Machine Learning in Engineering Science

N96084094 彭巧緣

期中報告

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('./ml-nckues-2020/train.csv') # trianing set
data.dropna(axis=0, inplace=True) # drop the data with space
data_r = data[data.select_dtypes(include=[np.number]).ge(0).all(1)] # drop the data with the value less than 0

X_origin = data_r.iloc[:,1:data_r.shape[1]-1]
y_origin = data_r.Label

# data_r['Label'].value_counts() # count for the number of the labels
```

將有空白的資料與有負值的資料視為不具參考性的資料，直接將其移除。

In [2]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

sel2 = SelectKBest(chi2, k=60) # chi-square distribution, select the best 60 features
X_new = sel2.fit_transform(X_origin, y_origin)

X_new = pd.DataFrame(X_new) # array to dataframe
y_origin.reset_index(drop=True,inplace=True) # reset the index from 0
```

用卡方分佈找出 60 個較有辨識性的特徵。

In [3]:
```
data_r = pd.concat([X_new,y_origin],axis=1)

excellent = data_r[data_r["Label"]=='Excellent']
good = data_r[data_r["Label"]=='good']
fair = data_r[data_r["Label"]=='fair']
bad = data_r[data_r["Label"]=='bad']

data_r_2 = pd.concat([data_r,fair.sample(frac=0.2),fair.sample(frac=0.2),
                      fair.sample(frac=0.2),fair.sample(frac=0.2)]) # let the number of data = 1.2*data
X = data_r_2.iloc[:,0:data_r_2.shape[1]-1]
y = data_r_2.Label

# data_r_2['Label'].value_counts() # count for the number of the labels
```

```
fair          3536
good          2943
bad            919
Excellent      302
Name: Label, dtype: int64
```

圖一 ： 原始數據數量

```
fair          6364
good          2943
bad            919
Excellent      302
Name: Label, dtype: int64
```

圖二 ： 1.2 倍數據數量

將數據數量增加 0.2 倍，讓數據整體增加，使後面在訓練時數據多一點。左圖為原始數據數量，右圖為將所有數據複製 1.2 倍。

In [4]:
```
from imblearn.over_sampling import SMOTE

pd.set_option('mode.chained_assignment', None) # set down the warning, do not use it in recommend

sm = SMOTE(random_state=42) # let the number of data of every label be the same
X_res, y_res = sm.fit_resample(X, y)

X_temp1 = X_res.iloc[:,0:X_res.shape[1]-2] # the data in binary
X_temp2 = X_res.iloc[:,X_res.shape[1]-2:X_res.shape[1]] # the data in real number

X_temp1[X_temp1<0.5] = 0 # round for the value into binary
X_temp1[X_temp1>=0.5] = 1 # round for the value into binary

data_r_2 = pd.concat([X_temp1,X_temp2,y_res],axis=1)

# data_r_2['Label'].value_counts() # count for the number of the labels
```

從前面的圖可以發現各 label 的數量資料不平均，這會導致訓練出來的 model 預測能力不好，因此利用 SMOTE 指令合成數據，將所有 label 的數據數量統一，並得下圖。

```
good          6364
bad           6364
fair          6364
Excellent     6364
Name: Label, dtype: int64
```

SMOTE 後的數據數量

```
In [5]: from sklearn.utils import shuffle

        data_random = shuffle(data_r_2) # random the order

        X_shuffle = data_random.iloc[:,0:data_random.shape[1]-1]
        y_shuffle = data_random.Label
```

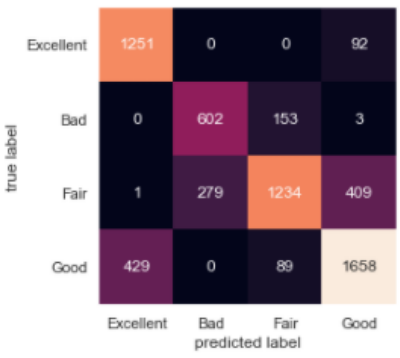
smote 過後的資料會照 label 順序排好，但這樣會讓 model 訓練效果不好，所以用 shuffle 將資料順序打亂。

```
In [6]: from sklearn.preprocessing import StandardScaler

        scaler = StandardScaler() # mean to zero, standard deviation to 1
        scaler.fit(X_shuffle) # set the model according to every feature respectively
        X_scaler = scaler.transform(X_shuffle)

        data_val = pd.read_csv('./ml-nckues-2020/val.csv') # validation set
        X_val_o = data_val.iloc[:,1:data_val.shape[1]-1]
        y_val = data_val.Label

        X_val = sel2.transform(X_val_o) # drop the features
        X_val_scaler = scaler.transform(X_val) # StandardScaler
```

利用 StandardScaler 將所有資料常態分佈化，使平均值會變為 0, 標準差變為 1，讓離群值影響降低。

```
In [7]: from xgboost.sklearn import XGBClassifier
        from sklearn.ensemble import VotingClassifier
        from sklearn.ensemble import BaggingClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import GradientBoostingClassifier

        # for voting
        model_rf = RandomForestClassifier(n_estimators=100, random_state=42, max_depth=10, max_features=0.8) # for bagging

        grb_clf = GradientBoostingClassifier(n_estimators=100,random_state=0)
        rnd_clf = BaggingClassifier(base_estimator=model_rf, n_estimators=10, random_state=0)
        xgb_clf = XGBClassifier(silent=0 ,learning_rate= 0.3, max_depth=6, gamma=0,subsample=1, max_delta_step=0,
                                colsample_bytree=1, reg_lambda=1, n_estimators=100, seed=1000)
        lr_clf = LogisticRegression(random_state=0,max_iter=1000,solver='newton-cg')

        # voting according to the probability(soft)
        model = VotingClassifier(estimators=[('gb', grb_clf), ('rf', rnd_clf),('xgb',xgb_clf),('lr',lr_clf)],voting='soft')

        model.fit(X_scaler,y_shuffle) # train the model

        testing_score = model.score(X_val_scaler,y_val) # validation
        print('testing scores : ',testing_score)

        testing scores :  0.8140322580645162
```

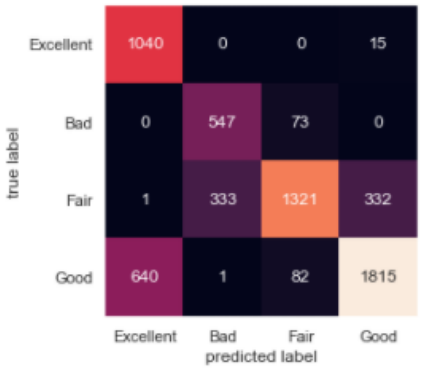
這裡使用 voting 分類器，因此要選擇適當的 classifier，下圖為四種 classifier—XGB、RandomForest、GradientBoosting 和 LogisticRegression 分類器的熱點圖。



RandomForest 分類器

GradientBoosting 分類器

LogisticRegression 分類器



XGB 分類器

這裡分為熱點圖的右上與左下去探討。右上有許多錯誤值，但可發現 XGB 與 RandomForest 錯誤數量較少，因此選用這兩個 classifier；左下亦有較多的錯誤數量，而 GradientBoosting 與 LogisticRegression 在左下的錯誤數量較少。總結上述四張圖，因此在此選用此四種 classifier 去做 voting。

In [8]:
```
data_test = pd.read_csv('./ml-nckues-2020/test.csv') # testing data

X_test = data_test.iloc[:,1:(data_test.shape[1])]
X_test_select = sel2.transform(X_test) # drop the features
X_test_scaler = scaler.transform(X_test_select) # StandardScaler

y_pred = model.predict(X_test_scaler) # pedict

y_pred_pd = pd.DataFrame(data=y_pred, columns=['Label']) # save the data to DataFrame

# change the label
y_pred_pd[y_pred_pd['Label'] == 'bad'] = 4
y_pred_pd[y_pred_pd['Label'] == 'fair'] = 3
y_pred_pd[y_pred_pd['Label'] == 'good'] = 2
y_pred_pd[y_pred_pd['Label'] == 'Excellent'] = 1

y_pred_pd = y_pred_pd.reset_index() # reset the index
y_pred_pd.to_csv('n96084094_22.csv',index=False) # save the data
```

在做 predict 前，先將 test 的資料刪除多餘的特徵與進行歸一化，最後輸出檔案。