

# Introduction to Machine Learning, Final Project

109550155 端木竣偉

## • Brief introduction

此次作業主要使用Logistic Regression為主要架構，結合諸多文章的參考以及自己的觀察，對資料進行預處理，並選出好的feature跟進行人工調參，本文最大的亮點應該是針對measurement17與其有關連的feature進行KNNImputer，使得我在private score的正確率從0.59085上漲到0.59157。

## • Environment & Github link

我採用的python version為Python 3.9.7

```
(env_test) (base) raytm9999@gpuserv3:~/HW5$ python --version
Python 3.9.7
```

由於原本是在server上train跟inference，requirement.txt高達三百多行，因此手動選出必要的套件版本

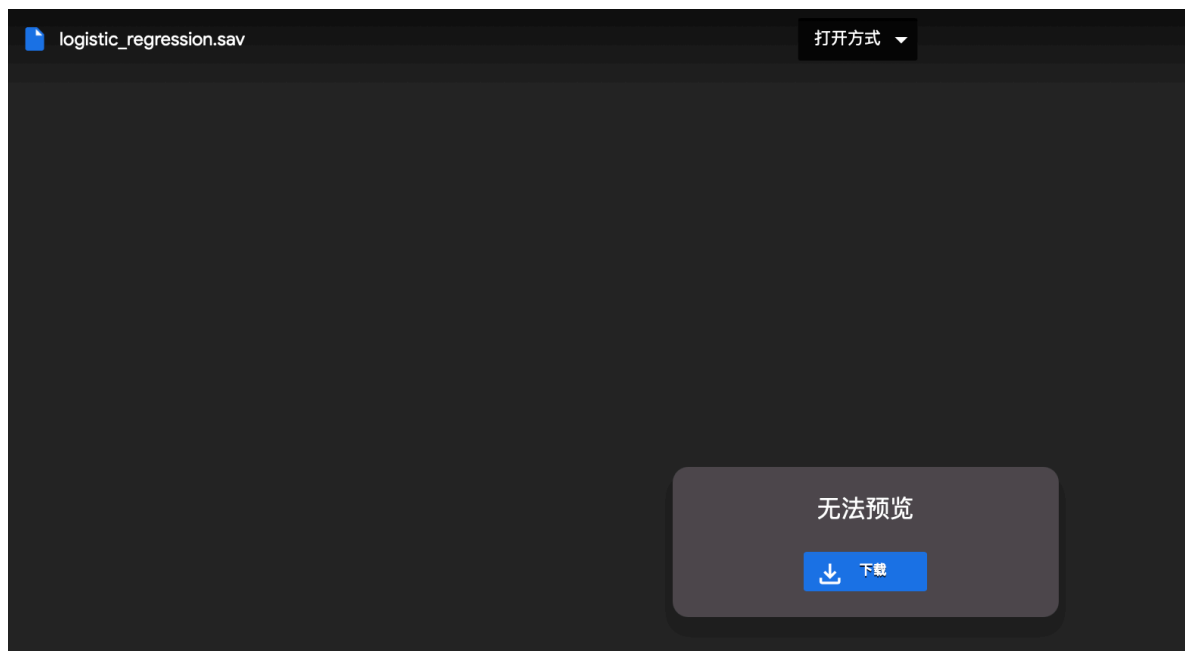
```
requirements.txt
numpy==1.20.3
pandas==1.3.4
scikit-learn==1.0.2
```

開啟新的虛擬環境，並使用pip install -r requirements.txt後，列出以下所安裝的所有套件

Package	Version
-----	-----
joblib	1.2.0
numpy	1.20.3
pandas	1.3.4
pip	22.3.1
python-dateutil	2.8.2
pytz	2022.7
scikit-learn	1.0.2
scipy	1.10.0
setuptools	65.6.3
six	1.16.0
threadpoolctl	3.1.0
wheel	0.38.4

我將model weight放在以下連結：

<https://drive.google.com/file/d/1FUFsbKHZQIXlp2tw1QM7PHrgXpA8EU5o/view?usp=sharing>



會看到名為logistic\_regression.sav的weight

底下是我的GitHub:

[https://github.com/rayray9999/ML\\_Final\\_project](https://github.com/rayray9999/ML_Final_project)

## • Implementation

### Preprocessing:

長時間的觀察還是沒什麼結論，直到我參考以下的文章：

<https://www.kaggle.com/code/ambrosm/tpsaug22-eda-which-makes-sense>

統計train的資料發現，大部分的feature都跟failure沒什麼關聯，只有loading有明顯的正相關，因此可以只挑選數個重要的feature作為model判斷的依據就好。

另外由於資料中有些feature（loading+m3~m17）缺損，我們除了需要考慮如何impute之外，是否能從中取得一些有助於預測的資料呢？



從上圖可以發現，我們可以發現m3的遺失跟failure呈現負相關，而m5的遺失則跟failure成正相關，因此我們可以多出兩個feature分別紀錄這兩項數據。

另外在這篇文章認為除了m17之外，其他的feature都是相對Independent的，不過我發現似乎未必是如此，並且利用了兩次KNNimputer，提升了0.00012的成績，詳見experiment。再來，我們還能發現「loading」的分佈是以對數常態分佈的，使用np.log1p讓他標準化（這提升了0.0002的準確度）。

從這篇文章：<https://www.kaggle.com/competitions/tabular-playground-series-aug-2022/discussion/342126>

觀察出attribute2跟3是EDA的長與寬，因此創造了area這項feature。

接下來放出我在train的程式碼：

```
def preprocess(data):
    feature = ['loading', 'measurement_0', 'measurement_1', 'measurement_2',
               'measurement_3', 'measurement_4', 'measurement_5', 'measurement_6',
               'measurement_7', 'measurement_8', 'measurement_9', 'measurement_10',
               'measurement_11', 'measurement_12', 'measurement_13', 'measurement_14',
               'measurement_15', 'measurement_16', 'measurement_17']

    m17_corre = {
        'A': ['measurement_5', 'measurement_6', 'measurement_8'],
        'B': ['measurement_4', 'measurement_5', 'measurement_7'],
        'C': ['measurement_5', 'measurement_7', 'measurement_8', 'measurement_9'],
        'D': ['measurement_5', 'measurement_6', 'measurement_7', 'measurement_8'],
        'E': ['measurement_4', 'measurement_5', 'measurement_6', 'measurement_8']
    }

    data['loading'] = np.log1p(data['loading'])
    data['m3_missing'] = data['measurement_3'].isna().astype('int64')
    data['m5_missing'] = data['measurement_5'].isna().astype('int64')
    data['area'] = data['attribute_2'] * data['attribute_3']

    for p_code in data.product_code.unique():
        print("start processing product_code:", p_code)
        c_data = data[data.product_code==p_code]
        correlated_measurement = m17_corre[p_code]
        c_data_nona = c_data[correlated_measurement+['measurement_17']].dropna()
        c_data_miss_only_m17 = c_data[(~c_data[correlated_measurement].isnull().any(axis=1)) & (c_data['measurement_17'].isnull())]
        model = HuberRegressor(eps=2)
        model.fit(c_data_nona[correlated_measurement], c_data_nona['measurement_17'])
        data.loc[(data.product_code==p_code)&(~c_data[correlated_measurement].isnull().any(axis=1))&(data['measurement_17'].isnull()), 'measurement_17'] =
            model.predict(c_data_miss_only_m17[correlated_measurement])
        for_rest = KNNImputer(n_neighbors=3)
        #for_rest = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
        data.loc[(data.product_code==p_code, correlated_measurement+['measurement_17'])] = for_rest.fit_transform(data.loc[(data.product_code==p_code,
            correlated_measurement+['measurement_17'])])
        data.loc[(data.product_code==p_code, feature)] = for_rest.fit_transform(data.loc[(data.product_code==p_code, feature)])
        #for i in range(3,17):
        #data[data.product_code==code][f'measurement_{i}'].fillna(value=data[data.product_code==code][f'measurement_{i}'].mean(), inplace=True)
    #data['measurement_avg'] = data[[f'measurement_{i}' for i in range(3, 17)]].mean(axis=1)
    return data
```

首先我根據前面以及以下的文章：

<https://www.kaggle.com/competitions/tabular-playground-series-aug-2022/discussion/343939>

知道了m17跟哪些measurement會有線性關係，而且他們的關聯是會依product code不同而改變。

於是我會不同的product code，選出沒有遺失correlated\_measurement&measurement17的數據，以HuberRegressor進行計算，最後填補缺漏。

至於其他的feature，我先將correlated\_measurement與measurement17用KNNimputer計算，最後再將所有的feature一起用KNNimputer。

以上的部分frame有參考自以下兩篇文章：

<https://www.kaggle.com/code/medali1992/tps-aug-logistic-regression>  
<https://www.kaggle.com/code/pourchot/hunting-for-missing-values>

這兩篇的preprocess算是把我前面看到的幾篇文章(畢竟都是most voted)給implement，第一篇就是只認為m17有correlation而第二篇則是找了八種correlation高的做線性回歸，因為筆者觀察多篇文章只得到第一種結論，因此也是只對m17做線性回歸（但後面KNN的創新作法使得我正確率大幅提升，詳見實驗）。

## Select feature:

```
select_feature = ['measurement_1', 'measurement_10', 'measurement_17', 'm3_missing', 'm5_missing', 'loading', 'area']
```

選擇這些feature作為模型的判斷依據，而選擇的方式則是先從前面的觀察來選定主要使用哪些，再留下能夠讓accuracy最好的，經實驗發現新增的幾個feature都能有效增加正確率。

## Standardize part :

在苦於提升正確率時，看到底下文章：

<https://www.kaggle.com/code/majidabdoos/tps-aug-2022-lgbmimputer-pca-lr-0-59041>

才知道有scaler能快速標準化，而我嘗試了minmaxscaler跟standardscaler，後者能提升了0.06之多！

以下是在train中的程式碼：

```
sc = StandardScaler()  
train_x = sc.fit_transform(train[select_feature])
```

## model architecture :

我訓練時是使用單一的LogisticRegression model：

```
model = LogisticRegression(max_iter=500, C=0.0001, penalty='l2', solver='newton-cg')  
model.fit(train_x, train_y)
```

max\_iter從0~1000選擇，發現1000會overfitting，而300~500的正確率都相同。

C則是從0.0001~1選擇，0.0001會是最好的。

Penalty選擇l1的正確率遠低於l2，因此選擇l2。

solver則是svg跟newton-cg都會是相同的正確率。

## Inference part :

修改相關路徑，並打開submission存放之後預測的結果

```
#print(pickle.format_version)  
WEIGHT_PATH="" #for weight path  
TEST_PATH = "tabular-playground-series-aug-2022" #for test.csv path  
SUBMISSION_PATH="tabular-playground-series-aug-2022" #for sample_submission.csv path  
  
submission = pd.read_csv(os.path.join(SUBMISSION_PATH, 'sample_submission.csv'))  
test_df = pd.read_csv(os.path.join(TEST_PATH, 'test.csv'))
```

之後的過程都與train相同。

而後用model\_file存放權重並放入model中，最後將預測的probability中的positive probability（即第二個column）放入submission的failure列。


```
model_file = open(os.path.join(WEIGHT_PATH, 'logistic_regression.sav'), 'rb')
model = pickle.load(model_file)
model_file.close()
print("--start predict--")
submission['failure']=model.predict_proba(test)[: , 1]
```

最後儲存結果，並將index設為false。

```
submission.to_csv('109550155.csv', index=False)
```

至此，就能夠從訓練並inference，獲得一樣的結果。

以下是我得到的最佳並復現在虛擬環境上的結果：

 109550155.csv	0.59157	0.59087	<input type="checkbox"/>
Complete (after deadline) · now			

## Experiment：

### KNNImputer vs Simpleimputer?

對於除了measurement17以外的feature，如果真的沒有任何correlated，那似乎兩種方式都不會有太大差異，但是底下是我純粹使用Simpleimputer的結果：

0.59096	0.5869
---------	--------


而底下是我只使用單次KNNImputer的結果：

0.59097	0.59067	<input type="checkbox"/>
---------	---------	--------------------------

可以看到在public score上明顯的非常低，因此我才認為這些measurement之間應該會有所關聯。

因此也導出了我的想法：

如果單獨將correlated measurement跟measurement17之間做KNN呢？

 109550155.csv Complete (after deadline) · now	0.59157	0.59087	<input type="checkbox"/>
--	---------	---------	--------------------------

這個想法讓我的正確率提升了0.0006之多，而這個結果可能有兩個原因，一個他們的關聯性使得measurement17能更接近原本缺失的值，而我們先將這些填補完成也導致之後impute其他feature時（如measurement10）也能更接近原本缺失的值。

這個部分還有待更多實驗，但我認為這個是全部裡面最novel的idea。

先or後填measurement17？

我懷疑如果我們先將correlated measurement填滿，我們之後再做m17的impute時能有更多數據，從而得到更好的結果(我有將這個方法實作但註解保留的code)。

下圖的上面是後impute m17、下面則是先impute m17

Private Score ⓘ	Public Score ⓘ	:
0.59058	0.59072	
0.59068	0.59075	

會導致這個結果可能是原本完整的小數據就能找到很好的線性關係，那些不太正確的數據反而導致更多noise。

measurement\_avg is useful？

在看前幾名的文章時有看到這項能夠提升accuracy，但我加上去卻降低了，足見這是case by case的。

**Similar work:**

<https://www.kaggle.com/code/medali1992/tps-aug-logistic-regression>

<https://www.kaggle.com/code/takanashihumbert/tps-aug22-9th-solution>

<https://www.kaggle.com/code/pourchot/hunting-for-missing-values>

以上三篇同樣是以logistic-regression來做的。

第一篇採單個model預測，且並未使用scaler。

第二篇則使用了好幾個model來做預測，最後調整一個比例做為答案。

第三篇特別的是對於preprocess對更多measurement做線性回歸

而其他除了模型、preprocess參數的不同以及feature的選擇相異之外（我相信他們應該都有挑到能使他們模型效果最佳的feature），最大的差異點還是在我對KNNImputer的二次使用，才是我超越他們的主要原因。

## Summary

我覺得這次的作業跟上次完全不同，上次我雖然也寫了很久但主要是自己寫出bug，主要還是套套模型就結束了。但這次要靠神奇的feature engineering，可是同樣的這次也有許多前人的觀察以及分享，除了能夠從他們想法的基礎上來思考之外，還能夠學到一些很快速的寫法

（ex:output的方法），也使得我對kaggle參賽有了初步的認識，雖然不知道大部分同學是否都有找到快速衝榜的方式，不過這次我自己的code能通過baseline並經過進一步的調整還有突發奇想打敗榜上的第一名，就已經感到非常滿意了。