

# ML Final Project Report - 109550175

## 0. Github link of my final project :

Github repo : <https://github.com/za970120604/NYCU-2022-fall-ML-final-project>

Link to logistic regression model weight : <https://github.com/za970120604/NYCU-2022-fall-ML-final-project/blob/main/ML%20final%20project/Logistic%20Regression%20model%20weight/logistic.joblib>

Link to NN model weight : <https://github.com/za970120604/NYCU-2022-fall-ML-final-project/blob/main/ML%20final%20project/Neural%20Network%20model%20weight/NN.h5>

## 1. Brief introduction :

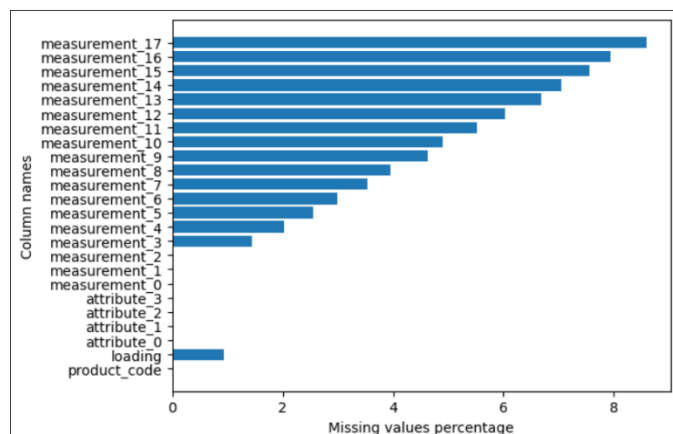
My method is based on simple Logistic-Regression model and some feature engineering method proposed by both other Kaggle competitor and me .

After doing the Grid Search for trying every possible combination of feature engineering and fine tune the model hyperparameter , this model can get a 0.59155 private score.

I also try a model based on Neural-Network and apply the same Grid Search process performed on Logistic-Regression based model , but the performance is not so well , only get a 0.58833 private score.

## 2. The first-step , a quick look into the dataset :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26570 entries, 0 to 26569
Data columns (total 24 columns):
#   Column              Non-Null Count  Dtype  
---  --
0   product_code        26570 non-null  object  
1   loading             26320 non-null  float64 
2   attribute_0         26570 non-null  object  
3   attribute_1         26570 non-null  object  
4   attribute_2         26570 non-null  int64   
5   attribute_3         26570 non-null  int64   
6   measurement_0       26570 non-null  int64   
7   measurement_1       26570 non-null  int64   
8   measurement_2       26570 non-null  int64   
9   measurement_3       26189 non-null  float64 
10  measurement_4       26032 non-null  float64 
11  measurement_5       25894 non-null  float64 
12  measurement_6       25774 non-null  float64 
13  measurement_7       25633 non-null  float64 
14  measurement_8       25522 non-null  float64 
15  measurement_9       25343 non-null  float64 
16  measurement_10      25270 non-null  float64 
17  measurement_11      25102 non-null  float64 
18  measurement_12      24969 non-null  float64 
19  measurement_13      24796 non-null  float64 
...
23  measurement_17      24286 non-null  float64 
dtypes: float64(16), int64(5), object(3)
```



The above 2 figure shows the attribute of this dataset(after dropping out "id" and "failure" columns) : the left one tells us that there are some "categorical" columns whose "Dtype" are labeled as "object" ; the right one tells us that there are quite a lot missing value in these columns , which means that a reasonable way to fill in these NaN blocks can help model perform well , and fortunately all categorical columns have no missing values , which saves our life.

### 3. A closer look into this dataset and derive some feature engineering ideas :

feature	fail	miss	failure rate	z	p-value
loading	: 44 / 250 = 0.176			-1.41	0.157
measurement_3	: 61 / 381 = 0.160			-2.50	0.012
measurement_4	: 128 / 538 = 0.238			1.43	0.151
measurement_5	: 172 / 676 = 0.254			2.66	0.008
measurement_6	: 171 / 796 = 0.215			0.15	0.879
measurement_7	: 197 / 937 = 0.210			-0.18	0.860
measurement_8	: 218 / 1048 = 0.208			-0.36	0.716
measurement_9	: 283 / 1227 = 0.231			1.54	0.123
measurement_10	: 277 / 1300 = 0.213			0.04	0.967
measurement_11	: 311 / 1468 = 0.212			-0.07	0.944
measurement_12	: 356 / 1601 = 0.222			0.95	0.340
measurement_13	: 373 / 1774 = 0.210			-0.24	0.809
measurement_14	: 413 / 1874 = 0.220			0.82	0.411
measurement_15	: 430 / 2009 = 0.214			0.16	0.876
measurement_16	: 436 / 2110 = 0.207			-0.67	0.502
measurement_17	: 499 / 2284 = 0.218			0.69	0.493



A missing feature might be the indicator of failure . As shown in above :

When measurement\_3 is missing, failure rate is 0.160 (much lower than average).

When measurement\_5 is missing, failure rate is 0.254 (much higher than average).

Not only considering the failure rate , these 2 columns have  $\text{abs}(z) > 2.5$  ;  
p-value < 2 % . It is sufficient to tell us that :

$P(\text{failure rates} \mid \text{missing measurement}_3)$  and

$P(\text{failure rates} \mid \text{missing measurement}_5)$

have a huge deviation when comparing to average failure rates , so we can consider adding 2 columns in the code :

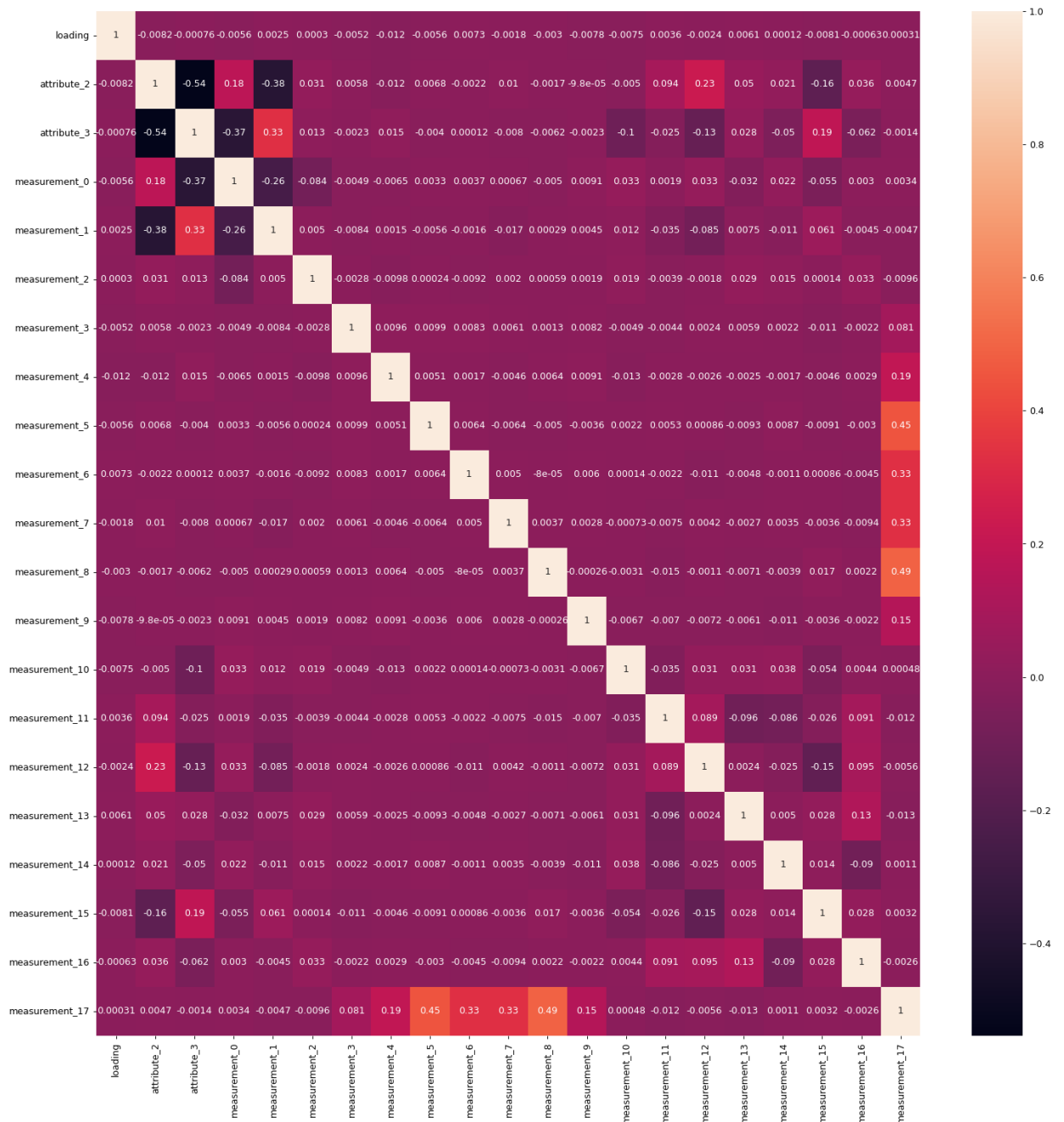
```
train_df['m_3_missing'] = train_df.measurement_3.isna()
train_df['m_5_missing'] = train_df.measurement_5.isna()
```

cited : <https://www.kaggle.com/code/ambrosm/tps22-eda-which-makes-sense/notebook>

```
attribute_2
6    10455
5     5765
8     5250
9     5100
Name: attribute_2, dtype: int64
attribute_3
8    11015
9     5343
6     5112
5     5100
Name: attribute_3, dtype: int64
product_code
C     5765
E     5343
B     5250
D     5112
A     5100
Name: product_code, dtype: int64
attribute_0
material_7    21320
material_5     5250
Name: attribute_0, dtype: int64
attribute_1
material_8    10865
material_5    10362
material_6     5343
Name: attribute_1, dtype: int64
```

Another feature engineering idea is that : although the dataset only gives us 3 categorical columns : product\_code , attribute\_0 and attribute\_1 , but we can observed that attribute\_2 and attribute\_3 only have 4 unique values , so we can

see both of them as categorical columns for the future imputation procedure.



Last but not least , after we printing out the correlations among different features using heatmap , we can see that attribute\_2 and attribute\_3 have higher negative correlation with other columns , so we can try to eliminate them to avoid the well known issue in machine learning data preprocessing : “problem of collinearity”

#### 4. Different model

```
def data_preprocessing(new_df , numerical_col , categorical_col):
    nn = []
    cc = []
    for f in new_df.columns:
        if f in numerical_col:
            nn.append(f)
        else:
            cc.append(f)
    # print(nn)
    # print(cc)
    numerical_pipe = Pipeline([
        ('impute', SimpleImputer()),
        ('scale', MinMaxScaler()),
    ])
    categorical_pipe = Pipeline([
        ('encode', OrdinalEncoder())
    ])
    preprocessor = ColumnTransformer([
        ('numeric', numerical_pipe, nn),
        ('categorical', categorical_pipe, cc),
    ])
    return preprocessor
```

Before trying different model , we need to impute the missing value in the dataset and scale them in order to make every block in the dataset contribute the same. In this project I use SimpleImputer() and MinMaxScaler() for numerical columns , and OrdinalEncoder() for categorical columns(Note that I mentioned that we can try to see attribute\_2 and attribute\_3 as categorical column , and they also don't have missing value just as the original categorical column product\_code , attribute\_0 and attribute\_1 , so we don't need imputer in the categorical pipeline).

```
def feature_engineering(df , num_col , cat_col , try_add_m3m5_missing , try_self_defined_columns , try_drop_attr2 , try_drop_attr3):
    df_modified = df.copy()
    if try_add_m3m5_missing == True:
        df_modified['m_3_missing'] = df_modified.measurement_3.isna()
        df_modified['m_5_missing'] = df_modified.measurement_5.isna()
        cat_col = cat_col + ["m_3_missing" , "m_5_missing"]
    if try_drop_attr2 == True :
        df_modified = df_modified.drop(["attribute_2"] , axis = 1)
        num_col = list(set(num_col) - set(["attribute_2"]))
    if try_drop_attr3 == True :
        df_modified = df_modified.drop(["attribute_3"] , axis = 1)
        num_col = list(set(num_col) - set(["attribute_3"]))
    if try_self_defined_columns == True and try_drop_attr2 == False and try_drop_attr3 == False:
        cat_col = cat_col + ["attribute_2" , "attribute_3"]
        num_col = list(set(num_col) - set(["attribute_2" , "attribute_3"]))
    return df_modified , num_col , cat_col
```

And the above code is for feature engineering ,

“try\_add\_m3m5\_missing” is whether to add 2 additional columns (m\_3\_missng and m\_5\_missing)to the original dataframe.

“try\_self\_defined\_columns” is whether to see attribute\_2 and attribute\_3 as categorical columns.

“try\_drop\_attrX” is whether to drop attribute\_X in the original dataset for training.

I'll run a big 2\*4 for loop for feature engineering Grid Search and submit all these result to Kaggle and pick the best private scored one for every model.

## Method 1 : Logistic-Regression based

```
import joblib
y = label_df.values
for try_add_m3m5_missing in [True , False]:
    for try_self_defined_columns in [True , False]:
        for try_drop_attr2 in [True , False]:
            for try_drop_attr3 in [True , False]:
                # Train
                new_train_df , num_col , cat_col = feature_engineering(train_df , numerical_columns ,
                                categorical_columns , try_add_m3m5_missing , try_self_defined_columns , try_drop_attr2 , try_drop_attr3)
                # print(num_col)
                # print(cat_col)
                preprocessor = data_preprocessing(new_train_df , num_col , cat_col)
                preprocessed_new_train = preprocessor.fit_transform(new_train_df)
                # print(np.argwhere(np.isnan(preprocessed_new_train)))
                model = LogisticRegression(**{'penalty': 'l1', 'tol': 0.031674753150821515, 'C': 0.1,
                'fit_intercept': True, 'solver': 'saga', 'max_iter': 2200})
                model.fit(preprocessed_new_train , y)
                joblib.dump(model , "logistic.joblib")
```

The above hyper parameter for model itself is the fine tuned one , and the best feature engineering parameter is “all True”

Submission and Description		Private Score	Public Score	Selected
	109550175.csv Complete (after deadline) · now	0.59155	0.5844	<input type="checkbox"/>

## Method 2 : NN based







```
y = label_df.values
for try_add_m3m5_missing in [True , False]:
    for try_self_defined_columns in [True , False]:
        for try_drop_attr2 in [True , False]:
            for try_drop_attr3 in [True , False]:
                # Train
                new_train_df , num_col , cat_col = feature_engineering(train_df , numerical_columns ,
                                categorical_columns , try_add_m3m5_missing , try_self_defined_columns , try_drop_attr2 , try_drop_attr3)
                preprocessor = data_preprocessing(new_train_df , num_col , cat_col)
                preprocessed_new_train = preprocessor.fit_transform(new_train_df)
                # print(np.argwhere(np.isnan(preprocessed_new_train)))
                model = keras.Sequential([
                    layers.Dense(32, input_shape=[preprocessed_new_train.shape[1]] , activation='swish'),
                    layers.Dropout(0.2),
                    layers.Dense(32, activation='swish'),
                    layers.Dropout(0.2),
                    layers.Dense(1, activation='sigmoid'),
                ])
                model.compile(
                    optimizer = tf.keras.optimizers.Adam(),
                    loss='binary_crossentropy',
                    metrics=[keras.metrics.AUC(name = 'auc')],
                )
                early_stopping = keras.callbacks.EarlyStopping(
                    patience=5,
                    min_delta=0.001,
                    restore_best_weights=True,
                )
                model.fit(
                    preprocessed_new_train, y,
                    batch_size = 128 ,
                    epochs=200,
                    callbacks=[early_stopping],
                    verbose = 0
                )
                model.save('NN_model.h5')
```

The above hyper parameter for model itself is the fine tuned one , and the best feature engineering parameter is “all True , without try\_self\_defined\_columns”

Submission and Description		Private Score	Public Score	Selected
	109550175_tryNN.csv Complete (after deadline) · now	0.58833	0.5829	<input type="checkbox"/>

After tons of times of trial and error , I figure out that the more complicated NN structure won't lead to higher private score , 2 or 3 layers is enough .

## 5. Abalation studies regarding feature engineering

Submission and Description	Private Score ⓘ	Public Score ⓘ	Selected
 <b>check.csv</b> Complete (after deadline) · now	0.5888	0.58331	<input type="checkbox"/>
 <b>check.csv</b> Complete (after deadline) · 1m ago	0.59062	0.58342	<input type="checkbox"/>
 <b>check.csv</b> Complete (after deadline) · 2m ago	0.5897	0.58353	<input type="checkbox"/>
 <b>check.csv</b> Complete (after deadline) · 2m ago	0.59004	0.58372	<input type="checkbox"/>
 <b>check.csv</b> Complete (after deadline) · 3m ago	0.58989	0.58317	<input type="checkbox"/>
 <b>check.csv</b> Complete (after deadline) · 4m ago	0.59008	0.58347	<input type="checkbox"/>

Below is 6 submissions without feature engineering(in the Grid Search for loop all 4 parameters set to False), which has an average of 0.589855 . So it shows that the feature engineering proposed above indeed does some improvement .  
(If TA try to evaluate this statement , simply put only “False” in every for loop in feature engineering Grid Search mentioned above)

## 6. Summary :

This seemingly complicated dataset can be solved by a simple logistic regression with a not bad answer surprisingly , and as refer to my previous experience the more complicated neural network architecture doesn't yield better result , with 2 or 3 levels is good enough. And I think the most valuable part of this project is that I figure out some reasonable feature engineering that actually help my score , and also my patient about spending such a long time on adjusting the hyperparameter. After finishing this project , I feel a great sense of accomplishment because this is the first Kaggle project I do it from scratch with a complete data preprocessing , feature engineering , model selection and fine tuning stages. This experience significantly enhance my enthusiasm to doing ML-related job.