Data Science HW1

B10915030 陳奕軒

1. 引入會用到的 library

```
#Essential
import pandas as pd
import numpy as np

#EDA
import seaborn as sns
from matplotlib import pyplot as plt

#Pipeline
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as imbPipeline
from sklearn.compose import ColumnTransformer

#Preprocess
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, OrdinalEncoder
#Up & down sampling for unbalanced data
from imblearn.combine import SMOTETomek

#Model
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

#Model selection
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
```

2. 資料前處理

i. 資料輸入

```
train = pd.read_csv('../input/2022-data-science-hw1/train.csv')
test = pd.read_csv('../input/2022-data-science-hw1/test.csv')
```

ii. 特徵選擇

```
features = train.columns[0: 16].tolist()
print("Selected features=")
print(features)

train_data = train[features]

#Yes/No to 1/0
y = train['Attribute17'].map({'Yes': 1, 'No': 0})
```

X 暫時選擇 Feature1~16

Y 為 Feature17

iii. 觀察資料是否平衡

```
y.value_counts()
```

```
0    13965
1     3138
Name: Attribute17, dtype: int64
```

由以上得知資料不平衡，需要特殊處理。

iv. 特殊狀況處理

- Attribute1 日期

```
#Divide by '-'
dateTime = pd.DatetimeIndex(train_data['Attribute1'])
#Train
#Year
train_data['Year'] = dateTime.year
train_data['Year'] = train_data['Year'].astype(str)
#Month
train_data['Month'] = dateTime.month
train_data['Month'] = train_data['Month'].astype(str)
#Drop
train_data.drop('Attribute1', axis=1, inplace=True)

#Divide by '-'
dateTime = pd.DatetimeIndex(test['Attribute1'])
#Train
#Year
test['Year'] = dateTime.year
test['Year'] = test['Year'].astype(str)
#Month
test['Month'] = dateTime.month
test['Month'] = test['Month'].astype(str)
#Drop
test.drop('Attribute1', axis=1, inplace=True)
```

取出年份、月份作為特徵，並刪除原本日期。

- Attribute2 氣象站地區

```
#Train
#train_data['Attribute2'] = train_data['Attribute2'].astype(str)
train_data.drop(['Attribute2'], axis=1)
#Test
#test['Attribute2'] = test['Attribute2'].astype(str)
test.drop(['Attribute2'], axis=1)
```

因為地區特徵值種類過多且不平均，所以刪除。

- Attribute14 下午三點,雲層遮蓋天空的比例

```
#Train
train_data['Attribute14'] = train_data['Attribute14'].astype(str)
#Test
test['Attribute14'] = test['Attribute14'].astype(str)
```

將原本的 numerical feature 轉換為 categorical feature，因為

地區並不是數值，比較像 Encode 後的結果。

v. 分類

```
# category column
categorical_cols = [cname for cname in train_data.columns
                    if train_data[cname].dtype == "object"]


# numerical columns
numerical_cols = [cname for cname in train_data.columns
                  if train_data[cname].dtype in ['int64', 'float64']]

print("Categorical columns:")
print(categorical_cols)

print("Numerical columns:")
print(numerical_cols)
```

```
Categorical columns:
['Attribute2', 'Attribute8', 'Attribute10', 'Attribute14', 'Attribute16', 'Year', 'Month']
Numerical columns:
['Attribute3', 'Attribute4', 'Attribute5', 'Attribute6', 'Attribute7', 'Attribute9', 'Attribute11', 'Attribute12', 'Attribute13', 'Attribute15']
```

根據 Feature 的 dtype，將他們分類為 Numerical column 和 Categorical column，

3. 定義前處理器

```
# 數值前處理
# Fill na with median
# Use scaler to norm
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()) #comment if using randomforest
])

# 分類前處理
# Fill na with most_frequent
# Using label encoder
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    #('encoder', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# 前處理綁在一起
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

為了簡化流程和程式碼，所以採用 pipeline 做為處理數值的方法。這邊定義兩個類別的前處理器：

- Numerical：使用中位數填補 null value。
- Categorical：採用 One hot Encoder，並忽略沒看過的值。

4. 尋找最佳的超參數

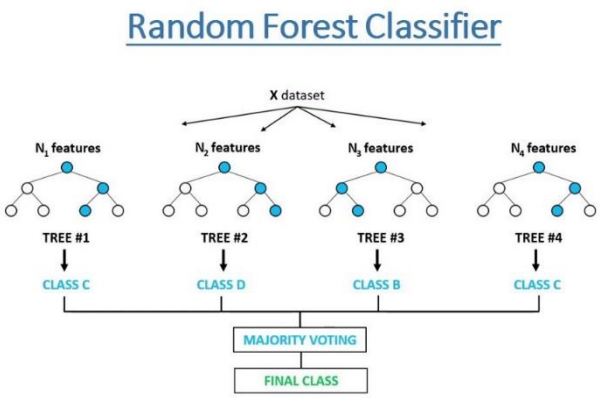
i. 定義流程

```
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', RandomForestClassifier(random_state=1))
])

my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', KNeighborsClassifier())
])
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', LogisticRegression(max_iter=1000))
])
```

這裡我測試了不同的 Classifier：Random forest classifier, KNN, Logistic regression。基於 kaggle 和 local 上的 score，我選用 Logistic 和 Random forest。這兩種 Classifier 都使用了前面定義的 preprocessor、SMOTETomek（同時進行 up sample 和 down sample 的工具）。

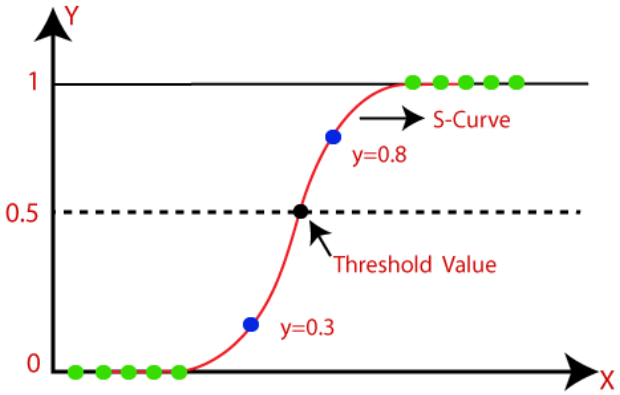
- Random forest 簡介：



如圖，Random forest 由多個 Decision tree 組成，每個 tree 都負責判斷不同的 variable，該 model 透過隨機的方式決定哪個 tree

得到哪些 data。

- Logistic regression 簡介：



如圖，該 model 會在資料點上找到一條線，來切分兩種不同的 class，資料比較靠近誰就分到哪類。

ii. 網格搜索

- 候選超參數

```
"""
param_grid = {
    'model__n_estimators': [300, 350, 400, 450, 500],
    'model__max_features': ['auto', None, 'log2'],
    'model__max_depth' : [3, 4, 5, 6, 7],
    'model__criterion' :['gini', 'entropy']
}
param_grid = {
    'model__n_neighbors': [3, 4, 5, 6, 7]
}
"""
```

  - RandomForest：
    - n_estimators：森林中樹木的數量
    - max_features：考慮 feature 的數目
    - max_depth：最大深度
    - criterion：衡量 split 後的品質的方法
  - KNN：

- n_neighbors：一次選擇多少鄰近值

- Logistic
  - None

- Fit

```
#cv -> StratifiedKFold
#grid = GridSearchCV(my_pipeline, param_grid=param_grid, cv=5, scoring='f1')
grid = GridSearchCV(my_pipeline, param_grid=param_grid, cv=5, scoring='accuracy')
grid_result = grid.fit(X, y)
```

我們使用上面建立的流程，並從定義好的超參數範圍進行搜索，衡量的指標有兩種版本，一種以 F1 score（結合精準度、召回率）做為衡量指標，另一種為直觀的準確度。

- 最佳 hyper parameter

```
#print(f'最佳F1值：{grid_result.best_score_}，最佳參數組合：{grid_result.best_params_}')
print(f'最佳準確值：{grid_result.best_score_}，最佳參數組合：{grid_result.best_params_}')
```

Random Forest：

```
18366.5s    47    最佳F1值：0.8119069524146525，最佳參數組合：{'model__criterion': 'gini', 'model__max_depth': 7, 'model__max_features': 'auto', 'model__n_estimators': 350}
```

註：此處應為 Accuracy (typo)

KNN：

```
最佳F1值：0.4819047203196529，最佳參數組合：{'model__n_neighbors': 6}
```

- 模型

```
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', RandomForestClassifier(
        random_state=1,
        max_features='auto',
        n_estimators=350,
        max_depth=7,
        criterion='gini'
        #max_features=grid_result.best_params_['model__max_features'],
        #n_estimators=grid_result.best_params_['model__n_estimators'],
        #max_depth=grid_result.best_params_['model__max_depth'],
        #criterion=grid_result.best_params_['model__criterion']
    ))
])

my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    #('model', KNeighborsClassifier(n_neighbors=grid_result.best_params_['model__n_neighbors']))
    ('model', KNeighborsClassifier(n_neighbors=6))
])
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', LogisticRegression(random_state=))
])
```

為求提交方便，所以直接從 best_params_屬性內獲得資訊，並註解掉 GridsearchCV。如果日後要使用模型的話需要直接修改參數，避免冗長的找參數過程。

- 訓練＆預測

```
my_pipeline.fit(X, y)
```

```
result = my_pipeline.predict(test_X)
```

- 輸出

```
result_df = pd.DataFrame({'id': [float(id) for id in range(len(result))],
                          'ans' : result.tolist()})

result_df.to_csv('result.csv', index=False)
```

輸出為 CSV 檔。

5. 如何執行

- Random Forest

- 註解 scaler

```
# 數值前處理
# Fill na with median
# Use scaler to norm
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()) #comment if using randomforest
])

# 分類前處理
# Fill na with most_frequent
# Using label encoder
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    #('encoder', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1))
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# 前處理綁在一起
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

- 取消 Random Forest 的註解並註解其他 code

```
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', RandomForestClassifier(random_state=1))
])

my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', KNeighborsClassifier())
])
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', LogisticRegression())
])
```

- 取消 Random Forest 的註解並註解其他 code，同時使用建議的 Hyperparameter

```
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', RandomForestClassifier(
        random_state=1,
        max_features='auto',
        n_estimators=350,
        max_depth=7,
        criterion='gini'
        #max_features=grid_result.best_params_['model__max_features'],
        #n_estimators=grid_result.best_params_['model__n_estimators'],
        #max_depth=grid_result.best_params_['model__max_depth'],
        #criterion=grid_result.best_params_['model__criterion']
    ))
])

my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    #('model', KNeighborsClassifier(n_neighbors=grid_result.best_params_['model__n_neighbors']))
    ('model', KNeighborsClassifier(n_neighbors=6))
])
"""
my_pipeline = imbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('UpSample & DownSample', SMOTETomek(random_state=1)),
    ('model', LogisticRegression(random_state=))
])
```

- Logistic Regression
  - 維持原樣即可