中国人民大学

课程报告

（2021-2022 学年秋季学期）

论文题目：客户细分问题与多分类模型封装

课程名称：编程基础

任课老师：罗翔宇

班　　级：应用统计专硕班

学　　号：2021103835

姓　　名：崔博涵

客户细分问题与多分类模型封装

摘　　要

本文从 Kaggle 上选择 customer-segmentation 数据集，旨在通过顾客的行为特征进行定位分类。基于选取的数据集，在 pandas 框架下对类别特征数据进行编码、归一化处理，将处理后的数据集输入经典的逻辑回归、决策树、等分类模型，调用 sklearn 中的 GridSearchCV() 函数进行 Xgboost 与 lightgb 分类模型超参数的寻优。基于优化后的模型，创建 customer 类，属性为客户的特征，通过异常构造函数的参数检查，若不合理，则触发异常拒绝生成对象，通过定义分类方法实现客户类别群的预判。该类将优化后的分类模型封装在一起，实现了客户类别预测的功能。本文代码和数据：https://github.com/Rabbicui/customer-segmentation.git

关键词：决策树；XGboost；Lightboost；customer 类；异常；

目　录

1 问题背景

1.1 背景简介

客户细分 (Customer Segmentation) 是将客户群体划分为与营销相关的特征相似的个人组的做法，例如根据年龄、性别、兴趣和消费习惯等客户信息进行群体的划分。采用客户细分的公司在这样一个背景下运作：每个客户都是不同的，如果针对特定的小群体，向这些消费者提供相关信息并引导他们购买东西，那么公司的营销工作将得到更好的服务。公司还希望通过发现每个细分市场最有价值的东西来更准确地针对该细分市场定制营销材料，从而更深入地了解客户的偏好和需求。

1.2 问题提出

一家汽车公司计划利用其现有产品（P1、P2、P3、P4 和 P5）进入新市场。经过深入的市场研究，他们推断出新市场的行为与现有市场相似。在他们现有的市场中，销售团队将所有客户分为 4 个细分市场（A、B、C、D）。然后，他们针对不同的客户类群进行了相应外展和沟通。这种策略对他们来说非常有效。他们计划在新市场上使用相同的策略，并确定了 2627 个新的潜在客户。本工作旨在帮助经理预测新客户的正确群体。

1.3 问题描述

针对上述背景，提出以下问题：

1. 针对已有的客户细分数据（Train.csv），总结各细分类群体的特征
2. 提出合理的模型预测潜在客户所在的细分群体
3. 定义一个客户类，实现以下功能
   (a) 属性为客户的特征，构造函数的参数检查，如果不合理（类型不合理或者取值不合理），请拒绝生成对象。比如年龄的范围是：15-100，数值型。
   (b) 初始化对象，打印输出客户的具体信息
   (c) 调用不同的分类器（随机森林,XGBoost,lgb），打印输出预测值与真实值的对比，返回 Bool 型变量

1.4 模型框架

为解决上述提出的问题，设定本文的模型框架如下：

1. 了解数据集
2. 对数据集进行探索性数据分析
   (a) 探索训练和测试数据并了解每列/特征表示的内容
   (b) 检查数据集中目标列的不平衡
   (c) 查看数据集各列特征的分布并可视化
3. 数据预处理

(a) 检查数据集中是否存在重复行，若存在，删除重复行

(b) 检查数据集中是否存在缺失值

(c) 填充/估算连续型数据的缺失值 - 平均值/中值/任何特定值和分类

(d) 填充离散型数据的缺失值，前向填充/后向填充

4. 特征工程

(a) 特征选择 - 选择最重要的现有特征

(b) 特征创建 - 从现有特征创建新特征

(c) 使用预测器（独立）和目标（相关）将训练数据拆分为训练和验证数据

(d) 数据编码 - 标签编码、OneHot 编码和数据缩放

(e) 数据归一化-MinMaxScaler、StandardScaler、RobustScaler

5. 为多类分类问题创建基线 ML 模型

6. 改进 ML 模型，使用 MODEL Evaluation METRIC - “Accuracy”调参，并预测 Traget “Outcome”

2 数据预处理

2.1 数据集基本信息

数据集选自 Kaggle https://www.kaggle.com/vetrirah/customer。本项目包括三个文件，训练集，测试集及测试集的标签。训练集包含 8068 条客户信息，每条信息由 11 个特征组成，测试集包含 2627 条客户信息, 经初步检查，其中有 2332 行与训练集重复，故本文直接从所给的 train.csv 文件中进行训练集与测试集的重划分。数据集具体解释说明如下表 1：

表 1 数据集基本特征描述

变量名称	数据类型	类别数目	解释说明
ID	数值型		唯一指定
性别	类别型	2	’Female’,’Male’
婚姻状况	类别型	2	’Yes’,’No’
学业状态	类别型	2	’Yes’,’No’
职业	类别型	9	客户的职业
工作年限	数值型		
消费等级	类别型	3	’Low’,’Average’,’High’
家庭成员数量	数值型		
Var-1	类别型	6	客户的匿名类别
群体细分	类别型	4	客户所在的细分群体

2.2 探索性数据分析

2.2.1 描述统计

调用 df.groupby() 与 df.describe() 函数分别对训练集中的数值型与离散型数据做描述性统计。

	ID	Age	Work_Experience	Family_Size
count	8068.000000	8068.000000	7239.000000	7733.000000
mean	463479.214551	43.466906	2.641663	2.850123
std	2595.381232	16.711696	3.406763	1.531413
min	458982.000000	18.000000	0.000000	1.000000
25%	461240.750000	30.000000	0.000000	2.000000
50%	463472.500000	40.000000	1.000000	3.000000
75%	465744.250000	53.000000	4.000000	4.000000
max	467974.000000	89.000000	14.000000	9.000000

	Gender	Ever_Married	Graduated	Profession	Spending_Score	Var_1	Segmentation
count	8068	7928	7990	7944	8068	7992	8068
unique	2	2	2	9	3	7	4
top	Male	Yes	Yes	Artist	Low	Cat_6	D
freq	4417	4643	4968	2516	4878	5238	2268

图 1 上下两图分别表示数值型、类别型数据描述统计分析

2.2.2 数据特征分布可视化

分别对训练集中的数值型与离散型数据进行特征分布的可视化得下图 2 和图 3。

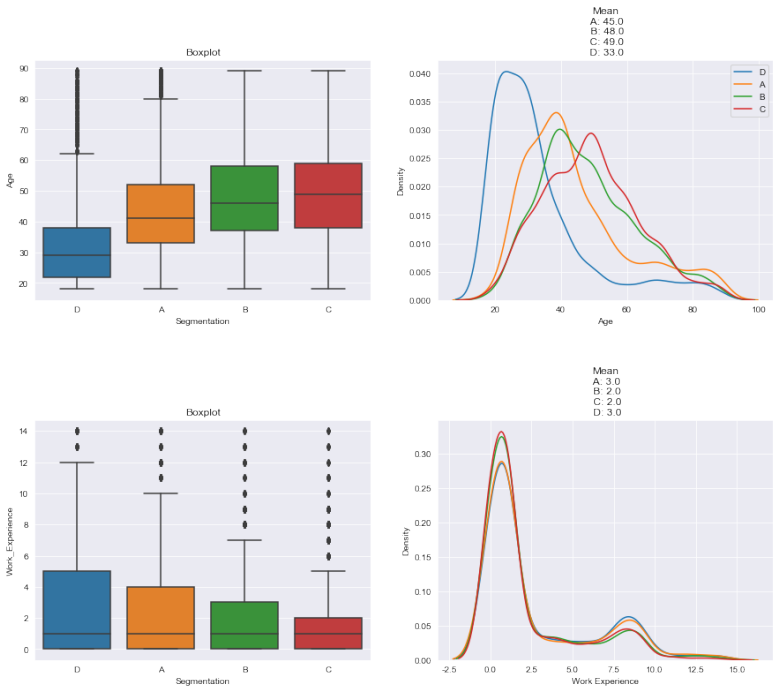

图 2 数值型数据可视化

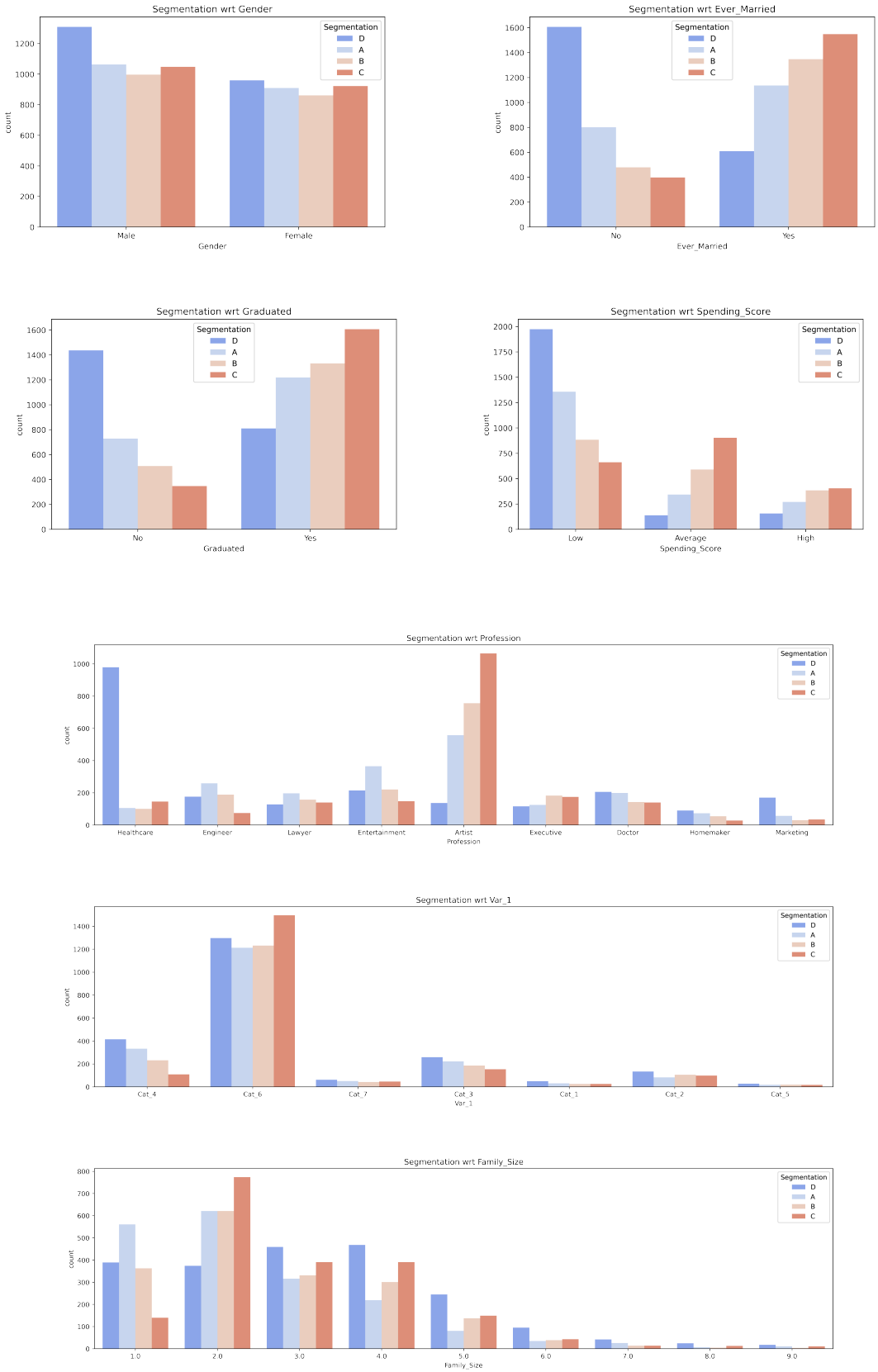
Segmentation wrt Gender
Segmentation
D
A
B
C
count
Male
Female
Gender
Segmentation wrt Ever_Married
No
Yes
Ever_Married
Segmentation wrt Graduated
Graduated
Segmentation wrt Spending_Score
Low
Average
High
Spending_Score
Segmentation wrt Profession
Healthcare
Engineer
Lawyer
Entertainment
Artist
Executive
Doctor
Homemaker
Marketing
Profession
Segmentation wrt Var_1
Cat_4
Cat_6
Cat_7
Cat_3
Cat_1
Cat_2
Cat_5
Var_1
Segmentation wrt Family_Size
Family_Size

图 3 类别型数据可视化

针对图 2 数值型数据的分布可视化可得：D 类群体的平均年龄为 33 岁，属于年轻群体，C 类群体的平均年龄为 49 岁，A 类和 B 类客户群体的平均年龄均在 45 岁以上，属于中年群体，而各类群体的工作年限分布基本一致。通过对图 3 类别型数据的分布可视化得出以下几点结论。（1）基于性别的细分类分布中，男性群体里 D 类客户较多，女性群体里的类别分布则较为均衡；（2）已婚人士这一群体的客户类别大部分属于 C 类，相对地，未婚人士群体的客户类别大都属于 D 类；（3）已经毕业的客户群体里 C 类占比较高，而非毕业生群体里，D 类客户占比较高；（4）职业为健康护理的群体中，D 类客户较多（5）低消费群体中，D 类客户较多。“Var-1”这一特征的类别分布较为均衡。

由此可得，C 类客户群体与 D 类客户群体特征区别较为明显，其中，C 类客户群体具有年龄在 50 岁左右，职业多从事文艺类行业，已婚，中等消费阶层等特征；D 类客户群体具有年轻化，未毕业，低消费阶层等特征。汽车公司的销售团队可根据用户的特征信息对所属类群进行初步划分。

2.3 数据预处理

数据处理全过程在 pandas 框架内进行。

2.3.1 数据编码

对于类别型特征，往往需要转换为特定的数值或者向量，才能被机器识别，这就需要使用数据编码进行处理，常见的数据编码方式有 label encoding 与 one-hot encoding 两种方式，前者主要对只有两个类别的数据通过 0 和 1 进行编码；后者则针对多类别数据进行编码，结果是一个稀疏向量。基于表 1 中的类别型特征，本文选择 label encoding 进行编码，尽管存在多分类数据，由于数据类别数目较小，使用标签编码不会造成数据量级的较大变化，统一处理起较为方便。

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1	Segmentation
0	462809	Male	No	22	No	Healthcare	1.0	Low	4.0	Cat_4	D
1	462643	Female	Yes	38	Yes	Engineer	NaN	Average	3.0	Cat_4	A
2	466315	Female	Yes	67	Yes	Engineer	1.0	Low	1.0	Cat_6	B
3	461735	Male	Yes	67	Yes	Lawyer	0.0	High	2.0	Cat_6	B
4	462669	Female	Yes	40	Yes	Entertainment	NaN	High	6.0	Cat_6	A

图 4 data.head()-编码前

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1	Segmentation
0	0	0.0	22	0.0	1.0	1.0	0	4.0	4.0	D
1	1	1.0	38	1.0	3.0	NaN	1	3.0	4.0	A
2	1	1.0	67	1.0	3.0	1.0	0	1.0	6.0	B
3	0	1.0	67	1.0	5.0	0.0	2	2.0	6.0	B
4	1	1.0	40	1.0	2.0	NaN	2	6.0	6.0	A

图 5 data.head()-编码后

2.3.2 缺失值处理

首先对数据集进行缺失值的检查，缺失数据最大的行数为 829 行，约占比 10%, 考虑到删除缺失值的成本较高，本文选择缺失值填充的方法对缺失值进行处理。定义填充规则如下：针对“婚姻状况”, “是否毕业”, “职业”, “Var-1”等类别型特征，设定 fillna 中的”method=ffill”，采用缺失值之前的最后一个有效值进行填充；针对家庭成员数目，工作年限等数值型数据，采用均值方法进行填充。

```
Gender               0
Ever_Married       140
Age                  0
Graduated           78
Profession         124
Work_Experience    829
Spending_Score       0
Family_Size        335
Var_1               76
Segmentation         0
dtype: int64
```

图 6 缺失值统计

2.3.3 特征选择与数据集划分

首先对经过处理后的数据集的特征之间进行相关性检验，若特征变量之间存在较强的相关性，则需要使用因子分析法提取核心变量，特征变量的相关系数表如下图所示：

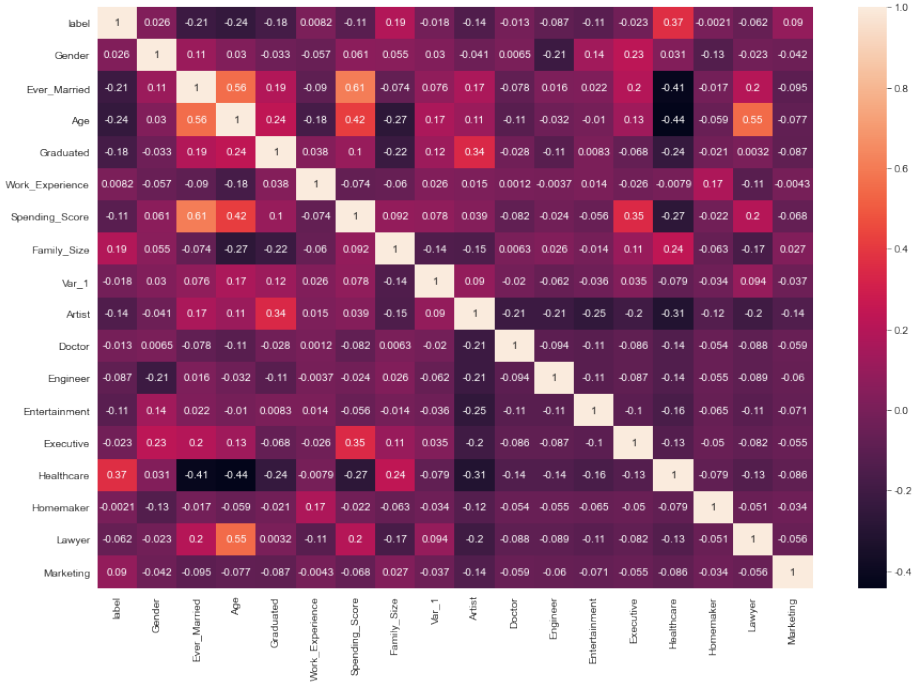

图 7 相关系数矩阵

从图中可以看出，各特征变量之间不存在明显的相关性关系，选择所有特征变量输入接下来的模型中并将数据集按照 3：1 的比例划分为训练集与测试集。

3 模型结果及参数优化

3.1 模型运行结果

从 sklearn 中调用 LogisticRegression(),SVC(),KNeighborsClassifier(),DecisionTreeClassifier(), RandomForestClassifier() 函数，安装并调用 Xgboost 和 lightgbm 模型对数据集进行训练，参数均

设定为默认参数，训练结果如表 2：

表 2 各模型训练准确率

模型名称	准确率
逻辑回归	0.4685
支持向量机	0.4586
K-临近法 ()	0.4531
决策树	0.4395
随机森林	0.4908
XGBClassifier()	0.5439
LGBMClassifier()	0.5290

3.2 优化调参

基于 lightgbm 模型，设置学习率为 0.15，通过 k-交叉验证选择最优的迭代次数为 1000，调用 sklearn 中的 GridSearchCV() 函数进行超参数'num-leaves' 的寻优，最优参数设定如下，最终训练准确率为 0.589：

```
1
2  lgb_model = LGBMClassifier(
3  boosting_type='gbdt',
4  max_depth=15,
5  learning_rate=0.15,
6  objective='multiclass', # 多分类
7  random_state=100,
8  n_estimators=1000 ,
9  reg_alpha=0,
10 reg_lambda=1,
11 n_jobs=-1)
```

从训练结果不难看出，基于该数据集的模型训练准确率比较低，准确率的进一步提高考虑从数据集与模型的优化两方面进行切入。首先针对数据集而言，特征变量相对较少，可尝试对多分类特征进行独热编码，或通过调用多项式回归函数扩充特征变量集。另外一方面从模型本身进行优化，优化方向见下图

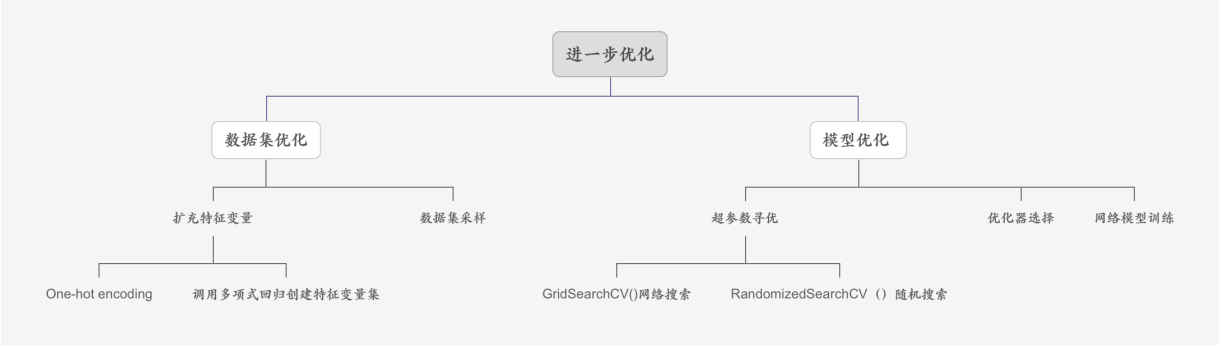

图 8 优化框架

4 Customer 类的建立与使用

4.1 Customer 类的创建

创建一个 Customer 类实现以下功能：

（1）输入客户信息，初始化类属性，构造函数的参数检查，如果不合理（类型不合理或者取值不合理），则引发异常，拒绝生成对象。（2）打印输出客户的具体信息（3）调用不同方法，实现该客户所在群体的预测，并与实际标签进行对比

```
class Customer:
#初始化客户信息
def init(self, gender,marriage,age,graduation,profession,work,spending,family,var_1
    ,segmentation):
self.Gender=gender
self.Ever_Married=marriage
self.Age=age
self.Graduated=graduation
self.Profession=profession
self.Work_Experience=work
self.Spending_Score=spending
self.Family_Size=family
self.Var_1=var_1
self.Segmentation=segmentation
if not isinstance(profession, str): raise TypeError('需要输入字符型')
if not isinstance(spending, str): raise TypeError('请在low,average,high中选择')
if not isinstance(age,int): raise TypeError('年龄为整数型')
if not isinstance(work,int): raise TypeError('工作经验为整数型')
if not isinstance(family,int): raise TypeError('工作经验为整数型')
if marriage not in ['Yes','No']:raise ValueError('请输入Yes or No')
if graduation not in ['Yes','No']:raise ValueError('请输入Yes or No')
if age < 15 or age > 100: raise ValueError('are you kidding')
#打印客户信息
def printinfo(self):
print("Hello, world!\n Gender: {}.\n Ever_Married: {}.\n Age: {}.\n Graduated:{}.\n
    Profession:{}.\n Work_Experience:{}.\n Spending_Score:{}.\n Family_Size:{}.\n
    Var_1:{}.\n Segmentation:{}".format(self.Gender,self.Ever_Married,self.Age,self.
    Graduated,self.Profession,self.Work_Experience,self.Spending_Score,self.
    Family_Size,self.Var_1,self.Segmentation) )
#随机森林
def rf_pre(self,xtrain,ytrain):
customer_list=[]
ll=[self.Gender,self.Ever_Married,self.Age,self.Graduated,self.Profession,self.
    Work_Experience,self.Spending_Score,self.Family_Size,self.Var_1]
for i in ll:
if i in ['No','Male','Artist','Cat_1','Low']:customer_list.append(0)
elif i in ['Yes','Female','Healthcare','Cat_2','Average']:customer_list.append(1)
elif i in ['Cat_3','High','Entertainment']:customer_list.append(2)
elif i in ['Cat_4','Engineer']:customer_list.append(3)
elif i in ['Cat_5','Doctor']:customer_list.append(4)
```

```
elif i in ['Cat_6','Lawyer']:customer_list.append(5)
elif i =='Marketing':customer_list.append(7)
elif i =='Homemaker':customer_list.append(8)
else:customer_list.append(i)
#对原始客户信息数据进行处理
model=RandomForestClassifier()
model.fit(xtrain,ytrain)
pred=model.predict((np.array(customer_list)).reshape(1,-1))
if pred[0]==0:ec='A'
elif pred[0]==1:ec='B'
elif pred[0]==2:ec='C'
elif pred[0]==3:ec='D'
print('真实值:{}  预测值:{}'.format(self.Segmentation,ec))
return(self.Segmentation==ec)

#xgb
def xgb_pre(self,xtrain,ytrain):
customer_list=[]
ll=[self.Gender,self.Ever_Married,self.Age,self.Graduated,self.Profession,self.
    Work_Experience,self.Spending_Score,self.Family_Size,self.Var_1]
for i in ll:
if i in ['No','Male','Artist','Cat_1','Low']:customer_list.append(0)
elif i in ['Yes','Female','Healthcare','Cat_2','Average']:customer_list.append(1)
elif i in ['Cat_3','High','Entertainment']:customer_list.append(2)
elif i in ['Cat_4','Engineer']:customer_list.append(3)
elif i in ['Cat_5','Doctor']:customer_list.append(4)
elif i in ['Cat_6','Lawyer']:customer_list.append(5)
elif i =='Marketing':customer_list.append(7)
elif i =='Homemaker':customer_list.append(8)
else:customer_list.append(i)
#print(ll)
model=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bynode=1, colsample_bytree=0.8, gamma=5,
learning_rate=0.3, max_delta_step=0, max_depth=4,
min_child_weight=5, missing=1, n_estimators=500, n_jobs=4,
nthread=None, objective='multi:softprob', eval_metric='mlogloss', use_label_encoder
    =False,
random_state=0,reg_alpha=0, reg_lambda=1, scale_pos_weight=None, seed=None,
silent=None, subsample=0.8, verbosity=None)
model.fit(xtrain,ytrain)
pred=model.predict((np.array(customer_list)).reshape(1,-1))
if pred[0]==0:ec='A'
elif pred[0]==1:ec='B'
elif pred[0]==2:ec='C'
elif pred[0]==3:ec='D'
print('真实值:{}  预测值:{}'.format(self.Segmentation,ec))
return(self.Segmentation==ec)

#lgb
def lgb_pre(self,xtrain,ytrain):
```

```
customer_list=[]
ll=[self.Gender,self.Ever_Married,self.Age,self.Graduated,self.Profession,self.
    Work_Experience,self.Spending_Score,self.Family_Size,self.Var_1]
for i in ll:
if i in ['No','Male','Artist','Cat_1','Low']:customer_list.append(0)
elif i in ['Yes','Female','Healthcare','Cat_2','Average']:customer_list.append(1)
elif i in ['Cat_3','High','Entertainment']:customer_list.append(2)
elif i in ['Cat_4','Engineer']:customer_list.append(3)
elif i in ['Cat_5','Doctor']:customer_list.append(4)
elif i in ['Cat_6','Lawyer']:customer_list.append(5)
elif i =='Marketing':customer_list.append(7)
elif i =='Homemaker':customer_list.append(8)
else:customer_list.append(i)
model= LGBMClassifier(
boosting_type='gbdt',
max_depth=15,
learning_rate=0.1,
objective='multiclass', # Multi Class Classification
random_state=100,
n_estimators=1000 ,
reg_alpha=0,
reg_lambda=1,
n_jobs=-1)
model.fit(xtrain,ytrain)
pred=model.predict((np.array(customer_list)).reshape(1,-1))
if pred[0]==0:ec='A'
elif pred[0]==1:ec='B'
elif pred[0]==2:ec='C'
elif pred[0]==3:ec='D'
print('真实值:{}  预测值:{}'.format(self.Segmentation,ec))
return(self.Segmentation==ec)
```

4.2 Customer 类的使用

下面通过具体实例说明该类的使用。创建一个客户对象 jone，并初始化；调用 printinfo() 方法打印客户的具体信息，基于给定的已处理好的训练集，调用三种不同分类方法实现预测：

```
jone=Customer()
gender,marriage,age,graduation,profession,work,spending,family,var_1,segmentation=test.iloc[0,1],test.iloc[0,2],int(tes
jone.init(gender,marriage,age,graduation,profession,work,spending,family,var_1,segmentation)
```

```
jone.printinfo()
```

```
Hello, world!
 Gender: Female.
 Ever_Married: Yes.
 Age: 36.
 Graduated:Yes.
 Profession:Engineer.
 Work_Experience:0.
 Spending_Score:Low.
 Family_Size:1.
 Var_1:Cat_6.
 Segmentation:A
```

```
jone.rf_pre(X_train,y_train)
```

```
真实值:A  预测值:B

False
```

```
jone.xgb_pre(X_train,y_train)
```

```
真实值:A  预测值:A

True
```

```
jone.lgb_pre(X_train,y_train)
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leaves. (num
_leaves=31).
真实值:A  预测值:A

True
```

设置年龄为 ndarray 类型，触发类型异常

```
jone=Customer()
gender,marriage,age,graduation,profession,work,spending,family,var_1,segmentation=test.iloc[0,1],test.iloc[0,2],test.il
jone.init(gender,marriage,age,graduation,profession,work,spending,family,var_1,segmentation)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-302-4703328abb34> in <module>
      1 jone=Customer()
      2 gender,marriage,age,graduation,profession,work,spending,family,var_1,segmentation=test.iloc[0,1],test.iloc[0,
2],test.iloc[0,3],test.iloc[0,4],test.iloc[0,5],int(test.iloc[0,6]),test.iloc[0,7],int(test.iloc[0,8]),test.iloc[0,
9],sub.iloc[0,1]
----> 3 jone.init(gender,marriage,age,graduation,profession,work,spending,family,var_1,segmentation)

<ipython-input-298-23702fc87243> in init(self, gender, marriage, age, graduation, profession, work, spending, family,
var_1, segmentation)
     13         if not isinstance(profession, str): raise TypeError('需要输入字符型')
     14         if not isinstance(spending, str): raise TypeError('请在low,average,high中选择')
---> 15         if not isinstance(age,int): raise TypeError('年龄为整数型')
     16         if not isinstance(work,int): raise TypeError('工作经验为整数型')
     17         if not isinstance(family,int): raise TypeError('工作经验为整数型')

TypeError: 年龄为整数型
```

本文创建的客户类方法中可以添加更多优化的模型，本质上是将各分类模型封装在一起使用，调用分类模型之前需要导入相应的 Python 包；需要注意的是训练集的处理方式必须要和类中针对客户信息数据的处理方式保持一致，因此基于客户类的优化方向主要有（1）对象构造过程的简易化，逐个传参不适用于自变量较多的数据集（2）客户信息数据处理的多样性，根据输入训练集的编码方式进行选择（3）模型的预测准确率，需要针对各分类模型进行优化。