上海交通大學

网络空间安全学院

链家网房源房价预测模型

实验人员：515021910069　张　浩

515021910129　张晋华

515021910323　徐　源

515030910362　王晨奕

515030910141　翟拙存

5140219262　　李玮腾

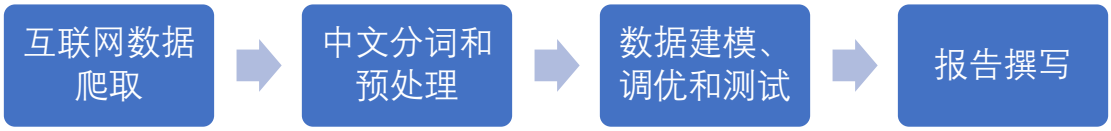
完成时间：　2018 年 6 月 24 日

目录

1 项目概述

1.1 内容综述

- 项目流程



1. 互联网数据爬取：从链家网爬取上海地区房源数据 ；
2. 中文分词和预处理：对爬取内容中包含的自然语言，做分词数据转换为结构化数据，并进行缺失值填充和数据清洗等操作；
3. 数据建模、调优和测试：建立房价预测模型，调优参数，并给出模型的性能；
4. 报告撰写：将主要流程撰写成文档，并上传代码和爬取的数据集。
5. 实现方式： Python

1.2 小组贡献

王晨奕、翟拙存：爬虫与预处理代码撰写，相关部分文档撰写

张浩：算法模型建立、调优和测试代码撰写，相关文档撰写

张晋华：文档主体撰写，文档修改与格式调整，算法调优

徐源：数据可视化分析代码，相关部分文档撰写

李玮腾：文档格式调整

2 数据爬取

2.1 运行环境

IDE：PyCharm、Spyder

解释器：python3.6

导入模块：

requests：python 实现的简单易用的 HTTP 库，主要用于请求目标网站。

parsel：Scrapy 包中的解析模块，可以方便地在网页内容中选择到需要的部分。主要使用 XPath 来解析网页源码，获取信息。

pandas：使用 dataFrame 数据结构进行数据的操作和写入文件。

matplotlib：一个 Python 的 2D 绘图库，它以各种硬拷贝格式和跨平台的交互式环境生成出版质量级别的图形。

openpyxl：excel 的支持库。

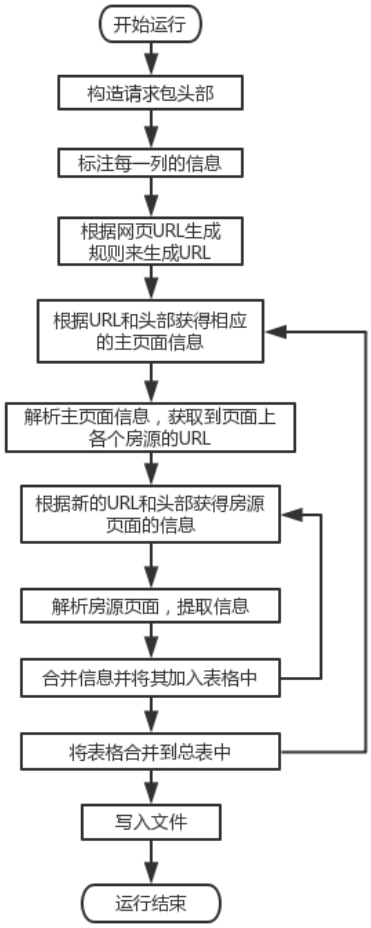
numpy：Python 的一种开源的数值计算扩展。这种工具可用来存储和处理大型矩阵，比 Python 自身的嵌套列表（nested list structure)结构要高效的多（该结构也可以用来表示矩阵（matrix））。

2.2 实现功能

- 根据不同区县，爬取链家网上海地区的房源信息：由于网站最大只显示一百页的信息，因此细化到每个区县，爬取每个区县所显示的一百页上的所有房源信息。
- 初步统一房源信息：每一条房源信息经过提取，被统一为以下格式：

  title，总价，均价，小区名，区域 1(所在区县)，区域 2(区县下的城镇)，房屋户型，所在楼层，建筑面积，户型结构，套内面积，建筑类型，房屋朝向，建筑结构，装修情况,梯户比例，配备电梯，产权年限，挂牌时间，交易权属，上次交易，房屋用途，房屋年限，产权所属，抵押信息，房本备件，特色标签，卖点
- 写入指定文件：最终所有信息被写入到指定的 excel 文件中
- 显示信息：在爬取过程中，提供必要的信息来帮助使用者了解爬虫的进展
- 使用 try，except 语句来防止爬取信息不规整，或无法爬去信息等错误

2.3 程序框图



1. 爬虫在爬取数据时，先获取主页面：

2. 解析主页面源码，根据主页面的源代码中提供的链接进入到各个房源页

面：

3. 再解析房源页面的源码，爬取到房源的信息。

2.4 信息爬取

爬取过程中，由于网页并非将房源信息统一存放，需要将信息从不同地方提取。

提取房源信息大标题：

```
title = new_sel.xpath\
        ('//div[@class="sellDetailHeader"]/div/div/div/h1/
/text()').extract()
title_column = ["title"]
```

提取房源总价，均价，小区名，区县，城镇，总价：

```
#爬取房源总价/均价(first)，小区名（second），区域（third）
profile_first = new_sel.xpath\
            ('//div[@class="overview"]/div[@class="content
"]\
              /div[@class="price "]//text()').extract()
profile_second = new_sel.xpath\
            ('//div[@class="overview"]/div[@class="content
"]\

              /div[@class="aroundInfo"]/div[@class="commun
ityName"]\
              /a[@class="info "]//text()').extract()
```

```
profile_third = new_sel.xpath
            ('//div[@class="overview"]/div[@class="content
"]\
              /div[@class="aroundInfo"]/div[@class="areaNa
me"]\
              /span[@class="info"]//text()').extract(
profile = [profile_first[0],profile_first[2],\
      profile_second[0],profile_third[0],profile_third[2]]
profile_column = ["总价","均价","小区名","区域 1","区域 2"]
```

其中均价是 profile_first 列表的第 0，2 两个元素，小区名是 profile_second 列表的首个元素，城镇，总价是是 profile_third 列表的第 0，2 两个元素。

提取房源基本信息：

```
basic_information_first = new_sel.xpath\
                    ('//div[@id="introduction"]/div/div/div
/div[@class="content"]/ul/li//text()').extract()
basic_information_second = [i.strip() for i in
basic_information_first]
basic_information_third = [i for i in
basic_information_second if i != '']
basic_information_column = []
basic_information = [
for i in basic_information_third:
  if(basic_information_third.index(i) % 2 == 0)
      basic_information_column.append(i)
  else:
      basic_information.append(i)
```

由 xpath 提取的信息是标签和属性值交替出现的列表，字符串可能出现空串或包含大量空格符，后续操作需要删除空串和空格符，并将标签和属性值分别存放在不同的列表中。

提取房源特色标签：

```
house_feature_label_first = new_sel.xpath\
                            ('//div[@class="introContent \
                              showbasemore"]/div[@class="t
ags clear"]\
                              /div[@class="content"]//text
()').extract()
```

```
house_feature_label_second = [i.strip() for i in
house_feature_label_first]
house_feature_label_third = [i for i in
house_feature_label_second if i != '']
house_feature_label = [''.join([i + ' ' for i in
house_feature_label_third])]
house_feature_label_column = ["特色标签"]
```

由 xpath 提取的字符串列表可能出现空串或在字符串中包含大量空格符，需要删除空串和空格符，并且将列表合并为一条字符串。

提取卖点：

```
#爬取卖点
house_feature_first = new_sel.xpath\
                  ('//div[@class="introContent
showbasemore"]/div[@class="baseattribute
clear"]//text()').extract()

house_feature_second = [i.strip() for i in
house_feature_first]
house_feature_third = [i for i in house_feature_second if
i != '']
house_feature = [''.join([i + ' ' for i in
house_feature_third])]
house_feature_column = ["卖点"]
```

原始字符串列表可能出现空串或在字符串中包含大量空格符，需要删除空串和空格符，并且将列表合并为一条字符串。

2.5 程序结果

最终所有的数据被存放入 house_info.xlsx 文件中，包含 39415 条房源信息数据，每个数据包含 28 个属性，包括：title，总价，均价，小区名，区域 1(所在区县)，区域 2(区县下的城镇)，房屋户型，所在楼层，建筑面积，户型结构，套内面积，建筑类型，房屋朝向，建筑结构，装修情况,梯户比例，配备电梯，产权年限，挂牌时间，交易权属，上次交易，房屋用途，房屋年限，产权所属，抵押信息，房本备件，特色标签，卖点。

3 数据预处理

3.1 预处理流程

刚爬到的数据有 28 个特征，其中有很多冗余特征，有很多缺失数据，还有很多数据格式不规整，不易处理。

“户型结构”和“套内面积”，其大多数数据为暂无数据，而产权年限大多数为未知故直接删除这三列。

“挂牌时间”与“上次交易时间”非房屋本身属性，而是交易属性，也暂且删除。

“房本备案”绝大多数为“已上传房本”，只有极少数未上传，参考价值不大，暂不予考虑。

“卖点”是一段卖主的个人描述，有“南北通透”，“采光充足”，“交通便利”，“随时看房”等高频出现的词汇，由于是卖家自述，难免有自卖自夸的嫌疑，主观性太强，暂不予考虑。用如下代码删除无用行。

```
# 删除无用行列
# col = [0, 2, 3, 5, 9, 10, 17, 18, 19, 20]
col = [0, 2, 3, 5, 9, 10, 17, 18, 19, 20, 22, 23,24]
data_raw = delete_column_row(data_raw, col)
```

处理“所在楼层”：

这一行的数据是这样的：

所在楼层
中楼层 (共 6 层)
低楼层 (共 6 层)
低楼层 (共 24 层)
低楼层 (共 17 层)
低楼层 (共 6 层)
中楼层 (共 11 层)
中楼层 (共 18 层)
中楼层 (共 5 层)
中楼层 (共 8 层)
中楼层 (共 18 层)
高楼层 (共 6 层)
高楼层 (共 13 层)
低楼层 (共 31 层)
中楼层 (共 18 层)

高楼层 (共 6 层)
中楼层 (共 6 层)
低楼层 (共 6 层)
低楼层 (共 14 层)
中楼层 (共 6 层)
低楼层 (共 14 层)
中楼层 (共 13 层)
中楼层 (共 6 层)

“中/低/高楼层”指的是该房相对整个楼的低中高，而括号里是整个楼的层数，考虑到相对高度与总高度的相关性，可以把总楼层按层数分为三个等级：0~6 层为低，6 到 12 层为中，12 层以上为高，相对高度与总高度组合在一起，形成：["低", "低"], ["低", "中"], ["低", "高"], ["中", "低"], ["中", "中"], ["中", "高"], ["高", "低"], ["高", "中"], ["高", "高"]共 9 种组合。空值，用众数，补全为低。

```
def floor_filter(data_raw):
    for i in data_raw.index:
        usr = data_raw.loc[i, "所在楼层"][0:1]
        total = re.findall(r"\d+", data_raw.loc[i, "所在
楼层"])
        total = int(total[0])
        if total <= 6:
            total = "低"
        elif total <= 12:
            total = "中"
        elif total < 300:
            total = "高
        else:
            total = "低"
        data_raw.at[i, "所在楼层"] = [usr, total]

    return data_raw
```

处理“建筑面积”：

这一列的数据较为规整，只是在描述建筑面积时没有使用方便建立数据模型的浮点数，而是以建筑面积数值加上单位“㎡”的字符串形式出现。我们要做的数据预处理工作就是去掉单位“㎡”，让这一行的数据成为仅仅标识面积值的浮点数即可。

```
def area_filter(data_raw):
    for i in data_raw.index:
        data_raw.loc[i, "建筑面积"] = data_raw.loc[i, "建筑面积
"].replace("㎡", "")

    # data_raw["建筑面积"] = data_raw["建筑面积
"].astype("float")
    return data_raw
```

处理“建筑类型”：

这一列的数据值包括“板楼”“塔楼”“暂无数据”，我们要做的数据预处理工作是填充“暂无数据”项。经过分析与讨论，我们发现绝大多数房屋的建筑类型为“板楼”，因此我们采用众数填充的方式将“暂无数据”项填充为“板楼”。

```
def type_filter(data_raw):
    for i in data_raw.index:
        if data_raw.loc[i, "建筑类型"] == "暂无数据":
            data_raw.loc[i, "建筑类型"] = "板楼"
    return data_raw
```

处理“房屋朝向”：

这一列的数据较为混乱，主要原因是卖家对于房屋的朝向定义不一致，有的简单采用南北划分，有的精确至东南、西北等具体方位，并且值不唯一，因为有些户型（比如别墅）有不止一个的朝向。我们将这一列数据处理为数值列表，其中阿拉伯数字 0-7 分别表示 8 个方向：″东″：0，″南″：1，″西″：2，″北″：3，″东南″：4，″西北″：5，″东北″：6，″西南″：7。对于未知数据的处理，我们仍然采用众数填充的方式，绝大多数的房屋朝向都是南，因此我们将未知数据填充为 1（“南”）。例：[0]表示房屋朝向为东，[0，2]表示房屋朝东且朝西。

```
def face_filter(data_raw):
    face_dic = {"东": 0, "南": 1, "西": 2, "北": 3, "东南": 4, "
西北": 5, "东北": 6, "西南": 7}
    for i in data_raw.index:
        face_dat = data_raw.loc[i, "房屋朝向"].split(' ')
        face_dat = [face_dic.get(fac, 1) for fac in face_dat]
        data_raw.at[i, "房屋朝向"] = face_dat

    return data_raw
```

处理“建筑结构”：

这一列数据初始值包括“砖混结构”“钢混结构”“未知结构”，数据预处理的工作同样为填充未知值。进过调查和对数据的分析，我们发现楼层较低的建筑多为砖混结构，而楼层较高的新建建筑多为钢混结构，因此我们对于未知数据的填充依据“所在楼层”项的处理结果来进行。对于楼层总数为“低”的房屋，我们将其填充为“砖混结构”，对于楼层总数为“高”的房屋，我们将其填充为“钢混结构”。

```
    def struc_filter(data_raw):
        for i in data_raw.index:
            if data_raw.loc[i, "建筑结构"] == "未知结构":
                if data_raw.loc[i, "所在楼层"][1] == "低"
                    data_raw.loc[i, "建筑结构"] = "砖混结构"
                else:
                    data_raw.loc[i, "建筑结构"] = "钢混结构"



        return data_raw
```

处理“梯户比例”：

这一列数据较为混乱，首先其没有规模较小的特征集合，几乎每一项数据的值都是特有的，其次它对于数字的描述方式为中文而非阿拉伯数字，这给我们的处理带来了较多困难。第一步，我们将字符串处理成一个含有两个元素的数组，即去掉“梯”“户”，并将表示电梯数和户数的字符串放进一个数组里；第二步，我们调用已经写好的中文转阿拉伯数字函数，将电梯数和户数从字符串类型转为浮点数类型；第三步，我们将计算好的梯户比例（电梯数/户数）代替原来的数据填入其中，并且设置小数点后保留两位，这样处理结果较为整齐。

```
def elev_filter(data_raw):
    for i in data_raw.index:
        elev_dat = re.split('[梯户]', data_raw.loc[i, "梯户比例
"])
        elev_dat = [chinese2digits(i) for i in elev_dat]
        data_raw.loc[i, "梯户比例"] = round(elev_dat[0] /
elev_dat[1],2)
```

```
    return data_raw
```

处理“配备电梯”：

这一列的数据初始值包括“有”“无”“未知”，数据预处理的工作同样为填充未知值。进过调查，结合实际经验，我们发现低楼层的建筑往往没有电梯，如老小区的 6 层居民楼，而中高楼层的建筑往往配备了电梯，如新建小区的几十层高的居民楼。因此，我们将“未知”项根据楼层总数预处理结果划分，总楼层低的填充为“无”，总楼层为中或高的填充为“有”。

```
def hase_filter(data_raw):
    for i in data_raw.index:
        if data_raw.loc[i, "配备电梯"] == "暂无数据":
            if data_raw.loc[i, "所在楼层"][1] == "低":
                data_raw.loc[i, "配备电梯"] = "无"
            else:
                data_raw.loc[i, "配备电梯"] = "有"

    return data_raw
```

处理“特色标签”与“卖点”：

进过初步分析，我们发现这两列数据主观色彩很重，多为业主对于房屋的夸赞，我们想要从中获得的较为客观的信息是交通方面的信息，因此我们对这两列数据做这样的处理：若在特色标签中含"地铁"，或者在卖点中含"地铁""交通""线""路""近"等字样的，最后的字符串为"交通便利"，否则最后返回的字符串为"未说明"。

```
def dist_filter(data_raw):
    for i in data_raw.index:
        dist_label = str(data_raw.loc[i, "特色标签"]).split('
')
        #print(type(data_raw.loc[i, "卖点"]))

        if "地铁" in dist_label or \
           "地铁" in str(data_raw.loc[i, "卖点"]) or \
            "交通" in str(data_raw.loc[i, "卖点"]) or \
            "线" in str(data_raw.loc[i, "卖点"]) or \
            "路" in str(data_raw.loc[i, "卖点"]) or \
            "近" in str(data_raw.loc[i, "卖点"]):
```

```
            data_raw.loc[i, "卖点"] = "交通便利"
        else:
            data_raw.loc[i, "卖点"] = "未说明"

    data_raw = data_raw.drop(data_raw.columns[-2], 1
    return data_raw
```

根据之前的数据预处理，现在的数据已经是规整的了，可以把数据的类型分为以下几种：“独热码”型，布尔型，数值型。独热码型有 ：区域、所在楼层、建筑类型、建筑结构、装修情况、房屋用途 6 个，可以直接调用 sklearn 提供的预处理模块中的 LabelEncoder 将字符串转化为数字，再用 sklearn 提供的 OneHotEncoder 将数值转化为独热码。

```
            class_le = LabelEncoder()
            for i in range(6):
                data[:, i] = class_le.fit_transform(data[:, i])

        ..........

    ohe = OneHotEncoder(categorical_features=list(range(5)))
    x=ohe.fit_transform(new_data).toarray()
```

布尔型两个量是有无电梯和交通是否便利（根据是否靠近地铁），可以直接循环赋值：

```
    data[:,6]=[ 1*(data[i,6]=='有') for i in range(data_num) ]
    data[:,7]=[ 1*(data[i,7]=='交通便利') for i in
range(data_num) ]
```

数值型有“建筑面积”和“梯户比例”，不作处理。

有两个需要特殊处理的量，“房屋户型”和“房屋朝向”：

房屋户型是类似这样的数据，“2 室 2 厅 1 厨 1 卫”，直接将其分裂成 4 个数值型变量，每个变量表示相应房间的个数即可。

房屋朝向共八个:东、南、西，北，东南、东北、西南、西北，现在的数据用 0~7 的整数编码每个方向，每个房子可以具有八个方向中的任意个方向，比如[1, 3]表示房子具有南、北两个朝向。想要把它处理成 8 维的布尔数组，每个维度表示是否具有这个朝向，比如[1,3]经过处理后为[0,1,0,1,0,0,0,0],可以通过以下代码实现：

```
for i in range(data_num):
……
    for j in range(8):
        new_data[i,13+j]=1*(data[i,1].find(str(j))!=-1)
```

至此，已经得到矩阵 x 和数组 y，x 是每个房子的各种属性，y 是房价。

然后要分割训练集与测试集。由于爬取的时候，数据时有序的，需要先打乱数据顺序，再按一定比例分割，代码如下：

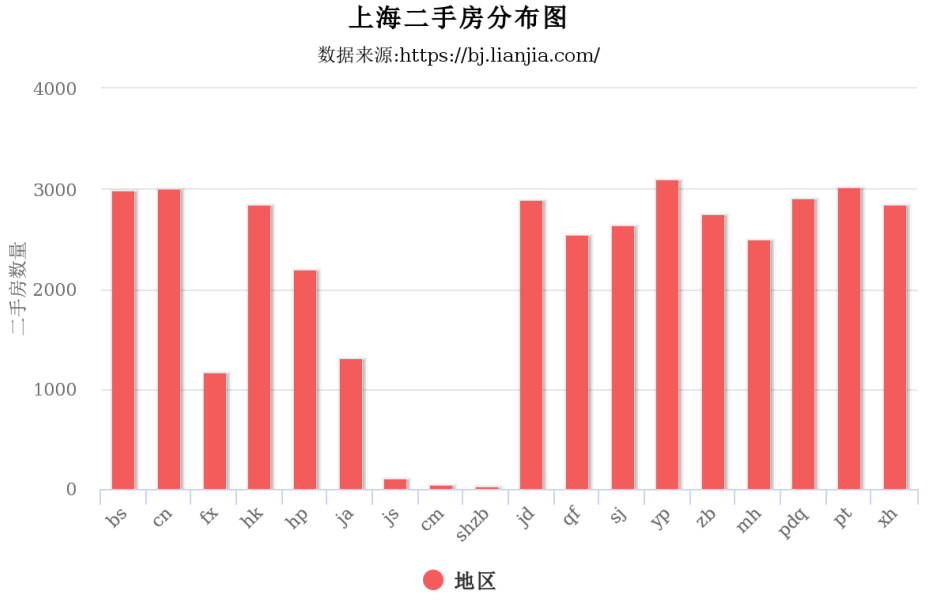
```
def my_shuffle(x,y,train_set_rate):
    assert x.shape[0]==y.shape[0
    data_num=y.shape[0]
    index=list(range(data_num))
    random.seed(a=None, version=2)
    random.shuffle(index)
    x=x[index]
    y=y[index]
    rate=train_set_rate
    X_train=x[:int(data_num*rate)]
    X_test=x[int(data_num*rate):]
    Y_train=y[:int(data_num*rate)]
    Y_test=y[int(data_num*rate):]
    return X_train,Y_train,X_test,Y_test
```

至此，已经得到了可用有效的训练数据集与测试数据集。

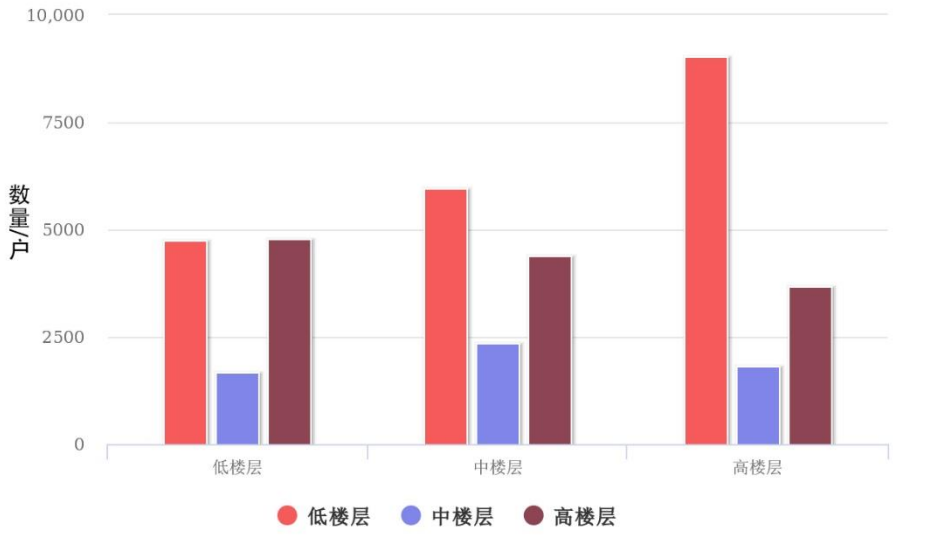
3.2 数据分析

在获取到有效的数据集之后，我们对数据的整体和局部规律进行了一系列解析：

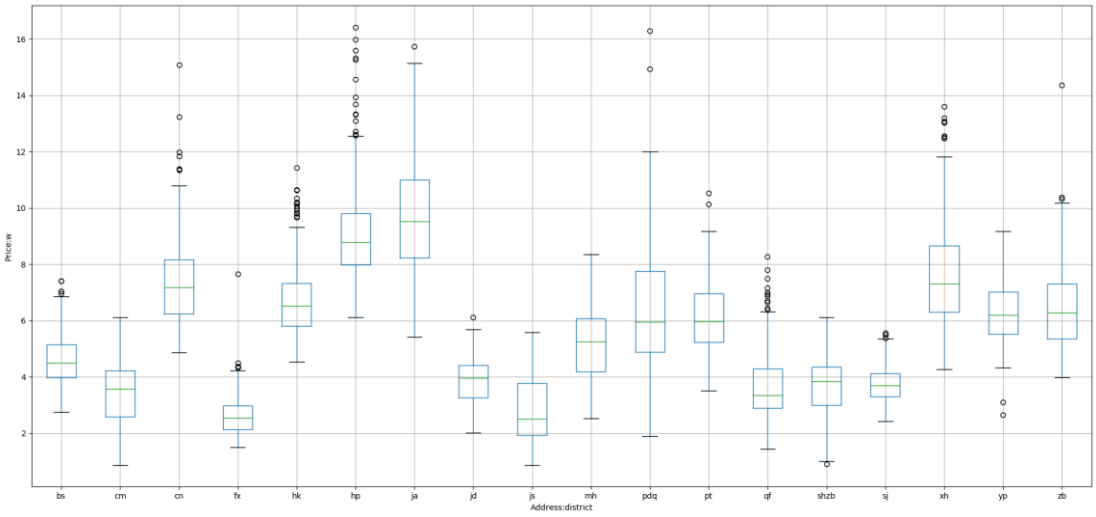
1. 二手房地区分布（各区名字取拼音缩写，其中浦东 pd 和函数名冲突，修正为 pdq（浦东区））：可以看出来，金山、崇明、上海周边三处二手房数目很少，几十到几百不等，奉贤有一千余户二手房，其他地区均在三千户左右。



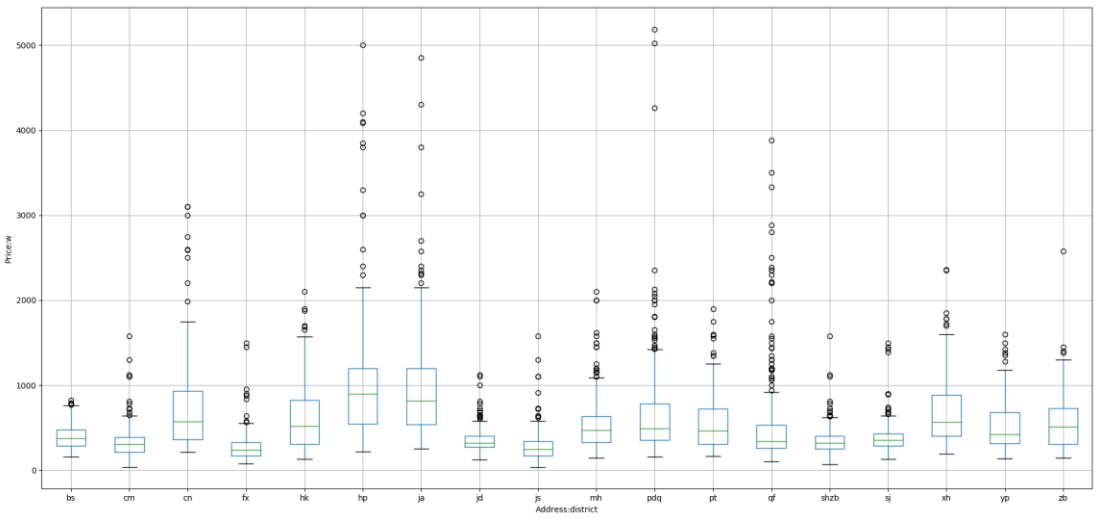
2. 二手房楼层分布：楼层不表示为包含两个元素的字符列表，包括["低"，"低"]、["低"，"中"]、["低"，"高"]、["中"，"低"]、["中"，"中"]、["中"，"高"]、["高"，"低"]、["高"，"中"]、["高"，"高"]，第一个元素表示楼层总高度，第二个表示所出售二手房相对总楼层数的相对高度。可以看出来无论楼层总高度如何，相对高度偏低和偏高的二手房居多，高度适宜的二手房较少，和实际情况相符，因为楼层适合的房子，一般不会进行抛售。



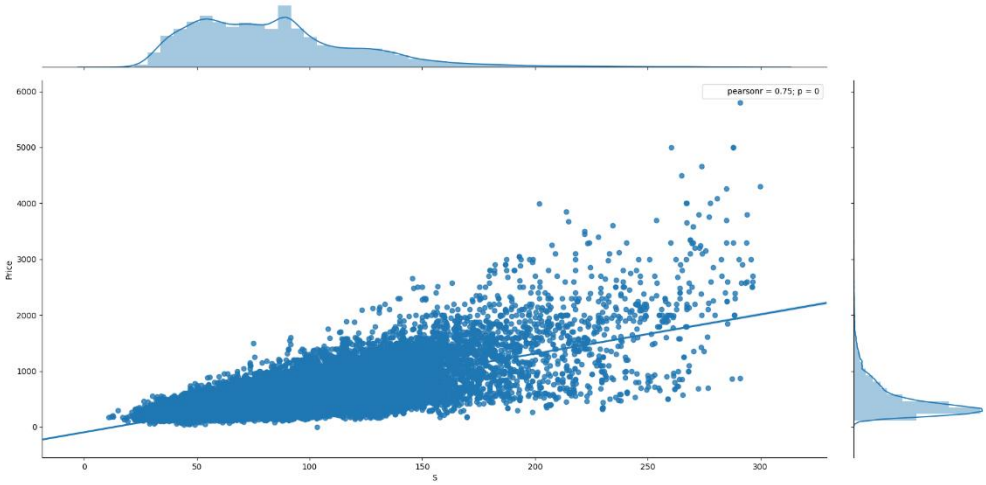
3. 每平米房价的箱线图：箱线图要求每个区的阳光本数量一致，故确立每个区的样本数量为 300 对于热手房数量不足三百的三个区，采取元素复制的方法扩充到三百个。由下图可知，每个区的箱线图均存在少量异常数据（房价奇高或者奇低，不过数量很少，影响不大）。经观察比较：上海二手房房价处于 2 万——10 万每平米，主要集中在 6 万左右，其中静安区平均价格最高，但是样本不集中，比较分散，跨度大。



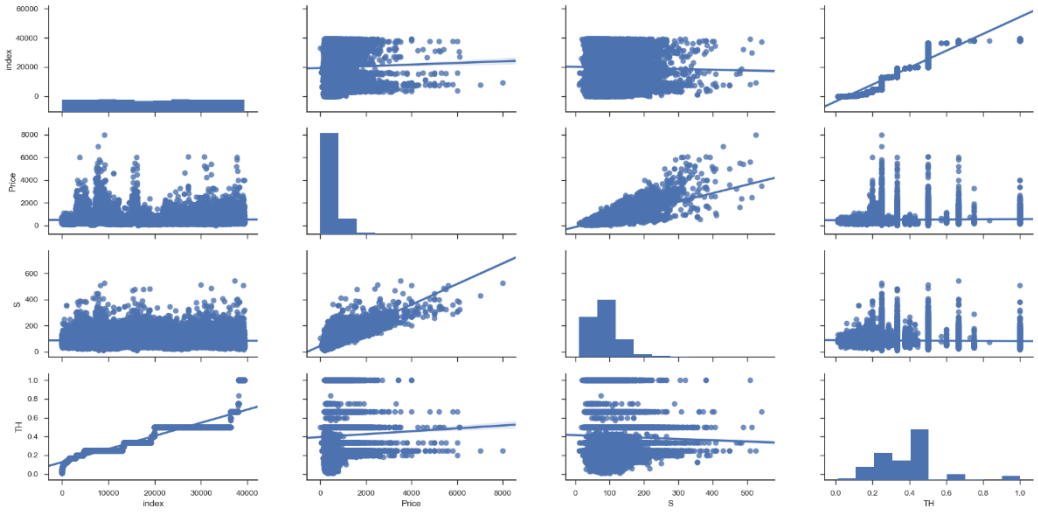
4. 二手房总价的箱线图：考虑面积因素，房子总价大都处于 500 万左右，其中静安、黄埔售价最高。宝山、长宁、奉贤、嘉定、金山房屋总价集中趋势较好。



5. Jointplot 回归图：针对上海所有二手房总价和面积作线性回归分析，pearson 相关系数为 0.75，房子总价和面积呈现清晰的线性关系，同时可看出房价和面积的分布情况。



6. Pairplot 多变量图：揭示了 index（序号）、Price、S、TH（楼户比例）两两之间的二元关系。其中二手房总价和面积的线性关系最为明显。



4 算法建模及评估

4.1 算法选取

我们初步共选取了共七种算法来建立模型，分为 3 大类：

广义线性模型：

LinearRegression 线性回归,

Lasso,LassoCV,LassoLarsCV，

BayesianRidge 贝叶斯岭回归，

PassiveAggressiveRegressor 被动攻击算法回归

集成方法：

RandomForest 随机森林

神经网络模型：

MLP 多层感知器

通过对以上算法的建模实现初步找出合适的算法，并针对性进行模型调优、模型评估实现模型优化，并进行分析。以下对每一项算法的具体实现进行介绍和展现。

每种方法的预测结果输出见 result 文件夹。

4.2 评估方式

本次作业中采用 R-squared 评估预测结果的准确性。

R-squared，又称决定系数，在统计学中用于度量因变量的变异中可由自变量解释部分所占的比例，以此来判断统计模型的解释力。

对于简单线性回归而言，决定系数为样本相关系数的平方。当加入其他回归自变量后，决定系数相应地变为多重相关系数的平方。

其计算公式如下

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

右边的分子的预测值与实际值的方差，而分母是实际值与其平均值之间的方差，或者说按其平均值来预测的方差，即：

$$SS_{total} = \sum_i \left(y_i - \bar{y}\right)$$

$$SS_{residual} = \sum_i \left(f_i - \bar{y} \right)$$

4.3 LinearRegression 线性回归

线性回归（Linear Regression）是利用称为线性回归方程的最小平方函数对一个或多个自变量和因变量之间关系进行建模的一种回归分析。通过拟合一个带有系数 w = (w_1, ..., w_p) 的线性模型，使得数据集实际观测数据和预测数据（估计值）之间的残差平方和最小。

数学算法描述如下：

$$\min_w ||Xw - y||_2^2$$

算法优点：简单，适合数值型和标称数据

算法缺点：对非线性数据拟合结果不好

用线性拟合的方式预测：

```
reg = LinearRegression()
reg.fit(X_train,Y_train)
predictions = reg.predict(X_test)
for i, prediction in enumerate(predictions):
    print('Predicted: %s, Target: %s' % (prediction,
Y_test[i]))
print('R-squared: %.2f' % reg.score(X_test, Y_test))
```

测得的 R-squared 值为 0.71

预测结果与真实结果的对比如下

```
Predicted: 654.3359375, Target: 525.0
Predicted: 334.6171875, Target: 362.0
Predicted: 1127.8828125, Target: 1350.0
Predicted: 383.3671875, Target: 385.0
Predicted: -8.046875, Target: 180.0
Predicted: 250.609375, Target: 305.0
Predicted: 185.671875, Target: 225.0
Predicted: 258.3984375, Target: 208.0
Predicted: 446.53125, Target: 245.0
```

```
Predicted: 741.484375, Target: 745.0
Predicted: 55.203125, Target: 230.0
Predicted: 407.421875, Target: 420.0
Predicted: 367.9140625, Target: 335.0
Predicted: 961.6171875, Target: 980.0
Predicted: 614.7109375, Target: 380.0
Predicted: 271.5234375, Target: 180.0
Predicted: 1224.734375, Target: 770.0
Predicted: 630.1328125, Target: 560.0
Predicted: 244.4921875, Target: 290.0
Predicted: 446.625, Target: 410.0
Predicted: 210.75, Target: 250.0
Predicted: 252.9921875, Target: 380.0
Predicted: 228.46875, Target: 210.0
Predicted: 305.9921875, Target: 290.
Predicted: 474.84375, Target: 450.0
Predicted: 992.828125, Target: 1160.0
......
R-squared: 0.71
```

发现一个严重问题是在第 5 行出现了预测结果，而在现实中，房价是不可能小于零的。

4.4 Lasso 套索回归

Lasso 是估计稀疏系数的线性模型。 它在一些情况下是有用的，因为它倾向于使用具有较少参数值的情况，有效地减少给定解决方案所依赖变量的数量。 因此，Lasso 及其变体是压缩感知领域的基础。 在一定条件下，它可以恢复一组非零权重的精确集。在数学公式表达上，它由一个带有 L1 先验的正则项的线性模型组成。 其最小化的目标函数是:

$$\min_{w} \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha||w||_1$$

Lasso 回归与 Ridge 回归相似的是，也会惩罚回归系数的绝对值，但不同的是使用的惩罚函数是绝对值而不是平方。这导致惩罚值是一些参数估计结果接近为零，所以需要从 n 个特征值中选择变量。实际效果是如果有一组高度相关的特征量，Lasso 会从中选取一个变量，其他的收缩为 0

LassoLars 是一个使用 LARS 算法的 lasso 模型，不同于基于坐标下降法的实现，它可以得到一个精确解，也就是一个关于自身参数标准化后的一个分段线性解。

用 Lasso 回归的方式预测：

```
reg= LassoLarsCV()
reg.fit(X_train,Y_train)
predictions = reg.predict(X_test)
for i, prediction in enumerate(predictions):
    print('Predicted: %s, Target: %s' % (prediction,
Y_test[i]))
print('R-squared: %.2f' % reg.score(X_test, Y_test))
```

测得的 R-squared 值为 0.73，效果比线性回归有明显提升，但还是存在负值，而且效果还是不够好。

4.5 BayesianRidge 贝叶斯岭回归

BayesianRidge 利用概率模型估算了回归问题，先验参数 w 由下面的球形高斯给出：

$$p(w|\lambda) = \mathcal{N}(w|0, \lambda^{-1}\mathbf{I_p})$$

算法优点：能根据已有的数据进行改变，能在估计过程中引入正则项

算法缺点：推断过程是非常耗时的

预测的代码：

```
reg=BayesianRidge()
reg.fit(X_train,Y_train)
predictions = reg.predict(X_test)
for i, prediction in enumerate(predictions):
    print('Predicted: %s, Target: %s' % (prediction,
Y_test[i]))
print('R-squared: %.2f' % reg.score(X_test, Y_test))
```

测得的 R-squared 值为 0.68，效果逊于前面两种方法，经过调节参数后，

最多达到 0.7,所以不做详细分析了。

4.6 PassiveAggressiveRegressor 被动攻击算法回归

被动攻击算法是大规模学习的一类算法。和感知机类似，它也不需要设置学习率，不过比感知机多出一个正则化参数 C 。

预测的代码：

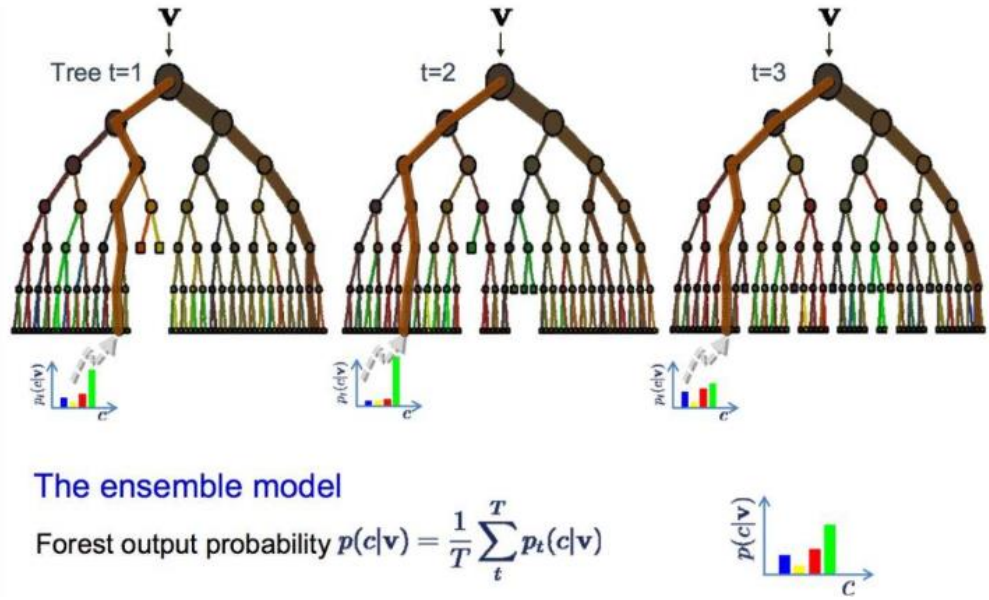
```
reg = PassiveAggressiveRegressor(random_state=0)
reg.fit(X_train,Y_train)
predictions = reg.predict(X_test)
for i, prediction in enumerate(predictions):
    print('Predicted: %s, Target: %s' % (prediction,
Y_test[i]))
print('R-squared: %.2f' % reg.score(X_test, Y_test))
```

测得的 R-squared 值为 0.65，也不予详细讨论。

4.7 RandomForest 随机森林

随机森林是一个包含多个决策树的分类器。在随机森林中集成模型中的每棵树构建时的样本都是由训练集经过有放回抽样得来的。 另外，在构建树的过程中进行结点分割时，选择的分割点不再是所有特征中最佳分割点，而是特征的一个随机子集中的最佳分割点。

由于这种随机性，森林的偏差通常会有略微的增大（相对于单个非随机树的偏差），但是由于取了平均，其方差也会减小，通常能够补偿偏差的增加，从而产生一个总体上更好的模型。



在 scikit-learn 中，RF 的分类类是 RandomForestClassifier，回归类是 RandomForestRegressor， 分类类 ExtraTreesClassifier，这里我们选择了回归类随机森林

算法优点：常在强分类器和复杂模型上使用时表现的很好，它能够处理很高维度的数据，并且不用做特征选择

算法缺点：随机森林已经被证明在某些噪音较大的分类或回归问题上会过拟，对于有不同取值的属性的数据，取值划分较多的属性会对随机森林产生更大的影响。

算法调优：

RF 决策树参数：

RF 划分时考虑的最大特征数 max_features：一般用默认的"auto"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。

决策树最大深度 max_depth: 模型样本量多，特征也多的情况下，需要限制这个最大深度，具体的取值取决于数据的分布。常用的可以取值 10-100 之间。

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

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。

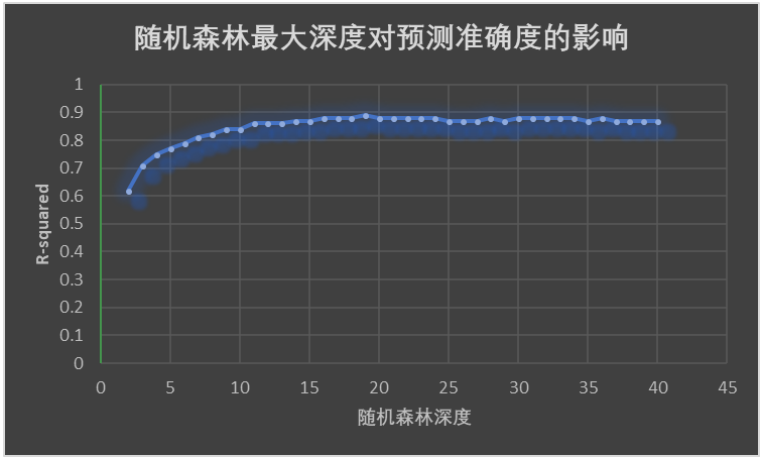
本次实验中针对随机森林的决策树深度做了调优测试

预测的代码：

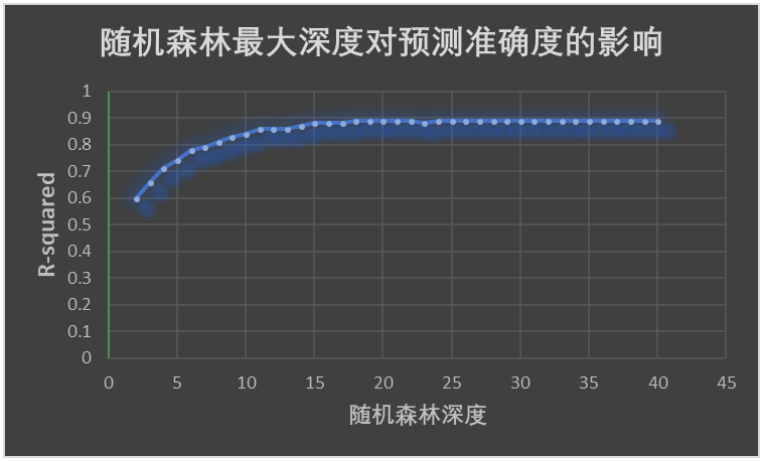
```
reg =RandomForestRegressor(max_depth=10, random_state=0)
reg.fit(X_train,Y_train)
predictions = reg.predict(X_test)
for i, prediction in enumerate(predictions):
    print('Predicted: %s, Target: %s' % (prediction,
Y_test[i]))
print('R-squared: %.2f' % reg.score(X_test, Y_test))
```

改变随机森林的深度，R-squared 变化如下图：

训练集占比 0.99 时：

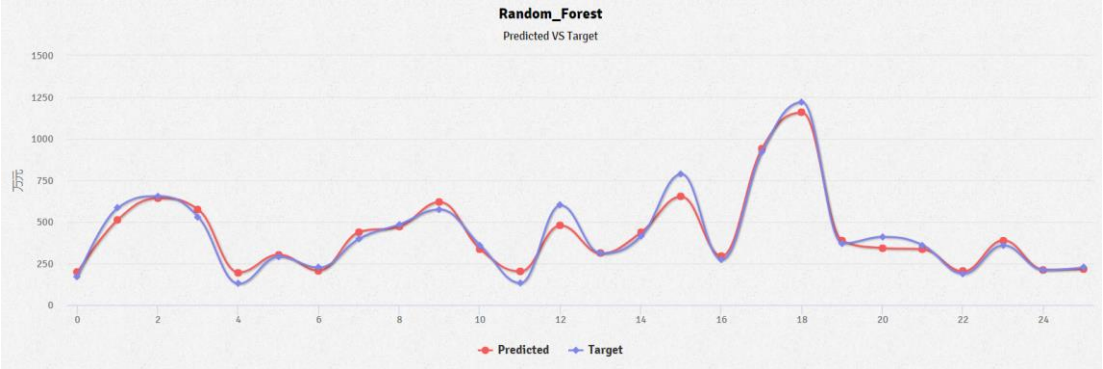


训练集占比 0.90 时：



发现最大深度从 2 增加到 40 的过程中，R-squared 值先增加，到 15 的时候，R-squared 基本稳定。

下面是部分预测结果与真实结果的对比示意图（取了 25 个点）：



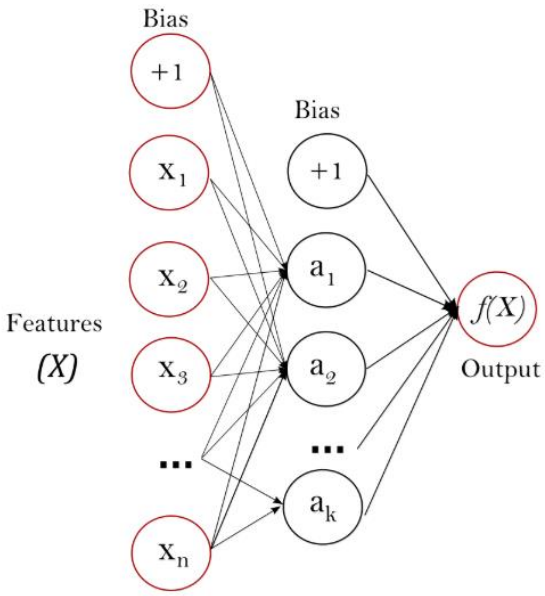
可以看到，预测结果与真实结果基本吻合.

4.8 MLP 多层感知器

多层感知器(MLP) 是一种监督学习算法，通过在数据集上训练来学习函数

$$f(\cdot): R^m \rightarrow R^0$$

，其中 m 是输入的维数，o 是输出的维数。 给定一组特征 X = {x_1, x_2, ..., x_m} 和标签 y ，它可以学习用于分类或回归的非线性函数。 与逻辑回归不同的是，在输入层和输出层之间，可以有一个或多个非线性层，称为隐藏层。



MLP 使用 Stochastic Gradient Descent（随机梯度下降）(SGD), Adam, 或者 L-BFGS 进行训练。 随机梯度下降（SGD） 使用关于需要适应的一个参数的损失函数的梯度来更新参数，

$$w \leftarrow w - \eta(\alpha\frac{\partial R(w)}{\partial w} + \frac{\partial Loss}{\partial w})$$

其中 w 是控制训练过程参数更新步长的学习率（learning rate）。 Loss 是损失函数（loss function）。

算法优点：可以学习得到非线性模型，使用``partial_fit`` 可以学习得到实时模型(在线学习)

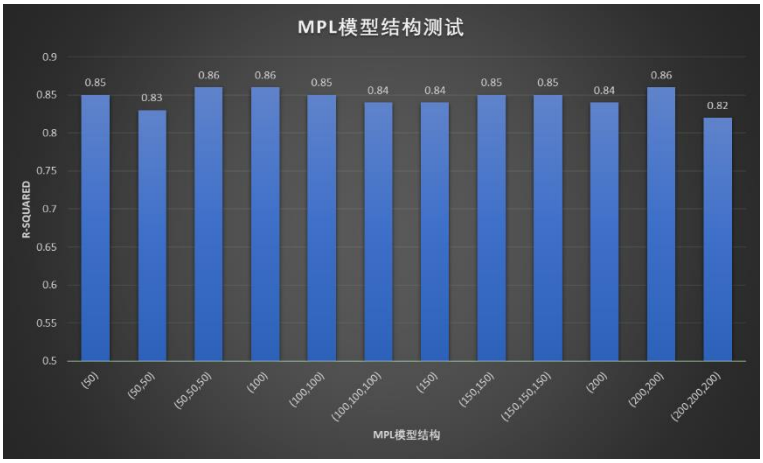
算法缺点：具有隐藏层的 MLP 具有非凸的损失函数，它有不止一个的局部最小值。 因此不同的随机权重初始化会导致不同的验证集准确率，MLP 对特征归一化很敏感.

代码如下：

```
16   reg = MLPRegressor(hidden_layer_sizes=(100))
 25  reg.fit(X_train,Y_train)
 26  predictions = reg.predict(X_test)
 27  for i, prediction in enumerate(predictions):
 28      print('Predicted: %s, Target: %s' % (prediction,
     Y_test[i]))
 29  print('R-squared: %.2f' % reg.score(X_test, Y_test))
```

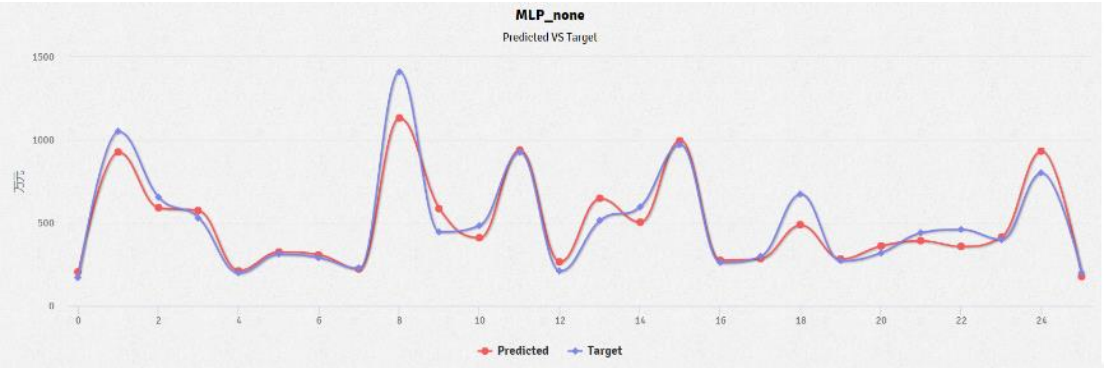
训练集占比 0.99 时，MLP 模型结构测试结果如下图：

训练集占比 0.90 时，MLP 模型结构测试结果如下图：



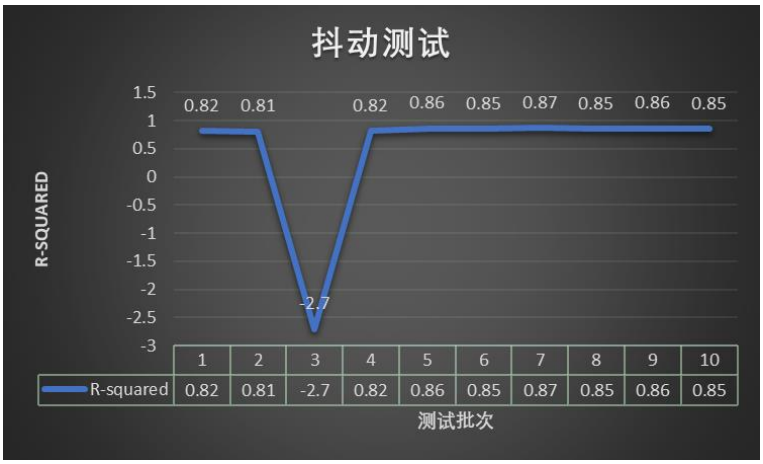
横坐标是个元组，表示隐藏层的神经元个数，比如（50,50,50）表示有 3 个隐藏层，每个隐藏层的神经元个数分别为 50,50,50。从测试结果中可以看出不同结构的 R-squared 差异不大，都能比较准确地预测房价，但是由于存在误差，不能说明哪种结构最优。

下面是 MLP 预测的部分结果与真实结果的对比图：



可见预测值与真实值极其接近。

由于 MLP 模型不太稳定，进行如下的抖动测试：

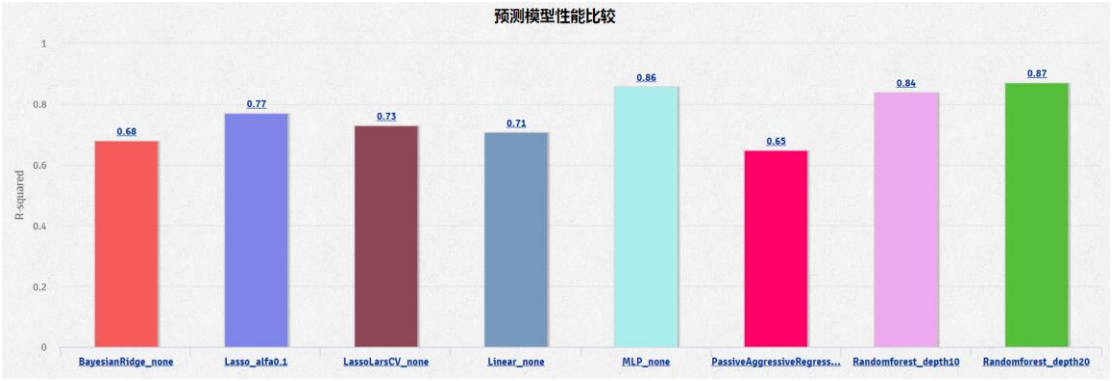


用多层感知机模型（MLP）在训练集比例为 0.9 的条件下，测试 10 次，每次重新打乱数据顺序，即每次的训练集、测试集都不同。实验发现，在第三次测试的时候，出现了严重的抖动，观察预测结果，发现并没有出现重大反常，分析原因后发现这一组的测试集的“房价”属性大多集中在 600 左右，比较接近，导致数据的方差本来就比较小，再加上预测结果不够准确，造成了较小的 R-squared。

4.9 小结

采用六种算法分别进行房价预测后，对其效果进行对比：

横坐标名称的含义是“算法_参数调整”，如 Lasso_alfa0.1 表示用 Lasso 回归，alfa 参数设为 0.1，none 表示没有调整参数。



可以发现，MLP 和随机森林的预测效果明显好于其他几种方法。

根据前面对这两种方法的测试得知，MLP 有抖动，而且参数较难调整，难以掌握规律，而随机森林的预测效果基本与最大深度成正相关。从训练时间的角度

分析，MLP 训练一次时间平均为 51.66s，而随机森林训练一次平均 1.55s。所以综合来看，对于本次房价预测，随机森林优于 MLP，以及上面使用的其他算法。

5 总结评价

本次实验项目共爬去数据量 39416 条，初步特征字段 28 个，包含文字描述，数据，关键字等多种数据。经过数据处理，形成有效结构化数据 38498 条，有效特征属性 14 个，包括数值和文字分析提取特征。采用 6 种 3 大类回归算法建立模型进行比较，以集成模型随机森林和神经网络多层感知机算法为主，进行了模型调优和评估。实现了性能良好的房价预测模型。

优势：数据集非常丰富，非常多元，近 4 万条数据，而且每条数据都有 28 个特征量；对每个特征量一一进行处理，每个特征量都经过了周密的考虑与详细的讨论；训练结果较好。

反思：训练集占比较大的情况才能训练出较好的结果；MLP 训练耗时太长，无法尝试足够多的模型结构，以选择最优结构。

感悟这次作业题目很有意义，让我们体验了一整个处理数据的流程，从数据爬取，到每个数据的分类与特征的提取。再到算法的选取与比较，参数的调优与模型的测试。虽然有 sklearn 提供的成熟的库函数，可以方便的调用，但是在选取算法与调试参数的过程中，我们对算法底层原理的理解也更加深入了。

整个过程充满挑战而且十分有趣，尤其是数据预处理和算法的选取。数据预处理需要把数据处理成可以供算法直接使用的数值，这非常考验逻辑思维，需要让一个数据最大化地暴露其本质：是处理为独热码还是数值或是要从大段文字中分词，选取对结果有较大影响的词汇来代表这个特征；而算法选取部分，要充分理解这个算法的底层原理和特点，了解它的使用场景，代入到题目中来。