## 项目报告

**机器学习课程项目：房价预测**

|  |  |  |
| --- | --- | --- |
| **队长:** | 钱克洋 | 201900800151 |
| **成员:** | 赵旭东 | 201900800170 |
|  | 徐晏辰 | 201900800127 |

**2020 年 12 月**

## 摘 要

在本实验报告中，我们将通过线性回归中的岭回归的方法完成预测房价模型。我们在数据清洗与特征重建过程中使用了缺失值补充、特征选择、数值型数据数值填充、one-hot编码填充非数值型数据、特征缩放、标准化等手段满足正则化线性回归的数据要求，包括线性、独立性、正态性、自变量间不存在多重共线、因变量为连续变量。由于数据集规模小，我们将所有数据作为验证集，使用1次10折交叉验证的方法完成模型构建、参数调整、模型评估，最后用所有数据训练模型将模型对test文件数据所预测的结果输出到文件。我们通过查阅资料与尝试总结出一套完整的调参流程，尝试使用opt.fmin\_tnc优化器并吸取经验，为了模型需要反复进行特征重建，建立多个模型，输出1次10折交叉验证得出的模型均方根对数误差（RMSLE）、均方误差（MSE）性能的平均结果并进行模型性能比较与可视化。

## 1 介绍

房价预测是吴恩达课程中被用作线性回归案例的提到的第一个机器学习问题，在Kaggle上也是十分热门的机器学习竞赛新手入门项目，许多人提供了或面向新手或进阶性的优质参考文档，其中许多文档也使用了线性回归有关算法。我们原以为这个项目会十分简单，却在实践中遇到大量意想不到的问题。

我们使用的生产工具是JupyterLab与Jupyter Notebook。将Jupyter Notebook路径转到’文件夹中文手写数字识别\data’（即代码文件所在位置）下即可运行.ipynb代码文件。

### 1.1 主要问题

1. 特征的数量应如何选择；

2. 数据清洗与特征重建需要满足哪些要求；

3. 应如何适当划分数据集；

4．如何合理评估特征重建与模型整体性能的好坏；

5. 如何最高效、最合理地完成模型参数调整；

6. 如何有效使用opt.fmin\_tnc优化器；

### 1.2 解决方案

1. 特征的数量应如何选择？

容易分析出，特征的数量并不是越多越好的，增加有效特征数量的效果是一条倒“U”曲线：最初模型性能随之上升，但由于数据集规模的限制，特征数量过多时则必然导致愈发严重的过拟合。

在Kaggle获得星数最高的面向新手的帮助文档只推荐使用4个特征（以下简称4特征法），即GrLivArea、TotalBsmtSF、YearBuilt、OverallQual，而许多其他优质文档则使用了几十个特征。在实践中我们最初使用并评估了4特征法，然后独立提取了数十个特征并试验调参，最后则使用了16个（算上偏置17个）特征的模型。使用数十个特征的模型时，特征矩阵规模会变得较大，而当矩阵规模增大时梯度下降速度α也要增大才能在回归次数不多的情况下得到模型最好结果（实际上对数十特征的数据集，α值要远大于1才能成功回归），这导致即使每个特征值几乎都已经缩放到正负1以内了，α = 1时在矩阵平方运算中仍然出现数据溢出（这是我们推测出现数据溢出的可能原因）。而α小的话，回归次数不多时执行结果远不如我们最初试用的α = 1e-6的4特征法。若增加回归次数，则程序运行时间比较让人难以接受。当然，也可改用64位变量或者使用高精度等等，但我们选择将此作为教训，重新评估特征重建过程并缩减变量个数。

实际上，若仅追求特征数量，还会导致无关特征引入增大计算量、特征相关性高等问题。

不过，第二个项目使用逻辑回归时未复现上述bug，所以可能是之前的特征提取结果或者模型有什么特殊性。

其他问题的解决方案就留到下一部分去说吧。

## 2 方法

由于本实验研究方法的结构并不复杂，直接用文字叙述流程。

我们使用的生产工具是JupyterLab与Jupyter Notebook。

首先进行数据清洗并重建特征，输出到文档中。

读取文档，将整个数据集用作交叉验证集，通过1次10折交叉验证的过程调试模型。

若对重建的特征不满意，则返回第一步，继续进行数据清洗与特征重建。

经过几个循环，最终调试完毕模型，然后继续用交叉验证的方法评估对比模型性能。

最后用所有数据训练模型，并将模型对test文件数据所预测的结果输出到文件。

### 2.1 数据清洗 & 特征重建

数据清洗与特征重建是为了满足模型优化的需要。

由于数据集规模小等原因，特征数量不能过多。由于使用正则化线性回归算法，理论上数据需要满足线性、独立性、正态性、方差齐性、自变量间不存在多重共线、因变量为连续变量等要求，因此我们的数据清洗与特征重建主要包含包含数据集理解与分析、缺失值补充、特征选择、数值填充数值型数据、one-hot编码填充非数值型数据、特征缩放、标准化、确认特征、反思重来等步骤。

以下为源代码，由于使用JupyterLab，代码&注释文本囊括了项目的全过程。为区分代码块，每个代码块间空一行，代码块执行结果附在代码块之下。

在这些代码执行完毕后，也对test文件数据集进行了同样的处理，由于过程雷同，在此不赘述了。

import pandas as pd

import numpy as np

all\_data = pd.read\_csv('train.csv')

pd.set\_option('display.max\_columns', None) # 显示完整的列

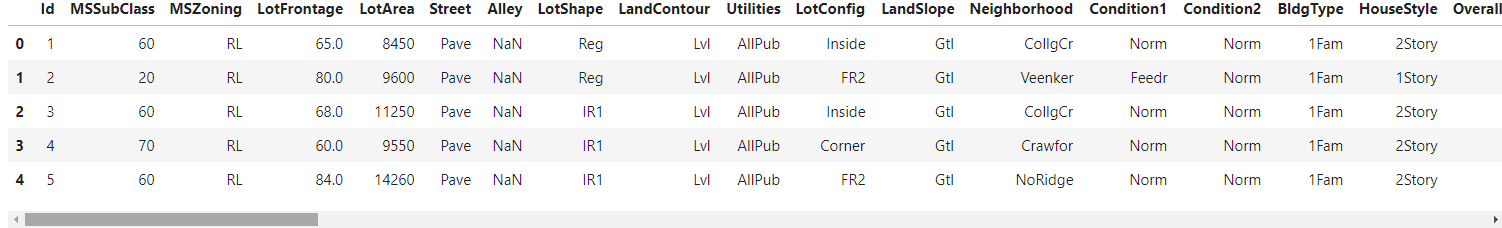
pd.set\_option('display.max\_rows', None) # 显示完整的行

pd.set\_option('display.expand\_frame\_repr', False) # 设置不折叠数据

pd.set\_option('display.max\_colwidth', 100)

# 显示前五行

all\_data.head(5)



# ID列对之后的模型建立没有帮助，应该删除

all\_data\_ID = all\_data['Id']# 删除前备份

all\_data.drop("Id", axis = 1, inplace = True)

# 显示当前样本和特征数量

print(all\_data.shape)

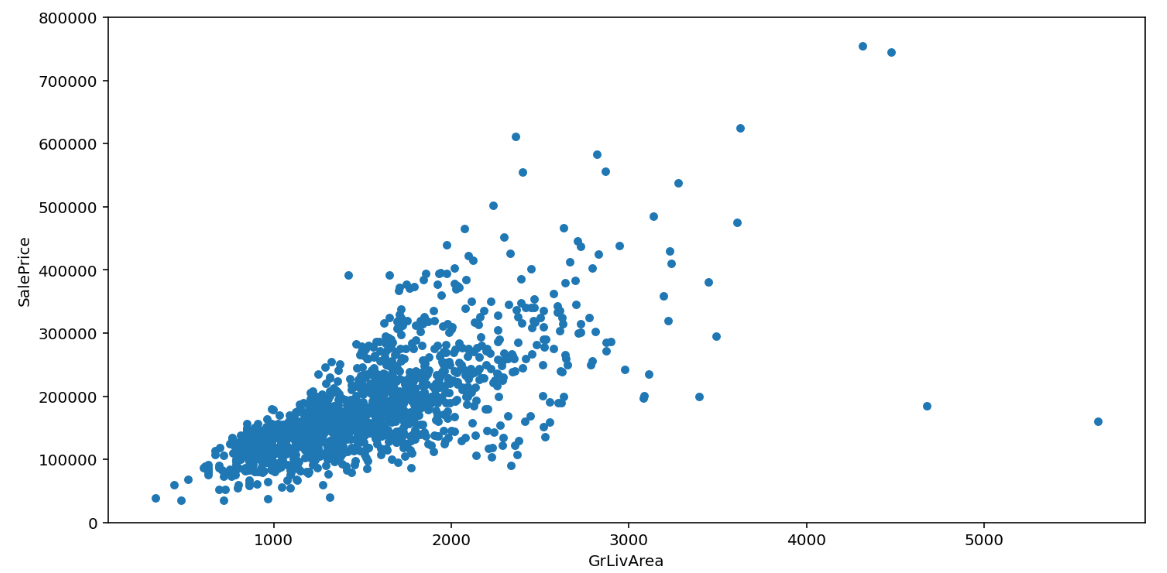


# 离群值处理

var = 'GrLivArea'

data = pd.concat([all\_data['SalePrice'], all\_data[var]], axis=1)

data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));



# 可以发现右侧有两个数据显然不合理，予以去除

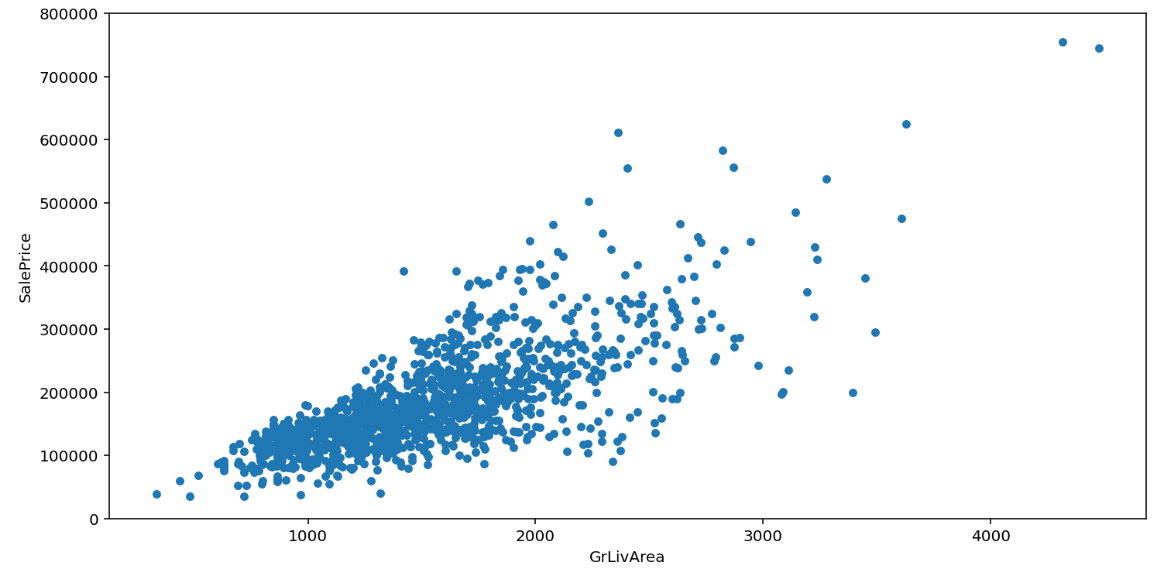
# 删除极端离群值通常对回归有益无害

all\_data = all\_data.drop(all\_data[(all\_data['GrLivArea']>4000) & (all\_data['SalePrice']<300000)].index)

var = 'GrLivArea'

data = pd.concat([all\_data['SalePrice'], all\_data[var]], axis=1)

data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));



# 将其他数值型数据按照同样方法图示分析，如果与价格没有明显关系，可以考虑删除（具体每个图示过程已省略）

# 如下方Street特征，很明显大部分样本都属'Pave’,仅有几个属于'Grvl'，该特征应予以删除

all\_data = all\_data.drop(all\_data[(all\_data['GrLivArea']>4000) & (all\_data['SalePrice']<300000)].index)

var = 'Street'# var换成不同数值型特征

data = pd.concat([all\_data['SalePrice'], all\_data[var]], axis=1)

data.plot.scatter(x=var, y='SalePrice', ylim=(0,800000));



# 缺失值补充

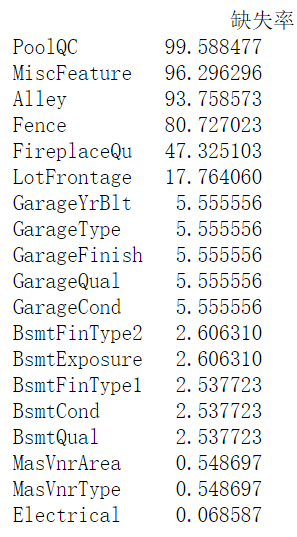
# 查看缺失率

all\_data\_na = (all\_data.isnull().sum() / len(all\_data)) \* 100

all\_data\_na = all\_data\_na.drop(all\_data\_na[all\_data\_na == 0].index).sort\_values(ascending=False)[:30]

missing\_data = pd.DataFrame({'缺失率' :all\_data\_na})

print(missing\_data)



# 依次补充缺失值

# PoolQC，代表泳池质量，NA代表无，因此用NA填充缺失值。

all\_data["PoolQC"] = all\_data["PoolQC"].fillna("NA")

# MiscFeature，代表其他未涵盖特征，NA代表无，因此用NA填充缺失值。

all\_data["MiscFeature"] = all\_data["MiscFeature"].fillna("NA")

# Alley，代表路的材质，NA代表无，因此用NA填充缺失值。

all\_data["Alley"] = all\_data["Alley"].fillna("NA")

# Fence，代表栅栏质量，NA代表无，因此用NA填充缺失值。

all\_data["Fence"] = all\_data["Fence"].fillna("NA")

# FireplaceQu，代表壁炉质量，NA代表无，因此用NA填充缺失值。

all\_data["FireplaceQu"] = all\_data["FireplaceQu"].fillna("NA")

# LotFrontage，代表最近街道的直线距离，因为由于每条街道与该房屋相连的区域与其邻近的其他房屋很可能相似，因此采用其社区的中位数来填充缺失值。

# 按照‘Neighborhood’分组，每组用中位数填充缺失值。

all\_data["LotFrontage"] = all\_data.groupby("Neighborhood")["LotFrontage"].transform(

lambda x: x.fillna(x.median()))

# GarageType,GarageFinish,GarageQual，GarageCond，代表与车库相关的非数值特征，因此用NA填充缺失值。

for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):

all\_data[col] = all\_data[col].fillna('NA')

# GarageYrBlt,GarageArea，GarageCars，代表与车库相关的数值特征，因此用0填充缺失值。

for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):

all\_data[col] = all\_data[col].fillna(0)

# BsmtFinSF1,BsmtFinSF2,BsmtUnfSF,TotalBsmtSF,BsmtFullBath,BsmtHalfBath,代表与地下室相关的数值特征，因此用0填充缺失值。

for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath'):

all\_data[col] = all\_data[col].fillna(0)

# BsmtQual,BsmtCond,BsmtExposure,BsmtFinType1,BsmtFinType2,代表与地下室相关的非数值特征，因此用NA填充缺失值。

for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):

all\_data[col] = all\_data[col].fillna('NA')

# MasVnrArea,MasVnrType,代表砖石漆面，NA或0代表无。

all\_data["MasVnrType"] = all\_data["MasVnrType"].fillna("NA")

all\_data["MasVnrArea"] = all\_data["MasVnrArea"].fillna(0)

# MSZoning，代表所在地区分类，用多数值（‘RL’）填充。

all\_data['MSZoning'] = all\_data['MSZoning'].fillna(all\_data['MSZoning'].mode()[0])

# Utilities，代表公共设施类型，该特征仅有几个样本存在不是‘ALLPUB’，对最终模型的建立帮助不大，予以去除

all\_data = all\_data.drop(['Utilities'], axis=1)

# Functional，用典型值替换

all\_data["Functional"] = all\_data["Functional"].fillna("Typ")

# Electrical，仅有一个样本缺失，用多数值('SBrkr')填充

all\_data['Electrical'] = all\_data['Electrical'].fillna(all\_data['Electrical'].mode()[0])

# KitchenQual，做法同上

all\_data['KitchenQual'] = all\_data['KitchenQual'].fillna(all\_data['KitchenQual'].mode()[0])

# Exterior1st，Exterior2nd，同上

all\_data['Exterior1st'] = all\_data['Exterior1st'].fillna(all\_data['Exterior1st'].mode()[0])

all\_data['Exterior2nd'] = all\_data['Exterior2nd'].fillna(all\_data['Exterior2nd'].mode()[0])

# SaleType，同上

all\_data['SaleType'] = all\_data['SaleType'].fillna(all\_data['SaleType'].mode()[0])

# 查看当前数据规模

print(all\_data.shape)



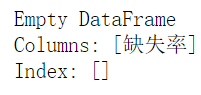
# 检查缺失值

all\_data\_na = (all\_data.isnull().sum() / len(all\_data)) \* 100

all\_data\_na = all\_data\_na.drop(all\_data\_na[all\_data\_na == 0].index).sort\_values(ascending=False)[:30]

missing\_data = pd.DataFrame({'缺失率' :all\_data\_na})

print(missing\_data)



# 缺失值已填充完毕。

# 显示并分析数值数据间的相关性

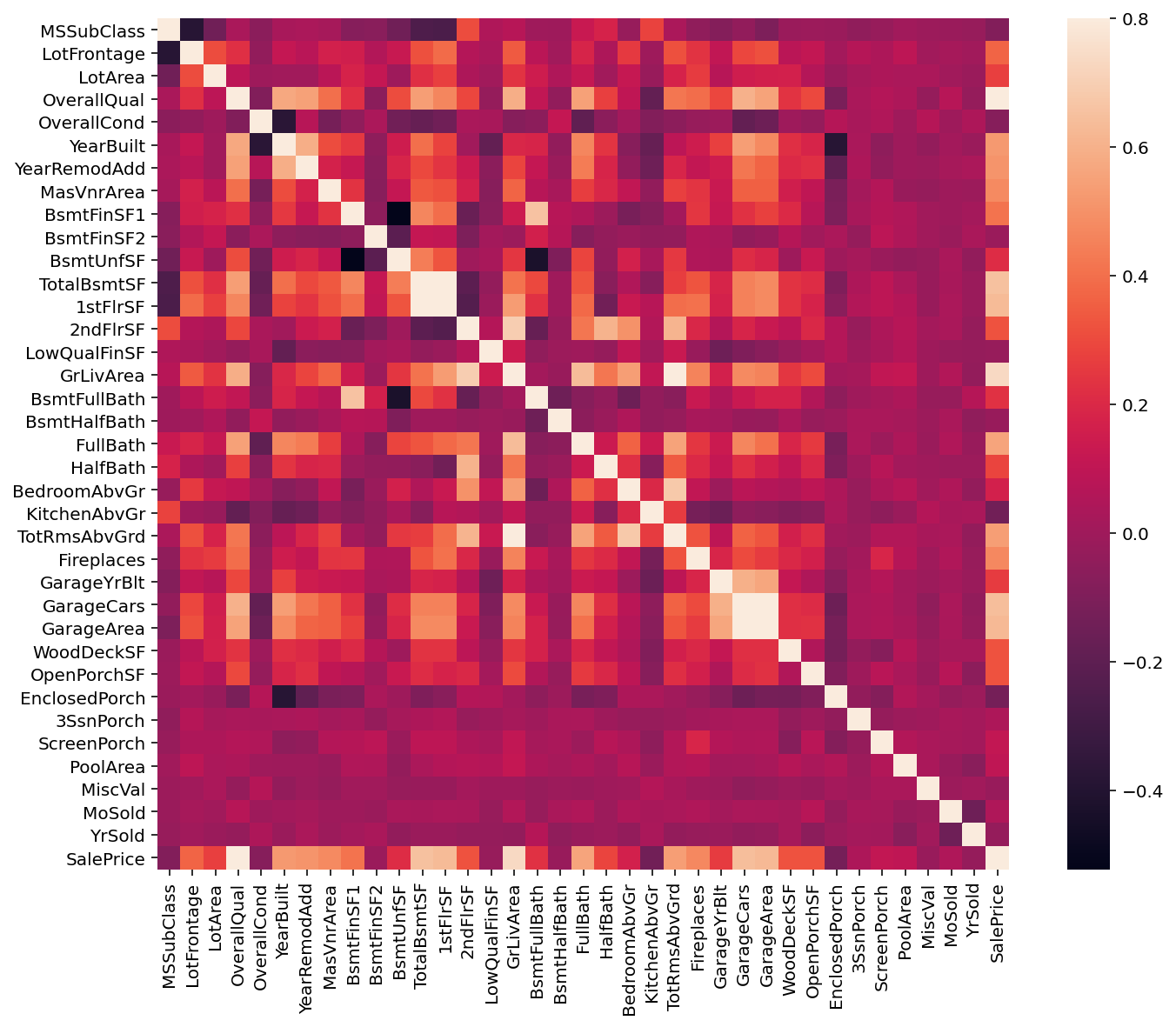
import matplotlib.pyplot as plt

import seaborn as sns

corrmat = all\_data.corr()

f, ax = plt.subplots(figsize=(12, 9))

sns.heatmap(corrmat, vmax=.8, square=True);



# 根据经验与相关矩阵，总面积对房价预测十分重要，添加一个每个房子的地下室、一楼和二楼的总面积特征

# all\_data['TotalSF'] = all\_data['TotalBsmtSF'] + all\_data['1stFlrSF'] + all\_data['2ndFlrSF']

t = all\_data['TotalBsmtSF'] + all\_data['1stFlrSF'] + all\_data['2ndFlrSF']

all\_data.insert(78, 'TotalSF', t)

# 根据特征间的相关性，特征与价格是否有明显关系，特征具体实质内容进行分析并删除一部分对最终模型建立帮助不大的特征

all\_data.drop(['MSZoning'], axis=1, inplace=True)

all\_data.drop(['LotArea'], axis=1, inplace=True)

all\_data.drop(['Street'], axis=1, inplace=True)

all\_data.drop(['Alley'], axis=1, inplace=True)

all\_data.drop(['LotShape'], axis=1, inplace=True)

all\_data.drop(['LotConfig'], axis=1, inplace=True)

all\_data.drop(['LandSlope'], axis=1, inplace=True)

all\_data.drop(['Neighborhood'], axis=1, inplace=True)

all\_data.drop(['Condition1'], axis=1, inplace=True)

all\_data.drop(['Condition2'], axis=1, inplace=True)

all\_data.drop(['BldgType'], axis=1, inplace=True)

all\_data.drop(['HouseStyle'], axis=1, inplace=True)

all\_data.drop(['RoofMatl'], axis=1, inplace=True)

all\_data.drop(['Exterior1st'], axis=1, inplace=True)

all\_data.drop(['Exterior2nd'], axis=1, inplace=True)

all\_data.drop(['MasVnrType'], axis=1, inplace=True)

all\_data.drop(['Foundation'], axis=1, inplace=True)

all\_data.drop(['BsmtQual'], axis=1, inplace=True)

all\_data.drop(['BsmtFinType1'], axis=1, inplace=True)

all\_data.drop(['BsmtFinSF1'], axis=1, inplace=True)

all\_data.drop(['BsmtFinType2'], axis=1, inplace=True)

all\_data.drop(['BsmtFinSF2'], axis=1, inplace=True)

all\_data.drop(['BsmtUnfSF'], axis=1, inplace=True)

all\_data.drop(['Heating'], axis=1, inplace=True)

all\_data.drop(['Electrical'], axis=1, inplace=True)

all\_data.drop(['LowQualFinSF'], axis=1, inplace=True)

all\_data.drop(['BsmtFullBath'], axis=1, inplace=True)

all\_data.drop(['BsmtHalfBath'], axis=1, inplace=True)

all\_data.drop(['KitchenQual'], axis=1, inplace=True)

all\_data.drop(['TotRmsAbvGrd'], axis=1, inplace=True)

all\_data.drop(['Functional'], axis=1, inplace=True)

all\_data.drop(['FireplaceQu'], axis=1, inplace=True)

all\_data.drop(['GarageType'], axis=1, inplace=True)

all\_data.drop(['GarageCond'], axis=1, inplace=True)

all\_data.drop(['PavedDrive'], axis=1, inplace=True)

all\_data.drop(['WoodDeckSF'], axis=1, inplace=True)

all\_data.drop(['OpenPorchSF'], axis=1, inplace=True)

all\_data.drop(['EnclosedPorch'], axis=1, inplace=True)

all\_data.drop(['3SsnPorch'], axis=1, inplace=True)

all\_data.drop(['ScreenPorch'], axis=1, inplace=True)

all\_data.drop(['PoolArea'], axis=1, inplace=True)

all\_data.drop(['Fence'], axis=1, inplace=True)

all\_data.drop(['MoSold'], axis=1, inplace=True)

all\_data.drop(['YrSold'], axis=1, inplace=True)

all\_data.drop(['SaleType'], axis=1, inplace=True)

all\_data.drop(['PoolQC'], axis=1, inplace=True)

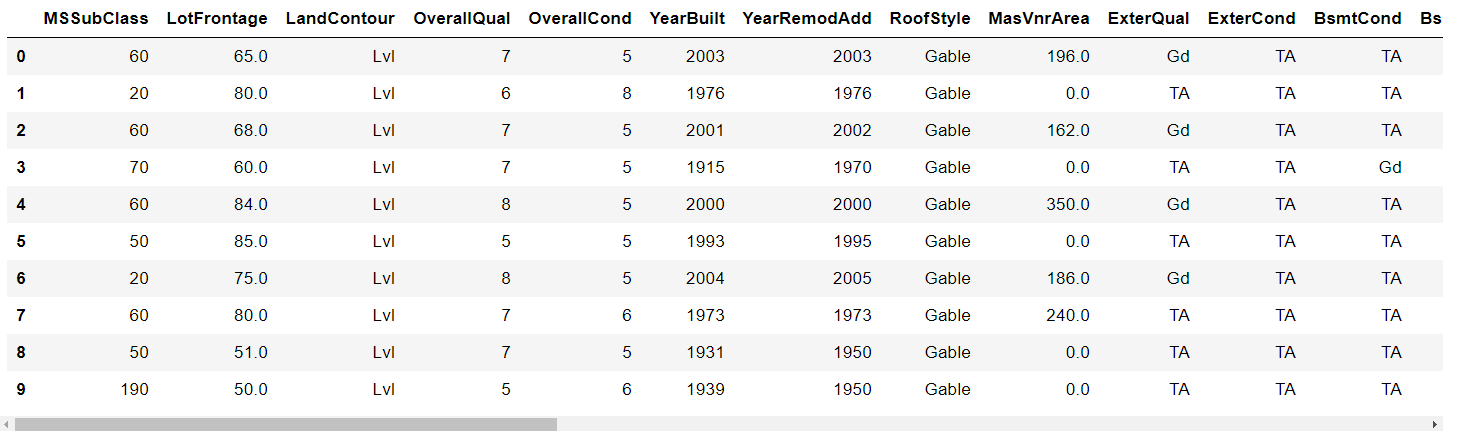
all\_data.drop(['MiscVal'], axis=1, inplace=True)

# 查看当前数据规模

print(all\_data.shape)



all\_data.head(10)



# 某些非数值特征具有明显的线性关系，可以用数值替换，避免不好处理的分类问题

# 如 ExterQual

# Ex Excellent

# Gd Good

# TA Average/Typical

# Fa Fair

# Po Poor

# 依次用5,4,3,2,1表示

all\_data.loc[all\_data['ExterQual']=='Ex','ExterQual']=5

all\_data.loc[all\_data['ExterQual']=='Gd','ExterQual']=4

all\_data.loc[all\_data['ExterQual']=='TA','ExterQual']=3

all\_data.loc[all\_data['ExterQual']=='Fa','ExterQual']=2

all\_data.loc[all\_data['ExterQual']=='Po','ExterQual']=1

# ExterCond

all\_data.loc[all\_data['ExterCond']=='Ex','ExterCond']=5

all\_data.loc[all\_data['ExterCond']=='Gd','ExterCond']=4

all\_data.loc[all\_data['ExterCond']=='TA','ExterCond']=3

all\_data.loc[all\_data['ExterCond']=='Fa','ExterCond']=2

all\_data.loc[all\_data['ExterCond']=='Po','ExterCond']=1

# BsmtCond

all\_data.loc[all\_data['BsmtCond']=='Ex','BsmtCond']=5

all\_data.loc[all\_data['BsmtCond']=='Gd','BsmtCond']=4

all\_data.loc[all\_data['BsmtCond']=='TA','BsmtCond']=3

all\_data.loc[all\_data['BsmtCond']=='Fa','BsmtCond']=2

all\_data.loc[all\_data['BsmtCond']=='Po','BsmtCond']=1

all\_data.loc[all\_data['BsmtCond']=='NA','BsmtCond']=0

# BsmtExposure

all\_data.loc[all\_data['BsmtExposure']=='Gd','BsmtExposure']=4

all\_data.loc[all\_data['BsmtExposure']=='Av','BsmtExposure']=3

all\_data.loc[all\_data['BsmtExposure']=='Mn','BsmtExposure']=2

all\_data.loc[all\_data['BsmtExposure']=='No','BsmtExposure']=1

all\_data.loc[all\_data['BsmtExposure']=='NA','BsmtExposure']=0

# HeatingQC

all\_data.loc[all\_data['ExterCond']=='Ex','ExterCond']=5

all\_data.loc[all\_data['ExterCond']=='Gd','ExterCond']=4

all\_data.loc[all\_data['ExterCond']=='TA','ExterCond']=3

all\_data.loc[all\_data['ExterCond']=='Fa','ExterCond']=2

all\_data.loc[all\_data['ExterCond']=='Po','ExterCond']=1

# CentralAir

all\_data.loc[all\_data['CentralAir']=='Y','CentralAir']=1

all\_data.loc[all\_data['CentralAir']=='N','CentralAir']=0

# GarageQual

all\_data.loc[all\_data['GarageQual']=='Ex','GarageQual']=5

all\_data.loc[all\_data['GarageQual']=='Gd','GarageQual']=4

all\_data.loc[all\_data['GarageQual']=='TA','GarageQual']=3

all\_data.loc[all\_data['GarageQual']=='Fa','GarageQual']=2

all\_data.loc[all\_data['GarageQual']=='Po','GarageQual']=1

all\_data.loc[all\_data['GarageQual']=='NA','GarageQual']=0

# MiscFeature

all\_data.loc[all\_data['MiscFeature']!='NA','MiscFeature']=1

all\_data.loc[all\_data['MiscFeature']=='NA','MiscFeature']=0

# HeatingQC

all\_data.loc[all\_data['HeatingQC']=='Ex','HeatingQC']=5

all\_data.loc[all\_data['HeatingQC']=='Gd','HeatingQC']=4

all\_data.loc[all\_data['HeatingQC']=='TA','HeatingQC']=3

all\_data.loc[all\_data['HeatingQC']=='Fa','HeatingQC']=2

all\_data.loc[all\_data['HeatingQC']=='Po','HeatingQC']=1

# 对于没有明显关系的变量采用one-hot编码（LandContour、RoofStyle、GarageFinish、SaleCondition）

# LandContour

all\_data.loc[all\_data['LandContour']=='Lvl','LandContour\_Lvl']=1

all\_data.loc[all\_data['LandContour']=='Bnk','LandContour\_Bnk']=1

all\_data.loc[all\_data['LandContour']=='HLS','LandContour\_HLS']=1

all\_data.loc[all\_data['LandContour']=='Low','LandContour\_Low']=1

for col in ('LandContour\_Lvl', 'LandContour\_Bnk', 'LandContour\_HLS','LandContour\_Low'):

all\_data[col] = all\_data[col].fillna(0)

# RoofStyle

all\_data.loc[all\_data['RoofStyle']=='Flat','RoofStyle\_Flat']=1

all\_data.loc[all\_data['RoofStyle']=='Gable','RoofStyle\_Gable']=1

all\_data.loc[all\_data['RoofStyle']=='Gambrel','RoofStyle\_Gambrel']=1

all\_data.loc[all\_data['RoofStyle']=='Hip','RoofStyle\_Hip']=1

all\_data.loc[all\_data['RoofStyle']=='Mansard','RoofStyle\_Mansard']=1

all\_data.loc[all\_data['RoofStyle']=='Shed','RoofStyle\_Shed']=1

for col in ('RoofStyle\_Flat','RoofStyle\_Gable','RoofStyle\_Gambrel','RoofStyle\_Hip','RoofStyle\_Mansard','RoofStyle\_Shed'):

all\_data[col] = all\_data[col].fillna(0)

# GarageFinish

all\_data.loc[all\_data['GarageFinish']=='Fin','GarageFinish\_Fin']=1

all\_data.loc[all\_data['GarageFinish']=='RFn','GarageFinish\_RFn']=1

all\_data.loc[all\_data['GarageFinish']=='Unf','GarageFinish\_Unf']=1

all\_data.loc[all\_data['GarageFinish']=='NA','GarageFinish\_NA']=1

for col in ('GarageFinish\_Fin','GarageFinish\_RFn','GarageFinish\_Unf','GarageFinish\_NA'):

all\_data[col] = all\_data[col].fillna(0)

# SaleCondition

all\_data.loc[all\_data['SaleCondition']=='Normal','SaleCondition\_Normal']=1

all\_data.loc[all\_data['SaleCondition']=='Abnorml','SaleCondition\_Abnorml']=1

all\_data.loc[all\_data['SaleCondition']=='AdjLand','SaleCondition\_AdjLand']=1

all\_data.loc[all\_data['SaleCondition']=='Alloca','SaleCondition\_Alloca']=1

all\_data.loc[all\_data['SaleCondition']=='Family','SaleCondition\_Family']=1

all\_data.loc[all\_data['SaleCondition']=='Partial','SaleCondition\_Partial']=1

for col in ('SaleCondition\_Normal','SaleCondition\_Abnorml','SaleCondition\_AdjLand','SaleCondition\_Alloca','SaleCondition\_Family','SaleCondition\_Partial'):

all\_data[col] = all\_data[col].fillna(0)

# 删除已被onehot编码的特征

all\_data.drop("LandContour", axis = 1, inplace = True)

all\_data.drop("RoofStyle", axis = 1, inplace = True)

all\_data.drop("GarageFinish", axis = 1, inplace = True)

all\_data.drop("SaleCondition", axis = 1, inplace = True)

# SalePrice放在最后一列，便于浏览

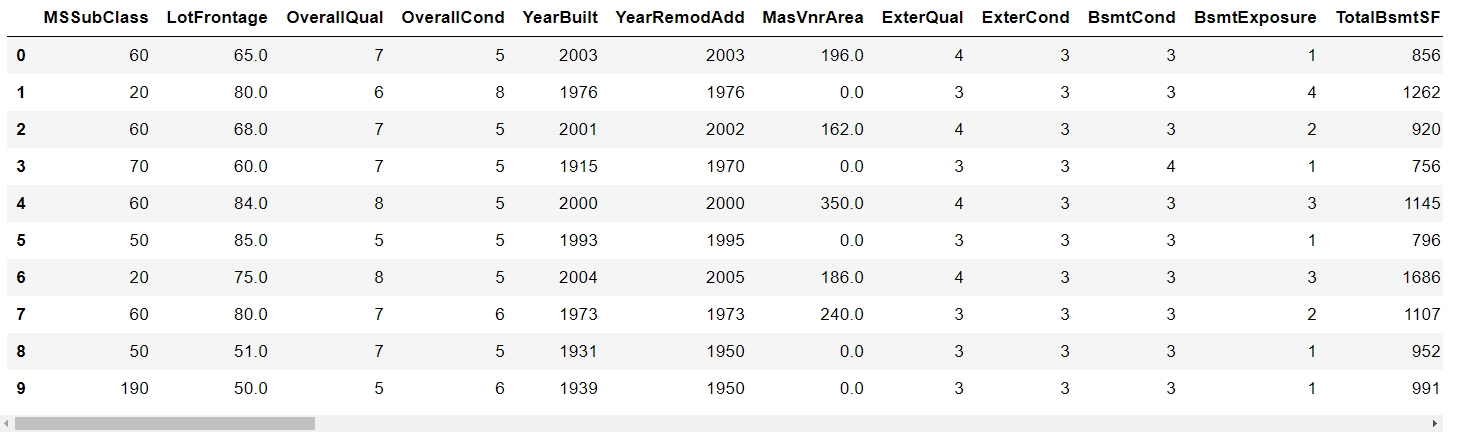
all\_data['Saleprice']=all\_data['SalePrice']

all\_data.drop("SalePrice", axis = 1, inplace = True)

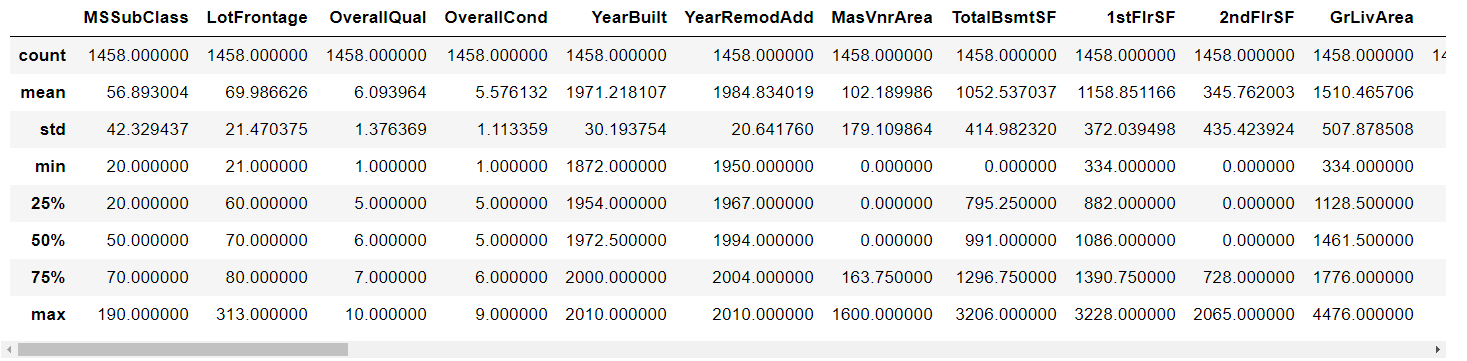
all\_data['SalePrice']=all\_data['Saleprice']

all\_data.drop("Saleprice", axis = 1, inplace = True)

all\_data.head(10)



all\_data.describe()



print(all\_data.shape)



# 因为采用岭回归而非纯粹线性回归，需对所有特征执行特征缩放

# x=x-平均值/标准差

for col in ('MSSubClass','LotFrontage','OverallQual','OverallCond','YearBuilt','YearRemodAdd','MasVnrArea','ExterQual','ExterCond','BsmtCond','BsmtExposure','HeatingQC','CentralAir','TotalBsmtSF','1stFlrSF','2ndFlrSF','GrLivArea','FullBath','HalfBath','BedroomAbvGr','KitchenAbvGr','Fireplaces','GarageYrBlt','GarageCars','GarageArea','GarageQual','MiscFeature','TotalSF','LandContour\_Lvl','LandContour\_Bnk','LandContour\_HLS','LandContour\_Low','RoofStyle\_Flat','RoofStyle\_Gable','RoofStyle\_Gambrel','RoofStyle\_Hip','RoofStyle\_Mansard','RoofStyle\_Shed','GarageFinish\_Fin','GarageFinish\_RFn','GarageFinish\_Unf','GarageFinish\_NA','SaleCondition\_Normal','SaleCondition\_Abnorml','SaleCondition\_AdjLand','SaleCondition\_Alloca','SaleCondition\_Family','SaleCondition\_Partial'):

all\_data[col] = (all\_data[col]-all\_data[col].mean())/all\_data[col].std()

# 线性回归模型拟合数据之前，首先要求数据应符合或近似符合正态分布

# 对因变量Saleprice也进行处理。首先分析Saleprice的分布规律

import matplotlib

import matplotlib.pyplot as plt

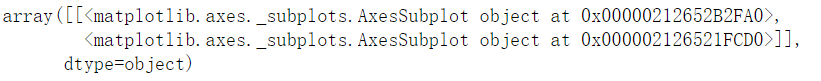
%config InlineBackend.figure\_format = 'retina' #set 'png' here when working on notebook

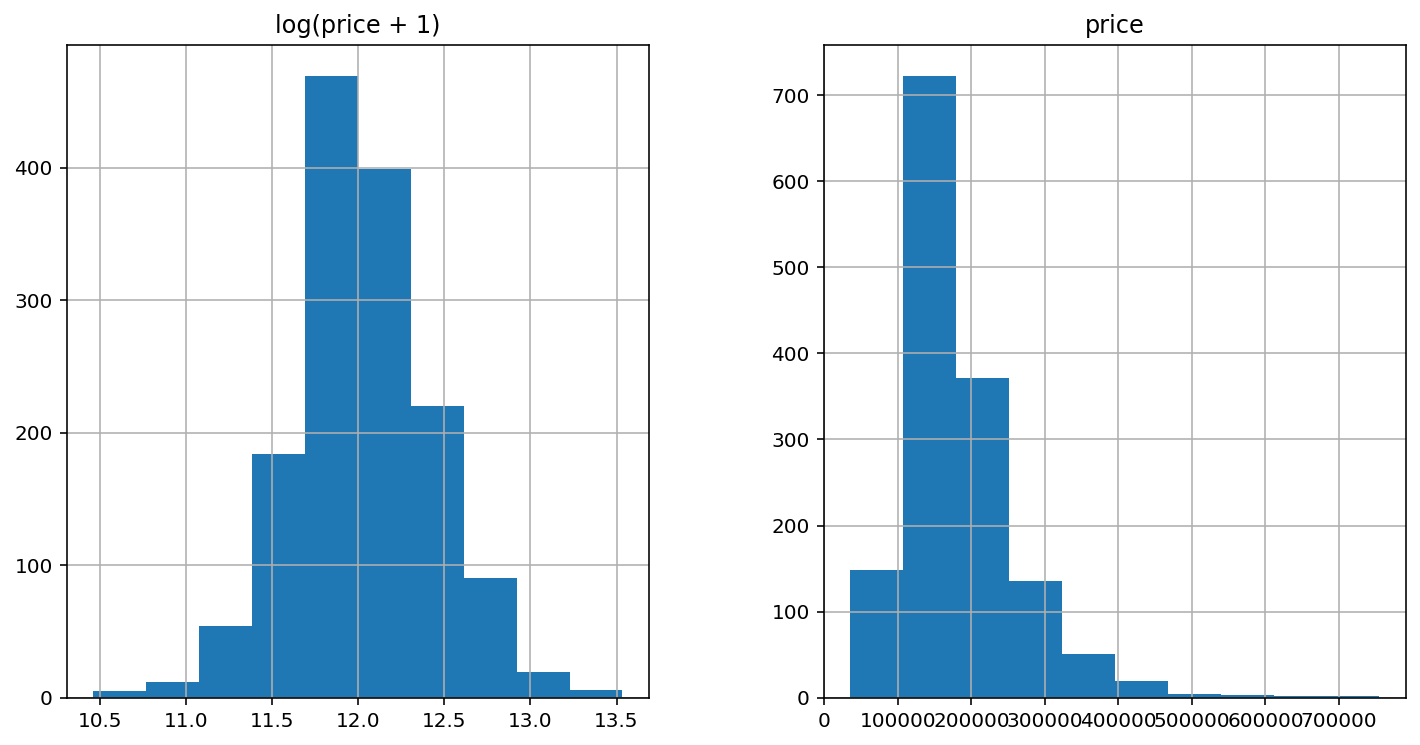
%matplotlib inline

matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)

prices = pd.DataFrame({"price":all\_data["SalePrice"], "log(price + 1)":np.log1p(all\_data["SalePrice"])})

prices.hist()

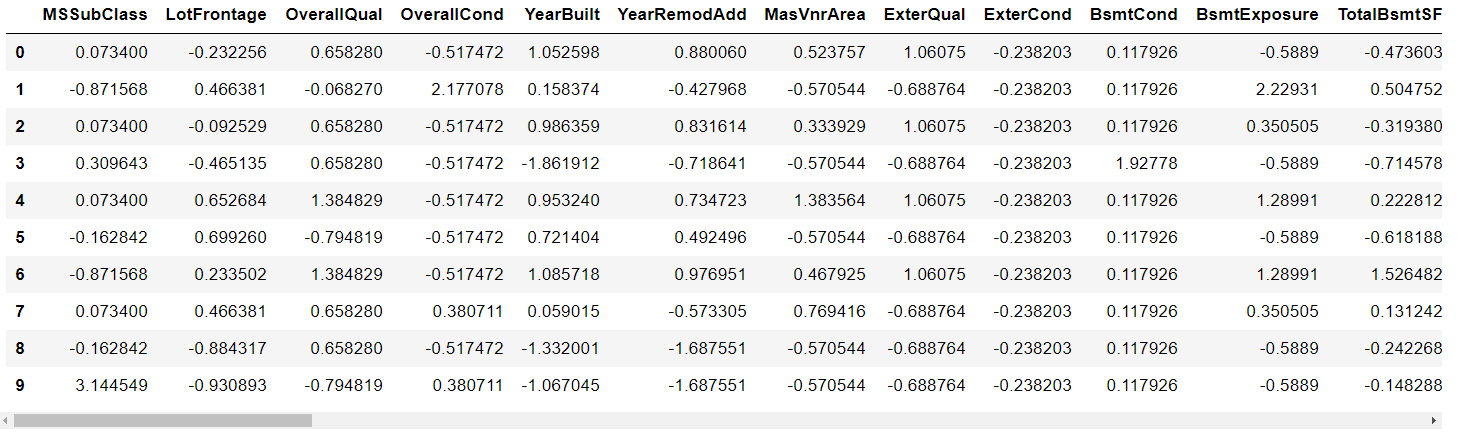




# 由图可以看出saleprice不满足正态分布，而log（price+1）（标准化）后更能满足正态分布

all\_data["SalePrice"] = np.log1p(all\_data["SalePrice"])

all\_data.head(10)



# 数据导出

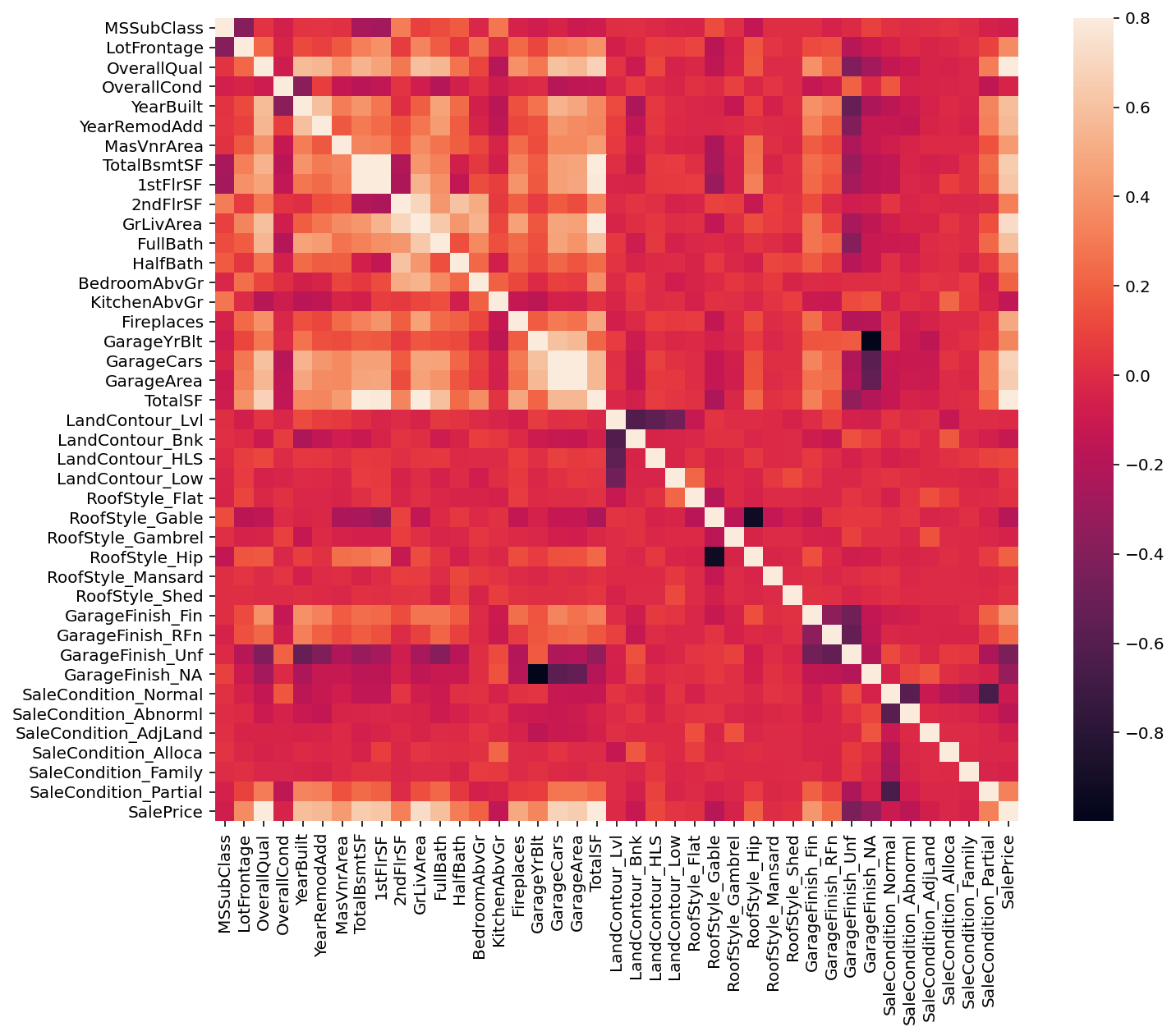
all\_data.to\_csv('train.csv', index=False)

# 建模后发现数据特征太多，效果不理想，进一步清洗。具体分析详见介绍部分。

corrmat = all\_data.corr()

f, ax = plt.subplots(figsize=(12, 9))

sns.heatmap(corrmat, vmax=.8, square=True);



# 重新按照原有步骤分析，再删除一些数据

all\_data.drop(['MSSubClass'], axis=1, inplace=True)

all\_data.drop(['1stFlrSF'], axis=1, inplace=True)

all\_data.drop(['2ndFlrSF'], axis=1, inplace=True)

all\_data.drop(['LandContour\_Lvl'], axis=1, inplace=True)

all\_data.drop(['LandContour\_Bnk'], axis=1, inplace=True)

all\_data.drop(['LandContour\_HLS'], axis=1, inplace=True)

all\_data.drop(['LandContour\_Low'], axis=1, inplace=True)

all\_data.drop(['RoofStyle\_Flat'], axis=1, inplace=True)

all\_data.drop(['RoofStyle\_Gable'], axis=1, inplace=True)

all\_data.drop(['RoofStyle\_Gambrel'], axis=1, inplace=True)

all\_data.drop(['RoofStyle\_Hip'], axis=1, inplace=True)

all\_data.drop(['RoofStyle\_Mansard'], axis=1, inplace=True)

all\_data.drop(['RoofStyle\_Shed'], axis=1, inplace=True)

all\_data.drop(['GarageFinish\_Fin'], axis=1, inplace=True)

all\_data.drop(['GarageFinish\_RFn'], axis=1, inplace=True)

all\_data.drop(['GarageFinish\_Unf'], axis=1, inplace=True)

all\_data.drop(['GarageFinish\_NA'], axis=1, inplace=True)

all\_data.drop(['SaleCondition\_Normal'], axis=1, inplace=True)

all\_data.drop(['SaleCondition\_Abnorml'], axis=1, inplace=True)

all\_data.drop(['SaleCondition\_AdjLand'], axis=1, inplace=True)

all\_data.drop(['SaleCondition\_Alloca'], axis=1, inplace=True)

all\_data.drop(['SaleCondition\_Family'], axis=1, inplace=True)

all\_data.drop(['SaleCondition\_Partial'], axis=1, inplace=True)

all\_data.drop(['BedroomAbvGr'], axis=1, inplace=True)

all\_data.drop(['HalfBath'], axis=1, inplace=True)

all\_data.drop(['Fireplaces'], axis=1, inplace=True)

all\_data.drop(['KitchenAbvGr'], axis=1, inplace=True)

all\_data.drop(['FullBath'], axis=1, inplace=True)

all\_data.drop(['BsmtExposure'], axis=1, inplace=True)

all\_data.drop(['ExterCond'], axis=1, inplace=True)

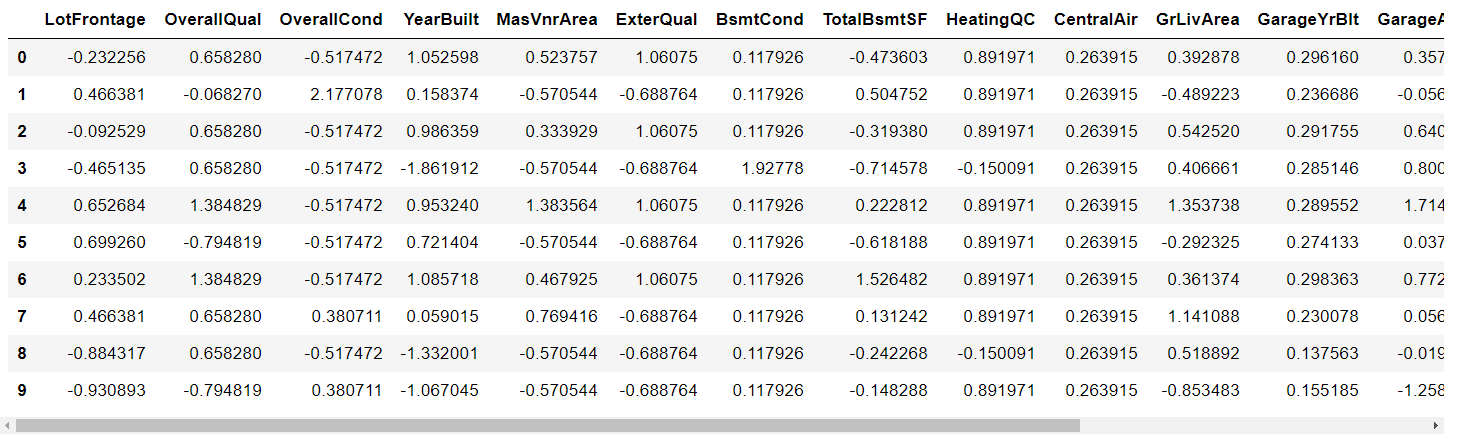
all\_data.drop(['YearRemodAdd'], axis=1, inplace=True)

all\_data.drop(['GarageCars'], axis=1, inplace=True)

print(all\_data.shape)



all\_data.head(10)



# 数据导出

all\_data.to\_csv('train.csv', index=False)

# 一共16个特征。建模实践表明结果可以接受。特征重建结束。

### 2.2 数据集划分

按照吴恩达课程第6周的推荐，数据集应划分为验证集、训练集与测试集，划分比例视数据集规模与需求而定，一般训练集占比至少在50%以上，每个数据集应严格不相交。但本项目中，处理后的数据集一共只包含1458个样本，对4特征法来说还好些，特征较多时严格划分3个数据集显得颇为不方便：测试集占比较少（如占比10%）时结果偶然性无疑会很大，验证集占比较少则难以调参且结果偶然性大。

介于此，我们在本项目采取了特殊的数据集划分策略，这种思路来自一名阿里程序员的回答（由于当时没有收藏，网址来源已几乎不可考）。

我们把所有的数据用于验证集，使用1次10折交叉训练的方法既完成模型调试也完成模型评估，最后再把所有的数据用作训练集，然后读取test文件，输出预测结果到文件中。

# 随机划分原始数据集为测试集约占90%、训练集约占10%。

def trainTestSplit(data, length, start, end):

df\_test=data[start \* length : end\*length]

df\_test\_index=list(df\_test.index)

# 都转换为list来判定成员资格

df\_test\_flag=data.index.isin(df\_test\_index)

diff\_flag = [not f for f in df\_test\_flag]

df\_train= data[diff\_flag]

X\_train = np.empty([len(df\_train), len(df\_train.values[0])])

X\_test = np.empty([len(df\_test), len(df\_test.values[0])])

X\_train[:, 0] = 1

X\_test[:, 0] = 1

for i in range(len(df\_train.values[0]) - 1):

X\_train[:, i + 1] = df\_train.values[:, i]

for i in range(len(df\_test.values[0]) - 1):

X\_test[:, i + 1] = df\_test.values[:, i]

y\_train = df\_train['SalePrice'].values

y\_test = df\_test['SalePrice'].values

return X\_train, y\_train, X\_test, y\_test

for i in range(10):

X\_train, y\_train, X\_test, y\_test = trainTestSplit(df\_train, 145, i, i + 1)

# 训练10个模型，即每次重新训练模型

theta = np.zeros(len(df\_train.values[0]))

theta = ridgeGradientDescent(X\_train, y\_train, theta, 2 \* 1e-1, 1000, 7)

下略。

### 2.3 模型搭建 & 训练

使用岭回归模型。

**J(θ) = ∑(y−Xθ)^2 + ∑λθ^2**

模型输出1次10折交叉验证的平均均方根对数误差（RMSLE）和还原为真实值后均方误差（MSE）代价函数值的平均值，并以此调参。

# mean squared error

def costFunction(theta, X, y):

error = np.exp(np.dot(X, theta)) - np.exp(y)

error = np.squeeze(error)

cost = np.sum(error \*\* 2) / (2 \* len(X))

return cost

def rootMeanSquaredLogarithmicError(y\_true, y\_pred):

return np.sqrt(np.mean(np.square(y\_pred - y\_true)))

# 随机划分原始数据集为测试集约占90%、训练集约占10%。

def trainTestSplit(data, length, start, end):

df\_test=data[start \* length : end\*length]

df\_test\_index=list(df\_test.index)

# 都转换为list来判定成员资格

df\_test\_flag=data.index.isin(df\_test\_index)

diff\_flag = [not f for f in df\_test\_flag]

df\_train= data[diff\_flag]

X\_train = np.empty([len(df\_train), len(df\_train.values[0])])

X\_test = np.empty([len(df\_test), len(df\_test.values[0])])

X\_train[:, 0] = 1

X\_test[:, 0] = 1

for i in range(len(df\_train.values[0]) - 1):

X\_train[:, i + 1] = df\_train.values[:, i]

for i in range(len(df\_test.values[0]) - 1):

X\_test[:, i + 1] = df\_test.values[:, i]

y\_train = df\_train['SalePrice'].values

y\_test = df\_test['SalePrice'].values

return X\_train, y\_train, X\_test, y\_test

# 岭回归

def ridgeGradientDescent(X, y, theta, alpha, iters, lambdas):

temp = np.zeros(theta.shape)

parameters = int(theta.shape[0])

for i in range(iters):

error = np.dot(X, theta) - y

error = np.squeeze(error)

for j in range(parameters):

X\_j = np.squeeze([x[j] for x in X])

if(j == 0):

temp[j] = theta[j] - (alpha / len(X)) \* np.sum(np.dot(error, X\_j))

else:

temp[j] = theta[j] - (alpha / len(X)) \* (np.sum(np.dot(error, X\_j)) + lambdas \* theta[j])

theta = temp

return theta

# 1次10折交叉验证调参/评估模型

cost = 0

rmsle = 0

for i in range(10):

X\_train, y\_train, X\_test, y\_test = trainTestSplit(df\_train, 145, i, i + 1)

# 训练10个模型，即每次重新训练模型

theta = np.zeros(len(df\_train.values[0]))

theta = ridgeGradientDescent(X\_train, y\_train, theta, 2 \* 1e-1, 1000, 7)

rmsle = rmsle + rootMeanSquaredLogarithmicError(y\_test, np.squeeze(np.dot(X\_test, theta)))

cost = cost + costFunction(theta, X\_test, y\_test)

predictions = pd.DataFrame({"real price":np.squeeze(np.exp(y\_test)), "linear prediction":np.squeeze(np.exp(np.dot(X\_test, theta)))})

predictions.plot(x = "real price", y = "linear prediction", kind = "scatter")

plt.plot(np.squeeze(np.exp(y\_test)), np.squeeze(np.exp(y\_test)), c = 'r', linewidth = 0.5)

plt.show()

cost = cost / 10

rmsle = rmsle / 10

print(cost)

print(rmsle)

模型搭建时，少数实现参考了助教在第一节助教课给出的线性回归模型。

真正重要的是模型的训练过程。除debug过程外，如何最高效、最合理地完成模型参数调整呢？以下为我们调试最终模型时总结的调参经验。

由于线性回归参数适宜取值区间较大，我们主要进行手工调参，这可能也有助于体会参数对模型的影响。

调参时，首先考虑α（梯度下降速度）与iters（梯度下降次数）的值，而排除*λ*的影响。网上有资料说*λ*在0.01附近大多数回归系数就趋于稳定，我们将*λ*直接设置为1e-8。

先不使用10折交叉验证，粗调α、iters。α最初应一个数量级（10倍）一个数量级地调，iters值参考网上一些其他项目线性回顾的经验先预设为1000。

在α调至合适数量级时，尝试调整iters，很快发现iters在大于300时增减对结果几乎无影响。这一点在4特征法时首先发现，后来也得到验证。

使用1次10折交叉验证法调参。比较仔细地调α值，发现当α取0.4-0.1，梯度下降次数约在100以上时，结果最好且RMSLE代价波动范围小于0.1%；设置iters = 300，取0.2时结果最好。

然后调lambda。由于使用了10折交叉验证法评估模型，所以只需选取最好的结果即可。当lambda = 7时结果最好。

设置iters = 10000，取得最终的1次10折交叉验证平均MSE(价格取了对数)与平均rmsle，结果与iters = 300时几乎没有差别。

然后改iters为1000，将价格还原为真实价格，进行可视化、模型评估、最终模型训练与test数据集结果输出等。下附用所有数据训练最终模型与test输出结果代码。

theta = np.zeros(len(df\_train.values[0]))

X = np.empty([len(df\_train), len(df\_train.values[0])])

X[:, 0] = 1

for i in range(len(df\_train.values[0]) - 1):

X[:, i + 1] = df\_train.values[:, i]

y = df\_train['SalePrice'].values

theta = ridgeGradientDescent(X, y, theta, 2 \* 1e-1, 50000, 1)

# test

df\_test = pd.read\_csv('test.csv')

y\_test = np.empty([len(df\_test)])

X\_test = np.empty([len(df\_test), len(df\_test.values[0])])

X\_test[:, 0] = 1

for i in range(len(df\_test.values[0]) - 1):

X\_test[:, i + 1] = df\_test.values[:, i + 1]

y\_test = np.squeeze(np.exp(np.dot(X\_test, theta)))

result = np.empty([len(df\_test), 2])

result[:, 0] = df\_test.values[:, 0]

result[:, 1] = y\_test

our\_submission = pd.DataFrame(result)

our\_submission.columns = ['Id', 'SalePrice']

our\_submission.to\_csv('our\_submission.csv', index = False)

### 2.4优化器优化

我们在使用4特征法师，就尝试使用opt自动而更高效地完成回归。

以下为最终模型的使用opt优化后的实验代码和结果输出。

def trainTestSplit(X, y, seed):

X\_train = []

y\_train = []

np.random.seed(seed)

for i in range(len(y)):

if np.random.randint(10) != seed:

X\_train.append(X[i, :])

y\_train.append(y[i])

X\_train = np.array(X\_train)

y\_train = np.array(y\_train)

return X\_train, y\_train

def ridgeGradientDescent(theta, X, y):

alpha = 0.5

lambdas = 0.01

iters = 100

temp = np.zeros(theta.shape)

parameters = int(theta.shape[0])

for i in range(iters):

error = np.dot(X, theta) - y

error = np.squeeze(error)

for j in range(parameters):

X\_j = np.squeeze([x[j] for x in X])

if(j == 0):

temp[j] = theta[j] - (alpha / len(X)) \* np.sum(np.dot(error, X\_j))

else:

temp[j] = theta[j] - (alpha / len(X)) \* (np.sum(np.dot(error, X\_j)) + lambdas \* theta[j])

theta = temp

return theta

def costFunction(theta, X, y):

error = np.dot(X, theta) - y

error = np.squeeze(error)

cost = np.sum(error\*\*2)/(2 \* len(X))

return cost

theta = np.zeros(len(df\_train.values[0]))

X = np.empty([len(df\_train), len(df\_train.values[0])])

X[:, 0] = 1

for i in range(len(df\_train.values[0]) - 1):

X[:, i + 1] = df\_train.values[:, i]

y = df\_train['SalePrice'].values

#for i in range(10):

#X\_train, y\_train = trainTestSplit(X, y, i)

#theta = ridgeGradientDescent(theta, X, y)

temp = opt.fmin\_tnc(func = costFunction, x0 = theta.flatten(), fprime = ridgeGradientDescent, args = (X, y.flatten()))

theta = temp[0]

cost = costFunction(theta, X, y)

print(cost)



然而，opt.fmin\_tnc优化函数的参数条件较为苛刻，以上代码最终会导致theta被赋值为0。

这一点我们在使用其他数据集时就发现了。使用4特征法时，若不使用opt.fmin\_tnc函数，则α取1e-4至1e-6时结果差不多，测试集代价函数结果波动范围小于0.1%，但使用优化函数后，只有α取le-6时theta才不会被赋值为零。

这是因为，opt.fmin\_tnc函数使用牛顿截断法，当梯度下降改变的大小不达预期时会被截断，故需要较严格的模型参数来使之工作。可能还有其他一些我们尚不了解的原因。

我们尝试了其他利用这个函数的方法，比如先进行几百次梯度下降再使用该函数，这些方法结果都不理想。

从4特征法的实践经验来看，opt.fmin\_tnc函数对结果的优化效果较小，其使用条件又太过苛刻，不使用这个函数的时间代价也完全能接受，故我们最终选择最终模型里不使用该函数。

下附4特征法使用opt函数的代码与运行结果。

import scipy.optimize as opt

def trainTestSplit(X, y, seed):

X\_train = []

y\_train = []

np.random.seed(seed)

for i in range(len(y)):

if np.random.randint(10) != seed:

X\_train.append(X[i, :])

y\_train.append(y[i])

X\_train = np.array(X\_train)

y\_train = np.array(y\_train)

return X\_train, y\_train

def ridgeGradientDescent(theta, X, y):

alpha = 0.000001

lambdas = 0.01

iters = 10

temp = np.zeros(theta.shape)

parameters = int(theta.shape[0])

for i in range(iters):

error = np.dot(X, theta) - y

error = np.squeeze(error)

for j in range(parameters):

X\_j = np.squeeze([x[j] for x in X])

if(j == 0):

temp[j] = theta[j] - (alpha / len(X)) \* np.sum(np.dot(error, X\_j))

else:

temp[j] = theta[j] - (alpha / len(X)) \* (np.sum(np.dot(error, X\_j)) + lambdas \* theta[j])

theta = temp

return theta

def costFunction(theta, X, y):

error = np.dot(X, theta) - y

error = np.squeeze(error)

cost = np.sum(error\*\*2)/(2 \* len(X))

return cost

theta = np.zeros(5)

X = np.empty([len(df\_train),5])

X[:, 0] = 1

X[:, 1] = df\_train['OverallQual'].values

X[:, 2] = df\_train['YearBuilt'].values

X[:, 3] = df\_train['TotalBsmtSF'].values

X[:, 4] = df\_train['GrLivArea'].values

y = df\_train['SalePrice'].values

for i in range(10):

X\_train, y\_train = trainTestSplit(X, y, i)

temp = opt.fmin\_tnc(func = costFunction, x0 = theta.flatten(), fprime = ridgeGradientDescent, args = (X\_train, y\_train.flatten()))

theta = temp[0]

cost = costFunction(theta, X, y)

print(cost)



对比一下，不使用opt函数的结果。

def costFunction(theta, X, y):

error = np.dot(X, theta) - y

error = np.squeeze(error)

cost = np.sum(error\*\*2)/(2 \* len(X))

return cost

def gradientDescent(X, y, theta, alpha, iters):

temp = np.zeros(theta.shape)

parameters = int(theta.shape[0])

cost = np.zeros(iters)

for i in range(iters):

error = np.dot(X, theta) - y

error = np.squeeze(error)

for j in range(parameters):

X\_j = np.squeeze(X[:, j])

temp[j] = theta[j] - (alpha / len(X)) \* np.sum(np.dot(error, X\_j))

theta = temp

cost = np.sum(error\*\*2)/(2 \* len(X))

return theta, cost

theta = np.zeros([5, 1])

X = np.empty([len(df\_train), 5])

X[:, 0] = 1

X[:, 1] = df\_train['OverallQual'].values

X[:, 2] = df\_train['YearBuilt'].values

X[:, 3] = df\_train['TotalBsmtSF'].values

X[:, 4] = df\_train['GrLivArea'].values

y = df\_train['SalePrice'].values

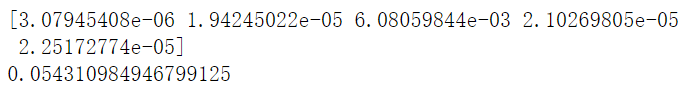
theta, cost = gradientDescent(X, y, theta, 1e-12, 1000)

theta = np.squeeze(theta)

cost = costFunction(theta, X, y)

print(theta)

print(cost)



差距很小，运行速度快了一些。

## 3 实验性能

为了和Kaggle上的结果比较，我们计算了均方根对数误差（RMSLE，但不完全严格，因为取对数前没有加一），同时按照文档要求，也计算还原为真实值后的均方误差（MSE）代价函数值。

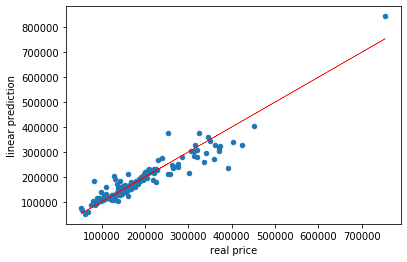
下为1次10折交叉验证输出的均方误差代价函数值的平均值与RMSLE的平均值。

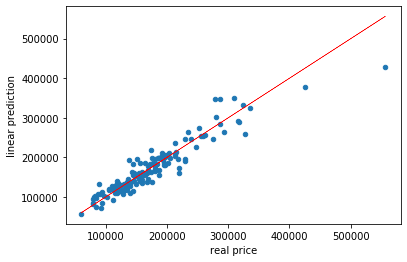
代码见上2.4部分。



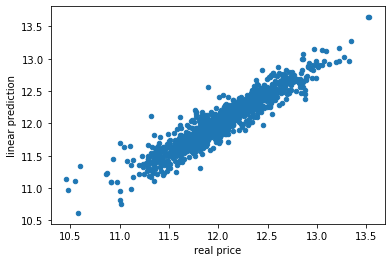
当增大iters到10000时，RMSLE的平均值会降到0.136左右。

在1次10折交叉验证的过程中，每次重新训练模型并验证时，我们也对验证结果进行了可视化输出，即一共10张图，详见提交的代码文件。以下为两张典型结果。





训练时，我们曾可视化未还原为真实价格的结果。



附代码：

# 可视化完成于还原价格之前

predictions = pd.DataFrame({"real price":y, "linear prediction":np.squeeze(np.dot(X, theta))})

predictions.plot(x = "real price", y = "linear prediction", kind = "scatter")

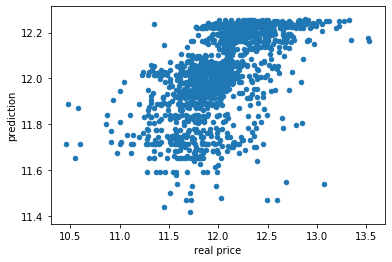
从以上图与数据，我们分析出以下几点：

1. 因为我们的模型使用取对数的saleprice进行训练，对于价格较高的个别长尾样本表现不好，偏差在MSE里被放大了，RMSLE评估可能会更为合适，这也是Kaggle使用的评估方式。
2. 从Kaggle上的提交数据来看，0.136是还算可以接受的岭回归RMSLE结果，尤其是考虑到很多模型使用了几十甚至上百个特征，而我们只使用了16个。有Kaggle共享文档指出，结果排在前0.3%的算法中用到的岭回归模型，训练集RMSLE也高于0.11。

我们还可以对比4特征法得到的结果。

当价格取对数时，得到的代价函数值为0.00947231528367319，而上面提到，4特征法得到的结果为0.054310984946799125，差距较大。

4特征法得到的预测结果图为：



附代码：

predictions = pd.DataFrame({"real price":y, "prediction":np.squeeze(np.dot(X, theta))})

predictions.plot(x = "real price", y = "prediction", kind = "scatter")

这个图是没有还原价格的结果，和同样没有还原价格的最终模型结果图相比差距显著。

### 4 讨论 & 结论

实验中所得出的结论，在上文中基本都已给出，第一部分所提到的问题也已被回答，考虑到回答过于琐碎，这里不再重复。

总结来说，我们使用较少的特征数量、较少的调参次数、较少的执行次数，通过反复迭代算法、迭代特征，达成了可以接受的算法效率。我们以团队合作的形式，实践了比较完整的机器学习项目流程，总结出一系列经验（上文已给出），还手写了10折交叉验证等算法。实际上连这个算法也经过了好几次的迭代，最初并没有使用交叉验证，然后用随机函数实现了交叉验证；模型最初也不是每次都重新训练，而是在已有θ基础上继续训练，结果比重新训练出来的结果要好，但却是错误的。

过程中，有的事令人印象深刻。我们团队精诚合作，主动互帮互助，由一而终的过程无疑是一件。在最初的4特征法实践中，我们曾模仿助教上的第1节课给的代码，发现最后给出的cost结果较大，而且使用opt函数时结果怎么也不如人意；在开始用这段代码的十几个小时以后，我们才发现其中有个致命的小错误，很不起眼，但对python初学者可能很困难，真是让人啼笑皆非！

实验也留下了不少遗憾，一些问题我们暂时没有能力、时间探究到底，比如到底怎样提取特征会更好、如何简单地解决数据溢出问题、opt函数是否有更好的使用方法、opt函数的具体原理是怎样的，等等。

本次实验要感谢金老师的耐心指导，在一节节下课后耐心回答我们的问题，让我们了解要做些什么。也要感谢助教们的帮助，包括无论何时都会耐心回答问题的魏心玮学姐、提供了我们建议的希浩宇学长等。

### 5 参考文献

数据清洗部分，我们部分参考了Kaggle共享文档《Stacked Regressions to predict House Prices》等。模型实现时，我们部分参考了助教上的第1节课给的study文件。我们还查阅了大量网上的资料，许多网址已不可考。吴恩达课程第6周的内容也给我们启发。

另外，还有以下Kaggle共享文档可能给了我们一些启发，包括（排名不分先后和参考顺序）：

*comprehensive data exploration with python*

*house prices lasso xgboost and a detailed eda*

*how I made top 0.3% on a Kaggle competition*

*regularized linear models*

*stacked regressions top 4% on leaderboard*