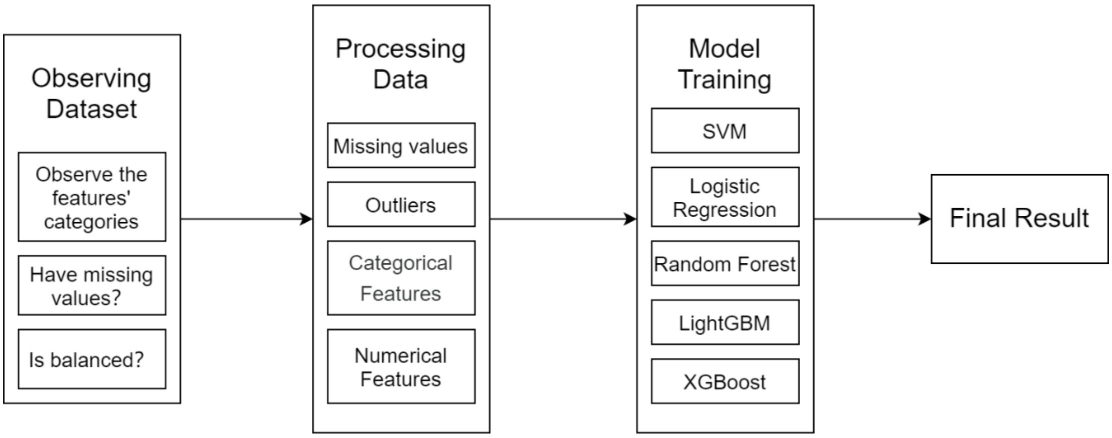
Data Science HW1

A10715006 張秋霞

https://www.kaggle.com/c/2020-data-science-hw1
Dataset：train.csv、test.csv
目標：利用每日天氣觀測樣本做訓練，給入當天的觀測數據，希望能預測隔天會不會降雨。
solution **程式碼檔案**：rain_tomorrow.ipynb
執行方式：Jupyter 中打開，直接 run 即可。

程式架構



演算法流程&實作思路

觀察資料集

1.首先讀取資料，並通過 head()、shape、info()查看資料集的 attributes 數量、資料數量以及 attributes 的資料類別。

```
In [2]: train_df = pd.read_csv('./input/train.csv')
        predict_df = pd.read_csv('./input/test.csv')
        train_df.head()
```

Out[2]:

	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	Attribute7	Attribute8	Attribute9	Attribute10	...	Attribute14	Attribute15	Attribute16	Attr
0	2008-12-02	2	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0	25.0	1010.6	
1	2008-12-05	2	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0	33.0	1010.8	
2	2009-01-10	2	17.0	30.8	0.0	NaN	NaN	NE	37.0	NNE	...	36.0	24.0	1013.4	
3	2009-01-19	2	13.9	36.6	0.0	NaN	NaN	WNW	39.0	SSE	...	39.0	10.0	1015.8	
4	2009-01-23	2	18.8	35.2	6.4	NaN	NaN	WNW	52.0	S	...	43.0	28.0	1007.9	

5 rows × 23 columns

```
In [3]: train_df.rename(columns={'Attribute1':"Today",'Attribute2':"Area",'Attribute3':"MinTemp",'Attribute4':"MaxTemp",'Attribute5':"Rainfall",
                                 'Attribute6':"Evap",'Attribute7':"Sunshine",'Attribute8':"StrWindDir",'Attribute9':"StrWindSpeed",'Attribute10':"WindD
                                 'Attribute11':"WindDir3pm",'Attribute12':"WindSpeed9am",'Attribute13':"WindSpeed3pm",'Attribute14':"Humi9am",'Attribut
                                 'Attribute16':"Press9am",'Attribute17':"Press3pm",'Attribute18':"Cloud9am",'Attribute19':"Cloud3pm",'Attribute20':"Tem
                                 'Attribute21':"Temp3pm",'Attribute22':"RainToday",'Attribute23':"RainTomorrow"},inplace=True)
        predict_df.rename(columns={'Attribute1':"Today",'Attribute2':"Area",'Attribute3':"MinTemp",'Attribute4':"MaxTemp",'Attribute5':"Rainfall",
```

這裏 attributes 不能很好區分，我將它們按照他們的 description 分別進行了 rename。

```
train_df.info() #observe the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17094 entries, 0 to 17093
Data columns (total 23 columns):
Today           17094 non-null object
Area            17094 non-null int64
MinTemp         17019 non-null float64
MaxTemp         17056 non-null float64
Rainfall        16937 non-null float64
Evap            9730 non-null float64
Sunshine        8843 non-null float64
StrWindDir      15964 non-null object
StrWindSpeed    15966 non-null float64
WindDir9am      15874 non-null object
WindDir3pm      16649 non-null object
WindSpeed9am    16931 non-null float64
WindSpeed3pm    16775 non-null float64
Humi9am         16891 non-null float64
Humi3pm         16640 non-null float64
Press9am        15384 non-null float64
Press3pm        15389 non-null float64
Cloud9am        10586 non-null float64
Cloud3pm        10151 non-null float64
Temp9am         16977 non-null float64
Temp3pm         16741 non-null float64
RainToday       16937 non-null object
RainTomorrow    17094 non-null object
dtypes: float64(16), int64(1), object(6)
memory usage: 3.0+ MB
```

觀察 type，加上作業附檔中的 description 可以知道 Categorical data 有 'StrWindDir','WindDir9am','WindDir3pm','RainToday','Area','Cloud9am','Cloud3pm','RainTomorrow'，其餘為 Numerical data。

2.通過 isnull()觀察資料缺失情況

```
train_df.isnull().sum()
```

```
Today              0
Area               0
MinTemp           75
MaxTemp           38
Rainfall         157
Evap            7364
Sunshine        8251
StrWindDir      1130
StrWindSpeed    1128
WindDir9am      1220
WindDir3pm       445
WindSpeed9am     163
WindSpeed3pm     319
Humi9am          203
Humi3pm          454
Press9am        1710
Press3pm        1705
Cloud9am        6508
Cloud3pm        6943
Temp9am          117
Temp3pm          353
RainToday        157
RainTomorrow       0
```

可以看到 Categorical data 和 Numerical data 都有 missing values，針對不同類型的 data 需要采取不同的填補策略。

3.通過查看 RainTomorrow 的 value_counts()查看樣本是否均衡。

```
y_test=pd.Series(train_df.RainTomorrow)
y_test.value_counts()[0]/y_test.value_counts()[1]
```

```
4.3907284768211925
```

```
y_test.value_counts()
```

```
No     13923
Yes     3171
Name: RainTomorrow, dtype: int64
```

可以看到樣本有輕微不均衡，在 train model 時需要注意這個問題。

資料處理

缺失值

1.RainToday

```
train_df["RainToday"]=train_df["RainToday"].fillna('1111111111')
rainnan=train_df[train_df.RainToday=='1111111111'].index.tolist()
train_df_rmrain=train_df.drop(rainnan)
train_df_rmrain.isnull().sum()
```

```
Today             0
Area              0
MinTemp          55
MaxTemp          37
Rainfall          0
Evap           7235
Sunshine       8137
StrWindDir     1106
StrWindSpeed   1104
WindDir9am     1177
WindDir3pm      431
WindSpeed9am    126
WindSpeed3pm    310
Humi9am         171
Humi3pm         442
Press9am       1678
Press3pm       1678
Cloud9am       6380
Cloud3pm       6822
Temp9am          87
Temp3pm         342
RainToday         0
```

根據觀察我們看到 RainToday 有 157 個 missing value，由於這個 attribute 對於明天是否下雨的影響較大，我們不能夠直接用 mode 或 median 對它填補，原本計劃用是否有 Rainfall 來進行判斷 RainToday，但 Rainfall 也同時是缺失的，因此只能 drop 掉。

2. Categorical data

```
train_rain=train_df_rmrain
train_notrain = train_df_rmrain
not_rain=train_df_rmrain[train_df_rmrain.RainTomorrow=='No'].index.tolist()
rain=train_df_rmrain[train_df_rmrain.RainTomorrow=='Yes'].index.tolist()
train_rain=train_rain.drop(not_rain)
train_notrain=train_notrain.drop(rain)
```

```
train_rain['RainTomorrow'].value_counts()
```

```
Yes    3112
Name: RainTomorrow, dtype: int64
```

```
train_notrain['RainTomorrow'].value_counts()
```

```
No    13825
Name: RainTomorrow, dtype: int64
```

```
for i in [train_rain]:
    i.index = range(i.shape[0])
```

```
for i in [train_notrain]:
    i.index = range(i.shape[0])
```

首先將資料集根據明天是否下雨劃分為有下雨的資料集 train_rain 和沒有下雨的資料集 train_norain。

Categorical data

```
cate=train_rain.columns[train_rain.dtypes=="object"].tolist()
cate_known = ['Area','Cloud9am','Cloud3pm']
cate = cate+cate_known
cate.remove('RainTomorrow')
cate
```

```
['StrWindDir',
 'WindDir9am',
 'WindDir3pm',
 'RainToday',
 'Area',
 'Cloud9am',
 'Cloud3pm']
```

```
si = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
si.fit(train_notrain.loc[:, cate])
```

```
SimpleImputer(strategy='most_frequent')
```

```
train_notrain.loc[:, cate] = si.transform(train_notrain.loc[:, cate])
train_notrain.loc[:, cate].isnull().mean()
```

```
StrWindDir     0.0
WindDir9am     0.0
WindDir3pm     0.0
RainToday      0.0
Area           0.0
Cloud9am       0.0
Cloud3pm       0.0
dtype: float64
```

```
si = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
si.fit(train_rain.loc[:, cate])
train_rain.loc[:, cate] = si.transform(train_rain.loc[:, cate])
train_rain.loc[:, cate].isnull().mean()
```

從資料集中選出 Categorical data 對應的 columns, 用 train_rain 和 train_norain 相應 attribute 的 mode 來分別填補底下的 missing values，這裏沒有直接用整個 dataset 的 mode，原因是如果直接填補，有可能會出現較大偏差，如下雨天的雲層數量和不下雨的雲層數量明顯是不同的。

3. Numerical data

```
col = train_notrain.columns.tolist()
for i in cate:
    col.remove(i)
col.remove('RainTomorrow')
col
```

```
['Month',
 'MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evap',
 'Sunshine',
 'StrWindSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humi9am',
 'Humi3pm',
 'Press9am',
 'Press3pm',
 'Temp9am',
 'Temp3pm']
```

從資料集中選出 Numerical data 對應的 columns

```
impmedian = SimpleImputer(missing_values=np.nan,strategy = "median")
# impmedian = SimpleImputer(missing_values=np.nan,strategy = "mean")
impmedian = impmedian.fit(train_notrain.loc[:,col])
train_notrain.loc[:,col] = impmedian.transform(train_notrain.loc[:,col])
```

```
train_notrain.loc[:, col].isnull().mean()
```

```
impmedian = SimpleImputer(missing_values=np.nan, strategy = "median")
# impmedian = SimpleImputer(missing_values=np.nan, strategy = "mean")
impmedian = impmedian.fit(train_rain.loc[:,col])
train_rain.loc[:,col] = impmedian.transform(train_rain.loc[:,col])
train_rain.loc[:, col].isnull().mean()
```

用 train_rain 和 train_norain 相應 attribute 的 median 來分別填補底下的 missing values，這裏也沒有直接用整個 dataset 的 mode，原因是如果直接填補，有可能會出現較大偏差，如下雨天的降雨量會明顯多于不下雨，另外，用 median 能避免 outlier 的影響。

Outliers

```
traindf=pd.concat([train_notrain,train_rain],axis=0, ignore_index=True)
total_X,total_Y = traindf.iloc[:, :-1], traindf.iloc[:, -1]
```

```
total_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16937 entries, 0 to 16936
Data columns (total 22 columns):
Month          16937 non-null float64
Area           16937 non-null int64
MinTemp        16937 non-null float64
MaxTemp        16937 non-null float64
Rainfall       16937 non-null float64
Evap           16937 non-null float64
Sunshine       16937 non-null float64
```

結束后將下雨和不下雨的資料集重新合并為総資料集

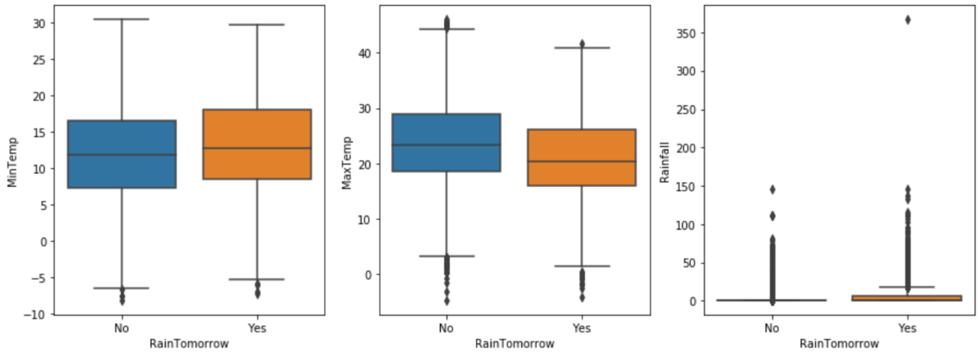
```
total_X.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.99]).T
```

	count	mean	std	min	1%	5%	10%	25%	50%	75%	90%	99%	max
Month	16937.0	6.396824	3.432371	1.0	1.000	1.00	2.0	3.0	6.0	9.0	11.0	12.000	12.0
Area	16937.0	23.668950	14.237378	0.0	0.000	2.00	4.0	11.0	24.0	36.0	43.0	48.000	48.0
MinTemp	16937.0	12.154573	6.424712	-8.2	-2.000	1.70	3.9	7.5	12.0	16.9	20.7	25.700	30.5
MaxTemp	16937.0	23.362207	7.149678	-4.8	9.200	12.70	14.5	18.1	22.8	28.4	33.1	40.100	46.1
Rainfall	16937.0	2.127065	7.882813	0.0	0.000	0.00	0.0	0.0	0.0	0.6	5.6	34.528	367.6
Evap	16937.0	5.221798	3.470944	0.0	0.400	1.20	2.0	3.8	5.0	5.4	8.4	16.000	145.0
Sunshine	16937.0	8.141548	3.022366	0.0	0.000	1.70	4.1	6.3	9.4	9.4	11.1	13.300	14.0
StrWindSpeed	16937.0	39.561729	12.964278	7.0	15.000	20.00	24.0	31.0	37.0	46.0	56.0	80.000	135.0
WindSpeed9am	16937.0	13.879672	8.799077	0.0	0.000	0.00	4.0	7.0	13.0	19.0	26.0	39.000	83.0
WindSpeed3pm	16937.0	18.534274	8.717609	0.0	2.000	6.00	9.0	13.0	17.0	24.0	30.0	43.000	87.0
Humi9am	16937.0	68.414772	18.958538	1.0	17.000	34.00	43.0	57.0	69.0	82.0	94.0	100.000	100.0
Humi3pm	16937.0	50.835331	20.264130	1.0	9.000	17.00	23.0	37.0	51.0	65.0	77.0	97.000	100.0
Press9am	16937.0	1017.764964	6.685243	982.2	1001.400	1006.80	1009.4	1013.7	1018.3	1021.9	1026.3	1034.000	1040.3

通過 describe 觀察到並沒有可以直接一眼就看出的異常值。

```
num_of_rows = 5
num_of_cols = 3
fig, ax = plt.subplots(num_of_rows, num_of_cols, figsize=(15,30))

i=0;j=0;k=0;
while i<num_of_rows:
    while (j<num_of_cols and k<14):
        sns.boxplot(x=total_Y.to_frame().iloc[:,0], y=total_X[col[k]], ax=ax[i, j])
        k+=1;j+=1
    j=0;i+=1
plt.show()
```




```python
_list = ['Rainfall','Evap','StrWindSpeed','WindSpeed9am','WindSpeed3pm','Press9am','Press3pm']

def find_outliers(df,source_df, feature):
    IQR = df[feature].quantile(0.75) - df[feature].quantile(0.25)
    Lower_fence = df[feature].quantile(0.25) - (IQR * 3)
    Upper_fence = df[feature].quantile(0.75) + (IQR * 3)
    print('{feature} outliers are values < {lowerboundary} or > {upperboundary}'\
          .format(feature=feature, lowerboundary=Lower_fence, upperboundary=Upper_fence))
    out_of_middan = (source_df[feature] < Lower_fence).sum()
    out_of_top = (source_df[feature] > Upper_fence).sum()
    print(f'the number of upper outlier {out_of_top}')
    print(f'the number of lower outlier {out_of_middan}')


for feature in _list:
    find_outliers(train_df, total_X, feature)
    print()
```

```
Rainfall outliers are values < -1.7999999999999998 or > 2.4
the number of upper outlier 2639
the number of lower outlier 0

Evap outliers are values < -11.800000000000002 or > 21.800000000000004
the number of upper outlier 66
the number of lower outlier 0
```

```
StrWindSpeed outliers are values < -24.0 or > 102.0
the number of upper outlier 7
the number of lower outlier 0

WindSpeed9am outliers are values < -29.0 or > 55.0
the number of upper outlier 10
the number of lower outlier 0

WindSpeed3pm outliers are values < -20.0 or > 57.0
the number of upper outlier 13
the number of lower outlier 0

Press9am outliers are values < 985.3000000000002 or > 1050.3999999999999
the number of upper outlier 0
the number of lower outlier 2

Press3pm outliers are values < 982.1 or > 1048.6
the number of upper outlier 0
the number of lower outlier 2
```

通過繪箱型圖和數據查看 outlier 分佈及數量

```python
#---------------removing the outlier and filled with the boundary value
numerical = ['Evap','StrWindSpeed','WindSpeed9am','WindSpeed3pm','Press9am','Press3pm']
lsUpper = []
lsLower = []
def removeOutliers(df,total_df,numerical):
    for i in range(len(numerical)):
        q1 = total_df[numerical[i]].quantile(0.25)
        q3 = total_df[numerical[i]].quantile(0.75)
        IQR = q3-q1
        minimum = q1 - (IQR * 3)
        maximum = q3 + (IQR * 3)
        print(minimum)
        print(maximum)
        df.loc[(df[numerical[i]] <= minimum), numerical[i]] = minimum
        df.loc[(df[numerical[i]] >= maximum), numerical[i]] = maximum
removeOutliers(total_X,train_df,numerical)
```

對 outlier 進行處理，大於 upper fence 的用 upper fence 的值來替代；小於 lower fence 的用 lower fence 來替代。

Feature Engineering

1.對日期進行處理

```
train_df.iloc[:,0].value_counts()
```

```
2011-01-31    16
2010-01-14    13
2017-01-24    13
2015-01-17    13
2008-07-07     1
2008-10-26     1
2011-01-30     1
2008-03-09     1
2008-07-24     1
Name: Today, Length: 3179, dtype: int64
```

```
train_rain['Today'] = train_rain['Today'].apply(lambda x: int(x.split('-')[1]))
train_rain = train_rain.rename(columns={"Today":"Month"})
train_notrain['Today'] = train_notrain['Today'].apply(lambda x: int(x.split('-')[1]))
train_notrain = train_notrain.rename(columns={"Today":"Month"})
# total_X.head()
```

通過觀察發現日期存在很多重複值，但同時又有多個值，不能算是 numerical data，但如果把他當作 categorical data，又會在 encoder 之後把 feature 變得特別多，所以在這裏我選用相應月份代替日期。

2. Categorical features

```
list_=["StrWindDir","WindDir9am","WindDir3pm",'RainToday',"Area"]
oe = OrdinalEncoder()
oe = oe.fit(total_X.loc[:,list_])
total_X_1=total_X
total_X_1.loc[:,list_]=oe.transform(total_X.loc[:,list_])
```

```
oh = OneHotEncoder()
oh = oh.fit(total_X.loc[:,list_])
total_X_2=oh.transform(total_X.loc[:,list_]).toarray()
```

```
# predict_df1.drop(columns=["StrWindDir", "WindDir9am", "WindDir3pm", "RainToday", "Area"], axis=1, inplace=True)
predict_df_1=predict_df
predict_df_1.loc[:,list_]=oe.transform(predict_df_1.loc[:,list_])
```

```
predict_df_1=oh.transform(predict_df.loc[:,list_]).toarray()
predict_df_2=pd.DataFrame(predict_df_1)

predict_df1=pd.merge(predict_df,predict_df_2, left_index=True, right_index=True, how='outer')
predict_df1.shape
```

對 categorical data 進行 ordinal encoder 之後再進行 one-hot encoder

3. Numerical features

```
ss=StandardScaler()
ss = ss.fit(Xtrain.loc[:,col])
Xtrain.loc[:,col]=ss.transform(Xtrain.loc[:,col])
Xtest.loc[:,col]=ss.transform(Xtest.loc[:,col])
total_X_1.loc[:,col]=ss.transform(total_X_1.loc[:,col])
predict_df1.loc[:,col]=ss.transform(predict_df1.loc[:,col])
```

對 numerical features 進行 standization

4.預測目標 RainTomorrow

```
memory usage: 132.4+ KB
```

```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(total_X1, total_Y, test_size=0.2, random_state=4)
```

將資料按兩折劃分為訓練集和測試集

```
encoder = LabelEncoder().fit(Ytrain)
Ytrain1 = pd.DataFrame(encoder.transform(Ytrain))
Ytest1 = pd.DataFrame(encoder.transform(Ytest))
total_Y = pd.DataFrame(encoder.transform(total_Y))
```

對預測目標 RainTomorrow 進行 label encoder

Model Training

SVM

```
print("starting")
for kernel in['linear','poly','rbf','sigmoid']:
    clf = SVC(kernel =kernel,
              gamma='auto',
              degree = 1,
              class_weight='balanced').fit(Xtrain,Ytrain1)
    result = clf.predict(Xtest)
    score = clf.score(Xtest,Ytest1)
    recall = recall_score(Ytest1,result)
    auc = roc_auc_score(Ytest1,clf.decision_function(Xtest))
    print("%s 'testing accuracy %f,recall: %f', auc: %f"%(kernel,score,recall,auc))
    #print(datetime.datetime.fromtimestamp(time()-times).strftime('%M:%s:%f'))
print("finished!")
```

由於樣本不均衡，在 balanced 情況下首先測試在哪種 kernel 效果會更好

```
print('start')
svm_ = SVC()
parameters = {'kernel': ['linear','poly'],
              'C':np.linspace(0.01,20,50),
              'gamma':np.linspace(0.1,20,20),
              'class_weight':['balanced']
             }
grid_search1 = GridSearchCV(estimator=svm_, param_grid=parameters, refit='roc_auc', scoring=['accuracy', 'roc_auc'], cv=3, n_jobs=-1)
grid_search1 = grid_search1.fit(Xtrain, Ytrain1)
print('finish!')
```

在 linear 和 poly 時表現更好，使用 Grid Search 進一步調參

```
final_svm_b = clf.predict(predict_df)
pd.DataFrame(final_svm_b, columns=['ans']).to_csv('./result2_bb.csv',index = False)
rs = pd.read_csv('./result2_bb.csv')

col_name=rs.columns.tolist()
col_name.insert(0,'id')
rs=rs.reindex(columns=col_name)

j = rs.shape[0]
for i in range(j):
    rs.iloc[i,0]=i
rs.to_csv('./final_svm_bb.csv',index=False)
```

選擇最佳參數進行 predict 並 submit

Logistic Regression

```
params = {'C':np.linspace(0.001,100,50),
          'max_iter':[1, 10, 100, 500],
          'class_weight':['balanced', None],
          'solver':['liblinear']
         }
lr = LogisticRegression()
grid_search1 = GridSearchCV(estimator=lr, param_grid=parameters, refit='roc_auc', scoring=['accuracy', 'roc_auc'], cv=3, n_jobs=-1)
grid_search1 = grid_search1.fit(Xtrain, Ytrain1)
print('finish!')
```

根據 SVM 可以看出，dataset 比較偏向 linear，所以直接在 liblinear 下進行 Grid Search 再根據結果進行進一步調參，選擇最佳參數進行 predict 並 submit。

Random Forest

```
rf = RandomForestClassifier(random_state=0)
parameters = {'bootstrap': [True, False],
              'min_samples_split':np.arange(2,10,2),
              'criterion':['entropy', 'gini'],
              'n_estimators':np.arange(100,200,10),
              'max_depth':np.arange(10,100,20)
             }
grid_search1 = GridSearchCV(estimator=rf, param_grid=parameters, refit='roc_auc', scoring=['accuracy', 'roc_auc'], cv=3, n_jobs=-1)
grid_search1 = grid_search1.fit(Xtrain, Ytrain1)
```

使用 Grid Search 確定最佳參數範圍

```
rf = RandomForestClassifier(random_state=0)
parameters = {'bootstrap': [False],
              'min_samples_split':[3,4,5],
              'criterion':['entropy'],
              'n_estimators':np.arange(170,190,4),
              'max_depth':np.arange(25,35,1),
              'min_samples_leaf':np.arange(1,10,2)
             }
grid_search1 = GridSearchCV(estimator=rf, param_grid=parameters, refit='roc_auc', scoring=['accuracy', 'roc_auc'], cv=3, n_jobs=-1)
grid_search1 = grid_search1.fit(Xtrain, Ytrain1)
```

```
rf = RandomForestClassifier(bootstrap= False, criterion= 'entropy', min_samples_split=4, n_estimators= 182,
                            random_state=10,max_depth=27,class_weight="balanced")
rf.fit(Xtrain, Ytrain1)
ypred = rf.predict(Xtest)
print("\tAccuracy:{}".format(rf.score(Xtest, Ytest1)))
print("\tRecall:{}".format(recall(Ytest1, ypred)))
print("\tAUC:{}".format(auc(Ytest1, rf.predict_proba(Xtest)[:, 1])))
```

```
	Accuracy:0.9406729634002361
	Recall:0.7619783616692427
	AUC:0.9681148420543952
```

```
final_RF = rf.predict(predict_df)
pd.DataFrame(final_RF, columns=['ans']).to_csv('./result2_RF.csv',index = False)
rs = pd.read_csv('./result2_RF.csv')

col_name=rs.columns.tolist()
col_name.insert(0,'id')
rs=rs.reindex(columns=col_name)

j = rs.shape[0]
for i in range(j):
    rs.iloc[i,0]=i
rs.to_csv('./final_RFb.csv',index=False)
```

再根據結果進行進一步調參，選擇最佳參數進行 predict 並 submit。

XGBoost

```
print('starting...')
params = {"n_estimators": np.arange(
    100, 200, 10), "learning_rate": np.arange(0.05, 0.3, 0.05), "scale_pos_weight":np.linspace(3,6,5)}
clf = XGBClassifier()
gscv = GridSearchCV(clf, param_grid=params, refit='roc_auc', scoring=["accuracy","roc_auc"], cv=3)
gscv.fit(Xtrain, Ytrain1)
print('finished!')
```

```
starting...
finished!
```

```
print("Best Parameters : ",gscv.best_params_)
print("Best AUC-ROC : ", gscv.best_score_)
```

```
Best Parameters :  {'learning_rate': 0.15000000000000002, 'n_estimators': 150, 'scale_pos_weight': 5.25}
Best AUC-ROC :  0.974137137972059
```

使用 Grid Search 確定 learning_rate、n_estimators、scale_pos_weight 最佳範圍

```
print('starting...')
params = {"n_estimators": np.arange(110, 130, 3), "learning_rate": np.linspace(0.2, 0.3, 3),
          "scale_pos_weight":np.linspace(4,6,3),'max_depth':[2,3,4],'min_child_weight':[2,3,4]}
clf = XGBClassifier()
gscv = GridSearchCV(clf, param_grid=params, refit='roc_auc', scoring=["accuracy","roc_auc"], cv=3)
gscv.fit(Xtrain, Ytrain1)
print('finished!')
```

```
starting...
finished!
```

進一步調整 max_depth、min_child_weight

```
print('starting...')
clf = XGBClassifier(n_estimators=125, max_depth=3, learning_rate=0.2,min_child_weight=3,
                    scale_pos_weight=4.7,random_state=100).fit(Xtrain, Ytrain1)
ypred = clf.predict(Xtest)
clf.score(Xtest, Ytest1)
print("\tAccuracy:{}".format(clf.score(Xtest, Ytest1)))
print("\tRecall:{}".format(recall(Ytest1, ypred)))
print("\tAUC:{}".format(auc(Ytest1, clf.predict_proba(Xtest)[:, 1])))
print('finished!')
```

```
preds= clf.predict(predict_df)
pd.DataFrame(preds, columns=['ans']).to_csv('result_XGB2.csv',index = False)
rs = pd.read_csv('./result_XGB2.csv')
col_name=rs.columns.tolist()
col_name.insert(0,'id')
rs=rs.reindex(columns=col_name)

j = rs.shape[0]
for i in range(j):
    rs.iloc[i,0]=i
# rs.to_csv('./XGB_1_ori.csv',index=False)
rs.to_csv('./XGB_1_b2_one.csv',index=False)
```

確定最佳參數，進行 predict 並提交

LightGBM

```
print('start...')
parameters = {
    'max_depth': np.arange(1,8,1),
    'num_leaves': np.arange(10,100,10),
    'learning_rate':np.linspace(0.1,0.7,7),
    'num_iterations' :np.arange(100,200,10)
}

gbm = lgb.LGBMClassifier()
# scale_pos_weight=5.3333,
gsearch = GridSearchCV(gbm, param_grid=parameters, refit='roc_auc', scoring=["accuracy","roc_auc"], cv=3)
gsearch.fit(Xtrain, Ytrain1)
print('best parameters:{0}'.format(gsearch.best_params_))
print('best auc:{0}'.format(gsearch.best_score_))
print(gsearch.cv_results_['mean_test_score'])
print(gsearch.cv_results_['params'])
```

使用 Grid Search 確定 learning_rate、num_leaves、max_depth、num_iterations 最佳範圍

```
def Lgbm(x):
    lgbm = lgb.LGBMClassifier(num_iterations=200, max_depth=9, learning_rate=0.1,num_leaves=10,min_child_samples=16,
                              scale_pos_weight = x,random_state=100).fit(Xtrain, Ytrain1)
    ypred = clf.predict(Xtest)
    lgbm.score(Xtest, Ytest1)
    print("\tAccuracy:{}".format(lgbm.score(Xtest, Ytest1)))
    print("\tRecall:{}".format(recall(Ytest1, ypred)))
    print("\tAUC:{}".format(auc(Ytest1, lgbm.predict_proba(Xtest)[:, 1])))
```

```
print('starting')
x_range = np.arange(2,7,0.5)
for i in x_range:
    print(i)
    Lgbm(i)
print('finish!')
```

逐個確定其他參數

```
preds= lgbm.predict(predict_df)
pd.DataFrame(preds, columns=['ans']).to_csv('result_LGBM.csv',index = False)
rs = pd.read_csv('./result_LGBM.csv')
col_name=rs.columns.tolist()
col_name.insert(0,'id')
rs=rs.reindex(columns=col_name)

j = rs.shape[0]
for i in range(j):
    rs.iloc[i,0]=i

rs.to_csv('./LGBM_one.csv',index=False)
```

確定最佳參數，進行 predict 並提交

總結

1. 可以看出 dataset 是綫性的，使用綫性 model 會有不錯的效果。
2. 雖然綫性 model 會有不錯效果，但是使用集成的 model 如 XGBoost 或 LBM 的效果會比 SVM、Logistic Regression 等效果好。
3. Dataset 的資料是有輕微 imbalance 的，所以在 train model 時需要考慮做 balance。
4. 如果 model 在 training data 上面的效果很好，而在 testing data 上效果並不好，就要看看是不是 over fitting 了。
5. 在 model 無法再有更大提升時，可以考慮換一種 model，如果所有性能較好的 model 都沒有很好的效果，就要在 feature engineering 部分再努力。