**房价影响因素分析**

10132510355 黄怿豪

摘要:

本文主要分析影响房价的因素，数据来源为链家网，机器学习模型的使用中，采用了三种线性模型，一种非线性模型，最后得出的结论是房子的大小，房子的位置，房子的建造年份以及房子的高度对房价影响较大。

一、问题描述

现在房价居高不下，特别是上海等一线城市，房价更是高的离谱，那么在决定一个房子的价格中，哪些因素占了主要的地位，如何让想买房的人快速获取大概的房价信息。那么本文介绍的就是如何用机器学习去训练上海房价信息并生成模型然后进行分析的过程。

二、数据收集及处理

（1）数据源选择

经过在网上对几个房价信息网的比较，



安居客房价信息



搜房网房价信息



链家网房价信息

可以看到，各个房价网站展示的房价信息差别不大，另外根据百度上查到链家网的房价与实际的房价差距较小，所以就把链家网的房价数据作为数据源。

目前上海的新房较少，链家网上上海的新房信息只有约400套，数据集过小，不适合进行训练，所以选择了二手房信息作为数据集进行训练。在链家网上，上海二手房数据有几万套，可以进行机器学习的训练。

（2）数据收集

使用爬虫从链家网上爬取房价信息的数据，爬虫使用python编写，并存入mongoDB数据库，以备之后的训练。爬虫代码见附录

（3）数据处理

由之前截图可见，房价信息数据还不能直接用于机器学习的训练，所以对其中的一些数据进行处理使得更适合训练。

首先是房型



可以看到这里可以比较轻松的把房型拆分为几个房间，几个客厅。



对于楼层高度，总共有6层高，房子位于中层，那么就用6\*0.5表示房子的高度。同样，对于低层和高层的房子，分别用楼高乘以0.2和0.8表示房子的高度。



对于地址，调用了高德地图的地址查询api把它转换成了经纬度信息。

其他的数据信息都比较标准，不需要进行额外的处理。

三、采用的模型及原因

所有的模型训练都是把已有的数据集按照80%作为训练集，20%作为测试集来进行模型的训练的。

（1）线性回归模型

线性回归模型是比较基本的房价训练模型，在预测波士顿房价的这个机器学习案例中使用的就是线性回归模型。并且，给每个房价的特征赋予一个系数来生成具体的房价是一个直观上感觉很正确的方法。

（2）神经网络

目前比较流行的机器学习方法，对于数据的训练通常会有比较好的结果，因为通过反向传播算法不断迭代使得预测结果逼近实际值，可以作为与线性回归模型的对比来看看是否会有更好的表现。

（3）支持向量机

也是比较流行的机器学习算法，对于线性模型的训练效果一般会优于线性回归模型，使用这个模型也是想看能不能对线性回归模型进行一些提高。

四、使用的python机器学习库

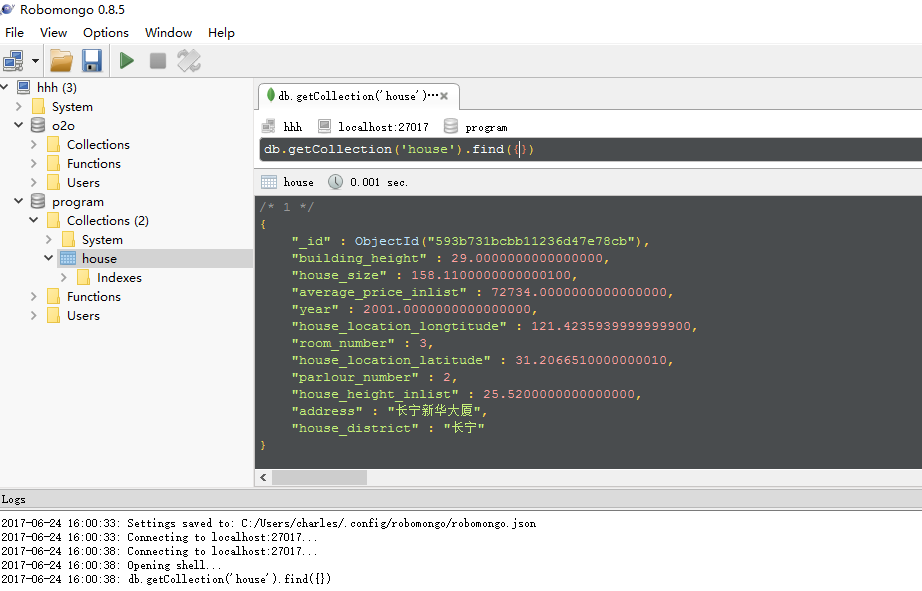
因为目前python对机器学习的支持比较好，有大量机器学习的库可以调用，而且我的爬虫也是使用机器学习编写的，所以决定使用python的sklearn库中的机器学习算法来进行训练及预测。

Sklearn作为目前机器学习中一个常用的python第三方模块，封装了许多机器学习的算法，它的官网是<http://scikit-learn.org/stable/>，在github上可以看到相应的介绍和源代码。这个库是在2007年Google的一个暑期项目中由David Cournapeau开发并由800多位contributor合力开发出的机器学习库。

五、建模过程

（1）数据特征分析

总共爬取了39622条房源信息，爬取的特征如下：



其大部分的特征都是number类型，根据平时对于房价的影响因素的了解，目前认为比较重要的特征有房子大小，房子位置，房子的建造年份这几个特征。

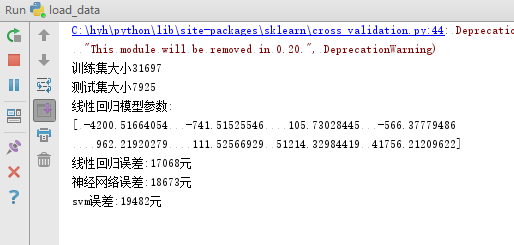
在训练时按序输入的特征如下：房子中的房间数，房子中的客厅数，房子的大小，房子的建造年份，房子的高度，房子所在那栋楼的高度，房子的经度，房子的纬度。一共八个特征。

（2）调参

在训练的过程中，根据训练出的结果中特征的重要程度不同进行调参，把不重要的特征删除并再次训练，直到剩下的特征都对结果影响很大。

（3）结果分析及模型对比

对于训练出的结果，线性回归模型，神经网络，svm的误差基本都在每平方米18000元左右，这个误差是比较大的。所以，这个房价的模型可能是非线性的，才会产生如此大的误差。

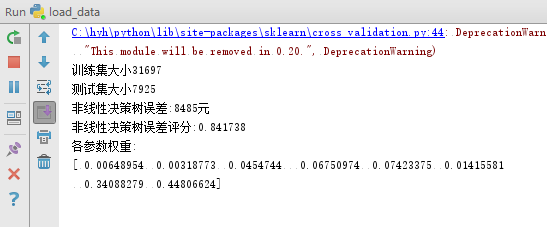


由上图可知，线性回归的表现较好，神经网络次之，svm最差。另外，从线性回归的模型参数来看，每多一个房间，每多一个客厅，房子建的越晚，都会导致房价变低，这显然是不符合常识的。所以这个房价模型应该是一个非线性模型。

六、非线性模型建模

（1）非线性决策树

使用了sklearn库中自带的DecisionTreeRegressor模型，经计算误差在8500元每平左右。评分达到了0.84（满分1分）

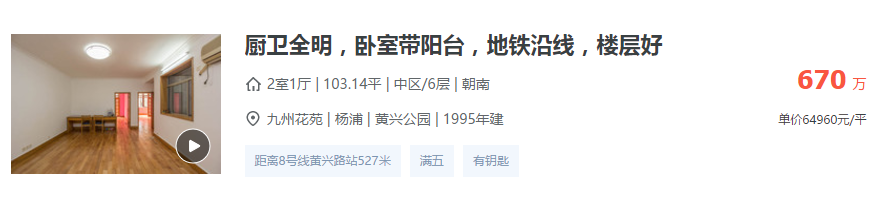


（2）结论

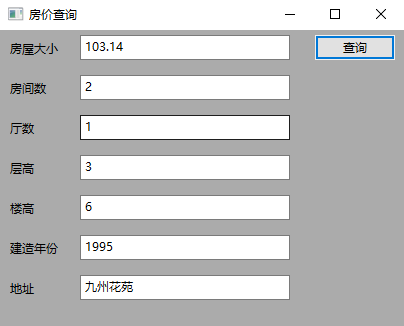
可以看出，在改用非线性模型之后，结果有了很大的改善。另外根据输出的模型参数权重，可以得知，房子的大小，建造年份，房子的高低以及房子所处的位置对房价有较大的影响，而房子的房型以及房子所在整栋楼的高度对房价的影响很小。另外，房子的经纬度对房价的影响非常大，说明在上海，市区和郊区的房价跨度很明显。

七、房价查询界面

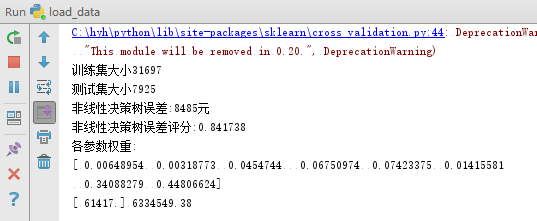
先在链家网上随意找一条房源信息



输入查询界面



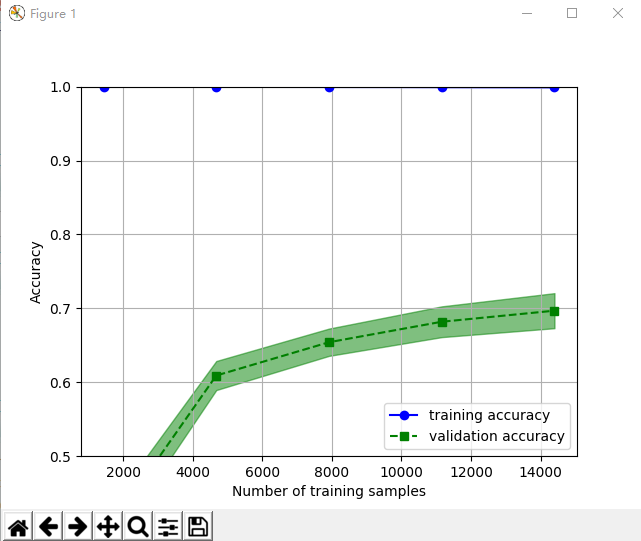
点击查询进行房价咨询



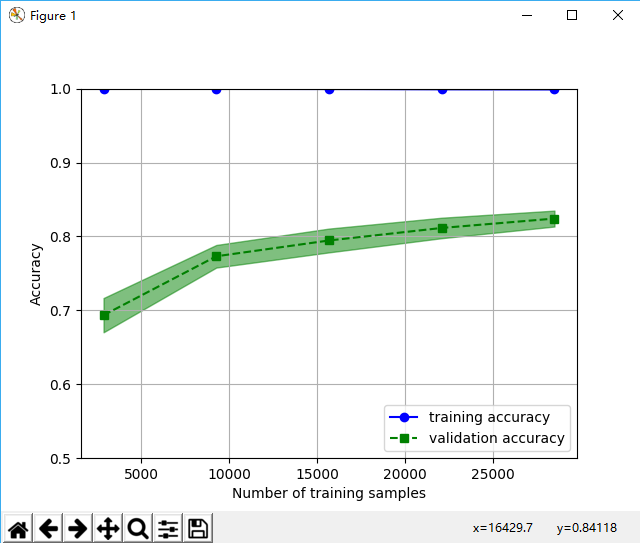
会输出房子的每平米价格及总价，可以看到，链家网上挂出的价格是每平米64960元，模型的预测结果为每平米61417元，总价相差40万，基本达到了帮助用户了解房价的需求。

八、改进措施

（1）收集更多数据



数据集在20000条以下时的训练学习曲线



数据集在40000条以下时的训练学习曲线

从上面的两张图可以看出，随着数据集变大，验证集的准确率不断上升，所以需要收集更多的训练数据来精化模型。目前的房价数据还不是非常的多，容易受到部分异常房价信息的影响

（2）寻找更多特征

影响房价的因素还有很多，比如房子周围是否交通便利，房子周围是否有大型商业圈，房子的装修好差，都对房价有影响。如果可以在房子的信息中加入这些特征，那么也可以提升模型精度。

附录

数据采集

*# coding:utf-8 ctrl+/ 注释***import** urllib2  
**from** bs4 **import** BeautifulSoup  
**import** sys  
**import** pymongo  
**import** requests  
  
**def** geocode(address):  
 parameters = {**'address'**: address, **'key'**: **'8ac4f59c23c73f503f350494ff9310d3'**,**'city'**:**'上海'**}  
 base = **'http://restapi.amap.com/v3/geocode/geo'  
 try**:  
 response = requests.get(base, parameters,timeout=1)  
 **except**:  
 **return** {}  
 answer = response.json()  
 **return** answer  
  
  
reload(sys)  
sys.setdefaultencoding(**"utf-8"**)  
  
connection = pymongo.MongoClient()  
tdb = connection.program  
post\_info = tdb.house  
  
  
*# 链家网d***def** find\_data(tmp\_url, tmp\_district, lists):  
 count = 0  
 *# 每个区的最大显示页数为100页* **for** page\_Num **in** range(1, 100):  
 f\_url = tmp\_url + tmp\_district + **"/d"** + str(page\_Num)  
 **print** f\_url  
 *# print f\_url* f\_page = urllib2.urlopen(f\_url)  
 f\_soup = BeautifulSoup(f\_page, **"html.parser"**)  
  
 page\_soup = f\_soup.find(class\_=**"m-list"**)  
 *# print page\_soup* ul\_soup = page\_soup.find(**'ul'**)  
 *# print ul\_soup* li\_list = ul\_soup.findAll(**'li'**)[0:]  
 *# print page\_Num* **for** tr **in** li\_list:  
 info = tr.findAll(class\_=**"info-row"**)[0:]  
 row1 = info[0].find(class\_=**"info-col row1-text"**).text  
  
 row1 = row1.strip()  
 row1 = row1.replace(**' '**, **''**)  
 *# print row1* cut\_1 = row1.index(**'|'**)  
 *# 几室几厅* house\_type = row1[0:cut\_1]  
 *#print house\_type* cut\_2 = row1[cut\_1 + 1:].index(**'平'**)  
 *# 房屋大小* house\_size = float(row1[cut\_1 + 1:cut\_1 + cut\_2 + 1])  
 *#print house\_size* **try**:  
 cut\_3 = row1.index(**'/'**)  
 **except** ValueError:  
 **continue** cut\_4 = row1.index(**'层'**)  
 *# 建筑总层高* building\_height = float(row1[cut\_3 + 1:cut\_4])  
 *#print building\_height* cut\_5 = row1.index(**'区'**)  
 *# 房屋层高* house\_height = row1[cut\_5 - 1:cut\_5]  
 *#print house\_height* row2 = info[1].find(class\_=**"info-col row2-text"**)  
 *# 均价* average\_price = info[1].find(class\_=**"info-col price-item minor"**).text.strip()  
 **print** average\_price  
 *# 位置* location = row2.findAll(**'a'**)[0:]  
  
 **try**:  
 year\_1 = row2.text.index(**'年建'**)  
 year = row2.text[year\_1 - 4:year\_1]  
 **except** ValueError:  
 **continue  
  
 print** year  
 *# 小区* housing\_estate = location[0].text  
 *#print housing\_estate  
 # 区县* house\_district = location[1].text  
 *#print house\_district* count = count + 1  
 *# print count  
 #  
 #  
 #整理出room和parlour数量* room\_1=house\_type.index(**'室'**)  
 room\_number=int(house\_type[0:room\_1])  
 parlour\_1=house\_type.index(**'厅'**)  
 parlour\_number=int(house\_type[room\_1+1:parlour\_1])  
  
 *#判断房子的具体高度* **if**(house\_height==**'中'**):  
 house\_height\_inlist=building\_height\*0.5  
 **elif**(house\_height==**'高'**):  
 house\_height\_inlist = building\_height \* 0.88  
 **elif**(house\_height == **'低'**):  
 house\_height\_inlist = building\_height \* 0.23  
  
 *#整理出具体房价* price\_1=average\_price.index(**'价'**)  
 price\_2=average\_price.index(**'元'**)  
 average\_price\_inlist=float(average\_price[price\_1+1:price\_2])  
  
 *#计算地址经纬度* address = house\_district+housing\_estate  
 **print** address  
  
 house\_first\_location=geocode(address)  
  
 **if "geocodes" not in** house\_first\_location.keys():  
 **print "no geocodes"  
 continue** house\_first\_location=house\_first\_location[**"geocodes"**]  
  
 *#print house\_first\_location* **if**(house\_first\_location==[]):  
 **continue** house\_location=house\_first\_location[0][**'location'**].encode(**'utf-8'**)  
 **print** house\_location  
 house\_location\_1=house\_location.index(**','**)  
 house\_location\_longtitude=float(house\_location[0:house\_location\_1])  
 **print** house\_location\_longtitude  
 house\_location\_latitude=float(house\_location[house\_location\_1+1:])  
 **print** house\_location\_latitude  
 *#house\_location\_latitude,house\_location\_longtitude,* list\_use = [room\_number,parlour\_number, house\_size, building\_height, house\_height\_inlist,float(year),house\_location\_longtitude,house\_location\_latitude,house\_district,address,  
 average\_price\_inlist]  
  
 *#print list\_use* lists.extend([list\_use])  
  
 **print "{\"户型\":\"%s\", \"大小\":\"%s\",\"楼高\":\"%s\",\"层高\":\"%s\",\"小区名\":\"%s\",\"市区\":\"%s\",\"均价\":\"%s\"}"** % (  
 house\_type, house\_size, building\_height, house\_height, housing\_estate, house\_district, average\_price)  
 data = {**"room\_number"**: room\_number, **"parlour\_number"**: parlour\_number,**"house\_size"**:house\_size,**"building\_height"**: building\_height,**"house\_height\_inlist"**: house\_height\_inlist,**"year"**: float(year),**"house\_location\_longtitude"**: house\_location\_longtitude,**"house\_location\_latitude"**:house\_location\_latitude,**"house\_district"**:house\_district,**"address"**:address,**"average\_price\_inlist"**: average\_price\_inlist}  
 post\_info.save(data)  
 *# print "end"  
 # print "all\_end"***def** use(district):  
 lists = []  
 **for** i **in** district:  
 find\_data(**'http://sh.lianjia.com/ershoufang/'**, i, lists)  
 **return** lists  
  
a = [**"pudongxinqu"**, **"minhang"**, **"baoshan"**, **"xuhui"**, **"putuo"**, **"yangpu"**, **"changning"**, **"songjiang"**, **"jiading"**, **"huangpu"**,  
 **"jingan"**, **"zhabei"**, **"hongkou"**, **"qingpu"**, **"fengxian"**, **"jinshan"**, **"chongming"**]  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 dataset = use(a)

数据训练

*# coding:utf-8 ctrl+/ 注释***import** sys  
**import** pymongo  
**import** numpy **as** np  
**import** price\_mongo  
  
reload(sys)  
sys.setdefaultencoding(**"utf-8"**)  
  
connection = pymongo.MongoClient()  
tdb = connection.program  
post\_info = tdb.house  
  
lists=[]  
count=0  
  
**for** item **in** post\_info.find():  
 count+=1  
  
 single\_item=[item[**"room\_number"**],item[**"parlour\_number"**],item[**"house\_size"**],item[**"year"**],  
 item[**"building\_height"**],item[**"house\_height\_inlist"**],item[**"house\_location\_longtitude"**],  
 item[**"house\_location\_latitude"**],item[**"average\_price\_inlist"**]]  
 lists.extend([single\_item])  
 *#print count***from** sklearn **import** metrics  
**from** sklearn **import** preprocessing  
**from** sklearn.linear\_model **import** LinearRegression  
**from** sklearn.cross\_validation **import** train\_test\_split  
  
\_array = np.array(lists)  
x = \_array[:, 0:8]  
*#normalized\_X=preprocessing.normalize(x)*y = \_array[:, 8]  
*#normalized\_Y=preprocessing.normalize(y)*X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y,test\_size=0.2)  
  
**print "训练集大小%d"**%(X\_train.shape[0])  
**print "测试集大小%d"**%(X\_test.shape[0])  
  
*#线性回归*clf = LinearRegression()  
clf.fit(X\_train, y\_train)  
**print "线性回归模型参数:"  
print** clf.coef\_  
y\_pred = clf.predict(X\_test)  
  
**print** (**"线性回归误差:%d元"**%(np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))))  
  
  
*#神经网络***from** sklearn.neural\_network **import** MLPRegressor  
kmodel = MLPRegressor(learning\_rate=**'adaptive'**,max\_iter=2000).fit(X\_train, y\_train)  
y\_kmodel\_pred = kmodel.predict(X\_test)  
**print** (**"神经网络误差:%d元"**%np.sqrt(metrics.mean\_squared\_error(y\_test, y\_kmodel\_pred)))  
  
*#svm模型***from** sklearn **import** svm  
smodel=svm.LinearSVR()  
smodel.fit(X\_train, y\_train)  
y\_smodel\_pred=smodel.predict(X\_test)  
  
**print "svm误差:%d元"**%np.sqrt(metrics.mean\_squared\_error(y\_test, y\_smodel\_pred))  
  
  
*#决策树模型***from** sklearn.tree **import** DecisionTreeRegressor  
dmodel=DecisionTreeRegressor()  
dmodel.fit(X\_train, y\_train)  
  
y\_dmodel\_pred=dmodel.predict(X\_test)  
**print "非线性决策树误差:%d元"**%np.sqrt(metrics.mean\_squared\_error(y\_test, y\_dmodel\_pred))  
**print "非线性决策树误差评分:%f"**%dmodel.score(X\_test,y\_test)  
  
**print "各参数权重:"  
print** dmodel.feature\_importances\_  
*# coding:utf-8 ctrl+/ 注释***import** wx  
app=wx.App()  
win =wx.Frame(None,title=**"房价查询"**,size=(420,335))  
win.Show()  
  
btn=wx.Button(win,label=**'查询'**,pos=(315,5),size=(80,25))  
  
  
house\_size\_input=wx.TextCtrl(win,pos=(80,5),size=(210,25))  
house\_size\_input\_label=wx.StaticText(win,label=**'房屋大小'**,pos=(10,10),size=(60,30))  
  
room\_number\_input=wx.TextCtrl(win,pos=(80,45),size=(210,25))  
room\_number\_input\_label=wx.StaticText(win,label=**'房间数'**,pos=(10,50),size=(60,30))  
  
parlour\_number\_input=wx.TextCtrl(win,pos=(80,85),size=(210,25))  
parlour\_number\_input\_label=wx.StaticText(win,label=**'厅数'**,pos=(10,90),size=(60,30))  
  
house\_height\_input=wx.TextCtrl(win,pos=(80,125),size=(210,25))  
house\_height\_input\_label=wx.StaticText(win,label=**'层高'**,pos=(10,130),size=(60,30))  
  
building\_height\_input=wx.TextCtrl(win,pos=(80,165),size=(210,25))  
building\_height\_input\_label=wx.StaticText(win,label=**'楼高'**,pos=(10,170),size=(60,30))  
  
year\_input=wx.TextCtrl(win,pos=(80,205),size=(210,25))  
year\_input\_label=wx.StaticText(win,label=**'建造年份'**,pos=(10,210),size=(60,30))  
  
address\_input=wx.TextCtrl(win,pos=(80,245),size=(210,25))  
address\_input\_label=wx.StaticText(win,label=**'地址'**,pos=(10,250),size=(60,30))  
  
  
**def** search(event):  
 hz = house\_size\_input.Value  
 rn=room\_number\_input.Value  
 pn=parlour\_number\_input.Value  
 hh=house\_height\_input.Value  
 bh=building\_height\_input.Value  
 yi=year\_input.Value  
 ai=address\_input.Value  
 location=price\_mongo.geocode(ai)[**"geocodes"**][0][**'location'**].encode(**'utf-8'**)  
 house\_location\_1=location.index(**','**)  
 house\_location\_longtitude=float(location[0:house\_location\_1])  
 house\_location\_latitude=float(location[house\_location\_1+1:])  
 list=[]  
 list.extend([[hz,rn,pn,hh,bh,yi,house\_location\_longtitude,house\_location\_latitude]])  
 out\_price=dmodel.predict(list)  
 **print** out\_price,float(out\_price[0])\*float(hz)  
 *#print hz,rn,pn,hh,bh,yi,house\_location\_longtitude,house\_location\_latitude*btn.Bind(wx.EVT\_BUTTON,search)  
  
app.MainLoop()