kaggle房价预测笔记

STEP1:检视数据

```
import pandas as pd
import numpy as np

train_df = pd.read_csv("input/train.csv", index_col=0)
test_df = pd.read_csv("input/test.csv", index_col=0)

# 默认看前五行的
print(train_df.head())
```

STEP2:数据预处理

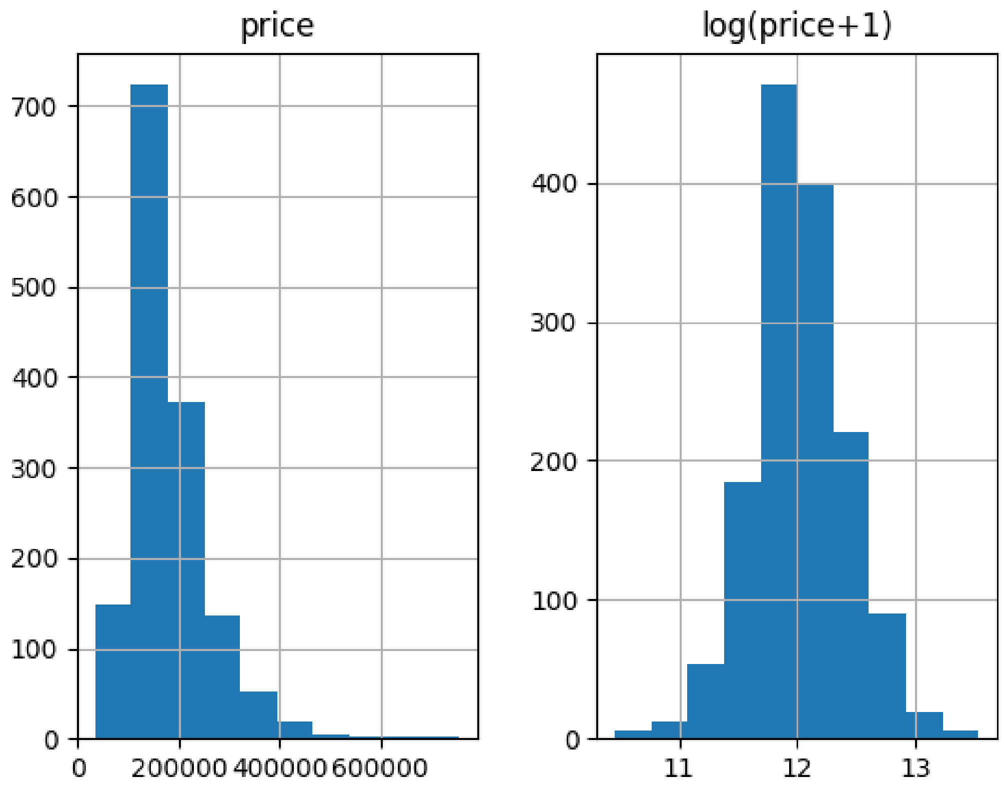
列表中很多数据都是英文单词还有些是无意义的数字，我们需要进行处理：

合并数据

我们要对所有数据进行处理，所以训练集和测试集都需要处理。

测试集中没有“SalePrice”列，所以我们需要对训练集里面的“SalePrice”进行处理工作。

```
# 观察SalePrice数据
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

train_df = pd.read_csv("input/train.csv", index_col=0)
test_df = pd.read_csv("input/test.csv", index_col=0)

prices = pd.DataFrame({"price": train_df["SalePrice"], "log(price+1)":
np.log1p(train_df["SalePrice"])})
prices.hist()
plt.show()
```




```
# 数据拼接
y_train = np.log1p(train_df.pop("SalePrice"))
all_df = pd.concat((train_df, test_df), axis=0)
```

类特征工程处理

观察到“MSSubClass”类虽然都是数字表示，但是数字只是一个代号，并没有大小关系：

```
Id,MSSubClass,MSZoning,LotFrontage,LotArea,Street,Alley,LotShape,Lar
1,60,RL,65,8450,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,CollgCr,Norm,Norm,
2,20,RL,80,9600,Pave,NA,Reg,Lvl,AllPub,FR2,Gtl,Veenker,Feedr,Norm,1F
3,60,RL,68,11250,Pave,NA,IR1,Lvl,AllPub,Inside,Gtl,CollgCr,Norm,Norr
4,70,RL,60,9550,Pave,NA,IR1,Lvl,AllPub,Corner,Gtl,Crawfor,Norm,Norm,
5,60,RL,84,14260,Pave,NA,IR1,Lvl,AllPub,FR2,Gtl,NoRidge,Norm,Norm,1F
6,50,RL,85,14115,Pave,NA,IR1,Lvl,AllPub,Inside,Gtl,Mitchel,Norm,Norr
7,20,RL,75,10084,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,Somerst,Norm,Norr
8,60,RL,NA,10382,Pave,NA,IR1,Lvl,AllPub,Corner,Gtl,NWAmes,PosN,Norm,
9,50,RM,51,6120,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,OldTown,Artery,Nor
10,190,RL,50,7420,Pave,NA,Reg,Lvl,AllPub,Corner,Gtl,BrkSide,Artery,A
11,20,RL,70,11200,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,Sawyer,Norm,Norr
12,60,RL,85,11924,Pave,NA,IR1,Lvl,AllPub,Inside,Gtl,NridgHt,Norm,Nor
13,20,RL,NA,12968,Pave,NA,IR2,Lvl,AllPub,Inside,Gtl,Sawyer,Norm,Norr
14,20,RL,91,10652,Pave,NA,IR1,Lvl,AllPub,Inside,Gtl,CollgCr,Norm,Nor
15,20,RL,NA,10920,Pave,NA,IR1,Lvl,AllPub,Corner,Gtl,NAmes,Norm,Norm,
16,45,RM,51,6120,Pave,NA,Reg,Lvl,AllPub,Corner,Gtl,BrkSide,Norm,Norr
17,20,RL,NA,11241,Pave,NA,IR1,Lvl,AllPub,CulDSac,Gtl,NAmes,Norm,Norr
18,90,RL,72,10791,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,Sawyer,Norm,Norr
19,20,RL,66,13695,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,SawyerW,RRAe,Nor
20,20,RL,70,7560,Pave,NA,Reg,Lvl,AllPub,Inside,Gtl,NAmes,Norm,Norm,1
```

对“MSSubClass”类的解释如下;

```
MSSubClass: Identifies the type of dwelling involved in the sale.

        20  1-STORY 1946 & NEWER ALL STYLES
        30  1-STORY 1945 & OLDER
        40  1-STORY W/FINISHED ATTIC ALL AGES
        45  1-1/2 STORY - UNFINISHED ALL AGES
        50  1-1/2 STORY FINISHED ALL AGES
        60  2-STORY 1946 & NEWER
        70  2-STORY 1945 & OLDER
        75  2-1/2 STORY ALL AGES
        80  SPLIT OR MULTI-LEVEL
        85  SPLIT FOYER
        90  DUPLEX - ALL STYLES AND AGES
       120  1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150  1-1/2 STORY PUD - ALL AGES
       160  2-STORY PUD - 1946 & NEWER
       180  PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190  2 FAMILY CONVERSION - ALL STYLES AND AGES
```

把category的变量转变成numerical表达形式

当我们用numerical来表达categorical的时候，要注意，数字本身有大小的含义，所以乱用数字会给之后的模型学习带来麻烦。于是我们可以用One-Hot的方法来表达category。

pandas自带的get_dummies方法，可以帮你一键做到One-Hot。

> 在数据分析和机器学习领域中，"category"（或 "categorical"）变量是指具有有限个离散取值的变量，它描述了数据的类别或标签。
>
> Category 变量通常用于表示具有固定类别的特征。这些类别可以是预定义的，例如性别（男、女）、颜色（红、绿、蓝）等，也可以是基于观察数据中的不同值进行自动识别的，例如用户的职业、产品的类型等。
>
> 与连续变量不同，Category 变量没有顺序或大小的概念，它们只是表示不同的类别或标签。在统计分析和机器学习中，处理 Category 变量通常需要进行特殊的编码或转换，以便将其表示为适合模型处理的数值形式。
>
> 常见的 Category 变量编码方法包括：
>
> 1. 无序编码（Nominal Encoding）：将每个类别分配一个唯一的整数或字符串标识符，例如使用独热编码（One-Hot Encoding）或标签编码（Label Encoding）。
> 2. 有序编码（Ordinal Encoding）：对类别进行排序，并为每个类别分配一个整数或字符串标识符，以表示它们之间的顺序关系。
>
> Category 变量在数据分析和机器学习中扮演重要角色，因为它们可以提供有关数据的分类信息，帮助描述和解释数据，并在许多任务中用作特征输入。

独热编码

```
pd.get_dummies(all_df["MSSubClass"], prefix='MSSubClass')
```

以此类推，将所有的Category 变量使用One-Hot Encoding表示：

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

train_df = pd.read_csv("input/train.csv", index_col=0)
test_df = pd.read_csv("input/test.csv", index_col=0)

y_train = np.log1p(train_df.pop("SalePrice"))
all_df = pd.concat((train_df, test_df), axis=0)
all_dummy_df = pd.get_dummies(all_df["MSSubClass"], prefix='MSSubClass')
all_df = pd.concat([all_df, all_dummy_df], axis=1)
all_dummy_df = pd.get_dummies(all_df)
all_dummy_df.pop("MSSubClass")
print(all_dummy_df.head())
```

处理缺失数据

最常见的为直接删除，取平均值，取众数：

```
# 取平均值

mean_cols = all_dummy_df.mean()
all_dummy_df = all_dummy_df.fillna(mean_cols)
print(all_dummy_df.isnull().sum().sum())
```

标准化numerical数据

一般来说回归的分类器对源数据要求比较高，需要数据在标准分布里面，不要让数据间相差太大。

注意：OneHot数据不需要标准化

寻找需要标准化的数据：

```
# 所有数据类的数据都需要标准化，
all_df = pd.concat((train_df, test_df), axis=0)
numeric_cols = all_df.columns[all_df.dtypes != 'object']
numeric_cols = numeric_cols.drop("MSSubClass")
```

标准化数据：

```
numeric_cols_means = all_dummy_df.loc[:, numeric_cols].mean()
numeric_cols_std = all_dummy_df.loc[:, numeric_cols].std()
all_dummy_df.loc[:, numeric_cols] = (all_dummy_df.loc[:, numeric_cols] -
numeric_cols_means)/numeric_cols_std
```

重新获得训练集和测试集：

```
dummy_train_df = all_dummy_df.loc[train_df.index]
dummy_test_df = all_dummy_df.loc[test_df.index]
```

Step3 建立模型

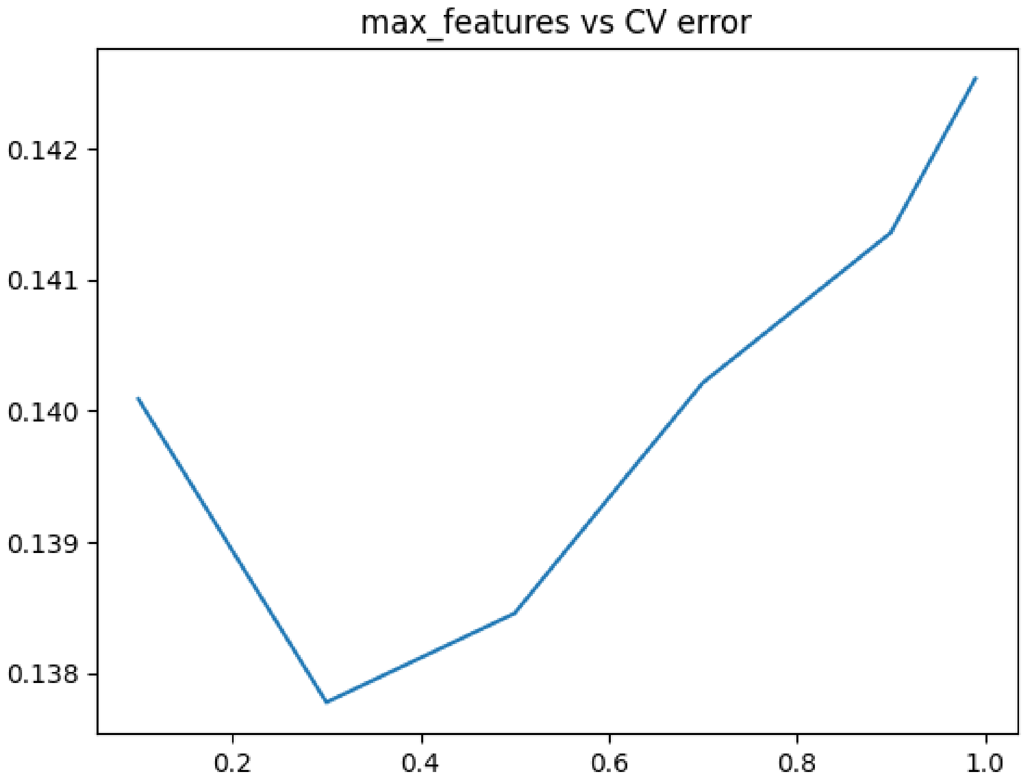
代入模型寻找最优值：

```python
alphas = np.linspace(0, 20, 20)
test_scores = []
for alpha in alphas:
    clf = Ridge(alpha)
    test_score = np.sqrt(-cross_val_score(clf, x_train, y_train, cv=10,
scoring='neg_mean_squared_error'))
    test_scores.append(np.mean(test_score))
```

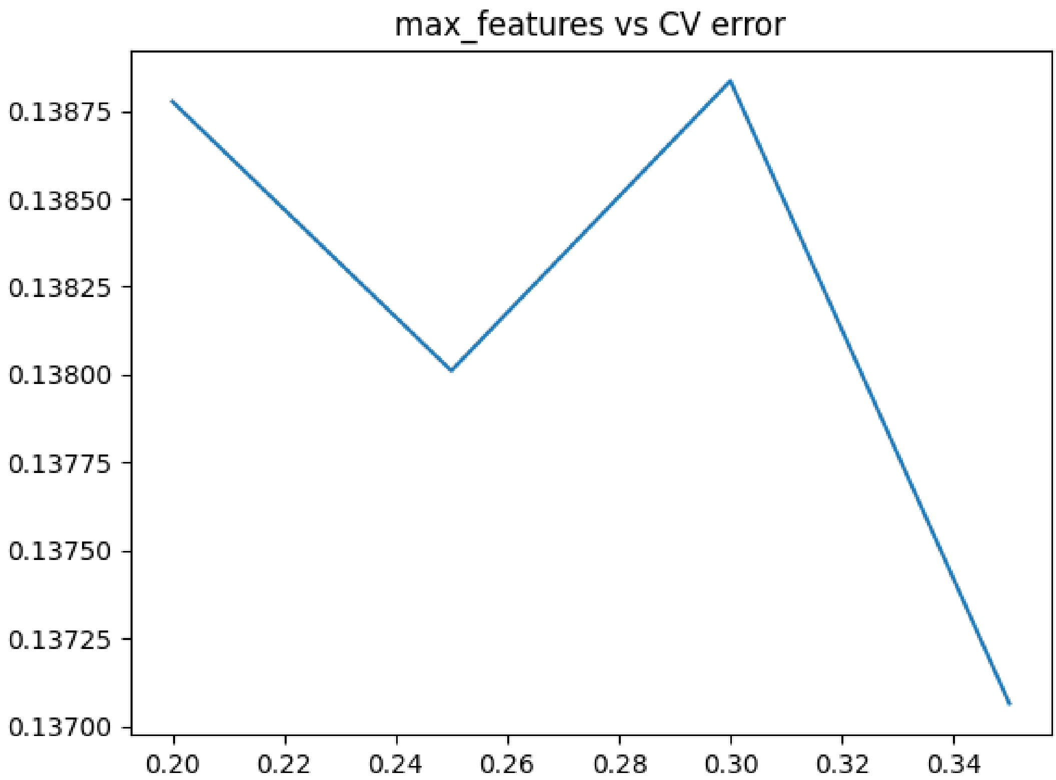
```python
# 使用 alpha=2.5 训练最终的 Ridge 模型
final_alpha = 2.5
final_clf = Ridge(final_alpha)
final_clf.fit(x_train, y_train)

# 预测测试集的 SalePrice
predicted_prices = np.expm1(final_clf.predict(x_test))
```

使用随机森林方法再做一次:

```python
max_features = [.1, .3, .5, .7, .9, .99]
test_scores = []
for max_feat in max_features:
    clf = RandomForestRegressor(n_estimators=200, max_features=max_feat)
    test_score = np.sqrt(-cross_val_score(clf, x_train, y_train, cv=5,
scoring="neg_mean_squared_error"))
    test_scores.append(np.mean(test_score))

plt.plot(max_features, test_scores)
plt.title("max_features vs CV error")
plt.show()
```


max_features vs CV error
0.142
0.141
0.140
0.139
0.138
0.2
0.4
0.6
0.8
1.0

进一步细分：


max_features vs CV error
0.13875
0.13850
0.13825
0.13800
0.13775
0.13750
0.13725
0.13700
0.20
0.22
0.24
0.26
0.28
0.30
0.32
0.34

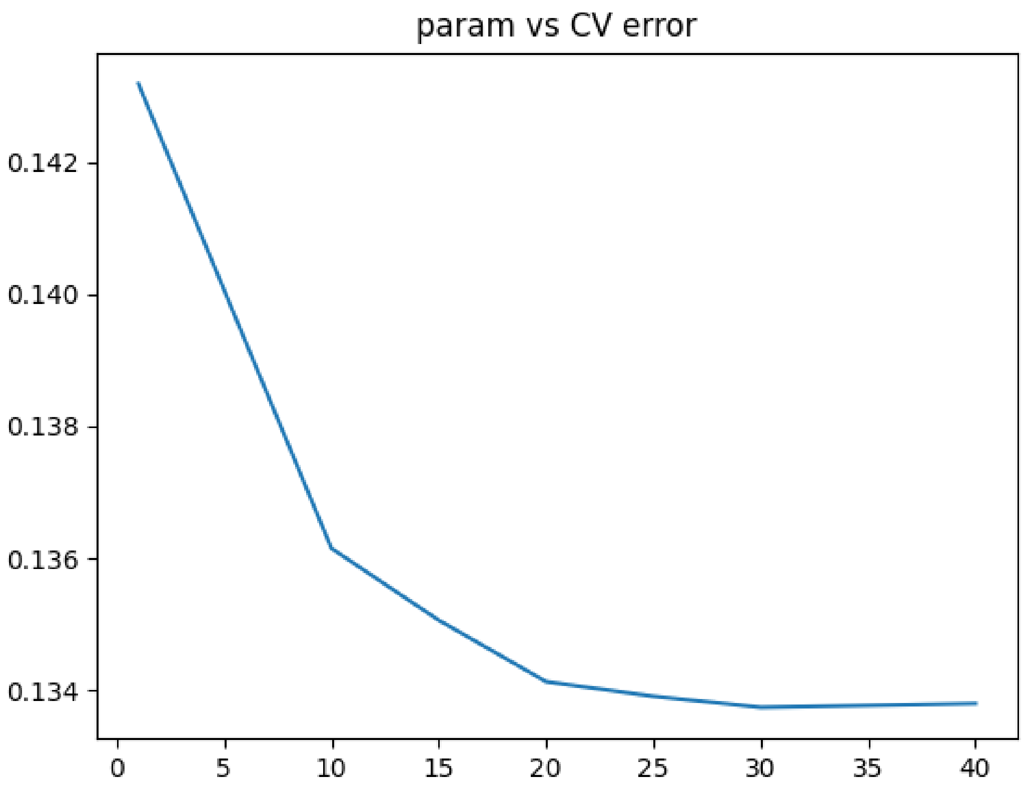
Ensemble

这里我们选择最简单的，取平均法：

```
# 使用 alpha=2.5 训练最终的 Ridge 模型
final_alpha = 2.5
final_clf = Ridge(final_alpha)
final_clf.fit(x_train, y_train)

# 预测测试集的 SalePrice
predicted_prices1 = np.expm1(final_clf.predict(x_test))

# 使用最佳参数模型
final_feature = 0.37
final_clf = RandomForestRegressor(n_estimators=200, max_features=final_feature)
final_clf.fit(x_train, y_train)

# 预测测试集的 SalePrice
predicted_prices2 = np.expm1(final_clf.predict(x_test))

# 将预测结果与对应的 Id 组合成 DataFrame
predictions_df = pd.DataFrame({"Id": test_df.index, "SalePrice":
(predicted_prices1+predicted_prices2)/2})

# 将 DataFrame 写入到 CSV 文件
predictions_df.to_csv("output/predictions_aver.csv", index=False)
```

使用BaggingRegressor对Ridge改进：

```
# 使用 alpha=15 训练最终的 Ridge 模型
final_alpha = 15
ridge = Ridge(final_alpha)
params = [1, 10, 15, 20, 25, 30, 40]
test_scores = []
for param in params:
    clf = BaggingRegressor(n_estimators=param, estimator=ridge)
    test_score = np.sqrt(-cross_val_score(clf, x_train, y_train, cv=10,
scoring="neg_mean_squared_error"))
    test_scores.append(np.mean(test_score))

plt.plot(params, test_scores)
plt.title("param vs CV error")
plt.show()
```

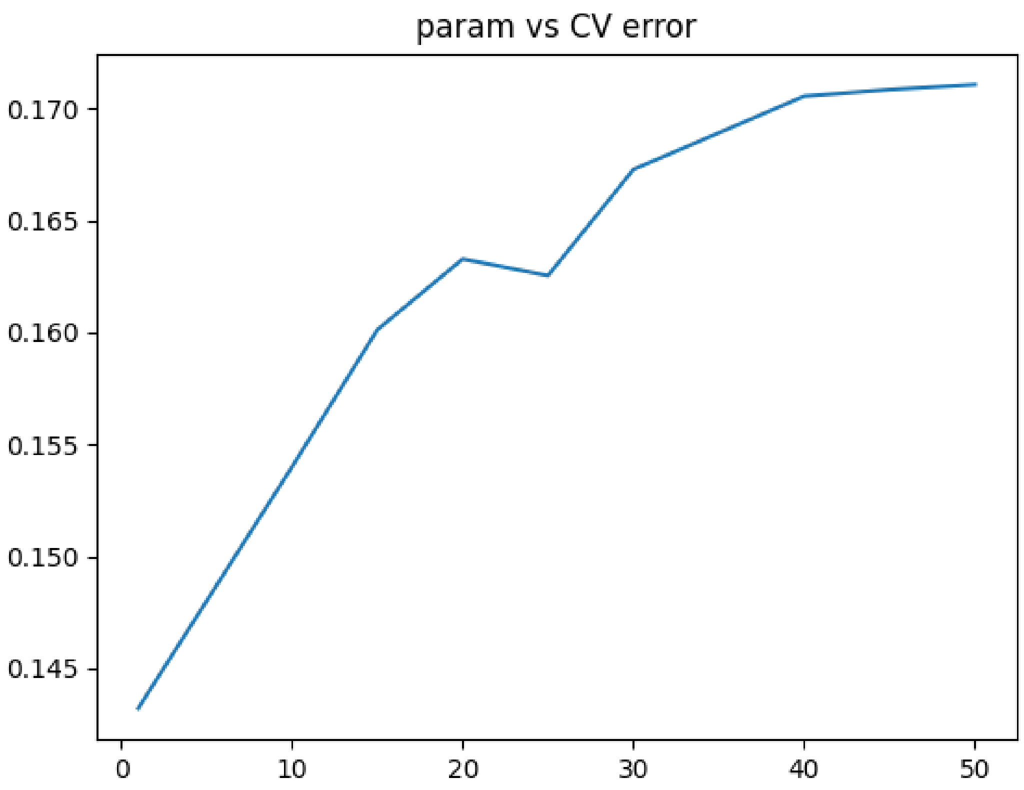


最终代码：

```
# 使用 alpha=15 训练最终的 Ridge 模型
final_alpha = 15
ridge = Ridge(final_alpha)
clf = BaggingRegressor(n_estimators=30, estimator=ridge)
clf.fit(x_train, y_train)

# 预测测试集的 SalePrice
predicted_prices2 = np.expm1(clf.predict(x_test))

# 将预测结果与对应的 Id 组合成 DataFrame
predictions_df = pd.DataFrame({"Id": test_df.index, "SalePrice":
predicted_prices2})

# 将 DataFrame 写入到 CSV 文件
predictions_df.to_csv("output/predictions_BaggingRegressor_Ridge.csv",
index=False)
```

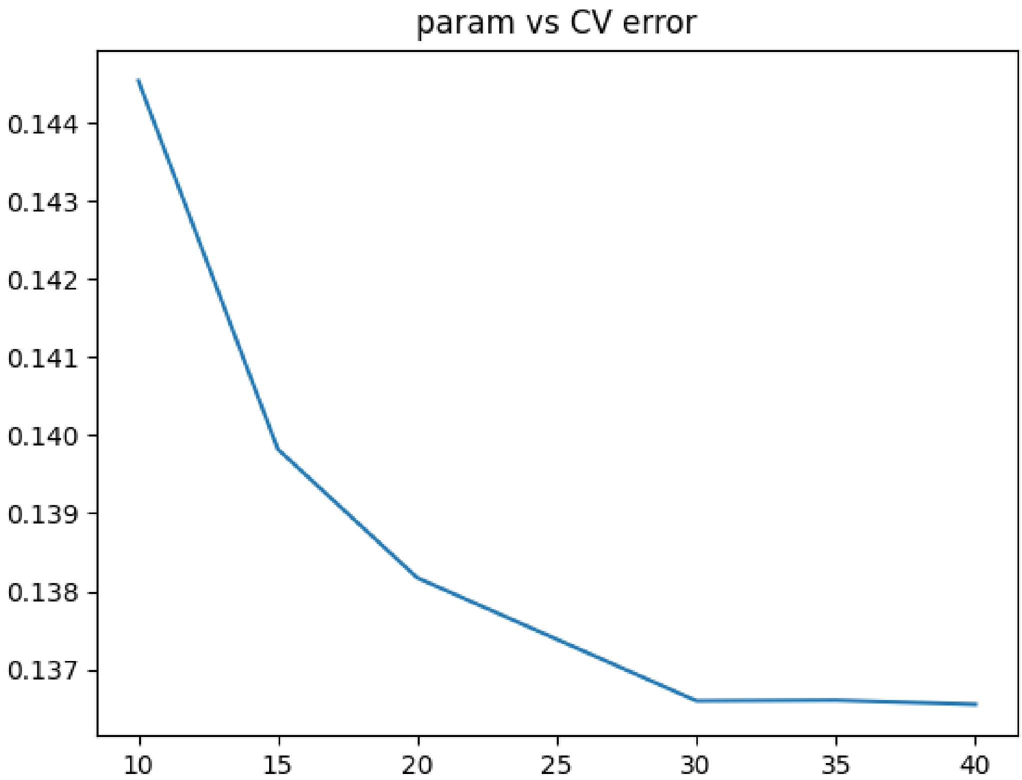
AdaBoostRegressor

```python
# 使用 alpha=15 训练最终的 Ridge 模型
final_alpha = 15
ridge = Ridge(final_alpha)
params = [1, 10, 15, 20, 25, 30, 40, 45, 50]
test_scores = []
for param in params:
    clf = AdaBoostRegressor(n_estimators=param, estimator=ridge)
    test_score = np.sqrt(-cross_val_score(clf, x_train, y_train, cv=10,
scoring="neg_mean_squared_error"))
    test_scores.append(np.mean(test_score))

plt.plot(params, test_scores)
plt.title("param vs CV error")
plt.show()
```



最强XGBoost

```python
# 使用 alpha=15 训练最终的 Ridge 模型
final_alpha = 15
ridge = Ridge(final_alpha)
params = [10, 15, 20, 30, 35, 40]
test_scores = []
for param in params:
    clf = XGBRegressor(n_estimators=param)
    test_score = np.sqrt(-cross_val_score(clf, x_train, y_train, cv=10,
scoring="neg_mean_squared_error"))
    test_scores.append(np.mean(test_score))

plt.plot(params, test_scores)
plt.title("param vs CV error")
plt.show()
```


param vs CV error
0.144
0.143
0.142
0.141
0.140
0.139
0.138
0.137
10
15
20
25
30
35
40