Contents

[**1 APP评分预测** 1](#_Toc59134262)

[1.1 实验介绍 1](#_Toc59134263)

[1.1.1 简介 1](#_Toc59134264)

[1.1.2 实验目的 1](#_Toc59134265)

[1.2 实验环境要求 1](#_Toc59134266)

[1.3 实验总体设计 2](#_Toc59134267)

[1.4 实验步骤 2](#_Toc59134268)

[1.4.1 数据读取 2](#_Toc59134269)

[1.4.2 线性回归 3](#_Toc59134270)

[1.4.3 **SVM回归** 4](#_Toc59134271)

[1.4.4 KNN回归 5](#_Toc59134272)

[1.4.5 思考题 7](#_Toc59134273)

[1.4.6 模型评估与选择 8](#_Toc59134274)

[1.5 实验小结 10](#_Toc59134275)

[1.6 参考答案 10](#_Toc59134276)

[**2 信用违约预测** 12](#_Toc59134277)

[2.1 实验说明 12](#_Toc59134278)

[2.2 实验建模流程要求 12](#_Toc59134279)

[2.2.1 环境要求 12](#_Toc59134280)

[2.2.2 实验实现步骤要求 12](#_Toc59134281)

[2.3 参考答案 15](#_Toc59134282)

[**3 电信用户分析** 16](#_Toc59134283)

[3.1 实验介绍 16](#_Toc59134284)

[3.1.1 简介 16](#_Toc59134285)

[3.1.2 实验目的 16](#_Toc59134286)

[3.2 实验环境要求 16](#_Toc59134287)

[3.3 实验总体设计 17](#_Toc59134288)

[3.4 实验详细设计与实现 17](#_Toc59134289)

[3.4.1 导入实验环境 17](#_Toc59134290)

[3.4.2 数据准备 17](#_Toc59134291)

[3.4.3 数据降维 18](#_Toc59134292)

[3.4.4 K-Means聚类模型训练 21](#_Toc59134293)

[3.4.5 聚类结果分析 23](#_Toc59134294)

[3.5 思考题 28](#_Toc59134295)

[3.6 实验小结 29](#_Toc59134296)

[3.7 思考题-汇总 29](#_Toc59134297)

[**4 购物篮分析** 30](#_Toc59134298)

[4.1 实验说明 30](#_Toc59134299)

[4.1.1 简介 30](#_Toc59134300)

[4.1.2 实验目的 30](#_Toc59134301)

[4.2 实验环境要求 30](#_Toc59134302)

[4.3 实验总体设计 31](#_Toc59134303)

[4.4 预备知识 31](#_Toc59134304)

[4.5 实验任务操作指导 32](#_Toc59134305)

[4.5.1 数据准备 32](#_Toc59134306)

[4.5.2 实验步骤 33](#_Toc59134307)

[4.6 实验小结 43](#_Toc59134308)

[**5 消费者聚类** 44](#_Toc59134309)

[5.1 实验说明 44](#_Toc59134310)

[5.2 根据提供的数据集，设计一个完整的聚类分析实验 44](#_Toc59134311)

[5.2.1 要求如下： 44](#_Toc59134312)

[5.3 参考答案 45](#_Toc59134313)

[**6 电影推荐实验** 46](#_Toc59134314)

[6.1 实验介绍 46](#_Toc59134315)

[6.1.1 简介 46](#_Toc59134316)

[6.1.2 实验目的 46](#_Toc59134317)

[6.2 实验环境要求 46](#_Toc59134318)

[6.3 实验总体设计 46](#_Toc59134319)

[6.4 实验过程 47](#_Toc59134320)

[6.4.1 代码和数据准备 47](#_Toc59134321)

[6.4.2 查看数据 48](#_Toc59134322)

[6.4.3 数据预处理 50](#_Toc59134323)

[6.4.4 创建相关矩阵进行电影推荐 51](#_Toc59134324)

[6.4.5 保存结果至OBS 53](#_Toc59134325)

[6.5 实验小结 54](#_Toc59134326)

[6.6 思考题参考答案 54](#_Toc59134327)

[6.6.1 查看用户的评分信息 54](#_Toc59134328)

[6.6.2 查看电影信息 55](#_Toc59134329)

[**7** 贷款产品预测 56](#_Toc59134330)

[7.1 贷款产品实验介绍 56](#_Toc59134331)

[7.2 实验总体设计 57](#_Toc59134332)

[7.2.1 实验整体方案 58](#_Toc59134333)

[7.3 实验详细设计与实现 59](#_Toc59134334)

[7.3.1 导入实验环境 59](#_Toc59134335)

[7.3.2 数据准备 59](#_Toc59134336)

[7.3.3 数据预处理 60](#_Toc59134337)

[7.3.4 建立模型 72](#_Toc59134338)

[7.3.5 模型评估与优化 78](#_Toc59134339)

[7.4 思考题 80](#_Toc59134340)

[7.5 实验小结 81](#_Toc59134341)

[7.6 思考题-汇总 81](#_Toc59134342)

[7.7 创新设计 82](#_Toc59134343)

[**8 房价预测** 83](#_Toc59134344)

[8.1 房价预测实验介绍 83](#_Toc59134345)

[8.2 实验总体设计 84](#_Toc59134346)

[8.2.1 实验整体方案 85](#_Toc59134347)

[8.3 实验详细设计与实现 86](#_Toc59134348)

[8.3.1 导入实验环境 86](#_Toc59134349)

[8.3.2 数据准备 86](#_Toc59134350)

[8.3.3 数据理解 87](#_Toc59134351)

[8.3.4 模型训练数据处理 99](#_Toc59134352)

[8.3.5 数据建模分析 100](#_Toc59134353)

[8.4 思考题 104](#_Toc59134354)

[8.5 实验小结 104](#_Toc59134355)

[8.6 思考题-汇总 105](#_Toc59134356)

### 

# **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：

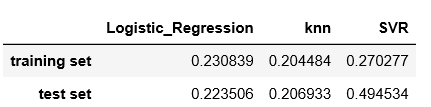
model\_mse = pd.DataFrame(data=[[linreg\_mse\_train,knn\_mse\_train,svr\_mse\_train],

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

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

model\_mse

输出：

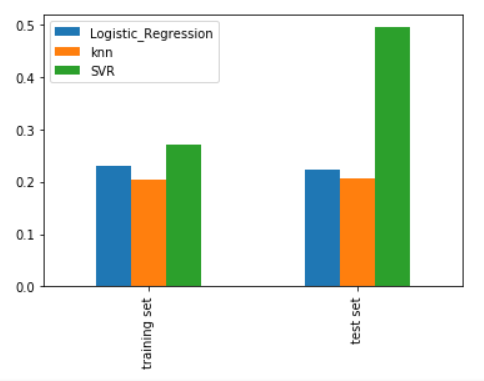


代码：

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

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

输出：



LR、KNN、SVR算法对比

### 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。

### 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

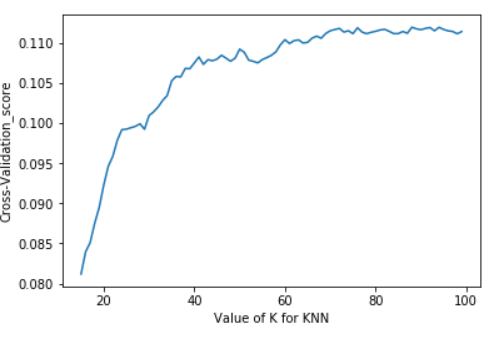
plt.plot(k\_range, k\_scores)

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

## 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。

## 参考答案

代码：

y\_train\_int = y\_train.astype(int)

y\_test\_int = y\_test.astype(int)

y\_train\_int.head()

输出：

2077 4

4387 3

8974 4

8189 4

6541 4

Name: Rating, dtype: int32

代码：

from sklearn.tree import DecisionTreeClassifier

初始化决策树模型

dtree=DecisionTreeClassifier(max\_depth=8, min\_samples\_leaf=5, random\_state=42)

#训练

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

#训练集上的MSE

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

dtree\_mse\_train = dtree.score(X\_train,y\_train\_int)

#输出测试集上的测试结果

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

dtree\_mse\_test =dtree.score(X\_test,y\_test\_int)

print("训练集MSE：", dtree\_mse\_train)

print("测试集MSE：", dtree\_mse\_test)

输出：

训练集MSE： 0.793701171875

测试集MSE： 0.7890625

# **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

## 参考答案

参考：信用违约预测.ipynb

# **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。

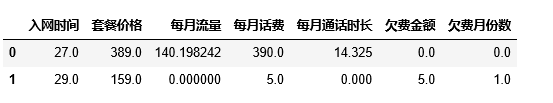
#encoding=utf8

# 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

from sklearn.decomposition import PCA

#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

pca = PCA(n\_components = 3)

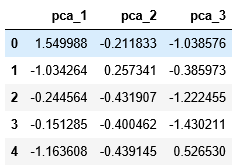
#将设置了维数的模型作用到标准化后的数据集并输出查看

X\_pca = pca.fit\_transform(X\_scaled)

X\_pca\_frame = pd.DataFrame(X\_pca,columns=['pca\_1','pca\_2','pca\_3'])

X\_pca\_frame.head(5)

将降维后的数据前5行显示出来如下：



降维后指标

降维指标与原指标关联

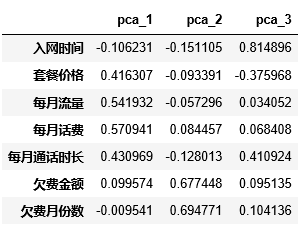
这三个指标与原始字段的系数可以通过pca的components\_属性获取。

#三个指标与原始字段的系数获取

pd.DataFrame(pca.components\_,columns = X.columns, index=['pca\_1','pca\_2','pca\_3']).T

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)

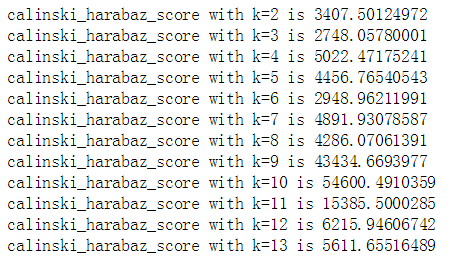
#评估不同K值聚类算法效果

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

d.update({k: score})

print('calinski\_harabasz\_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)

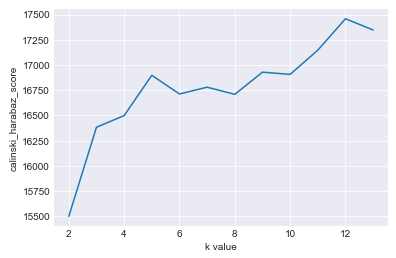
y.append(score)

plt.plot(x,y)

plt.xlabel(‘k value’)

plt.ylabel(‘calinski\_harabasz\_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

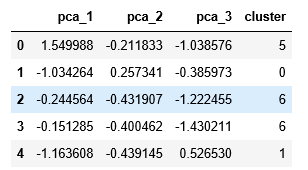
kmeans\_clustering\_labels = pd.DataFrame(est.labels\_,columns=['cluster'])

#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：

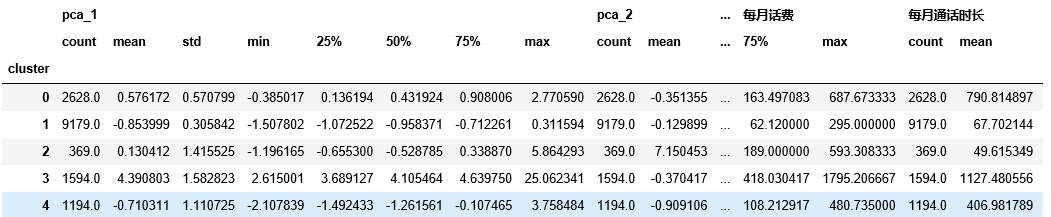
#1)主成分

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：



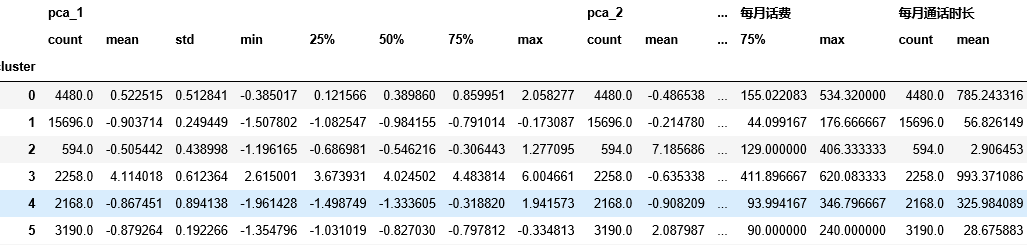
降维后数据示例

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')

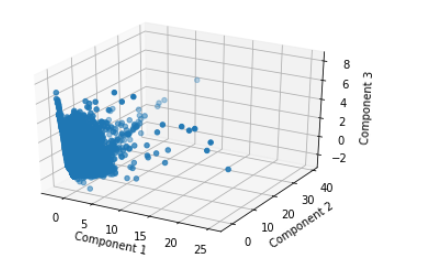
ax\_reduced\_data.scatter(X\_pca\_frame['pca\_1'].values,X\_pca\_frame['pca\_2'].values,X\_pca\_frame['pca\_3'].values)

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')

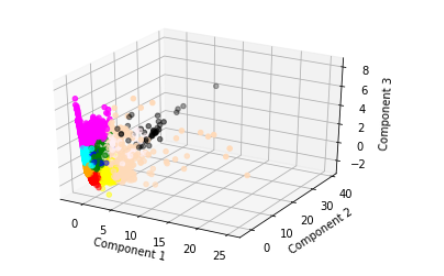
ax\_clustered\_data.scatter(X\_pca\_frame['pca\_1'].values,X\_pca\_frame['pca\_2'].values,X\_pca\_frame['pca\_3'].values,c=colors\_clustered\_data)

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')

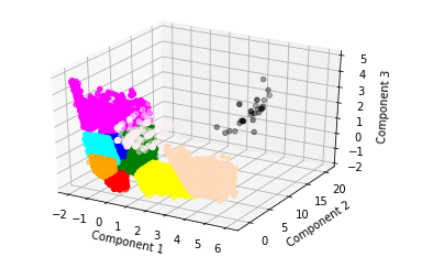
ax\_filtered\_data.scatter(result\_data['pca\_1'].values,result\_data['pca\_2'].values,result\_data['pca\_3'].values,c=colors\_filtered\_data)

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)

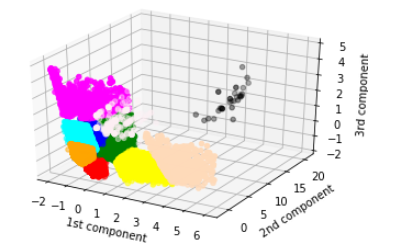
ax.set\_xlabel('1st component')

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

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

plt.show()

结果如下：



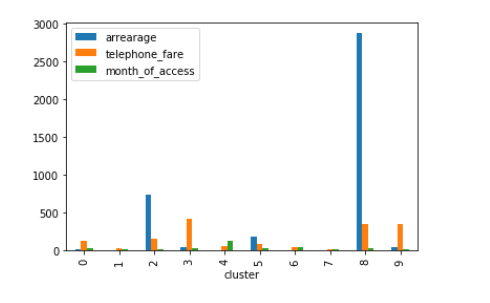
每个聚类的在pca1,pca2,pca3上的均值可视化图

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：

1. X\_full["arrearage"] = X\_full[u"欠费金额"]
2. X\_full["telephone\_fare"] = X\_full[u"每月话费"]
3. X\_full["month\_of\_access"] = X\_full[u"入网时间"]
4. %matplotlib inline
5. 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. 无监督学习适应场景？
2. K-means算法中K的确定还有哪些方法？

【答案】

## 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。

## 思考题-汇总

1. 无监督学习适应场景？
2. K-means算法中K的确定还有哪些方法？

【参考答案】

1. 无监督学习适用于数据集无标签的情况。无监督学习采用输入集，并尝试查找数据中的模式。比如，将其组织成群（聚类）或查找异常值（异常检测）。
2. 层次聚类法、稳定性方法等。此处我们实验指定k=10是否可以调整，尝试看下指定不同k值，然后观察聚类后的效果如何，选择聚类效果好的k值。（答案不唯一）

# **购物篮分析**

## 实验说明

### 简介

关联规则最初提出的动机是针对“购物篮分析”问题提出的。它的主要任务就是设法发现事物之间的内在联系。算法本质上是利用Apriori原理和其逆否命题，如果一个项集是非频繁的，那么它的所有超集也是非频繁的，频繁项集就是支持度大于最小支持度的集合。

该挖掘分析可以通过筛选出强相关性产品组合，用于相应的推荐场景等。本实验操作总体上，包括第1部分构建频繁项集和第2部分挖掘关联规则两个部分。利用Apriori算法（先验原理）对用户购物数据进行分析。

数据来源：Online Retail.xlsx

### 实验目的

通过本案例实践，掌握频繁模式挖掘的基本思路、应用和对业务数据的要求，同时结合实操加深对算法应用的理解，为后续的实际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，mlxtend使用pip安装命令如下：

pip install seaborn

pip install numpy

pip install mlxtend

pip install pandas

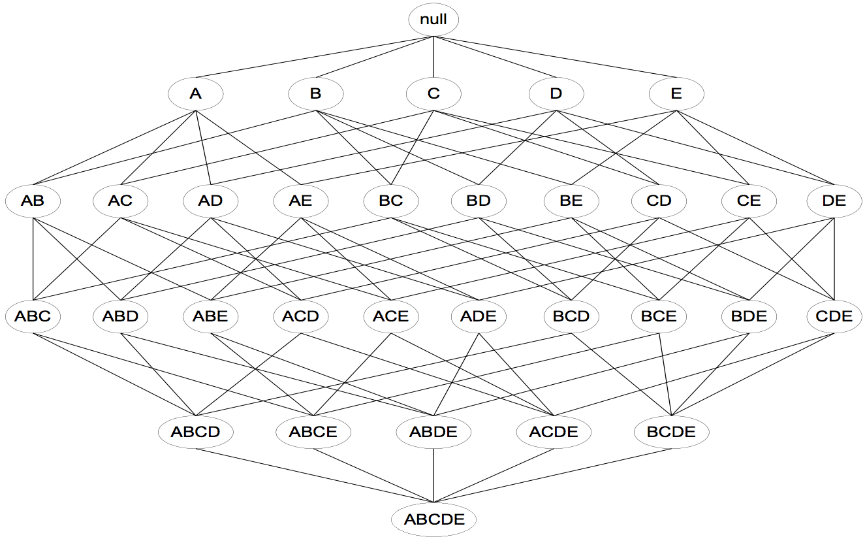
其他安装包如上进行安装。

## 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，进行基本的数据预处理操作，随后直接选择mlxtend模块中的Apriori算法进行建模分析，最终通过调整支持度和置信度，获取不同的商品组合，为套餐设计提供数据准备。

## 预备知识

我们的目标是分析每种商品的关联关系。这里用A，B，C，D，E各表示一种商品，如下项集图），在实验操作中为了描述简化，商品都用数字表示，如列表 [[1,2],[1,3,0],[0,1],[0,2],[1,2,3,0]] 就是一组由5个购物单组成的数据对象（根据集合元素知识，可知有31种集合可能），其中每个子列表代替一个购物单（如[1,3,4] 表示一笔购物同时买了商品1，3，4），自然的，当商品的种类越来越多，集合的组成数是指数级增长的，在计算处理中，工作量比较大，利用相关算法，如Apriori，帮助我们在分析过程中减少计算复杂度。如下图，常用来表示所有可能的项集，Apriori算法就是通过一种不用计算支持度，事先删除掉某些项集，减少候选项集从而达到降低计算复杂度的目的。



Apriori算法中集合关联关系

## 实验任务操作指导

### 数据准备

实验中所用的数据集来自UCI Machine Learning repository，名称为“Online Retail”。它来自 01/12/2010 - 09/12/2011 英国一家网上零售商量的共541909条交易记录。

它包含的属性特征信息解释如下：

InvoiceNo：发票号，6位连续整数，唯一对应每个发票，如果是C开头，表示该笔交易取消；

StockCode：产品代码，一个唯一对应每个商品的5位整数；

Description：商品描述，标称属性；

Quantity：每笔交易的每个产品（实验）的数量，是数值类型；

InvoiceDate：发票生成的日期，数值；

UnitPrice：英镑单位商品单价，数值；

CustomerID：用户id,唯一对应每位顾客的5位数字，标称属性；

Country：国家名称，记录每位客户声明的归属国。

### 实验步骤

导入相应模块

注意，此处引入一个由Sebastian Raschka 提供的具有Apriori算法的MLxtend库，以方便我们进一步分析数据，可以使用pip安装MLxtend，只有安装了MLxtend下面的代码才能真正运行，代码如下：

#coding: utf-8

#导入相关的包，注意mlxtend来自第三方包

读取并显示数据集

代码如下：

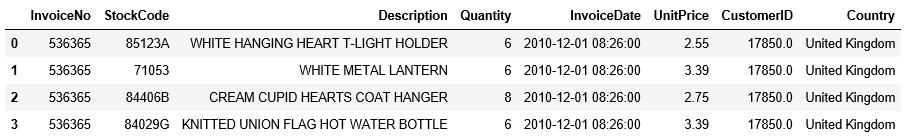
#读取数据，并显示数据前几行进行观察,利用pandas包可以直接读取excel一维表

df =\_\_\_\_\_\_\_\_\_\_ #加载数据集

df.head(10)

#数据集较大，处理会有延迟

结果如下：



数据集前几行数据

数据探索

查看数据维度特征等，代码如下：

#查看数据的特征数，各特征数据类型等信息

print("The shape of the data:{}".format(df.shape))

df.info()

结果如下：

The shape of the data:(541909, 8)

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

RangeIndex: 541909 entries, 0 to 541908

Data columns (total 8 columns):

InvoiceNo 541909 non-null object

StockCode 541909 non-null object

Description 540455 non-null object

Quantity 541909 non-null int64

InvoiceDate 541909 non-null datetime64[ns]

UnitPrice 541909 non-null float64

CustomerID 406829 non-null float64

Country 541909 non-null object

dtypes: datetime64[ns](1), float64(2), int64(1), object(4)

memory usage: 33.1+ MB

可以看到该数据集合，有54万多条记录，即购买行为8个特征属性，且均没有空值。

处理异常数据，代码如下：

#通过数据解读，了解该交易记录数据不会存在合理负值，此处对负值进行处理：全部取绝对值处理

df.Quantity = \_\_\_\_\_\_\_\_\_\_ #对Quantity求绝对值

查看每张发票购买量的分布情况，代码如下：

#查看发票购买商品的数据分布：按照每笔发票的单号聚合数据

grouped\_df =\_\_\_\_\_\_\_\_\_\_\_\_ #对'InvoiceNo'进行groupby操作，并对'Quantity'聚集求和

grouped\_df.head()

grouped\_df.plot.hist(bins=1000)

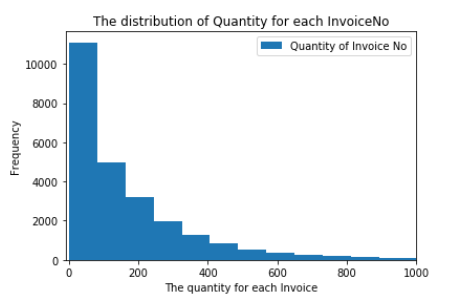
plt.xlim([-10,1000])

plt.xlabel('The quantity for each Invoice')

plt.legend(['Quantity of Invoice No'])

plt.title("The distribution of Quantity for each InvoiceNo")

可以看到，大量购物交易购买的商品总数主要集中在100件左右，随着购买数量的增加，交易数量迅速递减。



每笔购买商品数量分布

查看全部商品中最畅销的20种物品

在正式使用算法建模之前，可以从多个角度尝试去了解商品的数据特性，代码如下：

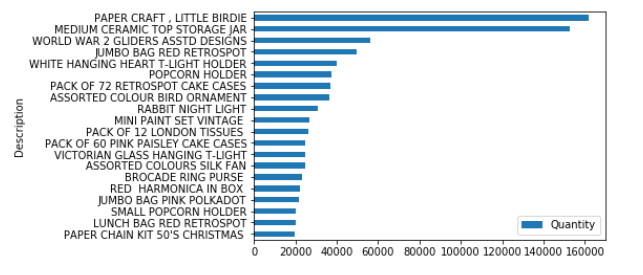
#再查看交易清单里最畅销的前20款商品

item\_df =\_\_\_\_\_\_\_ #对'StockCode','Description'进行聚合操作，并对'Quantity'进行求和聚合

item\_df.set\_index('Description').sort\_values(by = 'Quantity',ascending = False)[0:20][::-1].plot.barh()

#item\_df.Description.plot.barh()

结果如下：



最畅销的20种物品

数据预处理

去除空值，代码如下：

进入数据预处理阶段：清除商品描述中的空格和移除无效的发票号的记录

df.Description = df.Description.str.strip()

df.\_\_\_\_\_\_\_\_\_\_\_ #将所有含有nan项的row删除，且原数组直接就被替换

#输出清除无效发票后剩余数据大小

print('The shape of dropna dataset:{}'.format(df.shape))

结果如下：

The shape of dropna dataset:(541909, 8)

去除发票信息中为“C”（支票支付）的数据，代码如下：

#移除数据中，发票号码开头为C的记录

df['InvoiceNo'] = df['InvoiceNo'].astype('str')

df = df[~df['InvoiceNo'].str.contains('C')]

df.shape

结果如下：

(532621, 8)

读取法国部分的数据并按物品分组计算，代码如下：

鉴于Python的计算效率和Aprior算法的计算逻辑，此处为了将数据集的数据进一步缩小，只筛选出法国客户的消费记录进行下一步分析，即接下来我们的分析范围都是该网站此段时间法国消费者购买商品的相关关系，代码如下：

#查找国家标识=法国的记录，并且按照发票号和商品描述进行聚合

grouped\_df =\_\_\_\_\_\_\_\_\_\_

#可以得到一个多索引的数据框，类似数据透视效果

grouped\_df.head()

结果如下：

InvoiceNo Description

536370 ALARM CLOCK BAKELIKE GREEN 12

ALARM CLOCK BAKELIKE PINK 24

ALARM CLOCK BAKELIKE RED 24

CHARLOTTE BAG DOLLY GIRL DESIGN 20

CIRCUS PARADE LUNCH BOX 24

Name: Quantity, dtype: int64

对分组数据的行旋转为列，代码如下：

#将上面多索引的数据框，通过转置拆分成单索引数据表形式

unstack\_df =\_\_\_\_\_\_\_\_\_\_\_\_

unstack\_df.head(10)

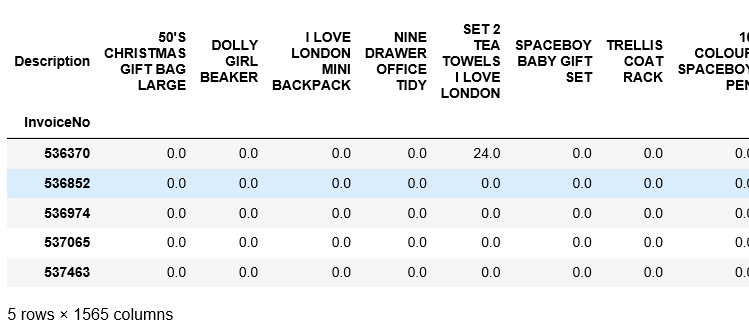
#将所有的空值填充为0

fillna\_df = unstack\_df.reset\_index().fillna(0)

#设置InvoiceNO列为索引

basket\_df = fillna\_df.set\_index('InvoiceNo')

basket\_df.head(5)



转换后的单索引数据表

数据转换为0-1形式，代码如下：

#定量特征二值化:大于阈值1的设置为1 ，小于等于0的设置为0

def encode\_units(x):

if x <= 0:

return 0

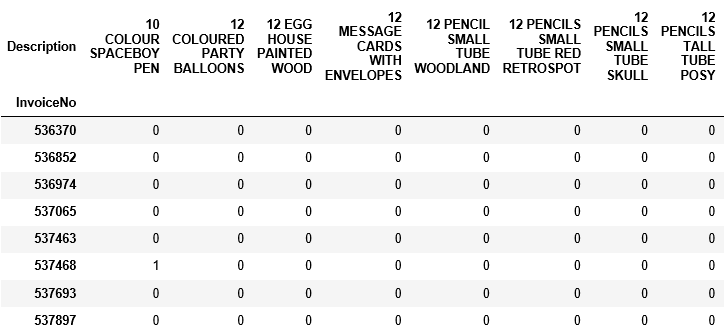
if x >= 1:

return 1

basket\_sets = basket\_df.applymap(encode\_units)

basket\_sets.head(10)

结果如下：



0-1转换后的数据集

随后，在二值化的基础上移除掉分析中不需要关注的列：邮费列，这一步非常重要，因为邮费作为商品列表一项非常频繁地出现，会干扰正常商品的频繁项组合结果。代码如下：

basket\_sets.drop('POSTAGE', inplace=True, axis=1)#axis=0表述列 ，axis=1表述行

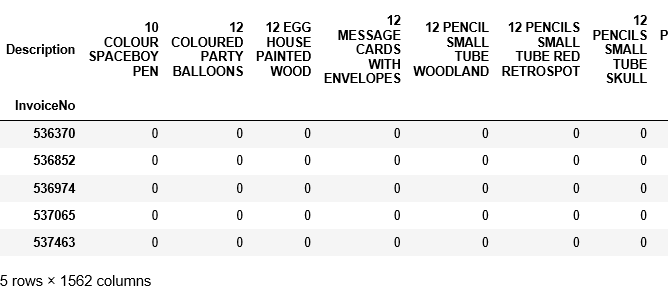
print('The shape of basket set:{}'.format(basket\_sets.shape))

basket\_sets.head()

结果如下：

The shape of basket set:(392, 1562)

以及



移除邮费列后数据集

创建频繁集

现在有了合适的数据结构，我们可以在此基础上生成频繁的项目集，代码如下：

#此处先设置，最小的支持度为 0.08 为了先尽量保留多的商品项，最小支持度设置的较小

frequent\_itemsets = \_\_\_\_\_\_\_\_\_\_\_\_\_\_

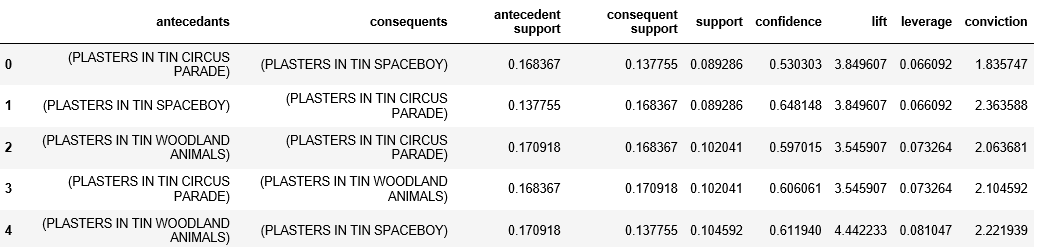
创建关联规则

最后一步，是生成根据相应的支持度、置信度和提升度生成的规则，代码如下:

rules = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

rules.head(5)

结果如下：



数据集支持度、置信度和提升度

结果可视化

定义可视化函数

注意其中导入的NetworkX是一款Python的软件包，用于创造、操作复杂网络，以及学习复杂网络的结构等。代码如下：

import networkx as nx

import numpy as np

def draw\_graph(rules, rules\_to\_show):

G1 = nx.DiGraph()

color\_map=[]

N = 50

colors = np.random.rand(N)

strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10', 'R11']

for i in range (rules\_to\_show):

G1.add\_nodes\_from(["R"+str(i)])

for a in rules.iloc[i]['antecedants']:

G1.add\_nodes\_from([a])

G1.add\_edge(a, "R"+str(i), color=colors[i] , weight = 2)

for c in rules.iloc[i]['consequents']:

G1.add\_nodes\_from([c])

G1.add\_edge("R"+str(i), c, color=colors[i], weight=2)

for node in G1:

found\_a\_string = False

for item in strs:

if node==item:

found\_a\_string = True

if found\_a\_string:

color\_map.append('yellow')

else:

color\_map.append('green')

edges = G1.edges()

colors = [G1[u][v]['color'] for u,v in edges]

weights = [G1[u][v]['weight'] for u,v in edges]

pos = nx.spring\_layout(G1, k=16, scale=1)

nx.draw(G1, pos, edges=edges, node\_color = color\_map, edge\_color=colors,

width=weights, font\_size=16, with\_labels=False)

for p in pos: #raise text positions

pos[p][1] += 0.07

nx.draw\_networkx\_labels(G1, pos)

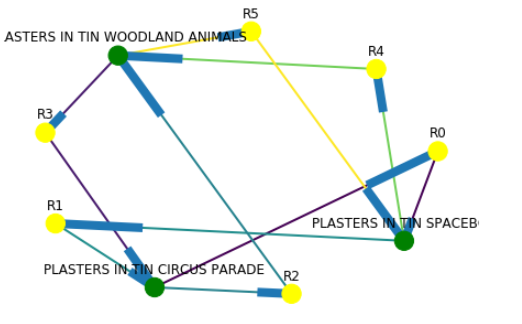
plt.show()

展示前6个关联规则R0，R1，…，R5，代码如下：

draw\_graph(rules,6)

#plt.show()

结果如下：



前6项关联组合

可以看到，出现频次最多的商品组合是由3种商品：PLASTERS IN TIN SPACEBOY、PLASTERS IN TIN WOODLAND ANIMALS、PLASTERS IN TIN CIRCUS PARADE的各种组合，其中，最明显的是， PLASTERS IN TIN CIRCUS PARADE总会带来PLASTERS IN TIN SPACEBOY的销售。

统计上述3种商品销售次数，代码如下：

print(basket\_sets['PLASTERS IN TIN CIRCUS PARADE'].sum())

print(basket\_sets['PLASTERS IN TIN SPACEBOY'].sum())

print(basket\_sets['PLASTERS IN TIN WOODLAND ANIMALS'].sum())

结果如下：

66

54

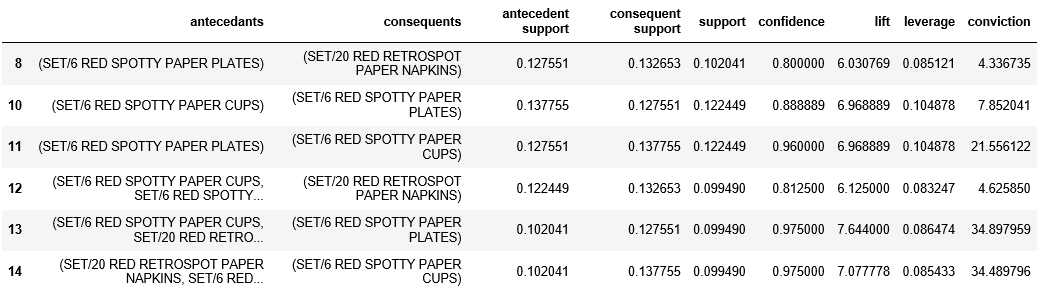
67

调整分析维度

限制置信度大于80%，查看此时对应的关联规则，代码如下：

rules[ (rules['confidence'] >= 0.8) ]

结果如下：

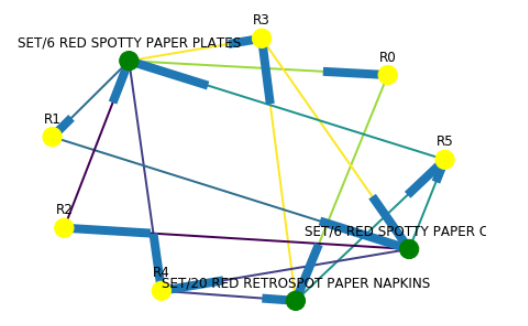


置信度限制后关联规则结果

制作可视化图

draw\_graph(rules[ (rules['confidence'] >= 0.8) ],6)

结果如下：



调整置信度约束后的前6个商品组合关系

通过上图规则可以看出，在高于总体概率水平的约束下，最频繁出现的商品组合是：SET/6 RED SPOTTY PAPER PLATES、SET/6 RED SPOTTY PAPER CUPS、SET/20 RED RETROSPOT PAPER NAPKINS的各种组合。

统计3个商品销售次数，代码如下：

basket\_sets['SET/6 RED SPOTTY PAPER PLATES'].sum()

basket\_sets['SET/6 RED SPOTTY PAPER CUPS'].sum()

basket\_sets['SET/20 RED RETROSPOT PAPER NAPKINS'].sum()

结果如下：

50

54

52

类似地，可以再查看德国客户的消费商品的关联性

basket2 = (df[df['Country'] =="Germany"]

.groupby(['InvoiceNo', 'Description'])['Quantity']

.sum().unstack().reset\_index().fillna(0)

.set\_index('InvoiceNo'))

basket\_sets2 = basket2.applymap(encode\_units)

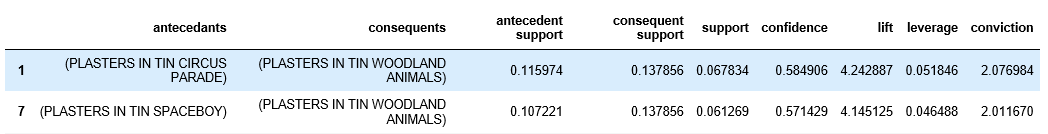
basket\_sets2.drop('POSTAGE', inplace=True, axis=1)

frequent\_itemsets2 = apriori(basket\_sets2, min\_support=0.05, use\_colnames=True)

rules2 = association\_rules(frequent\_itemsets2, metric="lift", min\_threshold=1)

rules2[ (rules2['lift'] >= 4) & (rules2['confidence'] >= 0.5)]

结果如下：

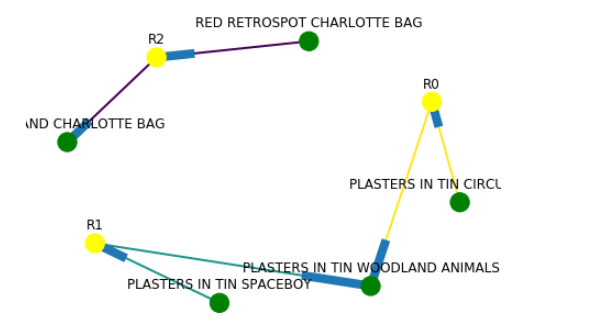


按约束规则得前3个关联规则

展示前3个规则

draw\_graph(rules2[ (rules2['lift'] >= 4) & (rules2['confidence'] >= 0.5)],3)

结果如下：



按约束规则得到的前3个关联规则

## 实验小结

该实验主要是利用Apriori算法对用户购物数据 “Online Retail” 进行分析，通过频繁项集构建和关联规则挖掘，筛选出强相关性产品组合，用于相应的推荐场景等。

# **消费者聚类**

## 实验说明

在本案例中，使用人工智能技术的聚类算法对超市购物中心客户的一些基本数据进行建模分析，把客户分成不同的群体，供营销团队参考并相应地制定营销策略。

我们使用的数据集是超市用户会员卡的基本数据以及根据购物行为得出的消费指数，总共有5个字段，解释如下：

CustomerID：客户ID

Gender：性别

Age：年龄

Annual Income (k$)：年收入

Spending Score (1-100)：消费指数

## 根据提供的数据集，设计一个完整的聚类分析实验

### 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；

结合不同的绘图方式，如柱状图、散点图、盒图等分析不同属性字段的统计分布规律；

对不同聚类取值（k）效果进行评估，获取较好的聚类个数；

采用最佳聚类值进行聚类，同时结合散点分布图对聚类效果进行可视化展示。

## 参考答案

参考“消费者聚类.ipynb”

# **电影推荐实验**

## 实验介绍

### 简介

本实验用协同过滤技术分析用户对电影的评分数据，并基于这个数据建立一个推荐系统，根据用户输入的一部感兴趣的电影，为其推荐其他可能感兴趣的电影。此案例中，我们使用的数据集是用户对电影的评分数据，包含用户数据、评分数据、电影数据。

### 实验目的

掌握协同过滤算法的基本原理。

掌握推荐系统的整体流程。

## 实验环境要求

华为云ModelArts

## 实验总体设计

## 实验过程

### 代码和数据准备

案例配置信息填写

案例中需要将运行结果上传至OBS中，因此，在实验之前需创建自己的obs桶，并且桶所在区域与所创建的notebook属于一个区域。我们需要设置以下相关参数（使用自己真实的桶名和唯一ID替换掉\*号）：

BUCKET\_NAME ： 自己的OBS桶名

UNIQUE\_ID : 唯一ID，填写自己的学号或者IAM子账号名称相关代码、数据和模型都已准备好存放在OBS中，执行下面一段代码即可将其拷贝到Notebook中。

BUCKET\_NAME = '\*\*\*\*\*\*\*\*\*\*\*'

#UNIQUE\_ID = '\*\*\*\*\*\*\*\*\*\*\*'

#OBS\_BASE\_PATH = BUCKET\_NAME + '/' + UNIQUE\_ID

OBS\_BASE\_PATH = BUCKET\_NAME

初始化ModelArts SDK

from modelarts.session import Session

session = Session()

下载源代码和数据

这一步准备案例所需的源代码和数据，相关资源已经保存在OBS中，我们通过ModelArts SDK将资源下载到本地，并解压到当前目录下。解压后，当前目录包含ml-100k目录，存有数据集。

session.download\_data(bucket\_path="ai-course-common-20-bj4/movie\_recommendation/movie\_recommendation.tar.gz", path="./movie\_recommendation.tar.gz")

# 使用tar命令解压资源包

!tar xf movie\_recommendation.tar.gz

输出结果：



导入基本工具库

执行下面方框中的这段代码，可以导入本次实验中使用的Python开发基本工具库。numpy是数据分处理工具,pandas是文件读取和数据处理工具，scipy是一个科学计算库，这里导入了cosine, correlation两种距离计算方法。此段代码只是引入Python包，无回显（代码执行输出）。

# import same usefull libraries

import numpy as np

import pandas as pd

from sklearn.metrics import pairwise\_distances

from scipy.spatial.distance import cosine, correlation

### 查看数据

使用pandas库导入用户的个人信息

用户数据的字段描述如下：

user\_id：用户ID

age：用户年龄

sex：性别

occupation：职业

zip\_code：邮编

评分数据的字段描述如下：

user\_id：用户ID

movide\_id：电影ID

rating：评分

unix\_tiemstamp：评分时间

电影数据的字段描述如下：

user\_id：用户ID

movide\_id：电影ID

rating：评分

unix\_tiemstamp：评分时间

查看用户信息

（1）定义用户信息

# 用户信息

users\_cols = ['user\_id', 'age', 'sex', 'occupation', 'zip\_code']

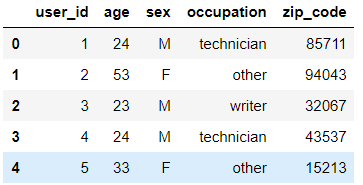
users = pd.read\_csv('./ml-100k/u.user', sep='|', names=users\_cols, parse\_dates=True)

（2）打印用户信息

打印前5个用户的个人信息，可以看到用户个人信息包含用户ID（user\_id）、年龄(age)、性别(sex)、职业(occupation)、邮编(zip\_code)。

users.head()

输出结果：



前5个用户的个人信息

（3）打印数据表格的大小，可以看到这是一个 943x5的矩阵， 其中943代表有943个用户，5代表每个用户有5项信息。

users.shape

输出结果：



查看用户的评分信息

参照查看用户信息的实现，请自行编码实现以下步骤：

（1）使用Pandas库导入用户的评分信息。

（2）打印前5个评分信息，可以看到评分信息包含用户ID（user\_id）、电影ID(movide\_id)、评分(rating)、评分时间(unix\_tiemstamp)。

（3）打印数据表格的大小，可以看到这是一个 10000x4的矩阵， 其中10000代表有10000条评论，4代表每个评论有5项信息。

查看电影信息

参照查看用户信息的实现，请自行编码实现以下步骤：

（1）使用pandas库导入电影的信息。

（2）打印前5个电影信息，可以看到电影信息包含电影ID(movide\_id)、电影名称(title)、发布时间(release\_date)、视频发布时间(video\_release\_date)、评论网站URL链接(imdb\_url)。

（3）打印数据表格的大小，可以看到这是一个 1682x5的矩阵， 其中1682代表有1682部电影，5代表每部电影有5项信息。

### 数据预处理

数据合并

（1）把电影数据表、评论数据表、用户信息数据表进行合并，最后得到一张数据信息总表。

# Merging movie data with their ratings

movie\_ratings = pd.merge(movies, ratings)

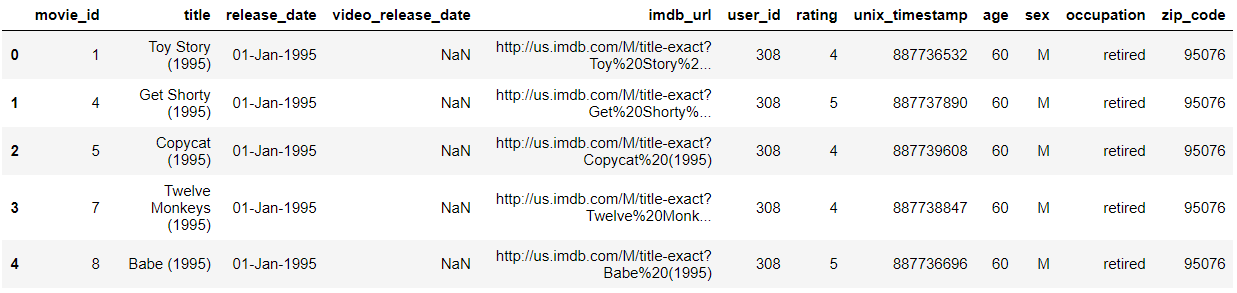
# merging movie\_ratings data with the User's dataframe

df = pd.merge(movie\_ratings, users)

（2）查看数据信息总表

df.head()

输出结果：



前5个用户的数据信息总表

（3）打印数据总表的大小，可以看到这是一个 10000x12的矩阵， 其中10000代表有10000条评论，12代表每条评论有12项属性，包括电影ID，电影信息，用户ID，评分，用户信息等。

df.shape

输出结果：



数据清洗

（1）去除一些无效或不需要的信息，比如video\_release\_date、imdb\_url、unix\_timestamp。

# pre-processing

# dropping colums that aren't needed

df.drop(df.columns[[3, 4, 7]], axis=1, inplace=True)

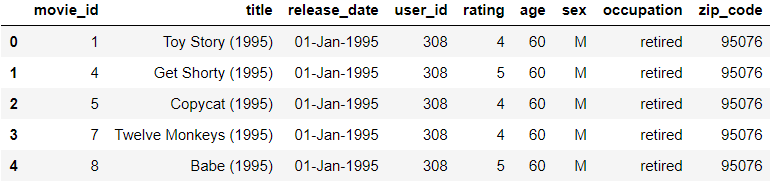
ratings.drop("unix\_timestamp", inplace=True, axis=1)

movies.drop(movies.columns[[3, 4]], inplace=True, axis=1)

（2）查看新的数据信息总表。

df.head()

输出结果：



前5个用户的新数据信息总表

### 创建相关矩阵进行电影推荐

创建用户-电影评分矩阵

（1）根据评分数据表(ratings),创建用户-电影评分矩阵。

# Pivot Table(This creates a matrix of users and movie\_ratings)

ratings\_matrix = ratings.pivot\_table(index=['movie\_id'], columns=['user\_id'], values='rating').reset\_index(drop=True)

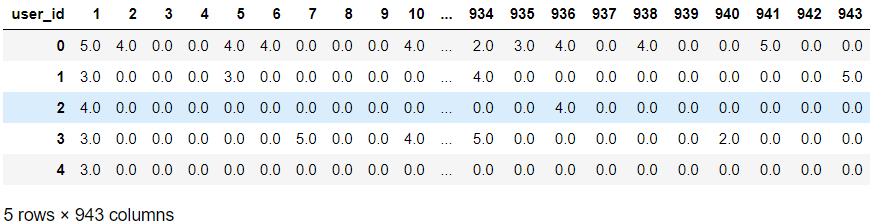
ratings\_matrix.fillna(0, inplace=True)

cmu = ratings\_matrix

（2）查看用户-电影评分矩阵，其中每一行代表每部电影来自所有用户的评分；每一列代表每个用户对所有电影的评分。

cmu.head()

输出结果：



用户-电影评分矩阵

打印数据总表的大小，可以看到这是一个 1682x943的矩阵， 其中1682代表有1682部电影，943代表有943个用户。

创建电影的相似矩阵

（1）根据943位用户对每部电影的评分，创建1682部电影的相似矩阵，矩阵大小为1682x1682的形状。

# Cosine Similarity(Creates a cosine matrix of similaraties ..... which is the pairwise distances

# between two items )

​

movie\_similarity = 1 - pairwise\_distances(ratings\_matrix.values, metric="cosine")

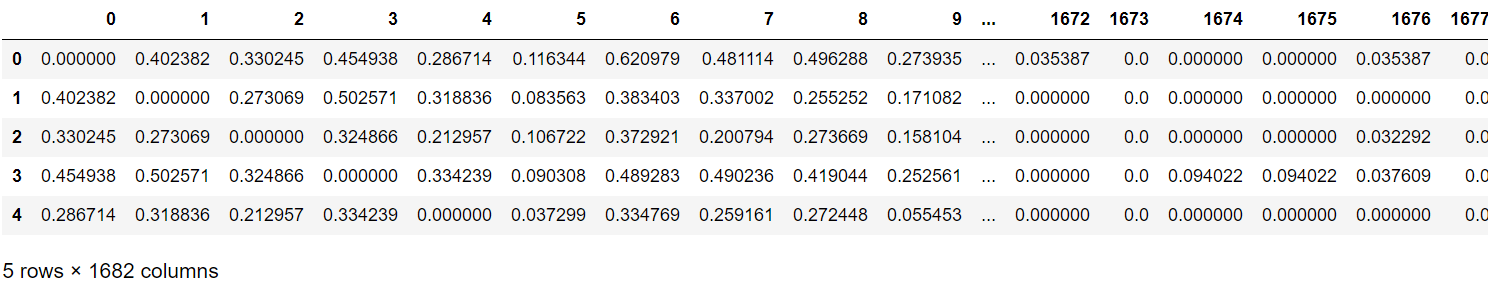
np.fill\_diagonal(movie\_similarity, 0)

ratings\_matrix = pd.DataFrame(movie\_similarity)

（2）查看电影相似矩阵，以第三行，第二列为例，数值为0.273069，这个值代表第二部电影与第三部电影的相似度。

ratings\_matrix.head()

输出结果：



电影相似矩阵

（3）查看形状

ratings\_matrix.shape

输出结果：



根据电影的相似矩阵，推荐电影

（1）当用户查看了 Copycat (1995)，那么根据电影的相似矩阵，推荐与 Copycat (1995) 近似分数比较高的电影。具体如下：根据电影名 Copycat (1995)， 查询电影信息表 (movies)中的index序号。

# user\_inp=input('Enter the reference movie title based on which recommendations are to be made: ')

user\_inp = "Copycat (1995)"

inp = movies[movies['title'] == user\_inp].index.tolist()

inp = inp[0]

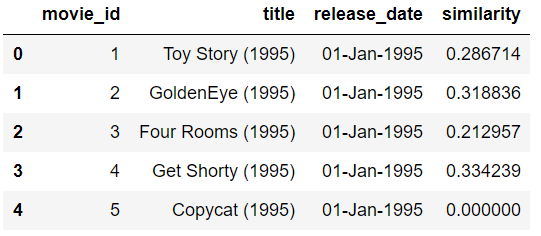
（2）根据电影index序号，去查电影的相似矩阵，得到1682部电影的相似值，并打印表格中的5部电影的相似值。

movies['similarity'] = ratings\_matrix.iloc[inp]

movies.columns = ['movie\_id', 'title', 'release\_date', 'similarity']

movies.head(5)

输出结果：



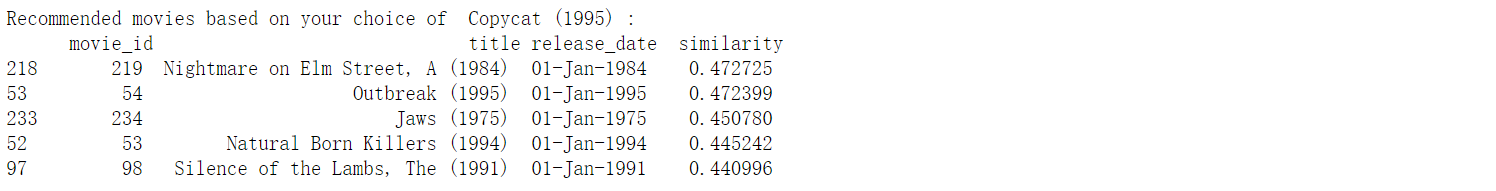
5部电影的相似值

（3）把相似值进行排序，并打印最相似的5部电影

recommended\_movies = movies.sort\_values(["similarity"], ascending=False)[1:6]

​print("Recommended movies based on your choice of ", user\_inp, ": \n", recommended\_movies)

输出结果：



最相似的5部电影

### 保存结果至OBS

我们将推荐的前5部电影的信息保存到文本文件中，并上传到OBS，以便以后查看。

写入本地文件

将电影的信息写入到文本文件中。会打印成功保存的信息。

import os

​if not os.path.exists('results'):

os.mkdir('results') # 创建本地保存路径

​with open('./results/recommended\_movies.txt', 'w') as f:

f.write(str(recommended\_movies)) # 写入本地文本文件

print('Successfully saved!')

输出结果：

Successfully saved!

上传文件至OBS

使用ModelArts SDK上传本地文件至OBS。可以看到上传成功的日志。

session.upload\_data(bucket\_path=OBS\_BASE\_PATH + '/movie\_recommendation/results/', path='./results/recommended\_movies.txt')

输出结果：

Successfully upload file ./results/recommend\_movies.txt to OBS professional-construction/movie\_recommendtion/results

注意：该案例所在的OBS存储路径下，results目录下，有模型文件recommended\_movies.txt。

## 实验小结

本实验基于ModelArts平台使用协同过滤算法中基于物品的协同过滤实现了电影推荐案例。

## 思考题参考答案

### 查看用户的评分信息

（1）使用Pandas库导入用户的评分信息

# Ratings

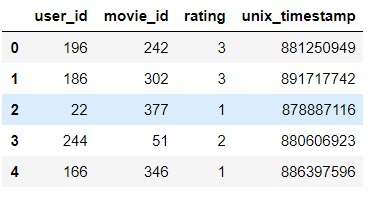
rating\_cols = ['user\_id', 'movie\_id', 'rating', 'unix\_timestamp']

ratings = pd.read\_csv('./ml-100k/u.data', sep='\t', names=rating\_cols)

（2）打印前5个评分信息，可以看到评分信息包含用户ID（user\_id）、电影ID(movide\_id)、评分(rating)、评分时间(unix\_tiemstamp)

ratings.head()

输出结果：



前5个用户的评分信息

（3）打印数据表格的大小，可以看到这是一个 10000x4的矩阵， 其中10000代表有10000条评论，4代表每个评论有5项信息

ratings.shape

输出结果：



### 查看电影信息

（1）使用pandas库导入电影的信息

# Movies

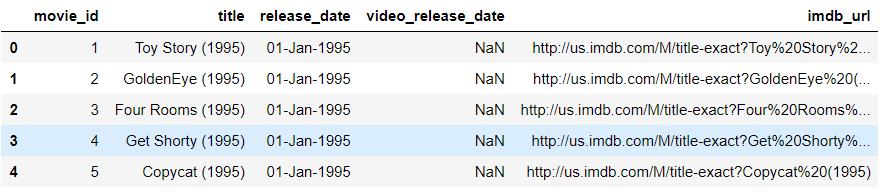
movie\_cols = ['movie\_id', 'title', 'release\_date', 'video\_release\_date', 'imdb\_url']

movies = pd.read\_csv('./ml-100k/u.item', sep='|', names=movie\_cols, usecols=range(5), encoding='latin-1')

（2）打印前5个电影信息，可以看到电影信息包含电影ID(movide\_id)、电影名称(title)、发布时间(release\_date)、视频发布时间(video\_release\_date)、评论网站URL链接(imdb\_url)

movies.head()

输出结果：



前5部电影信息

（3）打印数据表格的大小，可以看到这是一个 1682x5的矩阵， 其中1682代表有1682部电影，5代表每部电影有5项信息

movies.shape

输出结果：

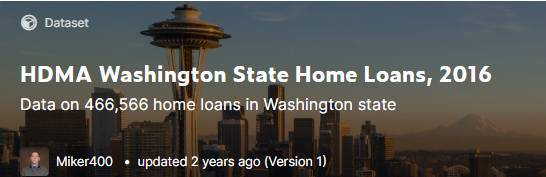


# 贷款产品预测

## 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>



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

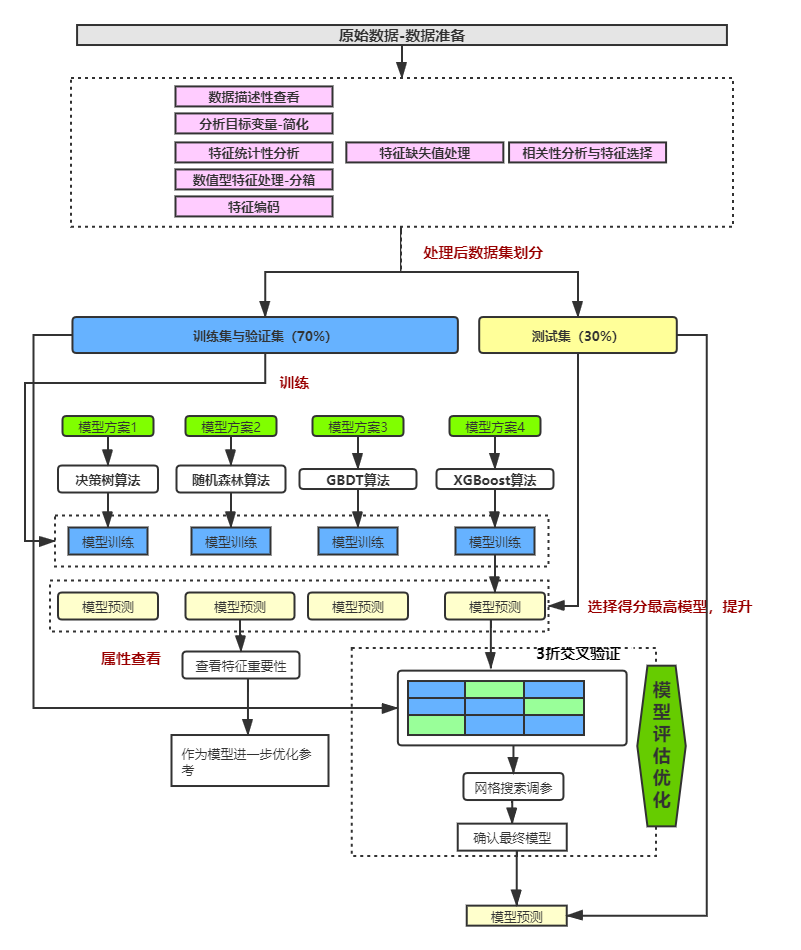
pip install numpy

其他安装包如上进行安装。

## 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。

* + 1. 实验整体方案



1. 实验方案流程图

## 实验详细设计与实现

* + 1. 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，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')

* + 1. 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。

hdma\_raw = pd.read\_csv(r"Washington\_State\_HDMA-2016.csv",low\_memory=False)

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

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。

查看数据尺寸

hdma.shape #获取数据规模

输出结果如下：

(466566, 47)

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。

* + 1. 数据预处理

#### 分析目标变量，建模问题简化

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）。

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']]

删除原先的目标列

del hdma['action\_taken\_name']

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。

查看目标变量的分布情况，并绘制出直方图：

print(hdma['loan\_status'].value\_counts())

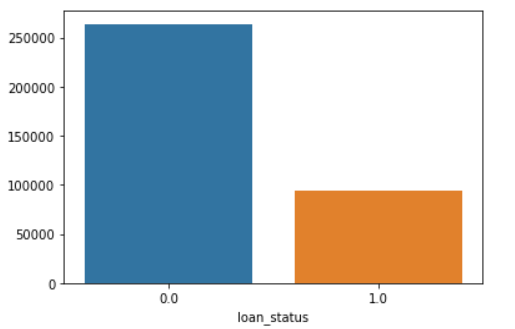
sns.countplot('loan\_status', data=hdma)

输出结果如下：

0.0 263712

1.0 94140

Name: loan\_status, dtype: int64



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）。

#### 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。

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）。

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：

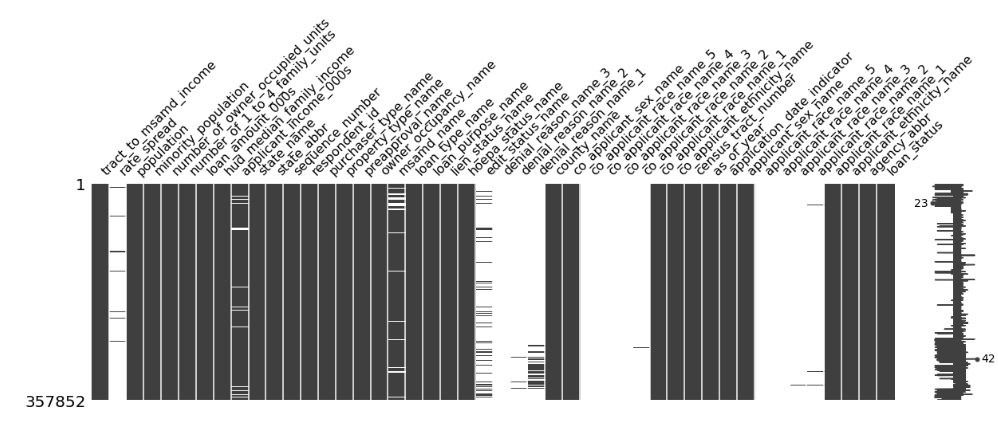
#### 处理缺失数据

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：



1. 缺失值分布

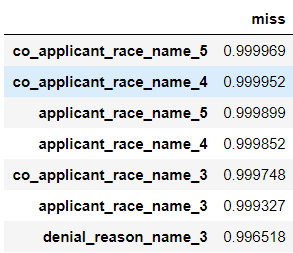
可以看出有几个列缺失非常严重，考虑基于一定的规则进行处理。

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：



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)

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。

删除缺失值占比超过80%的特征

hdma=hdma.drop(columns = del\_col)

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)

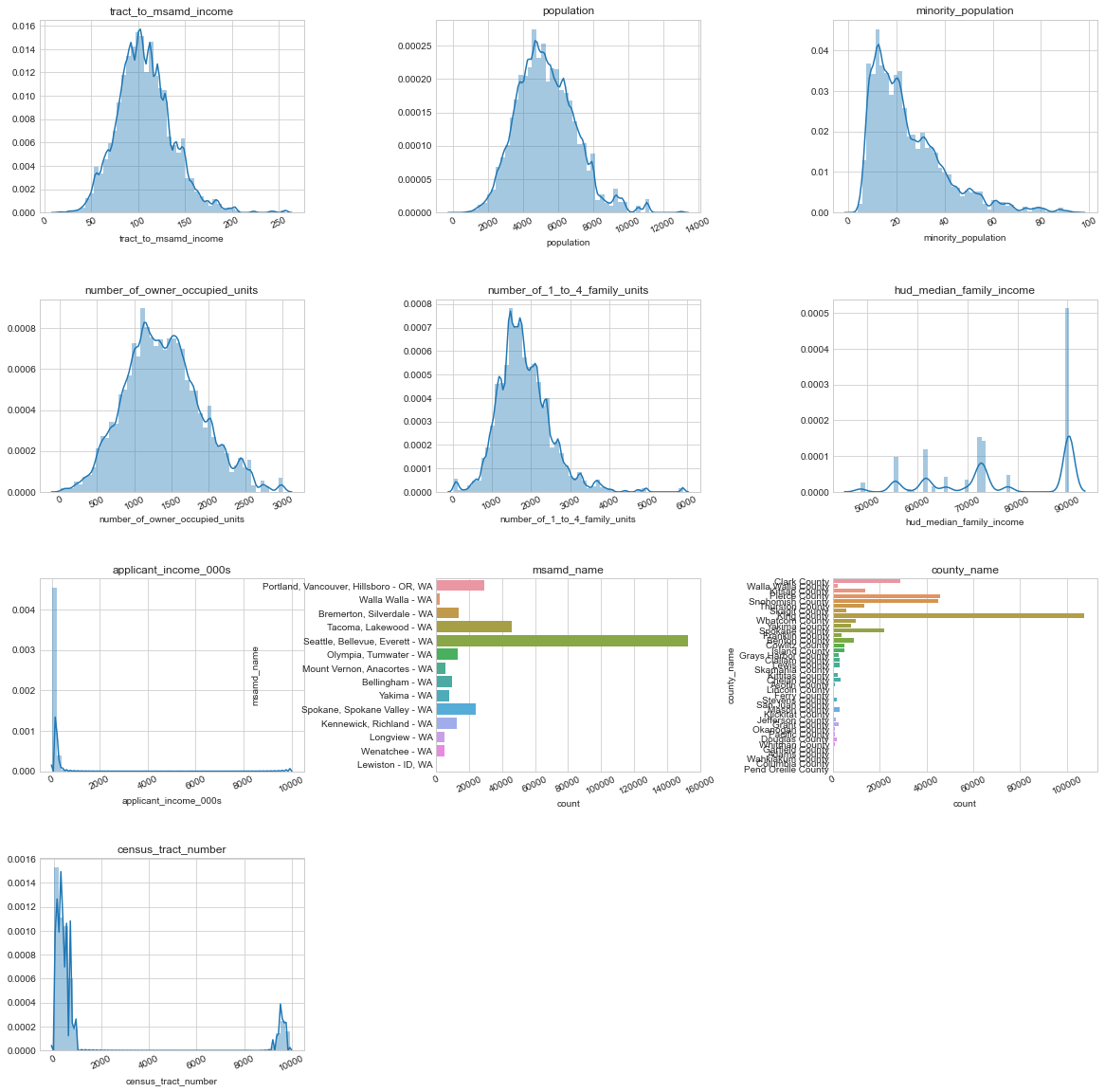
else:

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：



1. 各个特征的统计分布图

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。

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。

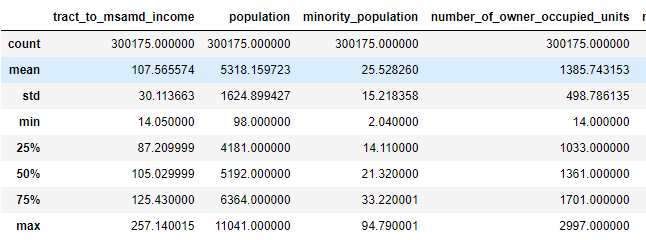
#### 根据特征类型，按照数值型与标称型分布分别分析

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。

查看数值型属性统计性描述

hdma.describe()

输出数据结果如下：



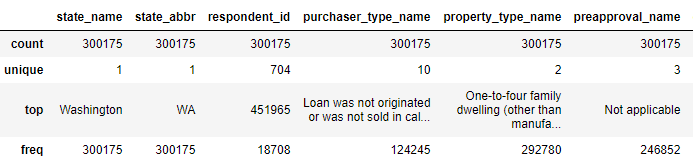
1. 描述性输出结果

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：



1. 标称属性统计数据查看

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。

#### 特征相关性分析

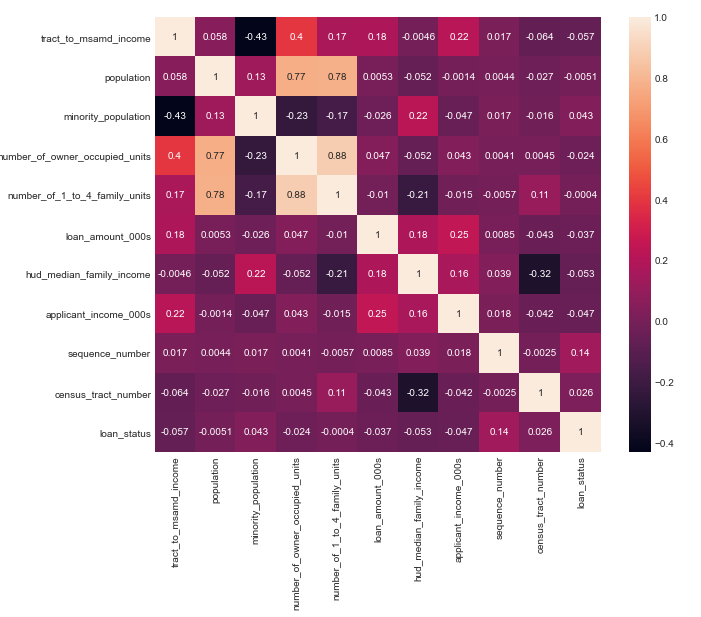
数值型特征相关性可视化

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：



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。

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']

#### 数据压缩：连续属性分箱

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']

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])

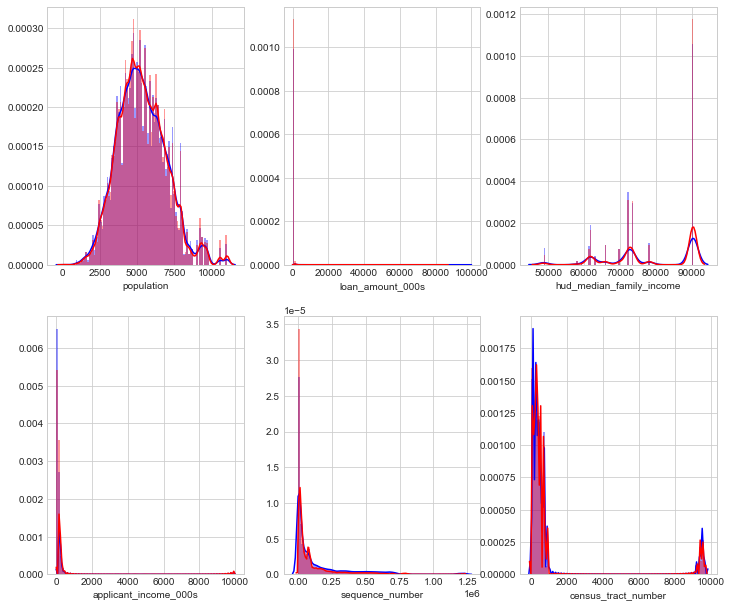
sns.distplot(hdma[j][hdma.loan\_status == 1], bins=100, color='b')

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：



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。

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)

#### 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。

转换为连续型数字变量

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

# print(dataset\_con['workclass'])

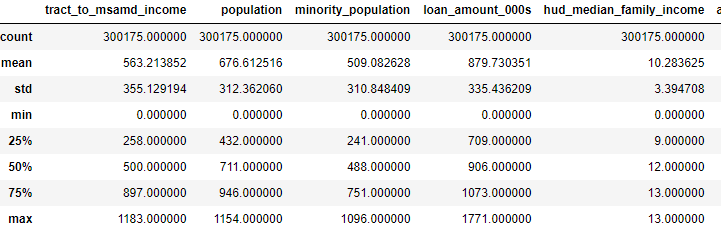
hdma = hdma.astype(str)

hdma\_enc = hdma.apply(encoder.fit\_transform)

查看编码后的数据描述

hdma\_enc.describe()

输出数据结果如下：



* + 1. 建立模型

本环节选择多个分类算法对数据进行分类预测。

#### 数据拆分

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)

#### 建立决策树模型

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

#### 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。

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

使用随机森林算法去查看属性重要性

plt.style.use('seaborn-whitegrid')

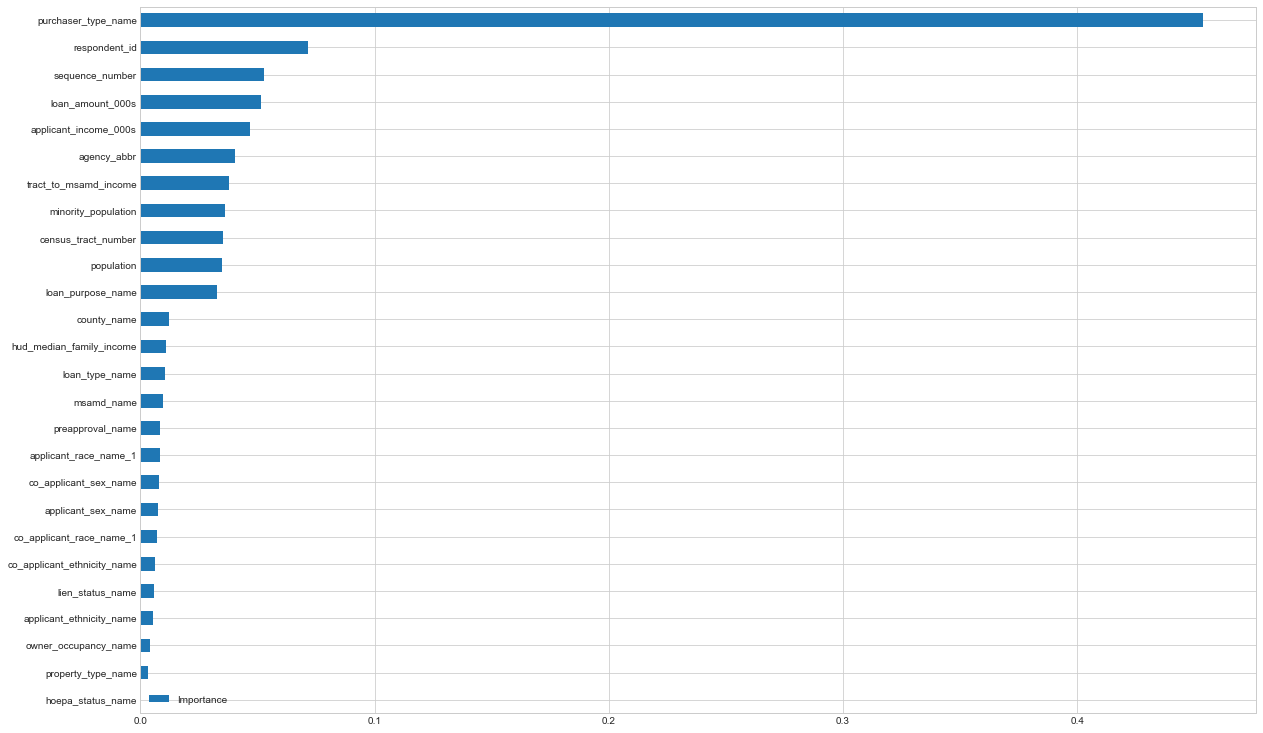
importance = rf\_model.feature\_importances\_

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：



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。

#### 建立GBDT模型

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

#### 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。

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。

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

* + 1. 模型评估与优化

#### 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**

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()

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

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

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

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']

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

#### 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。

## 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】

## 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。

## 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）

## 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。

1. **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>



1. Kaggle Housing Prices数据介绍

【实验环境要求】:

1、python3.7

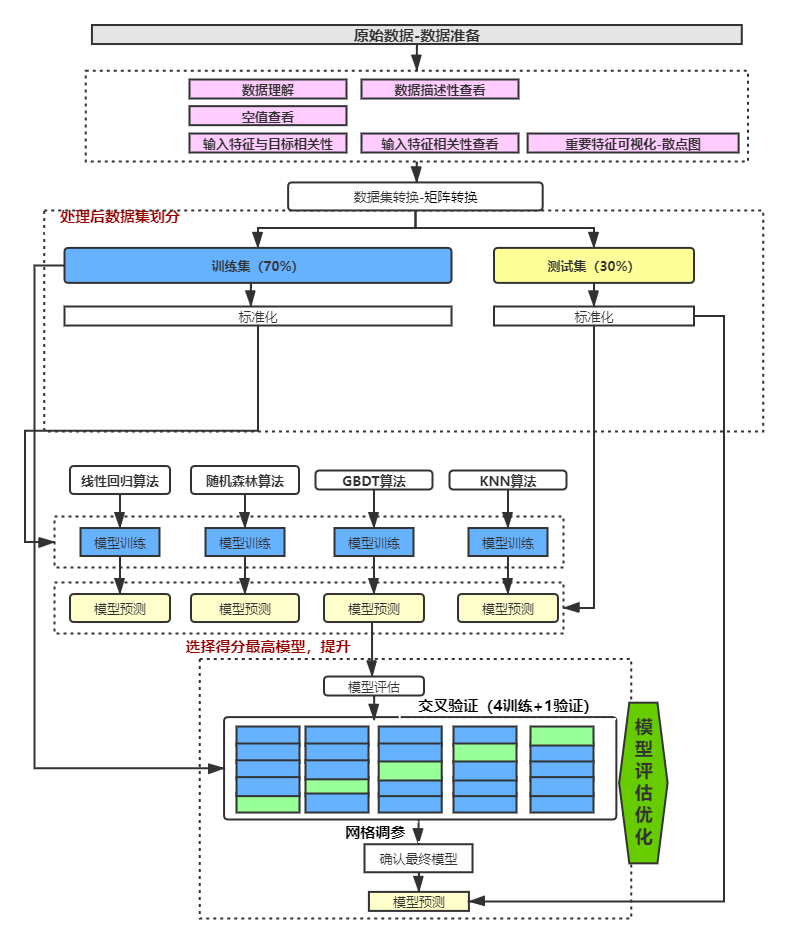
2、ModelArts平台

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/。

## 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。

* + 1. 实验整体方案



1. 实验方案流程图

## 实验详细设计与实现

* + 1. 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")

* + 1. 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。

* + 1. 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:]

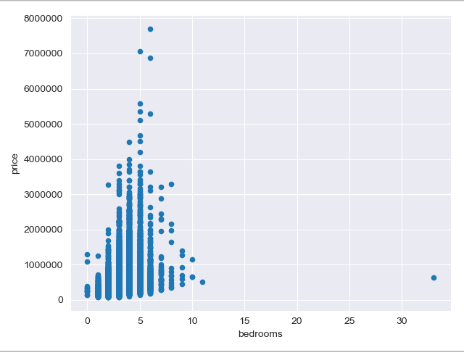
#分别分析所取的属性与价格的分布关系图

for x\_var in x\_vars:

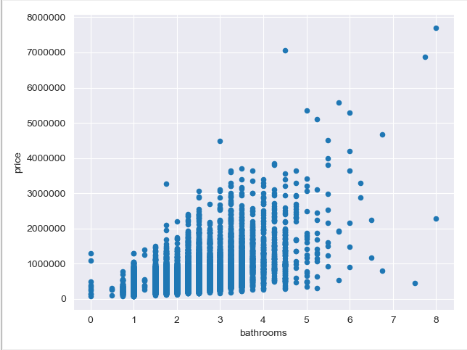
df.plot(kind='scatter',x=x\_var,y='price') #设置绘图的行和列

plt.show()

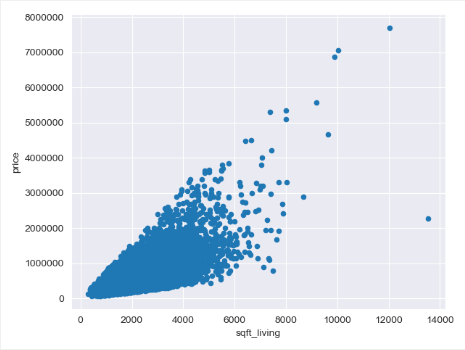
输出如下结果：



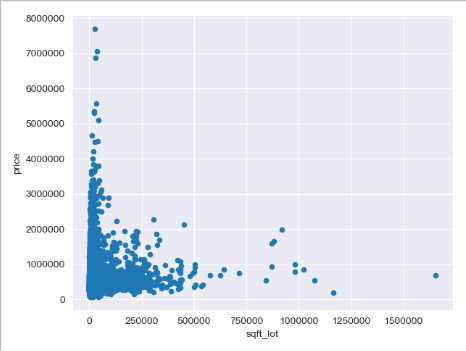
1. 卧室数与房价散点图



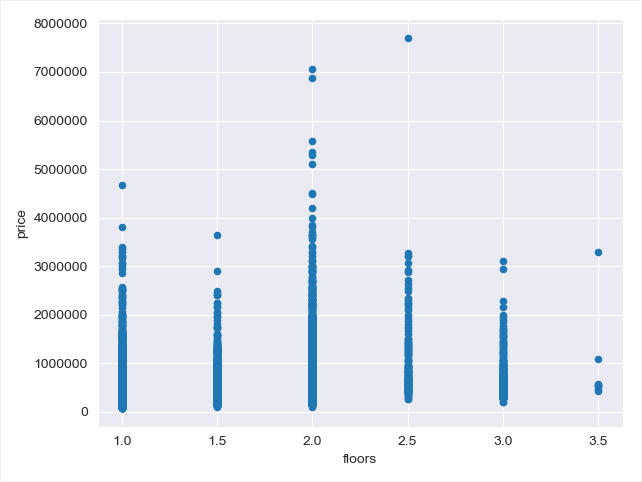
1. 卫生间数与房价散点图



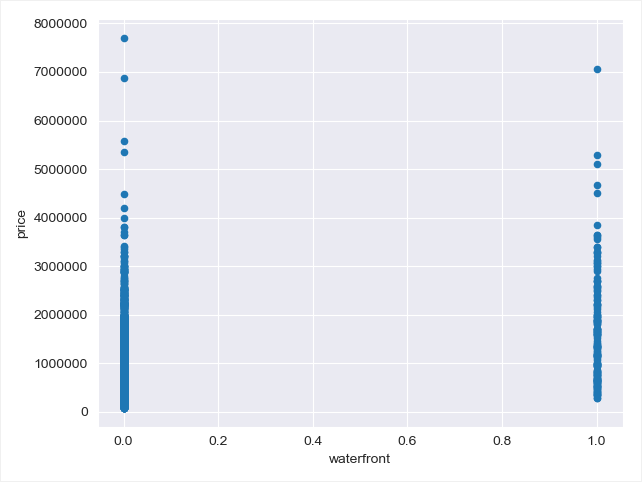
1. 住宅面积与房价散点图



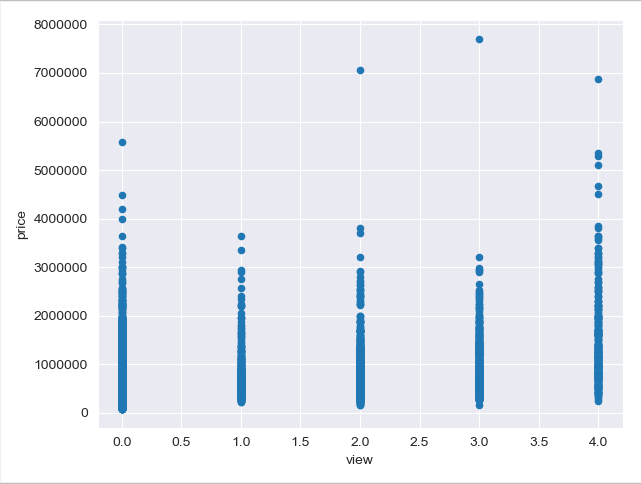
1. 停车场面积与房价散点图



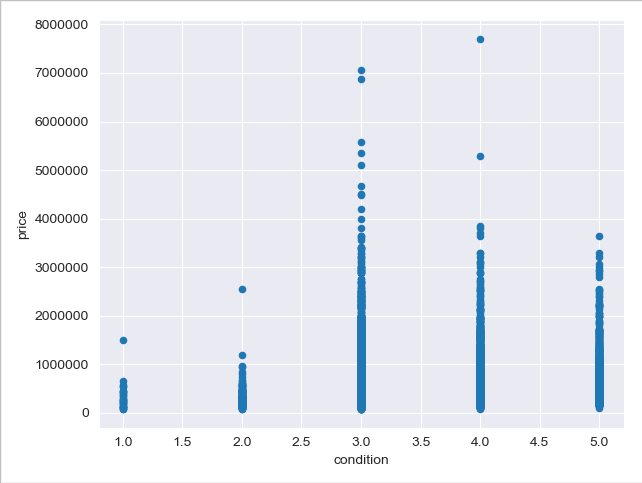
1. 楼层数与房价散点图



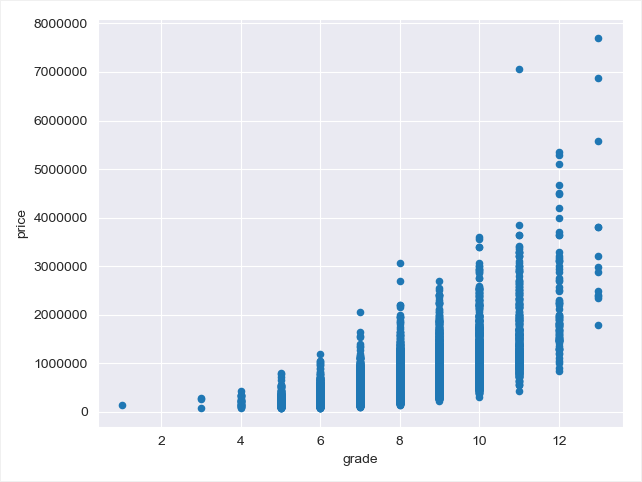
1. 泳池与房价散点图



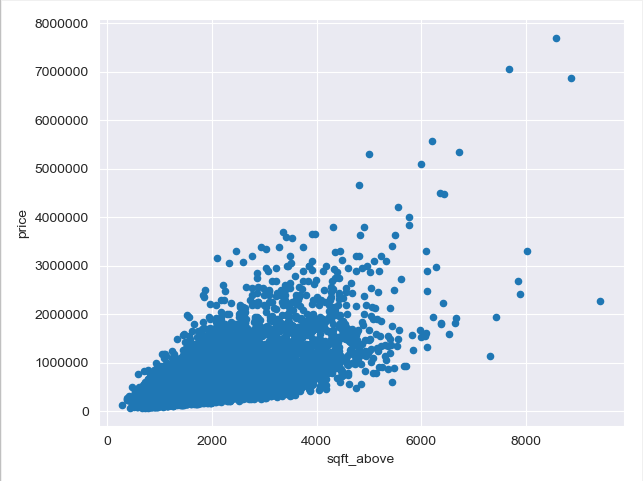
1. 外观与房价散点图



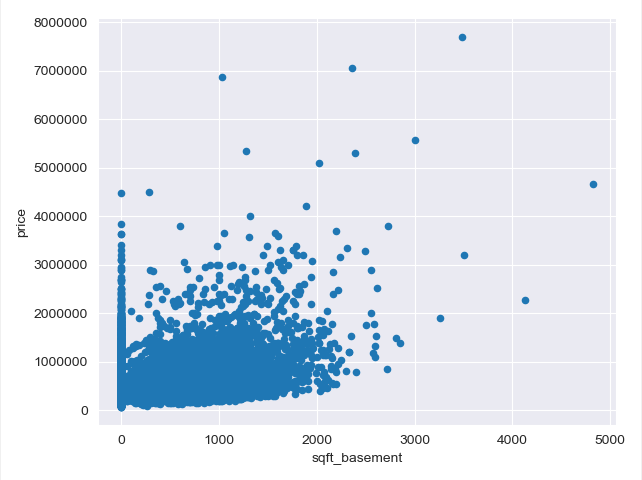
1. 房屋状态与房价散点图



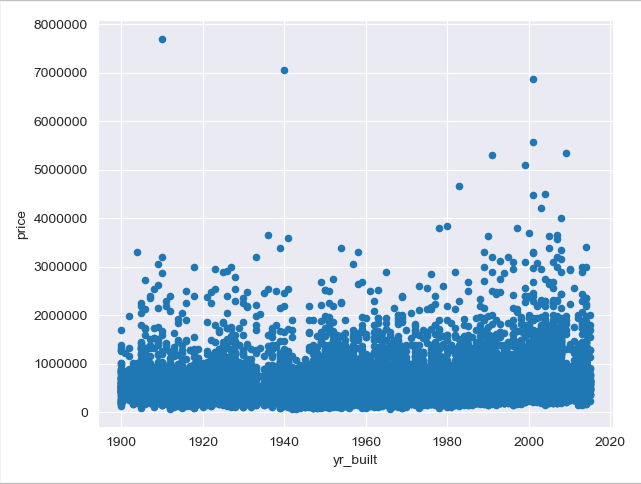
1. 房屋等级与房价散点图



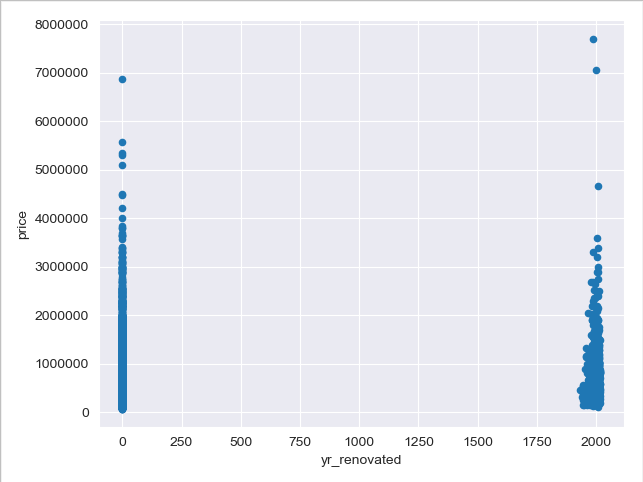
1. 地上面价与房价散点图



1. 地下室面积与房价散点图



1. 修建时间与房价散点图



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。

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()

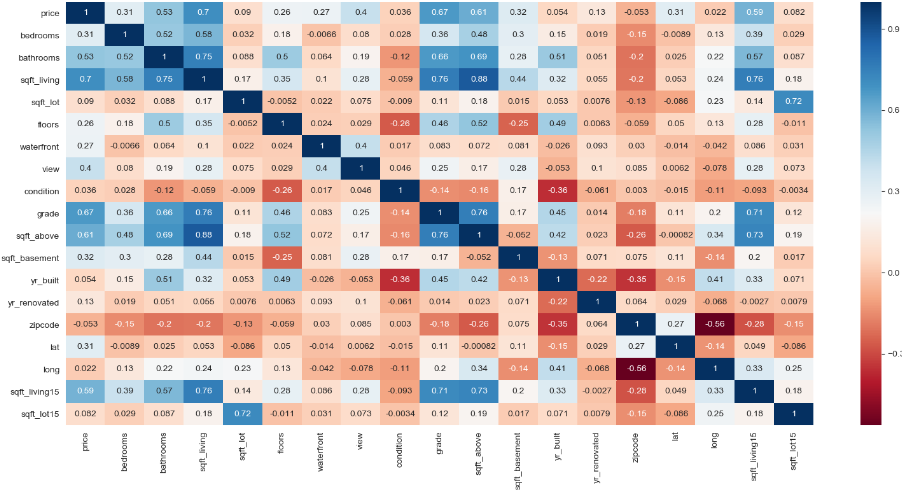
#绘制属性相关系数的热力图

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

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

plt.show()

输出如下结果：



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。

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

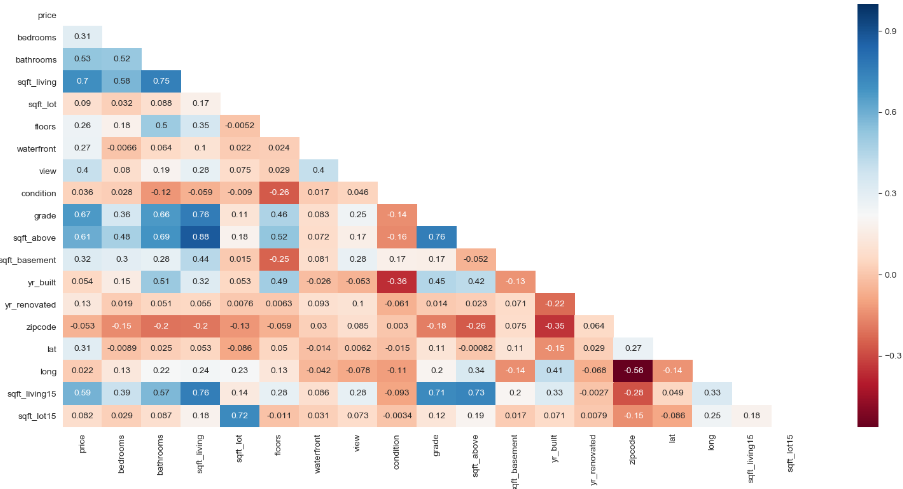
sns.set\_style(style="white")

#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：



1. 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

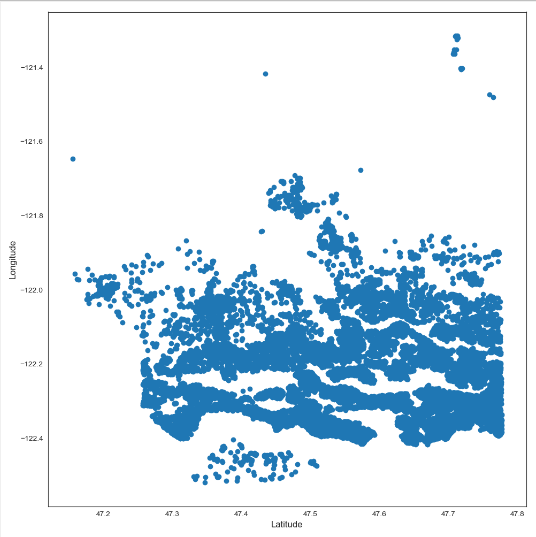
plt.scatter(df.lat, df.long)

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

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

plt.show()

输出如下结果：



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。

分析邮编与房价的分布关系

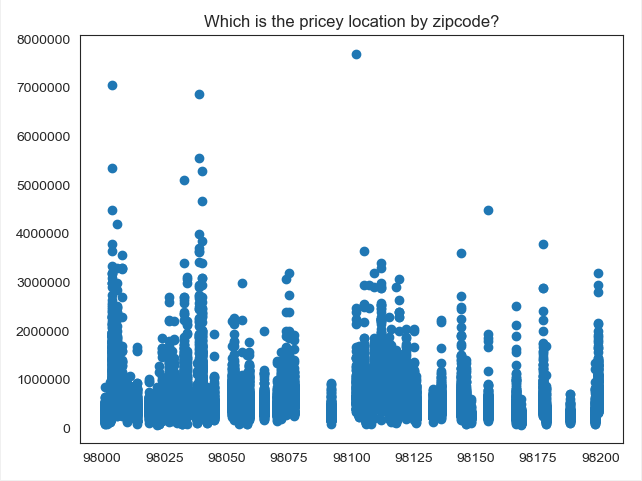
#绘制各区域具体的价格散点分布情况，了解每个区域的价格分布区间

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

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

plt.show()

输出如下结果：



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。

* + 1. 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)

* + 1. 数据建模分析

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。

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

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。

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？

【答案】

## 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。

## 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）