◆上 机 操 作

[1]:
```
#coding: utf-8
```

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

#读取Excel数据
filefullpath = r"D:/ML-Data/ALLOY_DATA/Paper One-1.xlsx"
data = pd.read_excel(filefullpath)
df = data.loc[:, ['ALLOYING DEGREE', 'T_MAX', 'T_AVE', 'RH_MIN', 'PRECIPIT', 'SOLAR', 'Cl',
       'SO2',  'Vcorr']].copy()
#iloc它接受的是一个数字，代表着要选择数据的位置；loc接受标签的索引
#读取数据特征
df.describe()
```

[2]:

	ALLOYING DEGREE	T_MAX	T_AVE	RH_MIN	PRECIPIT	SOLAR	Cl	SO2	Vcorr
count	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000	63.000000
mean	2.750667	33.947619	17.961905	45.190476	1648.519048	5351.719048	29.766667	3.830952	0.054214
std	2.458962	1.627670	4.272018	11.274947	390.037351	712.912690	20.256447	1.334814	0.041761
min	0.025300	31.300000	14.200000	23.000000	1166.400000	4193.300000	2.050000	1.800000	0.004300
25%	1.097200	33.000000	14.500000	37.000000	1344.500000	5216.100000	3.300000	2.600000	0.020350
50%	1.995300	33.400000	15.300000	50.500000	1511.500000	5229.000000	32.300000	4.400000	0.040400
75%	3.048700	35.000000	23.800000	56.000000	2046.500000	5740.000000	45.800000	5.100000	0.075800
max	9.198800	37.000000	24.000000	57.000000	2314.000000	6618.500000	55.400000	5.300000	0.176600

[3]:
```
df.shape
```

[3]:
```
(63, 9)
```

数据集拆分

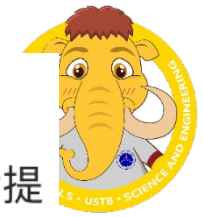
[4]:
```
# 划分训练、测试数据集
from sklearn.model_selection import train_test_split
TrainData, TestData = train_test_split(df, test_size=0.2)
```

[5]:
```
#划分输入和输出数据集
Train_X = TrainData.iloc[:, 0:8].copy()
Train_y = TrainData.iloc[:, 8].copy()
Test_X = TestData.iloc[:, 0:8].copy()
Test_y = TestData.iloc[:, 8].copy()
```

随机森林回归模型

[6]:
```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(random_state=10)
model.fit(Train_X, Train_y)
```

[6]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=10, verbose=0, warm_start=False)
```



核心参数

- **n_estimators**（默认100）：指定决策树数量，值越大精度越高，但超过特定值后提升效果有限。 1 2
- **criterion**（默认'mse'）：评估回归效果的指标，可选'mse'（均方误差）或'mae'（平均绝对误差）。 1 2

树生长控制

- **max_depth**（默认None）：限制树的最大深度，None表示无限分裂直至纯叶节点或达到min_samples_split。 1 2
- **min_samples_split**（默认2）：每个内部节点最少样本数，影响树的分裂条件。 1 2
- **min_samples_leaf**（默认1）：每个叶子节点最少样本数，过小可能导致过拟合。 1 2

特征处理

- **max_features**（默认'auto'）：控制分枝时考虑的特征数量，'auto'为sqrt(n_features)。 1 2
- **max_leaf_nodes**（默认None）：限制最大叶子节点数，None表示无限制。 1 2

训练集

[7]:
```
Train_pred = model.predict(Train_X) #预测结果
```

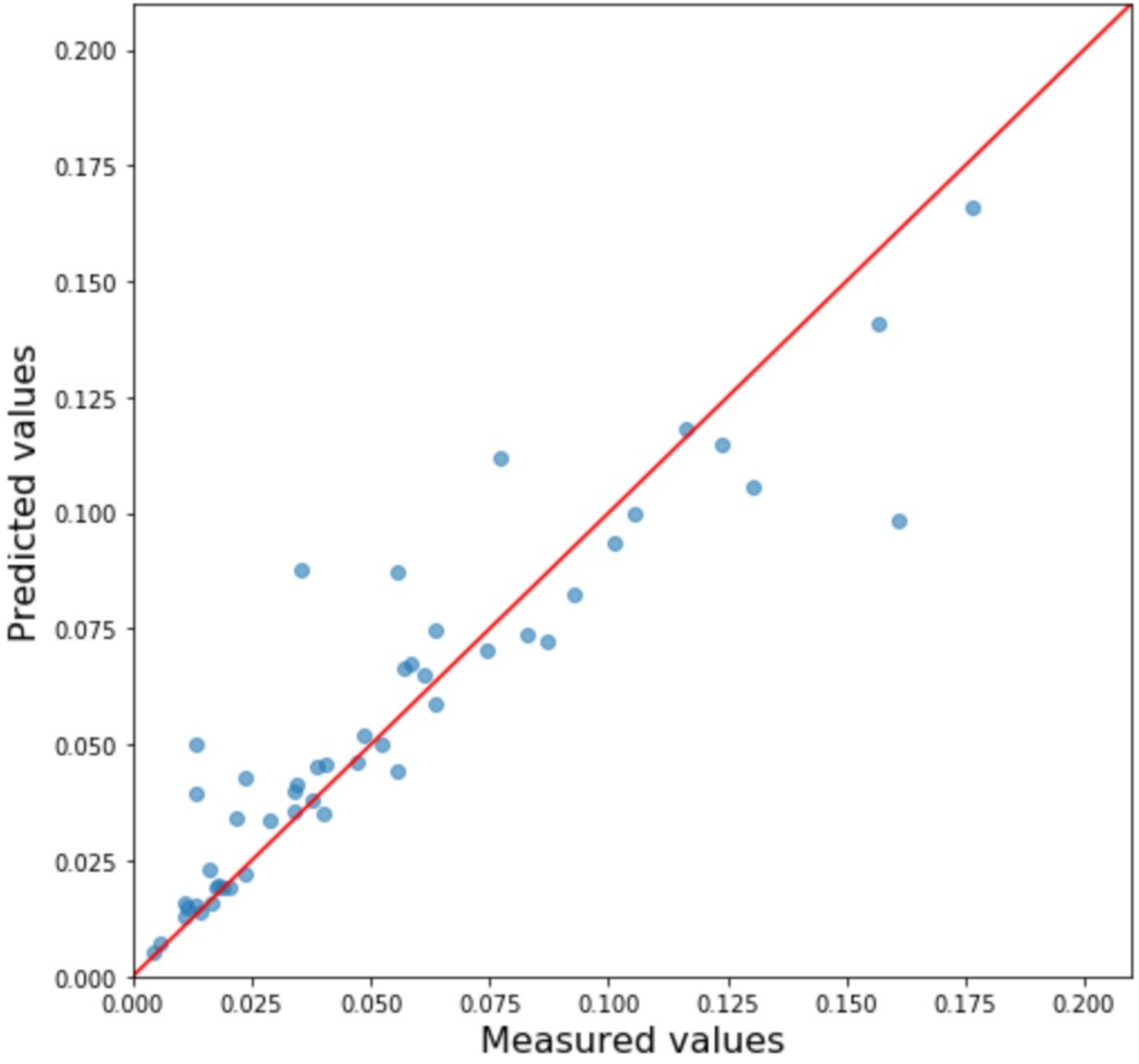
[8]:
```
#均方误差 (Mean squared error)
from sklearn.metrics import mean_squared_error
mean_squared_error(Train_y, Train_pred)
```

[8]:
```
0.00027723212999999999
```

[9]:
```
#求R方值
from sklearn.metrics import r2_score
r2_score(Train_y, Train_pred)
```

[9]:
```
0.8492366775076136
```

[10]:
```
plt.figure(figsize=(8, 8))
plt.scatter(Train_y, Train_pred, alpha=0.6)
plt.plot([0, 0.21], [0, 0.21], "r-")
plt.xlabel('Measured values', fontsize=16)
plt.ylabel('Predicted values', fontsize=16)
plt.xlim(0, 0.21)
plt.ylim(0, 0.21)
plt.show()
```


Predicted values
Measured values
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

测试集

[11]:
```
Test_pred = model.predict(Test_X) #求得测试集样品的腐蚀速率预测值
```

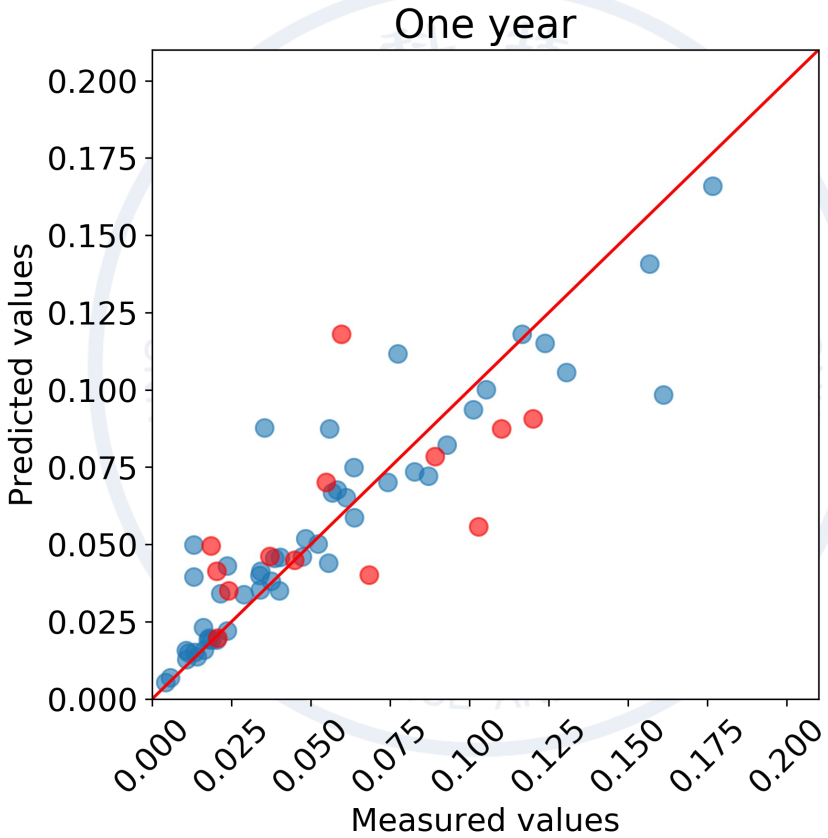
[12]:
```
#均方误差 (Mean squared error)
mean_squared_error(Test_y, Test_pred)
```

[12]:
```
0.0007512243290598295
```

[13]:
```
#求R方值
r2_score(Test_y, Test_pred)
```

[13]:
```
0.38048868797764956
```

[19]:
```
plt.figure(figsize=(6, 6),dpi=300)
plt.scatter(Train_y, Train_pred, s=80,alpha=0.6)
plt.scatter(Test_y, Test_pred,s=80,c='r', alpha=0.6)
plt.plot([0, 0.21], [0, 0.21], "r-")
plt.xlabel('Measured values', fontsize=16)
plt.ylabel('Predicted values', fontsize=16)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.xlim(0, 0.21)
plt.ylim(0, 0.21)
plt.xticks(rotation=45)
plt.title('One year',fontsize=20)
plt.show()
```

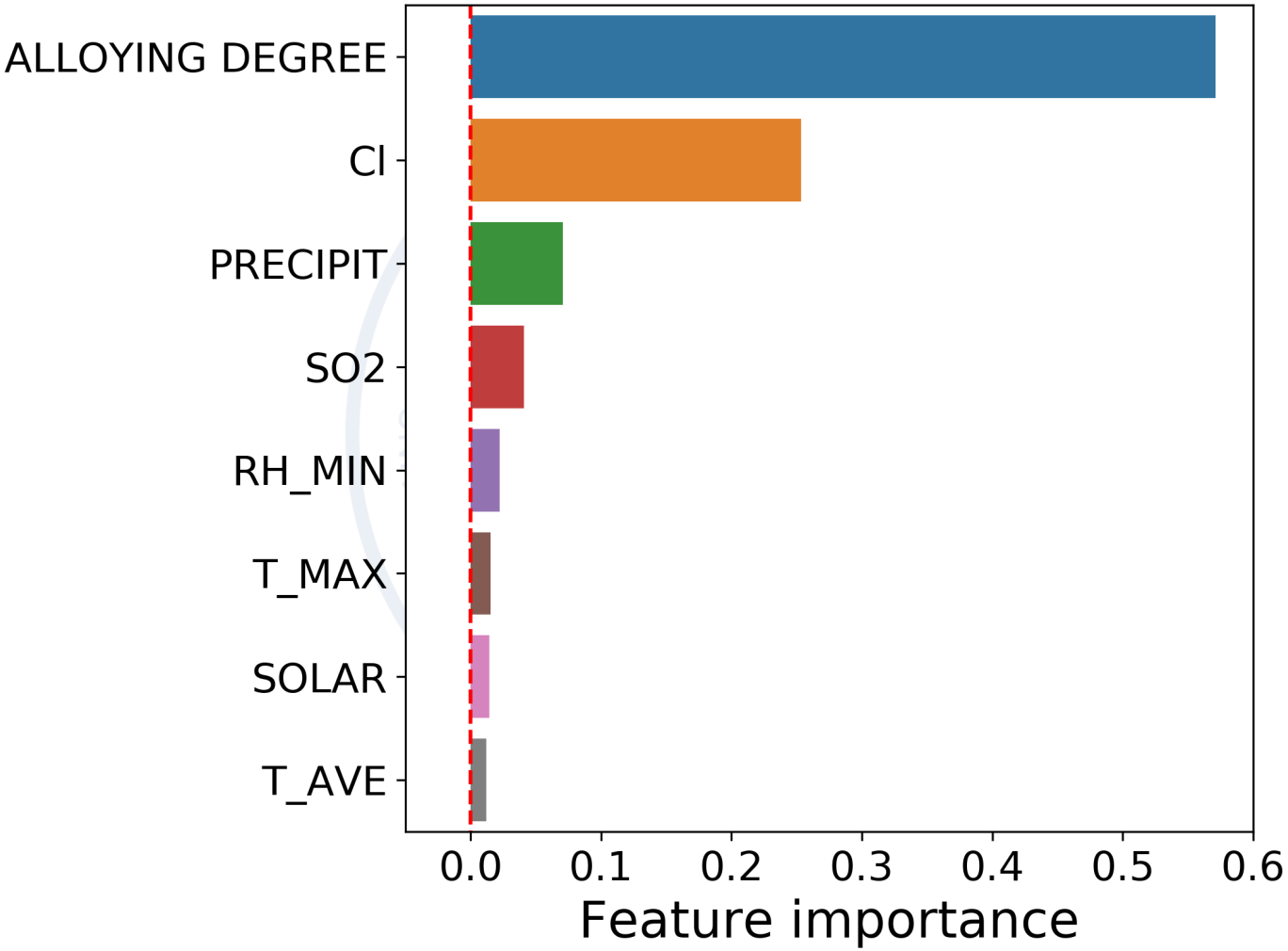

One year
Predicted values
Measured values
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

[15]:
```
X = df.iloc[:, 0:8].copy()
y = df.iloc[:, 8].copy()

def RFFeaImpor_(X_data,y_data):
    model.fit(X_data,y_data)
    result_ = {'var':X_data.columns.values
           ,'feature_importances_':model.feature_importances_}
    feature_importances_ = pd.DataFrame(result_, columns=['var','feature_importances_'])
    feature_importances_ = feature_importances_.sort_values('feature_importances_',ascending=False)
    return feature_importances_
feature_importances_ = RFFeaImpor_(X,y)
print (feature_importances_)
```

	var	feature_importances_
0	ALLOYING DEGREE	0.571172
6	Cl	0.253239
4	PRECIPIT	0.070801
7	SO2	0.041000
3	RH_MIN	0.022345
1	T_MAX	0.015245
5	SOLAR	0.014299
2	T_AVE	0.011899

[16]:
```
plt.figure(figsize=(6, 6),dpi=300)
sns.barplot(feature_importances_['feature_importances_'], feature_importances_['var'])
plt.plot([0, 0],[-2, 20],"r--")
plt.xlabel('Feature importance',fontsize=20)
plt.ylabel(' ',fontsize=20)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.xlim((-0.05,0.6))
plt.show()
```


ALLOYING DEGREE
Cl
PRECIPIT
SO2
RH_MIN
T_MAX
SOLAR
T_AVE
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Feature importance

GBDT模型

[7]:
```
from sklearn.ensemble import GradientBoostingRegressor
model = GradientBoostingRegressor(loss='ls')
model.fit(Train_X, Train_y)
```

[7]:
```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
```