《机器学习》

-集成学习

华为技术有限公司

目录

1 实验介绍

现实世界的复杂多样性，单个机器学习算法（学习器），泛化能力比较有限（很难适用于多种变化场景），而集成学习是有效的方法，不需要针对每一个场景都去训练学习一个大的模型（学习器），而是想办法使用多个模型（学习器）协同工作，会更加有效，特别是，当模型对原始数据的变化敏感时，集成学习效果更好。（有实验证明，集成学习在绝大多是情况下，比单个学习器效果更好）。

集成学习就是组合多个弱学习模型以期得到一个更好更全面的强学习模型，其主要思想是即便某一个弱学习模型得到了错误的结果，其它的弱学习模型也可以将错误纠正回来。

本章实验难度分为中级和高级。

中级实验：欺诈预测、信用卡欺诈预测

高级实验：犯罪类型预测、贷款产品预测、个人收入预测 、订购定期存款、网站点击率预测、房价预测

1.1 实验目的

本章实验的主要目的是掌握机器学习-监督学习集成算法相关基础知识点，了解集成学习相关知识，包括集成算法中的随机森林、GBDT、XGBoost 等算法应用和优化；

熟悉使用 Jupyter notebook 编译工具；

掌握建模实验的一般流程。

1.2 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

实验	简述	难度	软件环境	开发环境
贷款产品预测	基于抵押贷款数据，分别使用 GBDT、随机森林算法进行贷款产	高级	Python3.7、ModelArts	PC 64bit

	品的预测。			
犯罪类型预测	基于芝加哥犯罪数据，分别使用随机森林、GBDT算法进行犯罪类型预测。	高级	Python3.7、ModelArts	PC 64bit
信用卡欺诈预测	基于信用卡的历史交易数据，使用决策树算法进行欺诈预测。	中级	Python3.7、ModelArts	PC 64bit
个人收入预测	基于美国个人收入数据，使用多种分类算法实现对美国个人收入是否超过5000美元的预测。	高级	Python3.7、ModelArts	PC 64bit
订购定期存款	基于客户订购定期存款的数据，使用决策树、随机森林进行定期存款预测。	中级	Python3.7、ModelArts	PC 64bit
房价预测	基于kaggle网站提供的房价数据，分别使用线性回归、随机森林和GBDT算法实现房价预测。	高级	Python3.7、ModelArts	PC 64bit
网站点击率预测	某挑战赛中关于广告点击的数据，使用分类算法进行广告点击率的预测。	高级	Python3.7、ModelArts	PC 64bit

2 贷款产品预测

2.1 贷款产品实验介绍

HMDA(The Home Mortgage Disclosure Act)是某国一项法案，他规定金融机构必须将提供抵押贷款的数据向全社会公开。这些数据有助于显示出贷款机构是否为真正需要住房相关的客户而服务。通过分析这些数据，可以对政府官员的决策制定提供帮助，并且有可能揭示某些具有歧视性的贷款模式。原始数据含有变量共有 47 个，目标变量为 action_taken_name，可以简单的理解为申请贷款的结果，本案例的主要目标是探寻申请人的某些信息或者行为是否会影响到最终贷款的的发放的预测问题，从而展现一个完整的数据挖掘流程。(Washington_State_HDMA-2016.csv)

https://www.kaggle.com/miker400/washington-state-home-mortgage-hdma2016

图2-1 Kaggle HDMA 数据介绍

【实验环境要求】:

1、python3.7

2、ModelArts 平台

本实验在 python3.7 环境下完成，可下载 Anaconda，下载地址为:https://www.anaconda.com/distribution/。

另外需要安装 seaborn、imblearn，使用 pip 安装命令如下:

pip install seaborn

pip install numpy

其他安装包如上进行安装。

2.2 实验总体设计

本实验遵从数据挖掘的一般流程，首先对已经下载本地的数据进行读取，常规的探索后，进行数据预处理，随后直接选择 sklearn 模块中的决策树、随机森林、GDBT 以及 XGBoost 算法进行建模，选择出性能突出的模型做进一步的调参优化，最终确认模型，进行预测。

2.2.1 实验整体方案

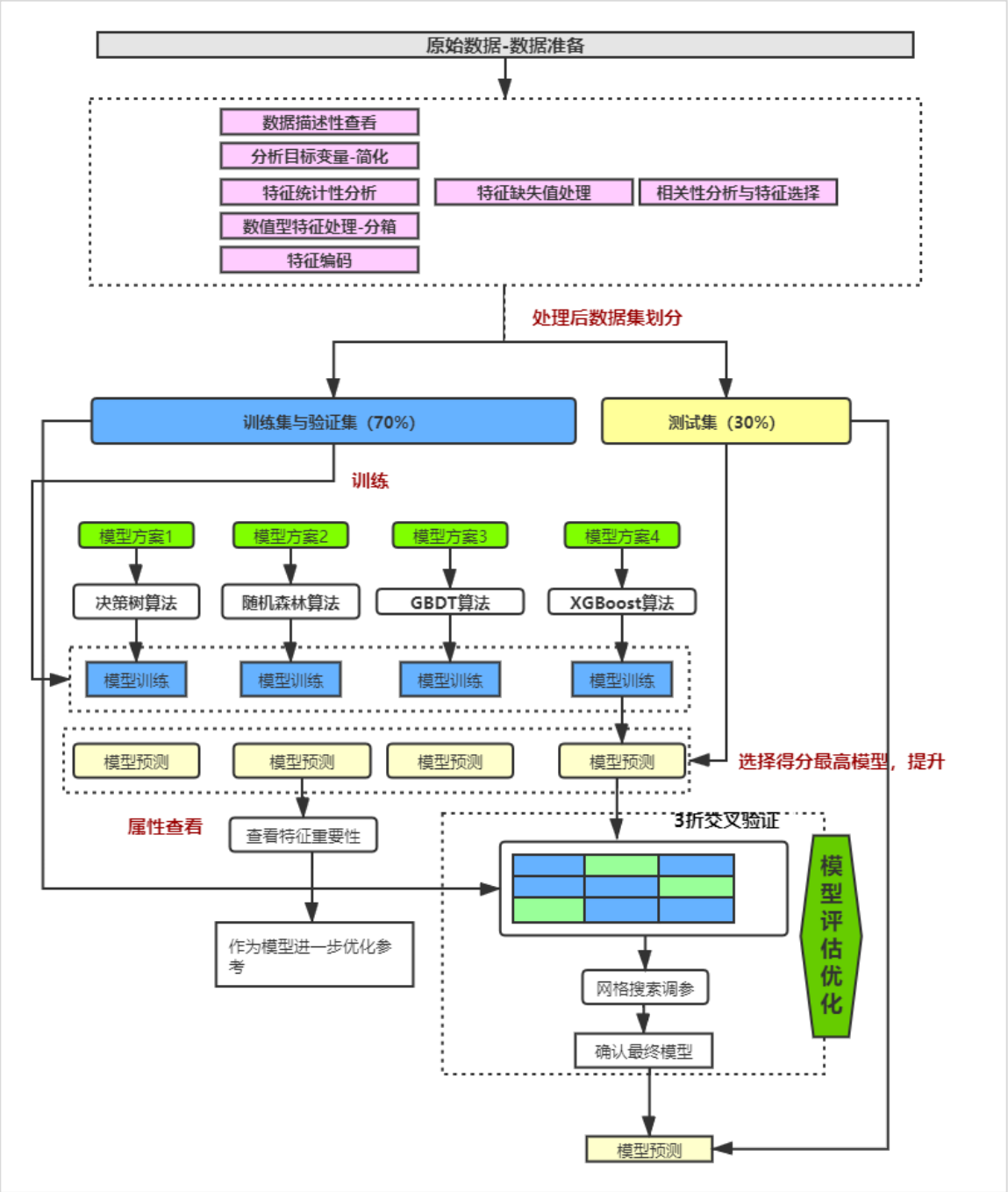


图1-1 实验方案流程图

2.3 实验详细设计与实现

2.3.1 导入实验环境

步骤 1 导入相应的模块

本实验使用到的框架主要包括 numpy，pandas，scikit-learn，matplotlib，missingno，seaborn 库。scikit-learn 库是 Python 的机器学习库，提供一些常用的机器学习算法模型及模型参数优化功能；numpy ，pandas 库是 Python 中结构化数据处理的库，主要用于结构化数据的统计分析及操作；matplotlib，seaborn 主要用于数据分析过程的可视化展示。

```
# 导入模块
import pandas as pd
import seaborn as sns
import missingno #专门用做缺失值的可视化处理库
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np

#程序运行过程中忽略警告信息
import warnings
warnings.filterwarnings("ignore")

#导入数据拆分算法 train_test_split 进行数据集的拆分
from sklearn.model_selection import train_test_split

# 在 Jupyter 上画图，#使用该魔法命令，不用写 plt.show()
get_ipython().run_line_magic('matplotlib', 'inline')
```

2.3.2 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。

```
hdma_raw = pd.read_csv(r"Washington_State_HDMA-2016.csv",low_memory=False)
```

步骤 2 在原始数据上复制一份数据

实验往复操作时不用再重新读取

```
hdma=hdma_raw
```

步骤 3 数据的重复性探索

```
flag = hdma.duplicated()
flag.any()
```

输出结果如下：

```
False
```

说明数据集没有重复的记录，不需要去重操作。

步骤 4 查看数据尺寸

```
hdma.shape #获取数据规模
```

输出结果如下：

```
(466566, 47)
```

步骤 5 查看数据前几行

```
hdma.head(2)
```

输出结果如下：

经过数据的初步探索，可以了解到了数据集的整体规模，46W+的记录数，47 个特征列（含目标列），通过对数据集前 2 行的简单查看，特征列既包含数值类型，也包含字符串类型，且字符串类型较多，后续会分别对各个特征进行探索分析，若部分字符串类型的特征列不可删去，后续考虑保留用 Label 编码，或者 One-Hot 编码的方式处理。

2.3.3 数据预处理

2.3.3.1 分析目标变量，建模问题简化

步骤 1 目标变量简要分析：对 action_taken_name 做简要分析，确定并简化分析目标

```
hdma['action_taken_name'].value_counts()
```

输出结果如下：

```
Loan originated                                   263712
Application denied by financial institution        64177
Application withdrawn by applicant                 60358
Loan purchased by the institution                  48356
File closed for incompleteness                     18176
Application approved but not accepted              11735
Preapproval request denied by financial institution   35
Preapproval request approved but not accepted         17
Name: action_taken_name, dtype: int64
```

其中 Loan originated 意味着贷款申请已经获得批准，Loan purchased 是指贷款人在二级市场上购买贷款，而目标旨在“探寻申请人的某些信息或者行为是否会影响到最终贷款的的发放”，因此将这两项相关数据集删掉，然后将问题简化，将问题转变为一个简单的二分类问题，新增字段 loan_status，如果贷款已获批准 Loan originated，赋值为 0.0，其他情况（即未或批准）为 1.0 也可设置（0 和 1）。

步骤 2 多分类目标问题转化为 2 分类问题

删除 action_taken_name 为“Application withdrawn by applicant”和“Loan purchased by the institution”，并生成新的特征列“loan_status”

```
hdma=hdma[hdma['action_taken_name']!="Application withdrawn by applicant"]
hdma=hdma[hdma['action_taken_name']!='Loan purchased by the institution']
hdma['loan_status']=[0.0 if x=="Loan originated" else 1.0 for x in hdma['action_taken_name']]
```

步骤 3 删除原先的目标列

```
del hdma['action_taken_name']
```

步骤 4 查看当前数据尺寸

```
hdma.shape
```

输出结果如下：

```
(357852, 47)
```

可以看出通过整合目标列的输出结果，转化为 2 分类问题后，原先数据集减少了十几万行。

步骤 5 查看目标变量的分布情况，并绘制出直方图：

```
print(hdma['loan_status'].value_counts())
sns.countplot('loan_status', data=hdma)
```

输出结果如下：

```
0.0    263712
1.0     94140
Name: loan_status, dtype: int64
```

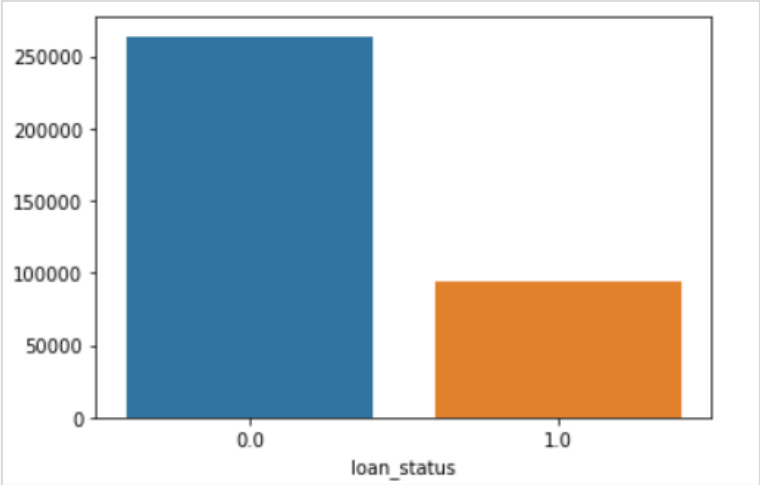


图2-1 目标变量分布

可以看到，正负样本比例大致为 3:1，并非典型的样本不均衡横问题，在建模前可以不需要做特别的抽样处理（如过采样、欠采样）。

2.3.3.2 分析输入特征

该步骤先对特征做初步的探索，根据字段类型的不同，将在后续步骤中采取不同的处理方式。

步骤 1 特征的初步分析

```
hdma.info()
```

输出结果如下：

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 357852 entries, 0 to 466565
Data columns (total 47 columns):
 #    Column                              Non-Null Count     Dtype
---   ------                              --------------     -----
 0    tract_to_msamd_income               357494 non-null    float64
 1    rate_spread                         8638 non-null      float64
.....
 46   loan_status                         357852 non-null    float64
dtypes: float64(10), int64(4), object(33)
memory usage: 141.0+ MB
```

查看各个特征的信息可以初步得知：数据存在数值型属性和标称型属性，且个别特征的缺失值比较严重（非缺失值小于千位数）。

步骤 2 初步处理数据

基于对数据初步已知的情况，agency_name，agency_abbr 两列实际含义相同，且取值一一致，选择删除 agency_name。

```
del hdma['agency_name']
```

输出结果如下：

2.3.3.3 处理缺失数据

步骤 1 缺失数据可视化

利用可视化方法继续查看数据集所有字段类型为 nan 的情况。

```
missingno.matrix(hdma, figsize = (20,5))
```

输出结果如下：

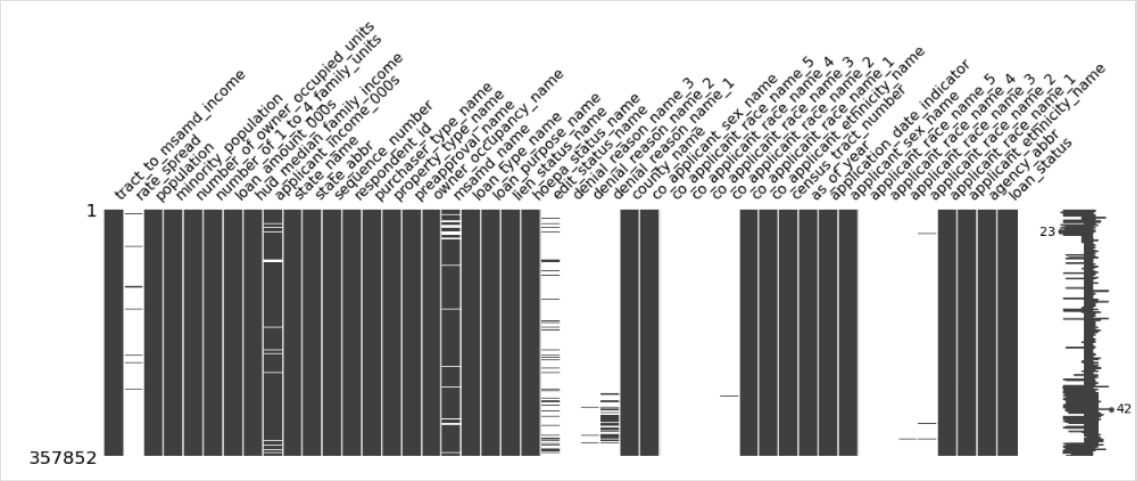


图3-1 缺失值分布

图4-1

可以看出有几个列缺失非常严重，考虑基于一定的规则进行处理。

步骤 2 统计各特征的缺失率

通过可视化展示我们可以获取特征缺失的直观感受，接着我们计算出准确的特征缺失比例，希望根据比例情况采取对应的缺失值处理方法。

```
pd.DataFrame(hdma.isnull().sum()/hdma.shape[0],columns=['miss']).sort_values(by='miss',ascending=False)[:15]
```

输出结果如下：

图5-1

	miss
co_applicant_race_name_5	0.999969
co_applicant_race_name_4	0.999952
applicant_race_name_5	0.999899
applicant_race_name_4	0.999852
co_applicant_race_name_3	0.999748
applicant_race_name_3	0.999327
denial_reason_name_3	0.996518

步骤 3　统计各特征的缺失率

一般缺失比例过高的特征补全缺失值的方法会导致数据信息不准确，对缺失情况大于 80%的特征进行删除操作。

```
del_col=[]
for i,column in enumerate(hdma.columns):
    if hdma[column].isnull().sum()/hdma.shape[0] >0.8:
        del_col.append(column)
```

步骤 4　查看缺失值占比高的特征

```
del_col
```

输出结果如下：

```
['rate_spread',
 'edit_status_name',
 'denial_reason_name_3',
 'denial_reason_name_2',
 'denial_reason_name_1',
 'co_applicant_race_name_5',
 'co_applicant_race_name_4',
 'co_applicant_race_name_3',
 'co_applicant_race_name_2',
 'applicant_race_name_5',
 'applicant_race_name_4',
 'applicant_race_name_3',
 'applicant_race_name_2']
```

缺失比例高的特征有 12 个，且大部分为标称型特征。

步骤 5　删除缺失值占比超过 80%的特征

```
hdma=hdma.drop(columns = del_col)
```

步骤 6　查看缺失比例小于 20%的缺失值情况

```
del_col2=[]
for i,column in enumerate(hdma.columns):
    if ((hdma[column].isnull().sum()/hdma.shape[0] <0.2)
&(hdma[column].isnull().sum()/hdma.shape[0] >0)):
        del_col2.append(column)
hdma_miss=hdma[del_col2]
# 绘制每个特征的分布，自定义数据框数据集全部特征分布图
def plot_distribution(df, cols=5, width=15, height=15, hspace=0.2, wspace=0.5):
    import math
    plt.style.use('seaborn-whitegrid')        #设置绘画图表风格
    fig = plt.figure(figsize=(width,height))   #创建 figure 实例
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace,
hspace=hspace) # 调整图表位置和大小间距
    rows = math.ceil(float(df.shape[1]) / cols)      # ceil 方法向上取整
    for i, column in enumerate(df.columns):          #返回索引和其对应的列名
        ax = fig.add_subplot(rows, cols, i + 1)          # 创建子图，类似于 subplot 方法，返
回的 ax 是坐标轴实际画图的位置，参数（子图总行数，总列数，子图位置）
        ax.set_title(column)                                  # 设置轴的标题
        if df.dtypes[column] == np.object:            # 通过列的类型来区分所选取的图像类型,
            g = sns.countplot(y=column, data=df)     #属性类型为 np.object 时，countplot 使
用条形显示每个分箱器中的观察计数，y 轴上的条形图
            plt.xticks(rotation=25)
        else:
            g = sns.distplot(df[column])     #不属于 np.object 类型即绘制 密度分布图
            plt.xticks(rotation=25)
plot_distribution(hdma_miss, cols=3, width=20, height=20, hspace=0.45, wspace=0.5)
```

输出结果如下：

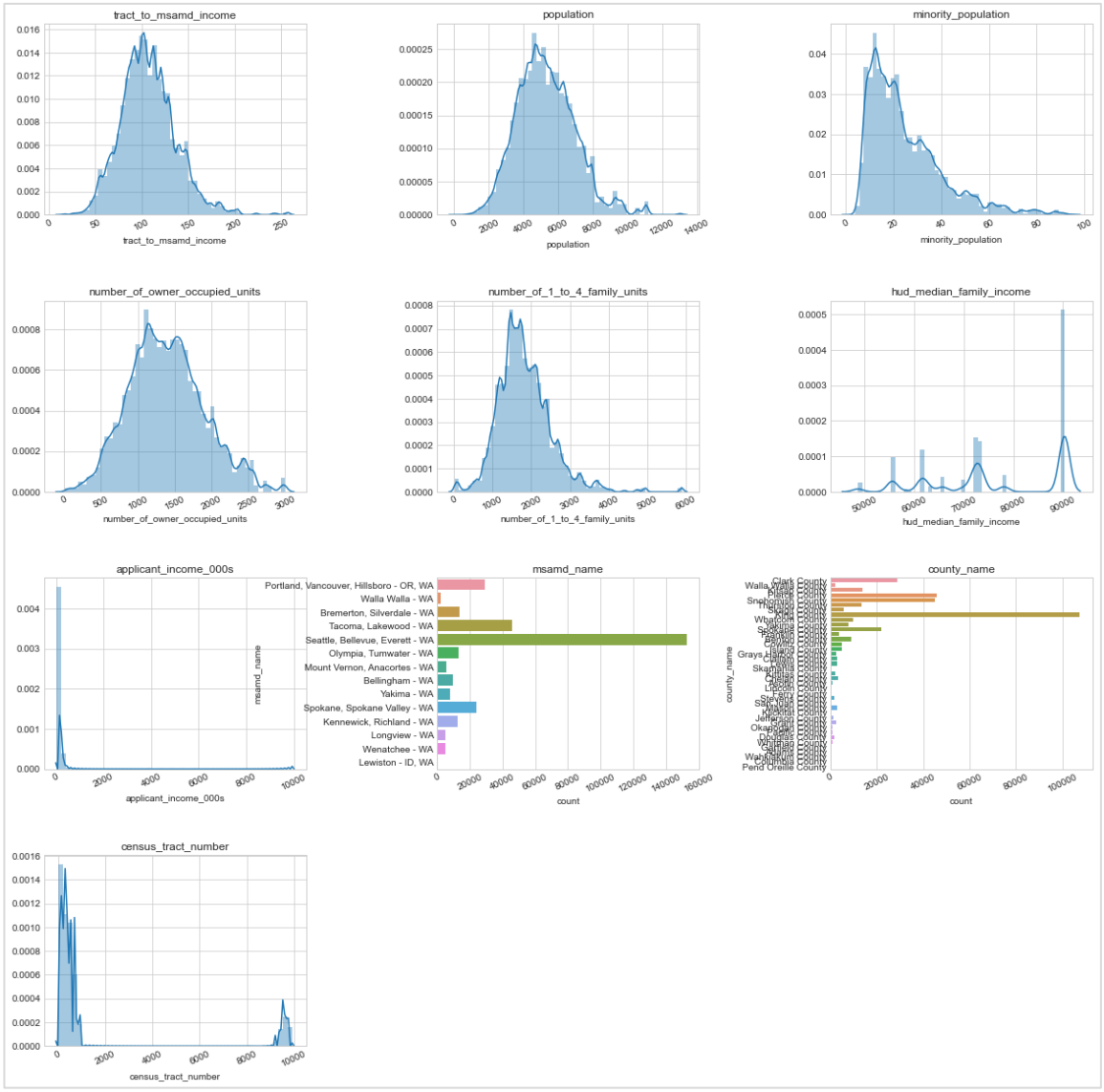


图6-1　　各个特征的统计分布图

步骤 7　少量缺失数据填充

可以看到各个小比例缺失特征的分布并没有非常极端，且因缺失值所在行的总数占比不大（小于 20%总行数），这里选择直接删除所有行。

```
hdma=hdma.dropna()　　#默认滤除任意有缺失值在的行
```

根据上面缺失值分析得出，这里分析的特征列只有极度少数含 nan，这里直接剔除所在行。

步骤 8　查看当前数据尺寸

```
hdma.shape
```

输出数据结果如下：

```
(300175, 33)
```

可以看到数据缺失去除后少了几万行。

2.3.3.4 根据特征类型，按照数值型与标称型分布分别分析

特征列的类型有 int64、float64、object，将这些类型分为两个大类进行数据分布层面的探索和分析，第一类，数值类型，包括 int64 和 float64，第二类字符串类型 object。

步骤 1 查看数值型属性统计性描述

```
hdma.describe()
```

输出数据结果如下：

	tract_to_msamd_income	population	minority_population	number_of_owner_occupied_units	r
count	300175.000000	300175.000000	300175.000000	300175.000000	
mean	107.565574	5318.159723	25.528260	1385.743153	
std	30.113663	1624.899427	15.218358	498.786135	
min	14.050000	98.000000	2.040000	14.000000	
25%	87.209999	4181.000000	14.110000	1033.000000	
50%	105.029999	5192.000000	21.320000	1361.000000	
75%	125.430000	6364.000000	33.220001	1701.000000	
max	257.140015	11041.000000	94.790001	2997.000000	

图7-1 描述性输出结果

步骤 2 标量型特征属性统计性描述

top 描述的是出现频率最高的字段，而 freq 则是描述该字段出现的次数。

```
hdma.describe(include=['O'])
```

输出数据结果如下：

	state_name	state_abbr	respondent_id	purchaser_type_name	property_type_name	preapproval_name
count	300175	300175	300175	300175	300175	300175
unique	1	1	704	10	2	3
top	Washington	WA	451965	Loan was not originated or was not sold in cal...	One-to-four family dwelling (other than manufa...	Not applicable
freq	300175	300175	18708	124245	292780	246852

图8-1 标称属性统计数据查看

步骤 3 某些分布只有 1 个值，对分类结果的影响应该可以视为没有，选择删除（思考下为什么）

根据统计描述，处理分布极端的特征：筛选出来取值只有 1 个值的特征。

```
del_col2=[]
for i, column in enumerate(hdma.columns):
    if len(hdma[column].unique())==1:
        del_col2.append(column)
print(del_col2)
hdma=hdma.drop(columns = del_col2)
hdma.shape
```

输出数据结果如下：

```
['state_name', 'state_abbr', 'as_of_year', 'application_date_indicator']
(300175, 29)
```

可以看到有 3 列没有意义，直接删除。

2.3.3.5 特征相关性分析

步骤 1 数值型特征相关性可视化

对其他数值类型的变量做相关性分析，去除相关性较高的特征之一。

```
plt.figure(figsize=(10, 8))
sns.heatmap(hdma.corr(method='pearson'), annot= True)
```

输出数据结果如下：

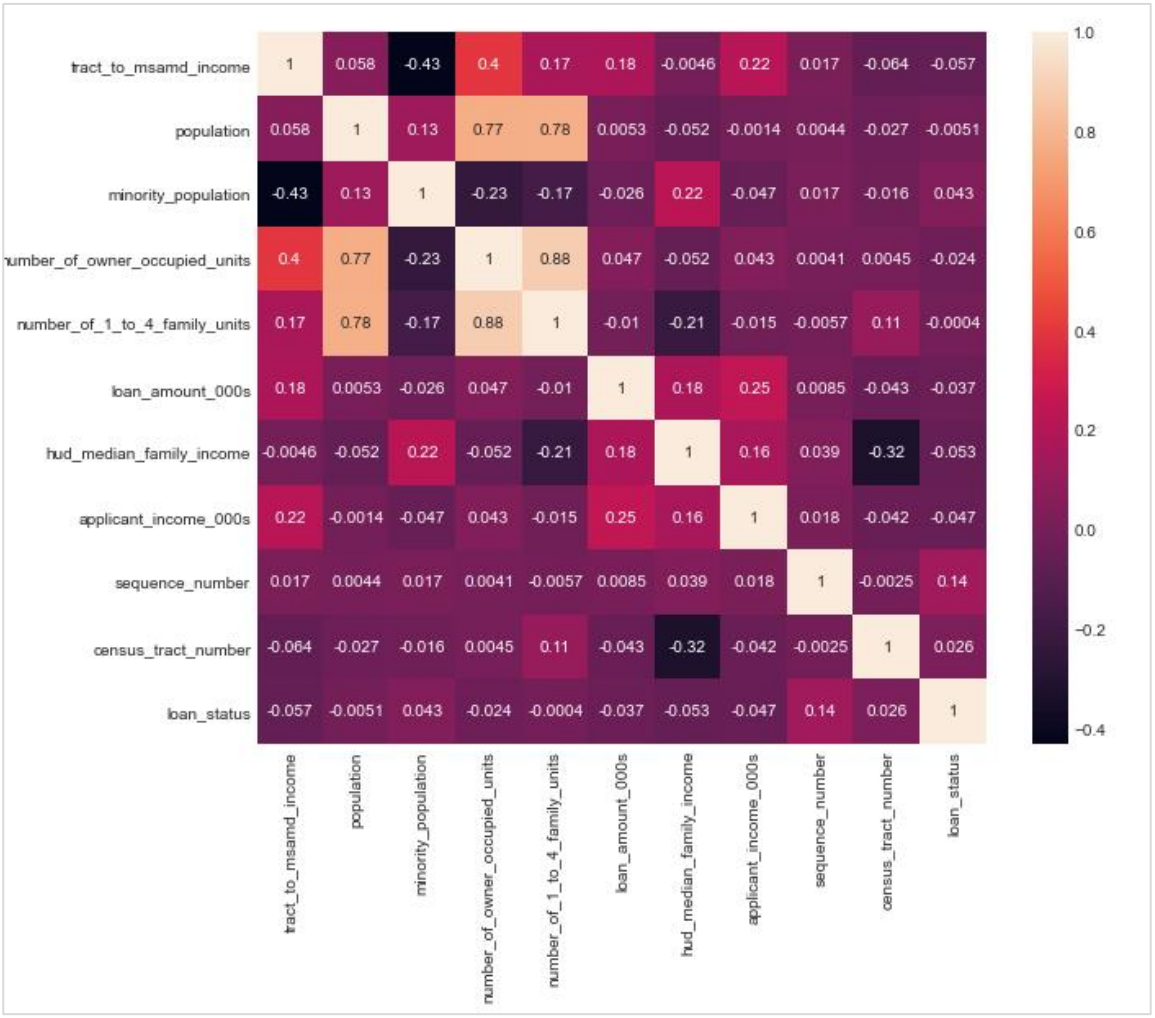


图9-1 数值特征可视化结果

可以看出，'population'、'number_of_owner_occupied_units'、'number_of_1_to_4_family_units'这 3 个特征相关度（大于 70%）很高，可以考虑只保留一个，这里保留 population 字段。

步骤 2 删除相关性高的特征之一

```
del hdma['number_of_owner_occupied_units']
del hdma['number_of_1_to_4_family_units']
```

2.3.3.6 数据压缩：连续属性分箱

步骤 1 筛选出来那些标准差大于 100 的特征

```
bins_col=[]
for i, column in enumerate(hdma.columns):
    if hdma.dtypes[column] != np.object:
        if hdma[column].std()>100:
```

```
                bins_col.append(column)    #记录下符合分箱条件的列
print(bins_col)     ##查看是哪些列被分箱
```

输出数据结果如下：

```
['population', 'loan_amount_000s', 'hud_median_family_income', 'applicant_income_000s',
'sequence_number', 'census_tract_number']
```

步骤 2　波动范围大的数值特征查看

```
plt.figure(figsize=(12,28*2))
gr = gridspec.GridSpec(10, 3)
for i,j in enumerate(bins_col):
        ax = plt.subplot(gr[i])
        sns.distplot(hdma[j][hdma.loan_status == 1], bins=100, color='b')
        sns.distplot(hdma[j][hdma.loan_status == 0], bins=100, color='r')
#         ax.set_title('feature: ' + str(j))
plt.show()
```

输出结果如下：

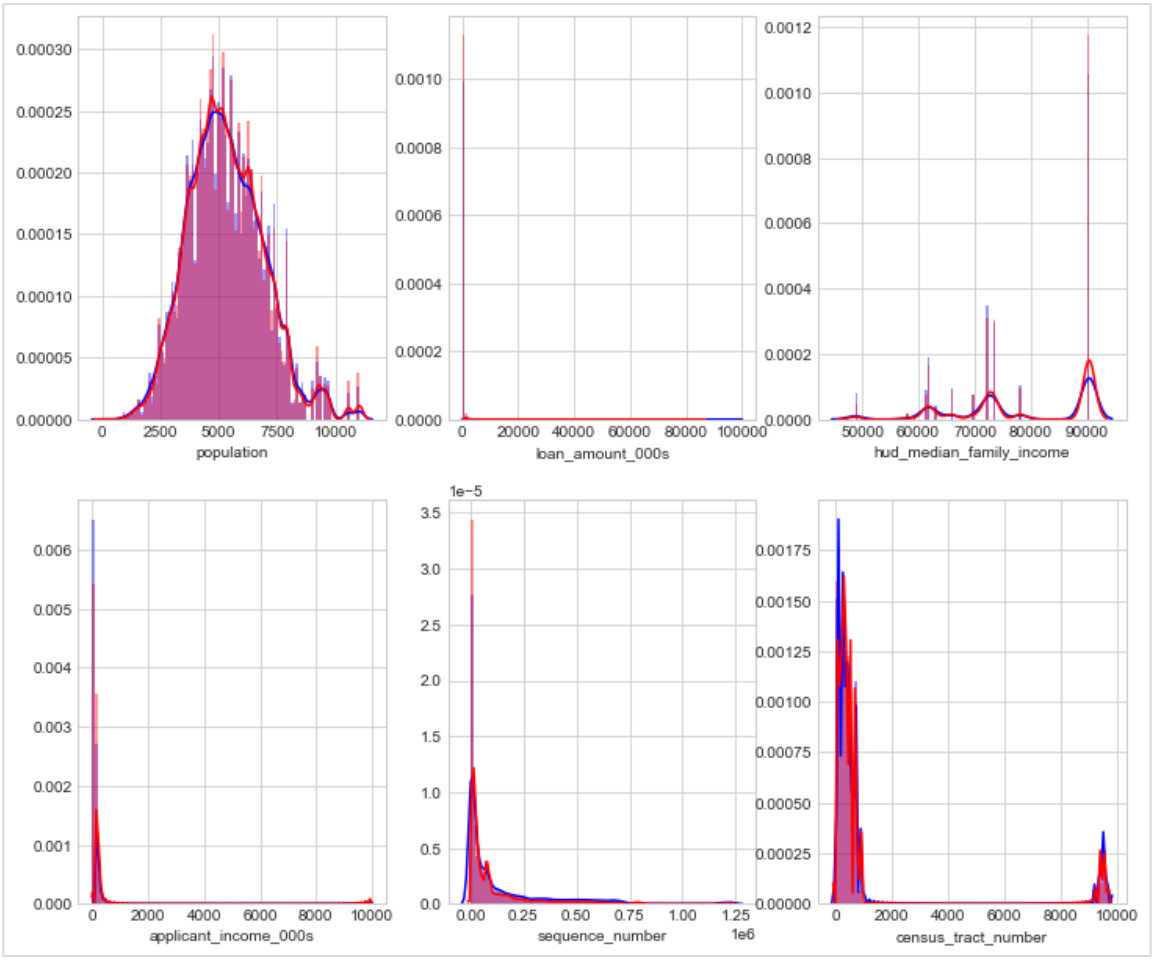


图10-1　波动大特征分布查看

将标准差较大的特征进行分区间，分目标属性值后查看，可以看到特征的取值波动确实比较大，部分特征在部分区间对两个目标的取值分布并不一致（蓝色和红色的密度曲线不一致），且分为 100 箱有一定的区分能力，所以接下来按照划分为 100 个箱子对这些特征进行分箱。

步骤 3 波动范围大的数值特征处理

对数值特征中，标准差大于 100 的特征进行分箱

```
hdma[column]= pd.cut(hdma[column], 100)
```

2.3.3.7 特征编码

考虑到 One-hot 编码会极大的扩充数据维度，造成计算困难，这里先选择 LabelEncoder 方式进行特征标签编码，即将离散型的数据转换成 0 到 n-1 之间的数，n 是一个列表的不同取值的个数，可以认为是某个特征的所有不同取值的个数。如果后续建模的得分不达到预期，可以考虑在此处优化，尝试其他编码方式。

步骤 1 转换为连续型数字变量

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
# print(dataset_con['workclass'])
hdma = hdma.astype(str)
hdma_enc = hdma.apply(encoder.fit_transform)
```

步骤 2 查看编码后的数据描述

```
hdma_enc.describe()
```

输出数据结果如下：

图11-1

	tract_to_msamd_income	population	minority_population	loan_amount_000s	hud_median_family_income	a
count	300175.000000	300175.000000	300175.000000	300175.000000	300175.000000	
mean	563.213852	676.612516	509.082628	879.730351	10.283625	
std	355.129194	312.362060	310.848409	335.436209	3.394708	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	258.000000	432.000000	241.000000	709.000000	9.000000	
50%	500.000000	711.000000	488.000000	906.000000	12.000000	
75%	897.000000	946.000000	751.000000	1073.000000	13.000000	
max	1183.000000	1154.000000	1096.000000	1771.000000	13.000000	

2.3.4 建立模型

本环节选择多个分类算法对数据进行分类预测。

2.3.4.1 数据拆分

步骤 1 将数据拆分为训练数据和测试数据

```
x_cols = [col for col in hdma_enc.columns if col!='loan_status' ]
y_col = 'loan_status'
X=hdma_enc[x_cols]
y=hdma_enc[y_col]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
```

2.3.4.2 建立决策树模型

步骤 1 导入决策树模型

对训练数据进行拟合，再对测试数据进行预测，并且基于模型本身的评估指标计算模型得分。

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
#采用决策树模型进行训练
dt_model.fit(X_train, y_train)
y_pred=dt_model.predict(X_test)
#打印模型评估值
print (dt_model.score(X_test, y_test))
```

输出数据结果如下：

```
0.8812255005385717
```

2.3.4.3 建立随机森林模型

随机森林是决策树的集成算法。随机森林包含多个决策树来降低过拟合的风险。随机森林同样具有易解释性、可处理类别特征、易扩展到多分类问题、不需特征缩放等性质。

随机森林分别训练一系列的决策树，所以训练过程是并行的。因算法中加入随机过程，所以每个决策树又有少量区别。通过合并每个树的预测结果来减少预测的方差，提高在测试集上的性能表现。

模型解读 class sklearn.ensemble.RandomForestClassifier(

n_estimators=10,

criterion='gini',

max_depth=None,

max_features='auto',

n_jobs=1,

random_state=None

)

n_estimators : 随机森林中树的个数，即学习器的个数。

max_features : 划分叶子节点，选择的最大特征数目 。

n_features：在寻找最佳分割时要考虑的特征数量。

max_depth : 树的最大深度，如果选择 default=None，树就一致扩展，直到所有的叶子节点都是同一类样本，或者达到最小样本划分（min_samples_split）的数目。

class_weight：类型权重参数，用于处理类别不平衡问题，由于不同类别数量不同，因此调高权重，还有误分类代价很高情况等。即为每一个类别赋权，默认为 None，即每个类别权重都为 1；'balanced'则自动根据样本集中的类别比例为算法赋权；其他权重设置示例：class_weight={0:0.9, 1:0.1}。

random_state：随机数种子。

步骤 1 导入随机森林算法模块

对训练数据进行拟合，再对测试数据进行预测，并且基于模型本身的评估指标计算模型得分。

```
from sklearn.ensemble import RandomForestClassifier
#配置模型中树的个数为 100
rf_model = RandomForestClassifier (n_estimators=100)
#采用随机森林进行模型训练
rf_model.fit(X_train, y_train)
y_pred=rf_model.predict(X_test)
#打印模型评分结果
print (rf_model.score(X_test, y_test))
```

输出数据结果如下：

```
0.901247043407771
```

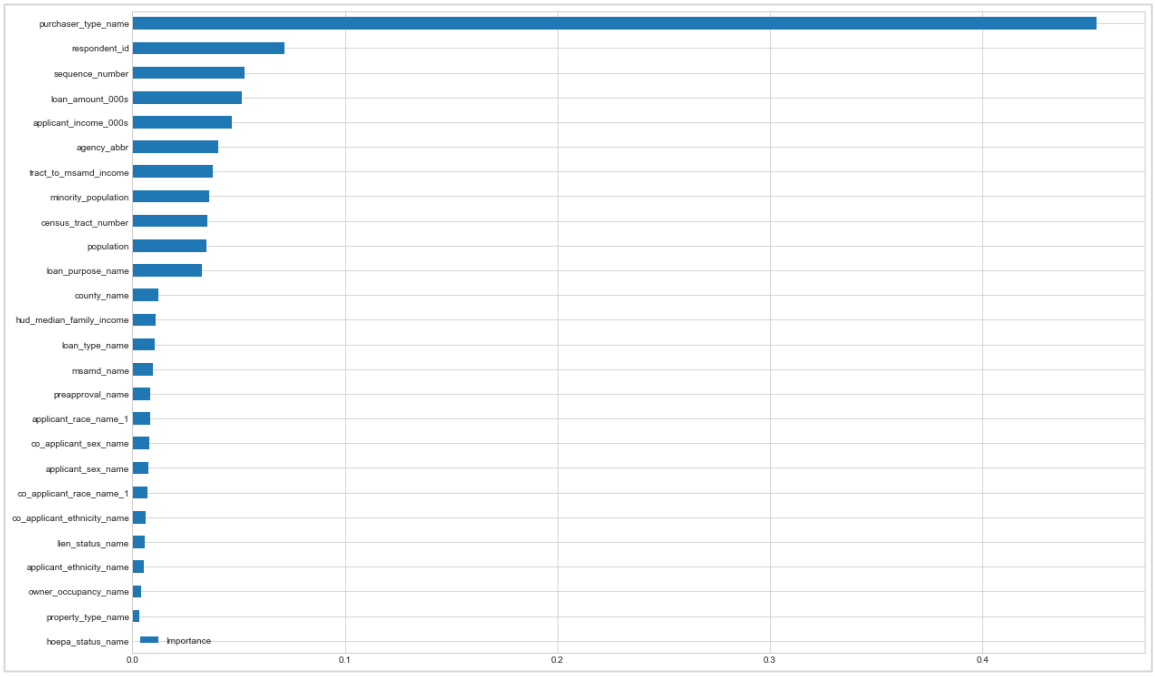
步骤 2 使用随机森林算法去查看属性重要性

```
plt.style.use('seaborn-whitegrid')
importance = rf_model.feature_importances_
importance = pd.DataFrame(importance, index=X.columns, columns=["Importance"])
#根据数值大小排序
```

```
importance.sort_values(by='Importance', ascending=True).plot(kind='barh',
figsize=(20,len(importance)/2))
```

输出数据结果如下：

图12-1



步骤 3　查看重要性突出的特征的分布

在原始数据中取值查看统计量

```
hdma['purchaser_type_name'].value_counts()
```

输出数据结果如下：

```
Loan was not originated or was not sold in calendar year covered by register      124245
Fannie Mae (FNMA)
61291
Freddie Mac (FHLMC)
39450
Ginnie Mae (GNMA)
26251
Life insurance company, credit union, mortgage bank, or finance company      17798
Commercial bank, savings bank or savings association      15883
Other type of purchaser
10506
Affiliate institution      3186
Private securitization
1552
Farmer Mac (FAMC)
13
```

```
Name: purchaser_type_name, dtype: int64
```

可以看到在对结果影响最大的特征'purchaser_type_name'中，'Loan was not originated or was not sold in calendar year covered by register'占比最高，即'贷款并未真正有效执行'这个结果，其实与最终分类结果 0 是一致的，所以对结果的影响很大。其他的类型就是对应各个贷款中介公司。

2.3.4.4 建立 GBDT 模型

步骤 1 导入 GBDT 模型

GBDT 的核心就在于，每一棵树学习的是之前所有树结论和的残差，这个残差就是一个加预测值后能得真实值的累加量。

直接调用 sklearn 中的 GBDT 分类器对训练数据进行拟合，再对测试数据进行预测，并且基于模型本身的评估指标计算模型得分。

```
from sklearn.ensemble import GradientBoostingClassifier
#配置 GBDT 分类器个数
gbdt_model = GradientBoostingClassifier(n_estimators=100)
#采用训练数据集进行模型训练
gbdt_model.fit(X_train, y_train)
y_pred=gbdt_model.predict(X_test)
#输出模型评估值
print (gbdt_model.score(X_test, y_test))
```

输出数据结果如下：

```
0.8925299545823016
```

2.3.4.5 建立 XGBoost 模型

作为 GBDT 的高效实现，XGBoost 是一个上限特别高的算法，在算法竞赛中比较受欢迎。对比原算法 GBDT，XGBoost 主要从下面三个方面做了优化：

一是算法本身的优化：在算法的弱学习器模型选择上，对比 GBDT 只支持决策树，还可以直接很多其他的弱学习器。在算法的损失函数上，除了本身的损失，还加上了正则化部分。在算法的优化方式上，GBDT 的损失函数只对误差部分做负梯度（一阶泰勒）展开，而 XGBoost 损失函数对误差部分做二阶泰勒展开，更加准确。

二是算法运行效率的优化：对每个弱学习器，比如决策树建立的过程做并行选择，找到合适的子树分裂特征和特征值。在并行选择之前，先对所有的特征的值进行排序分组，方便前面说的并行选择。对分组的特征，选择合适的分组大小，使用 CPU 缓存进行读取加速。将各个分组保存到多个硬盘以提高 IO 速度。

三是算法健壮性的优化：对于缺失值的特征，通过枚举所有缺失值在当前节点是进入左子树还是右子树来决定缺失值的处理方式。算法本身加入了 L1 和 L2 正则化项，可以防止过拟合，泛化能力更强。

步骤 1 导入 XGBoost 模型

对训练数据进行拟合，再对测试数据进行预测，并且基于模型本身的评估指标计算模型得分。

```
#导入 XGBoost 分类模型

import xgboost as xgb   #主要是 c 语言实现的，本实验是通过 python 接口调用
xgb_model_1 = xgb.XGBClassifier(
max_depth=6                    #构建树的深度，越大越容易过拟合,需要使用 CV 函数来进行调优。 典型值：3-10
,learning_rate=0.3             #学习率
,n_estimators=100              #树的个数
,scale_pos_weight=1            #两种类别的权重设置，如果设置为大于 1，会加快收敛,
,objective='binary:logistic'   #设置目标函数，此处为二分类对应的目标函数，输出对应分类结果的概率
,eval_metric='auc'             #分类任务(默认 error)  ,auc--roc 曲线下面积
,subsample=1                   #控制对于每棵树，随机采样的比例。 典型值：0.5-1
,seed=123                      #随机数种子,用于产生可复现的结果
)

#采用训练数据集进行模型训练
xgb_model_1.fit(X_train, y_train)
y_pred=xgb_model_1.predict(X_test)
#输出模型评估值
print (xgb_model_1.score(X_test, y_test))
```

输出数据结果如下：

```
0.9100973870942667
```

可以看到 XGBoost 在四个算法中预测结果比较突出，且运行效率也较高，接下来考虑选择 XGBoost 算法作为最终模型算法，从模型泛化能力和预测精度两个方面对模型进行提升。

步骤 2 选择 XGBoost 算法进行交叉验证

首先通过交叉验证的方式检查样本分布不均是否对模型造成影响，用 f1_socre 做评判标准。

交叉验证评估原始模型的优势：

1：交叉验证用于评估模型的预测性能，尤其是训练好的模型在新数据上的表现，可以在一定程度上减小过拟合。

2：还可以从有限的数据中获取尽可能多的有效信息。

具体函数说明：sklearn.cross_validation.cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_params=None,pre_dispatch='2*n_jobs')

estimator:估计方法对象(分类器)

X：数据特征(Features)

y：数据标签(Labels)

soring：调用方法(包括 accuracy 和 mean_squared_error 等等)

cv：几折交叉验证

n_jobs：同时工作的 cpu 个数（-1 代表全部）

返回值是：每次运行交叉验证的估计值数组。

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import recall_score,roc_auc_score,make_scorer,f1_score
from sklearn.model_selection import cross_val_score,GridSearchCV

cv_scores = cross_val_score(xgb_model_1, X_train, y_train,scoring=make_scorer(f1_score) ,cv=3)
print('mean f1_score socre of raw model {}'.format(np.mean(cv_scores)))
```

输出数据结果如下：

```
mean f1_score socre of raw model 0.8196795687334392
```

2.3.5 模型评估与优化

2.3.5.1 网格搜索参数调整

对最初性能表现突出的模型进行调参，选出 XGBoost 算法进行最后预测模型，对其参数进行网格搜索。网格搜索（GridSearch）用于选取模型的最优超参数。获取最优超参数的方式可以绘制验证曲线，但是验证曲线只能每次获取一个最优超参数。如果多个超参数有很多排列组合的话，就可以使用网格搜索寻求最优超参数的组合。 该方法的常用参数说明如下：

estimator：选择使用的分类器，并且传入除需要确定最佳的参数之外的其他参数；

param_grid：需要最优化的参数的取值，值为字典或者列表，键命名必须为模型本身的参数名称；

scoring：模型评价标准，默认 None(即选择的 estimator 的误差估计函数)。常为 'f1','accuracy','recall','roc_auc',‘average_precision’；

cv=None，交叉验证参数，默认 3；

尝试找的较匹配的参数，进而提升模型性能 根据前面的参数设置，仅从 XGBoost 中 3 个比较主要的参数入手进行网格搜索，这三个参数的基本描述为：

- n_estimators #树的个数
- max_depth #构建树的深度，越大越容易过拟合
- learning_rate #学习率

步骤 1 引入时间模块，进行计时

在这一环节考虑交叉验证和网格搜索比较耗时，引入记时模块，统计时间，作为后续参数设置和模型效率评估的参考。

```
import time,datetime
# 训练开始计时
start_time = time.time()
```

步骤 2 利用坐标下降的方式，对 XGBoost 模型的关键参数进行搜索，尝试获取更好的建模结果

网格搜索过渡方法-坐标下降：拿当前对模型影响最大的参数调优，直到最优化；再拿下一个影响最大的参数调优，如此下去，直到所有的参数调整完毕。这个方法的缺点就是可能会调到局部最优而不是全局最优，但是省时间省力。如下即使用坐标下降的思路进行超参数调优。

```
#首先对参数 n_estimators 进行搜索
param_test1 = {'n_estimators':range(90,151,20)}
gsearch1 = GridSearchCV(estimator = xgb_model_1,
                        param_grid = param_test1,
                        scoring=make_scorer(f1_score)
                       )
gsearch1.fit(X_train,y_train)
print('best params:{}'.format(gsearch1.best_params_))
print('best score:{}'.format(gsearch1.best_score_))
# 训练结束计时
print("Running Time: %s" % datetime.timedelta(seconds=(time.time() - start_time)))
```

输出数据结果如下：

```
best params:{'n_estimators': 150}
best score:0.8223933672226128
Running Time: 0:07:13.605446
```

步骤 3　对参数 max_depth 进行搜索

```
param_test2 = {'max_depth':range(3,10,2)}
gsearch2 = GridSearchCV(estimator = gsearch1.best_estimator_,
                        param_grid = param_test2,
                        scoring=make_scorer(f1_score))
gsearch2.fit(X_train,y_train)
print('best params2:{}'.format(gsearch2.best_params_))
print('best score:{}'.format(gsearch2.best_score_))
```

输出数据结果如下：

```
best params2:{'max_depth': 7}
best score:0.8217874608317808
```

步骤 4　对参数 learning_rate 进行搜索

```
param_test3 = {'learning_rate':[0.05,0.1,0.3,0.5,0.7]}
gsearch3 = GridSearchCV(estimator = gsearch2.best_estimator_,
                        param_grid = param_test3,
                        scoring=make_scorer(f1_score))
gsearch3.fit(X_train,y_train)
print('best params3:{}'.format(gsearch3.best_params_))
print('best score:{}'.format(gsearch3.best_score_))
```

输出数据结果如下：

```
best params3:{'learning_rate': 0.3}
best score:0.8217874608317808
```

步骤 5　模型保存

```
from sklearn.externals import joblib
joblib.dump(gsearch3.best_estimator_,'xgb_model.pkl')
```

输出数据结果如下：

```
['xgb_model.pkl']
```

步骤 6　利用最终确认的模型进行训练

```
model_load = joblib.load('xgb_model.pkl')
y_test_pred = model_load.predict(X_test)
print('f1 score of random forest score:{}'.format(f1_score(y_test,y_test_pred)))
```

输出数据结果如下：

```
f1 score of random forest score:0.8261861914585437
```

2.3.5.2 XGBoost 模型小结

该模型本身的参数优化 效果一般不是特别大，更为重要的是数据清洗和特征工程。

2.4 思考题

1、 “分析目标变量，建模问题简化”步骤中如果不对目标变量的输出结果进行简化，这个问题会是一个什么问题?

2、 本实验中对缺失值占比少的特征，采取直接删除其行的操作，是否有其他的方法？从提高精确度的角度，你建议采用什么方法?

3、 本实验中采用了 LabelEncoder 方式进行特征标签编码，还有其他方法吗？这个方法的缺点是什么?

4、 本实验中通过随机森林建模后，输出各个特征的重要性排名，发现重要性占比最高的特征’purchaser_type_name’中，’Loan was not originated or was not sold in calendar year covered by register’占比最高，即‘贷款并未真正有效执行’这个结果。针对这个结论，从优化模型的角度可以有哪些处理?

【答案】

2.5 实验小结

本实验主要介绍了如何针对实际分类问题进行建模分析，同时采用多种方法进行模型对比，最终选择 XGBoost 算法进行模型训练，然后对客户申请贷款产品的结果进行预测。在模型参数优化方面，采用网格搜索的方式进行最优化模型参数的搜索，从而找到最优化模型参数，有效提升了模型的准确率。实验结果表明采用该流程能够对此类分类问题进行预测分析，效果比较好，可以作为解决此类数据分析挖掘算法的一种有效手段。当然模型准确率还有提升的空间，学习者可以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，获取更好的分类模型。

2.6 思考题-汇总

1、 “分析目标变量，建模问题简化”步骤中如果不对目标变量的输出结果进行简化，这个问题会是多分类问题。

2、 本实验中对缺失值占比少的特征，还可以对每个缺失特征对目标变量影响进行分析，尝试用均值、众数或者常数方式填充，通过判断最后建模结果的得分高低，来确认最好的处理方式。

3、 采用ONE-Hot方法，LabelEncoder方法会引入顺序关系，其实是增加了人为误差，对于非分类的算法并不适用。

4、 可以在最后的建模特征中删除’purchaser_type_name’这一列，因为该特征含义其实与目标一样，属于贷款产品的结论，且对结果的影响极大，这样很难真正再挖掘出影响贷款成功的原因。删除后从新建模，查看建模得分是否可以提高，或者通过PCA降维对全部特征进行处理后再建模。 （答案不唯一）

2.7 创新设计

本实验使用决策树、随机森林、GBDT、XGBoost 四种算法，请以算法复杂度为评价指标，比较四种算法的优劣。

3 房价预测

3.1 房价预测实验介绍

本实验主要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，从而为此类价格类实际问题的处理提供技术参考。这本质上是一个回归问题。

本地离线数据集：kc_house_data.csv

数据详情可查阅如下网址了解：

https://www.kaggle.com/harlfoxem/housesalesprediction?select=kc_house_data.csv

图13-1 Kaggle Housing Prices 数据介绍

【实验环境要求】：

1、 python3.7

2、 ModelArts 平台

本实验在 python3.7 环境下完成，可下载 Anaconda，下载地址为：https://www.anaconda.com/distribution/。

3.2 实验总体设计

本实验遵从数据挖掘的一般流程，首先对已经下载本地的数据进行读取，常规的探索后，进行数据预处理，随后直接选择 sklearn 模块中的决策树、随机森林、GDBT、XGBoost 算法进行建模，选择出性能突出的模型做进一步的调参优化，最终确认模型，进行预测。

3.2.1 实验整体方案

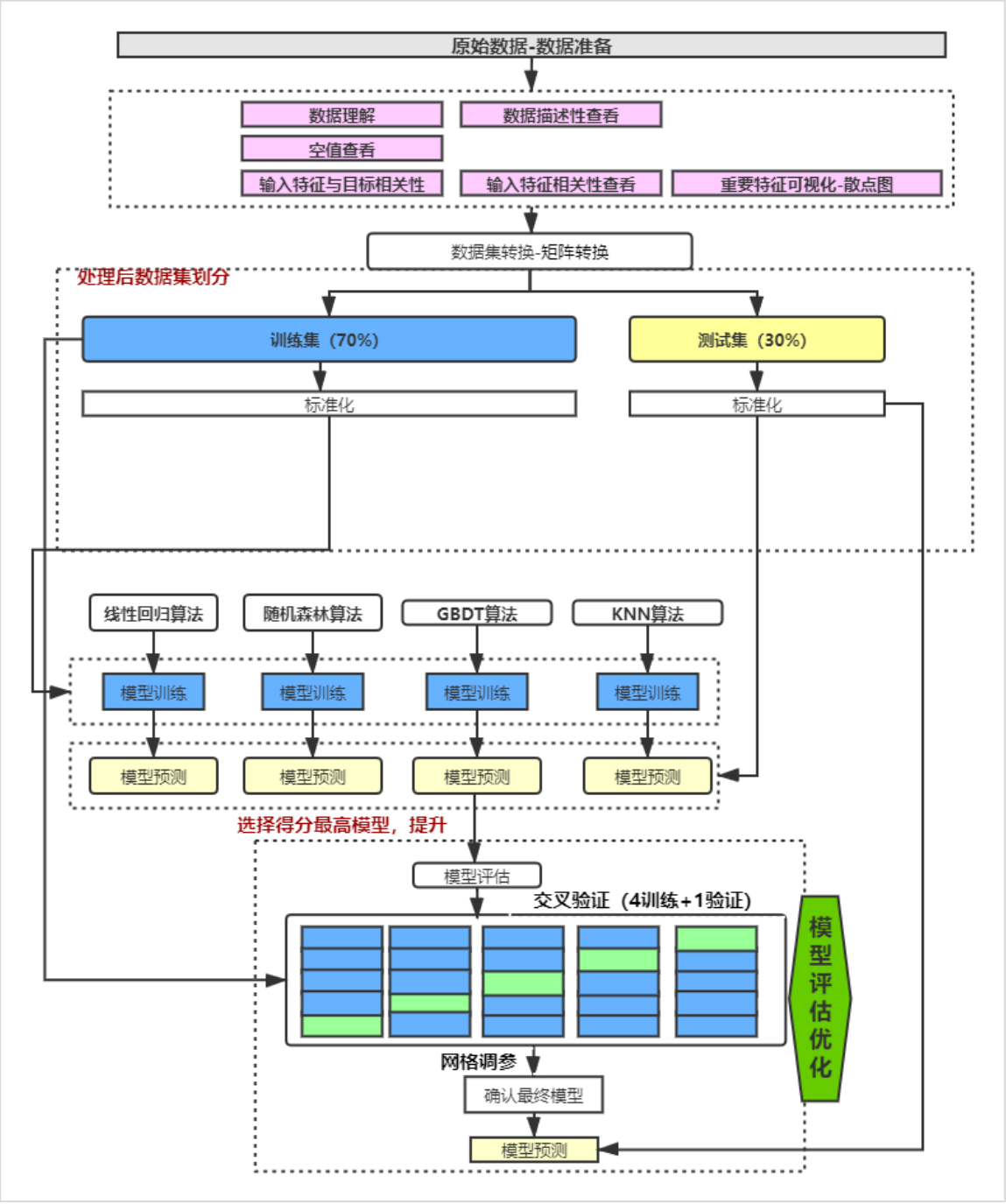


图14-1 实验方案流程图

3.3 实验详细设计与实现

3.3.1 导入实验环境

步骤 1 导入相应的模块

本实验使用到的框架主要包括 numpy，pandas，scikit-learn，matplotlib，seaborn 库。scikit-learn 库是 Python 的机器学习库，提供一些常用的机器学习算法模型及模型参数优化功能；numpy ，pandas 库是 Python 中结构化数据处理的库，主要用于结构化数据的统计分析及操作；matplotlib，seaborn 主要用于数据分析过程的可视化展示。

```
#加载 Python 库
import numpy as np
#加载数据预处理模块
import pandas as pd
#加载绘图模块
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style(style="darkgrid")
```

3.3.2 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。

```
df = pd.read_csv("kc_house_data.csv")
```

步骤 2 输出前 5 行数据

查看文件头信息，了解基本的数据记录，查看每条记录具体包含哪些内容。

```
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。

3.3.3 数据理解

步骤 1 输出属性信息

```
print(df.info())
```

输出结果如下：

```
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
id                21613 non-null int64
date              21613 non-null object
price             21613 non-null float64
bedrooms          21613 non-null int64
bathrooms         21613 non-null float64
sqft_living       21613 non-null int64
sqft_lot          21613 non-null int64
floors            21613 non-null float64
waterfront        21613 non-null int64
view              21613 non-null int64
condition         21613 non-null int64
grade             21613 non-null int64
sqft_above        21613 non-null int64
sqft_basement     21613 non-null int64
yr_built          21613 non-null int64
yr_renovated      21613 non-null int64
zipcode           21613 non-null int64
lat               21613 non-null float64
long              21613 non-null float64
sqft_living15     21613 non-null int64
sqft_lot15        21613 non-null int64
dtypes: float64(5), int64(15), object(1)
```

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。

步骤 2 输出描述信息

查看属性的统计描述信息，了解每种属性的记录数，区间范围，均值，分位值，方差，用于了解属性的分布和倾斜情况，为后续数据的分析和处理服务。

```
print(df.describe())
```

输出如下结果：

```
                 id          price    ...    sqft_living15     sqft_lot15
```

```
count    2.161300e+04    2.161300e+04          ...          21613.000000    21613.000000
mean     4.580302e+09    5.400881e+05          ...           1986.552492    12768.455652
std      2.876566e+09    3.671272e+05          ...            685.391304    27304.179631
min      1.000102e+06    7.500000e+04          ...            399.000000      651.000000
25%      2.123049e+09    3.219500e+05          ...           1490.000000     5100.000000
50%      3.904930e+09    4.500000e+05          ...           1840.000000     7620.000000
75%      7.308900e+09    6.450000e+05          ...           2360.000000    10083.000000
max      9.900000e+09    7.700000e+06          ...           6210.000000   871200.000000
```

上述输出属性的统计信息，主要输出记录数量，属性均值，方差，最小值，25%分位值，50%分位值，75%分位值，和最大值，可以看出每种属性的统计信息，通过对比 mean 值和 50%值，可以看出部分属性出现略有倾斜的情况，比如 sqft_living15，但是也有部分属性出现严重倾斜的情况比如 sqft_lot15，倾斜情况比较严重。针对类似问题，若是存在空值情况，则需要通过众数进行缺失值的填充。数据预处理

步骤 3 输出空值信息

```
#输出数据空值情况
print(df.isnull().any()) #这里主要调用 DataFrame 中的 isnull 方法进行属性空值检测
```

输出如下结果：

```
id               False
date             False
price            False
bedrooms         False
bathrooms        False
sqft_living      False
sqft_lot         False
floors           False
waterfront       False
view             False
condition        False
grade            False
sqft_above       False
sqft_basement    False
yr_built         False
yr_renovated     False
zipcode          False
lat              False
long             False
sqft_living15    False
sqft_lot15       False
dtype: bool
```

从上述结果可以看出没有出现空值现象，说明属性完整情况较好。

步骤 4 查看每种属性与房价的分布关系

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。

```
#获取第三列开始往后的所有属性名称，由于第一列为序号，第二列为房屋记录时间，第三列房屋价格
#因此此处从第四列开始获取属性集。
x_vars=df.columns[3:]
#分别分析所取的属性与价格的分布关系图
for x_var in x_vars:
    df.plot(kind='scatter',x=x_var,y='price')  #设置绘图的行和列
plt.show()
```

输出如下结果：

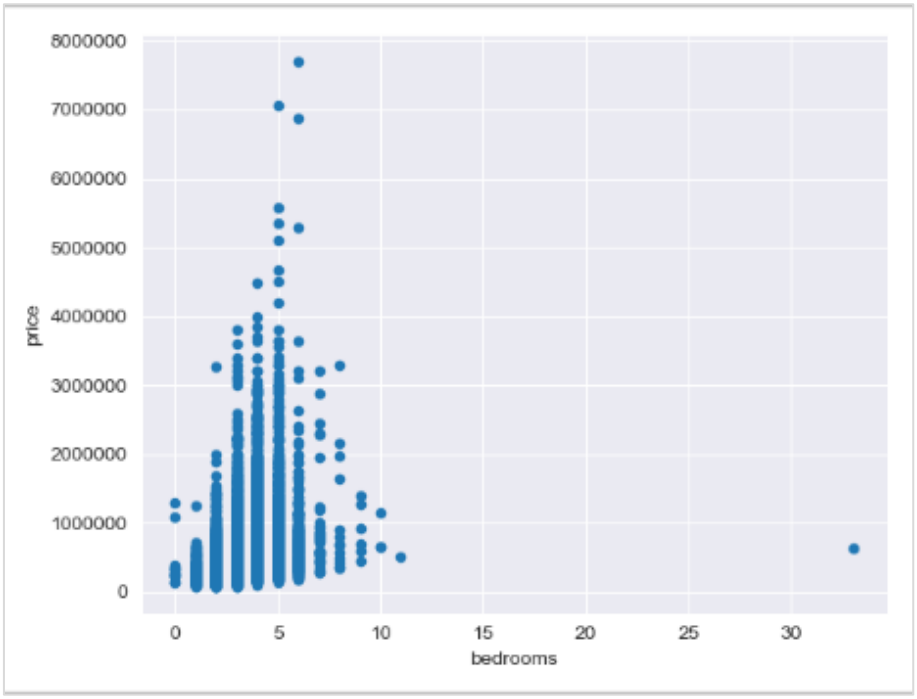


图15-1 卧室数与房价散点图

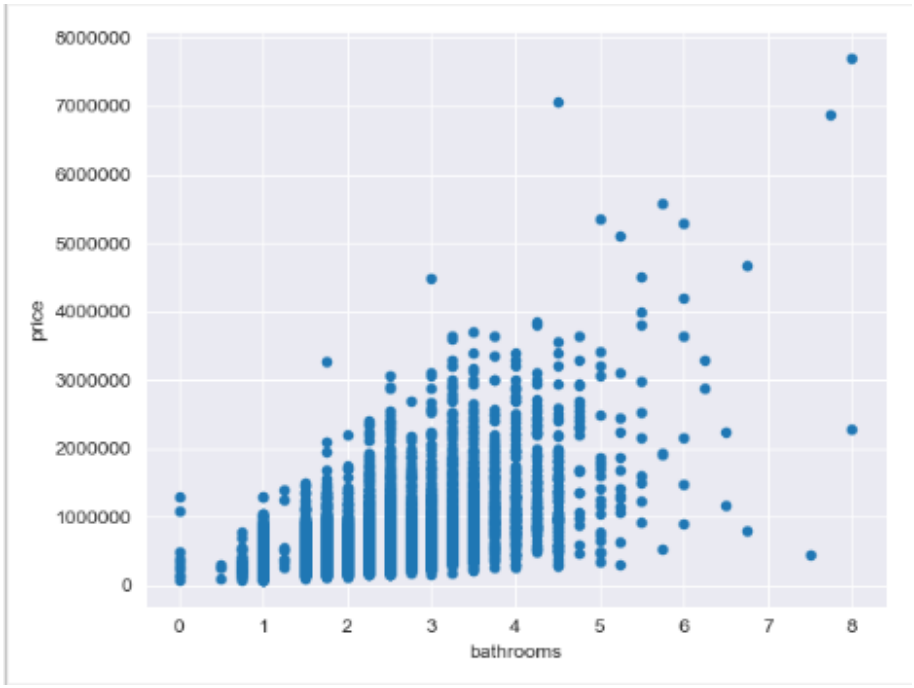


图16-1　卫生间数与房价散点图

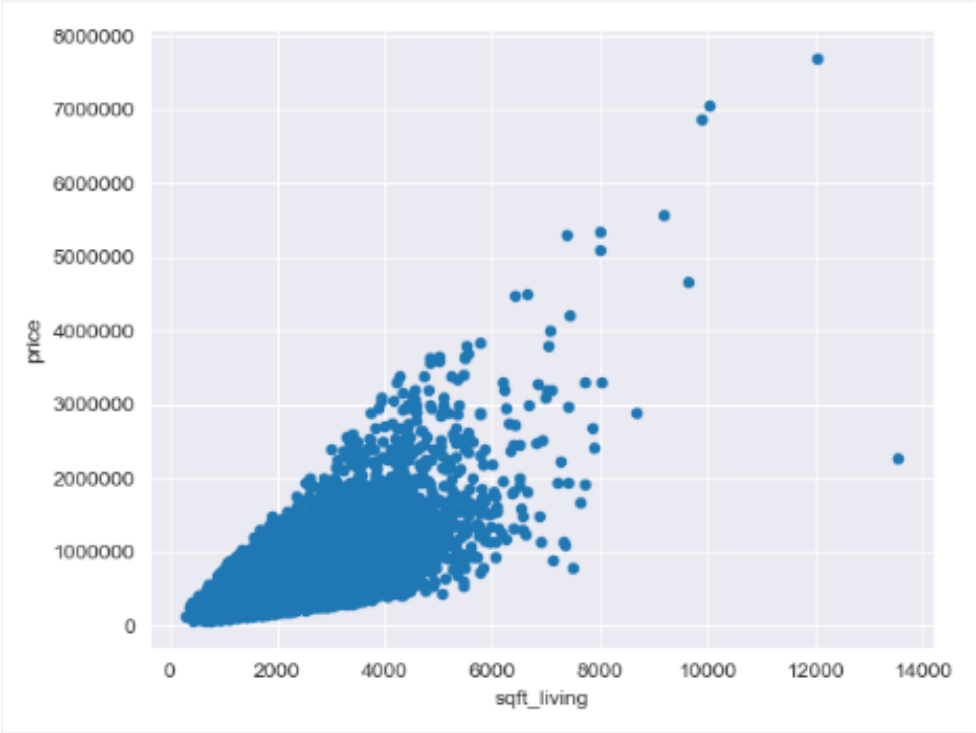


图17-1　住宅面积与房价散点图

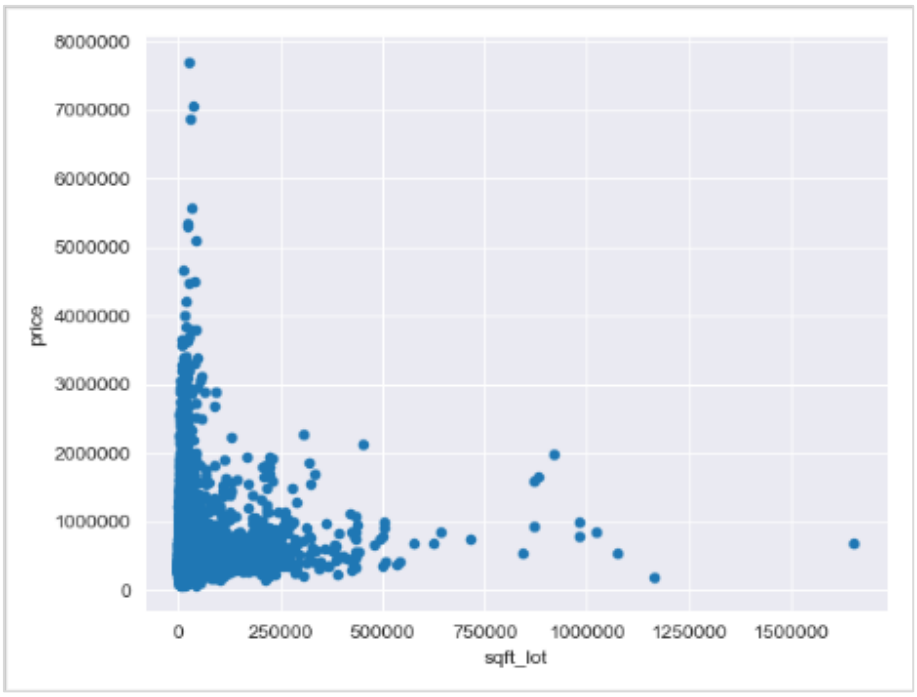


图18-1　停车场面积与房价散点图

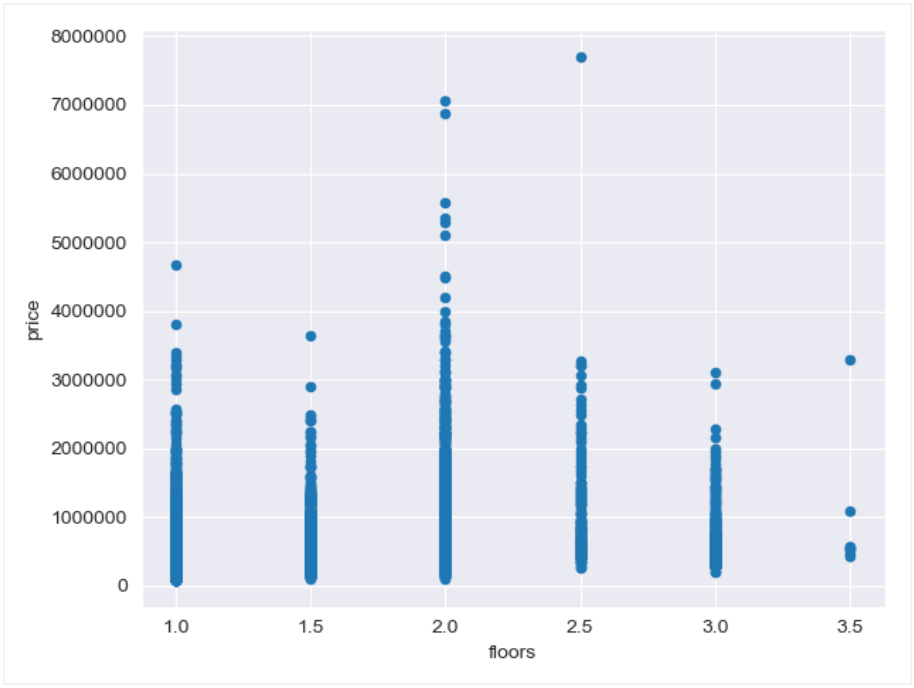


图19-1　楼层数与房价散点图

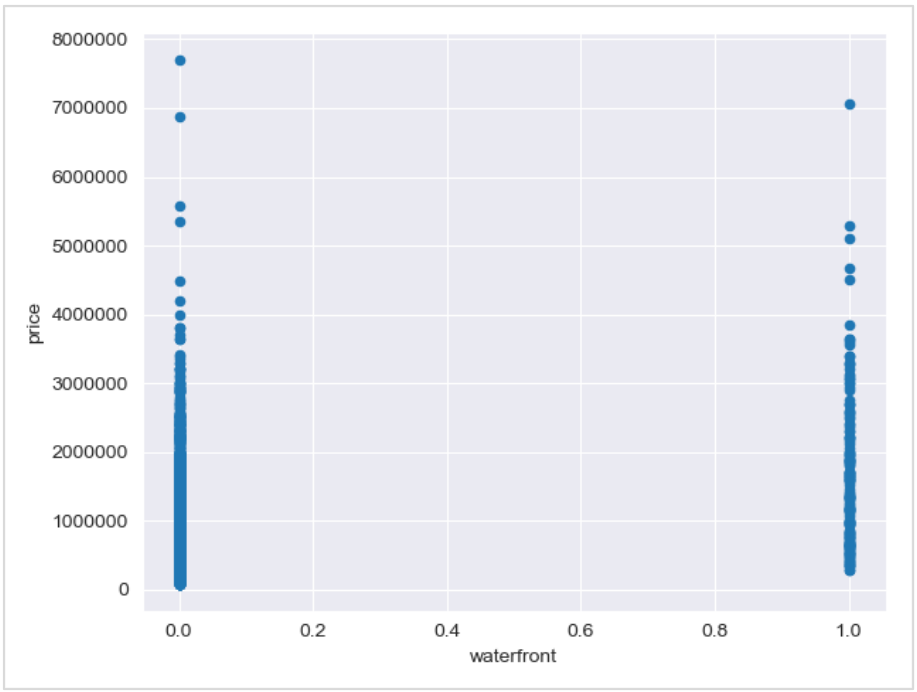


图20-1　泳池与房价散点图

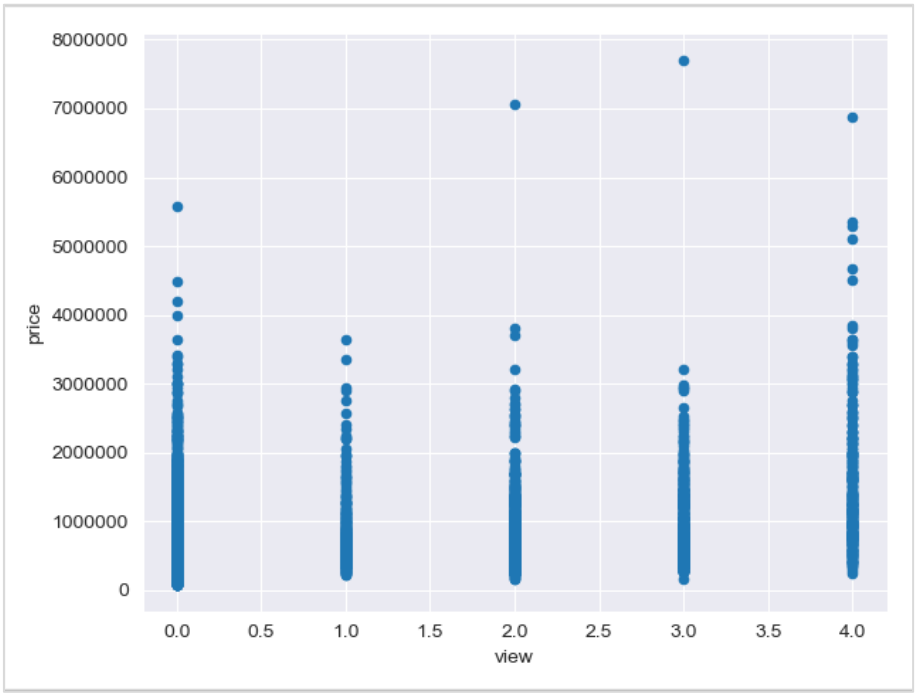


图21-1　外观与房价散点图

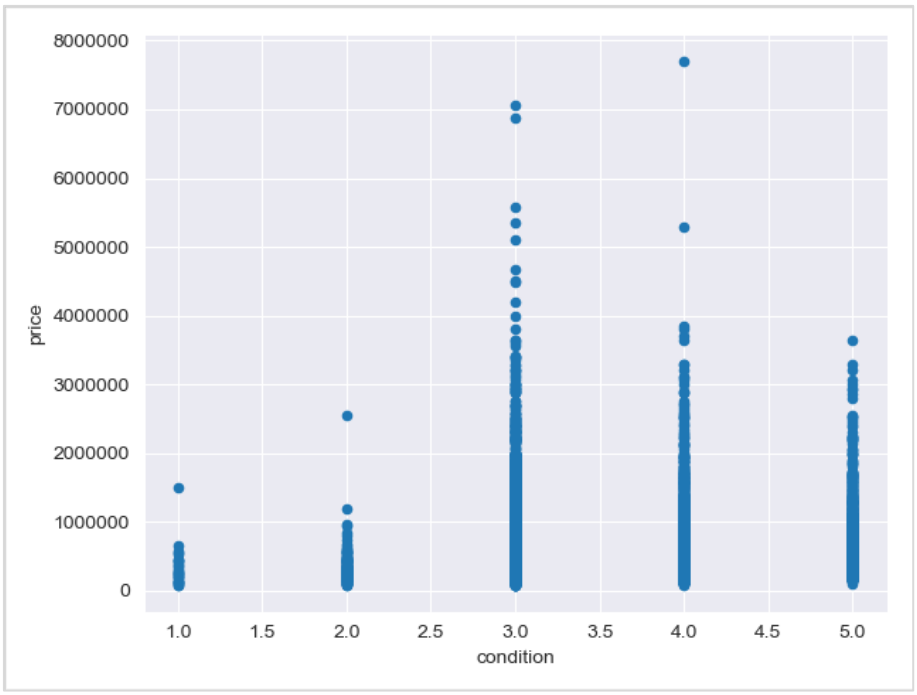


图22-1　房屋状态与房价散点图

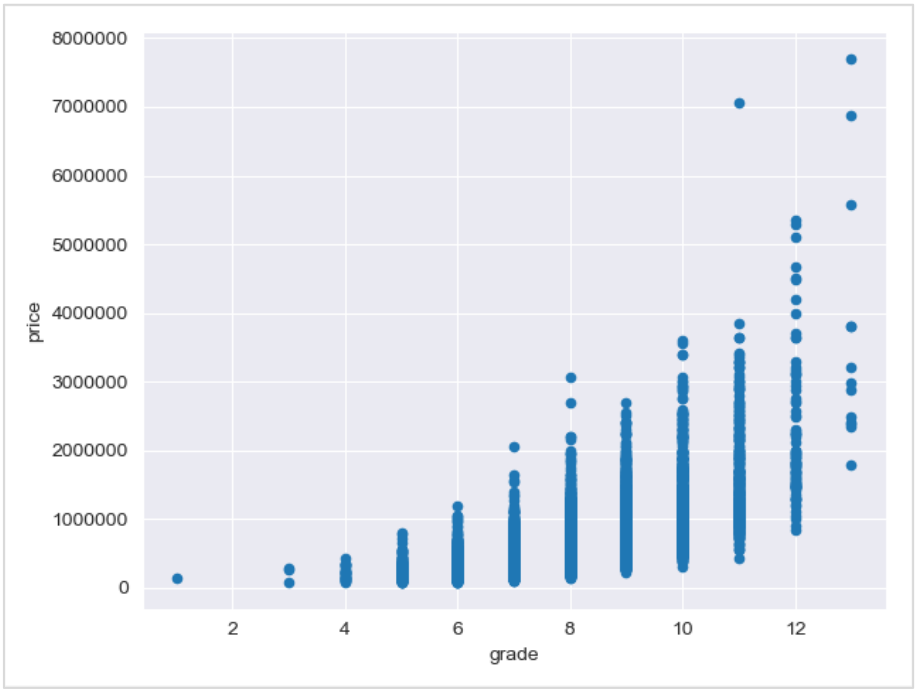


图23-1　房屋等级与房价散点图

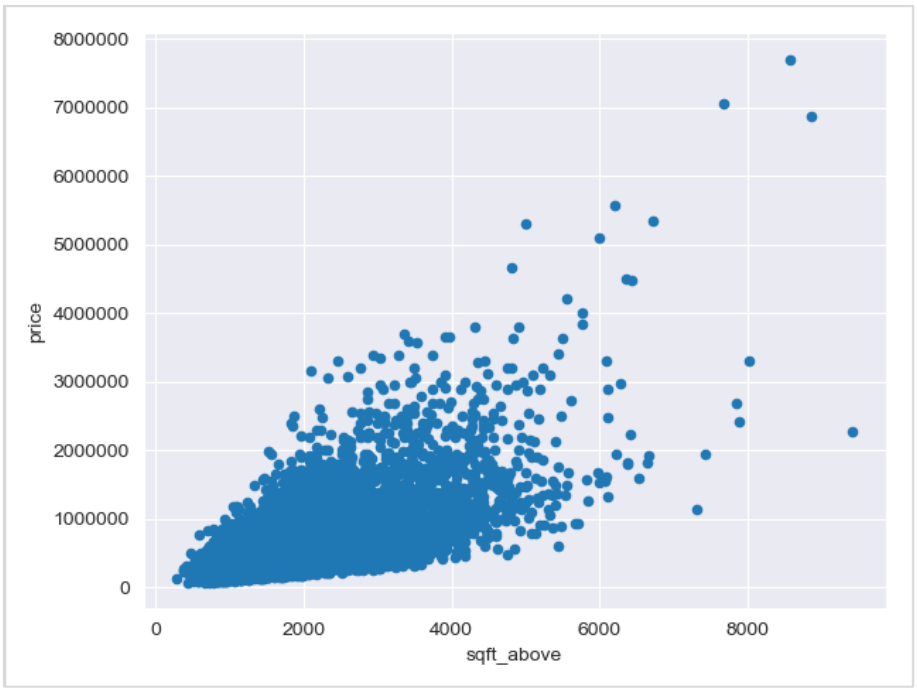


图24-1　地上面价与房价散点图

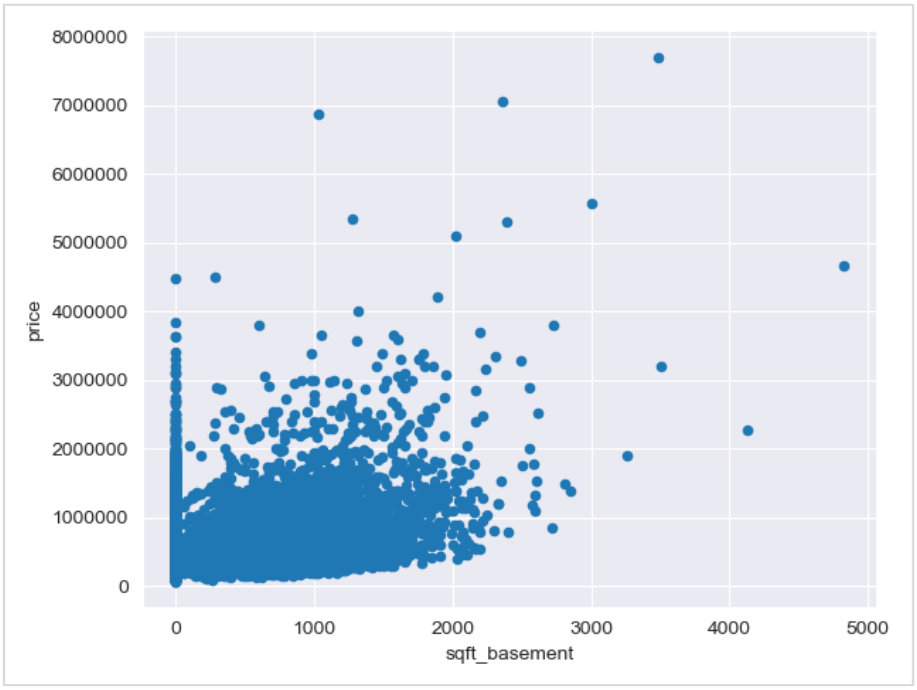


图25-1　地下室面积与房价散点图

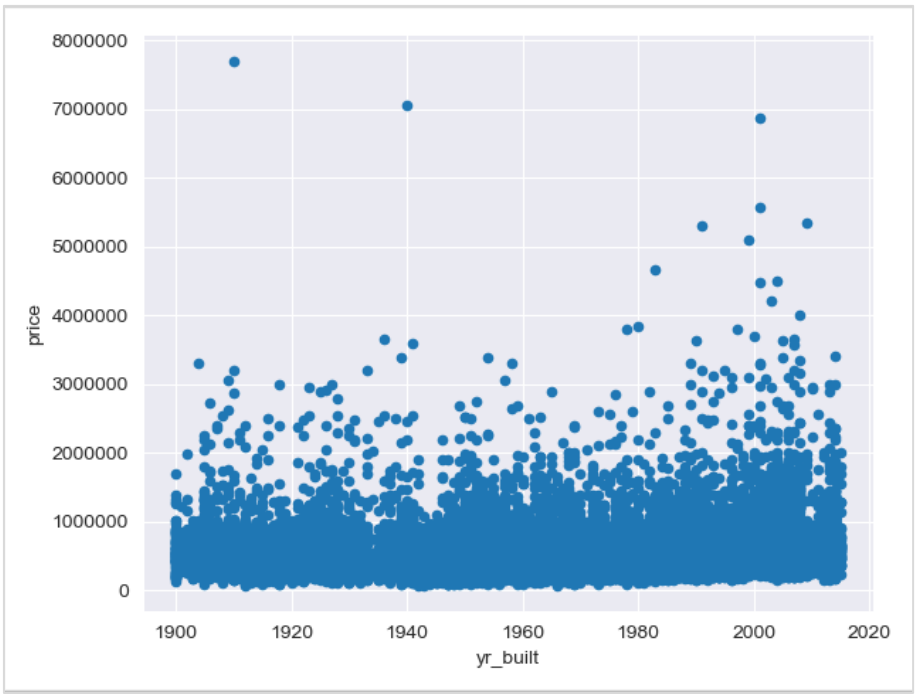


图26-1　修建时间与房价散点图

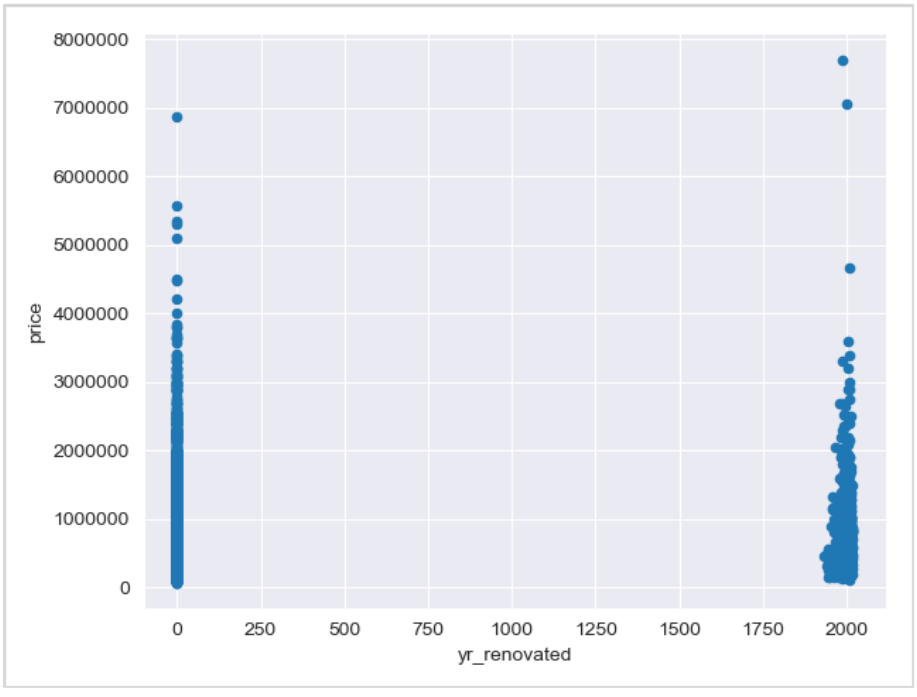


图27-1　翻修时间与房价散点图

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。

步骤 5　查看属性之间的相关性

```
#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()
```

输出如下结果：

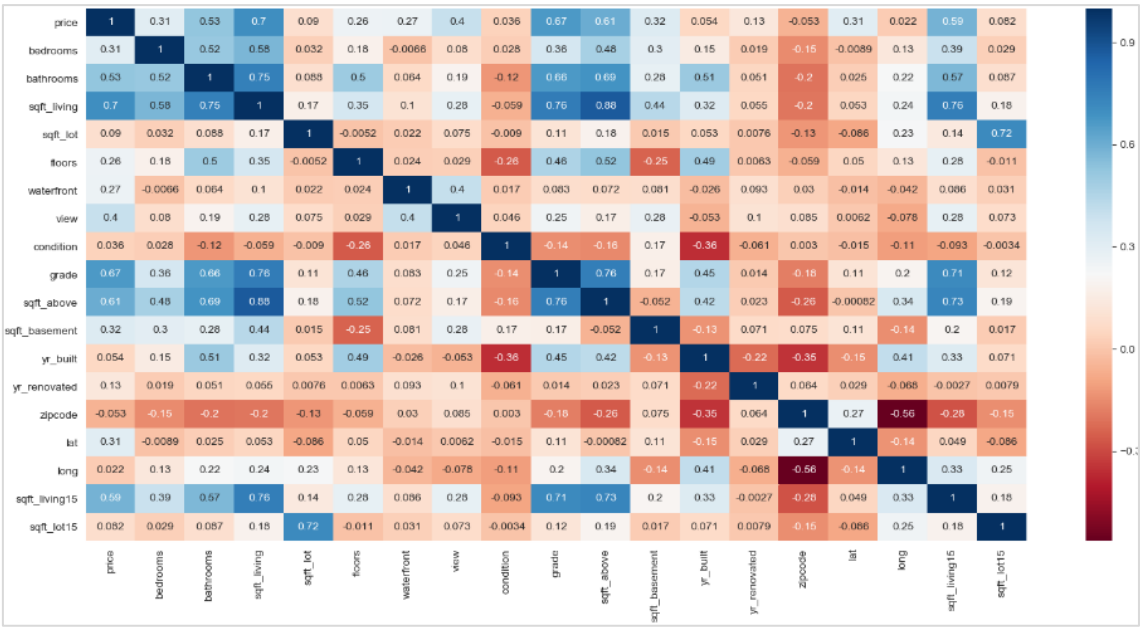


图28-1　属性相关系数分析

上图显示属性自身的相关性为 1，说明任何属性与其本身之间是强相关的，而属性之间相关系数都不大，说明属性之间普遍相关性不高，但是部分属性具备相关性挖掘的可能。

步骤 6　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。

由于相关系数热力图矩阵是对称的，因此只需呈现一部分即可分析属性间的关系。

```
plt.figure(figsize=(16,8))
#配置下三角热力图区域显示模式
mask = np.zeros_like(corr,dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.set_style(style="white")
#对相关系数图进行下三角显示
sns.heatmap(corr,annot=True,cmap="RdBu",mask=mask)
plt.show()
```

输出如下结果：

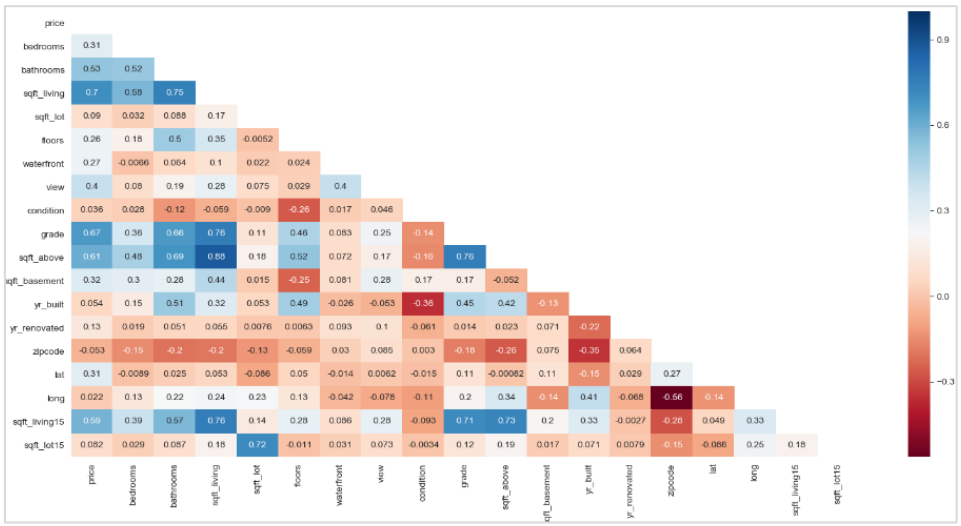


图29-1　属性相关分析下三角关系图

步骤 7　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))
#调用散点图模块，依据经纬度绘制散点图
plt.scatter(df.lat, df.long)
plt.ylabel('Longitude', fontsize=12)
plt.xlabel('Latitude', fontsize=12)
plt.show()
```

输出如下结果：

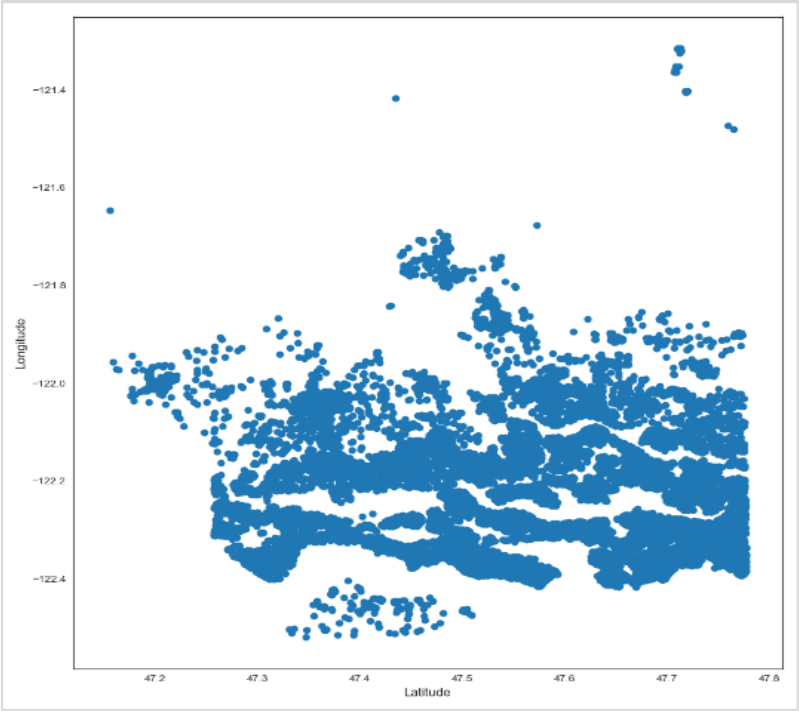


图30-1 房屋经纬度散点图

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。

步骤 8 分析邮编与房价的分布关系

```
#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()
```

输出如下结果：

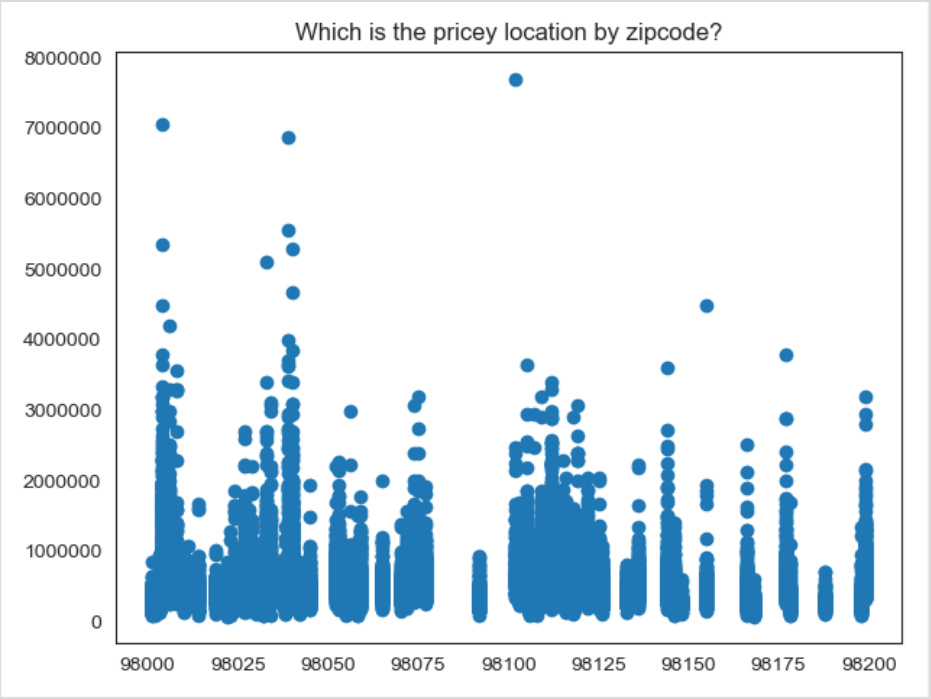


图31-1 区域价格图

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。价格普遍集中在 100~200 万附近，且 100 万左右占大多数，表明价格数据分布存在倾斜的情况，高价格房屋的价格赞比较少。但实际规律无法直观体现，需要采用数据挖掘算法进行建模分析。

3.3.4 模型训练数据处理

由于本案例是依据房屋的属性信息对房屋的价格进行预测，预测的是连续变量，因此这里主要采用回归模型进行预测。在回归模型中最常用的算法有线性回归，随机森林，GBDT，KNN，决策树等模型，这里首先采用线性回归进行数据分析。

步骤 1 配置训练数据与测试数据

```
#使用线性回归模型进行数据分析
from sklearn.linear_model import LinearRegression
#选择用于进行回归分析的属性集，由于数据拆分函数对数据类型有要求，因此此处对属性集进行矩阵转换
#此处由于属性维度较少，使用部分属性可能会造成信息丢失，因此此处采用全量属性进行分析，
#但在实际问题中，若是属性维度较大，一般会依据前面相关系数的结果对属性集进行筛选。
X = df.as_matrix(['bedrooms', 'bathrooms', 'sqft_living',\
        'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',\
        'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',\
```

```
            'lat', 'long', 'sqft_living15', 'sqft_lot15'])
#选择价格作为回归更新的标签值
y = df['price']
#导入数据拆分算法 train_test_split 进行数据集的拆分
from sklearn.model_selection import train_test_split
#将数据拆分为训练数据和测试数据
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
```

步骤 2　数据标准化处理

由于不同属性之间，区间范围差异较大，因此这里对属性特征进行标准化操作。

```
#调用数据标准化模块
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
#对属性数据进行标准化处理
sc.fit(X_train)
#对训练数据属性集进行标准化处理
X_train= sc.transform(X_train)
#对测试数据属性集进行标准化处理
X_test = sc.transform(X_test)
```

3.3.5 数据建模分析

步骤 1　采用线性回归建立回归模型

```
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 2　采用随机森林建立回归模型

```
#导入随机森林回归模型
from sklearn.ensemble import RandomForestRegressor
#配置模型中回归树的个数为 500
model = RandomForestRegressor(n_estimators=500)
#采用随机森林回归模型进行模型训练
model.fit(X_train, y_train)
#采用随机森林回归模型进行预测
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：0.8767796942656374
```

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 3　采用梯度提升树建立回归模型

```
#导入 GBDT 回归模型
from sklearn.ensemble import GradientBoostingRegressor
#配置 GBDT 回归模型的分类器个数
model = GradientBoostingRegressor(n_estimators=500)
#采用训练数据集进行模型训练
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)
#输出模型评估值
print (model.score(X_test, y_test))
```

输出如下模型结果：

```
模型得分值为：0.8862061273914489
```

对比三种模型评估值可知，GBDT 模型评估值相对较好。

步骤 4　采用最近邻算法建立回归模型

```
#导入最近邻回归模型
from sklearn.neighbors import KNeighborsRegressor
#配置最近邻回归模型参数
model = KNeighborsRegressor(n_neighbors=10)
#采用最近邻回归模型进行训练
model.fit(X_train, y_train)
```

```
#采用最近邻模型进行预测
y_pred=model.predict(X_test)
#打印最近邻回归模型评估值
print (model.score(X_test, y_test))
```

输出如下模型结果：

```
模型得分值为：0.7915905936059999
```

上述结果显示，最近邻回归模型预测结果相对较差。

步骤 5 采用梯度提升算法进行回归分析

```
#配置梯度提升树模型参数，树的棵数
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))
```

输出如下结果：

```
模型评估值：0.8863109786504659
平均绝对误差：67612.54474740554
均方根误差：15681513910.104532
```

3.3.5.2 模型评估与优化

步骤 1 采用网格搜索算法进行模型参数优化

```
model_gbr = GradientBoostingRegressor()
#导入网格搜索模块
from sklearn.grid_search import GridSearchCV
#对 loss，min_samples_leaf，alpha 三个参数值进行最优化网格搜索
```

```
parameters = {'loss': ['ls','lad','huber','quantile'],'min_samples_leaf': [1,2,3,4,5],'alpha':
[0.1,0.3,0.6,0.9]}
#调用网格搜索模型进行最优化参数搜索
model_gs = GridSearchCV(estimator=model_gbr, param_grid=parameters, cv=5)
model_gs.fit(X_train,y_train)
#输出最优的模型评估值和模型参数值
print('Best score is:', model_gs.best_score_)
print('Best parameter is:', model_gs.best_params_)
```

输出如下结果：

```
Best score is: 0.8658896218983708
Best parameter is: {'alpha': 0.3, 'loss': 'ls', 'min_samples_leaf': 2}
```

上述结果表明，最优的 alpha 为 0.3，loss 为 ls，min_samples_leaf 为 2。

步骤 2 采用最优参数进行数据建模分析

```
from sklearn.ensemble import GradientBoostingRegressor
#配置最优模型参数的模型
model = GradientBoostingRegressor(n_estimators=500,alpha=0.3,loss='ls',min_samples_leaf=2)
#调用最优模型参数进行训练
model.fit(X_train, y_train)
#使用最优模型进行模型预测
y_pred=model.predict(X_test)
#计算平均绝对误差，均方根误差，r2 模型值
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
#输出计算平均绝对误差，均方根误差，r2 模型值
print(model.score(X_test, y_test))
print(mean_absolute_error(y_test,y_pred))
print(mean_squared_error(y_test,y_pred))
```

输出如下结果：

```
模型评估值：0.8877197229776028
平均绝对误差：67634.09309992738
均方根误差：15487201007.244192
```

从模型评估值来看，参数优化后的模型相比之前的模型评估值更优，虽然平均绝对误差相对大些，但是均方根误差有很明显的改善，比之前相比下降了 1.3 个百分点，同时也为后续其他模型参数的优化提供了方式方法，相关经验可以为后续模型参数优化提供借鉴！

3.4 思考题

1、有监督学习中常见的评估指标有哪些?

2、在线性回归和决策树类的算法模型中，如何防止模型过拟合?

3、在建模过程中，通过需要进行特征选择，可以通过哪些方式实现特征选择?

4、与随机森林算法相比，为什么 GBDT 算法通过更加容易出现过拟合？在实际应用中可以通过控制哪些参数，对 GBDT 模型进行参数优化，防止过拟合?

【答案】

3.5 实验小结

本实验主要介绍了如何针对实际问题进行建模分析，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。在模型参数优化方面，采用交叉网格搜索的方式进行最优化模型参数的搜索，从而找到最优化模型参数，有效提升了模型的准确率，在原有基础上降低了模型的均方根误差，为后续相关参数的优化提供了技术准备。实验结果表明采用该流程能够对此类回归问题进行预测分析，效果比较好，可以作为解决此类数据分析挖掘算法的一种有效手段。当然模型准确率还有提升的空间，学习者可以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，获取更好的回归模型。

3.6 思考题-汇总

注：教师版需要给出参考答案，代码挖空题除外；学生版：需留出空白方框。

1、有监督算法主要分为两类，一类是分类， 一类是回归。回归评估方法主要有：MAE，MSE；分类问题的评估指标主要包括：准确率，召回率，精度，F1 值，ROC 曲线和 AUC 曲线等。

2、在回归模型中，可以通过添加 L1 正则项或 L2 正则项的方式，对模型参数进行惩罚，从而防止过拟合，在决策树类算法模型中，可以通过先剪枝和后剪枝的方式防止模型过拟合。

3、针对特征选择，可以从统计角度和算法应用角度进行特征重要性分析，统计的方法包括：方差，相关系数和卡方检验的方式，算法应用方面可以使用 L1 正则、决策树模型及递归特征消除法等对特征重要性进行分析和特征筛选，从而特征选择。

4、GBDT 模型属于加法模型，各学习器之间存在强依赖关系，且后面的学习器拟合的是前面学习器的残差，因此随着学习器个数的增加，模型更容易过拟合。可以通过调整 Max_depth,Min_samples_leaf,n_estimators 等参数实现优化，防止过拟合。（答案不唯一）

4 犯罪行为预测

4.1 实验说明

数据记录了发生在芝加哥的犯罪事件，该数据反应反映了 2001-2017 年发生在芝加哥的犯罪事件（除了每个受害者的谋杀数据）。数据来自芝加哥警察局的 CLEAR(公民执法分析和报告)系统。通过对犯罪历史数据的分析，找到犯罪的规律，有利于各区的警力部署，并通过一些特征对犯罪类型进行预测。通过数据挖掘相关算法来预测犯罪行为，该问题本质上是一个多分类问题。

数据详细情况可以查看：https://www.kaggle.com/currie32/crimes-in-chicago/kernels

4.2 实验建模流程要求

4.2.1 环境要求

Python 3.7

4.2.2 实验实现步骤要求

步骤 1　相关模块导入

要求导入相关数据读取、处理、分析、可视化，算法模块等

步骤 2　设置数据存放目录：data\Chicago_Crimes\

```
import os
import pandas as pd
os.chdir(r'data\Chicago_Crimes\ ')
file_chdir = os.getcwd()
```

步骤 3　创建 list 存放 csv 文件路径

```
#创建 list 存放 csv 文件路径
filecsv_list = []
```

```
#获取工作目录里面所有 csv 文件路径
for root, dirs, files in os.walk(file_chdir):
    for file in files :
        if os.path.splitext(file)[1] == '.csv':
            filecsv_list.append(file)
```

步骤 4 循环读取指定路径下的全部 csv 文件（4 个）数据，通过函数 append 全部整理到一个 df 数据框中，#装下全部的原始数据的数据框，命名为 df_raw。

```
df_raw = pd.DataFrame()
for csv in filecsv_list:
    df_raw = df_raw.append(pd.read_csv(csv,sep =',',error_bad_lines=False))
del df_raw['Unnamed: 0']          #删除原始数据中无意义的索引列
```

步骤 5 查看数据尺寸

步骤 6 数据规模较大，进行随机抽样，选取 1%的数据，并将抽样后的数据命名为 df，数据框形式，作为后续分析处理建模的数据集

提示：DataFrame.sample(n,frac,replace=False) n 按个数抽样 frac 按百分比抽样 replace 是否放回抽样，默认 false 为不放回

步骤 7 删除 df 中，列名为 IUCR，FBI Code ，Case Number，Updated On，ID，Arrest，Domestic 它们是一种主要类型本身的编码或者对分析结果没有意义，并且进行删除所有缺失值所在的行。

步骤 8 时间字段处理：将字段 Date 从 object 转为 datetime 型，再扩展字段，提取年，月，周，日，小时信息。转换后删除原先的 Date 字段。完成如下代码下划线部分

```
#导入 numpy，将字段 object 类型转为 datetime 类型
import numpy as np
df['Date'] = df['Date'].astype(np.datetime64)
#扩展字段，提取月，周，日，小时信息
df['week']=_____          ##提取星期，转换为 datetime 类型
df['month']=_____         ##提取月份，转换为 datetime 类型
df['day']= _____          ##提取日，转换为 datetime 类型
df['hour']=_____          ##提取小时，转换为 datetime 类型
#删除'Date'
del df['Date']
```

步骤 9 用 factorize 函数实现字符串类型字段数值化

步骤 10 采用 MinMaxScaler 对 X Coordinate、Y Coordinate 进行规范化，部分提示如下：

```
from sklearn.preprocessing import MinMaxScaler
df['X Coordinate'] = df['X Coordinate'].astype(float)
df['Y Coordinate'] = df['Y Coordinate'].astype(float)
```

步骤 11 导入相关 seaborn 和 matplotlib 绘图包，对目标变量各个分类标签进行汇总查看，查看各个犯罪类型的总数排名，将犯罪数量最多的类型排前。

```
#导入相关 seaborn 和 matplotlib 绘图包
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
xy = df.groupby([df['Primary Type']]).size()
xy = xy.reset_index()
xy = xy.sort_values(by = 0,ascending = False)#by 表示按照哪个字段进行排序，ascending 表示排序方法，False 表示升序
sns.set(style="darkgrid", context="notebook",font_scale=0.7,font='Yahei')
ax = sns.barplot(x = xy[0],y = xy['Primary Type'])
ax.set_title('Frequency of Crimes')
ax.set_xlabel('Amount of Crimes')
ax.set_ylabel('Primary Type')
```

步骤 12 用 factorize 函数将 Primary_Type 数值化，数值化的特征命名为新的名称：Primary Type，并将原先的目标变量从 df 中删除

```
制作剩余特征的相关性图
plt.figure(figsize = (10,6))
plt.rc('xtick', labelsize = 10)
plt.rc('ytick', labelsize = 10)
#计算相关系数并画热点图
cor = df.corr()#默认为 method='pearson',为 pearson 相关系数
sns.heatmap(cor, annot = True,cmap = 'YlGnBu')#annot 为是否显示相关系数值，cmap 颜色列表，可选从数据值到颜色空间的映射。
plt.show()
```

步骤 13 将数据集 df 进行拆分为训练集和测试集，拆分比例为 0.2，设置随机种子

步骤 14 选择 GBDT 算法建模，对训练集进行训练，对测试集进行预测

步骤 15 选择多个评估指标 accuracy_score、precision_score 、recall_score 、f1_score，对预测结果进行评估，并打印出在各个指标上得分。

部分提示如下：

```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
```

步骤 16 采用 GridSearchCV 方法调整参数。由于调整参数越多模型运行的时间越长，实验先对学习速率参数调优，选取[0.01,0.1,0.3]共 3 个参数，并采用 5 折交叉验证法（cv=5）进行模型评估。

步骤 17 选取决策树最大深度 max_depth 调参，选取范围[3,5,10]

步骤 18 根据两次参数调整后的模型重新对测试集进行预测，选择多个评估指标 accuracy_score、precision_score 、recall_score 、f1_score，对预测结果进行评估，并打印出在各个指标上得分

4.3 参考答案

参考：犯罪行为预测.ipynb

5 广告点击率预测

5.1 实验说明

CTR（Click-Through-Rate）即点击通过率，是互联网广告常用的术语，指网络广告（图片广告/文字广告/关键词广告/排名广告/视频广告等）的点击到达率，即该广告的点击量（严格的来说，可以是到达目标页面的数量）除以广告的浏览量（PV- Page View）。当前在线广告服务中，网站上广告的点击率（CTR）是评估广告效果的一个非常重要的指标，也是计算广告领域的核心问题，提升 CTR 预估的准确率是提升精准营销效果的核心技术点。因此，点击率预测系统是必不可少的，并广泛用于赞助搜索和实时竞价，是广告的主要收入来源之一。

本实验主要是做网站广告的点击率预测，希望学习者通过本实验的实践操作之后，借鉴精准广告推送的思想，对用户基本属性数据，广告详情数据，以及广告类别数据等数据集进行数据预处理，构建训练广告推荐数据集，并结合相关的数据分析算法，建立分析模型，从而实现精准广告推送的目的。

数据来源于 2019 年，华为举办的算法设计大赛。

5.2 实验建模流程要求

基于网站点击率数据，使用逻辑回归和 Xgboost 算法进行预测预测。实验基本流程主要包括如下几步：（1）数据读取；（2）数据探索；（3）特征工程；（4）匿名属性分析；（5）设备属性分析；（6）模型预测；（7）模型对比等

5.2.1 环境要求

Python 3.7

5.2.2 实验实现步骤要求

5.2.2.1 相关模块导入

步骤 1 要求导入相关数据读取、处理、分析、可视化，算法模块等

执行如下命令加载实验所需要的库，由于本实验需要使用 pandas 进行数据处理，同时还需要使用到迭代器和可视化工具，以及建模工具，所以这里加载了 pandas，numpy，itertools，matplotlib，seaborn，missingno，os，xgboost 和 sklearn 等库。

5.2.2.2 加载数据

步骤 1 设置数据访问文件路径，由于上述数据文件主要存放于同一文件夹下，因此这里提前指定数据文件夹路径，方便后续进行完整路径拼接和数据读取。

步骤 2 加载广告信息数据，设置数据列名

广告任务特征文件 ad_info.csv，每行代表一个广告任务的特征数据，格式为："adId, billId, primId, creativeType, intertype, spreadAppId"。

步骤 3 加载素材信息数据，设置数据列名

素材信息数据文件 content_info.csv，数据格式为："contentId, firstClass, secondClass"。

步骤 4 加载用户特征数据，设置数据列名

用户特征文件 user_Info.csv 每行代表一个用户的基础特征数据，格式为："uId, age, gender, city, province, phoneType, carrier"。

步骤 5 加载模型训练数据集，设置数据列名

训练数据文件 train.csv 每行代表一个训练样本，各字段之间由逗号分隔，格式为："label, uId, adId, operTime, siteId, slotId, contentId, netType"。其中，adId 唯一标识一个广告，uId 唯一标识一个用户。样本 label 的取值为 1 或 0，其中 1 表示点击，0 表示未点击。

5.2.2.3 信息完善

由于本案例考察的是指定用户在特定的时间点对于特定类型的广告是否会点击，因此这里需要依据训练数据对相关的属性信息进行拼接。

步骤 1 用户信息拼接

```
#执行如下代码，依据 uId 用户 Id 字段完成对用户信息的拼接。
traindata=pd.merge(traindata,User_info,on='uId')
traindata.head()
```

步骤 2 广告信息拼接

```
#接下来依据 contentId 和 adId，实现对广告信息的拼接。
```

5.2.2.4 数据理解

步骤 1 数据属性分析

主要是查看数据的记录数据、缺失情况、数据类型等相关信息。

步骤 2 firstClass 属性字段信息汇总

主要是查看，firstClass 离散属性各值的取值个数。

步骤 3 数据属性过滤

对于训练数据集中，包含 spreadAppId 和 secondClass 字段，这两个字段分别表示的是广告推广的方式和广告的第二类别信息，但是在实际数据集中却缺乏 APP 相关的数据描述和第二类别信息的描述，因此在实际数据处理的时候，考虑将这两个字段去除。通过代码，实现上述功能。

5.2.2.5 数据编码

从前面的数据理解中，可以看到，上述属性中包含部分对象型属性字段，而在实际数据分析中，对象型属性是无法直接参与计算的，因此这里需要对有价值的属性字段进行编码操作。对于某些离散无序属性当取值较少时可采用 One-hot 进行数据编码，如 billId，firstClass。

步骤 1 对 billId 进行 One-hot 编码

```
n_columns=["billId"]
dummy_df_01 = pd.get_dummies(traindata[n_columns])
```

步骤 2 billId 编码属性拼接及原属性剔除

```
traindata = pd.concat([traindata, dummy_df_01], axis=1) #当 axis = 1 的时候，concat 就是行对齐，然后将不同列名称的两张表合并
traindata = traindata.drop(n_columns, axis=1)
```

步骤 3 对 firstClass 字段进行 One-hot 编码，包括编码，拼接和剔除

步骤 4 平均数编码

如果某一个特征是定性的（categorical），而这个特征的可能值非常多（高基数），那么平均数编码（mean encoding）是一种高效的编码方式。在实际应用中，这类特征工程能极大

提升模型的性能。 在上述属性中，phoneType 表示不同人的手机类型，取值较多，且大多是离散值，为后方便计算，使用均值编码的方式进行数据预处理。均值编码代码实现如下：

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from itertools import product
class MeanEncoder:
    def __init__(self, categorical_features, n_splits=5, target_type='classification',
prior_weight_func=None):
        """
        :param categorical_features: list of str, the name of the categorical columns to encode

        :param n_splits: the number of splits used in mean encoding

        :param target_type: str, 'regression' or 'classification'

        :param prior_weight_func:
        a function that takes in the number of observations, and outputs prior weight
        when a dict is passed, the default exponential decay function will be used:
        k: the number of observations needed for the posterior to be weighted equally as the
prior
        f: larger f --> smaller slope
        """

        self.categorical_features = categorical_features
        self.n_splits = n_splits
        self.learned_stats = {}

        if target_type == 'classification':
            self.target_type = target_type
            self.target_values = []
        else:
            self.target_type = 'regression'
            self.target_values = None

        if isinstance(prior_weight_func, dict):
            self.prior_weight_func = eval('lambda x: 1 / (1 + np.exp((x - k) / f))',
dict(prior_weight_func, np=np))
        elif callable(prior_weight_func):
            self.prior_weight_func = prior_weight_func
        else:
            self.prior_weight_func = lambda x: 1 / (1 + np.exp((x - 2) / 1))

    @staticmethod
    def mean_encode_subroutine(X_train, y_train, X_test, variable, target, prior_weight_func):
        X_train = X_train[[variable]].copy()
```

```
            X_test = X_test[[variable]].copy()

            if target is not None:
                nf_name = '{}_pred_{}'.format(variable, target)
                X_train['pred_temp'] = (y_train == target).astype(int)    # classification
            else:
                nf_name = '{}_pred'.format(variable)
                X_train['pred_temp'] = y_train    # regression
            prior = X_train['pred_temp'].mean()

            col_avg_y = X_train.groupby(by=variable, axis=0)['pred_temp'].agg({'mean': 'mean',
'beta': 'size'})
            col_avg_y['beta'] = prior_weight_func(col_avg_y['beta'])
            col_avg_y[nf_name] = col_avg_y['beta'] * prior + (1 - col_avg_y['beta']) *
col_avg_y['mean']
            col_avg_y.drop(['beta', 'mean'], axis=1, inplace=True)

            nf_train = X_train.join(col_avg_y, on=variable)[nf_name].values
            nf_test = X_test.join(col_avg_y, on=variable).fillna(prior, inplace=False)[nf_name].values

            return nf_train, nf_test, prior, col_avg_y

        def fit_transform(self, X, y):
            """
            :param X: pandas DataFrame, n_samples * n_features
            :param y: pandas Series or numpy array, n_samples
            :return X_new: the transformed pandas DataFrame containing mean-encoded
categorical features
            """
            X_new = X.copy()
            if self.target_type == 'classification':
                skf = StratifiedKFold(self.n_splits)
            else:
                skf = KFold(self.n_splits)

            if self.target_type == 'classification':
                self.target_values = sorted(set(y))
                self.learned_stats = {'{}_pred_{}'.format(variable, target): [] for variable, target in
                                      product(self.categorical_features, self.target_values)}
                for variable, target in product(self.categorical_features, self.target_values):
                    nf_name = '{}_pred_{}'.format(variable, target)
                    X_new.loc[:, nf_name] = np.nan
                    for large_ind, small_ind in skf.split(y, y):
                        nf_large, nf_small, prior, col_avg_y =
MeanEncoder.mean_encode_subroutine(
```

```
                                X_new.iloc[large_ind], y.iloc[large_ind], X_new.iloc[small_ind],
variable, target, self.prior_weight_func)
                        X_new.iloc[small_ind, -1] = nf_small
                        self.learned_stats[nf_name].append((prior, col_avg_y))
            else:
                self.learned_stats = {'{}_pred'.format(variable): [] for variable in
self.categorical_features}
                for variable in self.categorical_features:
                    nf_name = '{}_pred'.format(variable)
                    X_new.loc[:, nf_name] = np.nan
                    for large_ind, small_ind in skf.split(y, y):
                        nf_large, nf_small, prior, col_avg_y =
MeanEncoder.mean_encode_subroutine(
                            X_new.iloc[large_ind], y.iloc[large_ind], X_new.iloc[small_ind],
variable, None, self.prior_weight_func)
                        X_new.iloc[small_ind, -1] = nf_small
                        self.learned_stats[nf_name].append((prior, col_avg_y))
            return X_new

        def transform(self, X):
            """
            :param X: pandas DataFrame, n_samples * n_features
            :return X_new: the transformed pandas DataFrame containing mean-encoded
categorical features
            """
            X_new = X.copy()
            print(self.target_values)

            if self.target_type == 'classification':
                for variable, target in product(self.categorical_features, self.target_values):
                    nf_name = '{}_pred_{}'.format(variable, target)
                    X_new[nf_name] = 0
                    for prior, col_avg_y in self.learned_stats[nf_name]:
                        X_new[nf_name] += X_new[[variable]].join(col_avg_y,
on=variable).fillna(prior, inplace=False)[nf_name]
                    X_new[nf_name] /= self.n_splits
            else:
                for variable in self.categorical_features:
                    nf_name = '{}_pred'.format(variable)
                    X_new[nf_name] = 0
                    for prior, col_avg_y in self.learned_stats[nf_name]:
                        X_new[nf_name] += X_new[[variable]].join(col_avg_y,
on=variable).fillna(prior, inplace=False)[
                            nf_name]
                    X_new[nf_name] /= self.n_splits
```

```
        return X_new
```

步骤 5 phoneType 的均值编码、属性拼接和原属性删除。

```
#均值编码
objectColumns1=["phoneType"]
me=MeanEncoder(objectColumns1)#对 phoneType 进行平均值编码
label_y=traindata["label"]
me_matrix=me.fit_transform(traindata[objectColumns1],label_y)
# phoneType 均值编码属性拼接
traindata=pd.concat([traindata,me_matrix],axis=1) #完成拼接
drop_list=["phoneType"]
traindata=traindata.drop(drop_list, axis=1)#剔除原始属性
```

5.2.2.6 时间字段细分

对于 CTR 广告点击率预测的实质是在合适的时间、合适的地点、把合适广告推荐给合适的人，因此在实践层面需要考虑月、日、周、时等基本信息，因此需要对时间进行细粒度的拆分。通过如下代码实现上述操作。

5.2.2.7 无效字段过滤

由于前面已经对实践字段进行了处理，因此这里对无效的字段进行删除，包括"operTime","date","adId","carrier"。（由于没有广告大多是互联网广告，不用区分运营商类型，属性集中已经包括了网络类型，因此这里不需要添加运营商类型）。

5.2.2.8 缺失值处理

步骤 1 缺失值检测

在数据挖掘中，由于大部分模型都无法容忍属性字段缺失，因此缺失值检测是最重要的一环，执行如下代码进行缺失值检测。

步骤 2 查看缺失字段信息

步骤 3 缺失值填充

5.2.3 数学建模

5.2.3.1 数据拆分

步骤 1 定义数据拆分函数

定义数据拆分函数将样本数据拆分成训练数据和测试数据,30%用于测试，70%用于训练。

步骤 2 数据拆分

5.2.3.2 数据建模分析

步骤 1 使用随机森林算法建模分析

```
data_train_X_clms=data_train_X.columns
objectColumns1=list(data_train_X_clms.drop(["uId"]))
#调用回归森林算法进行模型训练
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=55,random_state=123)
clf.fit(data_train_X[objectColumns1], data_train_y)
preds = clf.predict(X= data_test_X[objectColumns1])
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
print("Test set accuracy score: {:.5f}".format(accuracy_score(data_test_y, preds)))
print(classification_report(data_test_y, preds))
roc_auc = roc_auc_score(data_test_y, preds)
print("Area under the ROC curve : %f" % roc_auc)
```

步骤 2 使用网格搜索算法对随机森林算法进行参数优化

```
param_test1 = {'n_estimators':[55,120,160,200,250]}
from sklearn.model_selection import GridSearchCV
#param_test2 = {'max_depth':[1,2,3,5,7,9,11,13]} #,
'min_samples_split':[100,120,150,180,200,300]}
gsearch1 = GridSearchCV(estimator = RandomForestClassifier(),param_grid = param_test1)
gsearch1.fit(data_train_X[objectColumns1], data_train_y)
print( gsearch1.best_params_, gsearch1.best_score_)
preds=gsearch1.predict(data_test_X[objectColumns1])
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
print("Test set accuracy score: {:.5f}".format(accuracy_score(data_test_y, preds)))
print(classification_report(data_test_y, preds))
roc_auc = roc_auc_score(data_test_y, preds)
print("Area under the ROC curve : %f" % roc_auc)
```

步骤 3 使用 LogisticRegression 分类算法建模分析

步骤 4 使用 GBDT 分类算法建模分析

步骤 5 使用 Xgboost 分类算法建模分析

5.3 参考答案

参考：广告点击率预测.ipynb

6 订购定期存款

6.1 实验说明

存款业务是银行的主要业务之一，其中定期存款能够为银行带来较好的资金储备。本案例主要是采用来自加州大学欧文机器学习库的银行业务数据，包含与葡萄牙银行机构的直接营销活动相关的信息，通过决策树类算法实现客户订购定期存款的预测。

数据集为：data\bank-full.csv

输入：数据集。

元数据中 Target 属性列仅支持一列，其数据类型支持 Integer、String、Real 三种，其 Measurement level 支持 Nominal、Flag、Ordinal 三种；Input 属性列支持 String、Integer、Real 三种数据类型，Measurement level 支持 Nominal、Flag、Ordinal 和 Continuous 四种。

输出：决策树分类模型。

输出：随机决策森林分类模型。

6.2 根据提供的数据集，设计一个完整的分类分析实验

6.2.1 要求如下：

- 申请 ModelArts 的 notebook 实验环境申请，对接 OBS 数据源；
- 实现 OBS 数据加载与数据理解；
- 使用 pandas 工具实现离散属性和标签字段的转换；
- 使用 One-hot 编码，实现 job、marital、education、contact、month、poutcome、day 等属性字段的编码；
- 对数据进行洞察，实现数据集的相关性分析和数据集拆分（训练集和测试集）；
- 随机森林算法实现业务建模，输出最佳模型及测试集预测效果。

6.3 参考答案

参考：订购定期存款.ipynb

7 信用卡欺诈预测

7.1 实验说明

信用卡公司必须能够识别欺诈性的信用卡交易，以免向客户收取未购买商品的费用。数据集包含 2013 年 9 月欧洲持卡人通过信用卡进行的交易。

该数据集显示了两天内发生的交易，在 284,807 笔交易中，有 492 起欺诈。数据集高度不平衡，正样本（欺诈）占所有交易的 0.172%。用信用卡的历史交易数据，进行机器学习，构建信用卡反欺诈预测模型，提前发现客户信用卡被盗刷的事件。

本地离线数据地址和名称：data\creditcard.csv

数据集属性说明参考网址：

https://www.kaggle.com/mlg-ulb/creditcardfraud?select=creditcard.csv

7.2 根据提供的数据集，设计一个完整的分类预测实验

数据集为二分类数据，存在严重的不平衡情况，选择随机森林算法进行分类，考虑到样本极度不均衡，模型评价选用 classification_report 和 roauc。

要求如下：

- 数据读取与探索
- 去掉均匀分布且不是数值属性的 Time 列
- 特征工程：PC 降维，保留 20 个主成分
- 划分 训练集 & 测试集，分割比例为 0.3

- 模型选择 Random Forest ，指定部分重要参数（至少 3 个），对数据进行训练和预测，保证模型在测试集上的 roauc 得分大于 85%

7.3 参考答案

参考：信用卡欺诈预测.ipynb

8 个人收入预测

8.1 实验说明

这个案例中，主要使用美国人口普查收入数据集，根据人口普查数据预测个人收入是否超过每年 50,000 美元。这本质上是一个二分类问题。

本地离线数据地址和名称：data\dataset-credit-default.csv

提取由 Barry Becker 从 1994 年人口普查数据库中进行的。根据人口普查数据预测收入是否超过$ 50K /年。也称为“普查收入”数据集。

数据集属性说明：

Target：> 50K，<= 50K。

年龄：连续。

工作类别：私人，自营非收入，自营收入，联邦政府，地方政府，州政府，无薪，从未工作过。

fnlwgt：连续。

教育程度：学士，部分大学，11 年级，高中毕业生，教授学校，Assoc-acdm，Assoc-voc，9、7-8-8、12，硕士，1-4 至 10，博士学位，5-6 至学前班。

教育人数：连续。

婚姻状况：已婚公民配偶，离婚，未婚，分居，丧偶，已婚配偶缺席，已婚 AF 配偶。

职业：技术支持，工艺维修，其他服务，销售，执行管理，专业教授，装卸清洁员，机器操作检查，行政助理，农家捕鱼，运输移动，私人住宅 serv，防护 serv，武装部队。

关系：妻子，独生子女，丈夫，亲戚，其他亲戚，未婚。

种族：白色，亚洲人-帕斯岛，亚美-印度-爱斯基摩人，其他，黑人。

性别：女，男。

资本收益：连续。

资本损失：连续。

每周小时：连续。

祖国：美国，柬埔寨，英国，波多黎各，加拿大，德国，美国外围地区（关岛-USVI 等），印度，日本，希腊，南美，中国，古巴，伊朗，洪都拉斯，菲律宾，意大利，波兰，牙买加，越南，墨西哥，葡萄牙，爱尔兰，法国，多米尼加共和国，老挝，厄瓜多尔，台湾，海地，哥伦比亚，匈牙利，危地马拉，尼加拉瓜，苏格兰，泰国，南斯拉夫，萨尔瓦多，特立尼达和多巴哥，秘鲁，洪，荷兰霍兰。

8.2 根据提供的数据集，设计一个完整的分类预测实验

8.2.1 要求如下：

- 数据读取与合并
- “dataset”数据分为训练和测试数据集。将两个进行联合数据分析，然后在运行算法之前再将它们分开。
- 特征预处理：根据特征类型：数字型，可计算；标量型对各个特征进行描述性分析
- 特征预处理：属性编码
- 特征工程：可视化查看各个特征的关系热力图
- 特征工程：使用随机森林算法去查看属性重要性
- 划分 训练集 & 测试集
- 从如下算法中选择 4 个：KNN，Logistic Regression，Random Forest，Naive Bayes，Stochastic Gradient，Decent，Linear SVC，Decision Tree，Gradient Boosted Trees，训练模型，并且预测结果
- 从上述模型选择一个表现比较突出的进行参数调优，至少调节两个参数，且进行交叉验证，选择 ROC 指标进行评估

8.3 参考答案

参考：个人收入预测.ipynb