人们希望能够对房价进行较为准确的预测。房价变动不仅受时间、区域等因素影响， 而且与房屋年限、附近地理条件、人文、交通等因素存在一定联系， 其中涉及很多随机影响因素， 故无法使用 Logistic回归模型等简单模型进行有效预测， 通常使用多元回归模型、神经网络模型、随机森林模型等具有较强拟合能力的机器学习模型作为预测分析工具。这里使用随机森林模型基于BostonHousing 数据集进行房价预测分析。BostonHousing数据集涵盖了2006年1月至2010年7月间美国波士顿市的1460条带有实际价格标签的房屋交易数据和1459条无实际价格标签的房屋交易数据。每条房屋交易数据中除去实际价格标签之外， 还包括房屋编号、地理位置、房屋面积大小、车库面积等79个属性， 该数据集中部分数据的部分属性如图5-19所示。

随机森林模型属于 Bagging集成策略，对于回归任务可采用简单平均法集成多个回归决策树的预测结果从而得到最终的随机森林预测结果。首先， 需对数据进行预处理从而降低模型构

183

05 120RL 60 7332 Paré NA 110

图5-19 BostonHousing数据集中部分数据的部分属性

造的难度并保证最终的预测效果； 然后， 对经过预处理后的训练样本集进行自助采样并使用采样到的样本构造出对应的回归决策树模型； 最后， 使用简单平均法将所有决策树模型的预测结果集成起来构成随机森林的输出。现用 Python3 编程语言实现上述过程， 具体方法如下：

(1) 数据预处理。在模型构造过程当中所要用到的样本均为带有实际价格标签的房屋交易数据， 故需对 BostonHousing数据集进行筛选， 选择带有实际价格标签的1460条交易数据组成数据集 D， 其余数据组成真实测试集T。读取数据集D、T中的数据并对房屋价格进行统计分析， 具体代码如代码5-1所示。

import numpy as np, pandas as pd, os, seaborn as sns, matplotlib. pyplot

as plt

from statsmodels. stats. outliers \_ influence import variance \_ inflation \_ factor

from sklearn. preprocessing import StandardScaler

from sklearn. decomposition import PCA

data \_ train = pd. read \_ csv('./ data/ train. csv')

data \_ test = pd. read \_ csv('./ data/ test. csv')

print("data \_ train. head():", data \_ train. head())

print("data \_ test. head():", data \_ test. head())

print("数据的列名 data \_ train. columns:", data \_ train. columns)

print("每列的数据格式 data \_train. info)

data \_ train \_ dtypes = data \_ train. dtypes

print("数据结构为:", data \_ train \_ dtypes)

#查看因变量价格的情况，进行基础分析

sns. distplot(data \_ train['SalePrice'])

print("价格情况")

plt. show()

sns. set(style="darkgrid")

titanic = pd. DataFrame(data \_ train['SalePrice']. value \_ counts())

titanic. columns = ['SalePrice\_ count']

ax = sns. countplot(x="SalePrice\_ count", data= titanic)

plt. show()

代码5-1数据读取与统计分析

数据读取结果展示在图5-20中， 其中图5-20a显示了D 中前五个数据， 而图5-20b则显示了T中前五个数据。对房价的统计分析结果如图5-21 所示， 图中反映了数据集D中样本的房价分布情况。

图5-21 房价分布情况

SalePrice

100000 200000 300000 400000 500000 600000 700000 800000

数据集中存在部分数据缺失， 这会增加模型构造过程的难度并且影响最终随机森林模型的性能， 因此需对缺失数据进行处理。本节采用如下规则对缺失数据进行处理： 若某一属性缺失值较多， 则直接删除该属性， 若缺失值较少则进行填充。对于数值型变量， 采用中位数进行填充， 对于类别型数据则直接填充为“None”。缺失数据处理过程的具体代码如代码5-2所示。

miss \_ data = data \_ train. isnull(). sum(). sort \_ values(ascending= False)

total = data \_ train. isnull(). count()

miss \_ data \_ tmp = (miss \_ data/ total). sort \_ values(ascending= False)

def precent(X):

X = '%.2f%%'%(X \* 100)

return X

miss \_ precent = miss \_ data \_ tmp. map(precent)

miss \_ data \_ precent = pd. concat([total, miss \_ precent, miss \_ data \_ tmp], axis=1, keys=[' total',' Percent',' Percent \_ tmp']). sort \_ values(by=' Percent \_ tmp', ascending= False)

print("有缺失值的变量打印出来:", miss \_ data \_ precent[miss \_ data \_ precent[' Percent'] ! = '0.00%'])

drop \_ columns = miss \_ data \_ precent[miss \_ data \_ precent[' Percent \_ tmp']>0.15]. index

data \_ train = data \_ train. drop(drop \_ columns, axis=1)

data \_ test = data \_ test. drop(drop \_ columns, axis=1)

print("处理后的数据列名:", data \_ train. columns)

print("处理后的数据列名长:", len(data \_ train. columns))

class \_ variable = [

col for col in data \_ train. columns if data \_ train[col]. dtypes == 'O'] numerical \_ variable = [

col for col in data \_ train. columns if data \_ train[col]. dtypes! = '0'] print('类别型变量:%s' % class \_ variable,'数值型变量:%s'% numerical \_ variable)

from sklearn. preprocessing import Imputer

padding = Imputer(strategy=' median')

data \_ train[numerical \_ variable] = padding. fit \_ transform(

data \_ train[numerical \_ variable])

data \_ test[numerical \_ variable[:-1]

]= padding. fit \_ transform(data \_ test[numerical \_ variable[:-1]]) data \_ train[class \_ variable] = data \_ train[class \_ variable]. fillna(' None') data \_ test[class \_ variable] = data \_ test[class \_ variable]. fillna(' None')

data = pd. concat([data \_ train['SalePrice'], data \_ train['CentralAir']], axis=1)

fig = sns. boxplot(x='CentralAir', y="SalePrice", data= data)

plt. title('CentralAir')

plt. show()

data = pd. concat([data \_ train['SalePrice'], data \_ train['MSSubClass']], axis=1)

fig = sns. boxplot(x='MSSubClass', y="SalePrice", data= data)

plt. title('MSSubClass')

plt. show()

data = pd. concat([data \_ train['SalePrice'], data \_ train['MSZoning']], axis=1)

fig = sns. boxplot(x='MSZoning', y="SalePrice", data= data)

plt. title('MSZoning')

plt. show()

fig = plt. figure()

ax = fig. add \_ subplot(111)

ax. scatter(x= data \_ train['SalePrice'], y= data \_ train['LotArea'])

plt. xlabel('SalePrice')

plt. ylabel('LotArea')

plt. title('LotArea')

plt. show()

data= pd. concat([data \_ train['SalePrice'], data \_ train[' Street']], axis=1)

fig = sns. boxplot(x=' Street', y="SalePrice", data= data)

plt. title(' Street')

plt. show()

data = pd. concat([data \_ train['SalePrice'], data \_ train['LotShape']], axis=1)

fig = sns. boxplot(x='LotShape', y="SalePrice", data= data)

plt. title('LotShape')

plt. show())

plt. show()

代码5-2缺失数据处理

对数据集D进行缺失数据处理之后， 为提升构造单棵决策树的效率， 还需要对其中的数据进行PCA 降维处理。对数据集D 进行PCA 降维处理的具体代码如代码5-3所示。

185

VIF\_ list = [variance \_ inflation \_ factor(X, i) for i in range(X. shape[1])]VIF\_ list

print("降维处理后的测试数据集:", test)

代码5-3 PCA降维处理

PCA 降维的关键在于求解协方差矩阵并对其进行对角化， 在求得属性的协方差矩阵并进行对角化后， 应根据方差贡献率或预先设定的目标维度选择用于降维的基向量。由于需将原始数据降至9维， 故在对角化后取较大的9个特征值所对应的特征向量作为基向量， 对原始数据进行降维。图5-22展示了数据集 D中经过PCA 降维处理后的部分数据。

0

1

2

3

4

5

6

7

8

9

0-1.771871

0.391450 - 0.006668

1.097874 - 1.421750 - 0.897968 0.071829 - 0.032507 - 0.000231 - 0.392641

1 - 0.566627 - 0.710295 - 0.516170 0.420448 0.324016 0.215963 0.001246 - 0.357240 - 0.846454 0.320638

2 - 1.771871 0.391450 - 0.006668 1.541442 - 0.375103 - 0.446955 - 0.824441 0.087345 - 0.259276 - 0.129342

3 1.535040 0.346877 - 1.243908 - 0.302525 1.503985 0.136684 - 0.325437 - 1.534968 1.970617 - 0.380982

4 - 2.299063 0.523169-0.756719 2.772031 0.004415 - 0.293919 - 0.286884 - 0.738848 0.460618 - 0.297366

图5-22 数据集D中经过PCA 降维处理后的部分数据

(2) 构造随机森林。对预处理后仅含9个特征和实际价格标签的新样本进行处理。为构造集成模型并测试模型性能， 此处采用留出法构造训练样本集S和测试样本集V。假设数据集D经过数据预处理过程后得到的新数据集为D'，采用留出法对D'进行划分， 随机选择D'中80%的样本组成训练集S, 剩余20%的样本组成测试集 V。由于 Python3 编程语言中的 sklearn 库中包含随机森林模型的构造函数， 在确定了训练集S后， 只需确定某些超参数便可直接调用该函数构造随机森林模型。本节设置个体学习器数量为400， 并选用决定系数R²作为模型评估指标分别在训练集S和测试集V上测试模型的性能， 从而求得拟合率和准确率。构造随机森林

186

模型并对其进行性能测试的具体过程如代码5-4所示。

from sklearn. model \_ selection import train \_ test \_ split from sklearn. ensemble import RandomForestRegressor train \_ data, test \_ data, train \_ target, test \_ target = train \_ test \_ split(train, target, test \_ size=0.2, random \_ state=0)

\_

代码5-4构造随机森林模型并进行性能测试

经上述过程可构造一个以400棵回归决策树作为个体学习器的随机森林模型， 该模型的拟合率约为0.9804， 而其准确率约为0.8417， 因此可以认为该随机森林模型已经较好地拟合了训练样本并具备较强的泛化能力， 依据真实房屋信息使用该模型对房价进行预测分析所得到的预测结果的可信度较高。

(3) 房价预测。在完成随机森林模型构造后便可使用该模型对真实房屋信息数据进行房价预测， 此处对数据集T中的样本进行价格预测， 并将最终预测结果保存在 Predictions. csv文件当中， 这一过程的具体过程如代码5-5所示。

经上述过程可得到与数据集T中所有真实房屋信息数据所对应的房价预测值， 表5-12中展示了随机森林模型对序号为1461 至1474的14条真实房屋信息数据的房价预测结果。

from sklearn. model \_ selection import GridSearchCV

param \_ grid = {'n\_ estimators':[1,10,100,200,300,400,500,600,

700,800,900,1000,1200],' max \_ features':(' auto',' sqrt','log2')}

m = GridSearchCV(RandomForestRegressor(), param \_ grid)

m=m. fit(train \_ data, train \_ target. values. ravel())

print(m. best \_ score \_)

print(m. best \_ params \_)

m = RandomForestRegressor

(n\_ estimators=200, max \_ features=' sqrt')

m. fit(train \_ data, train \_ target. values. ravel())

predict = m. predict(test)

test = pd. read \_ csv('./ data/ test. csv')[' Id']

sub = pd. DataFrame()

sub[' Id'] = test

sub['SalePrice'] = pd. Series(predict)

sub. to \_ csv(' Predictions. csv', index= False)

print(' finished! ')

代码5-5对真实数据进行预测

表5-12 部分真实房屋信息数据的房价预测结果

序号 1461 1462 1463 1464 1465 1466 1467

房价(千美元) 121.0584 134.8603 187.8295 189.0859 198.3989 192.7535 186.4237

序号 1468 1469 1470 1471 1472 1473 1474

房价 (千美元) 188.7048 199.420 132.8975 201.9586 107.2034 107.6146 169.8359

由于通过模型评估过程已经确定所构造的随机森林模型已经较好地拟合了训练样本并具备较强的泛化能力， 故可认为该模型对于真实房屋信息数据的房价预测结果具有一定的可信性，故可将该随机森林模型的预测结果作为房屋交易等现实场景中房价制定的参考信息。