



ML Final Project

🕒 Created	@December 29, 2022 11:00 AM
🏷️ Tags	

Environment Setup

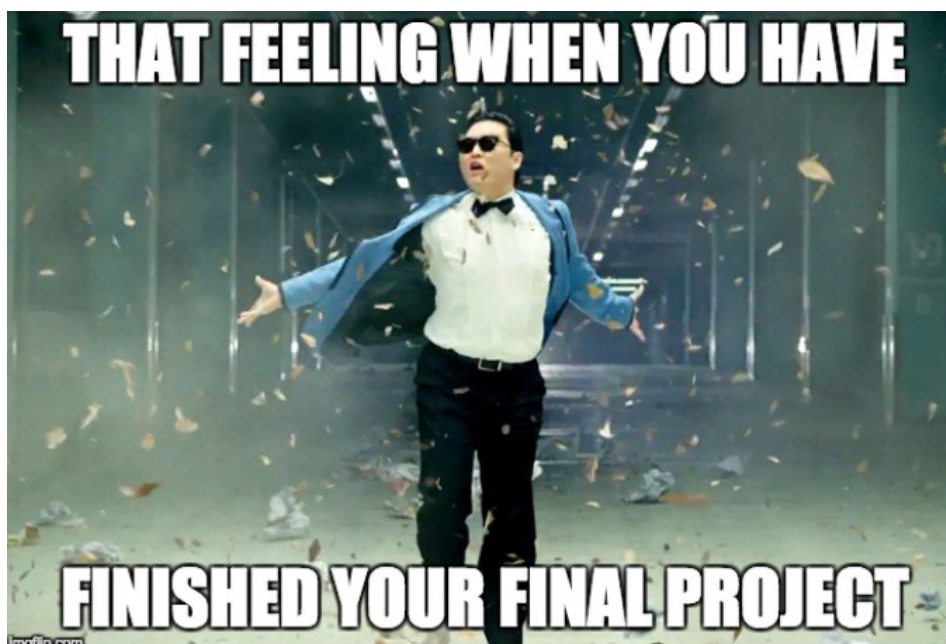
- Python 3.10.0

GitHub Link

- https://github.com/krz-max/ML_intro/tree/main/Final_Project

Model Weight

- <https://drive.google.com/drive/u/0/folders/1cnXAgGk6cMVc0HvN3ZOmyraj9tD7TSfW>
裡面的.sav檔



First Try

- 從discussion[1]得知 `measurement3 ~ measurement_16` 可以用mean, std結合成2個feature，以及將 `attribute_2`, `attribute_3` 兩個相乘後當成一個新的feature (`area`)。
 - 這時候data內的NaN我是直接用mean去填入
- 因為HW5我是用Pytorch做，所以想說先用Pytorch做看看。
- 整理好資料後直接利用三個FC層接看看會有甚麼結果

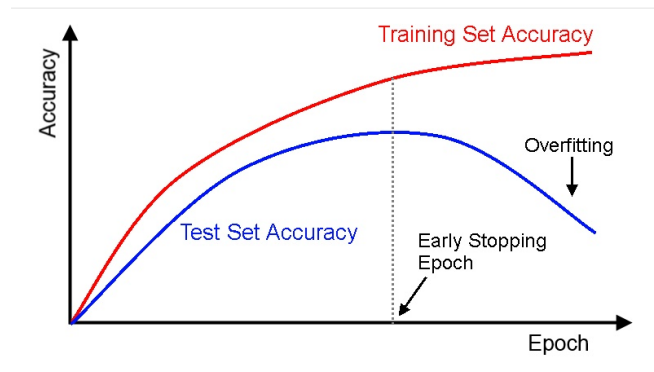
```
features = ['product_code', 'loading', 'attribute_0', 'attribute_1',
            'measurement_0', 'measurement_1', 'measurement_2', 'area',
            'measurement_avg', 'measurement_17']

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(10, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)
        self.output = nn.Sigmoid()
    def forward(self, x):
        return self.output(self.fc3(self.fc2(self.fc1(x))))
```

- loss function 改用 `nn.BCELoss()`，這是專門給binary classification用的
- optimizer用SGD，因為找不到適合Adam的learning rate，所以暫時先用SGD
- 原本以為只要接fully-connected layer應該就能train起來了，但後來看了一些文章才知道，batchnorm跟dropout layer的重要性
 1. batchnorm是將input的mini-batch做normalization，這樣可以去除在每一層計算完，產生的數值越來越大(或小)的影響，進而改善training的速度
 2. dropout則是隨機將某些node的權重設為0，避免資料的overfitting
- 所以第一次做的model基本上沒train出甚麼東西，private score最高只有0.52

Second Try

- 我後來參考在[2]的code，將模型改成跟他一樣(只是是用Pytorch)，發現還是train不起來，我認為可能的原因是：
 - 他有使用到 `tf.keras.callbacks.ReduceLROnPlateau` 以及 `tf.keras.callbacks.EarlyStopping`。這兩個function可以讓model在訓練的過程中若是遇到一些狀況時，對learning rate做一些調整，或是提早結束訓練。
 - [5]EarlyStopping



- ReduceLROnPlateau
 - [6]到模型訓練後期，模型逐漸接近全局最小值(Global minima)，適度地降低學習率，有助於找到最佳解(Global minima)。但模型仍有可能陷入鞍點(Saddle Point)，影響效能。
- 由於我查到的方法，若是要在Pytorch設定的話相對麻煩，加上我的電腦若是用neural network的話訓練的時間太長，所以我繼續尋找其他model。
- Model

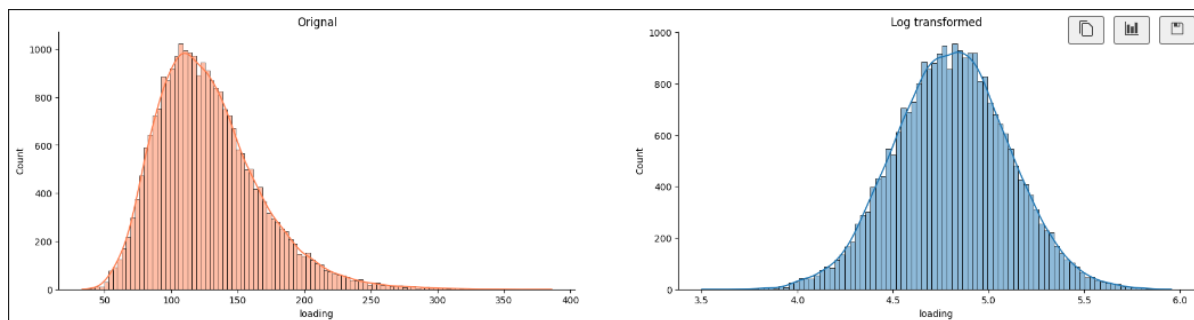
```
model = Sequential([
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(total_col, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(64, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(1, activation="sigmoid")
])
```

Preprocessing-Missing Value Imputation

- [2]使用 `HuberRegressor()`，[3]由於 `measurement_17` 和 `measurement_3` ~ `measurement_9` 有相關，對那些 `measurement_17` 有缺的資料，我們可以用其他measurement來預測他的值

```
data.loc[(data.product_code==code)&(data[column].isnull().sum(axis=1)==0)\
&(data['measurement_17'].isnull()), 'measurement_17'] \
= model.predict(tmp_test[column])
```

- 至於其他的missing value則是使用KNNImputer將失去的feature以K-nearest neighbor的該feature的平均填入
- [4] `loading` 的分布有點偏，可以取log讓他回到中間



- 不過我後來嘗試發現我自己跑出來的model用沒有取過log的 `loading` 來train表現比較好所以就拿掉了
- 在訓練時可以適度地對feature做scaling

Method	Score	Detail
StandardScaler	0.59184	數據的平均變為0且標準差變為1
MinMaxScaler	0.58958	最小值變成了0，最大值變成了1。數據會縮放到到[0,1]之間
MaxAbsScaler	0.59092	除以該列絕對值後的最大值。數據會縮放到到[-1,1]之間
RobustScaler	0.59072	用中位數和四分位數做標準化，較不受極值(outlier)影響
PowerTransformer	0.59156	transform each feature to make the data more Gaussian-like

- 整體的preprocess流程

- 產生新feature：

```
data['m3_missing'] = data['measurement_3'].isnull().astype(np.int8)
data['m5_missing'] = data['measurement_5'].isnull().astype(np.int8)
data['area'] = data['attribute_2'] * data['attribute_3']
```

- 選擇用來表現 `measurement_17` 的特徵，並用 `HuberRegressor()` 來填入該值，特徵的選擇：

```
full_fill_dict = {
    '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'],
    'F': ['measurement_4', 'measurement_5', 'measurement_6', 'measurement_7'],
    'G': ['measurement_4', 'measurement_6', 'measurement_8', 'measurement_9'],
    'H': ['measurement_4', 'measurement_5', 'measurement_7', 'measurement_8', 'measurement_9'],
    'I': ['measurement_3', 'measurement_7', 'measurement_8']
}
```

- 由於有些資料有缺失，不一定所有的空格都可以用前一步填滿，其他的資料使用 `KNNImputer` 做填入

```
model = HuberRegressor()
model.fit(tmp_train[column], tmp_train['measurement_17'])
data.loc[(data.product_code==code)&(data[column].isnull().sum(axis=1)==0)&(data['measurement_17'].isnull()), 'measurement_17'] = model.predict(tmp_test[column])
model2 = KNNImputer(n_neighbors=5)
print(f"KNN imputing code {code}")
data.loc[data.product_code==code, features] = model2.fit_transform(data.loc[data.product_code==code, features])
```

- 最後再用 `woeencoder` 把 `attribute_0` 的類別資料改為數值資料

- 有嘗試labelEncoder()但結果是一樣的，應該是因為attribute_0只有兩個label，直接用labelencoder轉成{0, 1}跟woeencoder轉成另外兩個數值，代表的意義是相同的。

Logistic Regression(LR)

- 後來我找到[9]的code來做參考，他是使用4個LR model來做預測，最後再用權重分配來預測，避免overfitting。Imputation也是按照上面的方式
- 除了以上，有幾個地方比較特別的就是：
 1. 手動挑選相關性高的feature：這部分可以參考[**Select Feature Importance**]，經過一些自己的嘗試有試出比原本更好的分數
 2. 模型預測不是採用原本的{0, 1}而是使用排序

Ranking as Binary Classification

- 我在trace code的過程，對於他使用ranking(排序)來當label的方式不是很理解，所以嘗試過把前半的product預測成沒有failure，但是分數卻只有0.56。

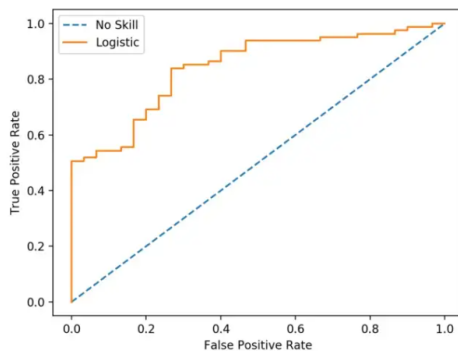
	id	failure	lr0	rank0
0	26570	7199.0	0.204152	7199.0
1	26571	4342.0	0.199444	4342.0
2	26572	6013.0	0.202289	6013.0
3	26573	5074.0	0.200723	5074.0
4	26574	19959.0	0.241204	19959.0

One LR Model Prediction

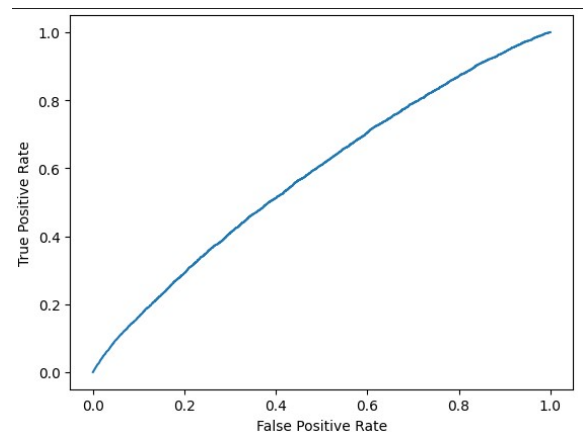
- 後來我直接丟上kaggle卻跑出了0.59的成績，所以我去查了些資料，發現可以這樣理解用排序來做二元分類的想法：[11]比如電商場景下，我們更關心用戶回購商品A的機率是否高於回購B的機率，然後把回購機率高的商品放在推薦列表前面，這就是一個典型的排序分類問題。
- 既然不是使用預測成功率，該如何對model評分？

How to measure the performance?

- 使用ranking當結果時，評分得改用ROC的AUC算：
 - 基本上他是用4種預測結果：True/False Positive/Negative 來對這個model做評分
 - $\text{True Positive Rate} = (\text{True Positive}) / (\text{True Positive} + \text{False Negative})$
 - $\text{False Positive Rate} = (\text{False Negative}) / (\text{False Positive} + \text{True Negative})$
 - **ROC Curve**是一個橫軸為False Positive Rate縱軸為True Positive Rate的Curve，從原點開始往右，對一個好的model而言，ROC curve會嚴格遞增，可以依此用來對model做評分
 - 可以想成當x往右的時候，對同個母群要達到False Positive Rate = x，會有多少True Positive rate = y
 - 越低的False Positive加上越高的True Positive代表這個模型預測的越好，所以才會用AUC當作評分標準
- ROC curve & AUC(Area Under Curve)
- No Skill代表隨機猜產生的曲線
- ROC of my model

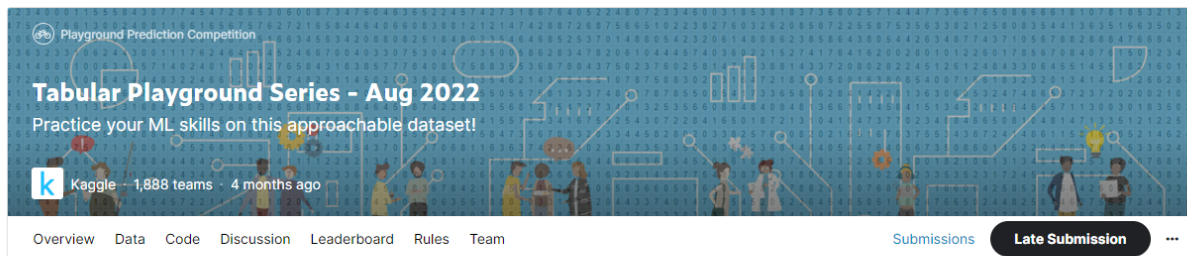


Example of ROC Curve



Why 4 Models?

- 後來了解roc_auc_score代表的意義之後，我看了一下自己的模型，發現第一組特徵(loading measurement_17 m5_missing attribute_0)會有最高的分數，所以我嘗試改變權重，讓他的權重高一點，預測分數也跟著上升，且依據分配的權重，分數會持續上升。
- 最後我索性只留下一個model，得到了0.59226的分數，提高了超過0.0003 Private Score



Submissions




You selected 0 of 2 submissions to be evaluated for your final leaderboard score. Since you selected less than 2 submission, Kaggle auto-selected up to 2 submissions from among your public best-scoring unselected submissions for evaluation. The evaluated submission with the best Private Score is used for your final score.

0/2

Submissions evaluated for final score

All Successful Selected Errors

Recent

Submission and Description	Private Score	Public Score	Selected
 109511028.csv Complete (after deadline) · now	0.59224	0.58879	<input type="checkbox"/>
 submission.csv Complete (after deadline) · 1d ago	0.59224	0.58879	<input type="checkbox"/>
 submission.csv Complete (after deadline) · 1d ago	0.59226	0.58885	<input type="checkbox"/>

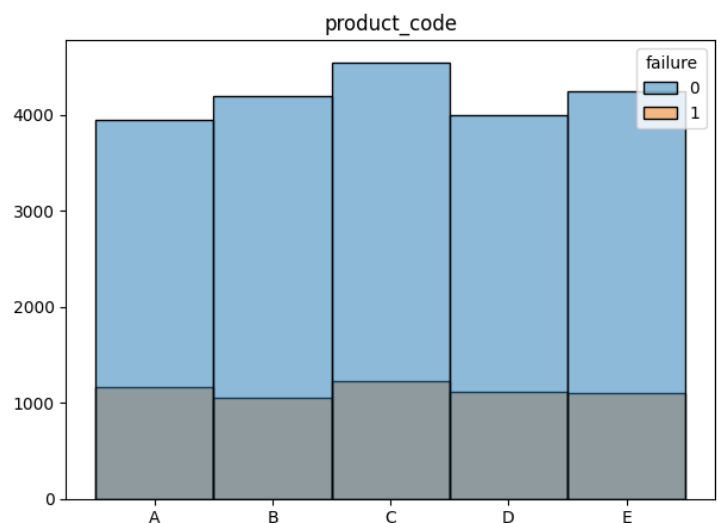
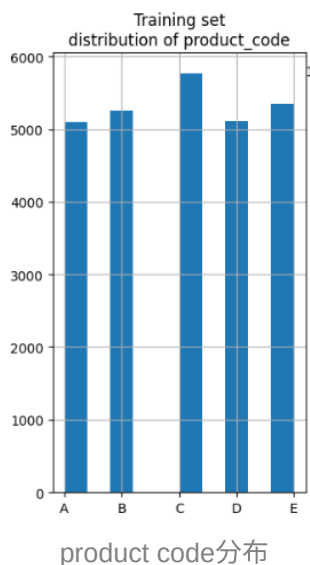
Select Feature Importance

- 由 `sklearn.feature_selection` 的 `SelectFromModel` 得知 `loading` `measurement_17` 和 `area` 是最相關的。根據此結果，固定好這三項特徵後，搭配其他的來看LR最後訓練出來選擇特徵的結果。發現 `m5_missing` `attribute_0` `m3_missing` `measurement_2` 是能夠表現比較好的特徵，加上 `m5_missing` 被選到的次數更高，所以最後的選擇是(1 LR model)：

```
1 # Not sure why `m5_missing` is not selected from the code above, but it has greater
2 select_features = ['loading', 'measurement_17', 'm5_missing', 'attribute_0']
✓ 0.6s
```

不同方式的比較

- k-fold : GroupKFold & StratifiedKFold
 - Fold的方式分別是 1)依照某個label對資料進行分組來做cross-validation 2)依照 target value的比例來進行分組
 - 使用4個LR時，兩個結果相差不多，分別是0.5918和0.59184
 - 改成只用一個LR時，兩個結果是完全一樣的
 - 我認為結果相似的原因在於我都是分成5組(`n_splits=5`)，加上training data的 `product_code` 分布的蠻平均的，所以選到的分布會差不多。



不同product中，failure所占的比率

Reference Links

1. **[Less can be more: Feature Engineering Ideas]**
<https://www.kaggle.com/competitions/tabular-playground-series-aug-2022/discussion/342126>
2. **[TPS AUG Neural Network]** <https://www.kaggle.com/code/nourhadrich/tps-aug-neural-network>
3. **[Easy imputation with correlation 👍]**
<https://www.kaggle.com/code/purist1024/easy-imputation-with-correlation>
4. **[TPS AUG'22 Top 2% : Logistic Regression + CV & FE]**
<https://www.kaggle.com/code/vishnu123/tps-aug-22-top-2-logistic-regression-cv-fe>
5. **[TF.Keras EarlyStopping Callbacks 介紹]**
<https://cynthiachuang.github.io/EarlyStopping-Callback/>
6. **[訓練模型-Learning Rate]** <https://ithelp.ithome.com.tw/articles/10273302>
7. **[資料清理&前處理]** <https://ithelp.ithome.com.tw/articles/10240494>
8. **[特徵工程-嘔心之作—深度瞭解特徵工程]**
<https://www.796t.com/content/1542164169.html>
9. **[TPS-Aug22 9th solution]**<https://www.kaggle.com/code/takanashihumbert/tps-aug22-9th-solution/notebook>
10. **[Binary Classification]**<https://zhuanlan.zhihu.com/p/21263665>
11. **[Binary classification - 聊聊评价指标的那些事儿]**https://blog.csdn.net/weixin_30851867/article/details/94876069