目录

1 数据预处理 1
2 模型选择 3
2.1 SVR 支持向量回归模型 3
2.2 Xgboost 模型 5
2.3 线性回归模型 8
2.4 决策树回归 8
2.5 梯度提升模型 8
2.6 Lasso 回归模型 10
2.7 K 近邻回归模型 10
2.8 随机森林模型 11
2.9 集成学习模型 11
2.10 弹性网络回归模型 12
2.11 极端森林回归模型 12
2.12 小结 13
3 模型评估 13
4 模型的问题与改进方向 15
5 模型问题的部分解决 15
参考文献 17

摘　要

本次人工智能大作业“房价预测”，实质上是经典机器学习入门项目“波士顿房价预测”的一个子问题，区别主要在于数据的读入方式和参数数量。在模型构建的全流程中，先对数据进行预处理，在探索性数据分析（EDA）后进行了数据标准化；然后，对各种回归模型（SVR、Xgboost、线性回归、决策树、梯度提升、Lasso、K 近邻、随机森林、集成学习、弹性网络、极端森林等机器学习方法）分别进行必要的调参，以 RMSE 为衡量标准选择最优模型及最优参数，建立了极端森林回归模型。接着，进行了 1000 次重复测试，据测试结果对模型进行了鲁棒性和准确性评估。最后分析了模型存在的问题与改进方向，并通过结合 SVR 模型建立融合模型解决了部分问题，减小了预测误差。最终，在 1000 次重复实验中，最佳预测结果的 MAE 值为 40607，MAPE 值为 10.11%。大作业所用全部源码可见于压缩包内另一文件 AI_hw2.ipynb。

关键词：数据标准化；支持向量回归；极端森林回归；模型评估；融合模型

1 数据预处理

首先从提供的表格 housing.csv 中读取数据集，用变量 X 储存自变量 RM、LSTAT 和 PTRATIO，变量 Y 存储因变量 MEDV(即房价）。

然后将数据集划分为训练集和测试集。对于数据集的划分过程，本文调用了 python 自带的 train_test_split 函数。通过使用该函数，可将 X 划分为训练集 X_train 和测试集 X_test，将 Y 划分为训练集 Y_train 和测试集 Y_test。其中，通过设定 test_size=0.15,将训练集与测试集的大小比例设为 8.5:1.5；通过设定 random_state=233，固定随机种子，以减少运行结果的变化。

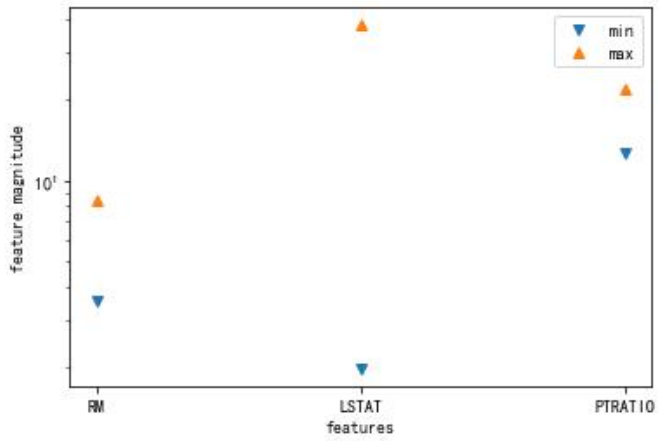
```
import pandas as pd
from sklearn.model_selection import train_test_split
data_train = pd.read_csv("housing.csv")
data_test = pd.read_csv("housing.csv")

#删除不相关属性
X = data_train.drop('MEDV', axis=1)
Y = data_train.MEDV

#准备训练集和测试集
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.15,random_state=233)
```

然后，为了获得对数据的初步了解，进行探索性数据分析（EDA），输出各数据特征的最小值和最大值。

```
#特征数值中最小值和最大值数据可视化（防止不同数据特征量级差异较大）
import matplotlib.pyplot as plt
plt.plot(X.min(axis=0),'v',label='min')#axis=0表示列
plt.plot(X.max(axis=0),'^',label='max')
plt.yscale('log')
plt.legend(loc='best')
plt.xlabel('features')
plt.ylabel('feature magnitude')
plt.show()
print(X.RM.min(axis=0),X.RM.max(axis=0))
print(X.LSTAT.min(axis=0),X.LSTAT.max(axis=0))
print(X.PTRATIO.min(axis=0),X.PTRATIO.max(axis=0))
```




```
3.561 8.398
1.98 37.97
12.6 22.0
```

从代码运行结果中可以看出，参数 RM 的值在[3.561,8.398]内，参数 LSTAT 的值在[1.98,37.97]内，参数 PTRATIO 的值在[12.6,22.0]，不同数据特征间具有显著的量级差异。

为了减小不同数据间的量级差异，需将数据标准化。具体来讲，是对每一个特征维度进行去均值和方差归一化，使得经过处理的数据符合标准正态分布，即均值为 0，标准差为 1，其转化函数为：

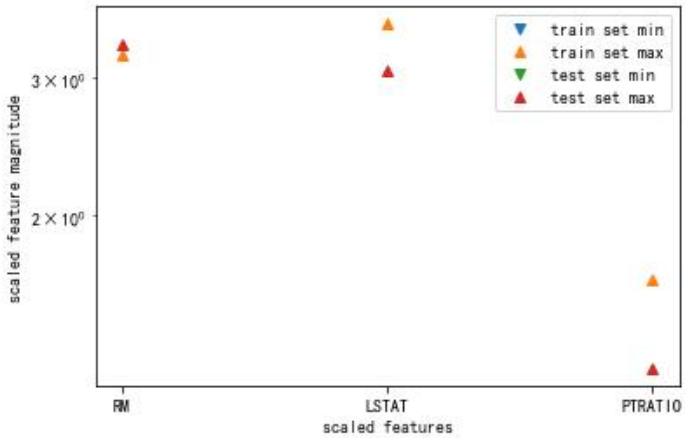
$$x^* = \frac{x - \mu}{\sigma}$$

其中 μ 为所有样本数据的均值，σ 为所有样本数据的标准差。本文通过调用 StandardScaler()函数实现该过程。下面的代码中，仅对 X 数据进行了标准化。对 Y 数据的标准化更靠后一些，因为开始时并没有意识到 Y 数据标准化的重要性。

```
#数据预处理(减小数据量级差异)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)

cols = X_train.columns
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=cols)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=cols)

plt.plot(X_train_scaled.min(axis=0),'v',label='train set min')
plt.plot(X_train_scaled.max(axis=0),'^',label='train set max')
plt.plot(X_test_scaled.min(axis=0),'v',label='test set min')
plt.plot(X_test_scaled.max(axis=0),'^',label='test set max')
plt.yscale('log')
plt.legend(loc='best')
plt.xlabel('scaled features')
plt.ylabel('scaled feature magnitude')
plt.show()
```



随后输出各个特征维度的最大最小值散点图，可以发现不同数据间的量级差

异大大缩小，具有可比性。同时观察可得，数据集中不存在缺失值和异常值的情况，数据预处理达到预期目的。

2 模型选择

由于目标变量 MEDV 是连续变量，故需要建立一个回归模型来进行预测。由于机器学习中的回归模型种类众多，且各有擅长的领域。故笔者通过调用 python 自带函数，训练多种不同模型，并在必要时调整超参数，得出各模型可求得的最优预测结果。然后将各个最优预测结果进行对比，选出预测结果平均误差最小的模型进行进一步的研究。

本文选用了两个指标来衡量平均误差和回归模型的效果。

（1）R 方：该指标也称为决定系数。给定一系列真值y_i和对应的预测值$\hat{y}_i$，则 R 方的定义为

$$R^2 = 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i (y_i - \bar{y})^2}$$

R 方衡量的是预测值对于真实值的拟合好坏程度。R 的取值范围为负无穷到 1，R 值越接近 1，表明平均误差越小，拟合效果越好。不同取值代表的含义如下：

- **R 方=1：**最理想情况，所有的预测值等于真值。
- **R 方=0：**一种可能情况是"简单预测所有 y 值等于 y 平均值"，但也有其他可能。
- **R 方<0：**模型预测能力差，比"简单预测所有 y 值等于 y 平均值"的效果还差。这表示可能用了错误模型，或者模型假设不合理。

（2）RMSE：被称为均方根误差，可衡量观测值与真实值之间的偏差，常用来作为机器学习模型预测结果衡量的标准。计算公式为：

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x_i) - y_i)^2}$$

RMSE 值越小，表明预测值与真实值越接近，模型预测效果越好。

在模型建立阶段，本文使用 RMSE 作为衡量平均误差的唯一标准。在文章后几章中，还将使用 MAE（平均绝对误差）和 MAPE（平均绝对百分比误差）来评估模型预测效果。MAE 和 MAPE 与 RMSE 类似，都是值越小，效果越好，故不再赘述。

2.1 SVR 支持向量回归模型

通过调用 SVR()函数来实现 SVR 模型，并调用 score（）函数求得 R 方值，调用 mean_squared_error（）函数求得 RMSE 值并输出。

```
#使用预处理后的数据训练SVR模型
from sklearn.svm import SVR
from sklearn.svm import SVC
import numpy as np
from sklearn.metrics import mean_squared_error
for kernel in ['linear','rbf']:
    svr=SVR(kernel=kernel)
    svr.fit(X_train_scaled,Y_train)
    svr_pre=svr.predict(X_test)
    print(kernel,'核函数的模型训练集得分：{:.3f}'.format(svr.score(X_train_scaled,Y_train)))
    print(kernel,'核函数的模型测试集得分：{:.3f}'.format(svr.score(X_test_scaled,Y_test)))
    print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,svr_pre))))
```

```
linear 核函数的模型训练集得分：-0.001
linear 核函数的模型测试集得分：-0.001
RMSE:163143.635
rbf 核函数的模型训练集得分：-0.006
rbf 核函数的模型测试集得分：-0.006
RMSE:164336.856
```

可以发现，无论使用 linear 核还是 rbf 核，SVR 模型的 R 方值都接近 0，远低于 1，预测效果很不理想。于是考虑调整参数来改进预测效果。其中，kernel 参数的选择范围为 linear 核或 rbf 核，C 值的为 1,2 或 4，gamma 值的为 0.125,0.25,0.5,1,2,4。调整参数可通过调用 GridSearchCV（）函数实现，后面模型中的类似过程不再赘述。

```
#调整SVR模型的C参数和gramma参数
from sklearn import model_selection
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np
param_grid={'kernel':('linear', 'rbf'), 'C':[1, 2, 4], 'gamma':[0.125, 0.25, 0.5 ,1, 2, 4]}
svr=model_selection.GridSearchCV(SVR(),param_grid)
svr.fit(X_train_scaled,Y_train)
svr_pre=svr.predict(X_test)
print('参数的最佳取值：{0}'.format(svr.best_params_))
print("调整参数后的模型在训练集得分：{:.3f}".format(svr.score(X_train_scaled,Y_train)))
print("调整参数后的模型在测试集得分：{:.3f}".format(svr.score(X_test_scaled,Y_test)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,svr_pre))))
```

```
参数的最佳取值：{'C': 4, 'gamma': 0.125, 'kernel': 'linear'}
调整参数后的模型在训练集得分：0.014
调整参数后的模型在测试集得分：0.015
RMSE:160793.453
```

发现 SVR 模型测试结果的 R 方值仅有 0.015，而 RMSE 值高达 16 万，预测效果仍然不佳，这说明出现了严重的过拟合问题。经过和同学讨论，得出的解决方法是先用标准化后的 Y 值数据训练模型，待得出预测结果后再将 Y 值复原。后面所有模型也都应用了标准化后的 Y 值。

```
#Y值标准化
Y_mean=np.mean(Y,axis=0)
Y_std=np.std(Y,axis=0)
Y_train_scaled=(Y_train-Y_mean)/Y_std
Y_test_scaled=(Y_test-Y_mean)/Y_std
def get_Y_recover(Y):
    return Y*Y_std+Y_mean
```

```
#再次尝试训练SVR模型
from sklearn import model_selection
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np
param_grid={'kernel':('linear', 'rbf'), 'C':range(1,7),
            'gamma':[0.01,0.05,0.1,0.125, 0.25, 0.5 ,1, 2, 4]}
svr=model_selection.GridSearchCV(SVR(),param_grid)
svr.fit(X_train_scaled,Y_train_scaled)
svr_pre=get_originY(svr.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(svr.best_params_))
print("调整参数后的模型在训练集得分：{:.3f}".format(svr.score(X_train_scaled,Y_train_scaled)))
print("调整参数后的模型在测试集得分：{:.3f}".format(svr.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,svr_pre))))
```

```
参数的最佳取值：{'C': 2, 'gamma': 0.25, 'kernel': 'rbf'}
调整参数后的模型在训练集得分：0.859
调整参数后的模型在测试集得分：0.876
RMSE:57587.635
```

再次改进后，SVR 模型的预测结果得到了显著改善，其中 R 方值为 0.876，RMSE 值为 5 万有余，预测效果理想。

2.2 Xgboost 模型

首先设定各参数初始值，如下图中 other_params 所示（不设初始值会过拟合）。然后调整该模型的 n_estimators 参数，调整范围如下图中 cv_params 所示。

```
#xgboost模型
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
#调n_estimators
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
cv_params = {'n_estimators': [25,50,75,100,200,300,400, 500]}
other_params = {'learning_rate': 0.1, 'n_estimators': 500, 'max_depth': 5,
                'min_child_weight': 1,'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.8,
                'gamma': 0, 'reg_alpha': 0,'reg_lambda': 1}
model=xgb.XGBRegressor(**other_params)
optimized_GBM=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,
                           verbose=1,n_jobs=4)
optimized_GBM.fit(X_train_scaled, Y_train_scaled)
gbm_pre=get_originY(optimized_GBM.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
print('模型训练集得分：{:.3f}'.format(optimized_GBM.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(optimized_GBM.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gbm_pre))))
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
参数的最佳取值：{'n_estimators': 50}
模型训练集得分：0.945
模型测试集得分：0.857
RMSE:61852.891
```

得出 n_estimators 参数的最优值为 50。然后继续调整其它参数。

```
#调mid_child_weight和max_depth
cv_params = {'max_depth': [2, 3, 4, 5, 6, 7, 8], 'min_child_weight': [1, 2, 3, 4, 5, 6]}
other_params = {'learning_rate': 0.1, 'n_estimators': 50, 'max_depth': 5, 'min_child_weigh
                'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0, 'reg_alph
                'reg_lambda': 1}
model=xgb.XGBRegressor(**other_params)
optimized_GBM=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=
optimized_GBM.fit(X_train_scaled, Y_train_scaled)
gbm_pre=get_originY(optimized_GBM.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
print('模型训练集得分：{:.3f}'.format(optimized_GBM.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(optimized_GBM.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gbm_pre))))
```

```
Fitting 5 folds for each of 42 candidates, totalling 210 fits
参数的最佳取值：{'max_depth': 3, 'min_child_weight': 1}
模型训练集得分：0.896
模型测试集得分：0.866
RMSE:59953.372
```

```
#调gamma
cv_params = {'gamma': [0,0.1, 0.2, 0.3, 0.4, 0.5, 0.6]}
other_params = {'learning_rate': 0.1, 'n_estimators': 50, 'max_depth': 3, 'min_child_weigh
                'seed': 0, 'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0, 'reg_alp
                'reg_lambda': 1}
model=xgb.XGBRegressor(**other_params)
optimized_GBM=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=
optimized_GBM.fit(X_train_scaled, Y_train_scaled)
gbm_pre=get_originY(optimized_GBM.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
print('模型训练集得分：{:.3f}'.format(optimized_GBM.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(optimized_GBM.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gbm_pre))))
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
参数的最佳取值：{'gamma': 0.3}
模型训练集得分：0.893
模型测试集得分：0.865
RMSE:60103.107
```

```
#调subsample和colsample_bytree
cv_params = {'subsample': [0.6, 0.7, 0.8, 0.9], 'colsample_bytree': [0.6, 0.7, 0.8, 0.9]}
other_params = {'learning_rate': 0.1, 'n_estimators': 50, 'max_depth': 3, 'min_child_weigh
                'seed': 0, 'subsample': 0.8, 'colsample_bytree': 0.8, 'gamma': 0.3, 'reg_a
                'reg_lambda': 1}
model=xgb.XGBRegressor(**other_params)
optimized_GBM=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=
optimized_GBM.fit(X_train_scaled, Y_train_scaled)
gbm_pre=get_originY(optimized_GBM.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
print('模型训练集得分：{:.3f}'.format(optimized_GBM.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(optimized_GBM.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gbm_pre))))
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
参数的最佳取值：{'colsample_bytree': 0.7, 'subsample': 0.8}
模型训练集得分：0.893
模型测试集得分：0.865
RMSE:60103.107
```

```
#调reg_alpha和reg_lambda
cv_params = {'reg_alpha': [0,0.05, 0.1, 1, 2, 3], 'reg_lambda': [0,0.05, 0.1, 1, 2, 3]}
other_params = {'learning_rate': 0.1, 'n_estimators': 50, 'max_depth': 3, 'min_child_weigh
                'seed': 0,'subsample': 0.8, 'colsample_bytree': 0.7, 'gamma': 0.3, 'reg_al
                'reg_lambda': 1}
model=xgb.XGBRegressor(**other_params)
optimized_GBM=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=
optimized_GBM.fit(X_train_scaled, Y_train_scaled)
gbm_pre=get_originY(optimized_GBM.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
print('模型训练集得分：{:.3f}'.format(optimized_GBM.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(optimized_GBM.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gbm_pre))))
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
参数的最佳取值：{'reg_alpha': 0.1, 'reg_lambda': 1}
模型训练集得分：0.888
模型测试集得分：0.864
RMSE:60334.724
```

```
#调learning_rate
cv_params = {'learning_rate': [0.01, 0.05, 0.07, 0.1, 0.2]}
other_params = {'learning_rate': 0.1, 'n_estimators': 50, 'max_depth': 3, 'min_child_weigh
                'seed': 0, 'subsample': 0.8, 'colsample_bytree': 0.7, 'gamma': 0.3, 'reg_a
                'reg_lambda':1}
model=xgb.XGBRegressor(**other_params)
optimized_GBM=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=
optimized_GBM.fit(X_train_scaled, Y_train_scaled)
gbm_pre=get_originY(optimized_GBM.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(optimized_GBM.best_params_))
print('模型训练集得分：{:.3f}'.format(optimized_GBM.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(optimized_GBM.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gbm_pre))))
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
参数的最佳取值：{'learning_rate': 0.1}
模型训练集得分：0.888
模型测试集得分：0.864
RMSE:60334.724
```

最后用效果最好的参数组合来设定 Xgboost 模型并进行训练和测试，结果如下。

```
#应用挑选出的最佳参数进行训练
xg_reg = xgb.XGBRegressor(learning_rate=0.1, n_estimators=50, max_depth=3, min_child_weigh
#xg_reg = xgb.XGBRegressor()
xg_reg.fit(X_train_scaled, Y_train_scaled)
xg_pre=get_originY(xg_reg.predict(X_test_scaled))
print('模型训练集得分：{:.3f}'.format(xg_reg.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(xg_reg.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,xg_pre))))
```

```
模型训练集得分：0.896
模型测试集得分：0.866
RMSE:59953.372
```

可以发现，调参后的 Xgboost 模型测试结果的 R 方值为 0.866，预测结果的 RMSE 值接近 6 万，预测效果较为理想。

2.3 线性回归模型

```
#开一把线性回归模型
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train_scaled, Y_train_scaled)
lr_pre=get_originY(lr.predict(X_test_scaled))
print('模型训练集得分：{:.3f}'.format(lr.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(lr.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,lr_pre))))
```

```
模型训练集得分：0.711
模型测试集得分：0.753
RMSE:81379.523
```

可以发现，调参后的线性模型测试结果的 R 方值为 0.753，预测结果的 RMSE 值为 8 万左右，预测效果一般。

2.4 决策树回归

```
#决策树回归
from sklearn.tree import DecisionTreeRegressor
cv_params = {'max_features': ['sqrt', 'log2', None],'max_depth': range(2,100)}
model = DecisionTreeRegressor()
tr=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=4
tr.fit(X_train_scaled, Y_train_scaled)
tr_pre=get_originY(tr.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(tr.best_params_))
print('模型训练集得分：{:.3f}'.format(tr.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(tr.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,tr_pre))))
```

```
Fitting 5 folds for each of 294 candidates, totalling 1470 fits
参数的最佳取值：{'max_depth': 4, 'max_features': None}
模型训练集得分：0.868
模型测试集得分：0.708
RMSE:88570.145
```

观察可得，决策树模型测试结果的 R 方值为 0.708，RMSE 值 8 万有余，预测效果一般。

2.5 梯度提升模型

梯度提升模型与 Xgboost 模型类似，也通过逐个调参来实现最优化。

```
#梯度提升
#调n_estimators
from sklearn import ensemble
cv_params = {'n_estimators':range(20,81,10)}
other_params = {}
model=ensemble.GradientBoostingRegressor(**other_params)
gb=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=4
gb.fit(X_train_scaled, Y_train_scaled)
gb_pre=get_originY(gb.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(gb.best_params_))
print('模型训练集得分：{:.3f}'.format(gb.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(gb.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gb_pre))))
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
参数的最佳取值：{'n_estimators': 50}
模型训练集得分：0.908
模型测试集得分：0.865
RMSE:60234.609
```

```
#调max_depth和min_samples_split
from sklearn import ensemble
cv_params = {'max_depth':range(3,14,2), 'min_samples_split':range(2,100)}
other_params = {'n_estimators':50}
model=ensemble.GradientBoostingRegressor(**other_params)
gb=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=4
gb.fit(X_train_scaled, Y_train_scaled)
gb_pre=get_originY(gb.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(gb.best_params_))
print('模型训练集得分：{:.3f}'.format(gb.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(gb.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gb_pre))))
```

```
Fitting 5 folds for each of 588 candidates, totalling 2940 fits
参数的最佳取值：{'max_depth': 3, 'min_samples_split': 40}
模型训练集得分：0.901
模型测试集得分：0.866
RMSE:60001.837
```

```
#调min_samples_split和min_samples_leaf
from sklearn import ensemble
cv_params = {'min_samples_split':range(2,100), 'min_samples_leaf':range(1,10)}
other_params = {'n_estimators':50,'max_depth':3,'min_samples_split':40}
model=ensemble.GradientBoostingRegressor(**other_params)
gb=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=4
gb.fit(X_train_scaled, Y_train_scaled)
gb_pre=get_originY(gb.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(gb.best_params_))
print('模型训练集得分：{:.3f}'.format(gb.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(gb.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gb_pre))))
```

```
Fitting 5 folds for each of 882 candidates, totalling 4410 fits
参数的最佳取值：{'min_samples_leaf': 6, 'min_samples_split': 30}
模型训练集得分：0.897
模型测试集得分：0.867
RMSE:59628.110
```

```
#用最优参数看看成效
from sklearn import ensemble
gb=ensemble.GradientBoostingRegressor(n_estimators=60,max_depth=3,min_samples_split=30,
                                      min_samples_leaf=6)
gb.fit(X_train_scaled, Y_train_scaled)
gb_pre=get_originY(gb.predict(X_test_scaled))
print('模型训练集得分：{:.3f}'.format(gb.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(gb.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,gb_pre))))
```

```
模型训练集得分：0.901
模型测试集得分：0.868
RMSE:59438.213
```

综上，梯度提升模型用测试集得到的测试结果 R 方值为 0.868，RMSE 值接近 6 万，预测效果较为理想。

2.6 Lasso 回归模型

```
#Lasso回归
from sklearn.linear_model import Lasso
lo = Lasso()
lo.fit(X_train_scaled, Y_train_scaled)
lo_pre=get_originY(lo.predict(X_test_scaled))
print('模型训练集得分：{:.3f}'.format(lo.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(lo.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,lo_pre))))
```

```
模型训练集得分：0.000
模型测试集得分：-0.000
RMSE:163806.466
```

发现预测效果极差，于是尝试用原始 Y 数据训练模型并预测。

```
#Lasso回归
from sklearn.linear_model import Lasso
lo = Lasso()
lo.fit(X_train, Y_train)
lo_pre=lo.predict(X_test)
print('模型训练集得分：{:.3f}'.format(lo.score(X_train,Y_train)))
print('模型测试集得分：{:.3f}'.format(lo.score(X_test,Y_test)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,lo_pre))))
```

```
模型训练集得分：0.711
模型测试集得分：0.753
RMSE:81379.359
```

可以看出，Lasso 回归模型用测试集得到的测试结果 R 方值为 0.753，RMSE 值 8 万有余，预测效果一般。

2.7 K 近邻回归模型

```
#K近邻回归
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor()
cv_params = {'weights': ['uniform', 'distance'],'n_neighbors': range(2,100)}
kn=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=4
kn.fit(X_train_scaled, Y_train_scaled)
kn_pre=get_originY(kn.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(kn.best_params_))
print('模型训练集得分：{:.3f}'.format(kn.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(kn.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,kn_pre))))
```

```
Fitting 5 folds for each of 196 candidates, totalling 980 fits
参数的最佳取值：{'n_neighbors': 9, 'weights': 'distance'}
模型训练集得分：1.000
模型测试集得分：0.858
RMSE:61630.630
```

可以看出，K 近邻回归模型用测试集得到的测试结果 R 方值为 0.858，RMSE 值接近 6 万，预测效果较为理想。

2.8 随机森林模型

```
#随机森林
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
cv_params = {'min_samples_split':range(5,20),'n_estimators':range(1,20)}
rf=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=4
rf.fit(X_train_scaled, Y_train_scaled)
rf_pre=get_originY(rf.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(rf.best_params_))
print('模型训练集得分：{:.3f}'.format(rf.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(rf.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,rf_pre))))
```

```
Fitting 5 folds for each of 285 candidates, totalling 1425 fits
参数的最佳取值：{'min_samples_split': 9, 'n_estimators': 11}
模型训练集得分：0.931
模型测试集得分：0.860
RMSE:61376.666
```

可以看出，随机森林模型用测试集得到的测试结果 R 方值为 0.860，RMSE 值 6 万有余，预测效果较为理想。

2.9 集成学习模型

```
#集成学习
from sklearn.ensemble import AdaBoostRegressor
model=AdaBoostRegressor()
cv_params = {'n_estimators': range(20,30), 'learning_rate' : [0.01,0.05,0.1,0.15,0.2,0.25]
abr=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=
abr.fit(X_train_scaled, Y_train_scaled)
abr_pre=get_originY(abr.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(abr.best_params_))
print('模型训练集得分：{:.3f}'.format(abr.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(abr.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,abr_pre))))
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
参数的最佳取值：{'learning_rate': 0.2, 'loss': 'square', 'n_estimators': 23}
模型训练集得分：0.854
模型测试集得分：0.865
RMSE:60103.529
```

可以看出，集成学习模型用测试集得到的测试结果 R 方值为 0.865，RMSE 值接近 6 万，预测效果较为理想。

2.10 弹性网络回归模型

```
#弹性网络回归
from sklearn.linear_model import ElasticNet
en = ElasticNet()
en.fit(X_train, Y_train)
en_pre=en.predict(X_test)
print('模型训练集得分：{:.3f}'.format(en.score(X_train,Y_train)))
print('模型测试集得分：{:.3f}'.format(en.score(X_test,Y_test)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,en_pre))))
```

```
模型训练集得分：0.683
模型测试集得分：0.706
RMSE:88783.038
```

可以看出，弹性网络回归模型用测试集得到的测试结果 R 方值为 0.706，RMSE 值 8 万有余，预测效果不够理想。

2.11 极端森林回归模型

```
#极端森林回归
from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor()
cv_params = {'min_samples_split':range(5,15),'n_estimators':range(60,80)}
etc=GridSearchCV(estimator=model,param_grid=cv_params,scoring='r2',cv=5,verbose=1,n_jobs=
etc.fit(X_train_scaled, Y_train_scaled)
etc_pre=get_originY(etc.predict(X_test_scaled))
print('参数的最佳取值：{0}'.format(etc.best_params_))
print('模型训练集得分：{:.3f}'.format(etc.score(X_train_scaled,Y_train_scaled)))
print('模型测试集得分：{:.3f}'.format(etc.score(X_test_scaled,Y_test_scaled)))
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,etc_pre))))
```

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
参数的最佳取值：{'min_samples_split': 10, 'n_estimators': 68}
模型训练集得分：0.924
模型测试集得分：0.879
RMSE:57092.690
```

可以看出，极端森林回归模型用测试集得到的测试结果 R 方值为 0.879，RMSE 值 5 万有余，预测效果理想。

2.12 小结

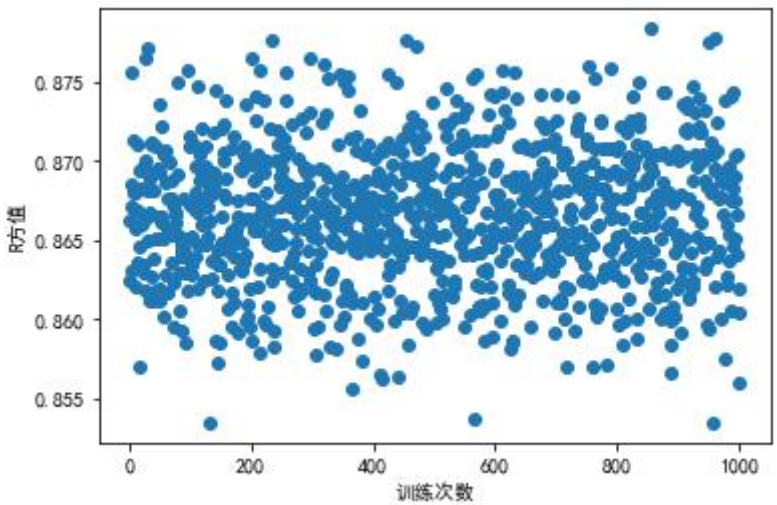
上述 11 个模型的预测效果总结如下表所示。

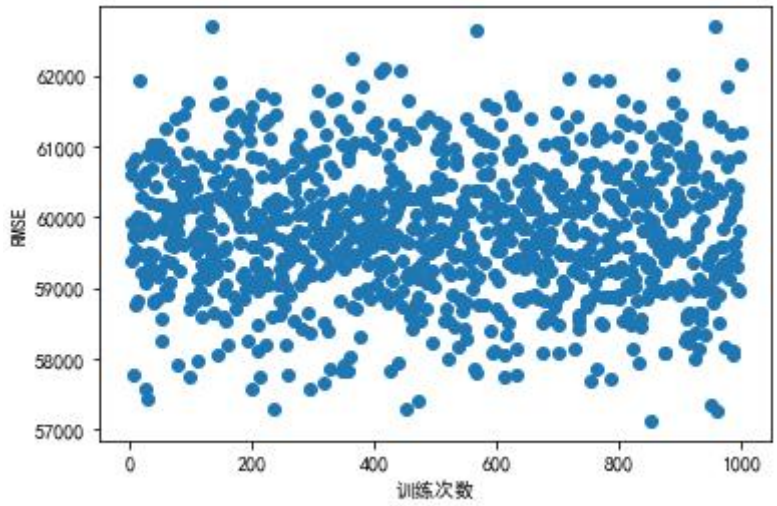
选用模型	训练集 R 方	测试集 R 方	测试集 RMSE
SVR 模型	0.859	0.876	57587
Xgboost 模型	0.896	0.866	59953
线性回归模型	0.711	0.753	81379
决策树回归模型	0.868	0.708	88570
梯度提升模型	0.901	0.868	59438
Lasso 回归模型	0.711	0.753	81379
K 近邻回归模型	**1.000**	0.858	61630
随机森林模型	0.931	0.860	61376
集成学习模型	0.854	0.865	60103
弹性网络回归模型	0.683	0.706	88783
极端森林回归模型	0.924	**0.879**	**57092**

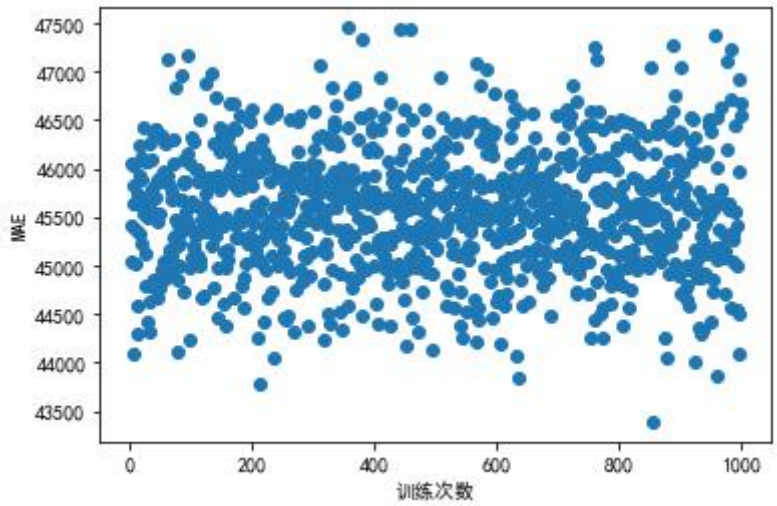
观察可得，极端森林回归模型在测试集上的 R 方值最大，RMSE 值最小，**故对房价预测问题，选择建立极端森林回归模型。**

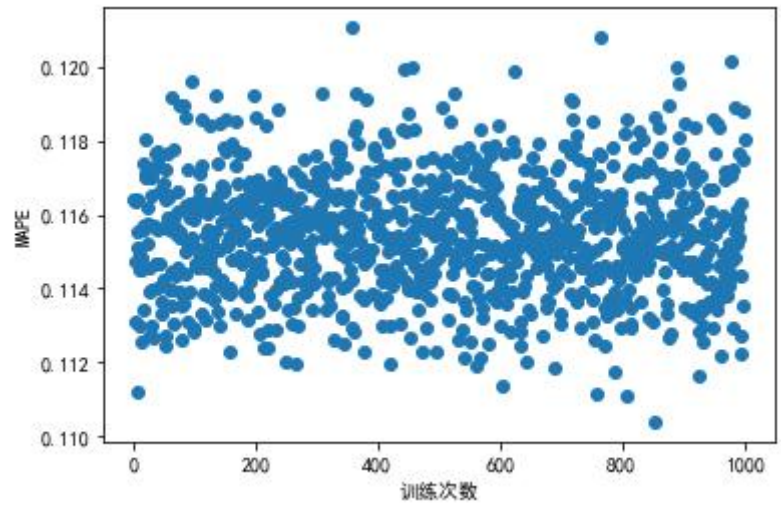
3 模型评估

对极端森林回归模型训练 1000 次，观察不同轮次训练的 R 方值、RMSE 值、RAE 值和 RAPE 值的变化，测试结果如下图所示：









由上图可知，在不同轮次中，模型训练后的 R 方值的波动区间长度（约 0.02）远小于各轮次 R 方值均值（约 0.866），RMSE 的波动区间长度（约 4000）远小于各轮次 RMSE 均值（约 59500），MAE 的波动区间长度（约 3000）远小于各轮次 MAE 均值（约 45500），这在一定程度上反映了模型不错的鲁棒性。

同时，MAE 在[43500,47500]内波动，这远小于数据集中的房价平均值 454343，

这在一定程度上反映了极端森林回归模型的预测具有较小的偏差。观察可得，NAPE 的平均值为 11.5%，这也能说明模型预测误差较小。

最后，R 方值均值约为 0.865 也说明了，模型能以 86%左右的比率解释因变量房价的变异性。

4 模型的问题与改进方向

- 在数据预处理的过程中，未考虑 PCA（为保持模型可解释性故而放弃使用 PCA）、主成分分析等降维方法，对削弱各特征间的耦合关系未采取任何措施。
- 极端森林模型的超参数可能未调至最优，这限制了该模型的预测效果，使得 R 方值偏小，MAE、MAPE 和 RMSE 值偏大。
- 没有尝试将多个模型融合起来预测，这限制了模型的预测效果。
- 训练集测试集划分比例等初始参数可能不是最优的。

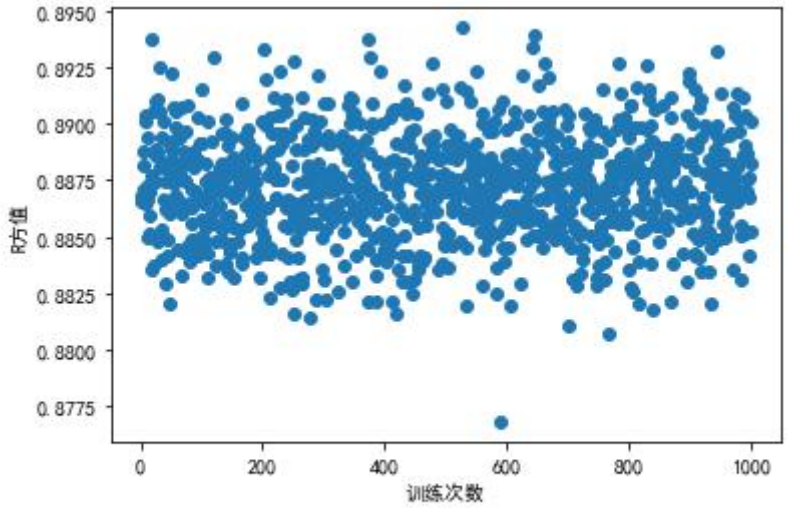
5 模型问题的部分解决

Python 中的 VotingRegressor 函数，可以实现模型融合的效果。其具体原理是依照设定的权重对多个模型的预测结果求平均值。根据 2.12 中的结果，极端森林回归模型和 SVR 模型预测结果的 RMSE 值最小，故尝试融合这两个模型，以最小化 RMSE、MAE 和 MAPE 值。

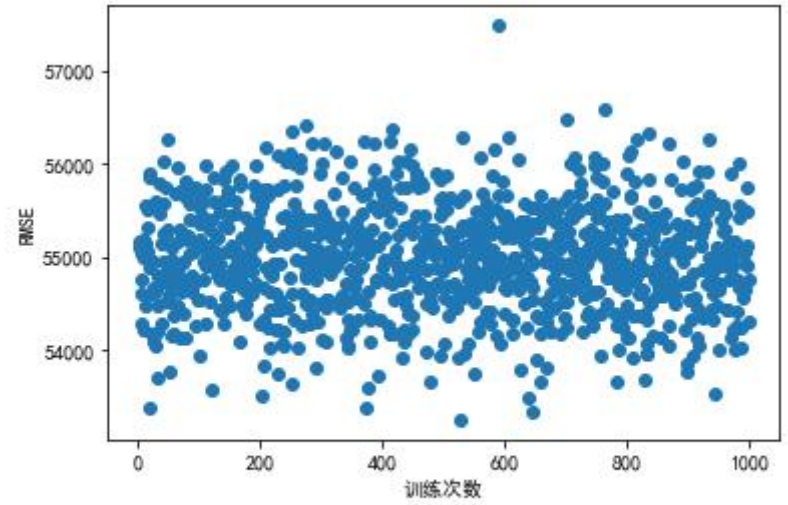
```
#尝试融合模型
from sklearn.ensemble import VotingRegressor
r1 = SVR(C=3,gamma=0.1,kernel='rbf')
r2 = KNeighborsRegressor(n_neighbors=21,weights='distance')
r3 = ExtraTreesRegressor(min_samples_split=9,n_estimators=50)
com = VotingRegressor([('lr', r1), ('r3', r3)])
com.fit(X_train_scaled,Y_train_scaled)
com_pre=get_originY(com.predict(X_test_scaled))
print("调整参数后的模型在训练集得分：{:.3f}".format(com.score(X_train_scaled,Y_train_scale
print("调整参数后的模型在测试集得分：{:.3f}".format(com.score(X_test_scaled,Y_test_scaled)
print('RMSE:{:.3f}'.format(np.sqrt(mean_squared_error(Y_test,com_pre))))
```

```
调整参数后的模型在训练集得分：0.897
调整参数后的模型在测试集得分：0.887
RMSE:54977.920
```

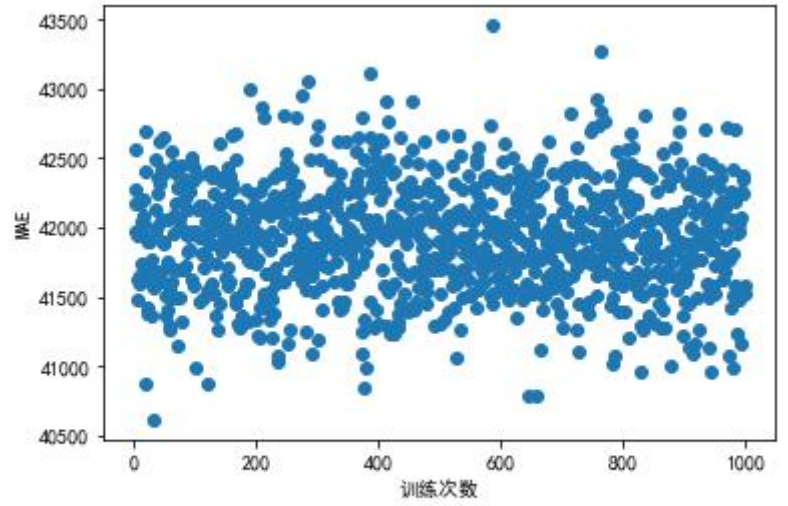
与极端森林回归模型的 RMSE 值 57092 相比，融合模型取得了一定幅度的优化效果。然后再重复测试 1000 次，观察该融合模型的最佳预测结果，并输出每个指标的最优值。



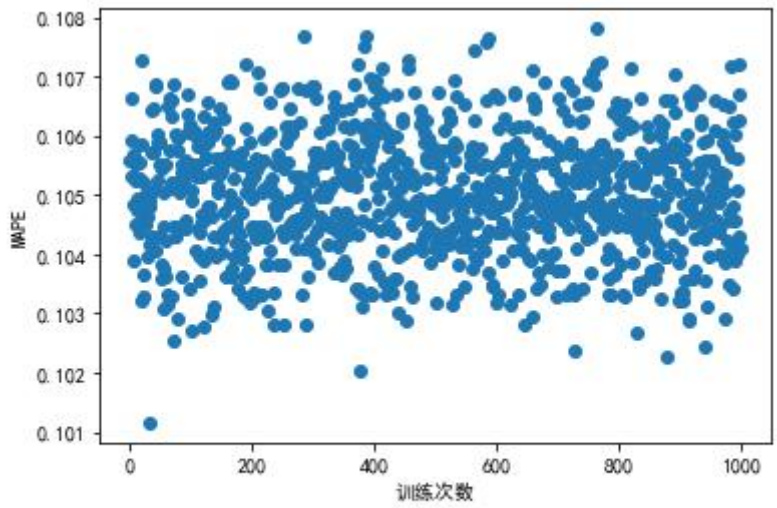
0.8943131289399053



53252.6614445043



40607.96811511493



0.10114563850614391

由以上测试结果图可知，融合模型的预测结果的最大 R 方值为 0.894，最小 RMSE 值为 53252，最小 MAE 值为 40607，最小 MAPE 值为 10.11%。这表明融合模型预测效果较单一模型更理想，极端森林模型中存在的问题得到了部分解决。

参考文献

[1] https://zhuanlan.zhihu.com/p/492072843
[2] https://www.jianshu.com/p/fc3d3c2cd3ca
[3] https://dandelioncloud.cn/article/details/1530345112005787649
[4] https://zhuanlan.zhihu.com/p/480224154
[5] https://zhuanlan.zhihu.com/p/89873990
[6] https://zhuanlan.zhihu.com/p/89453104
[7] https://mp.weixin.qq.com/s/tDn9_4-EFolRth87O-E4NA
[8] https://blog.csdn.net/weixin_48419914/article/details/121671548
[9] https://blog.csdn.net/FrankieHello/article/details/82024526
[10] http://www.javashuo.com/article/p-olgikdmz-gp.html
[11] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingRegressor.html?highlight=voting#sklearn.ensemble.VotingRegressor
[12] https://zhuanlan.zhihu.com/p/69415215