模型沒有意義

2. **product\_code** (捨去)  
categorical value, 但 train data 和 test data 完全沒有一樣的 product\_code, 因此在訓練時完全可以忽略

```
train_data['product_code'].value_counts()
C    5765
E    5343
B    5250
D    5112
A    5100
Name: product_code, dtype: int64

test_data['product_code'].value_counts()
F    5422
I    5228
G    5107
H    5018
Name: product_code, dtype: int64
```

3. **loading** (保留) (inspired by [\[1\]](#))

在計算 failure 和不同 feature 的 Pearson correlation 時, loading 和 failure 的直接相關度最高, 因此我把他留下來

```
failure    1.000000
loading    0.129089
measurement_17  0.033905
attribute_3    0.019222
measurement_5    0.018079
measurement_8    0.017119
measurement_7    0.016787
measurement_2    0.015808
measurement_6    0.014791
measurement_1    0.010810
measurement_4    0.010488
measurement_0    0.009646
attribute_2    0.006337
measurement_14    0.006211
measurement_11    0.004801
measurement_12    0.004398
measurement_9    0.003587
measurement_3    0.003577
measurement_15    0.003544
measurement_16    0.002237
measurement_13    0.001831
measurement_10    0.001515
Name: failure, dtype: float64
```

4. **attribute 0, 1, 2, 3** (保留不重複的) (inspired by [\[2\]](#))

attribute0, 1 和 attribute2, 3 就只是將分類字串轉成 int 格式而已, 因此要保留也是在 0, 1 和 2, 3 之間選一組

雖然 attribute 和 product\_code 的關聯性相當高 (同一 attribute 組合有一樣的 product\_code)

但因為實驗結果, 我沒有像 product\_code 一樣捨棄他 (實驗結果在 abalation), 而是分別保留了 attribute\_0 和 attribute\_1

5. **area** (新增) (inspired by [\[3\]](#))

由 attribute\_2 \* attribute\_3 得到

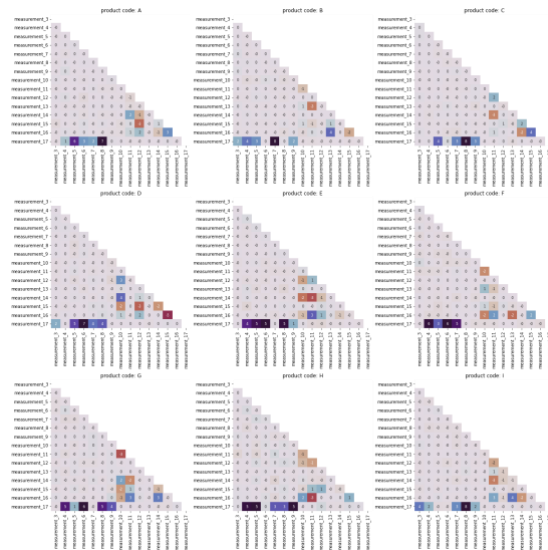
因為同一 product\_code 有相同的 attribute 組合, 將他看成此 material 的體積組成似乎滿合理的, 因此將他們想成長\*寬, 多設一個 feature 為面積

6. **measurement 0, 1, 2** (保留)

用 isnull().sum() 去看結果, 發現這三個 feature 都沒有 missing data, 不用自己填, 準確率可能更高, 因此三個都留下來了

## 7. **measurement17** (保留) (inspired by [4])

計算所有 measurement 的 correlation matrices 時，發現 measurement17 那一條色塊最多顏色也最深，代表 measurement17 和其他 measurement 的相關度夠高，對結果的影響應該不差



## 8. **measurement\_3~16** (取平均保留) (inspired by [4])

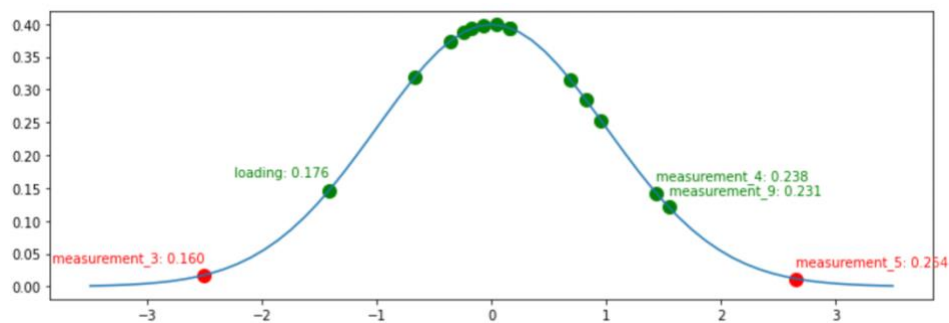
剩餘的 measurement 並沒有明顯的關聯性，在選擇不出來的情況下，我認為將他們相加取平均是既可以保留對 failure 的影響，也能減少多餘 feature 的方法

## 9. **missing data** (新增的) (inspired by [5])

也許 failure 和 measurement 的缺失有關，也就是說，measurement data 的遺失和我們要求的 failure 有一定程度的關聯我們將 conditional failure rate (measurement 缺失的情況下) 和 train\_data 的 failure rate(0.212608)做比較

同時，因為偏差量可能很小，計算 z-score 和 p-value 已取得 normal distribution

結果發現，缺失 measurement\_3 和缺失 measurement\_5 的條件故障率明顯「偏離」平均故障率，因此加入 features : m3\_missing 和 m5\_missing (用 isnull()取得是否缺失，並將回傳的 T/F 轉成 int)



## 10. **missing date 相乘** (新增的) (inspired by [\[6\]](#))

將 m3\_missing 和 m5\_missing 相乘，得到 missing(3\*5)

\*原本也有嘗試同一篇 reference 的 additional feature :

measurement(3\*5)，但反而讓效果降低

### ➤ Data Pre-processing

#### 1. 切分 training data 和 testing data

一開始為了測試模型的好壞，給了 random seed 並以 8 : 2 的比例切分，同時將 X 和 y 分開

#### 2. 建立 additional feature

選擇好需要的 features 後，將原本不存在於 data frame 中的 feature 創造出來 (例如：missing\_3, missing\_5, area...)，同時，將與 measurement 相關的 feature 除以 loading (inspired by [\[2\]](#))

#### 3. 處理 missing data

用 scikit-learn 的 SimpleImputer 填補 missing data

以資料型態分別處理 missing data

##### - category

填入最常見的類型

用 OneHotEncoder 將類別轉換成數值資料

把條件加上 handle\_unknown，這樣當出現 training data 中沒有的類別時，直接填 0，讓維度保持一致

##### - int

填入中位數，add\_indicator = True，數據後面會多一條{0, 1}陣列，用來表示此資料原本是否為 missing data

用 StandardScaler 做標準化，數據的平均值為 0，平方差為 1

##### - float

填入平均值，add\_indicator = True，數據後面會多一條{0, 1}陣列，用來表示此資料原本是否為 missing data

用 StandardScaler 做標準化，數據的平均值為 0，平方差為 1

### ➤ Model

使用 scikit-learn 的 Logistic Regression CV [\[7\]](#)，並將其套用於切好的 training data 上

但 CV stands for cross validation，因此上面切分 training data 和 testing data 是不必要的，甚至因此減少了訓練資料

到這個階段已經可以用全部的資料訓練了，但我仍然將其切分，並只丟 training data 進去，原因是因為將全部資料丟入的結果比切分後糟糕，因此也就將它留下來了

```

('model',
 LogisticRegressionCV(Cs=[0.001, 0.005, 0.01, 0.05, 0.1, 0.5,
                          1.0],
                      cv=5,
                      l1_ratios=array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
                      max_iter=5000, penalty='elasticnet',
                      random_state=0, scoring='roc_auc',
                      solver='saga', tol=0.001)))

```

- Cs: inverse of regularization strength  
[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0]
- cv: cross-validation 的 k-fold  
5, 相當於 8:2 切
- penalty: norm used in the penalization  
'elasticnet', both L1 and L2 penalty terms are added
- scoring: use as cross-validation criteria  
roc\_auc (Area Under the Receiver Operating Characteristic Curve)
- solver: algorithm to use in the optimization problem  
'saga', 適合 large dataset
- random\_state: 因為 solver 是 'saga', 所以需要 shuffle data  
0, 給 int 相當於 seed
- max\_iter: maximum number of iterations  
5000, default 為 100
- tol: tolerance for stopping criteria  
1e-3, default 為 1e-4
- l1\_ratio: Elastic-Net mixing parameter  
np.arange(0, 1.01, 0.1), 相當於 [0, 0.1, 0.2, ..., 1.0]

#### ➤ save model

使用 joblib, 儲存訓練好的 model

#### ➤ generate prediction

1. 先用 joblib.load 載入 pre-trained model
2. 將 test\_data 依照 data pre-processing 的第二點, 同樣創建一樣的 additional features
3. 再將其丟入 model, 以 predict\_prob 預測其為 failure 的機率
4. 將 id 和 prediction 結合為 pandas 格式
5. 將此 dataframe 轉成 csv 檔

## Ablation studies (of features)

➤ My Final Score: (private)0.59184 / (public)0.58339

1. measurement 相關的 features 沒有除以 loading

(private)0.59052 / (public)0.58284

2. without attribute\_0 and attribute\_1

(private)0.59123 / (public)**0.58417**

3. without attribute\_1

(private)0.59157 / (public)0.58326

4. without area

(private)0.59183 / (public)**0.58378**

5. without missing(3\*5)

same

6. without measurement\_0









(private)0.5918 / (public)0.58331

7. without measurement\_1

(private)0.56932 / (public)0.58026

8. without measurement\_17

(private)0.58914 / (public)0.57858

 109550137_submission_without_measurement17.csv Complete (after deadline) - now	0.58914	0.57858
 109550137_submission_without_measurement1.csv Complete (after deadline) - 3m ago	0.56932	0.58026
 109550137_submission_without_measurement0.csv Complete (after deadline) - 7m ago	0.5918	0.58331
 109550137_submission_without_attribute1.csv Complete (after deadline) - 8m ago	0.59157	0.58326
 109550137_submission_without_attribute01.csv Complete (after deadline) - 9m ago	0.59123	0.58417
 109550137_submission_without_missing35.csv Complete (after deadline) - 12m ago	0.59184	0.58339
 109550137_submission_without_loading.csv Complete (after deadline) - 12m ago	0.59052	0.58284
 109550137_submission_without_area.csv Complete (after deadline) - 16m ago	0.59183	0.58378

## Interesting finding

1. 做出來的大部分結果都是 private score 比 public score 好

若能知道 public data 和 private data 的差別可能會很有用，知道自己忽略了哪個部分

但自己做的 cross validation 結果應該和 private score 比較接近

2. 在處理資料型態為 int 和 float 的 missing data 時，若在條件加上 add\_indicator=True，能讓表現更好 (0.59128 to 0.59184)

我認為主因是 float 資料型態，就像前面提到的，measurement data 的遺失可能和要求的 failure 有一定程度的關聯

當加上額外表示「是否為 missing data」的 column，model 可以學習到 missing values 和 target variable 的關係

3. 使用切好的 X\_train, y\_train 訓練，反而比完整的 train\_data 丟進去表現更好

我認為這真的只是運氣問題，慶幸當初在切 train-test 時，有設 random seed，才可以複製那個最好的結果

## Summary

### ➤ Approach

一開始我使用 Logistic Regression，在做 Feature Selection & Construction 之前（只有填 missing data）得到的 private score 是 0.58729，看起來和 baseline 的 0.58990 似乎相差不遠，但真正做了才發現，連 0.0005 都是遙遠的距離

最終我仍繼續使用 Logistic Regression CV，並參考了大家選擇的不同 feature，在不斷嘗試下找到了可以過 baseline 的最高分

### ➤ What did I learn

對於我使用的方法，最重要的其實是 feature 的選擇，這也是之前沒有深入研究過的，也讓我理解機器學習很重視資料分析，我也學到了許多分析 feature 的方法（包括 correlation, normal distribution, heatmap 等）

### ➤ Result: private = 0.59184

109550137\_submission.csv  
Complete (after deadline) · 3h ago

0.59184

0.58339

## Reference

- [1] <https://www.kaggle.com/code/alvinleenh/tps0ct22-6-basic-feature-selection-techniques?scriptVersionId=107676046>
- [2] <https://www.kaggle.com/code/scgupta/tps-2022-aug-eda-baseline-logistic-regression/notebook?scriptVersionId=104304388>
- [3] <https://www.kaggle.com/code/maxsarmiento/lb-0-58978-standing-on-the-shoulder-of-giants?scriptVersionId=102785631>
- [4] <https://www.kaggle.com/code/takanashihumbert/tps-aug22-lb-0-59013>
- [5] <https://www.kaggle.com/code/ambros/tps-aug22-eda-which-makes-sense>
- [6] <https://www.kaggle.com/code/desalegngeb/tps08-logisticregression-qlattice/notebook>
- [7] [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)