基于集成学习的房价预测

Abstract：随着国民经济的发展，人们对房产的需求不断增加，对房价的关注度也越来越高。房价受到很多因素的影响，不易预测。传统的机器学习算法存在精度低、容易过拟合等问题。针对这些问题，本文使用集成学习算法对房价进行预测，选用 Bagging、Random Forest、AdaBoost、GBDT 算法在加州房价数据集上进行对比，实验结果表明 Random Forest 与 GBDT 算法的预测效果较好，均方误差分别为 49457.318、49796.238，具备一定的实际应用价值。

1 背景

随着国民经济的发展，人们对房产的需求不断增加，对房价的关注度也越来越高。房价受到很多因素的影响，不易预测。因此需要一种有效的算法帮助人们购房。

机器学习[2]是一门多学科交叉专业，涵盖概率论知识，统计学知识，近似理论知识和复杂算法知识，使用计算机作为工具并致力于真实实时的模拟人类学习方式，并将现有内容进行知识结构划分来有效提高学习效率。机器学习的任务分为分类、回归、聚类、降维四部分，房价预测属于回归问题。传统的回归算法有线性回归、决策树回归、SVR 以及 KNN 回归。这些算法存在预测精度不理想、数据噪声大时容易过拟合等问题，不够成熟。为此本文使用集成学习算法对房价进行预测。

在机器学习的有监督学习算法中，我们的目标是学习出一个稳定的且在各个方面表现都较好的模型，但实际情况往往不这么理想，有

时我们只能得到多个有偏好的模型（弱监督模型，只在某些方面表现的比较好）。集成学习[3]就是组合多个弱监督模型以得到一个更好更全面的强监督模型，即便某一个弱分类器得到了错误的预测，其他的弱分类器也可以将错误纠正回来。本文选用 Bagging、Random Forest、AdaBoost、GBDT 算法进行房价预测，在包含 20640 个样本的加州房价数据集上进行训练，并对每个模型进行参数调优。最后通过实验对比展现了各个算法的预测能力。

文章的第二部分介绍了算法模型，包括传统的线性回归、决策树回归、SVR、KNN 回归算法，以及 Bagging、Random Forest、AdaBoost、GBDT 等集成学习算法；第三部分是我们的实验，包括数据导入、测试集划分、可视化分析、数据预处理、模型训练（交叉验证）、模型调优、模型评估对比；第四部分是文章的总结。

2 模型

2.1 线性回归

当能够用一个直线较为精确地描述数据之间的关系，可以用线性回归做数据预测。

假设有 m 个样本，每个样本有 n 个特征和一个标签，形式如下：

$$\left(x_1^{(0)},x_2^{(0)},...,x_n^{(0)},y_0\right),\left(x_1^{(1)},x_2^{(1)},...,x_n^{(1)},y_1\right),...,\left(x_1^{(m)},x_2^{(m)},...,x_n^{(m)},y_m\right)$$

线性回归的模型如下：

$$h_\theta(x_1,x_2,...,x_n)=\theta_0+\theta_1x_1+...+\theta_nx_n$$

我们要做的就是找到参数$(\theta_1,\theta_2,...,\theta_n)$。

线性回归中一般用均方差作为损失函数，函数形式如下：

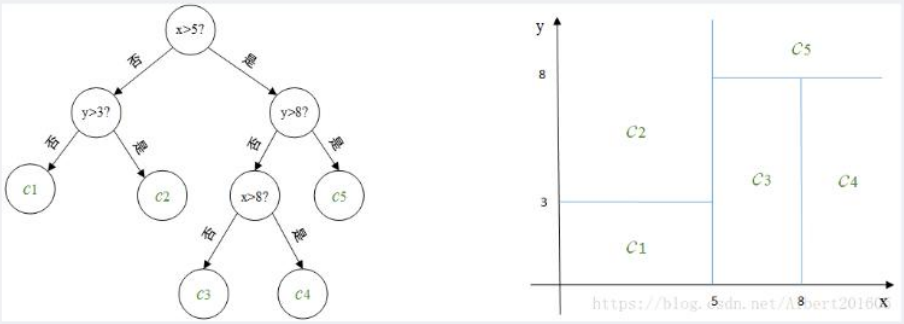
$$J(\theta)=\sum_{i=0}^{m}(h_\theta(x_0,x_1,...,x_n)-y_i)^2$$

我们通常用梯度下降法或最小二乘法求解损失函数最小化时的参数，详细内容可以参考[4]。

2.2 决策树回归

决策树是一种基本的分类与回归方法，这里叙述是回归部分。回归决策树主要指 CART(classification and regression tree)算法，内部结点特征的取值为“是”和“否”， 为二叉树结构。回归决策树就是将特征空间划分成若干单元，每一个划分单元有一个特定的输出。因为每个结点都是“是”和“否”的判断，所以划分的边界是平行于坐标轴的。对于测试数据，我们只要按照特征将其归到某个单元，便得到对应的输出值。

【例】左边为对二维平面划分的决策树，右边为对应的划分示意图，其中c1,c2,c3,c4,c5是对应每个划分单元的输出。



如现在对一个新的向量(6,6)决定它对应的输出。第一维分量6介于5和8之间，第二维分量6小于8，根据此决策树很容易判断(6,6)所在的划分单元，其对应的输出值为c3.

回归决策树的核心问题是切分点的选择与输出值的确定。这里我们用最小二乘法选择切分点，用单元内均值确定输出值。

我们对特征空间采用启发式划分方法，每次划分综合考虑当前集合中所有特征的全部取值，根据平方误差最小化准则选择最优切分点。如将训练集中的第 j 个特征变量作为切分变量，将其取值 s 作为

切分点,并定义如下两个区域：

$$R_1(j,s)=\{x \mid x^{(j)} \le s\}$$

$$R_2(j,s)=\{x \mid x^{(j)} > s\}$$

为找到最优的 j 与 s，对下式求解：

$$\min_{j,s}\left[\min_{c1}\sum_{x_i\in R_1(j,s)}(y_i-c1)^2+\min_{c2}\sum_{x_i\in R_2(j,s)}(y_i-c2)^2\right]$$

其中，c1、c2 为划分后两个区域内的最优输出值，易得这两个最优输出值就是各自对应区域内 y 的均值，证明可参考[5]。由此上式可以写为：

$$\min_{j,s}\left[\sum_{x_i\in R_1(j,s)}(y_i-\hat{c}1)^2+\sum_{x_i\in R_2(j,s)}(y_i-\hat{c}2)^2\right]$$

$$\hat{c}1=\frac{1}{N1}\sum_{x_i\in R_1(j,s)}y_i$$

$$\hat{c}2=\frac{1}{N2}\sum_{x_i\in R_2(j,s)}y_i$$

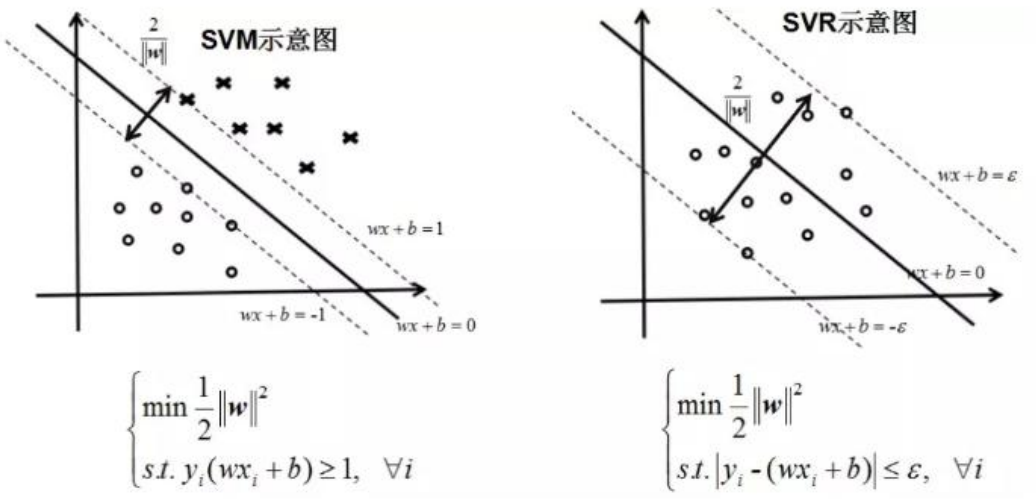
找到最优的切分点(j,s)后，我们依次将输入空间划分为两个区域，接着对每个区域重复上述划分过程，直到满足停止条件为止。这样就生成了一棵回归树，这样的回归树通常称为最小二乘回归树

2.3 SVR

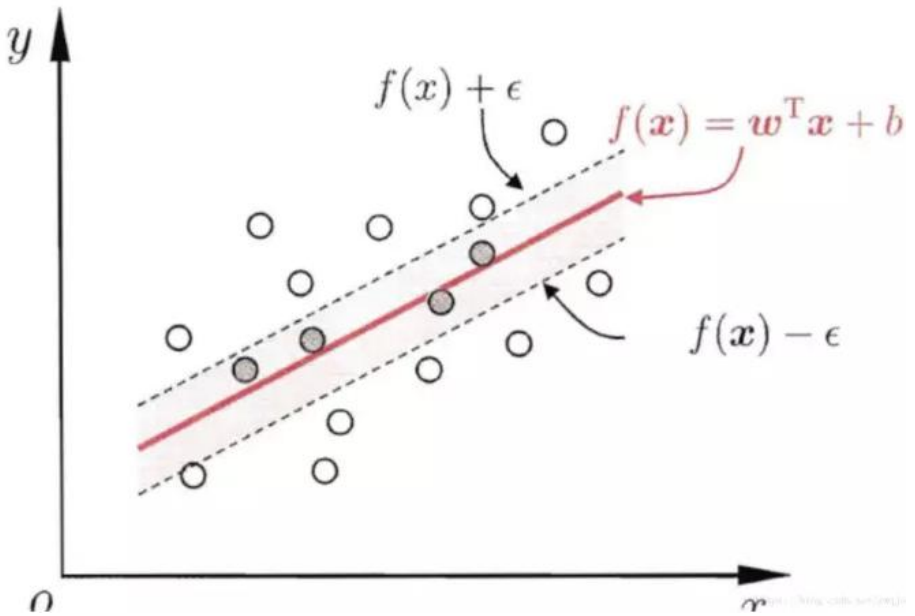
支持向量机(SVM)本身是针对二分类问题提出的，而 SVR（支持向量回归）是 SVM（支持向量机）中的一个重要的应用分支。SVR 回归与 SVM 分类的区别在于，SVR 的样本点最终只有一类，它所寻求的最优超平面不是 SVM 那样使两类或多类样本点分得“最开”，而是使所有的样本点离着超平面的总偏差最小。

SVM 是要使到超平面最近的样本点的“距离”最大；

SVR 则是要使到超平面最远的样本点的“距离”最小。



传统的回归方法当且仅当回归 f(x)完全等于 y 时才认为是预测正确，需计算其损失；而 SVR 认为只要 f(x)与 y 偏离程度不是很大，既可视为预测正确，不用计算损失。具体做法是设置一个阈值 α，只计算 |f(x)-y|>α 的数据的 loss。如下图所示。SVR 表示只要在虚线内部的值都可认为是预测正确，只要计算虚线外部的值的损失即可。



在 SVM/SVR 中，如果没有核映射思想的引入，那么 SVM/SVR 就是

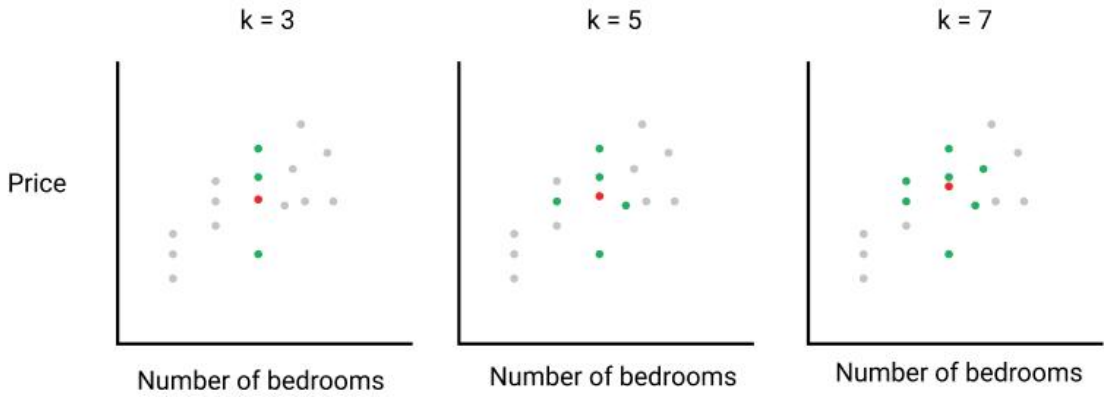
一种加了距离限制的 PLA。针对于非线性支持向量回归机，通常做法就是将低维数据映射到高维的空间，在高维空间中找到线性可分的超平面，最后再把高维空间的超平面映射回低维空间，这样就可以实现 SVM 的分类或者 SVR 的回归。但是，将低维的数据映射到高维的空间，在高维空间做计算，计算量特别大，尤其是当维度很高的情况下，而且也会容易过拟合。

核函数就是为了解决这个问题产生的，用核函数代替线性方程中的线性项可以使原来的线性算法非线性化，即能做非线性回归，此时引进核函数达到了升维的目的，也可以有效的控制过拟合。通俗的讲就是应用核函数就是在低维时就对数据做了计算，这个计算可以看做是将低维空间的数据映射到高维空间中做的计算（就是一个隐式变换）。核函数一般有多项式核、高斯径向基核、指数径向基核、多隐层感知核、傅立叶级数核、样条核等，在回归模型中，不同的核函数对拟合的结果会有较大的影响。具体内容可以参考[6]。

2.4 KNN 回归

KNN 算法不仅可以用于分类，还可以用于回归。通过找出一个样本的 k 个最近邻居，将这些邻居的某个（些）属性的平均值赋给该样本，就可以得到该样本对应属性的值。

比如，我有个 3 个卧室的房子，房租要收多少钱呢？不知道的话，就去看看别人 3 个卧室的房子都租多少钱吧！



其中，K 代表我们的候选对象个数，也就是找和我房间数量最相近的 K 个房子的价格，做一定的处理后（例如平均），作为我们房子的出租价格。

那么，如何衡量“最相近”呢？如何评估预测结果的好坏呢？我们可以使用两个特征向量之间的欧氏距离进行衡量：

设有两个点 P 和 Q，其中 $P=\{p_1,p_2,...,p_n\}$ $Q=\{q_1,q_2,...,q_n\}$，那么 P 与 Q 之间的距离 d 可以表示为：

$$d=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+...+(p_n-q_n)^2}$$

不同特征向量的分布对距离计算的影响不同，应该将它们进行标准化或归一化。具体可参考[7]。

一般采用均方根误差（root mean squared error，RMSE）作为误差评估指标，误差越大，说明预测效果越差。

$$RMSE=\sqrt{\frac{(x_1^r-x_1^p)^2+(x_2^r-x_2^p)^2+...+(x_n^r-x_n^p)^2}{n}}$$

2.5 Bagging

Bagging 是一种提高分类模型精度的方法，算法流程如下：

(1) 从训练集 S 中有放回的随机选取数据集 M(|M|<|S|);

(2) 生成一个分类模型 C;

(3) 重复以上步骤 m 次，得到 m 个分类模型 C1,C2,...,Cm;

(4)对于分类问题，每一个模型投票决定，少数服从多数原则;

(5)对于回归问题，取平均值。

注意：这种抽样的方式会导致有的样本取不到，大约有 $\lim_{n->\infty}(1-\frac{1}{n})^n = 36.8$ 的样本取不到，这部分可用来做测试集。

优点: 通过减少方差来提高预测结果。

缺点: 模型较为复杂。

2.6 Random Forest

Random Forest 是一种基于树模型的 Bagging 算法改进的模型。假定数据集中有 M 个特征和 N 个观测值。每一个树有放回的随机抽出 N 个观测值 m(m=M 或者 m=logM)个特征。把每一个单一决策树的结果综合起来。

优点:

(1) 减少了模型方差，提高了预测准确性。

(2) 不需要给树做剪枝。

(3) 在大规模数据集，尤其是特征较多的情况下，依然可以保持高效率。

(4) 不用做特征选择，并且可以给出特征变量重要性的排序估计。

缺点:

(1) 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合

(2) 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，所以随机森林在这种数据上产出的属性权值是不可信的。

2.7 Adaboost

给定数据集 S，它包含 n 个元组(X1,y1),(X2,y2),...,(Xn,yn)，其中 yi 是数据对象 Xi 的类标号。

(1) 开始时，Adaboost 对每个训练元组赋予相等的权重 1/n。组合分类器包含 T 个基本分类器。

(2) 针对第 t 个分类器 Mt：

首先，从 S 中的元组进行抽样，形成大小为 n 的训练集 St，此处抽样方式为有放回的抽样，抽样过程中，每个元组被选中的机会由它的权重决定；

然后，根据 St 导出（训练出）分类器 Mt，使用 St 检验分类器 Mt 的分类误差，并计算该分类器的“表决权”的权重；

最后，训练元组的权重根据分类器 Mt 的分类情况调整。

如果元组被错误分类，则它的权重增加。

如果元组被正确分类，则它的权重减少。

元组的权重反映元组被分类的困难程度——权重越高，被错误分类的可能性越高。然后，使用这些权重，为下一轮分类器（下一个分类器）产生训练样本。

其基本的思想是，当建立分类器时，希望它更关注上一轮分类器（上一个分类器）错误分类的元组。整个分类过程中，某些分类器对

某些“困难”元组的分类效果可能比其他分类器好。这样，建立了一个互补的分类器系列。

用于二分类或多分类的应用场景。

优点：

(1)很好的利用了弱分类器进行级联。

(2)可以将不同的分类算法作为弱分类器。

(3)AdaBoost 具有很高的精度。

(4)相对于 bagging 算法和 Random Forest 算法，AdaBoost 充分考虑的每个分类器的权重。

缺点：

(1) AdaBoost 迭代次数也就是弱分类器数目不太好设定，可以使用交叉验证来进行确定。

(2)数据不平衡导致分类精度下降。

(3)训练比较耗时，每次重新选择当前分类器最好切分点。

2.8 GBDT

采用决策树作为弱分类器的 Gradient Boosting 算法被称为 GBDT，有时又被称为 MART（Multiple Additive Regression Tree）。GBDT 中使用的决策树通常为 CART。

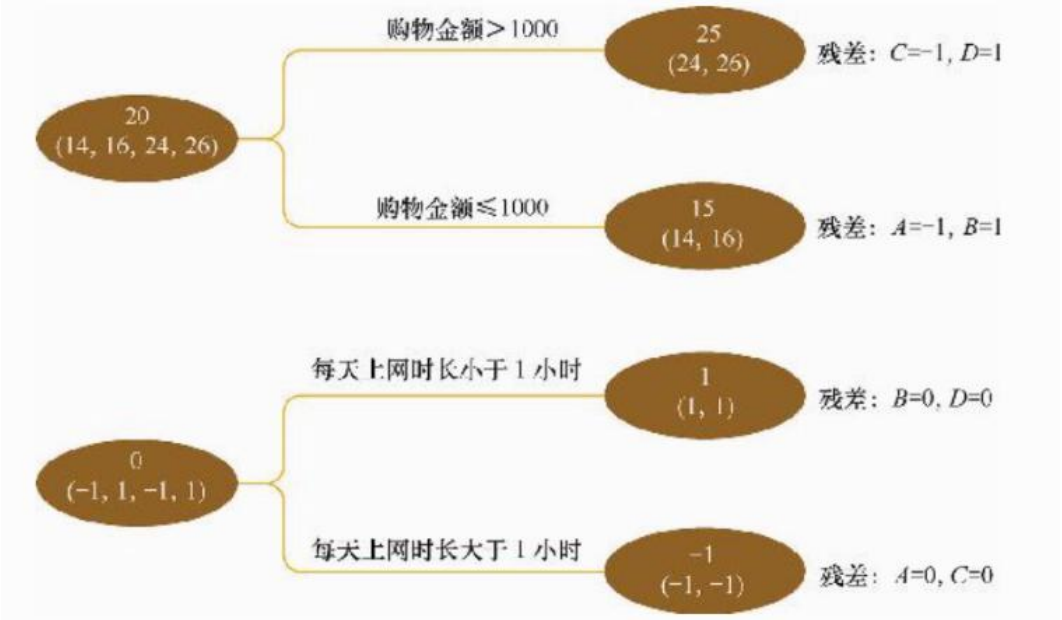
用一个很简单的例子来解释一下 GBDT 训练的过程，如图下图所示。模型的任务是预测一个人的年龄，训练集只有 A、B、C、D 4 个人，他们的年龄分别是 14、16、24、26，特征包括了 ″月购物金额″、″上网时长″、″上网历史″ 等。

下面开始训练第一棵树:

训练的过程跟传统决策树相同，简单起见，我们只进行一次分枝。训练好第一棵树后，求得每个样本预测值与真实值之间的残差。

可以看到，A、B、C、D 的残差分别是-1、1、-1、1。

这时我们就用每个样本的残差训练下一棵树，直到残差收敛到某个阈值以下，或者树的总数达到某个上限为止。



由于 GBDT 是利用残差训练的，在预测的过程中，我们也需要把所有树的预测值加起来，得到最终的预测结果。

优点：

（1）预测阶段的计算速度快，树与树之间可并行化计算。

（2）在分布稠密的数据集上，泛化能力和表达能力都很好，这使得 GBDT 在 Kaggle 的众多竞赛中，经常名列榜首。

（3）采用决策树作为弱分类器使得 GBDT 模型具有较好的解释性和鲁棒性，能够自动发现特征间的高阶关系，并且也不需要对数据进

行特殊的预处理如归一化等。

缺点:

（1）GBDT 在高维稀疏的数据集上，表现不如支持向量机或者神经网络。

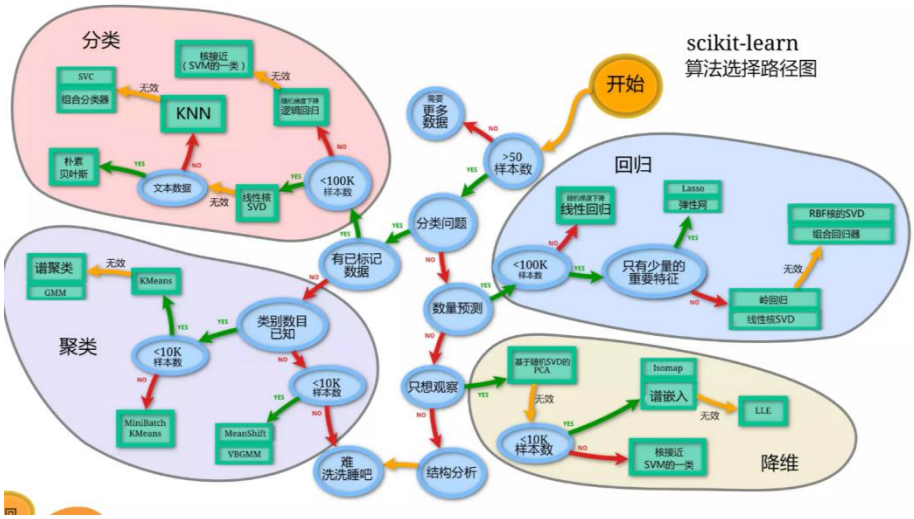
（2）GBDT 在处理文本分类特征问题上，相对其他模型的优势不如它在处理数值特征时明显。

（3）训练过程需要串行训练，只能在决策树内部采用一些局部并行的手段提高训练速度。

关于集成学习的更多内容可以参考[8]。

3 实验

Scikit-learn(sklearn)[9]是机器学习中常用的第三方模块，对常用的机器学习方法进行了封装，包括回归(Regression)、降维(Dimensionality Reduction)、分类(Classfication)、聚类(Clustering)等方法。当我们面临机器学习问题时，便可根据下图来选择相应的方法。



我们的实验环境为sklearn+python3.7+pycharm+Windows 10+Int

el i7 7700HQ+16MB 内存。实验代码可在 github 上获取（https://github.com/Zongyin-Hao/HousePricePrediction/tree/master）。

3.1 加载数据

我们使用加利福尼亚房价数据集[10]作为本次实验的数据集，使用 pandas 加载数据集并做一些简单的分析：

```
# 加载数据
data_set = pandas.read_csv("housing.csv")
print("============================================================")
print(data_set.head())
print("============================================================")
print(data_set.info())
print("============================================================")
print(data_set.describe())
print("============================================================")
data_set.hist(figsize=(15, 10))
plt.show()
```

通过 data_set.head()可以看到数据有 longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,median_house_value,ocean_proximity 等变量，其中 mdeia_houese_value 是我们的目标变量。（限于篇幅我们没有显示全部变量）

```
============================================================
   longitude  latitude  ...  median_house_value  ocean_proximity
0    -122.23     37.88  ...            452600.0         NEAR BAY
1    -122.22     37.86  ...            358500.0         NEAR BAY
2    -122.24     37.85  ...            352100.0         NEAR BAY
3    -122.25     37.85  ...            341300.0         NEAR BAY
4    -122.25     37.85  ...            342200.0         NEAR BAY

[5 rows x 10 columns]
```

通过 data_set.info()可以查看更详细的的信息，我们发现数据供 20640 条样本，其中只有 20433 条样本具有 total_bedrooms 特征，

且 ocean_proximity 是非数值型特征。这些都需要在后文中进行数据清洗。

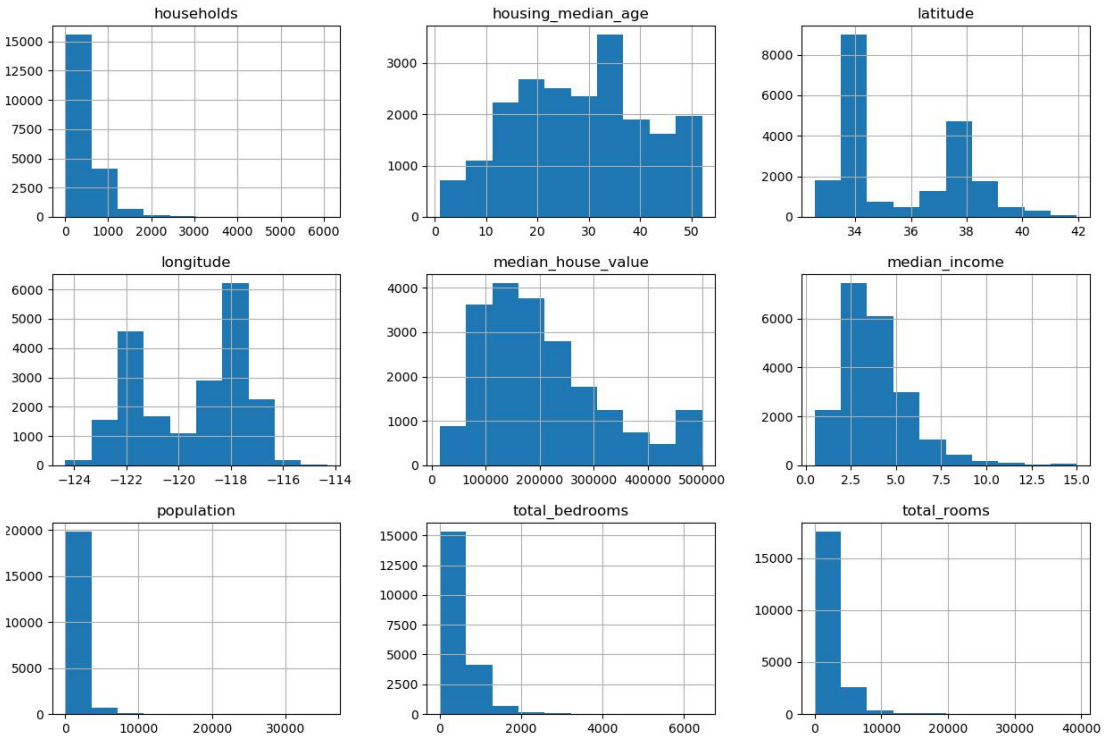
```
============================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

通过 data_set.describe()可以查看数据的分布。

```
============================================================
          longitude      latitude  ...  median_income  median_house_value
count  20640.000000  20640.000000  ...   20640.000000        20640.000000
mean    -119.569704     35.631861  ...       3.870671       206855.816909
std        2.003532      2.135952  ...       1.899822       115395.615874
min     -124.350000     32.540000  ...       0.499900        14999.000000
25%     -121.800000     33.930000  ...       2.563400       119600.000000
50%     -118.490000     34.260000  ...       3.534800       179700.000000
75%     -118.010000     37.710000  ...       4.743250       264725.000000
max     -114.310000     41.950000  ...      15.000100       500001.000000

[8 rows x 9 columns]
```

通过数据的直方图我们可以进一步观察数据的分布，可以看到各个特征变量的取值范围差异较大，median_income 为[0.5, 15]，而 total_bedrooms 为[0, 4000]，为保证预测效果，我们需要在后文进行数据缩放。



3.2 划分训练集与测试集

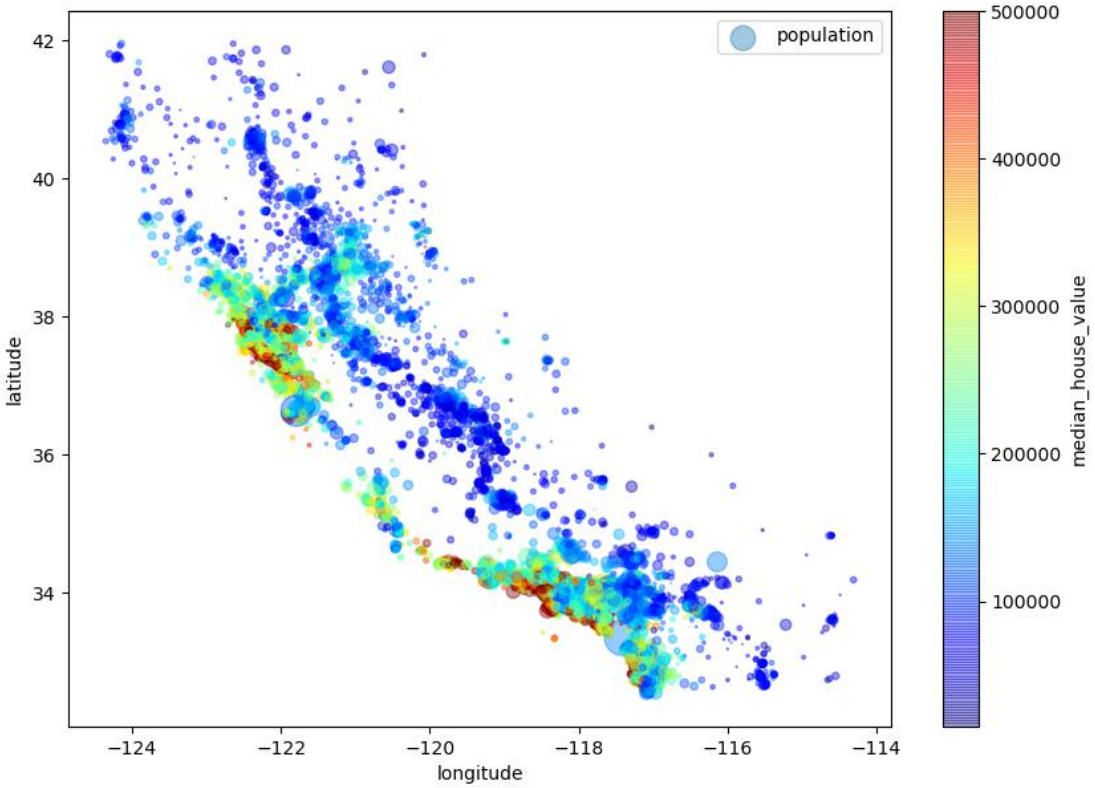
我们规定以 80%、20%的比例划分训练集与测试集。可以采用随机采样与分层采样的方式进行划分，如[12]中的做法。这里我们使用 sklearn 提供的库函数进行划分。

```
# 划分训练集与测试集
train_set, test_set = train_test_split(data_set, test_size=0.2, random_state=24)
```

3.3 可视化分析

为了防止原始数据被意外修改我们建立一份副本。分析地理位置、人口与房价之间的关系。

```
# 可视化分析
copy = train_set.copy()
copy.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
          s=copy["population"] / 100, label="population", figsize=(10, 7),
          c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
          sharex=False)
plt.show()
```



可以看到图像的轮廓与加利福尼亚的地图轮廓很像，不同地区的房价不同。另外可以观察到人口密集地区房价往往较高。这些关联因素可以为我们的模型训练、参数调优带来一定的启发。

我们还可以用皮尔逊相关系数来分析各个因素之间的相关性，如[13]采用的方法。

```
corr_matrix = data_set.corr()
print(corr_matrix["median_house_value"].sort_values(ascending=False))
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

相关系数的取值范围为[-1,1]，当值趋近 1 时，表示特征之间具

有正相关性，反之为负相关。值趋近于 0 表示特征之间不存在线性关系。值得注意的是，这里说的相关性只针对线性相关。如果为非线性关系则该衡量标准失效。

3.4 数据预处理

在 3.1 中我们发现只有 20433 条样本具有 total_bedrooms 特征，且 ocean_proximity 是非数值型特征。很多分类器是无法在有缺失值的数据上运行的，处理方法有：

（1）将存在缺失数据的样本去除掉；

（2）将存在缺失数据的特征去除掉；

（3）将缺失值用统一的值替换，如：均值、中值等。

类似的，很多分类器无法处理标签类特征，也需要进行一些处理，处理方法有：

（1）将非数值型特征删除；

（2）将标签类特征转化成数值型特征。

这里我们将缺失数据的样本删除，将非数值型特征删除，其余处理方法可参考[14]。

```
def initialize_data(data):
    data_i = data
    data_i = data_i.drop("ocean_proximity", axis=1)
    data_i = data_i.dropna(subset=["total_bedrooms"])
    return data_i


train_set = initialize_data(train_set)
test_set = initialize_data(test_set)
```

然后划分特征与标签：

```
feature = train_set.drop("median_house_value", axis=1)
label = train_set["median_house_value"].copy()
```

我们发现各个特征变量的取值范围差异较大，median_income 为[0.5, 15]，而 total_bedrooms 为[0, 4000]，这会影响算法的精度，因此我们需要对数据进行缩放，这里选用 sklearn 提供的 StandardScaler 类对数据进行标准化。其余方法可参考[14]。

```
ss = StandardScaler()
feature = ss.fit transform(feature)
```

3.5 模型训练

sklearn 为我们提供了各类算法的模型训练函数。在进行训练前要考虑如何评估算法的好坏，这里先使用平方根误差作为评估标准。以下是线性回归与决策树回归的误差：

```
[LinearRegression]
mean_squared_error =  69834.26272616169
[DecisionTreeRegression]
mean_squared_error =  0.0
```

可以看到决策树回归的平方根误差为 0，一般而言算法有部分误差是正常的，误差为 0 很有可能是过拟合导致的，因此我们应考虑其他验证方式。

sklearn 提供了 K-fold 交叉验证方法，我们采用和[15]类似的方法，将训练集随机分为 10 份，在 9 份上面训练，在剩下的一份上面测试。代码如下：

```
def regression(name, model, fea, lab):
    model.fit(fea, lab)
    predictions = model.predict(fea)
    error = numpy.sqrt(mean_squared_error(lab, predictions))
    score = cross_val_score(model, fea, lab, cv=10)
    print("["+name+"]")
    print("mean_squared_error = ", error)
    print("mean_score = ", score.mean())
```

其中cross_val_score()函数为模型的评测结果打分，分数为[0,1]之间的浮点数，得分越高越好。注意若模型评测结果比随机猜测还差的话会出现负分。

我们使用交叉验证再次比较线性回归与决策树回归：

```
# 1 线性回归
regression("LinearRegression", LinearRegression(), feature, label)
# 2 决策树回归
regression("DecisionTreeRegression", DecisionTreeRegressor(), feature, label)
```

```
[LinearRegression]
mean_squared_error =  69834.26272616169
mean_score =  0.6323575166425803
[DecisionTreeRegression]
mean_squared_error =  0.0
mean_score =  0.6403035762516108
```

可以看到决策树回归的得分仅比线性回归高一点，效果不是很好。我们继续对其他传统机器学习模型进行评测：

SVR：

```
# 3 SVR
regression("SVR", SVR(), feature, label)
```

```
[SVR]
mean_squared_error =  118375.30090907356
mean_score =  -0.05062542003002237
```

可以看到SVR的预测效果非常差，甚至还不如随机猜测准确。

KNN回归：

```
# 4 KNN回归
regression("KNeighborsRegression", KNeighborsRegressor(), feature, label)
```

```
[KNeighborsRegression]
mean_squared_error =  50515.21045140387
mean_score =  0.7055658545568635
```

KNN回归在四种传统机器学习模型中效果是最好的，接下来我们

看一下集成学习模型的效果。

Bagging 回归：

```
# 5 Bagging回归
regression("BaggingRegression", BaggingRegressor(), feature, label)
```

```
[BaggingRegression]
mean_squared_error =  21590.014173789663
mean_score =  0.7996043399272101
```

Random Forest 回归：

```
# 6 Random Forest回归
regression("RandomForestRegression", RandomForestRegressor(), feature, label)
```

```
[RandomForestRegression]
mean_squared_error =  18316.34305448073
mean_score =  0.8167581996029802
```

Adaboost 回归：

```
# 7 Adaboost回归
regression("AdaBoostRegression", AdaBoostRegressor(), feature, label)
```

```
[AdaBoostRegression]
mean_squared_error =  91224.82371961577
mean_score =  0.37021449940032125
```

GBDT 回归：

```
# 8 GBDT回归
regression("GradientBoostingRegression", GradientBoostingRegressor(), feature, label)
```

```
[GradientBoostingRegression]
mean_squared_error =  52767.382881962265
mean_score =  0.7739469233083771
```

从上面的结果我们暂时可以得出一个结论：随机森林在这几种集成算法中效果是最好的，Adaboost 模型不适用于此数据集，其效果甚至还没有传统回归算法好。但这个结论只是暂时的，因为我们还没有对模型进行调优。

3.6 模型调优

列举出参数组合，直到找到比较满意的参数组合，这是一种调优

方法，当然如果手动选择并一一进行实验这是一个十分繁琐的工作，sklearn 提供了 GridSearch 网格搜索方法，我们只需要将每一个参数的取值告诉它，网格搜索将使用交叉验证方法对所有情况进行验证，并返回结果最好的组合。这里给出一个参数网格的例子：

```
parameters_grid = [
    {"n_estimators": [5, 15], "max_features": [3, 6]},
    {"n_estimators": [10, 20, 30], "max_features": [2, 4, 8]}
]
```

网格中列出了两种组合方式，各有 2*2=4 种组合以及 3*3=9 种组合，共 13 种组合。n_estimators 和 max_features 分别指分类器个数和特征个数。现在我们对 Bagging、Random Forest、AdaBoost、GBDT 进行调优，分别输其最优参数、最优分类器、最优得分（各个参数间的组合调优可以参考[16]）：

Bagging:

```
parameters_grid = [
    {"n_estimators": [30, 40, 50], "max_features": [6, 8]}
]

# Bagging调优
improvement("BaggingRegression", BaggingRegressor(), feature, label)
[BaggingRegression]
best_params: {'max_features': 6, 'n_estimators': 40}
best_estimator: BaggingRegressor(base_estimator=None, bootstrap=True, bootstrap_features=False,
                max_features=6, max_samples=1.0, n_estimators=40, n_jobs=None,
                oob_score=False, random_state=None, verbose=0,
                warm_start=False)
best_score: 0.8126995921235434
```

Random Forest:

```
parameters_grid = [
    {"n_estimators": [100, 125, 150], "max_features": [6, 8]}
]
# Random Forest调优
improvement("RandomForestRegression", RandomForestRegressor(), feature, label)
```

```
[RandomForestRegression]
best_params: {'max_features': 6, 'n_estimators': 100}
best_estimator: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=6, max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
best_score: 0.8187984279468237
```

Adaboost:

```
parameters_grid = [
    {"n_estimators": [100, 105, 110]}
]

# AdaBoost调优
improvement("AdaBoostRegression", AdaBoostRegressor(), feature, label)
```

```
[AdaBoostRegression]
best_params: {'n_estimators': 100}
best_estimator: AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear',
                  n_estimators=100, random_state=None)
best_score: 0.38718673279702776
```

GBDT:

```
parameters_grid = [
    {"n_estimators": [400, 450, 500], "max_features": [6, 8]}
]

# GBDT调优
improvement("GradientBoostingRegression", GradientBoostingRegressor(), feature, label)
```

```
[GradientBoostingRegression]
best_params: {'max_features': 6, 'n_estimators': 500}
best_estimator: GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=6, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=500,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
best_score: 0.8196657296714289
```

经过调优后，各算法的预测能力都有所提升，可以看到 Random Forest 与 GBDT 的效果不相上下，GBDT 稍胜一筹。Adaboost 仍然效果

不佳。下面我们将在测试集上进行模型评估，选出最优模型。

3.7 模型评估对比

首先按照 3.4 中的方法预处理一下测试集：

```
feature_test = test_set.drop("median_house_value", axis=1)
label_test = test_set["median_house_value"].copy()
feature_test = ss.transform(feature_test)
```

为了方便这里我们将训练、测试步骤整合，并赋予 3.6 得出的最优参数，用均方误差进行模型评估。

```
def final_predict(name, model, fea1, lab1, fea2, lab2):
    model.fit(fea1, lab1)
    final_prediction = model.predict(fea2)
    final_error = numpy.sqrt(mean_squared_error(lab2, final_prediction))
    print("[" + name + "]")
    print("mean_squared_error = ", final_error)
# 1 线性回归
final_predict("LinearRegression", LinearRegression(), feature, label, feature_test, label_test)
# 2 决策树回归
final_predict("DecisionTreeRegression", DecisionTreeRegressor(), feature, label, feature_test, label_test)
# 3 SVR
final_predict("SVR", SVR(), feature, label, feature_test, label_test)
# 4 KNN回归
final_predict("KNeighborsRegression", KNeighborsRegressor(), feature, label, feature_test, label_test)
# 5 Bagging回归
final_predict("BaggingRegression", BaggingRegressor(n_estimators=40,max_features=6), feature, label,
              feature_test, label_test)
# 6 Random Forest回归
final_predict("RandomForestRegression", RandomForestRegressor(n_estimators=100,max_features=6), feature, label,
              feature_test, label_test)
# 7 Adaboost回归
final_predict("AdaBoostRegression", AdaBoostRegressor(n_estimators=100), feature, label,
              feature_test, label_test)
# 8 GBDT回归
final_predict("GradientBoostingRegression", GradientBoostingRegressor(n_estimators=500,max_features=6), feature, label,
              feature_test, label_test)
```

```
[LinearRegression]
mean_squared_error =  68463.82311729799
[DecisionTreeRegression]
mean_squared_error =  68126.2906255943
[SVR]
mean_squared_error =  118071.80187668382
[KNeighborsRegression]
mean_squared_error =  61888.363579511984
[BaggingRegression]
mean_squared_error =  50972.060984674
[RandomForestRegression]
mean_squared_error =  49457.318434693065
[AdaBoostRegression]
mean_squared_error =  95083.77693928278
[GradientBoostingRegression]
mean_squared_error =  49796.23761956351
```

最终我们可以看到 Random Forest 与 GBDT 都具有良好的效果，实际应用中我们可以选择这两个模型进行房价预测。

4 总结

我们从房价预测问题出发，分析了传统机器学习算法与 Bagging、Random Forest、Adaboost、GBDT 等集成学习算法，通过实验对比得出 Random Forest 与 GBDT 算法的预测效果较好，均方误差分别为 49457.318、49796.238，具备一定的实际应用价值。

由于这是我首次接触机器学习技术，文中对算法的理解仅停留在原理阶段，实验中也只是简单调用现有框架的库函数，没有针对各个算法做深层优化。后面需要继续阅读机器学习的相关文献，进一步研究相关的主流技术，寻找适合的应用场景，并学习、研究过程中提出自己的算法。

Reference

[1] 杨博文, 曹布阳. 基于集成学习的房价预测模型[J]. 电脑

知识与技术, 2017(29).

[2] 机器学习（多领域交叉学科）：https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/217599?fr=aladdin

[3] 机器学习--集成学习（Ensemble Learning）：https://www.cnblogs.com/zongfa/p/9304353.html

[4] 线性回归（Linear Regression）：https://www.cnblogs.com/huangyc/p/9782821.html

[5] 决策树—回归：https://blog.csdn.net/Albert201605/article/details/81865261

[6] SVR（Support Vactor Regerssion）：支持向量回归机：https://www.jianshu.com/p/399ddcac2178

[7] 经典算法之 K 近邻（回归部分）：https://www.cnblogs.com/pythoner6833/p/9296035.html

[8] 集成学习总结：https://www.cnblogs.com/zingp/p/11076362.html

[9] Python 之 Sklearn 使用教程：https://www.jianshu.com/p/6ada34655862

[10] 加利福尼亚房价数据集：https://github.com/ageron/handson-ml/tree/master/datasets/housing

[11] 使用 sklearn 进行数据挖掘-房价预测(1)：https://www.cnblogs.com/wxshi/p/7725814.html

[12] 使用 sklearn 进行数据挖掘-房价预测(2)—划分测试集：h

ttps://www.cnblogs.com/wxshi/p/7725850.html

[13] 使用 sklearn 进行数据挖掘-房价预测(3)—绘制数据的分布：https://www.cnblogs.com/wxshi/p/7764501.html

[14] 使用 sklearn 进行数据挖掘-房价预测(4)—数据预处理：https://www.cnblogs.com/wxshi/p/7764518.html

[15] 使用 sklearn 进行数据挖掘-房价预测(5)—训练模型：https://www.cnblogs.com/wxshi/p/7787047.html

[16] 使用 sklearn 进行数据挖掘-房价预测(6)—模型调优：https://www.cnblogs.com/wxshi/p/7789288.html