

ML-Final-Report

109550116 楊傑宇

- GitHub link of your code:

https://github.com/jerrychild999922/ML_final

- Reference:

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

<https://www.kaggle.com/code/qw1zzard/tps-aug-2022>

- Brief introduction:

首先，我需要先把資料的空缺補上(preprocessing)，並增加幾個好用的 features，最後使用 logistic regression 配合 fold 來 train 再輸出 submission。

- Methodology:

1. Data-preprocessing:

這部分其實是最沒有方向的，所以我看了一份 discussion—hunting for missing value，裡面說可以先用 huber regressor 用線性的方式先補一部分，剩下的再用 KNNImputer 補完。

```
for code in data.product_code.unique():
    for m_col in list(full.keys()):
        tmp = data[data.product_code == code]
        column = full[m_col][code]
        ttrain = tmp[column + [m_col]].dropna(how="any")
        #print(tmp_train)
        ttest = tmp[(tmp[column].isnull().sum(axis=1) == 0) & (tmp[m_col].isnull())]
        #print(tmp_test)
        model = HuberRegressor(epsilon=1.9,max_iter=3000)
        model.fit(ttrain[column], ttrain[m_col])
        data.loc[(data.product_code == code) & (data[column].isnull().sum(axis=1) == 0) & (data[m_col].isnull()), m_col] = model.predict(data.loc[(data.product_code == code) & (data[column].isnull().sum(axis=1) == 0) & (data[m_col].isnull()), column])

NA = data.loc[data["product_code"] == code, n_col].isnull().sum().sum()
model1 = KNNImputer(n_neighbors=3)
data.loc[data.product_code == code, feature] = model1.fit_transform(data.loc[data.product_code == code, feature])
```

2. Feature select:

我看了很多人的 discussion 分析了每個 feature 的重要性，確定的是當 measurement_3 不取時跟 failure 有負相關，反而 measurement_5 有正相關，因此我多了這兩個 features，再經過多次的取捨最後確定我選擇的 features

```
features = ["loading","attribute_0","measurement_17","measurement_0","measurement_1","measurement_2","m3_missing","m5_missing","measurement_avg"]
```

3. Model architecture:

我用的是 logistic regression 配合不同的 fold 來 train。

```

for fold in folds_dict.keys():
    print(f'#### {fold} ####')
    xtest = test.copy()
    #print(df)
    xtrain, ytrain = df[df['product_code'].isin(folds_dict[fold][0])][FEATURES].values, df[df['product_code'].isin(folds_dict[fold][0])][target].values
    xvalid, yvalid = df[df['product_code'].isin(folds_dict[fold][1])][FEATURES].values, df[df['product_code'].isin(folds_dict[fold][1])][target].values
    #print(xtrain)
    model = LogisticRegression(max_iter=200, C=0.0001, penalty="l2", solver="newton-cg")
    model.fit(xtrain, ytrain)
    preds_valid = model.predict_proba(xvalid)[:, 1]
    test_preds = model.predict_proba(xtest.values)[:, 1]
    #print(test_preds)
    #final_test_predictions.append(test_preds)
    fold_score = roc_auc_score(yvalid, preds_valid)
    print(fold_score)
#final_test_predictions.append(test_preds)
return (model, final_test_predictions)

```

4. Hyperparameters:

```

model = HuberRegressor(epsilon=1.9, max_iter=3000)
model1 = KNNImputer(n_neighbors=3)
model = LogisticRegression(max_iter=200, C=0.0001, penalty="l2",
solver="newton-cg")


```

在 KNNImputer 的 neighbors 有試過 1/3/5/7 後來發現 3 的效果最好在 logisticRegression 的 max_iter 有試過 200/1000/3000 發先效果一樣就選擇少的就好。

● Result :

因為我使用的是 logistic regression 他沒有辦法一直去更新他的 model，又加上我使用的是 n-fold，將 n 次的結果平均來當成 submission，然而儲存 model 時我不想儲存那麼多 model 所以我盡量挑最好的 model，所以我會有兩種 accuracy。

1. No-save-model:

 109550116submission.csv	0.59019	0.59019	<input type="checkbox"/>
Complete (after deadline) · 31m ago			

2. Save-model:

 109550116submission (5).csv	0.58996	0.59018	<input type="checkbox"/>
Complete (after deadline) · now			

(有一次我不小心去到某次 fold 有很高的分數，但忘記紀錄)

 109550116submission (2).csv	0.59041	0.58964	<input type="checkbox"/>
Complete (after deadline) · 18m ago			

● Summary:

這份作業一開始真的沒什麼方向，單開始看了很多人的結果逐漸有了雛形，然而當我自己實作完 accuracy 一直卡在 0.586...附近，嘗試了很多方法最後找到跟改 fold 取法，與 feature 的選擇，終於把 accuracy 衝上 0.59，真的很開心，也很有成就感，最後感謝助教與教授這學期的付出，我非常喜歡這種自己探索自己進步的教學模式。

- **Bonus:**

我看了很多 discussion 都在探討不同的 model 做出的效果，然而我認為取 fold 的方式會不會有也影響，因此我做了三種 fold 來比較。

1. 3 vs 2 fold:

主要就是將 train 拆成 3:2 去 train

```

folds_dict = {f'Fold 1': [['C', 'D', 'E'], ['A', 'B']],
              'Fold 2': [['B', 'D', 'E'], ['A', 'C']],
              'Fold 3': [['A', 'C', 'E'], ['B', 'D']],
              'Fold 4': [['B', 'C', 'D'], ['A', 'E']],
              'Fold 5': [['A', 'D', 'E'], ['B', 'C']],
              'Fold 6': [['A', 'B', 'C'], ['D', 'E']],
              'Fold 7': [['A', 'C', 'D'], ['B', 'E']],
              'Fold 8': [['A', 'B', 'E'], ['C', 'D']],
              'Fold 9': [['A', 'B', 'D'], ['C', 'E']],
              'Fold 10': [['B', 'C', 'E'], ['A', 'D']]}

```

2. K-fold:

與上幾次的功課 k-fold 一樣

```

folds_dict = {f'Fold 1': [['B', 'C', 'D', 'E'], ['A']],
              'Fold 2': [['A', 'B', 'D', 'E'], ['C']],
              'Fold 3': [['A', 'B', 'C', 'D'], ['E']],
              'Fold 4': [['A', 'B', 'D', 'E'], ['C']],
              'Fold 5': [['A', 'D', 'C', 'E'], ['B']]}

```

3. random-pick:

單純隨機分成 8:2

```

xtrain, xvalid = train_test_split(df, random_state=777, train_size=0.8)
#print(xtrain)
ytrain = xtrain[TARGET]
#print(ytrain)
yvalid = xvalid[TARGET]
xtrain = xtrain[FEATURES]
xvalid = xvalid[FEATURES]

```

這裡我只討論平均下來的結果，不討論存 model 的結果

 submission_no_fold (2).csv Complete (after deadline) · 3h ago	0.58953	0.59075	
 submission_kfold (3).csv Complete (after deadline) · 3h ago	0.59011	0.59024	
 submission (16).csv Complete (after deadline) · 3h ago	0.59022	0.59003	

可以看到 3 vs 2 的效果最好，其次是 k-fold，最後是 random，因此得出結論有時不只是改變參數可以變好，改變 fold 的取法也可以使 accuracy 增加。