# Python 快速入门

## 变量和简单数据类型

python声明变量时,不用指明数据类型,解释器会根据实际数据类型自动推导

### 变量

name = "John"

number = 4

print(type(name))

print(type(number))

print(type(number2))

---------------------------------------

# 结果

<class 'str'>

<class 'int'>

<class 'float'>

### 字符串

first\_name = "ada"

last\_name = "dddd"

full\_name = f"{first\_name} {last\_name}"

message = f"Hello,{full\_name.title()}!"

print(message)

f：是format 简写，通过花括号内的变量替换为其值来设置字符串的格式

title() ：将字符串首字母大写

rstrip() ：剔除字符串右边的空白

lstrip() ：剔除字符串左边的空白

strip() : 剔除字符串两边的空白

name = " python"

print(name)

print(name.strip())

-------------------------------------

# 结果

python

python

### 数

乘方

# 计算4的三次方

print(4\*\*3)

------------------------

# 结果

64

使用\_对数字进行分组，使其更加清晰

number3 = 10\_000\_000

print(number3)

-------------------------

# 结果

10000000

## 列表

dogs = ["a", "d", "e", "f"]

print(dogs)

dogs.insert(1, "g") # 在列表下标为1增加一个元素

del dogs[0] # 删除列表中小标为0

dogs[2] = "m" # 修改

dogs.append("w") # 末尾追加

print(dogs)

x = dogs.pop() # 删除末尾元素并返回 类似于栈

print(x)

print(len(dogs)) # 求长度

dogs.remove("d") # 根据值删除

print(dogs)

dogs.reverse() # 将列表反转

print(dogs)

-------------------------------------------

# 结果

['a', 'd', 'e', 'f']

['g', 'd', 'm', 'f', 'w']

w

4

['g', 'm', 'f']

['f', 'm', 'g']

Process finished with exit code 0

切片：处理列表的一部分元素

players = ["ok", "sw", "dw", "oo", "fs"]

print(players[1:3]) # 返回索引 1-2的元素

print(players[1:]) # 返回索引 1-列表末尾的元素

'''

两边都为空表示 返回从头到尾的元素

'''

gamer = players[:]

print(gamer)

--------------------------

['sw', 'dw']

['sw', 'dw', 'oo', 'fs']

['ok', 'sw', 'dw', 'oo', 'fs']

## if和循环

elif 就相当于 else if

b = eval(input("请输入：")) # 类型转换成 输入的类型 自动转换

if b >= 18:

print("hello")

elif b >= 13:

print("Yes")

elif b >= 8:

print("happy")

else:

print("No")

for i in range(b): # [0,b)

print(i)

for i in range(10, 30): # [10,30)

print(i)

for i in range(10, 30, 2): # [10,30) 步长为2

print(i)

while i <= 5:

print(i)

i+=1

numbers = list(range(5, 10)) # 转换成列表输出

print(numbers)

-------

[5, 6, 7, 8, 9]

numbers = range(1, 20)

print(min(numbers)) # 求最小值

print(sum(numbers)) # 求最和

print(min(numbers)) # 求最小值

-----------------------

1

190

19

## 元组

列表适合用于存储在程序运行期间可能引起变化的数据集，列表是可以修改

不可变的列表称为元组.

dimensions = (20, 60, 80)

print(dimensions[0])

print(dimensions)

--------------------------------

20

(20, 60, 80)

## 字典

字典，使用键值对存储数据

cats = {"name": "john", "age": 20}

print(cats)

a = cats.get("name", "john")

print(cats.values())

for key, value in cats.items():

print(f"{key}")

print(f"{value}")

-------------------------------

{'name': 'john', 'age': 20}

dict\_values(['john', 20])

{'name': 'john', 'age': 20}

dict\_values(['john', 20])

name

john

age

20

"""

items：返回所有键值对列表

keys:返回所有键

valuse：返回所有值

"""

字典也可以存储列表

users = {"manage": {

"name": "Tom",

"age": 30,

"location": "A",

},

"student": {

"name":"mae",

"age":20,

"location":"S",

},

}

-----------------------------------

{'manage': {'name': 'Tom', 'age': 30, 'location': 'A'}, 'student': {'name': 'mae', 'age': 20, 'location': 'S'}}

## 函数

定义：

def 函数名(形式参数)

def amb(a, b):

return (a + b) \*\* 2

print(amb(1, 3))

# 可以使用默认值

def amb(a, b=3):

return (a + b) \*\* 2

print(amb(1))

-------------------------

16

16

# 如何传任意个数的实参 args 相当于一个元组 与java相同

def js(\*args):

print(args)

js("a", "b", "c")

js("e", "f")

('a', 'b', 'c')

('e', 'f')

将函数存储在模板中

# test07

def amb(a, b=3):

return (a + b) \*\* 2

---------------------

"""

improt test07 as t:导入整个模板 as可以起别名 非必选

t.amb(2, 3)

"""

from test07 import amb #导入特定的函数

print(amb(1, 2))

## 类

init()：为构造函数

形参self必不可少

每个方法定义都有self。以self为前缀的变量可供类中所有的方法使用

# test09

class Dog:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def bite(self):

print("wangwang")

---------------------------------

import test09 as t

my\_dog = t.Dog('Yel', 5)

my\_dog.bite()

---------------------------------

# 结果：

wangwang

继承：

在继承类定义在类名后加括号写上继承的类

class Dog:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def bite(self):

print("wangwang")

class BlueDog(Dog):

def \_\_init\_\_(self, name, age, log):

super().\_\_init\_\_(name, age)

self.log = log

def describe(self):

print(f"{self.name} {self.age} {self.log}")

---------------------------------------------

Wa 20 mm

## 文件读写

# 写入文件

with open("username", 'w') as file\_object:

file\_object.write("I Love Python\n")

file\_object.write("I also love Java\n")

-----------------------------

I Love Python

I also love Java

"""

第二个参数

w：表示写，会覆盖原有内容

a: 表示在原有文件后追加内容

"""

# 读取文件

with open("username") as file\_object2:

contents = file\_object2.read()

print(contents)

"""

readlines():表示读取一行

"""

with open("username") as file\_object2:

lines = file\_object2.readlines()

p\_string = ''

for line in lines:

p\_string += line.rstrip()

print(p\_string)

--------------------------------------

I Love PythonI also love Java

## 异常

### ZeroDivisionError 异常

try:

print(5 / 0)

except ZeroDivisionError:

print("not divide by zero!"

-----------------------------------

not divide by zero!

### FileNotFoundError 异常

try:

with open("username", 'w') as file\_object:

file\_object.write("I Love Python\n")

file\_object.write("I also love Java\n")

except FileNotFoundError:

print("File not exist")

### else 代码块

a = int(input("input a number: "))

try:

b = 5 / a

except ZeroDivisionError:

print("not divide by zero!")

else:

print(b)

-----------------------------

input a number: 0

not divide by zero!

input a number: 1

5.0

## 测试代码

# test09

class Dog:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def bite(self):

print("wangwang")

class BlueDog(Dog):

def \_\_init\_\_(self, name, age, log):

super().\_\_init\_\_(name, age)

self.log = log

def describe(self):

print(f"{self.name} {self.age} {self.log}")

------------------------------------------------

import unittest

import test09 as t

class DogTestCase(unittest.TestCase):

def test\_describe(self):

my\_dog = t.BlueDog("john", 18, "hahaha")

my\_dog.describe()

# 运行结果：

Testing started at 22:10 ...

Ran 1 test in 0.003s

OK

Launching unittests with arguments python -m unittest test12.DogTestCase.test\_describe in D:\pythonProject\Exer

Process finished with exit code 0

john 18 hahaha

## 思考题

1.有四个数字：1、2、3、4，能组成多少个互不相同且无重复数字的三位数？各是多少？1

2.从键盘输入一个字符串，将小写字母全部转换成大写字母，然后输出到一个磁盘文件"test"中保存。98

# 特征工程

## 实验介绍

### 关于本实验

本次实验我们将介绍特征选择和降维技术，作为数据挖掘工程中的重要环节，两种技术都可以提升模型的性能、模型运行效率和数据的可用性。

特征选择和降维都是在数据建模之前，对原始数据特征进行选择或是压缩的重要环节，也是数据挖掘过程中非常需要处理技巧和消耗时间较多的环节。本章选择最常用的特征选择和降维方法进行练习，希望进一步加强读者对特征选择和降维方法的理解，帮助读者掌握常规的特征选择和降维方法。

本次实验数据的特征数量其实是比较少的，但是这里我们为了为大家演示不同特征选择方法的使用，我们分别使用了Filter、Wrapper和Embedded三种方法。最后为大家介绍PCA的实现方法。

### 实验目的

本章包括两大部分实验：

1、特征选择，包括特征选择3种常见方法Filter、Wrapper、Embedding所涉及的经典选择方法；

2、PCA降维，具体题目如下：

特征选择 - Filter方法

特征选择 - Wrapper方法

特征选择 - Embedding方法

降维 - PCA原理实现

## 实验步骤

### 特征选择

特征选择是从特征集中找出与目标变量有影响且具有较高区分性特征去训练模型，获得预测性能更好的模型。特征选择的主要作用：减少特征数量、降维，使模型泛化能力更强，减少过拟合；增强对特征和特征值之间的理解。

选择特征首先是考虑特征的发散性和相关性两方面：

特征是否发散：如果一个特征不发散，例如方差接近于0，也就是说样本在这个特征上基本上没有差异，这个特征对于样本的区分并没有什么用。

特征与目标的相关性：优先选择与目标相关性高的特征。从特征的方差和相关性考虑。

特征选择方法有很多，主要包含特征减少和特征扩增。这里我们主要介绍特征减少的操作。主要包括：

单变量特征选择方法：

Filter（过滤法）

基于模型的特征选择方法：

Wrapper（包装法）

Embedded（嵌入法）

#### Filter（过滤法）

Filter用来衡量每个特征对目标属性的重要性程度，以此来对所有特征/属性进行排序，或者进行优选操作，特征选择的过程和后续的学习器无关（区别于另外两个方法)。过滤法是一种单变量统计方法，没有考虑到特征之间的关系特征之间的组合效应难以挖掘，因此很可能选择出重要但是冗余的特征

常用的具体技术有下述四种：方差选择法、卡方检验、互信息法和相关系数法。

前三种方法通过sklearn中的子模块中feature\_selection的数据调用：

方差选择法（调用VarianceThreshold方法）

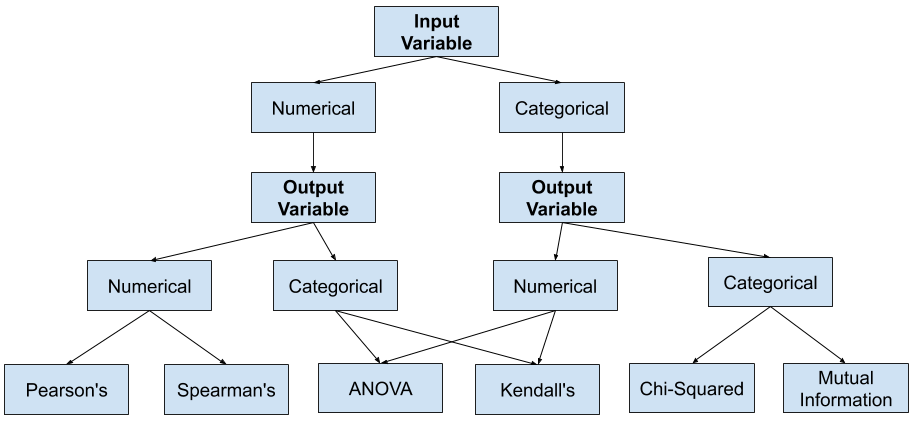
卡方检验法（调用SelectKBest函数）

互信息法（调用mutual\_info\_classif函数）

相关系数法需单独调用scipy模块中的stats.pearsonr()函数直接计算。

过滤法的选择：

方差选择法：可以作为特征选择的预处理，先去掉那些取值变化小的特征，然后再从接下来提到的的特征选择方法中选择合适的进行进一步的特征选择。



特征选择的方法

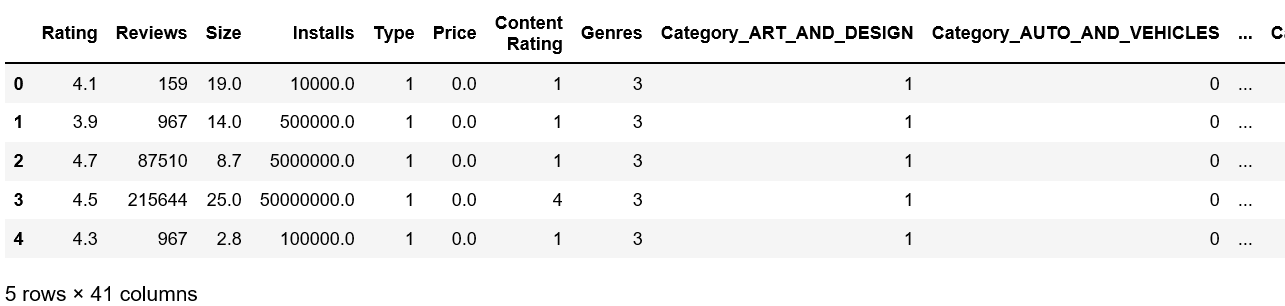
数据读取

代码：

raw\_df = pd.read\_csv("AppDataV2.csv",index\_col=0)#读取数据。index\_col=0：读取时不自动添加行号。

raw\_df.head()

输出：



代码：

raw\_data = raw\_df.drop(["Rating"], axis=1)#删除指定标签列

labels= raw\_df["Rating"]#标签

无量纲化

无量纲化的常用方法：

标准化：标准化的前提是特征值服从正态分布，标准化后，其转换成标准正态分布。

最大最小归一化：最小值-最大值归一化是将训练集中原始数据中特征的取值缩放到0到1之间。这种特征缩放方法实现对原始数据的等比例缩放，比较适用于数值比较集中的情况。

代码：

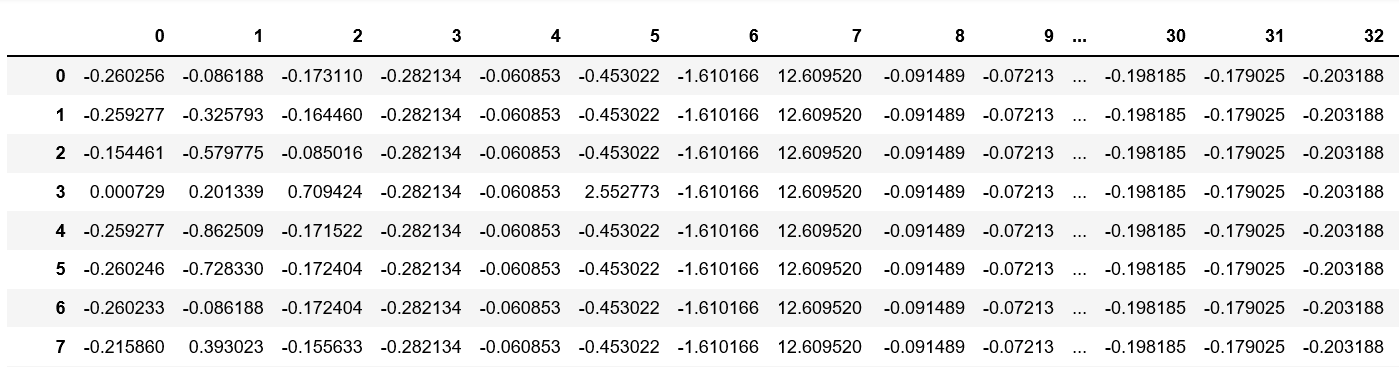
from sklearn.preprocessing import StandardScaler

sc\_X=StandardScaler(copy=True)

data = pd.DataFrame(sc\_X.fit\_transform(raw\_data))

data

输出：



请大家分别判断处理的输入变量和输出变量是什么类型的变量（离散数值变量、连续数值变量、有序分类变量、无序分类变量），并判断我们应该用哪种过滤法？

思路：输出是一个数值数据。Reviews/Size/Installs/Price是数值数据，用皮尔森关系系数。Type/Category/Content Rating/Genres是分类数据，用ANOVA。

方差选择法

首先我们用方差选择方作为特征选择的预处理

代码：

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import chi2

from sklearn.feature\_selection import VarianceThreshold

#这里我们使用原始数据

data\_after\_var = VarianceThreshold(threshold=0.01).fit\_transform(raw\_data,labels) #我们用了很多哑编码，需要把阈值设置的小些，使用阈值 0.01进行选择

data\_after\_var.shape

输出：

(10240, 31)

代码：

data\_after\_var = pd.concat([pd.DataFrame(data\_after\_var),labels],axis=1)

data\_after\_var.to\_csv("data\_after\_var")

皮尔森关系系数

对于数值数据我们，首先用皮尔森关系系数做特征选择。皮尔森相关系数是一种最简单的、能帮助理解特征和标签变量之间关系的方法。它衡量的是变量之间的线性相关性，其值在-1,1 之间；其中1 代表变量完全正相关，-1 代表完全负相关。

pearsonr(X,Y):

X：样本数据；

Y：样本标签

返回值：

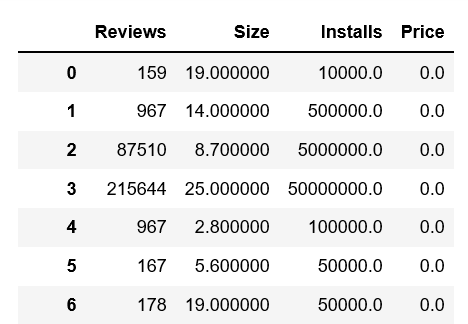
第一项是皮尔森相关系数，第二项是p\_value值。一般来说皮尔森相关系数越大，p\_value越小，线性相关性就越大

代码：

data\_numerical =raw\_data[["Reviews","Size","Installs","Price"]]

data\_numerical

输出：



代码：

from sklearn.feature\_selection import SelectKBest

from scipy.stats import pearsonr

print(pearsonr(data\_numerical["Reviews"],labels))

print(pearsonr(data\_numerical["Size"],labels))

print(pearsonr(data\_numerical["Installs"],labels))

print(pearsonr(data\_numerical["Price"],labels))

输出：

(0.11670456569847402, 2.1845948875648524e-32)

(0.057223017556035455, 6.848278283946546e-09)

(0.03962971755113228, 6.037723764673582e-05)

(-0.022532025751931757, 0.022602245106414908)

根据皮尔森关系系数，比较重要的特征依次为Reviews、Size、Installs和Price。

如果特征比较多的话，用SelectKBest保留前k个特征（取top k)，方法具体为：

sklearn.feature\_selection.SelectKBest(score\_func=<function f\_classif>, k=10)，

主要参数如下：

score\_func：选择卡方检验方法，其函数名称为chi2，并返回一对数组（得分，pvalues）或带有分数的单个数组。默认值为f\_classif（适用于分类任务）。

k：默认= 10，表示所选特征数。

主要属性为：

scores\_：array-like，shape =（n\_features，），即该特征在该特征选择方法下的得分；

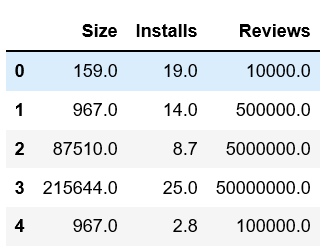
pvalues\_：array-like，shape =（n\_features，），特征分数的p值，如果score\_func仅返回分数，则为无。

代码：

data\_numerical = SelectKBest(lambda X,Y:np.array(list(map(lambda x:pearsonr(x,Y),X.T))).T[0],k=3).fit\_transform(data\_numerical,labels)

data\_numerical = pd.DataFrame(data\_numerical,columns={"Reviews","Size","Installs"})

data\_numerical.head()

输出：  


Type/Category/Content Rating/Genres是分类数据，下面我们用ANOVA进行特征选择。

在SelectKBest(score\_func=,k=10) 中参数score\_func给出统计指标：

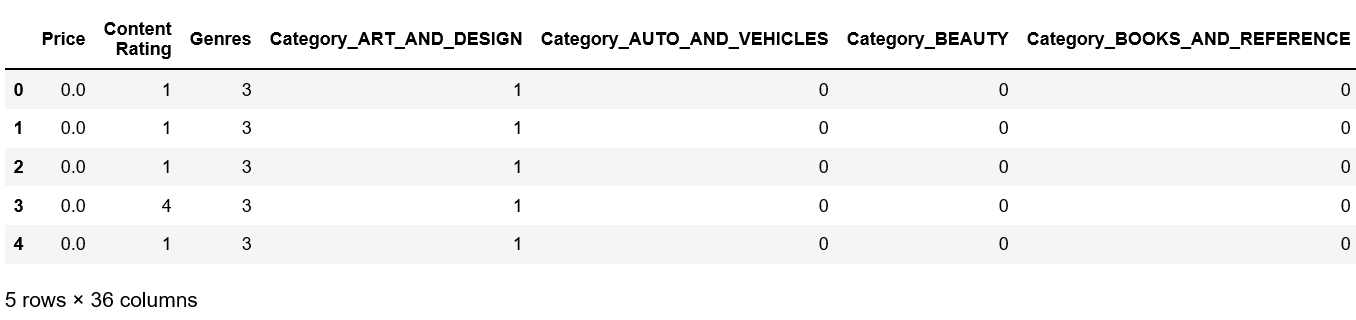
sklearn.feature\_selection.f\_classif:根据方差分析（ANOVA）的原理，以F-分布为依据，利用平方和与自由度所计算的祖居与组内均方估计出F值，适用于分类问题。

代码：

data\_categorical = pd.concat([raw\_data.iloc[:,4],raw\_data.iloc[:,5:]],axis=1)

data\_categorical.head()

输出：



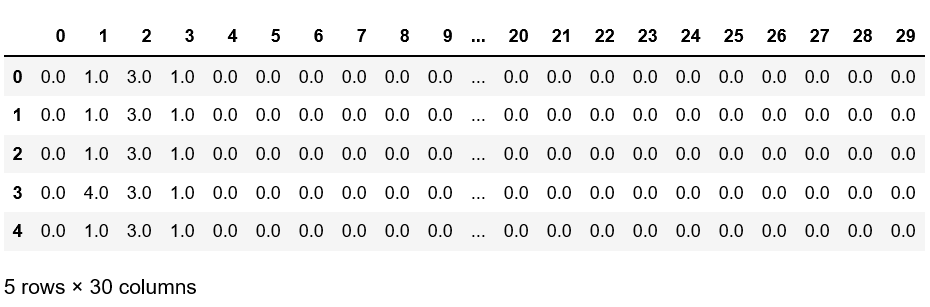
代码：

from sklearn.feature\_selection import f\_classif

data\_categorical = pd.DataFrame(SelectKBest(f\_classif, k=30).fit\_transform(data\_categorical, labels))

data\_categorical.head()

输出：



最后将数值和分类数据拼接在一起。

代码：

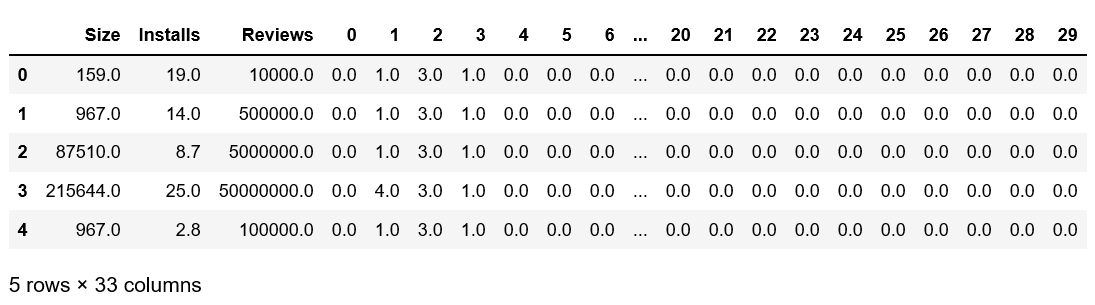
df\_after\_filter = pd.concat([data\_numerical,data\_categorical],axis=1)

df\_after\_filter.head()

df\_after\_filter = pd.concat([df\_after\_filter,labels],axis=1)

df\_after\_filter.to\_csv("df\_after\_filter.csv")

输出：



#### Wrapper（包裹法）

Wrapper，包装法，也形象地称为“弯刀法”，它解决思路没有过滤法直接，它是在确认后续的算法模型后，把模型本身的性能作为评价准则：选择一个目标函数来一步步的筛选特征。常用包装法是递归特征消除法，简称RFE，使用一个基模型来进行多轮训练，每轮训练后，移除若干权值系数的特征，再基于新的特征集进行下一轮训练。

在sklearn中，我们这里一起学习递归消除特征法，使用一个基模型来进行多轮训练，每轮训练后，消除若干权值系数的特征，再基于新的特征集进行下一轮训练。使用feature\_selection库的RFE类来选择特征。 sklearn.feature\_selection.RFE(estimator, step=1)

estimator：该参数传入用于递归构建模型的有监督型基学习器，要求该基学习器具有fit方法，且其输出含有coef\_或feature\_importances\_这种结果；

step：数值型，默认为1，控制每次迭代过程中删去的特征个数。

函数返回值：

n\_features\_：通过交叉验证过程最终剩下的特征个数；

support\_：被选择的特征的被选择情况（True表示被选择，False表示被淘汰）；

ranking\_：所有特征的评分排名；

estimator\_：利用剩下的特征训练出的模型

代码：

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import Lasso

#递归特征消除法，返回特征选择后的数据

#参数estimator为基模型

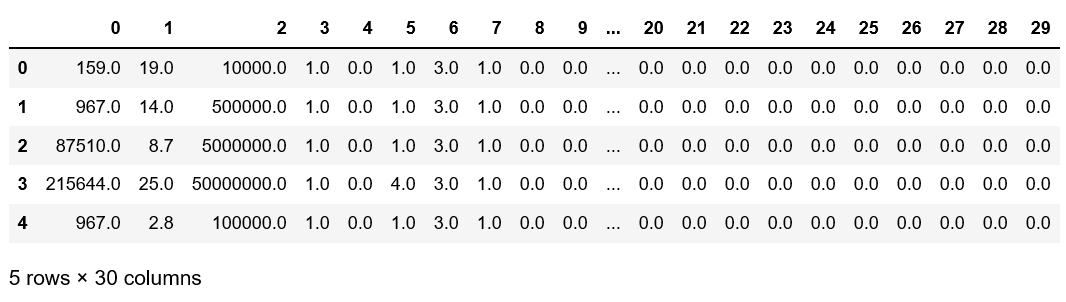
#参数n\_features\_to\_select为选择的特征个数

wrapper\_selection = RFE(estimator=Lasso(), n\_features\_to\_select=30)

df\_after\_wrapper = pd.DataFrame(wrapper\_selection.fit\_transform(raw\_data,labels))

df\_after\_wrapper.head()

输出：



代码：

wrapper\_selection.ranking\_

输出：

array([ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 5,

7, 9, 11, 10, 8, 6, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1])

代码：

df\_after\_wrapper.to\_csv("df\_after\_wrapper.csv")

#### Embedded (嵌入法)

Embedded，即嵌入法，相比前两种方法更加复杂，它利用机器学习算法和模型进行训练，得到各个特征的权值系数，根据权值系数从大到小来选择特征。常用嵌入法技术主要有两类方法：线性模型和正则化，其中包括具体的练习有2个：基于线性回归模型方法、基于L1的正则化方法；另一类是基于树模型的特征选择，这里仅练习基于随机森林的嵌入方法，随机森林具有准确率高、稳定性强、易于使用等优点，是目前最流行的机器学习算法之一，基于随机森林的预测模型能够用来计算特征的重要程度，因此能用来去除不相关的特征。

在sklearn中，我们选择使用SelectFromModel来进行嵌入法的特征选择。SelectFromModel主要包括：L1-based feature selection （基于 L1 的特征选取）、Tree-based feature selection（基于 Tree（树）的特征选取） 等。这里我们主要来学习基于 L1 的特征选取。

class sklearn.feature\_selection.SelectFromModel(estimator, threshold=None,prefit=False)：

estimator：构建特征选择实例的基本分类器。稀疏估计量对于回归中的 linear\_model.Lasso、分类中的linear\_model.LogisticRegression和svm.LinearSVC都很有用。

threshold：默认为None。该参数指定特征选择的阈值，词语在分类模型中对应的系数值大于该值时被保留，否则被移除。可设置的值有”mean”表示系数向量值的均值，”median”表示系数向量值的中值。当该参数设置值为None时，如果分类器具有罚项且罚项设置为l1，则阈值为1e-5，否则该值为”mean”。

prefit：布尔类型。默认值为False。是否对传入的基本分类器事先进行训练。如果设置该值为True，则需要对传入的基本分类器进行训练，如果设置该值为False，则只需要传入分类器实例即可

代码：

from sklearn.feature\_selection import SelectFromModel

from sklearn.linear\_model import Lasso# 以L1正则化的线性模型Lasso为例

#带L1惩罚项的逻辑回归作为基模型的特征选择。对于Lasso，参数alpha越大，被选中的特征越少

lasso = Lasso(alpha=0.001)

lasso.fit(data,labels)

model = SelectFromModel(lasso, prefit=True)

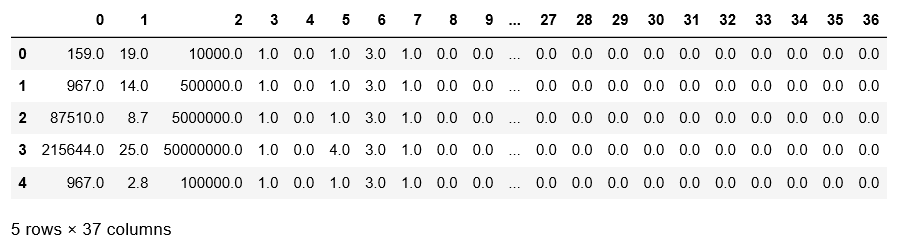
df\_after\_embedded = model.transform(raw\_data)

df\_after\_embedded = pd.DataFrame(df\_after\_embedded)

df\_after\_embedded.head()

df\_after\_embedded.to\_csv("df\_after\_embedded.csv")

输出：



以上为大家演示了Filter、Wrapper和Embedded三种特征选择的方法。

### 降维

PCA的思想是将n维特征映射到k维上(k<n)，这k维是全新的正交特征。这k维特征称为主成分，是重新构造出来的k维特征，而不是简单地从n维特征中去除其余n-k维特征（这也是与特征选择特征子集的方法的区别）。 在sklearn中，实现PCA降维的方法为：

sklearn.decomposition.PCA(n\_components=None, copy=True, svd\_solver='auto')

参数说明：

n\_components：PCA降维后的特征维度数目，或者是主成分的方差和所占的最小比例阈值。

copy：表示是否将原始数据复制一份。默认为True，则运行PCA算法后，原始数据的值不会有任何改变。

svd\_solver：指定奇异值分解SVD的方法。有4个可以选择的值：{‘auto’, ‘full’, ‘arpack’, ‘randomized’}。

'randomized' 一般适用于数据量大，数据维度多同时主成分数目比例又较低的PCA降维，它使用了一些加快SVD的随机算法。

'full' 则是传统意义上的SVD，使用了scipy库对应的实现。

'arpack' 和randomized的适用场景类似，区别是randomized使用的是scikit-learn自己的SVD实现，而arpack直接使用了scipy库的sparse SVD实现。当svd\_solve设置为'arpack'时，保留的成分必须少于特征数，即不能保留所有成分。

默认是'auto'，即PCA类会自己去在前面讲到的三种算法里面去权衡，选择一个合适的SVD算法来降维。一般来说，使用默认值就够了。

PCA实现

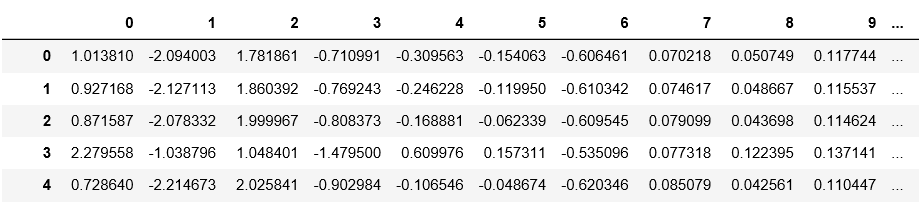
代码：

from sklearn.decomposition import PCA

pca = PCA(n\_components=0.85)

data\_after\_pca=pd.DataFrame(pca.fit\_transform(data))

data\_after\_pca

输出  


经过PCA降维后，我们将91维的特征转换为42维的特征，同时保存了85%的信息量。

可视化

代码：

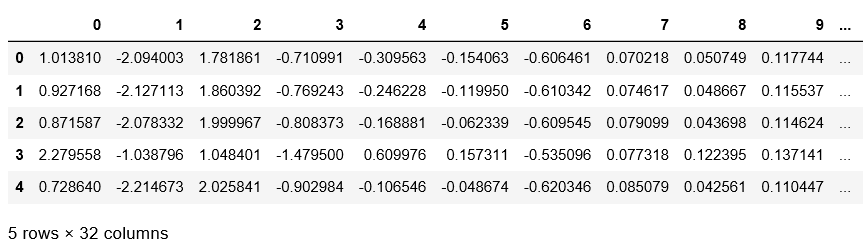
scaled\_labels = pd.DataFrame(labels.astype("int"),columns={"Rating"})

data\_after\_pca\_withlabels = pd.concat([data\_after\_pca,labels],axis=1)

data\_after\_pca\_withlabels.head()

data\_after\_pca\_withlabels.to\_csv("after\_pca.csv")

输出：



代码：

import matplotlib.pyplot as plt

def draw\_graph(X):

for i in range(5):

plt.scatter(data\_after\_pca\_withlabels.loc[data\_after\_pca\_withlabels.Rating==i,0],

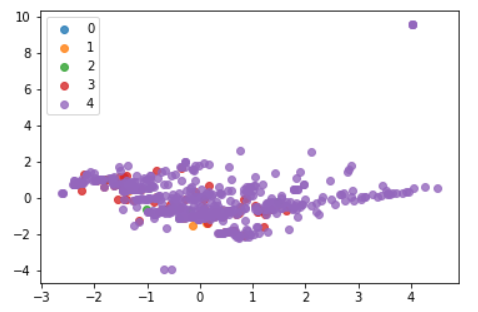
data\_after\_pca\_withlabels.loc[data\_after\_pca\_withlabels.Rating==i,1], alpha=0.8, label='%s' % i)

plt.legend()

plt.show()

draw\_graph(data)

输出：



降维后的数据

## 实验小结

本实验主要讲解特征工程与数据降维的实现，包括Filter、Wrapper和Embedded三种特征选择方法和PCA降维的实现。

## 思考题

鸢尾花数据集是数据挖掘练习中最常使用的学习数据集，分别有4个特征表征花的不同特点，目标变量target是3种不同的鸢尾花类型。4个特征分别是：sepal length (cm)（花萼长度）、sepal width (cm)（花萼宽度）、petal length (cm)（花瓣长度）、petal width(cm)（花瓣宽度）。目标变量target是三种鸢尾花：setosa（山鸢尾）、versicolor（杂色鸢尾）、virginica（维吉尼亚鸢尾）。

利用开源的鸢尾花数据集分别完成Filter方法中的方差选择法、卡方检验方法、相关系数法、互信息法4个具体方法的编程实现。

# 电信用户分析

## 实验介绍

### 简介

电信客户信用度是指根据客户在网时间、缴费情况、客户积分等相关要素对集团客户或个人客户进行评分，用于衡量客户缴费行为的好坏。

本案例使用电信用户的通信行为数据集，目的是进行用户信用分析且最终对客户进行分群。由于是没有标注的训练样本，特征仅7个维度，在大数据挖掘实验中并非特征较多的情景。本实验重点在掌握降维算法的实践，以及对降至3维后的特征进行可视化呈现，最后利用聚类这一无监督学习算法将用户进行分群，针对不同客户群体再结合原始业务指标进行分析，确定每个聚类群体的信用行为特点。

数据集：(data\telecom.csv.csv)

### 实验目的

通过案例实践，掌握数据分析挖掘的基本流程，及通过聚类实现群体细分和通过PCA降维实现主成分分析的基本思想，并结合实践操作加深理论的理解和认识，为实际的业务应用奠定理论和技术基础。

## 实验环境要求

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，pandas，sklearn使用pip安装命令如下：

pip install seaborn

pip install numpy

pip install sklearn

pip install pandas

其他安装包如上进行安装。

## 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，进行数据降维操作，随后直接选择sklearn模块中的KMeans算法进行建模，通过轮廓系数对KMeans算法进行K值调优，选择最优的K值进行模型训练，最后通过统计分析对聚类结果进行画像分析，从而明确每一类人群的特征。

## 实验详细设计与实现

### 导入实验环境

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")

### 数据准备

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。

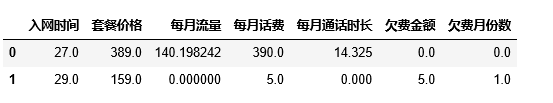
#encoding=utf8

# 模块导入与数据读取

X = pd.read\_csv(r"data\telecom.csv",encoding="utf8")

X.head(2)

输出如下结果：



数据集前2行

数据标准化

通过如下代码，实现数据集的标准化处理。

# 数据标准化

from sklearn import preprocessing

X\_scaled = preprocessing.scale(X)

X\_scaled

输出如下结果：

array([[-0.32484896, 1.63037135, -0.14052452, ..., -0.5953273 ,

-0.2199407 , -0.31623138],

...,

[-0.12245085, -0.29646995, -0.16747396, ..., 0.20572063,

-0.2199407 , -0.31623138]])

### 数据降维

使用PCA进行数据降维

通过降维实现主要特征提取，为后续的聚类提供数据准备。

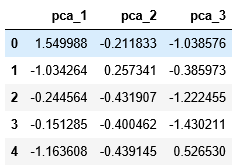
# 使用PCA进行数据降维

#此处的主成分维度我们人为设定为3，对于属性较少的数据集，属于常规会选择的维度数，后面也会看到，这个也是出于可以可视化的需求

#将设置了维数的模型作用到标准化后的数据集并输出查看

#将降维后的数据前5行显示出来如下：

X\_pca\_frame.head(5)



降维后指标

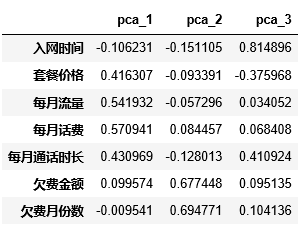
降维指标与原指标关联

这三个指标与原始字段的系数可以通过pca的components\_属性获取。

#三个指标与原始字段的系数获取

降维后生成的3个特征'pca\_1','pca\_2','pca\_3'与原始的特征之间的系数关系如下图表，这一步并不是实验数据挖掘的必要环节，但是对理解PCA的变换效果非常有必要。

输出如下结果：



指标与原始字段的系数

对不同的K值进行计算，筛选最优的K值

当K值设置不同时，聚类效果会存在较大的差别，因此在实际应用中需要依据聚类效果，对K值进行优化。执行如下代码，通过Calinski-Harabasz分数值对不同的K值进行评估。

# 对不同的K值进行计算，筛选最优的K值

from mpl\_toolkits.mplot3d import Axes3D

from sklearn import metrics

from sklearn.cluster import KMeans

#KMeans算法实例化，将其设置为K=range(2,14)

d={}

fig\_reduced\_data=plt.figure(figsize=(12,12))

for k in range(2,14):

est=KMeans(n\_clusters=k,random\_state=111)

#作用到降维后的数据集上

y\_pred=est.fit\_predict(X\_pca)

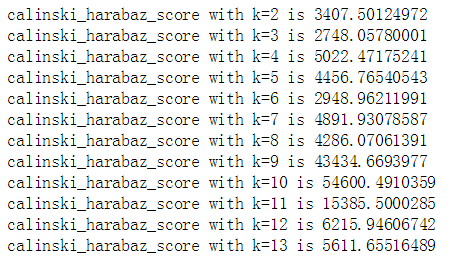
#评估不同K值聚类算法效果

score=metrics.calinski\_harabaz\_score(X\_pca\_frame,y\_pred)

d.update({k: score})

print('calinski\_harabaz\_score with k={0} is {1}'.format(k,score))

输出如下结果：



不同K取值聚类效果评估

绘制不同k值对应的score，找到最优的K值

x=[]

y=[]

for k,score in d.iteritems():

x.append(k)

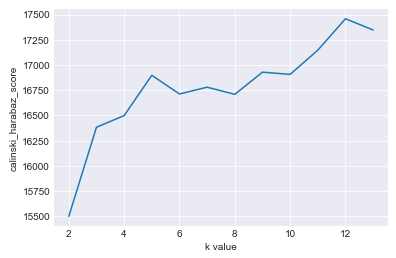
y.append(score)

plt.plot(x,y)

plt.xlabel(‘k value’)

plt.ylabel(‘calinski\_harabaz\_score’)

输出如下结果：



聚类效果Calinski-Harabasz分数值

从聚类结果的图片可以看出，当K为12时，聚类效果较好。

### K-Means聚类模型训练

在降维的数据上使用K-means聚类算法将数据聚成10类，这里的10类是事先人工指定的划分群数，通常是结合后续的业务运营方案（如有10类运营方案支持划分10类），或是结合业务专家的建议凭经验指定，或者是指定不同的K值后比较得来。代码如下：

训练简单模型

# K-means聚类建模

#运行KMeans聚类算法

from sklearn.cluster import KMeans

#此处指定K=12

est = KMeans(n\_clusters=12)

est.fit(X\_pca)

#获取数据标签值

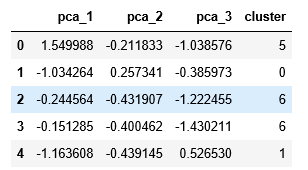
kmeans\_clustering\_labels = pd.DataFrame(est.labels\_,columns=['cluster'])

#将聚类结果与降维特征数据进行拼接

X\_pca\_frame = pd.concat([X\_pca\_frame,kmeans\_clustering\_labels], axis=1)

X\_pca\_frame.head(5)

输出如下结果：



聚类示例

样本筛选

通过对聚类数据进行筛选，删除每一个聚类中的噪音和异常点，提取每一个聚类中的典型用户。为了避免离群点的影响，按照每一类的分位数筛选离中心点较近的用户作为该类的代表用户。这里我们引入了一个经典的箱型图筛选技术。代码如下：

#样本筛选

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

X.index = X\_pca\_frame.index

X\_full = pd.concat([X, X\_pca\_frame],axis=1)

grouped = X\_full.groupby('cluster')

result\_data = pd.DataFrame()

#此处利用箱型图进行异常值分析

for name,group in grouped:

print (name,group['pca\_1'].count())

desp = group[['pca\_1','pca\_2','pca\_3']].describe()

for att in ['pca\_1','pca\_2','pca\_3']:

lower25 = desp.ix['25%',att]

upper75 = desp.ix['75%',att]

IQR = upper75 - lower25

min\_value = lower25 - 1.5\*IQR

max\_value = upper75 + 1.5\*IQR

group = group[(group[att] > min\_value) & (group[att] < max\_value) ]

result\_data = pd.concat([result\_data, group],axis=0)

print(name,group['pca\_1'].count())

#分别列出 筛选前后每个特征的数量

print('remanin sample : ',result\_data['pca\_1'].count())

筛选前后的用户样本按信用类别列举如下：

0 11129

0 9530

1 6613

1 5844

2 2021

2 1500

3 1947

3 1606

4 47

4 30

5 2891

5 2218

6 3602

6 2808

7 21

7 20

8 369

8 297

9 1360

9 1248

查看仍保留下的数据数量

remanin sample : 25101

### 聚类结果分析

查看降维数据

代码如下：

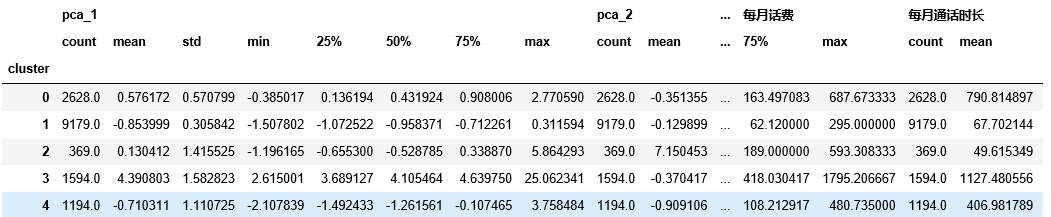
#1)主成分

components\_frame = pd.DataFrame(pca.components\_,index=[u'主成分1',u'主成分2',u'主成分3'],columns=X.columns)

#2）降维数据和聚类初步结果

X\_full.groupby('cluster').describe()

结果如下：



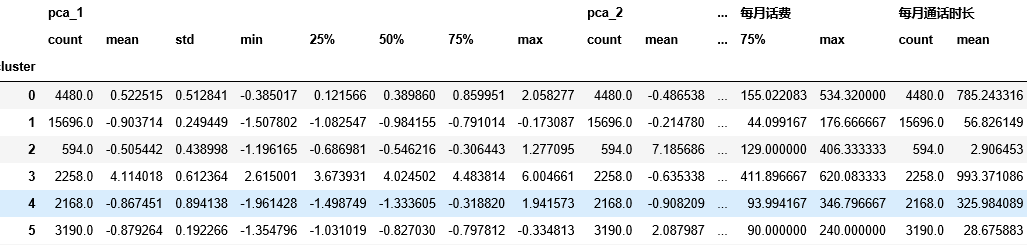
降维后数据示例

代码如下：

#筛选样本聚类统计

result\_data.groupby('cluster').describe()

结果如下：



聚类结果可视化

代码如下：

#绘图

cluster\_2\_color = {0:'red',1:'green',2:'blue',3:'yellow',4:'cyan',5:'black',6:'magenta', 7:'#fff0f5', 8:'#ffdab9',9:'#ffa500' }

#1. 原始数据降维后的可视化图

fig\_reduced\_data = plt.figure()

ax\_reduced\_data = fig\_reduced\_data.add\_subplot(111, projection='3d')

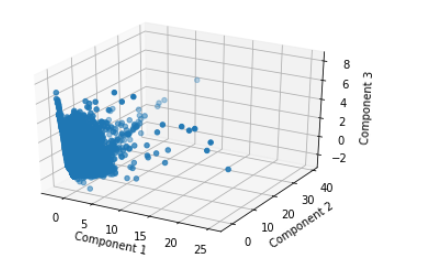
ax\_reduced\_data.scatter(X\_pca\_frame['pca\_1'].values,X\_pca\_frame['pca\_2'].values,X\_pca\_frame['pca\_3'].values)

ax\_reduced\_data.set\_xlabel('Component 1')

ax\_reduced\_data.set\_ylabel('Component 2')

ax\_reduced\_data.set\_zlabel('Component 3')

结果如下：



原始数据降维后的可视化图

对不同群分别指定不同颜色表示，代码如下：

#2. 聚类之后的数据可视化图

colors\_clustered\_data = X\_pca\_frame['cluster'].map(cluster\_2\_color)

fig\_clustered\_data = plt.figure()

ax\_clustered\_data = fig\_clustered\_data.add\_subplot(111, projection='3d')

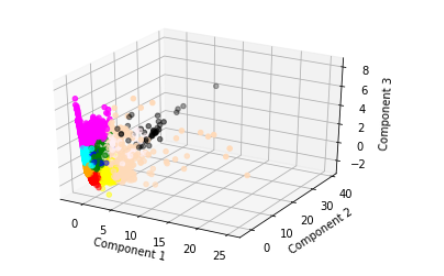
ax\_clustered\_data.scatter(X\_pca\_frame['pca\_1'].values,X\_pca\_frame['pca\_2'].values,X\_pca\_frame['pca\_3'].values,c=colors\_clustered\_data)

ax\_clustered\_data.set\_xlabel('Component 1')

ax\_clustered\_data.set\_ylabel('Component 2')

ax\_clustered\_data.set\_zlabel('Component 3')

结果如下：



聚类之后的数据可视化图

查看去除异常值后的不同群，代码如下：

#3. 筛选后的数据聚类可视化图

colors\_filtered\_data = result\_data['cluster'].map(cluster\_2\_color)

fig\_filtered\_data = plt.figure()

ax\_filtered\_data = fig\_filtered\_data.add\_subplot(111, projection='3d')

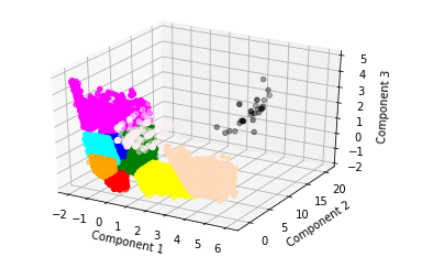
ax\_filtered\_data.scatter(result\_data['pca\_1'].values,result\_data['pca\_2'].values,result\_data['pca\_3'].values,c=colors\_filtered\_data)

ax\_filtered\_data.set\_xlabel('Component 1')

ax\_filtered\_data.set\_ylabel('Component 2')

ax\_filtered\_data.set\_zlabel('Component 3')

结果如下：



筛选后的数据聚类可视化图

代码如下：

#4. 每个聚类的在pca1,pca2,pca3上的均值可视化图

grouped = result\_data.groupby('cluster')

#每一个聚类的盒图

colors = result\_data['cluster'].map(cluster\_2\_color)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(result\_data['pca\_1'].values,result\_data['pca\_2'].values,result\_data['pca\_3'].values,c=colors)

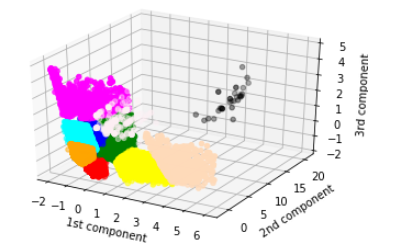
ax.set\_xlabel('1st component')

ax.set\_ylabel('2nd component')

ax.set\_zlabel('3rd component')

plt.show()

结果如下：



每个聚类的在pca1,pca2,pca3上的均值可视化图

用户画像分析

对于10个聚类中的用户，通过"欠费金额","每月话费"和"入网时间"三个原始字段的均值来分析不同类用户的特点。为了显示正常，新建三个英文名命名的字段arrearage(欠费金额)，telephone\_fare(每月话费)和month\_of\_access（入网时间）。代码如下：

X\_full["arrearage"] = X\_full[u"欠费金额"]

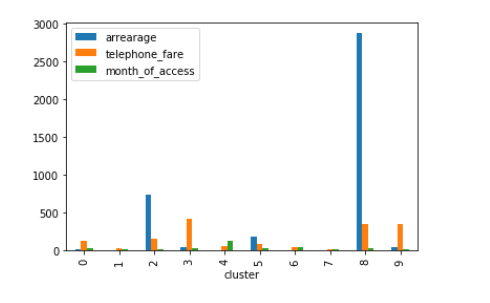
X\_full["telephone\_fare"] = X\_full[u"每月话费"]

X\_full["month\_of\_access"] = X\_full[u"入网时间"]

%matplotlib inline

X\_full.groupby("cluster")[["arrearage","telephone\_fare","month\_of\_access"]].mean().plot(kind="bar")

结果如下：



用户分类结果

从上图可以分析出，第4、10类用户（下标为3、9）入网时间较长，消费能力中等，但是基本无欠费行为，综合信用水平较好。第3、9类用户（下标为2与8）欠费金额很高，消费能力也较强，且大部分入网时间较短，尤其是对于第9类用户属于需要进一步细分和评估，其中会存在一些劣质用户。第1、5、6类用户消费水平一般，但有较长的入网时间，可以尝试进一步营销活动或者防止流失策略。

注：由于K-means聚类具有随机性（每次初始质心的选择不同造成），不同的运行结果可能会与上图存在差异。

## 思考题

1. 根据2.2.2，补全3.4.3数据降维部分缺少的代码。

2. 无监督学习适应场景？

3. K-means算法中K的确定还有哪些方法？

## 实验小结

本章主要讲解了电信用户实际业务场景下的数据聚类实验的相关操作，包含数据的标准化，数据降维、建模，最终的用户分析等过程。

# APP评分预测

## 实验介绍

### 简介

本次实验利用数据预处理与特征工程中处理好的数据集来训练一个回归和分类模型，对评分的预测。

首先线性回归、SVM、KNN回归算法，训练三个回归模型。接下来将评分数据离散化，用决策树模型训练分类模型。这样可以更好学习算法的使用和模型的评估。

### 实验目的

掌握线性回归算法的应用实践

掌握SVM算法的应用实践

掌握决策树算法的应用实践

掌握KNN算法的应用实践

## 实验环境要求

本地PC，Python3

## 实验总体设计

## 实验步骤

### 数据读取

代码：

#导入相关库

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

#读取数据集

data\_after\_pca = pd.read\_csv('after\_pca.csv',index\_col=0)

data = pd.read\_csv('AppDataV2.csv',index\_col=0)

data\_after\_var = pd.read\_csv("data\_after\_var",index\_col=0)

data\_after\_filter = pd.read\_csv("df\_after\_filter.csv",index\_col=0)

#首先确定样本的数据的标签

X = data.drop(["Rating"],axis='columns')

Y = data["Rating"]

X\_var = data\_after\_var.drop(["Rating"],axis='columns')

Y\_var = data\_after\_var["Rating"]

X\_pca = data\_after\_pca.drop(["Rating"],axis='columns')

Y\_pca = data\_after\_pca["Rating"]

X\_filter = data\_after\_filter.drop(["Rating"],axis='columns')

Y\_filter = data\_after\_filter["Rating"]

X.info()

输出：

<class 'pandas.core.frame.DataFrame'>

Int64Index: 10240 entries, 0 to 10239

Data columns (total 40 columns):

Reviews 10240 non-null int64

Size 10240 non-null float64

Installs 10240 non-null float64

Type 10240 non-null int64

Price 10240 non-null float64

Content Rating 10240 non-null int64

Genres 10240 non-null int64

Category\_ART\_AND\_DESIGN 10240 non-null int64

Category\_AUTO\_AND\_VEHICLES 10240 non-null int64

Category\_BEAUTY 10240 non-null int64

Category\_BOOKS\_AND\_REFERENCE 10240 non-null int64

…

代码：

#数据集划分

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 10)

### 线性回归

sklearn.linear\_model.LinearRegression(fit\_intercept=True, normalize=False, copy\_X=True)

参数说明：

fit\_intercept：默认True，是否计算模型的截距，为False时，则数据中心化处理。

normalize：默认False，是否中心化，或者使用sklearn.preprocessing.StandardScaler()。

copy\_X：默认True，否则X会被改写。

代码：

from sklearn.linear\_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_squared\_error,mean\_absolute\_error,accuracy\_score,accuracy\_score,r2\_score

#初始化线性回归模型

linreg = LinearRegression()

#训练模型

linreg.fit(X\_train,y\_train)

#训练集上的MSE

linreg\_pred\_train = linreg.predict(X\_train)

linreg\_mse\_train = mean\_squared\_error(linreg\_pred\_train,y\_train)

#输出测试集上的测试结果

linreg\_pred\_test=linreg.predict(X\_test)

linreg\_mse\_test = mean\_squared\_error(linreg\_pred\_test,y\_test)

print("训练集MSE：", linreg\_mse\_train)

print("测试集MSE：", linreg\_mse\_test)

输出：

训练集MSE： 0.2308388846541144

测试集MSE： 0.22350603712434897

### **SVM回归**

sklearn.svm.SVR（kernel ='rbf'，degree = 3，gamma ='auto\_deprecated'，coef0 = 0.0，tol = 0.001，C = 1.0，verbose = False，max\_iter = -1 ）

参数说明：

kernel：指定要在算法中使用的内核类型。可以是'linear'，'poly'，'rbf'，'sigmoid'，'precomputed'或者callable之一，默认为rbf。

degree：多项式poly函数的维度，默认是3，选择其他核函数时会被忽略。

gamma：‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。默认是’auto’，则会选择1/n\_features

coef0：核函数的常数项。对于‘poly’和 ‘sigmoid’有用，默认值= 0.0。

tol：停止训练的误差值大小，默认值= 1e-3。

C：惩罚参数，默认= 1.0。C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样对训练集测试时准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。

verbose：日志。

max\_iter：最大迭代次数。-1为无限制。

**代码：**

from sklearn.svm import SVR

from sklearn.metrics import accuracy\_score

#初始化决策树模型

svr=SVR(kernel='rbf',C=1)

#训练

svr.fit(X\_train,y\_train\_int)

#训练集上的MSE

svr\_pred\_train = svr.predict(X\_train)

svr\_mse\_train = mean\_squared\_error(svr\_pred\_train,y\_train)

#输出测试集上的测试结果

svr\_pred\_test=svr.predict(X\_test)

svr\_mse\_test = mean\_squared\_error(svr\_pred\_test,y\_test)

print("训练集MSE：", svr\_mse\_train)

print("测试集MSE：", svr\_mse\_test)

输出：

训练集MSE： 0.28852538417710893

测试集MSE： 0.44229386180754227

### KNN回归

sklearn.neighbors.KNeighborsRegressor(n\_neighbors=5, weights='uniform', algorithm='auto', leaf\_size=30)：

参数说明：

n\_neighbors：knn算法中指定以最近的几个最近邻样本具有投票权，默认参数为5

algrithm：即内部采用什么算法实现。有以下几种选择参数：

'ball\_tree':球树、

'kd\_tree':kd树、

'brute':暴力搜索、

'auto':自动根据数据的类型和结构选择合适的算法。默认情况下是‘auto’。

暴力搜索就不用说了大家都知道。具体前两种树型数据结构哪种好视情况而定。KD树是对依次对K维坐标轴，以中值切分构造的树,每一个节点是一个超矩形，在维数小于20时效率最高ball tree 是为了克服KD树高维失效而发明的，其构造过程是以质心C和半径r分割样本空间，每一个节点是一个超球体。一般低维数据用kd\_tree速度快，用ball\_tree相对较慢。超过20维之后的高维数据用kd\_tree效果反而不佳，而ball\_tree效果要好，具体构造过程及优劣势的理论大家有兴趣可以去具体学习。

leaf\_size:这个值控制了使用KD树或者球树时， 停止建子树的叶子节点数量的阈值。这个值越小，则生成的KD树或者球树就越大，层数越深，建树时间越长，反之，则生成的KD树或者球树会小，层数较浅，建树时间较短。默认是30.

请根据线性回归的实现和KNN的参数说明，训练一个KNN模型。代码填写：

#初始化knn模型

knn\_model = KNeighborsRegressor(n\_neighbors=50)

#训练

knn\_model.fit(X\_train,y\_train)

#训练集上的MSE

knn\_pred\_train = knn\_model.predict(X\_train)

knn\_mse\_train = mean\_squared\_error(knn\_pred\_train,y\_train)

#输出测试集上的测试结果

knn\_pred\_test=knn\_model.predict(X\_test)

knn\_mse\_test = mean\_squared\_error(knn\_pred\_test,y\_test)

print("训练集MSE：", knn\_mse\_train)

print("测试集MSE：", knn\_mse\_test)

输出：

训练集MSE： 0.2044843076171875

测试集MSE： 0.20693282421875

接下来简单对三个模型的输出精度进行对比。

代码：

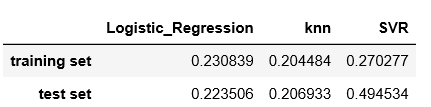
model\_mse = pd.DataFrame(data=[[linreg\_mse\_train,knn\_mse\_train,svr\_mse\_train],

[linreg\_mse\_test,knn\_mse\_test,svr\_mse\_test]],

columns=['Logistic\_Regression','knn','SVR'],index=["training set","test set"])

model\_mse

输出：

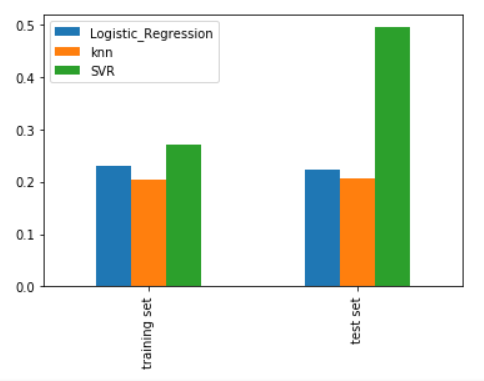


代码：

plt.figure(figsize=(20, 10))

model\_mse.plot(kind = 'bar')

输出：

LR、KNN、SVR算法对比

### 模型评估与选择

通过上面的对比，可以看出4个模型都欠拟合的状态。接下来将使用交叉验证、网格搜索和随机搜索的方式，选择模型的超参数。

实验目的：

（1）掌握交叉验证算法的应用实践

（2）掌握网络搜索的实现

（3）掌握随机搜索的实现

#### 交叉验证

将用交叉验证来搜索KNN的模型中n\_neighbors的最佳参数值。

代码：

from sklearn.model\_selection import cross\_val\_score # K折交叉验证模块

#建立测试参数集

k\_range = range(15, 100)

k\_scores = []

#藉由迭代的方式来计算不同参数对模型的影响，并返回交叉验证后的平均准确率

for k in k\_range:

knn = KNeighborsRegressor(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=10)

k\_scores.append(scores.mean())

#可视化数据

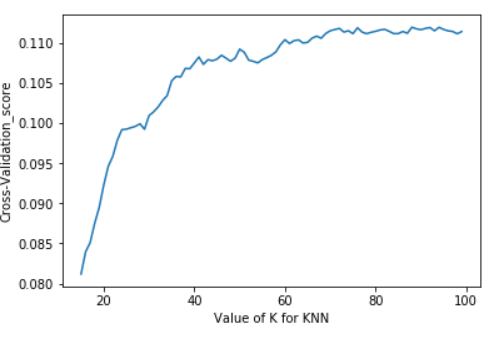
plt.plot(k\_range, k\_scores)

plt.xlabel('Value of K for KNN')

plt.ylabel('Cross-Validation\_score')

plt.show()

输出：

不同K值对模型的影响

根据上图结果，可知n\_neighbors的数值在62左右最佳，大于此值后模型的表现没有明显提升。

#### 参数搜索

本小节分别用将用网格搜索和随机搜索的方式，对决策树分类器和SVM回归模型的超参数进行搜索。

代码：

###决策树分类器

from sklearn.model\_selection import GridSearchCV

params = [{'criterion':['gini'],'max\_depth':[30,50,60,100],'min\_samples\_leaf':[2,3,5,10],'min\_impurity\_decrease':[0.1,0.2,0.5]},

{'criterion':['gini','entropy']},

{'max\_depth': [30,60,100], 'min\_impurity\_decrease':[0.1,0.2,0.5]}]

best\_model = GridSearchCV(dtree, param\_grid=params,cv = 5,scoring ="accuracy")

best\_model.fit(X\_train,y\_train\_int)

print('最优分类器:',best\_model.best\_params\_,'最优分数:', best\_model.best\_score\_) # 得到最优的参数和分值

输出：

最优分类器: {'criterion': 'gini', 'max\_depth': 30, 'min\_impurity\_decrease': 0.1, 'min\_samples\_leaf': 2} 最优分数: 0.781982421875

接下来用随机搜索搜索SVM回归模型的参数。

代码：

from sklearn.model\_selection import RandomizedSearchCV

params\_svr = {'kernel': ['rbf'], 'C': np.logspace(-3, 2, 6), 'gamma':np.arange(0,10,2)}

best\_svr\_model = RandomizedSearchCV(svr, param\_distributions=params\_svr,cv = 3,scoring ="neg\_mean\_squared\_error")

best\_svr\_model.fit(X,Y)

print('最优分类器:',best\_svr\_model.best\_params\_,'最优分数:', best\_svr\_model.best\_score\_) # 得到最优的参数和分值

输出：

最优分类器: {'kernel': 'rbf', 'gamma': 4, 'C': 1.0} 最优分数: -0.24169698734960568

## 思考题

将数据的标签变为整型，自行编码训练一个决策树分类模型，其中决策树模型使用方法如下：

DecisionTreeClassifier(criterion="mse",splitter="best",max\_depth=None,min\_samples\_split=2,min\_samples\_leaf=1,min\_weight\_fraction\_leaf=0., max\_features=None,random\_state=None,max\_leaf\_nodes=None)

参数说明：

criterion:切分质量的评价准则。默认为'mse'(mean squared error)。

splitter:指定了在每个节点切分的策略。有两种切分策略：

(1).splitter='best':表示选择最优的切分特征和切分点。

(2).splitter='random':表示随机切分。

max\_depth:指定树的最大深度。如果为None，则表示树的深度不限，直到每个叶子都是纯净的。

min\_samples\_split:默认为2。它指定了分裂一个内部节点(非叶子节点)需要的最小样本数。如果为浮点数(0到1之间)，最少样本分割数为ceil(min\_samples\_split \* n\_samples)

min\_samples\_leaf:指定了每个叶子节点包含的最少样本数。如果为浮点数(0到1之间)，每个叶子节点包含的最少样本数为ceil(min\_samples\_leaf \* n\_samples)

min\_weight\_fraction\_leaf:指定了叶子节点中样本的最小权重系数。默认情况下样本有相同的权重。

max\_feature:

(1).如果是整数，则每次节点分裂只考虑max\_feature个特征。

(2).如果是浮点数(0到1之间)，则每次分裂节点的时候只考虑int(max\_features \* n\_features)个特征。

(3).如果是字符串'auto',max\_features=n\_features。

(4).如果是字符串'sqrt',max\_features=sqrt(n\_features)。

(5).如果是字符串'log2',max\_features=log2(n\_features)。

(6).如果是None，max\_feature=n\_feature。

random\_state:随机数生成器

max\_leaf\_nodes:

(1).如果为None，则叶子节点数量不限。

(2).如果不为None，则max\_depth被忽略。

## 实验小结

本章通过代码实践，帮助学习者了解了机器学习算法实践应用的流程，并使用处理过的APP评分数据进行回归和分类建模，最后通过交叉验证、网络搜素和随机搜索等算法对模型进行超参数寻优。

1. **糖尿病预测**

## 实验说明

该数据集(pima-indians-diabetes.data)是来自美国疾病控制预防中心的数据，背景是记录美国的糖尿病症状信息，现在美国1/7的成年人患有糖尿病。但是到2050年，这个比例将会快速增长至高达1/3。可以利用从UCI机器学习数据库里一个关于印第安人糖尿病数据集，通过数据挖掘相关算法来预测糖尿病，该问题本质上是一个二元分类问题。

## 实验建模流程要求

基于Diabete数据，使用朴素贝叶斯进行糖尿病预测。

### 环境要求

Python 3.7

### 实验实现步骤要求

#### 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

#### 数据导入与初步探索

要求载入本地数据集(pima-indians-diabetes.data)，以dataframe形式存放后，命名为df

查看数据尺寸、打印信息，判断特征的类型（名称性、数值型），目标变量分布以及查看是否均衡

对df特征进行相关性可视化

对df每个特征的分布进行可视化查看

对输入特征进行降维，选择PCA，并按提示补充如下代码中划线部分内容。

### 对输入特征进行降维处理

from sklearn.decomposition import PCA

from sklearn import preprocessing #调用标准化模块

#降维训练前需要对数据标准化

pca = PCA( ) # 保留99%信息的主成分个主成分

X\_pca =pca.fit(X\_std).transform(X\_std)

结合相关性分析和降维后的结论，选择数据进行拆分为训练集和测试集，拆分比例设置为0.1，指定以Target的比例做分层抽样。部分提示如下：

from collections import Counter

from sklearn.model\_selection import train\_test\_split

选择朴素贝叶斯算法对拆分后的数据进行建模训练和预测，其中要求将原始模型做5折交叉验证，评估指标选择f1。

部分提示如下：

#引入朴素贝叶斯和交叉验证的库

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import cross\_val\_score

#引入评价指标的库

from sklearn.metrics import f1\_score

确认模型，进行预测。

## 思考题

补充缺失的代码。

# 信用违约预测

## 实验说明

风险管控已经成为了今年金融市场的重要主题之一，银行作为贷方，随时都面临着借贷者违约的风险。传统的专家规则在金融科技时代逐渐过时，机器学习和金融业务的交叉也延伸到信贷领域。违约预测就是其中一重要应用。本实验基于信贷业务场景中一个月内的抽样数据，数据集有34个维度，Target表示客户在接下来一个月是否有违约。模型生成后可使用当前月的数据预测接下来一个月客户是否会违约。

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

## 实验建模流程要求

违约预测只有违约和没有预约两种结果，属于二分类问题。针对二分类问题，可使用的算法有逻辑回归、朴素贝叶斯、支持向量机、树模型等。考虑到实验的完整性和实用性，本实验选用业界常用的逻辑回归和随机森林来做对比。考虑到样本极度不均衡，模型评价选用综合指标f1\_score。

### 环境要求

Python 3.7

### 实验实现步骤要求

要求导入相关数据读取、处理、分析、可视化，算法模块等

数据读取，数据框类型，命名为 df

查看Target的分布 ,是否违约（1是，0否）

可视化观察特征相关性

存储相关性过高的特征对,对于相关性过高的的特征，删除其中一个（根据工程经验，以0.8为界）

# 选择出符合内容的单元格对应的行、列标签

cols\_pair\_to\_drop = []

for index\_ in corr\_matrix.index:

for col\_ in corr\_matrix.columns:

if corr\_matrix.loc[index\_,col\_] >= 0.8 and index\_!=col\_ and (col\_,index\_) not in cols\_pair\_to\_drop:

cols\_pair\_to\_drop.append((index\_,col\_))

#丢弃特征对中的一个

cols\_to\_drop = np.unique([col[1] for col in cols\_pair\_to\_drop]) #对于一维数组或者列表，unique函数去除其中重复的元素，并按元素由大到小返回一个新的无元素重复的元组或者列表

df.drop(cols\_to\_drop,axis=1,inplace=True)

df.head()

打印出缺失率最高的前15个特征以及对应的缺失率

可视化：针对Couple\_Year\_Income和Couple\_L12\_Month\_Pay\_Amount制作箱型图来判定下如何填充。

选择IOR方法筛选Couple\_Year\_Income异常值

item = 'Couple\_Year\_Income'

iqr = df[item].quantile(0.75) - df[item].quantile(0.25)

q\_abnormal\_L = df[item] < df[item].quantile(0.25) - 1.5 \* iqr

q\_abnormal\_U = df[item] > df[item].quantile(0.75) + 1.5 \* iqr

#取异常点的索引

print(item + '中有' + str(q\_abnormal\_L.sum() + q\_abnormal\_U.sum()) + '个异常值')

item\_outlier\_index = df[q\_abnormal\_L|q\_abnormal\_U].index

根据筛选出的异常值索引，删除Couple\_Year\_Income的异常值并用中位数填补缺失值。

选择IOR方法筛选Couple\_L12\_Month\_Pay\_Amount异常值

根据筛选出的异常值索引，删除Couple\_L12\_Month\_Pay\_Amount的异常值并用中位数填补缺失值

查看df中仍有少量缺失值的特征是哪些，提取出列名，封装到列表结构中，列表命名为null\_col

使用众数填充null\_col中每列缺失值，直接改变df。

从df中删除无分类意义的特征列Cust\_No。

使用factorize函数，对df剩余全部名称型特征进行标签编码，部分提示如下：

### 查看数据集剩余的名称性特征

con\_col=[]

for col in df.columns:

if df.dtypes[col] == np.object:

con\_col.append(col)

con\_col

将数据集作9:1的切分成训练集和测试集，并查看两个集中不同两类别的数量。

引入StandardScaler标准化工具库，对训练集和测试集做标准化

使用Logistic Regression对标准化后的数据进行建模

对训练集X\_train\_std，y\_train进行过采样，指定过采样比例，此处为2:1。

from imblearn import over\_sampling

print('Original dataset shape {}'.format(Counter(y\_train)))

#ratio指定过采样比例，此处为2:1

smote\_model = over\_sampling.SMOTE(random\_state=7, ratio=0.5)

X\_train\_res,y\_train\_res = smote\_model.fit\_sample(X\_train\_std,y\_train)

print('Resampled dataset shape {}'.format(Counter(y\_train\_res)))

使用Logistic Regression对过采样且标准化后的数据X\_train\_res,y\_train\_res进行建模，且使用交叉验证（5折，默认scoring）进行查看。

对上一步中的Logistic Regression模型，调节常用的C和solver两个参数，使用网格搜索+交叉验证法

查看逻辑回归模型在测试集上的效果，选择多个评估指标accuracy\_score、precision\_score 、recall\_score 、f1\_score

使用RandomForest对标准化后的数据***（注意此处不是过采样后的数据）***进行建模，设置样本权重参数class\_weight='balanced'，然后对测试集进行预测，并选择多个评估指标accuracy\_score、precision\_score 、recall\_score 、f1\_score查看预测得分。

参数调优，选择坐标下降方式对class\_weight进行搜索，部分提示如下:

param\_test0 = {'class\_weight':[{0:1,1:3},{0:1,1:5},{0:1,1:10},{0:1,1:20},'balanced']}

参数调优，选择坐标下降方式对n\_estimators进行搜索，部分提示如下:

param\_test1 = {'n\_estimators':range(10,101,10)}

根据两次参数调整后的随机森林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

## 思考题

补充缺失的代码。

# 房价预测

## 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>



Kaggle Housing Prices数据介绍

【实验环境要求】:

1、python3.7

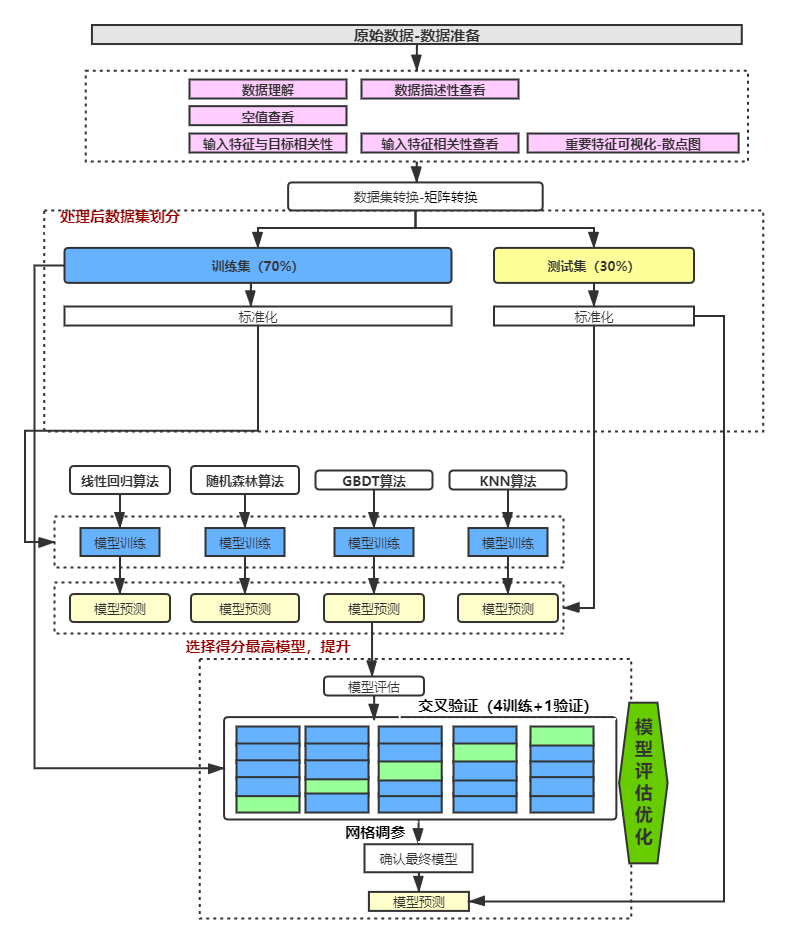
2、ModelArts平台

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/。

## 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。

### 实验整体方案



实验方案流程图

## 实验详细设计与实现

### 导入实验环境

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")

### 数据准备

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")

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。

### 数据理解

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。

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

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。

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:]

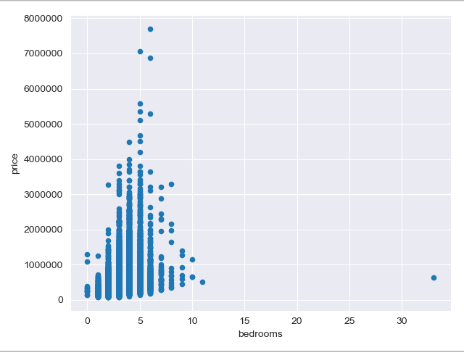
#分别分析所取的属性与价格的分布关系图

for x\_var in x\_vars:

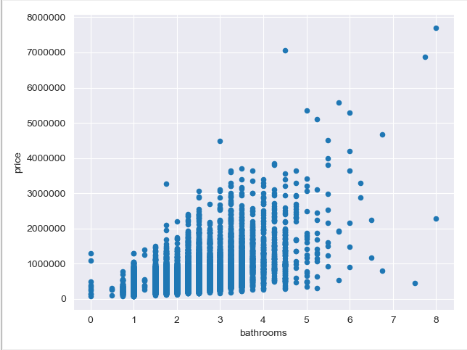
df.plot(kind='scatter',x=x\_var,y='price') #设置绘图的行和列

plt.show()

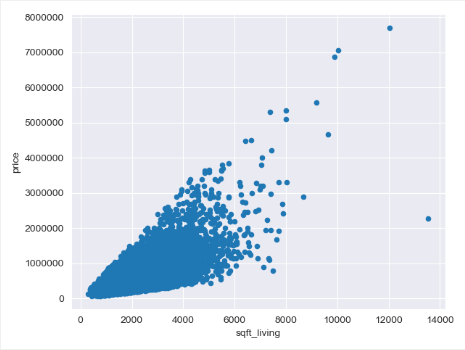
输出如下结果：



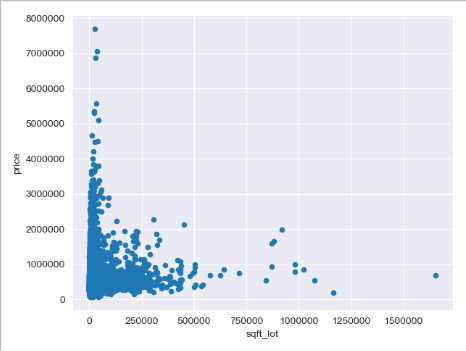
卧室数与房价散点图



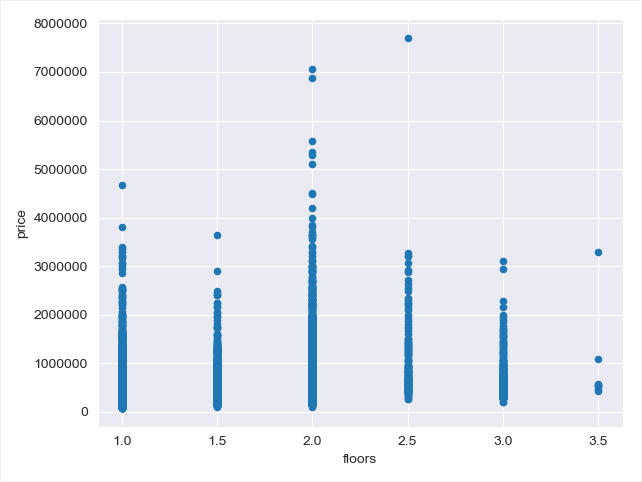
卫生间数与房价散点图



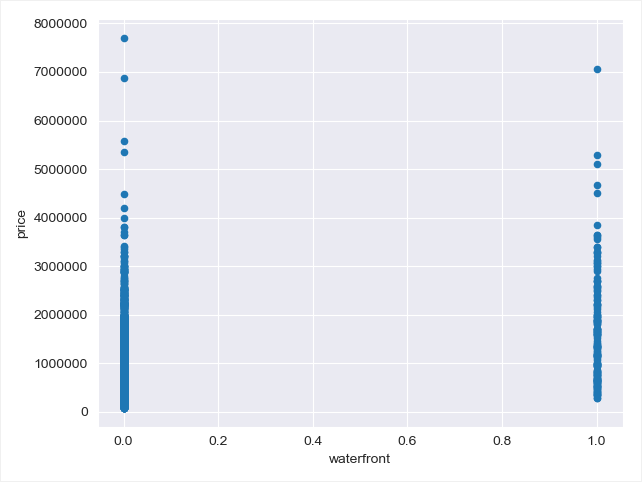
住宅面积与房价散点图



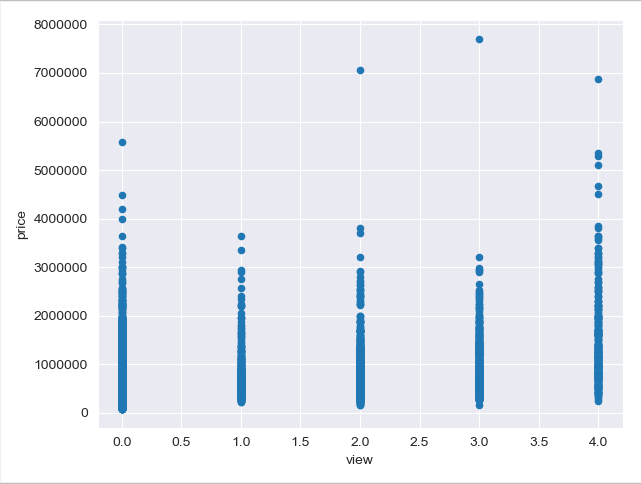
停车场面积与房价散点图



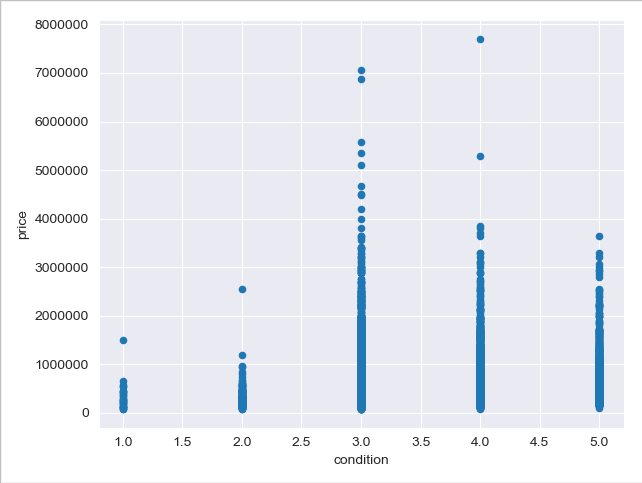
楼层数与房价散点图



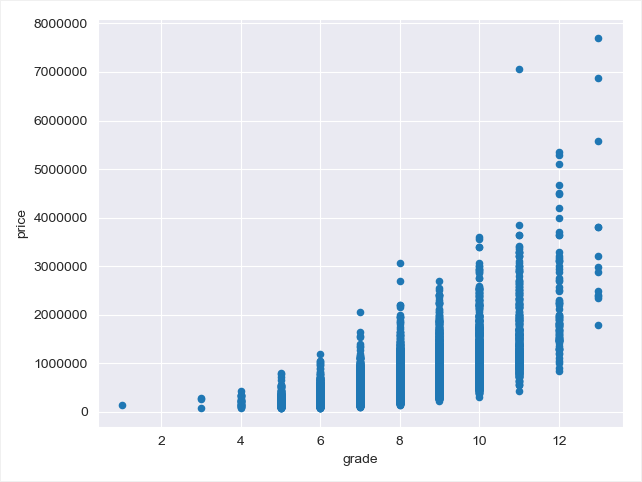
泳池与房价散点图



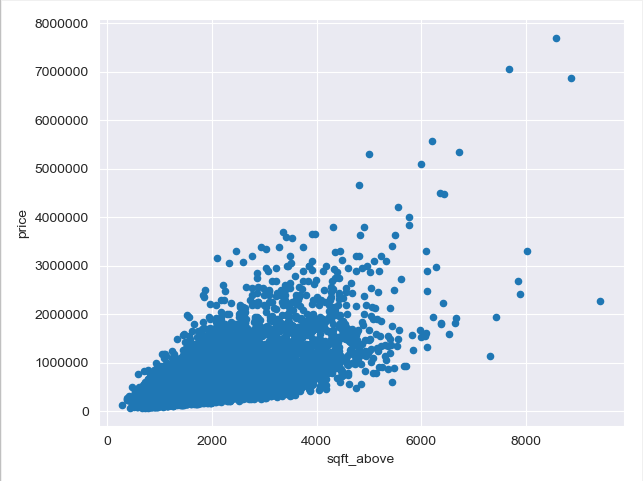
外观与房价散点图



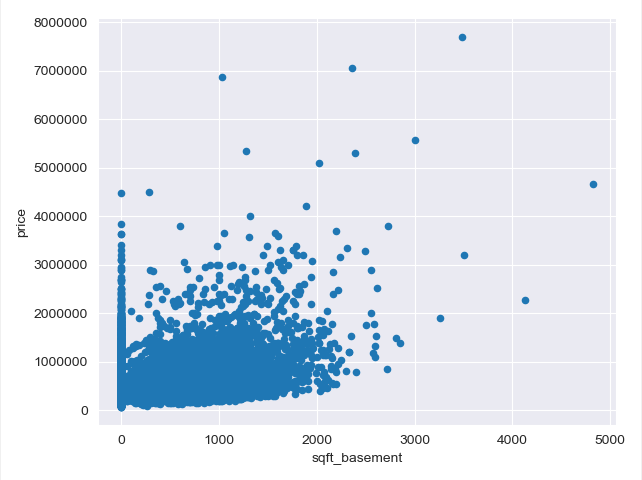
房屋状态与房价散点图



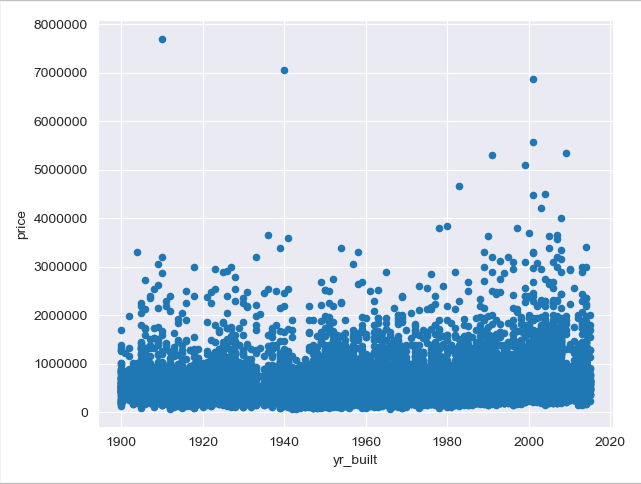
房屋等级与房价散点图



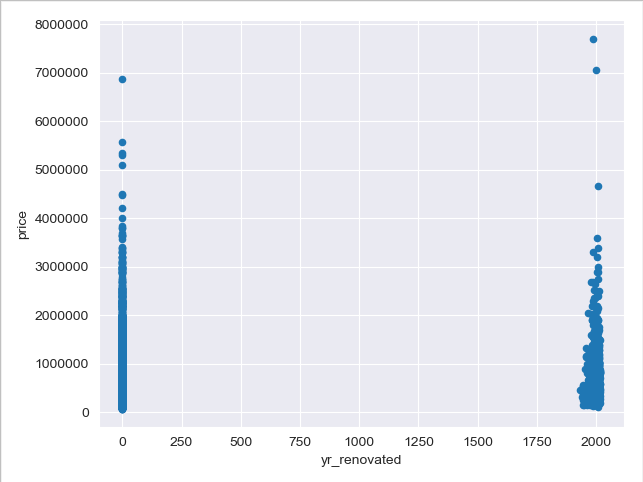
地上面价与房价散点图



地下室面积与房价散点图



修建时间与房价散点图



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。

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()

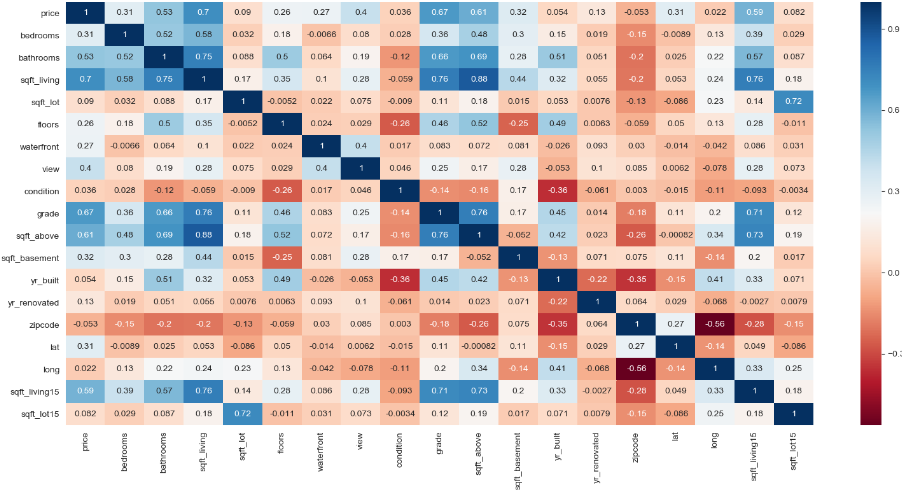
#绘制属性相关系数的热力图

plt.figure(figsize=(16,8))

sns.heatmap(corr,annot=True,cmap="RdBu")

plt.show()

输出如下结果：



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。

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

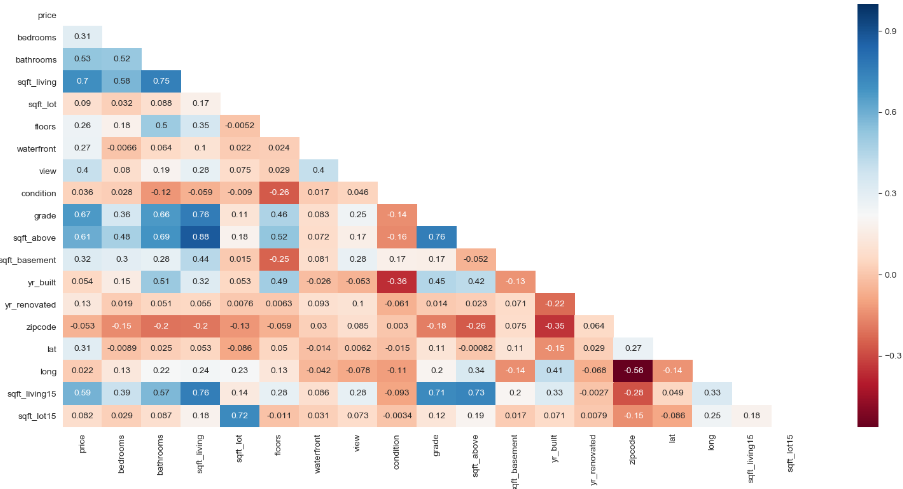
sns.set\_style(style="white")

#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：



属性相关分析下三角关系图

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

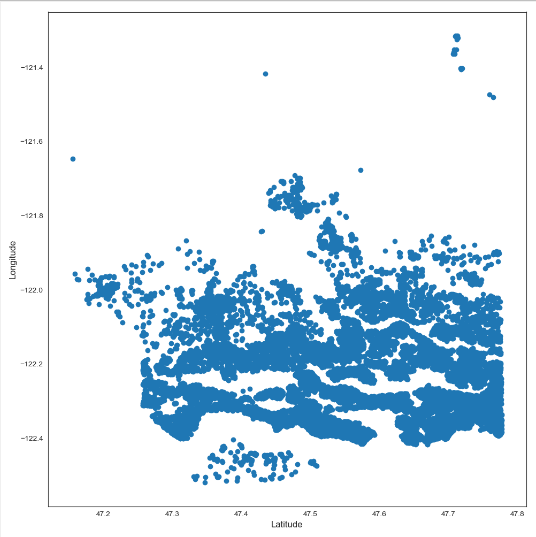
plt.scatter(df.lat, df.long)

plt.ylabel('Longitude', fontsize=12)

plt.xlabel('Latitude', fontsize=12)

plt.show()

输出如下结果：



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。

分析邮编与房价的分布关系

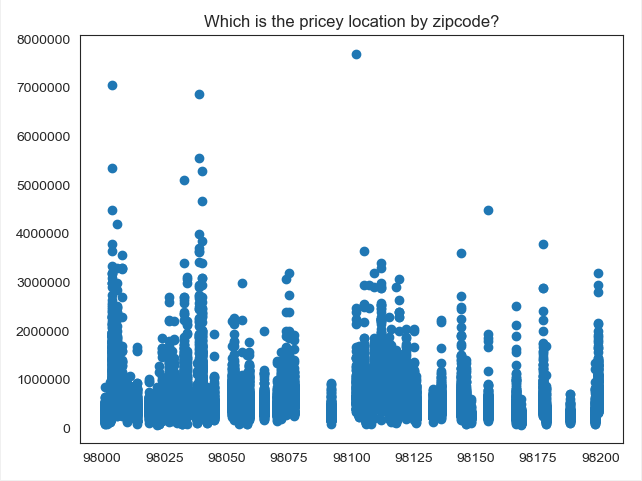
#绘制各区域具体的价格散点分布情况，了解每个区域的价格分布区间

plt.scatter(df.zipcode,df.price)

plt.title("Which is the pricey location by zipcode?")

plt.show()

输出如下结果：



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。

### 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。

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)

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)

### 数据建模分析

采用线性回归建立回归模型

#从sklearn包导入线性回归

#初始化线性回归模型

#采用线性回归进行模型训练

#获取模型预测结果

#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：

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。

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。

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。

采用最近邻算法建立回归模型

#导入最近邻回归模型

#配置最近邻回归模型参数，设置n\_neighbors=10

#采用最近邻回归模型进行训练

#采用最近邻模型进行预测

#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：

上述结果显示，最近邻回归模型预测结果相对较差。

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

#### 模型评估与优化

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。

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！

## 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？

5、根据第四章内容，完成7.3.5部分缺失的代码

## 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。