

**2024年《机器学习》**

**工程报告**

**（完整版）**



**课 程：**  **机器学习**

**姓名+学号：陈嘉乐（2021218152）**

**完成时间： 2024.5.12**

一． 工程摘要与每人贡献

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 角色  （组长，组员，独自完成） | 工作量  比例 | 负责内容 |
| 陈嘉乐 | 独自完成 | 100% | 完成工程报告（完整版）的撰写和  华为实验的相关复现 |

二、研究背景与意义

本工程的目的是依据房屋的属性信息，包括房屋的卧室数量，卫生间数量，房屋的大小，房屋地下室的大小，房屋的外观，房屋的评分，房屋的修建时间，房屋的翻修时间，房屋的位置信息等，对房屋的价格进行预测，即根据数据预测房价；这本质上是一个回归问题。研究意义在于可以为此类价格类的实际问题的处理提供技术参考。

研究背景在于机器学习服务提供可视化的操作界面来编排机器学习模型的训练、评估和预测过程，无缝衔接数据分析和预测应用，降低机器学习模型的生命周期管理难度，为用户的数据挖掘分析业务提供易用、高效、高性能的平台服务。

主要内容是如何针对实际问题进行建模分析，包括了解数据的分布特征，根据实际项目理解每列数据的意义，同时采用多种方法进行模型对比，最终选择梯度提升树进行模型训练，然后进行房价的预测。采用所学的特征工程、模型选择和模型参数优化等方面的知识进行最优模型的寻找，获取回归模型。

三、模型方法

**一、线性回归模型**

获得房价数据集进行模型的训练和预测，房价数据集中，和房屋相关的值共有14个：前13个用来描述房屋相关的各种信息，即模型中的 *x i* ；最后一个值为我们要预测的该类房屋价格的中位数，即模型中的 *y i* 。因此，我们的模型就可以表示成：

***Y ^=ω1​X1​+ω2​X2​+…+ω13​X13​+b***

*Y* ^表示模型的预测结果，用来和真实值Y区分。模型要学习的参数即：ω 1 , … , ω 13 , 建立模型后，我们需要给模型一个优化目标，使得学到的参数能够让预测值Y^ 尽可能地接近真实值Y。对于线性回归模型来讲，最常见的损失函数就是均方误差（Mean Squared Error， MSE）了，它的形式是：

***MSE=n1​i=1∑n​(Yi​^​−Yi​)2***

即对于一个大小为*n*的测试集，*MSE*是***n***个数据预测结果误差平方的均值。对损失函数进行优化所采用的方法一般为梯度下降法。梯度下降法是一种一阶最优化算法。如果*f(x)*在点*Xn*有定义且可微，则认为*f(x)*在点*Xn*沿着梯度的负方向***− ▽ f ( x n )***下降的是最快的。反复调节*x* ，使得*f(x)*接近最小值或者极小值，调节的方式为：

***xn​+1=xn​−λ▽f(x),n≧0***

其中***λ***代表学习率。这种调节的方法称为梯度下降法。

1. **随机森林模型**

随机森林通过自助法(bootstrap)重采样技术，从原始训练样本集N中有放回地重复随机抽取k个样本(k一般和N相同)生成新的训练样本集，然后根据自助样本集生成n个分类树组成随机森林。其实质是对决策树算法的一种改进，将多个决策树合并在一起，每棵树的建立依赖于一个独立抽取的样本集。

特征选择采用随机方法，然后比较不同情况下产生的误差，能检测到内在估计误差、分类能力和相关性决定选择特征的数目。单棵树分类能力可能很小，但在随机产生大量决策树后，一个测试样品可通过每一棵树的分类结果经统计后选择最可能的分类

输入样本都不是全部样本，相对不容易出现over-fitting。然后进行列采样从M个feature中选择m个(m << M)。之后是对采样后的数据使用完全分裂方式建立出决策树，这样决策树某一叶子节点要么是无法继续分裂的，要么里面所有样本都是指向的同一个分类。一般很多决策树算法都一个重要步骤—剪枝，但这里不这样干，由于两个随机采样过程保证了随机性，所以就算不剪枝，也不会出现over-fitting

从原始训练数据集中，应用bootstrap方法有放回地随机抽取k个新自助样本集，并由此构建k棵决策树，每次未被抽到的样本组成Ｋ个袋外数据。2.设有n 个特征，在每一棵树的每个节点处随机抽取mtry 个特征，通过计算每个特征的蕴含信息量，在特征中选择一个最具分类能力的特征进行节点分裂。3.每棵树最大限度生长不做任何剪裁。4.将生成的多棵树组成随机森林，用随机森林对新数据进行分类，分类结果按树分类器投票多少而定

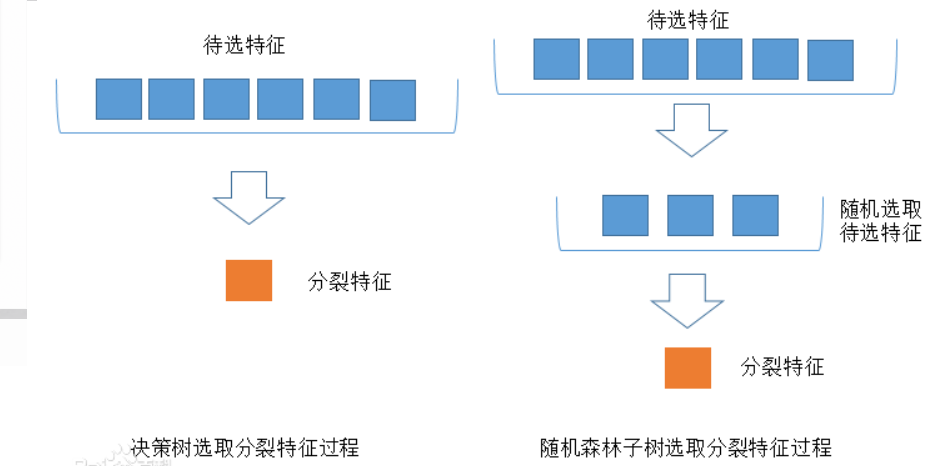
1、数据随机选取

首先从原始数据集中采取有放回抽样，构造子数据集，子数据集数据量是与原始数据集相同的，不同子数据集的元素可以重复，同一子数据集中的元素也可重复。第二利用子数据集构建子决策树，将这个数据放到子决策树中，每个子决策树输出一个结果。最后如果有新数据需要通过随机森林得到分类结果，就可通过对子决策树的判断结果投票，得到随机森林输出结果。假设随机森林中有3棵子决策树，2棵子树分类结果是A类，1棵子树分类结果是B类，那么随机森林的分类结果就是A类

2、待选特征随机选取

与数据集随机选取类似，随机森林子决策树的每一分裂过程(即每一枝节点处)并未用到所有待选特征，而是从所有待选特征中随机选取一定数量特征(三分之一，值越小模型越健壮，无放回抽取)，之后再在随机选取特征中选取最优特征。这样能使随机森林中的决策树都能彼此不同，提升系统多样性从而提升分类性能。

下图蓝色方块代表所有可被选择的特征，也就是目前待选特征。黄色方块是分裂特征。左边是一棵决策树的特征选取过程，通过在待选特征中选取最优分裂特征完成分裂。右边是一颗随机森林子树的特征选取过程



分枝优度准则是基于离均差平方和，假设有p个自变量X=(X1,X2,…,Xp)和连续型因变量Y。对于树某一节点t的样本量为N(t)，可计算该节点的离均差平方和。假定该阶段t内所有可能的分枝集合(含变量和相应切点)为A，分枝s将节点t分裂为两个子节点tl与tr，最佳分枝即使t节点离均差平方和与分裂后两个子节点对应离均差平方和之和差距最大的分枝，即分裂后效果优于分裂前，使得各子节点内的变异最小。将生成的b颗回归树组成随机森林回归模型，回归效果评价采用袋外数据预测均方残差MSE及拟合优度 R2。

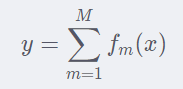
3、重要性评分

定义为袋外数据自变量值发生轻微扰动后的分类正确率与扰动前分类正确率的平均减少量，OOB误差。

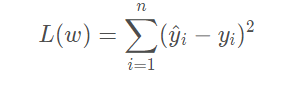
**三、GBDT模型**

1. GBDT 的损失函数：

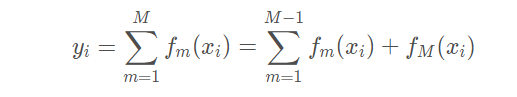
Boosting 的模型是个迭代多轮的加法模型。Boosting 每轮迭代输出的是一个基模型 f m ( x ) f m(x)，其中m 表示第 m 轮迭代。最终，每轮迭代输出的模型经过加法求和，就得到了最终Boosting 模型的输出，也就是：



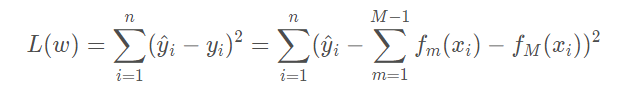
GBDT 模型的基模型为DT（决策树），即对于GBDT 下的每轮迭代，输出的f m(x) 为决策树。而最终的GBDT 模型为所有决策树输出结果之和。Boosting 的基模型采用的都是弱模型，因此，通常GBDT 基模型的决策树树深不会太深（一般少于 5 55 层），这些可以在实际实现中灵活处理。回归决策树的 损失函数 采用平方误差，GBDT 也可以保持一致。平方误差的公式为：



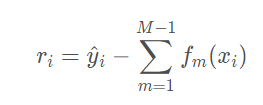
此处的模型和损失函数都是参数 w的函数，参数 w为每个基模型的每个分裂特征和每个分裂阈值。考虑此时要最优化的目标变量是什么，此处可以从损失函数入手，对于损失函数中的预测值 y i有：



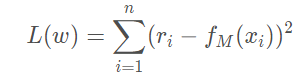
两式计算得



上式发现损失函数中有 3 33 项。第一项是真实值，第二项是截止到第M-1棵树的预测值，第三项是第 M棵树的输出结果，若令：



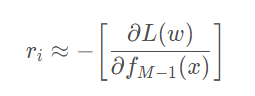
表示的是，在训练第 M M*M* 棵树时，截止到当前的残差值（误差），有：



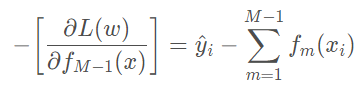
该公式表明，为了让损失函数数值最小，在训练某个基模型时，其训练目标是去拟合残差 ri。

1. GBDT 的最优化求解

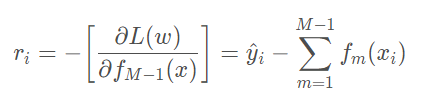
上述讨论是基于损失函数为平方误差的情况。一般地，如果损失函数不是平方误差，则每个基模型的训练目标就不是残差。GBDT 利用损失函数的负梯度在当前模型的值作为残差的近似值，即：



特别地，当损失函数采用平方误差时，损失函数的负梯度就是先前推导的残差：



假设有一个回归问题，特征只有一个维度，取值范围为1～ 5的自然数，样本对应的真实值取值为− 1～1的自然数。现利用GBDT 算法建立模型，假设基模型采用树深为1的 CART 回归树算法，损失函数采用平方误差函数，则公式为：



建模通过多轮迭代，每一轮都学习一棵CART 树，学习的目标是残差。对于只有 1层结点的CART 树建模，采用选择平方误差最小的分裂点和阈值即可。

**四、KNN模型**

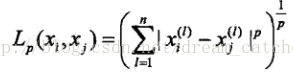
1.算法流程：

（1）根据给定的距离度量，在训练集中找出与样本x最近的k点

（2）在这k点中，根据分类决策原则（比如多数表决），决定x的类别y

2.KNN三大基本要素

（1）距离度量：特征空间中2个实例点的距离是2个实例点相似程度的反映



（2）K值的选择：K值较小使模型复杂化容易过拟合，K值较大使模型简单化容易欠拟合

（3）分类决策规则：一般是多数表决

四、系统设计

1. **数据查看**
   1. 数据准备

步骤 1离线数据读取

这里读取的数据是与项目文件同级目录下，或同一个文件夹中，下面是本实验的读取文件目录代码

df = pd.read\_csv("./data/kc\_house\_data.csv")

步骤 2 输出前5行数据

print(df.head())

输出如下结果：

id date ... sqft\_living15 sqft\_lot15

0 7129300520 20141013T000000 ... 1340 5650

1 6414100192 20141209T000000 ... 1690 7639

2 5631500400 20150225T000000 ... 2720 8062

3 2487200875 20141209T000000 ... 1360 5000

4 1954400510 20150218T000000 ... 1800 7503

从上述输出结果中可以查看数据的前5行信息，包括id，时间，大小，楼层，住宅面积等基本房屋信息。

* 1. 数据理解

步骤 1 输出属性信息

print(df.sample(5))

print(df.info())

输出结果如下：

RangeIndex: 21613 entries, 0 to 21612

Data columns (total 21 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 id 21613 non-null int64

1 date 21613 non-null object

2 price 21613 non-null float64

3 bedrooms 21613 non-null int64

4 bathrooms 21613 non-null float64

5 sqft\_living 21613 non-null int64

6 sqft\_lot 21613 non-null int64

7 floors 21613 non-null float64

8 waterfront 21613 non-null int64

9 view 21613 non-null int64

10 condition 21613 non-null int64

11 grade 21613 non-null int64

12 sqft\_above 21613 non-null int64

13 sqft\_basement 21613 non-null int64

14 yr\_built 21613 non-null int64

15 yr\_renovated 21613 non-null int64

16 zipcode 21613 non-null int64

17 lat 21613 non-null float64

18 long 21613 non-null float64

19 sqft\_living15 21613 non-null int64

20 sqft\_lot15 21613 non-null int64

dtypes: float64(5), int64(15), object(1)

memory usage: 3.5+ MB

None

上述属性描述信息可以看出，所有的属性都是数值型的，记录数，和空值情况，上述信息显示所有属性都不存在空值的情况。

但是其中的date 数据属于object类型的数据，对后面的数据训练和测试会造成影响，在初次训练的时候我们发现无法从object类型转换成float类型。所以在这里采取将date数据删除或者转换成日期数据集的日期标注，在这里我们将其转换成了数据集的日期标注。

df['date'] = pd.to\_datetime(df['date'])

df = df.set\_index('date')

由此可以来划分数据进行训练和测试。

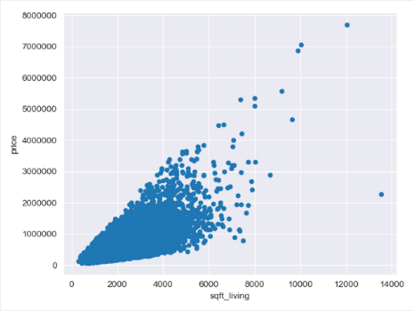
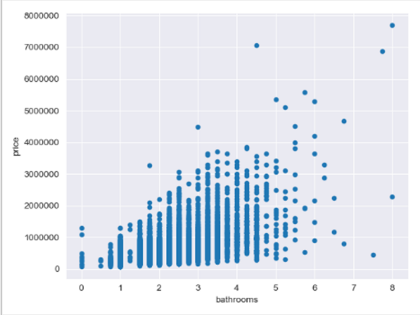
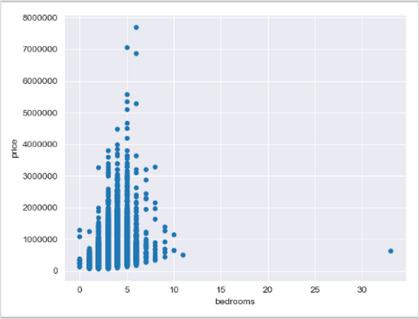
1. **属性分析**
   1. 房价属性分布规律

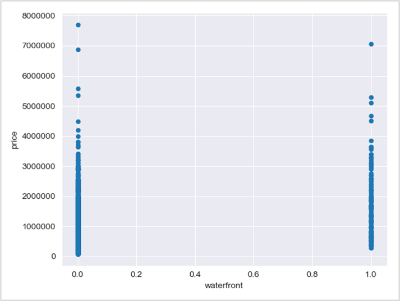
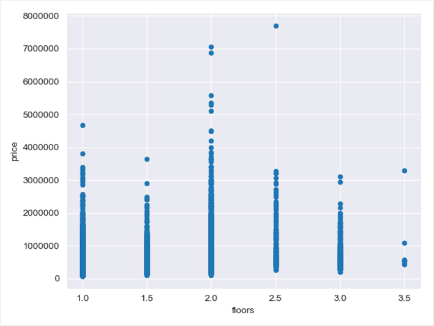
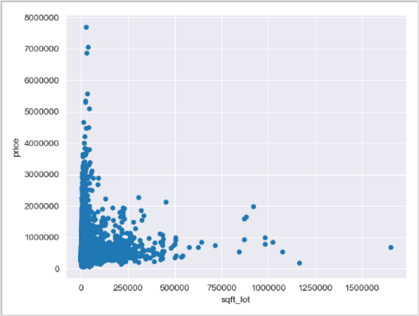
查看属性与房价之间的分布规律，用于探索单一属性与房价的变化规律，明确房价的决定因素有哪些，或哪些属性对房价有明确的营销。

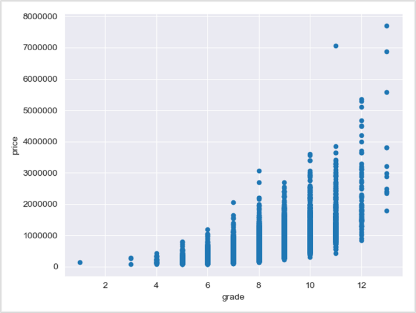
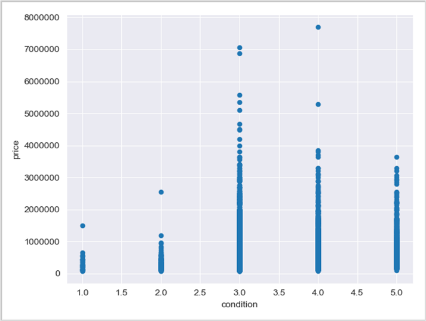
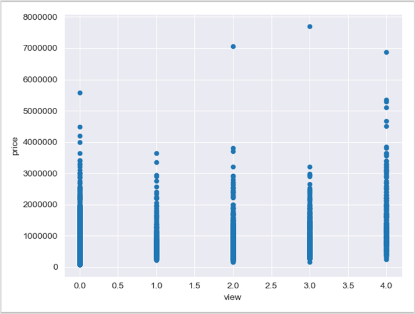
x\_vars=df.columns[3:]

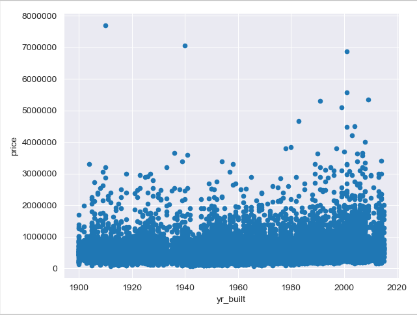
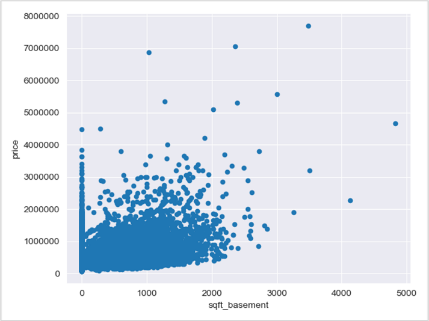
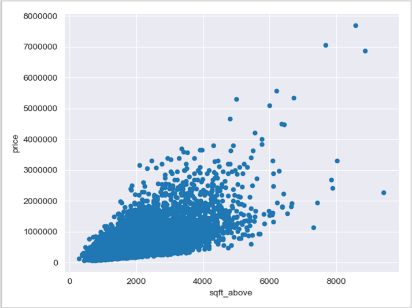
for x\_var in x\_vars:

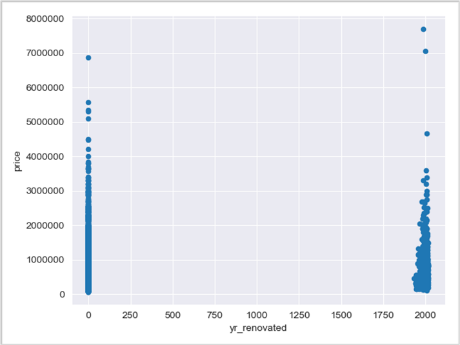
   df.plot(kind='scatter',x=x\_var,y='price') *#设置绘图的行和列*











上述这些图呈现了每种属性与房价的散点分布图，从图中可以看出从上述这些图可以看出，属性的分布不服从整体分布，分布规律不明显。因此无法直接分析单个属性的变化对房价预测结果的影响，即使用简单的多元线性回归算法对房价预测的效果可能会比较差，后续需要分析属性之间的相关性，用于采用相对复杂的回归模型进行预测。

* 1. 属性之间的相关性

显示下三角的相关系数，用于呈现属性之间的相关系数图，简化相关系数图。

*#删除原始数据中的索引id*

df.drop(["id"],axis=1,inplace=True)

*#计算属性间的相关系数图*

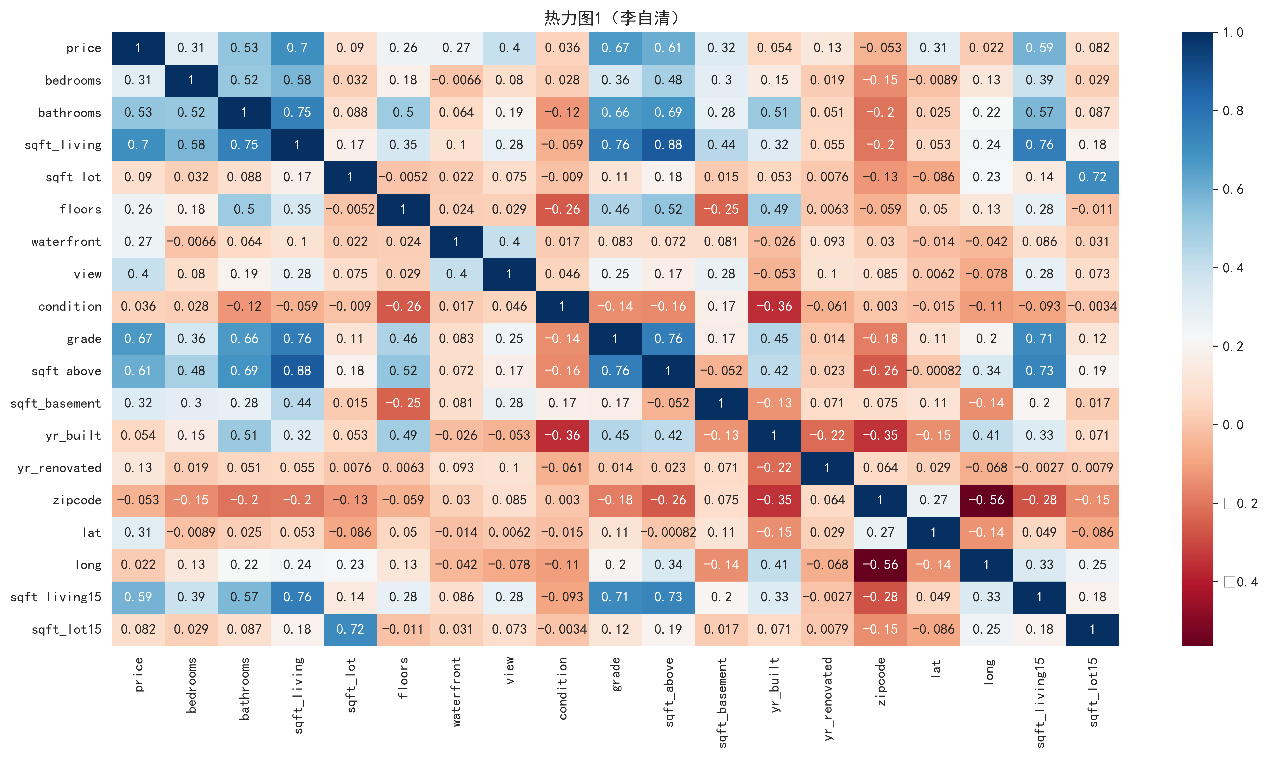
corr = df.corr()

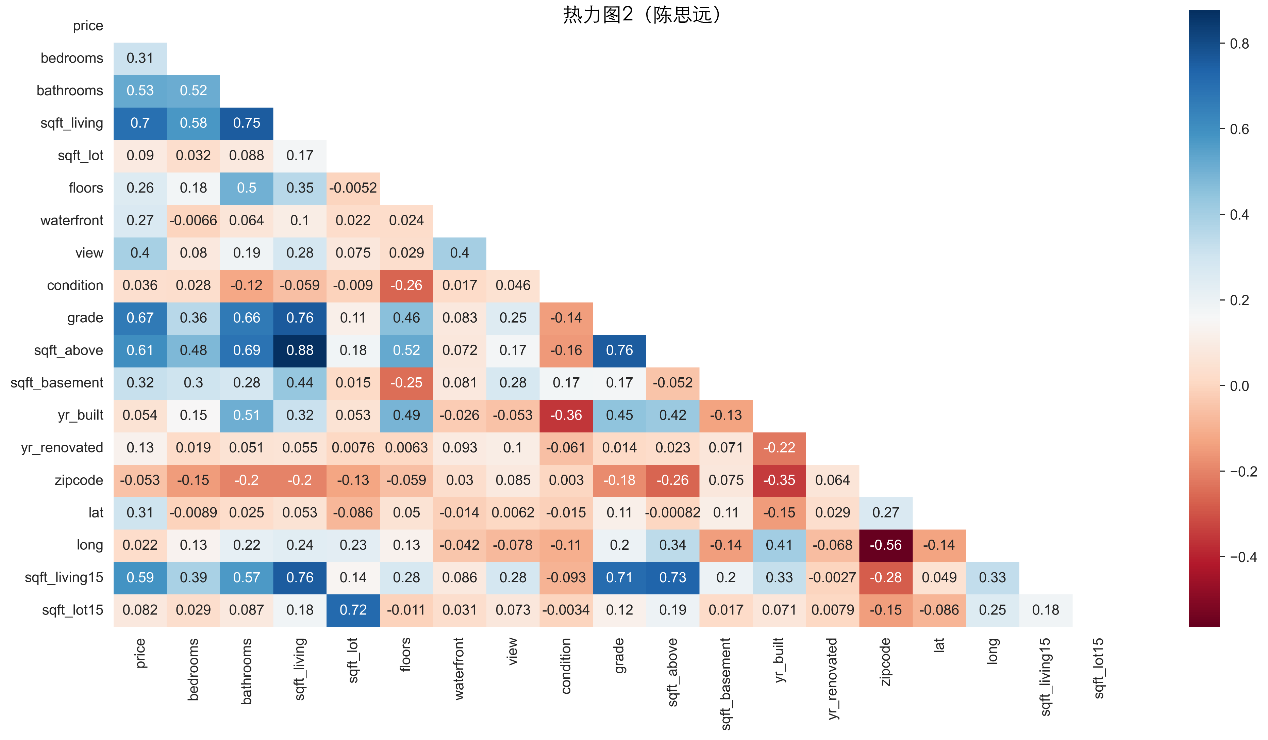
*#绘制属性相关系数的热力图*

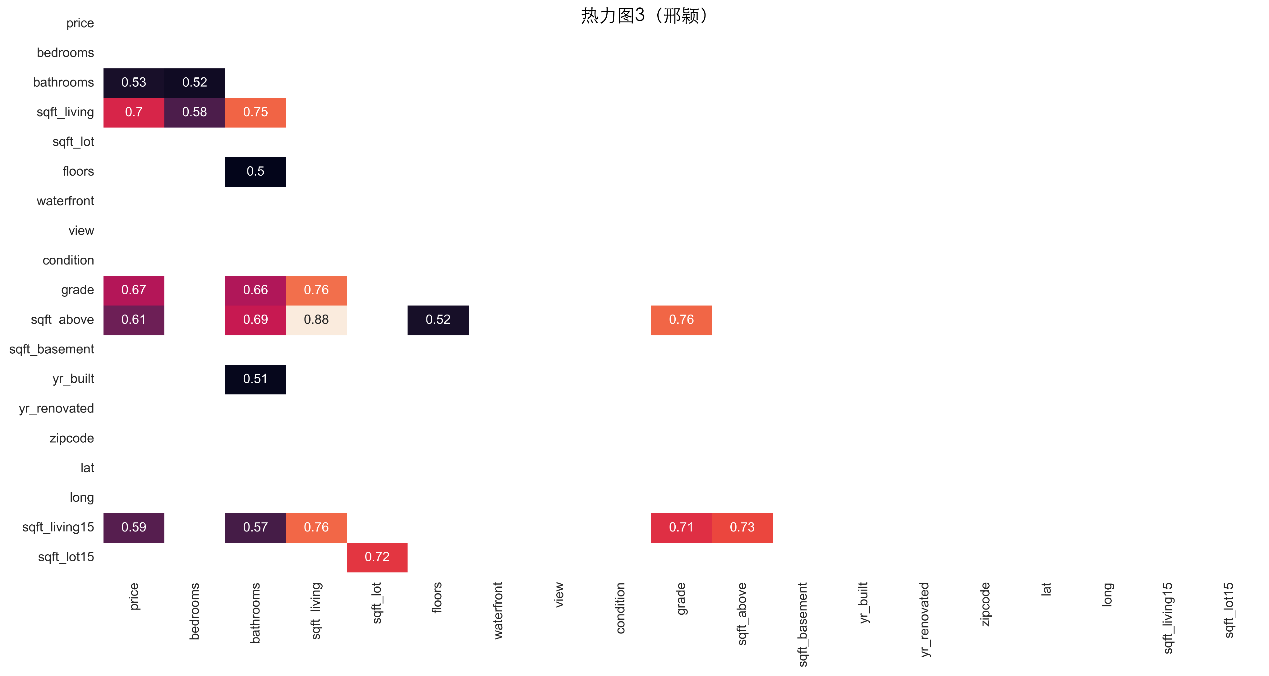
plt.figure(figsize=(16,8))

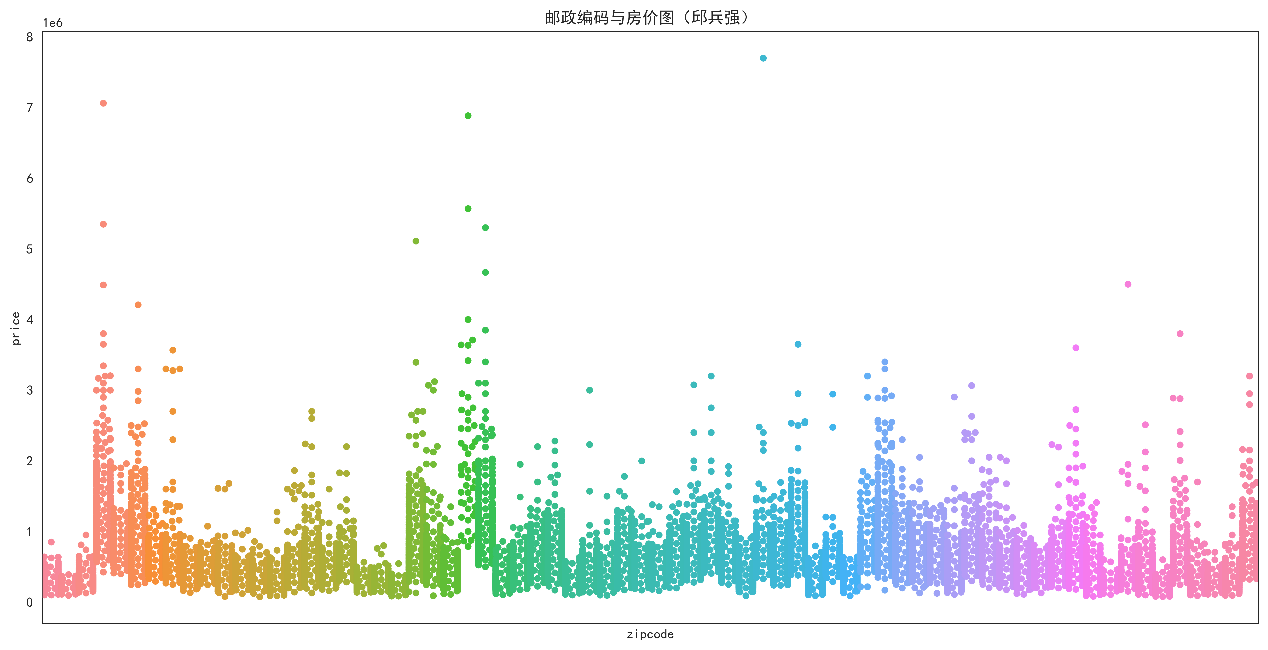
sns.heatmap(corr,annot=True,cmap="RdBu")

plt.show()









由热力图可以看出相关系数比较低的几个属性有‘long’、‘yr\_built’、‘condition’、‘zipcode’等，下面再具体分析 ‘long’（地理经纬度位置）和’zipcode（邮政编码）’与房价的关系。

* 1. 分析地理位置与房价的相关性

查看所有房子的地理分布的散点图，查看房屋的地理分布是否存在相关的分布规律

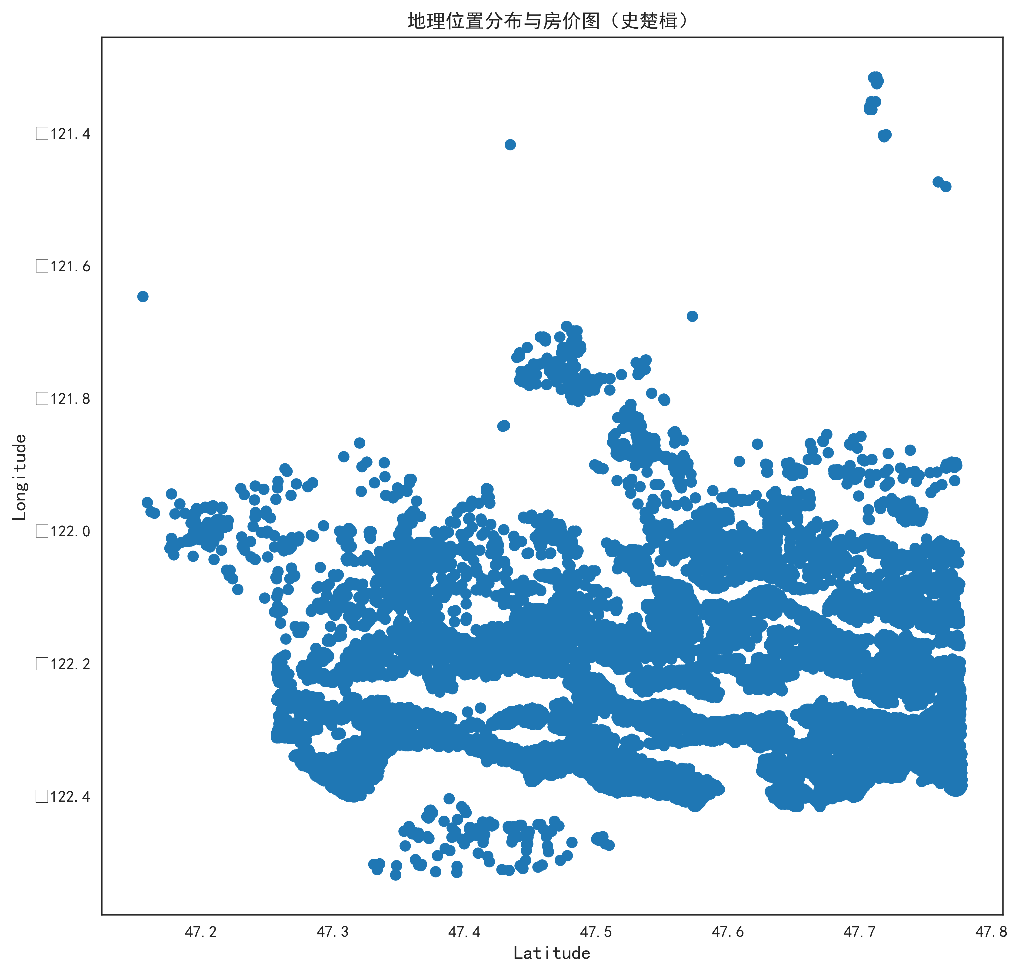
plt.figure(figsize=(10,10))

plt.scatter(df.lat, df.long)

plt.ylabel('Longitude', fontsize=12)

plt.xlabel('Latitude', fontsize=12)

plt.show()



由上图可知，房屋的地理位置分布相对比较集中，但无法看到各区的房屋价格分布情况。所以其可以去掉。

X = df.drop(['price','long'],axis=1)

X = X.values

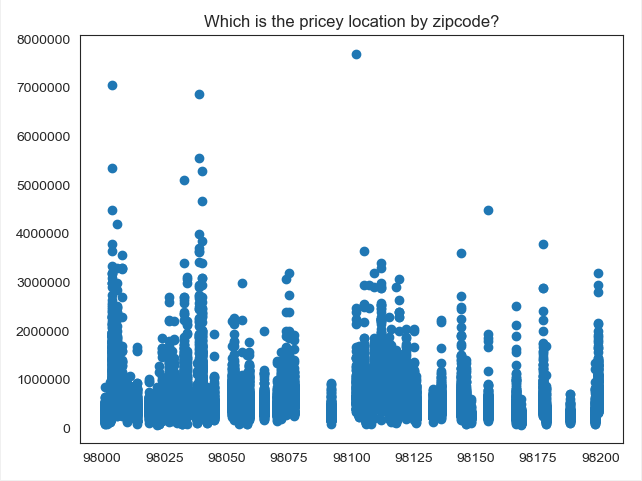
* 1. 分析邮编与房价的相关性

绘制各区域具体的价格散点分布情况，了解每个区域的价格分布区间

plt.scatter(df.zipcode,df.price)

plt.title("Which is the pricey location by zipcode?")

plt.show()



从图中可以看出，部分地区的房价跨度较大，有些地区房价比较便宜，因此邮编也是影响房价重要的因素。虽然其相关性低，但是不能去掉。

1. **模型建立与训练**

**3.1步骤一：**

采用线性回归建立回归模型

from pylab import \*

mpl.rcParams['font.sans-serif'] = ['SimHei']

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

*#采用线性回归进行模型训练*

model.fit(X\_train, y\_train)

*#let us predict*

*#获取模型预测结果*

y\_pred=model.predict(X\_test)

*#打印模型评分结果*

print (model.score(X\_test, y\_test))

输出如下模型结果：

模型得分值为：0.7101113356082593

模型评估值反映的是模型的整体预测效果，该值的取值范围是[0,1]，模型评估值越接近1表示模型越好，此处模型评估值为0.71，该值不是很大，表明模型效果不是特别理想。

**步骤二：**

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n\_estimators=500)

*#配置模型中回归树的个数为500*

model.fit(X\_train, y\_train)

y\_pred=model.predict(X\_test)

*#可视化*

y\_t=sorted(y\_test)

y\_p=sorted(y\_pred)

plt.plot(y\_t[-200:],color='r',label="Actual price")

plt.plot(y\_p[-200:],color='b',label="Predicted price")

plt.title("随机森林模型下的预测值与真实值比较（李自清）")

plt.legend()

plt.savefig('p3.png', dpi=500, bbox\_inches='tight') *# 解决图片不清晰，不完整的问题*

plt.show()

print (model.score(X\_test, y\_test))

输出如下模型结果：  
模型得分值为：0.8767796942656374  
通过模型对比可知，随机森林回归模型预测效果相对较好，与线性回归相比，预测效果有所改进。

**步骤三**  
*#导入相关库*

from sklearn.ensemble import GradientBoostingRegressor

*#配置GBDT回归模型的分类器个数*

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

model = GradientBoostingRegressor(n\_estimators=500)

model.fit(X\_train, y\_train)

*#采用测试数据集进行模型预测*

y\_pred=model.predict(X\_test)

*#可视化*

y\_t=sorted(y\_test)

y\_p=sorted(y\_pred)

plt.plot(y\_t[-200:],color='r',label="Actual price")

plt.plot(y\_p[-200:],color='b',label="Predicted price")

plt.title("GDBT回归下的预测值与真实值比较（李自清）")

plt.legend()

plt.savefig('p4.png', dpi=500, bbox\_inches='tight') *# 解决图片不清晰，不完整的问题*

plt.show()

print (model.score(X\_test, y\_test))

对比三种模型评估值可知，GBDT模型评估值相对较好。

**步骤四：**

采用最近邻算法建立回归模型

from sklearn.neighbors import KNeighborsRegressor

*#配置最近邻回归模型参数*

model = KNeighborsRegressor(n\_neighbors=10)

model.fit(X\_train, y\_train)

y\_pred=model.predict(X\_test)

*#可视化*

y\_t=sorted(y\_test)

y\_p=sorted(y\_pred)

plt.plot(y\_t[-200:],color='r',label="Actual price")

plt.plot(y\_p[-200:],color='b',label="Predicted price")

plt.title("最近邻回归下的预测值与真实值比较（李自清）")

plt.legend()

plt.savefig('p5.png', dpi=500, bbox\_inches='tight') *# 解决图片不清晰，不完整的问题*

plt.show()

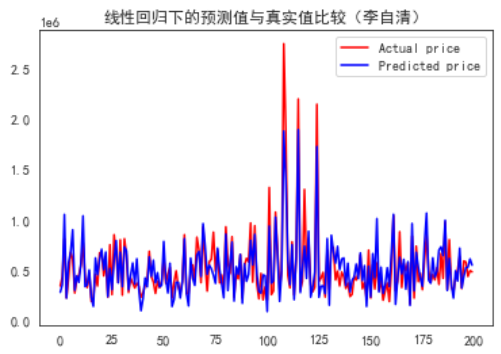
print (model.score(X\_test, y\_test))

模型得分值为：0.7915905936059999

五．实验结果分析、对比和讨论

1. **可视化结果**

对于前面使用四种回归模型，对第一种线性回归模型，使用数据可视化，画出预测值与真实值之间的对比关系图。由于测试集数量过于巨大会影响可视化，在这里只采用最后的200组数据进行可视化。



由于曲线分布无规律，难以看出预测效果。所以采用价格升序重新可视化。

y\_t=sorted(y\_test)

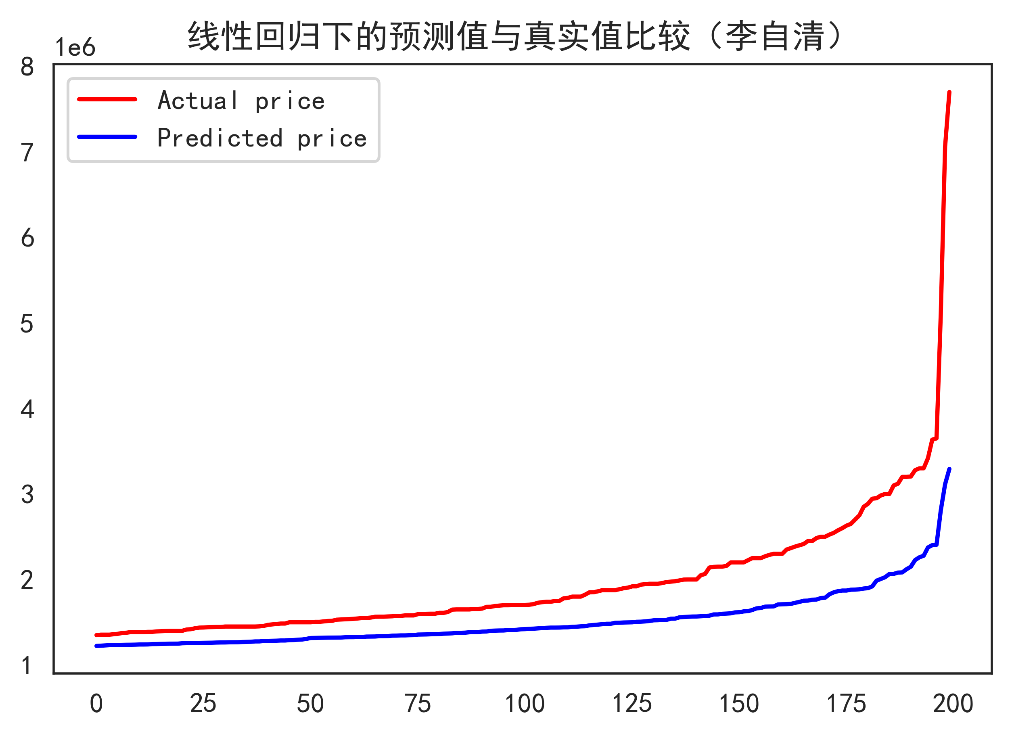
y\_p=sorted(y\_pred)

plt.plot(y\_t[-200:],color='r',label="Actual price")

plt.plot(y\_p[-200:],color='b',label="Predicted price")

plt.title("线性回归下的预测值与真实值比较（李自清）")

plt.legend()

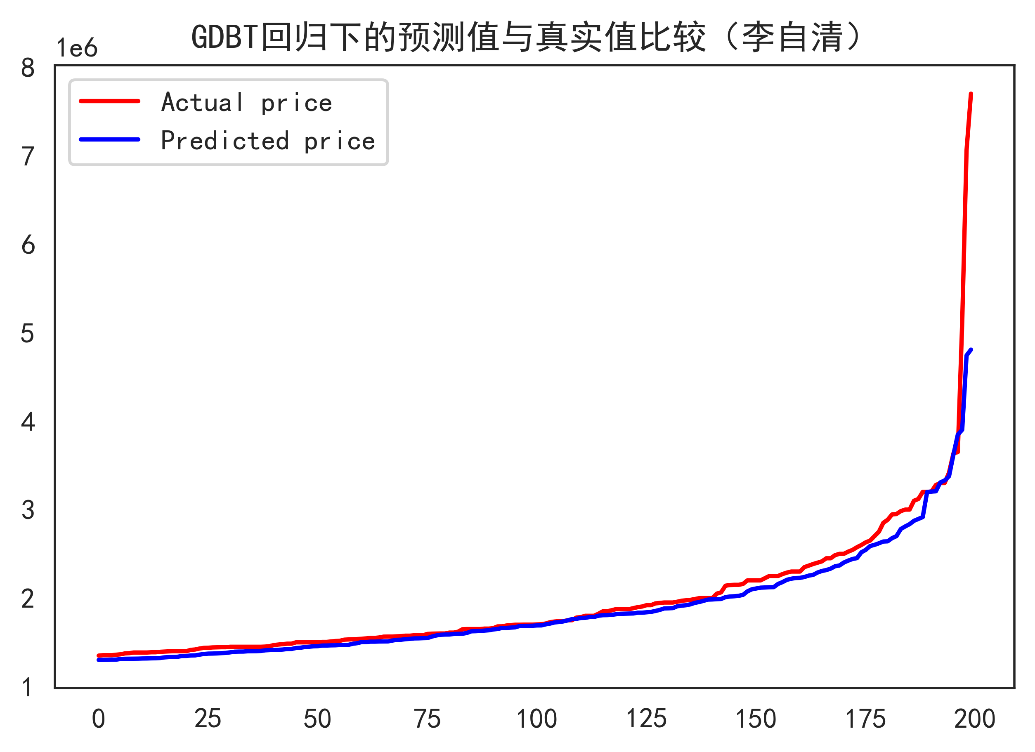


由此可以直观看出预测价格与真实价格之间还存在一定发误差。经过计算其综合系数

print(model\_EN.score(X\_test,y\_test))

计算得其系数为0.7102093703489807，还有提升的空间。

在原方案中系数最高的一个算法是GBDT回归模型，综合系数大小为0.8874803439872003



由此我们引进了其它算法加以改进。在我们引进了ElasticNet回归、Lasso回归、岭回归、支持向量回归。经训练其在测试集上的得分均不高，都在0.7左右，效果不佳，下面是尝试新模型的过程。

新模型引入：

**ElasticNet回归**

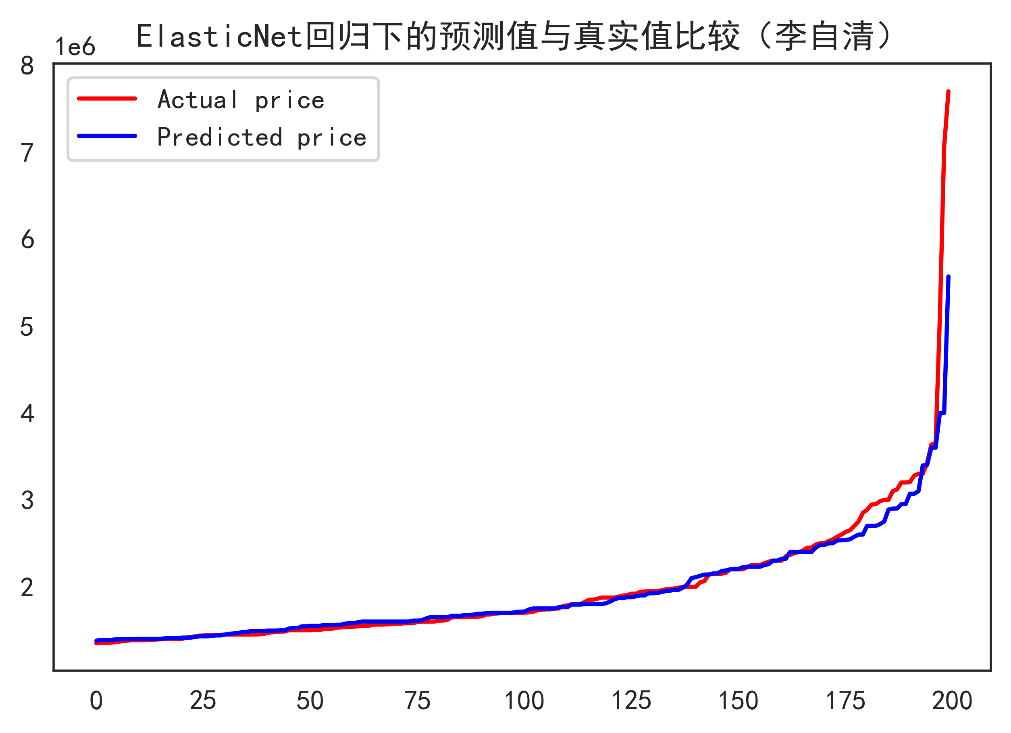
from sklearn.linear\_model import LinearRegression,Lasso,ElasticNet

from sklearn.metrics import r2\_score

model\_EN=ElasticNet(0.01)  *#实例化弹性网络回归对象*

model\_EN.fit(X\_train,y\_train) *#训练*

y\_pred=model.predict(X\_test) *#预测*



模型评分为0.7096093814267334

**Lasso回归**

la = Lasso()

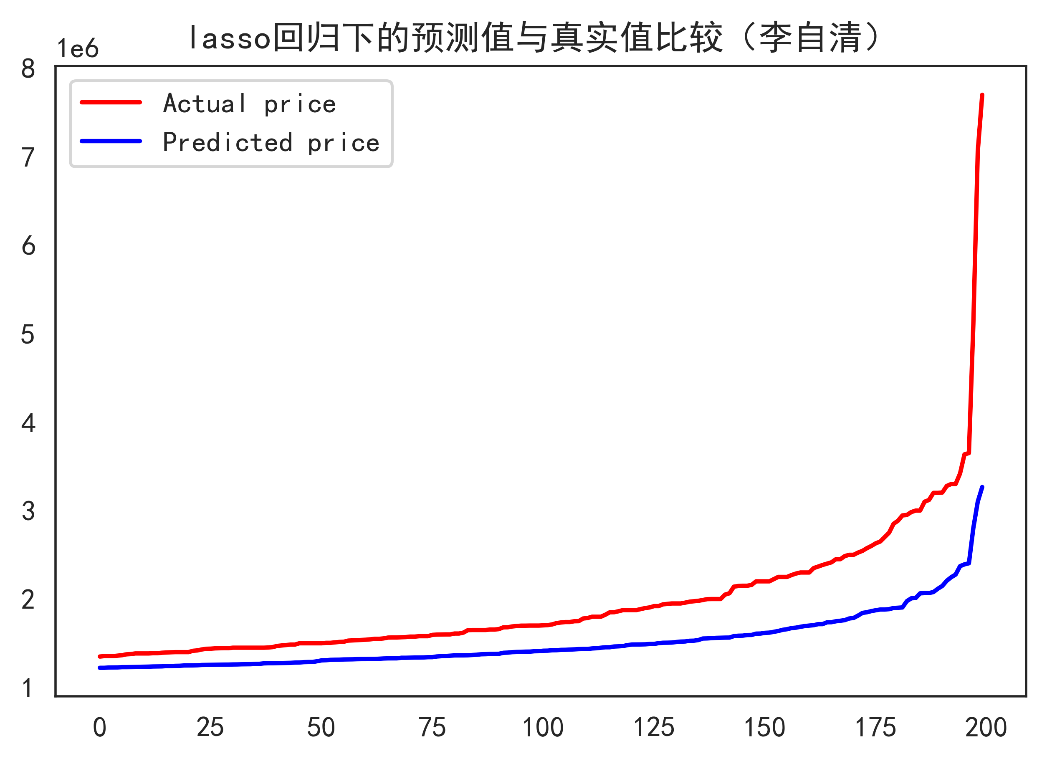
la.fit(X\_train, y\_train)*#拟合*

y\_pred=la.predict(X\_test) *#预测*

*#评价*

print(r2\_score(y\_pred,y\_test))

y\_predt=la.predict(X\_train)  *#查看训练集上的效果*



模型评分为:0.7097657798427943

**贝叶斯岭回归**

from sklearn.linear\_model import  BayesianRidge

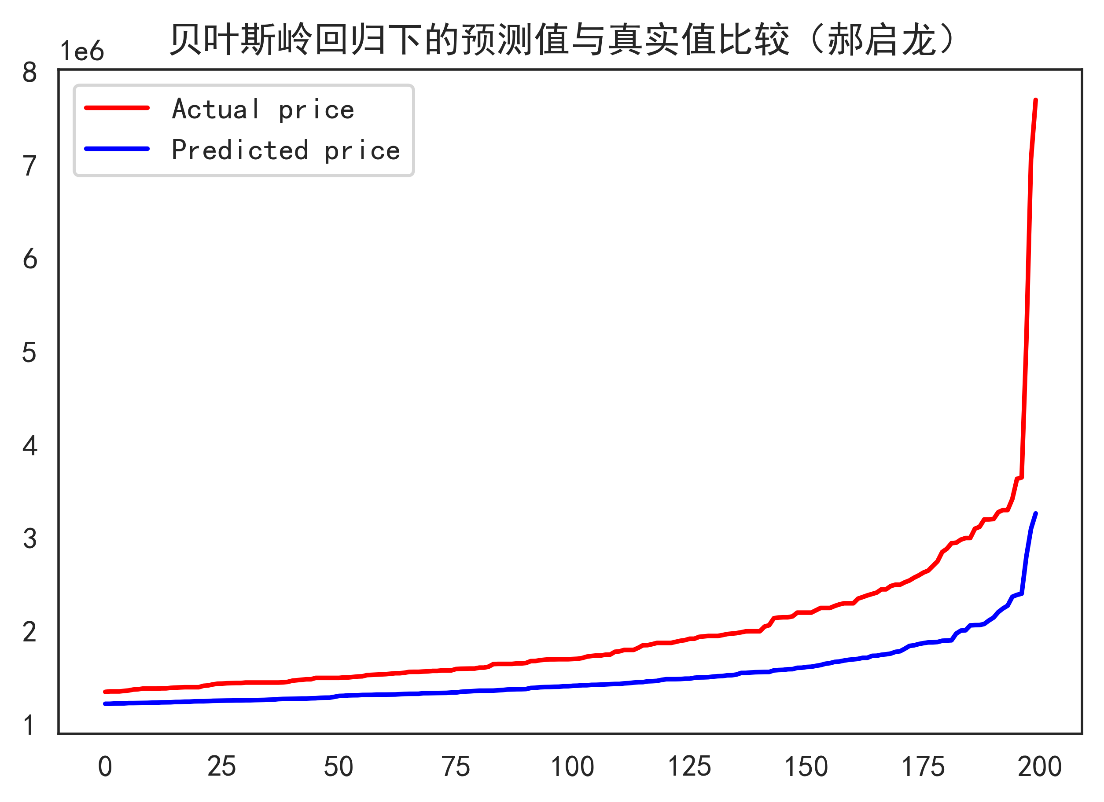
Brd=BayesianRidge()

Brd.fit(X\_train,y\_train)

y\_pred=Brd.predict(X\_test)

print(r2\_score(y\_pred,y\_test))

y\_predt=Brd.predict(X\_train)



模型评分为:0.7097657798427943

**支持向量回归**

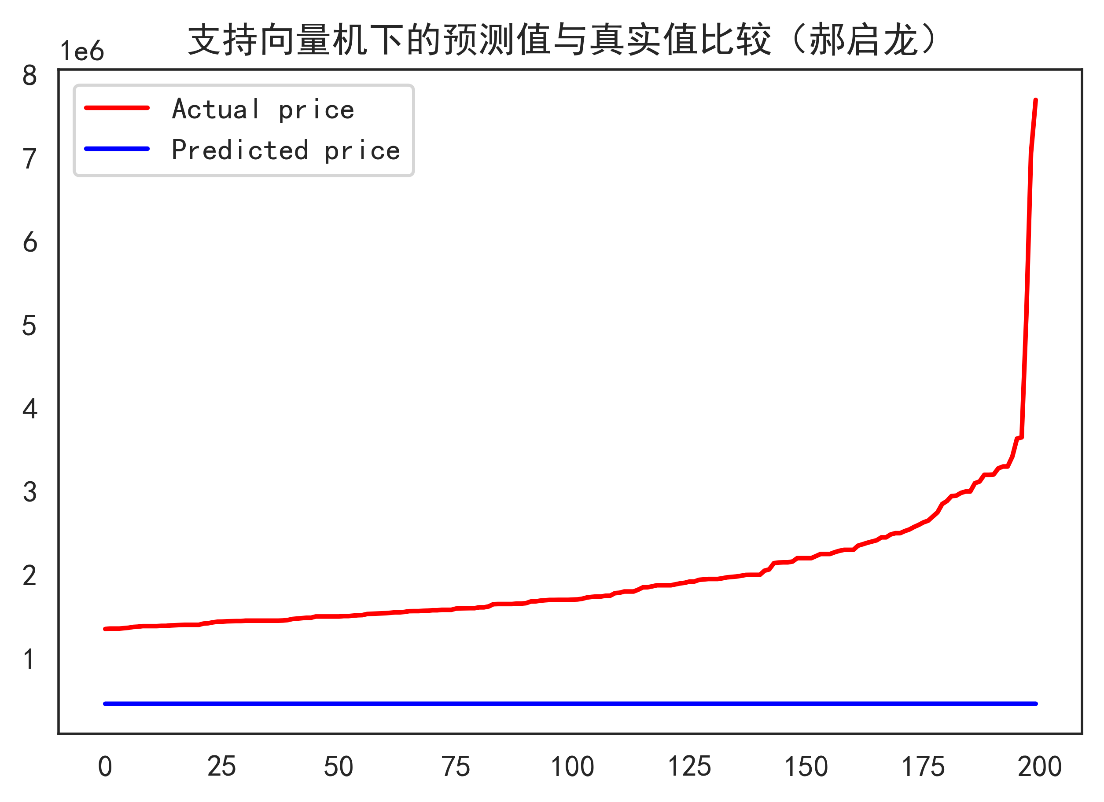
from sklearn.svm import  SVR

svr=SVR()

svr.fit(X\_train,y\_train)

y\_pred=svr.predict(X\_test)

y\_predt=svr.predict(X\_train)



模型评分为:-0.052430661915995325（效果特别差）

在尝试了几种新模型后，发现GDBT模型的评分最高，所以我们采用此模型，然后对其进行超参数优化。

1. **优化方案**

所以我们针对GBDT回归模型，其模型评估分数最高，首先配置该模型，计算平均绝对误差，均方根误差，r2模型值。

model = GradientBoostingRegressor(n\_estimators=500)

*#采用训练数据进行模型训练*

model.fit(X\_train, y\_train)

*#采用测试数据进行模型预测*

y\_predicted = model.predict(X\_test)

*#导入模型结果评估模块平均绝对误差，均方根误差和r2值*

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

*#计算平均绝对误差，均方根误差，r2模型值*

mean\_absolute\_error(y\_test,y\_predicted)

mean\_squared\_error(y\_test,y\_predicted)

r2\_score(y\_test,y\_predicted)

*#输出平均绝对误差，均方根误差，r2模型值*

print(r2\_score(y\_test,y\_predicted))

print(mean\_absolute\_error(y\_test,y\_predicted))

print(mean\_squared\_error(y\_test,y\_predicted))

结果如下

0.8870750863487923

67546.83697871598

15576117932.921446

即

模型评估值为0.8870750863487923

平均绝对误差：67546.83697871598

均方根误差：15576117932.921446

其误差和评估值还有待提升，由于在此回归模型使用的是默认参数，可能有些参数值并不符合本方案模型的最优值。所以我们需要找出其参数最优值。下面先给出关于GBDT模型的超参数列表

1. 划分时考虑的最大特征数max\_features: 可以使用很多种类型的值，默认是"None",意味着划分时考虑所有的特征数；如果是""意味着划分时最多考虑个特征；如果是"sqrt"或者"auto"意味着划分时最多考虑个特征。如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑（百分比×）取整后的特征数。其中为样本总特征数。一般来说，如果样本特征数不多，比如小于50，我们用默认的"None"就可以了，如果特征数非常多，我们可以灵活使用刚才描述的其他取值来控制划分时考虑的最大特征数，以控制决策树的生成时间。
2. 决策树最大深度max\_depth: 默认可以不输入，如果不输入的话，默认值是3。一般来说，数据少或者特征少的时候可以不管这个值。如果模型样本量多，特征也多的情况下，推荐限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值10-100之间。
3. 内部节点再划分所需最小样本数min\_samples\_split: 这个值限制了子树继续划分的条件，如果某节点的样本数少于min\_samples\_split，则不会继续再尝试选择最优特征来进行划分。默认是2，如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。
4. 叶子节点最少样本数min\_samples\_leaf: 这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。 默认是1,可以输入最少的样本数的整数，或者最少样本数占样本总数的百分比。如果样本量不大，不需要管这个值。如果样本量数量级非常大，则推荐增大这个值。
5. 叶子节点最小的样本权重和min\_weight\_fraction\_leaf：这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。 默认是0，就是不考虑权重问题。一般来说，如果我们有较多样本有缺失值，或者分类树样本的分布类别偏差很大，就会引入样本权重，这时我们就要注意这个值了。
6. 最大叶子节点数max\_leaf\_nodes: 通过限制最大叶子节点数，可以防止过拟合，默认是"None”，即不限制最大的叶子节点数。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。如果特征不多，可以不考虑这个值，但是如果特征分成多的话，可以加以限制，具体的值可以通过交叉验证得到。
7. 节点划分最小不纯度min\_impurity\_split: 这个值限制了决策树的增长，如果某节点的不纯度(基于基尼系数，均方差)小于这个阈值，则该节点不再生成子节点。即为叶子节点 。一般不推荐改动默认值1e-7。

接下来是超参数调节，我们主要使用两种调节方法，第一个是GridSearchCV网格搜索法，第二个是RandomizedSearchCV随机搜索法。

* 1. loss 、min\_samples\_leaf、alpha参数调优

由于此参数组合初次尝试使用网格搜索法GridSearchCV对于大量数据消耗时间较长，所以更换为随机搜索RandomizedSearchCV。

对这三种参数的取值范围为

loss：squared\_error , absolute\_error, huber, quantile

（均方误差、绝对误差、均方绝对结合误差、分位数损失）

min\_samples\_leaf：[1,2,3,4,5]

alpha：[ 0.001 ,0.003, 0.01, 0.03, 0.1,0.3,0.6,0.9]（以乘3为因子）

使用随机搜索代码如下

model\_gbr = GradientBoostingRegressor()

parameters1 = {'loss': ['squared\_error', 'absolute\_error', 'huber', 'quantile'],'min\_samples\_leaf': [1,2,3,4,5],

              'alpha': [0.001 ,0.003, 0.01, 0.03, 0.1,0.3,0.6,0.9] }

rsearch1= RandomizedSearchCV(estimator=model\_gbr, param\_distributions=parameters, cv=5)

rsearch1.fit(X\_train,y\_train)

print('Best score is:', rsearch1.best\_score\_)

print('Best parameter is:', rsearch1.best\_params\_)

运行结果为

Best score is: 0.8845072041696216

Best parameter is: {'loss': squared, 'min\_samples\_leaf': 4, 'alpha':0.6}

所得的最优超参数为：

'loss' = squared , 'min\_samples\_leaf' = 4 , 'alpha' = 0.6

将所得超参数代入原模型

model = GradientBoostingRegressor(n\_estimators=500,alpha=0.6,

loss='squared\_error',min\_samples\_leaf=4)

得模型综合评分

0.8897557192533482

67893.62595403525

15206369106.854612

相比之前的默认参数，此次优化中

模型评估值：0.8897557192533482，增加了0.302％

平均绝对误差：67893.62595403525，增加了0.503％

平均相对误差：15206369106.854612，降低了2.373％

可见模型评估值稍有提升，但是平均绝对误差也稍有增大，下面继续优化。

* 1. max\_depth和min\_samples\_split参数调优

这里的参数组合使用GridSearchCV网格搜索法

max\_depth：[3,12]（增长步数为2）

min\_samples\_split：[100,800]（增长步数为200）

model\_gbr = GradientBoostingRegressor(n\_estimators=500,alpha=0.6,

loss='squared\_error',min\_samples\_leaf=4)

parameters2 = {'max\_depth':range(3,14,2), 'min\_samples\_split':range(100,801,200)}

gsearch1 = GridSearchCV(estimator=model\_gbr, param\_grid=paramters2,cv=5)

运行结果为

Best score is: 0.8775022091626203

Best parameter is: {'max\_depth': 7, 'min\_samples\_split': 100}

所得的最优超参数为：

'max\_depth' = 7 , 'min\_samples\_split' = 100

将所得参数和之前得到的参数一起代入模型

model=GradientBoostingRegressor(n\_estimators=500,alpha=0.6,loss='square

d\_error',min\_samples\_leaf=4, max\_depth = 7,min\_samples\_split = 100)

运行结果为

0.8977136881179598

64391.03371570426

14108699358.57775

相比之前的默认参数，此次优化中

模型评估值0.8977136881179598，增加了1.21％

平均绝对误差：64391.03371570426，降低了4.67％

平均相对误差：14108699358.57775，降低了9.42％

* 1. min\_samples\_split和min\_samples\_leaf调优

使用GridSearchCV网格搜索法。由于决策树深度7是一个比较合理的值，我们把它定下来，对于内部节点再划分所需最小样本数min\_samples\_split，我们暂时不能一起定下来，因为这个还和决策树其他的参数存在关联。下面我们再对内部节点再划分所需最小样本数min\_samples\_split和叶子节点最少样本数min\_samples\_leaf一起调参。

'min\_samples\_split' : range[100,800]（步长为200）

‘min\_samples\_leaf' : [1,2,3,4,5,6]}

model\_gbr = GradientBoostingRegressor(n\_estimators=500,alpha=0.6,

loss='squared\_error',min\_samples\_leaf=4,max\_depth=7)

parameter3 = {'max\_depth':range(3,14,2), 'min\_samples\_split':range(100,801,200)}

gsearch3 = GridSearchCV(estimator=model\_gbr, param\_grid= parameter3,cv=5)

Best score is: 0.8881434737056428

Best parameter is: {'min\_samples\_split': 100，'min\_samples\_leaf': 3}

所得的最优超参数为：

'min\_samples\_split'= 100，'min\_samples\_leaf'= 3

将所得参数与前面所得一起代入原模型：

model=GradientBoostingRegressor(n\_estimators=500,alpha=0.6,loss='square

d\_error',min\_samples\_leaf=3, max\_depth = 7,min\_samples\_split = 100)

运行结果

0.9018703240318775

64061.30735416651

13535360410.545647

相比之前的默认参数，此次优化中

模型评估值0.9018703240318775，增加了1.678％

平均绝对误差：64061.30735416651，降低了5.160％

平均相对误差：13535360410.545647，降低了13.102％

* 1. max\_features调优

现在我们再对最大特征数max\_features进行网格搜索,使用GridSearchCV网格搜索法。

max\_features : [7,20]（步数为2）

model=GradientBoostingRegressor(n\_estimators=500,alpha=0.6,

loss='squared\_error',min\_samples\_leaf=3,max\_depth = 7,min\_samples\_split = 100)

parameter 4 = {'max\_features':range(7,20,2)}

gsearch4 = GridSearchCV(estimator=model\_gbr, param\_grid= parameter4,cv=5)

运行结果

Best score is: 0.8861432731036921

Best parameter is: {'max\_features': 13}

将所得参数与前面所得一起代入原模型：

model=GradientBoostingRegressor(n\_estimators=500,alpha=0.6,

loss='squared\_error',min\_samples\_leaf=3,max\_depth = 7,

min\_samples\_split = 100,max\_features=13)

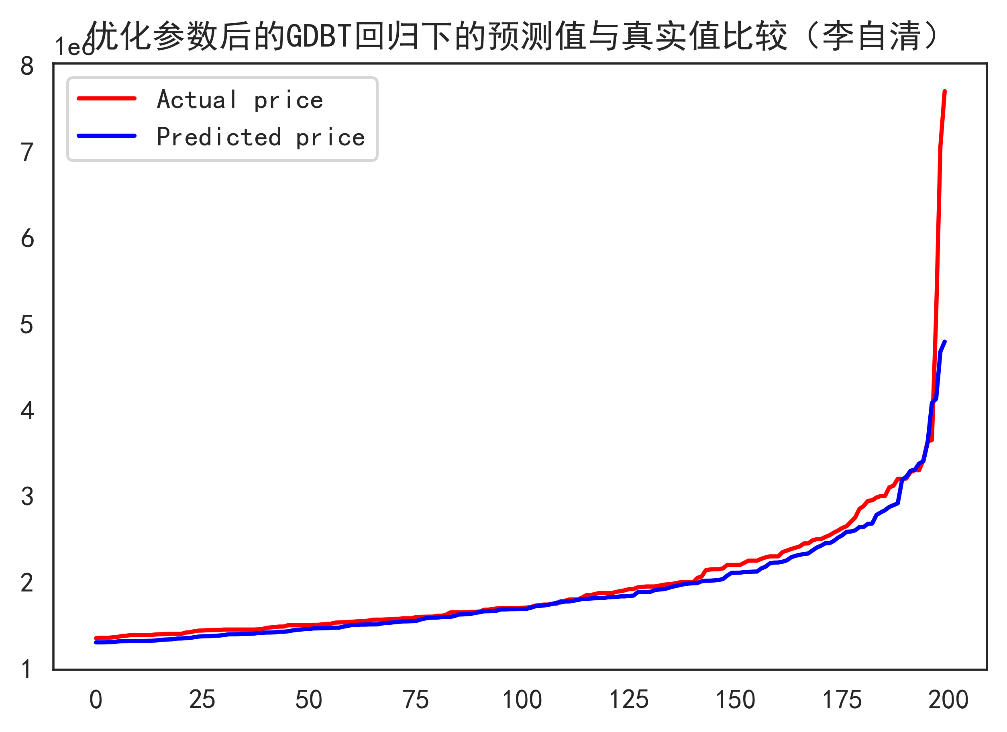
运行结果为

0.9026807120493243

63838.011742899325

13423580831.327494

比较图如下：（见下页）



相比之前的默认参数，此次优化中

模型评估值0.9026807120493243，增加了1.760％

平均绝对误差：63838.011742899325，降低了5.491％

平均相对误差：13423580831.327494，降低了13.819％

1. **更换数据**

更换的数据选择经典的波士顿房价**（Boston housing）数据集预测**

数据来源[Boston housing dataset | Kaggle](https://www.kaggle.com/datasets/altavish/boston-housing-dataset)

其中相关变量的解释如下

|  |  |  |
| --- | --- | --- |
| **属性名** | **解释** | **类型** |
| **CRIM** | **该镇的人均犯罪率** | **连续值** |
| **ZN** | **占地面积超过25平方英尺的住宅用地比例** | **连续值** |
| **INDUS** | **非零售商业用地比例** | **连续值** |
| **CHAS** | **是否临近Charies River** | **离散值，1=邻近；0=不邻近** |
| **NOX** | **一氧化氮浓度** | **连续值** |
| **RM** | **每栋房屋的平均客房数** | **连续值** |
| **AGE** | **1940年之前建成的自用单位比例** | **连续值** |
| **DIS** | **到波士顿5个就业中心的加权平均值** | **连续值** |
| **RAD** | **到径向公路的可达性指数** | **连续值** |
| **TAX** | **全值财产税率** | **连续值** |
| **PIRATIO** | **学生与教师的比例** | **连续值** |
| **B** | **1000（BK-0.63）^2，其中BK为黑人比** | **连续值** |
| **LSTAT** | **低收入人群占比** | **连续值** |
| **MEDV** | **同类房屋价格的中位数** | **连续值** |

**更换数据报告书见附件。**

七．对本门课的感想、意见和建议

在机器学习这门课的学习中，我们一定要注意知识点前后的联系。如果只是每一章都看完，顶多就是了解了一种方法，相互之间是割裂的，这样当碰到一个实际问题，仍然无法思路完整的求解，因而有必要在阅读时就要有意识甚至刻意的建立起知识架构。实际上，所谓的机器学习，是面对一个具体的问题，从给定的数据中产生模型的算法，也就是说脱离了实际问题谈机器学习算法是毫无意义的。

参考本书的章节内容，加上自己学习中的一些理解，简单总结一下基于机器学习的一般性问题解决方法。在给定数据集（所谓大数据）和具体问题的前提下，一般解决问题的步骤可以概括如下：

1、数据抽象

将数据集和具体问题抽象成数学语言，以恰当的数学符号表示。这样做自然是为了方便表述和求解问题，而且也更加直观。

2、设定性能度量指标

机器学习是产生模型的算法，一般来说模型都有误差。如果模型学的太好，把训练样本自身的一些特点当成所有潜在样本具有的一般性质，这种情况称为过拟合，这样的模型在面对新样本时就会出现较大误差，专业表述就是导致模型的泛化性能下降。

与之相对的是欠拟合，模型对样本的一般性质都没学好，这种情况一般比较好解决，扩充数据集或者调整模型皆可。

而一般来说无论是机器学习还是现在很火的深度学习，面对的主要问题都是过拟合。那么为了保证模型的泛化能力足够强，必须要有衡量模型泛化能力的评价标准，也就是性能度量的设定。

很显然不同的性能度量会导致不同的评判结果，好的性能度量能够直观的显示模型的好坏，同时也能看到不同模型，或者模型的不同参数下对解决问题的程度好坏。

当然更为重要的是，仅仅设定好性能度量是不够的，不同模型或者不同参数下得到的性能度量结果一般是不同的，一般来说不能简单的比较结果，而应该基于统计假设检验来做效果判定。也就是说通过比较检验的方法，我们就可以判断，如果观察到A比B好，在统计意义上A的泛化性能是否优于B，以及这个判断的把握有多大。

3、选定模型

在数据集完美的情况下，接下来就是根据具体问题选定恰当的模型了。一种方式是根据有没有标记样本考虑。如果是有标记样本，可以考虑有监督学习，反之则是无监督学习，兼而有之就看半监督学习是否派的上用场。无监督学习方法主要提到的是聚类。随机选定几个样本，通过一定的算法不停迭代直至收敛或者达到停止条件，然后便将所有样本分成了几类。

对有监督学习而言，根据最终所需要的输出结果，如果涉及到分类，可以参考的模型有线性回归及其非线性扩展、决策树、神经网络、支持向量机SVM、规则学习等。如果是回归问题，可以认为是分类的连续形式，方法便是以上模型的变种或扩展。如果涉及到概率，可以参考的有神经网络、贝叶斯、最大似然、EM、概率图、隐马尔科夫模型、强化学习等。

4、训练及优化

选定了模型，如何训练和优化也是一个重要问题。如果要评估训练集和验证集的划分效果，常用的有留出法、交叉验证法、自助法、模型调参等；如果模型计算时间太长，可以考虑剪枝；如果是过拟合，则可通过引入正则化项来抑制（补偿原理）；如果单个模型效果不佳，可以集成多个学习器通过一定策略结合，取长补短（集成学习）。

5、应用

模型训练好之后，自然就是应用，针对未知样本做预测、推荐等。

八．华为实验指导书相关实验的复现报告书

**见附件：**