第*8*章

## 第8章 实战案例：Elo Merchant Category Recommendation（Kaggle）

本章将以Kaggle平台2019年的Elo Merchant Category Recommendation竞赛为例进行用户画像相关的实战讲解，端到端讲解完整的实战流程与注意事项。本章将主要分为五个部分来进行讲解，分别是赛题理解、探索性分析、特征工程、模型选择、赛题总结，这是本书所有实战案例章节的组织结构，也是一场竞赛的重要组成流程。相信在本书的指引下，读者能快速地熟悉竞赛流程，并进行实战。

### 8.1 赛题理解

所谓磨刀不误砍柴工，做比赛前应对赛题相关信息进行充分了解，理解背后的需求，进而达到正确审题的目的。

#### 8.1.1 赛题背景

想象一下，当您在一个不熟悉的地方饿着肚子想要找到好吃的东西时，你会得到基于你的个人喜好而专属推荐的餐馆，而且该建议附带了您的信用卡提供商为您提供的附近的本地地方的折扣！

目前，巴西最大的支付品牌之一Elo已经与商家建立了合作关系，以便向持卡人提供促销或折扣。但这些促销活动对消费者或商家都有效吗？顾客喜欢他们的体验吗？商家看到重复交易了吗？要回答这些问题，个性化是其中的关键。

Elo建立了机器学习模型，以了解客户生命周期中从食品到购物等最重要的方面和偏好。但到目前为止，它们都不是专门为个人或个人资料量身定做的，这也就是这场比赛举办的原因。

在这场竞赛中，希望参赛者将开发算法，通过揭示客户忠诚度的信号，识别并为个人提供最相关的机会。您的意见将改善客户的生活，帮助Elo减少不必要的活动，为客户创造正确的体验。

#### 8.1.2 赛题数据

为了保证用户隐私与信息安全，本次竞赛的所有数据都是模拟与虚构或者说进行了脱敏的，并非真实的消费者数据。

具体包含的文件如下，

train.csv - 训练集

test.csv - 测试集

sample\_submission.csv - 正确与规范的提交示例文件，含有需要参赛者预测的所有的card\_id

historical\_transactions.csv - card\_id的交易记录，其中每个id的记录最多含有三个月

merchants.csv - 数据集中所有商家（商家id）的附加信息

new\_merchant\_transactions.csv - 在历史交易数据中未出现的、每张信用卡card\_id最多两个月的、在所有商家的购物数据

Data\_Dictionary.xlsx - 数据字典的说明文件，提供了上述各表的字段含义，包括train、history、new\_merchant\_period和merchant的相应说明，相信参赛者跟笔者一样疑惑这个new\_merchant\_period又是什么鬼，且继续往下读。

Data Dictionary.xlsx - 与上面的文件一样，可能是主办方不小心多加了一个，对此读者不必感到气愤，事实上，大多数实际业务中的数据描述不一定能有如此详细了，因此参赛者可不必纠结相关命名的不完全一致。当然对于编程这件事来说，有强迫症与完美主义倾向则是一件十分有益的事，毕竟干净整洁的东西总是能给人带来舒适

#### 8.1.3 赛题任务

预测测试集里面所有信用卡card\_id的忠诚度分数

#### 8.1.4 评价指标

本次竞赛采用均方根误差作为评价指标，用来计算参赛者提交结果的成绩，具体计算方式如下，



其中y^ 是参赛者对每个card\_id预测的忠诚度分数，而 y 是对应的card\_id的真实忠诚度分数。

#### 8.1.5 赛题FAQ

Q：比赛提供了这么多文件，我至少需要哪些才能完成建模？

A：至少需要train.csv和test.csv两个文件，它们包括了所有将会被用来进行训练与的信用卡card\_id. 另外historical\_transactions.csv 和new\_merchant\_transactions.csv两个文件包含了每张信用卡的交易记录。其中 historical\_transactions.csv包含了每张卡最多三个月在任意给定的商户中的消费记录.而new\_merchant\_transactions.csv则包含了用户在两个月时间内访问的新商家用户记录。merchants.csv包含有关数据集中表示的每个商家的汇总信息。

Q：参赛者应该期望的数据格式是什么？

A： train.csv和test.csv包含card\_ids和有关卡本身的信息如卡激活的第一个月等。此外train.csv还包含目标值，即这部分用户有确定提供的忠诚度用户分值。history\_transactions.csv和new\_merchant\_transactions.csv设计为与train.csv，test.csv和merchants.csv结合在一起。如上所述它们包含有关每张卡交易的信息，可以将商家与交易记录结合在一起以提供额外的商家级别等信息。

### 8.2 探索性分析

相信很多参赛者与笔者一样，即使读完赛题理解的各项内容依然感觉到有些迷惑，老话说，Talk is cheap，show me the data. 千言万语都不如直接理解数据来得实在，相信很多问题通过数据的观察分析就能得到解决。在数据挖掘领域有个专有名词叫做探索性数据分析（EDA），这不仅能帮助参赛者理解题目的真实含义，了解数据概况，还能对接下来的特征工程与建模思路产生引导作用，进一步加强参赛者对业务的理解和技术的应用，因此首先我们需要做的便是对数据及进行探索。读到这里也许有的读者已经开始跃跃欲试打算开动码力，在写代码之前我想要建议的是可以的话，借助Excel这一强大的表格工具，先打开竞赛提供的各种文件获得一个直观的感受，本题目就可以直接查看train.csv、test.csv、sample\_submission.csv、以及Data\_Dictionary.xlsx，一般来讲50M及以上大小的Excel文件就不便直接用Excel打开，因为容易造成电脑卡顿。当然也要注意Excel本身数据格式对文件呈现带来的影响，如科学计数法、文本、以及日期等。

#### 8.2.1 字段类别含义

在进行探索性分析前，参赛者首先应该明确表格的介绍以及字段含义，便于理解赛题和搭建分析逻辑。参考赛题主办方提供的字段信息表Data\_Dictionary.xlsx可知，五张csv数据表格的字段含义如下，

|  |  |  |  |
| --- | --- | --- | --- |
| **Columns** | **Description** | **含义** | 举例 |
| card\_id | Unique card identifier | 信用卡id | C\_ID\_92a2005557 |
| first\_active\_month | 'YYYY-MM', month of first purchase | 首次使用的月份 | 2017-04 |
| feature\_1 | Anonymized card categorical feature | 匿名离散特征1 | 3 |
| feature\_2 | Anonymized card categorical feature | 匿名离散特征2 | 2 |
| feature\_3 | Anonymized card categorical feature | 匿名离散特征3 | 1 |
| target | Loyalty numerical score calculated  2 months after historical and evaluation period | 忠诚度分数目标列 | 0.392913 |

表格1 train.csv与test.csv的字段与含义

通过查看训练集与测试集字段的含义可知，三个feature都是匿名的信用卡离散字段，还有一个首次购物的月份，而标签是在历史和评估时期后的两个月进行量化计算得到的忠诚度分数，需要注意的是这里的evaluation period应该是指new\_merchant\_transactions.csv中的信息，同时也是对应Data\_Dictionary.xlsx里面的new\_merchant\_period字段。同时校验一下数据的正确性可以发现训练集与测试集的card\_id均为唯一值，且训练集与测试集的card\_id无重复。

|  |  |  |  |
| --- | --- | --- | --- |
| **Columns** | **Description** | **含义** | **举例** |
| card\_id | Card identifier | 信用卡id | C\_ID\_415bb3a509 |
| month\_lag | month lag to reference date | 距离参考日期的月份 | historical\_transactions.csv [-12, -1]  new\_merchant\_transaction.csv [0, 2] |
| purchase\_date | Purchase date | 购买日期(时间) | 2018-03-11 14:57:36 |
| authorized\_flag | Y' if approved, 'N' if denied | 是否推荐 | Y/N |
| category\_3 | anonymized category | 匿名类别特征3 | A/B/C/D/E |
| installments | number of installments of purchase | 购买商品的数量 | 1 |
| category\_1 | anonymized category | 匿名类别特征1 | Y/N |
| merchant\_category\_id | Merchant category identifier (anonymized ) | 商品种类id（经过匿名处理） | 307 |
| subsector\_id | Merchant category group identifier (anonymized ) | 商品种类群id(经过匿名处理) | 19 |
| merchant\_id | Merchant identifier (anonymized) | 商品id(经过匿名处理) | M\_ID\_b0c793002c |
| purchase\_amount | Normalized purchase amount | 标准化的购买金额 | -0.557574 |
| city\_id | City identifier (anonymized ) | 城市id(经过匿名处理) | 330 |
| state\_id | State identifier (anonymized ) | 州id(经过匿名处理) | 9 |
| category\_2 | anonymized category | 匿名类别特征2 | 1 |

表格2 historical\_transactions.csv 和 new\_merchant\_transaction.csv 的字段与含义

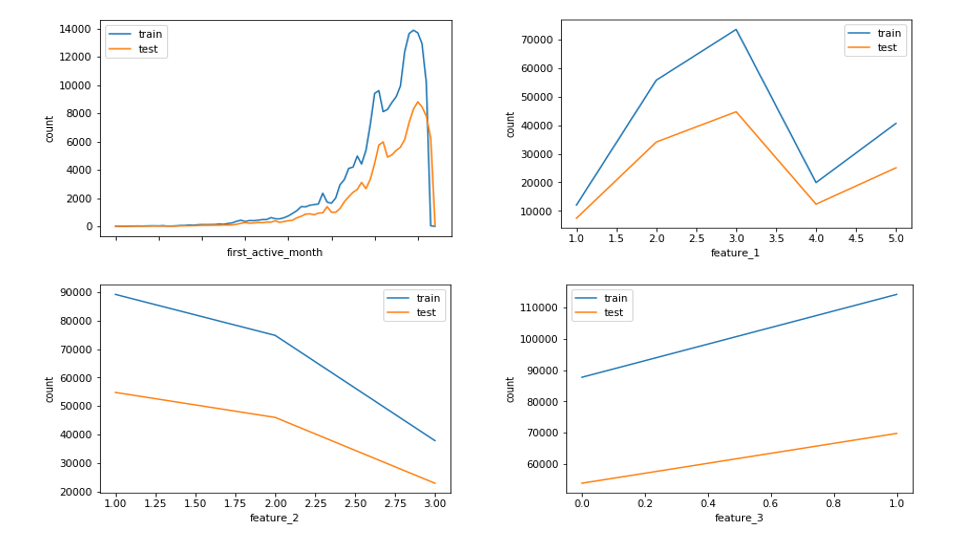
|  |  |  |  |
| --- | --- | --- | --- |
| **Columns** | **Description** | **含义** | **举例** |
| merchant\_id | Unique merchant identifier | 商品id(经过匿名处理) | M\_ID\_b0c793002c |
| merchant\_group\_id | Merchant group (anonymized ) | 商品组 | 8353 |
| merchant\_category\_id | Unique identifier for merchant  category (anonymized ) | 商品种类id（经过匿名处理） | 307 |
| subsector\_id | Merchant category group (anonymized ) | 商品种类群id(经过匿名处理) | 19 |
| numerical\_1 | anonymized measure | 匿名衡量数值特征1 | -0.057471 |
| numerical\_2 | anonymized measure | 匿名数值特征2 | -0.057471 |
| category\_1 | anonymized category | 匿名类别特征1 | N/Y |
| most\_recent\_sales\_range | Range of revenue (monetary units) in  last active month --> A > B > C > D > E | 在最近活跃月份的销售额等级 | A/B/C/D/E |
| most\_recent\_purchases\_range | Range of quantity of transactions in  last active month --> A > B > C > D > E | 在最近活跃月份的交易数量等级 | A/B/C/D/E |
| avg\_sales\_lag3 | Monthly average of revenue in last 3 months  divided by revenue in last active month | 过去3个月的月平均收入  除以上个活动月的收入 | -82.13 |
| avg\_purchases\_lag3 | Monthly average of transactions in last 3 months  divided by transactions in last active month | 最近3个月的月平均交易量  除以上个活动月的交易记录 | 9.6667 |
| active\_months\_lag3 | Quantity of active months within last 3 months | 最近3个月内活动月数 | 3 |
| avg\_sales\_lag6 | Monthly average of revenue in last 6 months  divided by revenue in last active month | 过去6个月的月平均收入  除以上个活动月的收入 | -0.74 |
| avg\_purchases\_lag6 | Monthly average of transactions in last 6 months  divided by transactions in last active month | 最近6个月的月平均交易量  除以上个活动月的交易记录 | 1.291667 |
| active\_months\_lag6 | Quantity of active months within last 6 months | 最近6个月内活动月数 | 6 |
| avg\_sales\_lag12 | Monthly average of revenue in last 12 months  divided by revenue in last active month | 过去12个月的月平均收入  除以上个活动月的收入 | -2.32 |
| avg\_purchases\_lag12 | Monthly average of transactions in last 12  months divided by transactions in last active month | 过去12个月的月平均交易量  月份除以上一个活动月份的交易记录 | 1.687500 |
| active\_months\_lag12 | Quantity of active months within last 12 months | 过去12个月内活动月数 | 12 |
| category\_4 | anonymized category | 匿名类别特征4 | Y/N |
| city\_id | City identifier (anonymized ) | 城市id(经过匿名处理) | 330 |
| state\_id | State identifier (anonymized ) | 州id(经过匿名处理) | 9 |
| category\_2 | anonymized category | 匿名类别特征2 | 5 |

表格3 merchants.csv的字段与含义

#### 8.2.2字段取值状况

在梳理完各个表格的字段信息含义以后，参赛者可以具体查看每个表格每个字段的具体取值状况，通常来说除了字段含义以外，要结合字段含义判定字段的取值类型，主要可以分为字符(object)和数值（int/float）两种，需要注意的是字段含义的离散与否与其字段取值的数值与否不存在必然的联系，离散型字段的取值可能是数值，如上述的city\_id这种，虽然它的取值都是数值，但并没有大小关系，数值型字段的取值也可能是字符，如上述的most\_recent\_sales\_range字段，它的取值类型虽然是字符，但可以明显感觉到它的取值存在大小关系。无论是什么类型字段，参赛者需要关心的主要是两点，一是缺失值状况，一是大概的取值范围与分布。离散特征的关注点在于特征值的数量分布，而数值特征则关注其取值范围以及异常值、离群点等。这里以本次赛题的目标列为例，目标列为连续值，可采用series的describe方法分析其取值范围和区间。有趣的是，若同时采用分析离散特征分布的value\_counts方法，参赛者可以惊喜的发现目标列有一个极端异常值-33.219281，占比约在1%，在后面的建模任务中参赛者会逐渐觉察这一发现的重要性与特殊性。

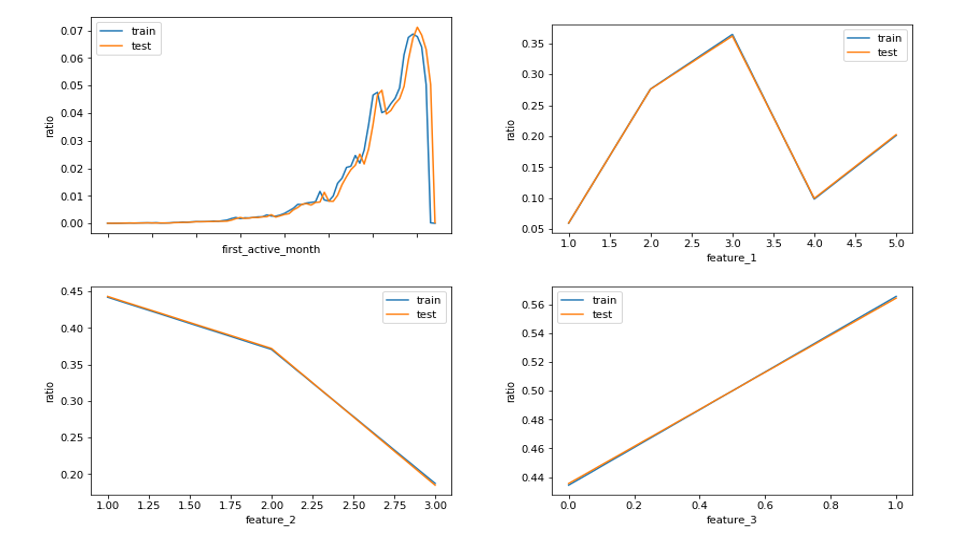
#### 8.2.3数据分布差异



机器学习领域有三个特别的数据集称谓，分别是训练集、验证集与测试集，模型从训练集中学习特征与标签之间的关联关系，同时利用验证集进行评估，以避免过拟合和欠拟合的发生，学到合适的程度后就可以应用到测试集上进行预测。要想模型的预测效果优异的其中一个前提就是验证集、测试集与训练集的数据分布相似，尤其是特征与标签的联合分布一致，这样模型学到的关联关系才是可以进行泛化的。验证集的选取根据建模任务的不同有多种选取方式，一般而言不涉及时间先后顺序的建模就可以随机划分一定比例的训练集和验证集，测试集由于无从得知标签也就是目标列，就只能通过一些特征的分布或是联合分布进行判断。本赛题将以train.csv与test.csv为例进行数据分布差异的探索分析。

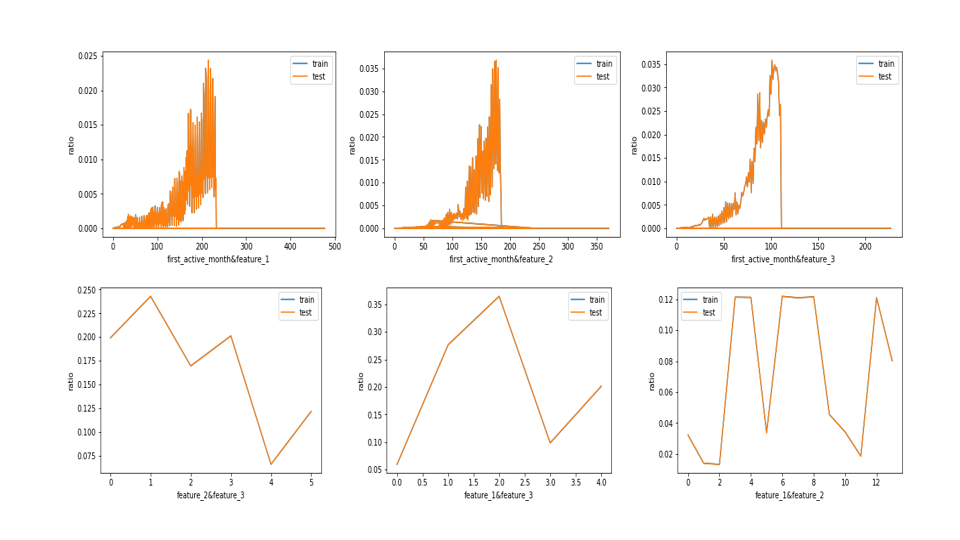
对train.csv和test.csv的几个特征进行单变量分布画图如下，

由此可以看出训练集与测试集在所有单变量上的绝对数量分布形状极其相似，需要进一步查看相对占比分布才能得到更准确的结论，继续对这4个字段进行单变量占比分布画图如下，



由此可得出初步结论：训练集与测试集在所有单变量上的相对占比分布形状基本一致，猜想训练集与测试集的生成方式一样，可继续验证联合分布以加强此猜想的事实依据。

需要注意的是，上面这里画图分析有一不严谨之处，即训练集与测试集的单变量取值范围可能不完全一样，由此两根线画在同一张图上有可能会出错，如发生偏移等，有兴趣的读者可自行验证二者的横坐标是否完全一样？如果不一样，运行同样的这段画图代码会发生什么？在下面的联合分布验证中，我们将会填补这一遗漏之处。

在查看多变量的联合分布时，通常来说可以使用散点图，但这里的四个特征均是离散特征，因此散点图不太适合继续延续上面画单变量图的思想，参赛者可以通过将两个变量拼到一起转变为单变量的分布。

修正上述遗漏后参赛者可以发现训练集与测试集的两变量联合分布也基本保持一致，由此基本可以判定，训练集与测似集的生成方式基本一摸一样，即测试集与训练集是同一批数据随机划分后的结果，有兴趣的参赛者可继续验证三变量和四变量分布。假定关于验证集与测试集的这一猜想成立，会极大地增添参赛者后续进行特征工程的信心，对建模方式也会有一个整体把握。

#### 8.2.4 表格关联关系

由上述探索可以看出train.csv和test.csv帮助参赛者明确了训练集与测试集，以及建模目标，而historical\_transactions.csv与new\_transactions.csv具有同样的字段，只是二者时间上有所区别，给参赛者提供了丰富的用户交易信息，merchant.csv则描述了商家的经营状况。参赛者需要结合商家基本信息表与用户交易记录表对用户的消费行为进行数据挖掘，找到经尽可能丰富的与目标列相关的信息，从而达到优异的预测效果。

#### 8.2.5 数据预处理

为了后续方便特征提取以及数据的整洁性，在探索性数据分析的同时参赛者可以进行相应的数据清洗，这方面针对不同数据的处理技巧多种多样，但目的都是为了后续的特征提取扫清障碍。这里只给出详细步骤，具体代码请见EDA.ipynb

##### train.csv&test.csv

这两个表格只有测试集的first\_active\_month字段有一个缺失值，总体来说一个缺失值的影响不大，且这个字段是object类型，因此需要对其进行编码处理，考虑到其实质上具有先后顺序的关系，采用字典排序进行编码即可。

##### merchants.csv

处理步骤如下：

1、根据业务含义划分离散字段category\_cols与连续字段numeric\_cols。

2、对非数值型的离散字段进行字典排序编码。

3、为了能够更方便统计，进行缺失值的处理，对离散字段统一用-1进行填充。

4、对离散型字段探查发现有正无穷值，这是特征提取以及模型所不能接受的，因此需要对无限值进行处理，此处采用最大值进行替换。

5、对于离散字段的缺失值处理方式也有多样，这里先使用平均值进行填充，后续有需要再进行优化处理。

6、去除与transaction交易记录表格重复的列，以及merchant\_id的重复记录。

##### new\_merchant\_transactions.csv&historical\_transactions.csv

处理步骤如下：

1、为了统一处理，首先拼接new和history两张表格，后续可以month\_lag>=0进行区分。

2、同样划分离散字段、连续字段以及时间字段。

3、可仿照merchant的处理方式对字符型的离散特征进行字典序编码以及缺失值填充。

4、进行时间段的处理，简单起见进行月份、日期的星期数（工作日与周末）、以及时间段（上午、下午、晚上、凌晨）的信息提取。

5、对新生成的购买月份离散字段进行字典序编码。

6、处理完transaction与merchant两个表格后，为了方便特征的统一计算将其merge合并，重新划分相应字段种类。

### 8.3 特征工程

经过基本的数据探索性分析之后，相信参赛者对数据以及对赛题任务有了一个良好的理解，本赛题的重点便是挖掘用户的各种交易行为与目标列的关系，进而达到良好的模型学习效果，准确预测测试集用户的忠诚度分数。因此这是一个关注信用卡用户局部消费偏好画像的题目，通过找到相似的训练集用户来类推测试集用户的忠诚度分数，进而对高价值人群进行区分，给商家与信用卡银行提供决策支持，同时也能够提升消费者的购物体验，因此特征工程可集中于用户的交易行为画像，即用户在各个维度上的购物行为量化，比如在最近一个月的消费金额与购买数量等等。

在针对评估用户价值的画像领域，有个经典的RFM理论，即Recent、Frequency和Money，结合前面的探索性分析，参赛者能够明确到这一理论的可行性，这里将使用购买数量模拟其Frequency频次，消费金额作为Money金钱。本赛题不仅在建模目标上具有广泛性，其数据结构也具有典型的特点，即主要利用用户的行为记录表格进行信息挖掘。由此这里将分别介绍两种办法进行特征的提取，一种是借助python的原生字典结构，一种则借助于pandas这一强大的数据处理工具。接下来将介绍利用字典结构进行通用特征的提取，借助pandas的统计函数进行业务特征的提取。

#### 8.3.1 通用特征

字典的键值结构很好地提供了很方便的映射关系，这里的特征提取可以以用户为第一层键值，特征字段作为第二层键值，统计完成后再转成Pandas.DataFrame格式，简单来说就是想知道用户在每个类别字段的每个取值下的购买数量与消费金额。

首先创建一个字典存储生成的统计特征，并给每个card\_id赋值

features = {}

card\_all = train['card\_id'].append(test['card\_id']).values.tolist()

for card in card\_all:

features[card] = {}

其次，记录好每个字段的索引方便按行处理的时候直接获取目标值，

columns = transaction.columns.tolist()

idx = columns.index('card\_id')

category\_cols\_index = [columns.index(col) for col in category\_cols]

numeric\_cols\_index = [columns.index(col) for col in numeric\_cols]

然后，按行进行相应字段的特征提取和更新，

# 记录运行时间

s = time.time()

num = 0

for i in range(transaction.shape[0]):

va = transaction.loc[i].values

card = va[idx]

for cate\_ind in category\_cols\_index:

for num\_ind in numeric\_cols\_index:

col\_name = '&'.join([columns[cate\_ind], va[cate\_ind], columns[num\_ind]])

features[card][col\_name] = features[card].get(col\_name, 0) + va[num\_ind]

num += 1

if num%1000000==0:

print(time.time()-s, "s")

del transaction

gc.collect()

最后，将字典转换成特征表格，

df = pd.DataFrame(features).T.reset\_index()

del features

cols = df.columns.tolist()

df.columns = ['card\_id'] + cols[1:]

表格生成后，就可以拼接训练集与测试集，从而进行后续的模型训练，为区别后续特征，此处特征集命名为dict。

#### 8.3.2 业务特征

基于字典结构的通用特征提取优势在于可以按行读取处理，速度上和内存上有一定保障，同时可以面面俱到的量化到每个子类下的用户行为，其缺点也比较明显，需要固定的数据结构，同时也会产生较高的维度。另一种方案就是使用pandas的groupby功能进行统计，这种方式简单很多，但同时对内存性能要求较高需要加载全部数据。需要注意的是这里为了符合pandas的统计需要，不再对缺失值以及离散字段的数据类型进行转化，同时增加两个用户购买的间隔特征。

transaction['purchase\_day\_diff'] = transaction.groupby("card\_id")['purchase\_day'].diff()

transaction['purchase\_month\_diff'] = transaction.groupby("card\_id")['purchase\_month'].diff()

首先，根据字段的种类设置相应想获取的统计量，并给定相应的字段列表，

aggs = {}

for col in numeric\_cols:

aggs[col] = ['nunique', 'mean', 'min', 'max','var','skew', 'sum']

for col in categorical\_cols:

aggs[col] = ['nunique']

aggs['card\_id'] = ['size', 'count']

cols = ['card\_id']

for key in aggs.keys():

cols.extend([key+'\_'+stat for stat in aggs[key]])

然后，针对history、new以及全时间段分别进行计算和统计，

df = transaction[transaction['month\_lag']<0].groupby('card\_id').agg(aggs).reset\_index()

df.columns = cols[:1] + [co+'\_hist' for co in cols[1:]]

df2 = transaction[transaction['month\_lag']>=0].groupby('card\_id').agg(aggs).reset\_index()

df2.columns = cols[:1] + [co+'\_new' for co in cols[1:]]

df = pd.merge(df, df2, how='left',on='card\_id')

df2 = transaction.groupby('card\_id').agg(aggs).reset\_index()

df2.columns = cols

df = pd.merge(df, df2, how='left',on='card\_id')

可以看出，利用groupby的方法统计出的特征数量会少很多，聚焦为用户各种行为的统计量，为区别后续特征，此处特征集命名为groupby。

#### 8.3.3 文本特征

除去上述常规的特征之外，结合本赛题的数据结构还有一类特征可以进行提取，它就是基于CountVector和自然语言处理领域的TF-IDF向量特征，不同于前面的dict和groupby特征，这里只针对部分离散字段进行词频统计，CountVector与dict部分的feature比较像，而TF-IDF则是对多变量联合分布的补充。

首先将相应字段处理成标准的输入格式，然后调用sklearn的相关方法进行计算，需要注意的是这部分特征采用的是scipy的sparse稀疏矩阵结构，因此在处理上与dict和groupby有所不同。

#### 8.3.4 特征选择

常见的特征选择方法主要有两种，一种是filter过滤式选择，另一种是wrapper包裹式选择，前者利用一些统计学上面的相关性系数进行过滤，后者通过模型评估过程中的特征重要性进行选择，一般来讲，特征选择的功能主要有提升模型训练速度与精度两个方面的考虑。

### 8.4 模型训练

在准备好基础特征后，参赛者就可以开始走训练与预测的全流程，为尽可能多的给参赛者介绍一些处理技巧，本节将会介绍三种模型的全流程，同时夹杂不同的特征选择与参数调优方法。

#### 8.4.1 RandomForest

首先是sklearn库里的随机森林模型，本模型的要素组成为特征选取dict和groupby两部分，特征选择采用基于Pearson相关系数计算的Filter方法取top300特征，参数寻优方面则同样使用sklearn的GridSearch网格搜索。模型全流程分为四个模块，即读取数据、特征选择、参数调优以及训练预测。

首先，读取指定的特征集并且进行拼接，

def read\_data(debug=True):

print("read\_data...")

NROWS = 10000 if debug else None

train\_dict = pd.read\_csv("preprocess/train\_dict.csv", nrows=NROWS)

test\_dict = pd.read\_csv("preprocess/test\_dict.csv", nrows=NROWS)

train\_groupby = pd.read\_csv("preprocess/train\_groupby.csv", nrows=NROWS)

test\_groupby = pd.read\_csv("preprocess/test\_groupby.csv", nrows=NROWS)

# 去除重复列

for co in train\_dict.columns:

if co in train\_groupby.columns and co!='card\_id':

del train\_groupby[co]

for co in test\_dict.columns:

if co in test\_groupby.columns and co!='card\_id':

del test\_groupby[co]

train = pd.merge(train\_dict, train\_groupby, how='left', on='card\_id').fillna(0)

test = pd.merge(test\_dict, test\_groupby, how='left', on='card\_id').fillna(0)

print("done")

return train, test

然后采用基于Pearson相关系数计算的Filter方法取top300特征进行选择，这里的300是随意取的一个数字，参赛者可以多试几个数量选择效果最佳的数量，

def feature\_select\_pearson(train, test):

print('feature\_select...')

features = train.columns.tolist()

features.remove("card\_id")

features.remove("target")

featureSelect = features[:]

# 去掉缺失值比例超过0.99的

for fea in features:

if train[fea].isnull().sum() / train.shape[0] >= 0.99:

featureSelect.remove(fea)

# 进行pearson相关性计算

corr = []

for fea in featureSelect:

corr.append(abs(train[[fea, 'target']].fillna(0).corr().values[0][1]))

se = pd.Series(corr, index=featureSelect).sort\_values(ascending=False)

feature\_select = ['card\_id'] + se[:300].index.tolist()

print('done')

return train[feature\_select + ['target']], test[feature\_select]

接着就是基于网格搜索的参数调优，网格搜索实际上是参数不同取值的交叉组合，这部分有可能需要根据调优结果多次手动迭代参数空间，当然每次迭代的时候在上一次最佳参数的基础上增加未搜索过的参数区域，

def param\_grid\_search(train):

print('param\_grid\_search')

features = train.columns.tolist()

features.remove("card\_id")

features.remove("target")

parameter\_space = {

"n\_estimators": [80],

"min\_samples\_leaf": [30],

"min\_samples\_split": [2],

"max\_depth": [9],

"max\_features": ["auto", 80]

}

print("Tuning hyper-parameters for mse")

clf = RandomForestRegressor(

criterion="mse",

min\_weight\_fraction\_leaf=0.,

max\_leaf\_nodes=None,

min\_impurity\_decrease=0.,

min\_impurity\_split=None,

bootstrap=True,

oob\_score=False,

n\_jobs=4,

random\_state=2020,

verbose=0,

warm\_start=False)

grid = GridSearchCV(clf, parameter\_space, cv=2, scoring="neg\_mean\_squared\_error")

grid.fit(train[features].values, train['target'].values)

print("best\_params\_:")

print(grid.best\_params\_)

means = grid.cv\_results\_["mean\_test\_score"]

stds = grid.cv\_results\_["std\_test\_score"]

for mean, std, params in zip(means, stds, grid.cv\_results\_["params"]):

print("%0.3f (+/-%0.03f) for %r"

% (mean, std \* 2, params))

return grid.best\_estimator\_

最后则根据参数调优的最佳结果进行模型训练与预测，

def train\_predict(train, test, best\_clf):

print('train\_predict...')

features = train.columns.tolist()

features.remove("card\_id")

features.remove("target")

prediction\_test = 0

cv\_score = []

prediction\_train = pd.Series()

kf = KFold(n\_splits=5, random\_state=2020, shuffle=True)

for train\_part\_index, eval\_index in kf.split(train[features], train['target']):

best\_clf.fit(train[features].loc[train\_part\_index].values, train['target'].loc[train\_part\_index].values)

prediction\_test += best\_clf.predict(test[features].values)

eval\_pre = best\_clf.predict(train[features].loc[eval\_index].values)

score = np.sqrt(mean\_squared\_error(train['target'].loc[eval\_index].values, eval\_pre))

cv\_score.append(score)

print(score)

prediction\_train = prediction\_train.append(pd.Series(best\_clf.predict(train[features].loc[eval\_index]),

index=eval\_index))

print(cv\_score, sum(cv\_score) / 5)

pd.Series(prediction\_train.sort\_index().values).to\_csv("preprocess/train\_randomforest.csv", index=False)

pd.Series(prediction\_test / 5).to\_csv("preprocess/test\_randomforest.csv", index=False)

test['target'] = prediction\_test / 5

test[['card\_id', 'target']].to\_csv("result/submission\_randomforest.csv", index=False)

return

这里的模型训练与预测结构采用的是五折交叉验证方法，一方面可以避免训练集的过拟合，另一方面也可以使得测试集的结果更加鲁棒，同时还有一个顺带的好处是生成可用于Stacking的特征，即训练集中作为验证集部分和测试集的模型预测结果，同样保留下来以备后续模型融合做准备。预测结果出来以后，进行提交测试所得分数如下，

|  |  |  |
| --- | --- | --- |
| Submission and Description | Private Score | Public Score |
| submission\_randomforest.csv  Dict+GroupBy + Pearson + GridSearch + RandomForest  CV：3.687109361199584 | 3.65493 | 3.75283 |

#### 8.4.2 LightGBM

接着采用相同的特征集进行LightGBM建模，不同的是在特征选择阶段采用Wrapper方法，参数调优则选择hyperopt框架，

首先，数据读取与前面一致，这里的Wrapper特征选择简单起见就采用特征重要性选取top300特征进行建模训练，同样地这个数字可以根据建模效果做改变，

def feature\_select\_wrapper(train, test):

print('feature\_select\_wrapper...')

label = 'target'

features = train.columns.tolist()

features.remove('card\_id')

features.remove('target')

# 配置模型的训练参数

params\_initial = {

'num\_leaves': 31,

'learning\_rate': 0.1,

'boosting': 'gbdt',

'min\_child\_samples': 20,

'bagging\_seed': 2020,

'bagging\_fraction': 0.7,

'bagging\_freq': 1,

'feature\_fraction': 0.7,

'max\_depth': -1,

'metric': 'rmse',

'reg\_alpha': 0,

'reg\_lambda': 1,

'objective': 'regression'

}

ESR = 30

NBR = 10000

VBE = 50

kf = KFold(n\_splits=5, random\_state=2020, shuffle=True)

fse = pd.Series(0, index=features)

for train\_part\_index, eval\_index in kf.split(train[features], train[label]):

# 模型训练

train\_part = lgb.Dataset(train[features].loc[train\_part\_index],

train[label].loc[train\_part\_index])

eval = lgb.Dataset(train[features].loc[eval\_index],

train[label].loc[eval\_index])

bst = lgb.train(params\_initial, train\_part, num\_boost\_round=NBR,

valid\_sets=[train\_part, eval],

valid\_names=['train', 'valid'],

early\_stopping\_rounds=ESR, verbose\_eval=VBE)

fse += pd.Series(bst.feature\_importance(), features)

feature\_select = ['card\_id'] + fse.sort\_values(ascending=False).index.tolist()[:300]

print('done')

return train[feature\_select + ['target']], test[feature\_select]

然后是hyperopt的参数调优框架，

def params\_append(params):

params['objective'] = 'regression'

params['metric'] = 'rmse'

params['bagging\_seed'] = 2020

return params

def param\_hyperopt(train):

label = 'target'

features = train.columns.tolist()

features.remove('card\_id')

features.remove('target')

train\_data = lgb.Dataset(train[features], train[label], silent=True)

def hyperopt\_objective(params):

params = params\_append(params)

print(params)

res = lgb.cv(params, train\_data, 1000,

nfold=2,

stratified=False,

shuffle=True,

metrics='rmse',

early\_stopping\_rounds=20,

verbose\_eval=False,

show\_stdv=False,

seed=2020)

return min(res['rmse-mean'])

params\_space = {

'learning\_rate': hp.uniform('learning\_rate', 1e-2, 5e-1),

'bagging\_fraction': hp.uniform('bagging\_fraction', 0.5, 1),

'feature\_fraction': hp.uniform('feature\_fraction', 0.5, 1),

'num\_leaves': hp.choice('num\_leaves', list(range(10, 300, 10))),

'reg\_alpha': hp.randint('reg\_alpha', 0, 10),

'reg\_lambda': hp.uniform('reg\_lambda', 0, 10),

'bagging\_freq': hp.randint('bagging\_freq', 1, 10),

'min\_child\_samples': hp.choice('min\_child\_samples', list(range(1, 30, 5)))

}

params\_best = fmin(

hyperopt\_objective,

space=params\_space,

algo=tpe.suggest,

max\_evals=30,

rstate=RandomState(2020))

return params\_best

不同与gridsearch输出含参数最佳分类器的是，hyperopt输出的是最佳参数字典，最后则按照同样的方式进入模型训练与预测，

def train\_predict(train, test, params):

label = 'target'

features = train.columns.tolist()

features.remove('card\_id')

features.remove('target')

params = params\_append(params)

kf = KFold(n\_splits=5, random\_state=2020, shuffle=True)

prediction\_test = 0

cv\_score = []

prediction\_train = pd.Series()

ESR = 30

NBR = 10000

VBE = 50

for train\_part\_index, eval\_index in kf.split(train[features], train[label]):

# 模型训练

train\_part = lgb.Dataset(train[features].loc[train\_part\_index],

train[label].loc[train\_part\_index])

eval = lgb.Dataset(train[features].loc[eval\_index],

train[label].loc[eval\_index])

bst = lgb.train(params, train\_part, num\_boost\_round=NBR,

valid\_sets=[train\_part, eval],

valid\_names=['train', 'valid'],

early\_stopping\_rounds=ESR, verbose\_eval=VBE)

prediction\_test += bst.predict(test[features])

prediction\_train = prediction\_train.append(pd.Series(bst.predict(train[features].loc[eval\_index]),

index=eval\_index))

eval\_pre = bst.predict(train[features].loc[eval\_index])

score = np.sqrt(mean\_squared\_error(train[label].loc[eval\_index].values, eval\_pre))

cv\_score.append(score)

print(cv\_score, sum(cv\_score) / 5)

pd.Series(prediction\_train.sort\_index().values).to\_csv("preprocess/train\_lightgbm.csv", index=False)

pd.Series(prediction\_test / 5).to\_csv("preprocess/test\_lightgbm.csv", index=False)

test['target'] = prediction\_test / 5

test[['card\_id', 'target']].to\_csv("result/submission\_lightgbm.csv", index=False)

return

同样地保存三个文件，并记录下CV得分与线上提交得分，

|  |  |  |
| --- | --- | --- |
| Submission and Description | Private Score | Public Score |
| submission\_lightgbm.csv  Dict+GroupBy + Wrapper + Hyperopt + LightGBM  CV：3.677305261912256 | 3.64490 | 3.73817 |

#### 8.4.3 XGBoost

前面两个模型都只用到了dict和groupby两组特征，接下来的XGBoost模型将尝试把nlp特征一起加入进行训练，同时略过特征选择考虑使用全集进行建模，调优框架则换成beyesian.

首先依然是读取数据，不同的是需要与nlp特征合并成sparse稀疏矩阵，

def read\_data(debug=True):

print("read\_data...")

NROWS = 10000 if debug else None

train\_dict = pd.read\_csv("preprocess/train\_dict.csv", nrows=NROWS)

test\_dict = pd.read\_csv("preprocess/test\_dict.csv", nrows=NROWS)

train\_groupby = pd.read\_csv("preprocess/train\_groupby.csv", nrows=NROWS)

test\_groupby = pd.read\_csv("preprocess/test\_groupby.csv", nrows=NROWS)

# 去除重复列

for co in train\_dict.columns:

if co in train\_groupby.columns and co!='card\_id':

del train\_groupby[co]

for co in test\_dict.columns:

if co in test\_groupby.columns and co!='card\_id':

del test\_groupby[co]

train = pd.merge(train\_dict, train\_groupby, how='left', on='card\_id').fillna(0)

test = pd.merge(test\_dict, test\_groupby, how='left', on='card\_id').fillna(0)

features = train.columns.tolist()

features.remove('card\_id')

features.remove('target')

train\_x = sparse.load\_npz("preprocess/train\_nlp.npz")

test\_x = sparse.load\_npz("preprocess/test\_nlp.npz")

train\_x = sparse.hstack((train\_x, train[features])).tocsr()

test\_x = sparse.hstack((test\_x, test[features])).tocsr()

print("done")

return train\_x, test\_x

然后是参数调优阶段，不同于hyperopt的是，beyesian以最大化评估分值进行优化，而rmse应是越小越好，因此采用负值的rmse作为优化目标，

def params\_append(params):

params['objective'] = 'reg:squarederror'

params['eval\_metric'] = 'rmse'

params["min\_child\_weight"] = int(params["min\_child\_weight"])

params['max\_depth'] = int(params['max\_depth'])

return params

def param\_beyesian(train):

train\_y = pd.read\_csv("data/train.csv")['target'].values

train\_data = xgb.DMatrix(train, train\_y, silent=True)

def xgb\_cv(colsample\_bytree, subsample, min\_child\_weight, max\_depth,

reg\_alpha, eta,

reg\_lambda):

params = {'objective': 'reg:squarederror',

'early\_stopping\_round': 50,

'eval\_metric': 'rmse'}

params['colsample\_bytree'] = max(min(colsample\_bytree, 1), 0)

params['subsample'] = max(min(subsample, 1), 0)

params["min\_child\_weight"] = int(min\_child\_weight)

params['max\_depth'] = int(max\_depth)

params['eta'] = float(eta)

params['reg\_alpha'] = max(reg\_alpha, 0)

params['reg\_lambda'] = max(reg\_lambda, 0)

print(params)

cv\_result = xgb.cv(params, train\_data,

num\_boost\_round=1000,

nfold=2, seed=2,

stratified=False,

shuffle=True,

early\_stopping\_rounds=30,

verbose\_eval=False)

return -min(cv\_result['test-rmse-mean'])

xgb\_bo = BayesianOptimization(

xgb\_cv,

{'colsample\_bytree': (0.5, 1),

'subsample': (0.5, 1),

'min\_child\_weight': (1, 30),

'max\_depth': (5, 12),

'reg\_alpha': (0, 5),

'eta':(0.02, 0.2),

'reg\_lambda': (0, 5)}

)

xgb\_bo.maximize(init\_points=21, n\_iter=5) # init\_points表示初始点，n\_iter代表迭代次数（即采样数）

print(xgb\_bo.max['target'], xgb\_bo.max['params'])

return xgb\_bo.max['params']

最后，按照同样的方式进行五折交叉训练验证，同时保存三个关键结果，

def train\_predict(train, test, params):

train\_y = pd.read\_csv("data/train.csv")['target']

test\_data = xgb.DMatrix(test)

params = params\_append(params)

kf = KFold(n\_splits=5, random\_state=2020, shuffle=True)

prediction\_test = 0

cv\_score = []

prediction\_train = pd.Series()

ESR = 30

NBR = 10000

VBE = 50

for train\_part\_index, eval\_index in kf.split(train, train\_y):

# 模型训练

train\_part = xgb.DMatrix(train.tocsr()[train\_part\_index, :],

train\_y.loc[train\_part\_index])

eval = xgb.DMatrix(train.tocsr()[eval\_index, :],

train\_y.loc[eval\_index])

bst = xgb.train(params, train\_part, NBR, [(train\_part, 'train'),

(eval, 'eval')], verbose\_eval=VBE,

maximize=False, early\_stopping\_rounds=ESR, )

prediction\_test += bst.predict(test\_data)

eval\_pre = bst.predict(eval)

prediction\_train = prediction\_train.append(pd.Series(eval\_pre, index=eval\_index))

score = np.sqrt(mean\_squared\_error(train\_y.loc[eval\_index].values, eval\_pre))

cv\_score.append(score)

print(cv\_score, sum(cv\_score) / 5)

pd.Series(prediction\_train.sort\_index().values).to\_csv("preprocess/train\_xgboost.csv", index=False)

pd.Series(prediction\_test / 5).to\_csv("preprocess/test\_xgboost.csv", index=False)

test['target'] = prediction\_test / 5

test[['card\_id', 'target']].to\_csv("result/submission\_xgboost.csv", index=False)

return

### 8.5 模型融合

在经历完特征工程与模型训练阶段后，参赛者可能会发现自己的得分仍旧不尽人意，

本章为了尽可能简而泛的介绍机器学习竞赛的相关技巧，多使用比较通用或者比赛圈常说的一把梭办法，并没有针对此赛题做极致细化的方案，因为那并不是本书想要达到的目的，因此单模型的分数显得并不高，本节将分别尝试模型加权融合与Stacking融合的办法进行分数的提升。这里还有一点想要告诉参赛者的是，机器学习竞赛中团队与开源的力量极其强大，一个人的思路和时间精力往往有限，而不同人之间的建模方法又有着极大的差异，因此往往能够带来极大的融合收益。此外，大部分比赛尤其是Kaggle的竞赛，允许参赛者自由讨论甚至开源代码，这同样是上分的一个很好的资源，可以利用参赛者自身的算法结合开源方案，从而融合取得更好的分数。

#### 8.5.1 加权融合

理论部分本书已经将结果加权融合的原理，可按照分数与相关性赋予权值进行提交，结果如下，

|  |  |  |
| --- | --- | --- |
| Submission and Description | Private Score | Public Score |
| data['randomforest']\*0.2+data['lightgbm']\*0.3 + data['xgboost']\*0.5 | 3.63741 | 3.73135 |

#### 8.5.2 Stacking融合

在跑前面的三个模型时，顺带生成了对应模型的Stacking特征，也就是训练集与测试集的模型预测结果，这个结果可以看作是特征集信息的一个提炼压缩，结合开源的一个较高分数的方案，注意此开源代码有几处bug，更改后不影响使用。将XGBoost的Stacking特征加入其中训练可得到一个分数尚可的模型，其结果如下，

|  |  |  |
| --- | --- | --- |
| Submission and Description | Private Score | Public Score |
| simple-lightgbm-without-blending  + XGBoost Stacking | 3.60871 | 3.68825 |

这里由于篇幅有限只罗列了分数高的stacking结果以及分数低的加权融合结果，有兴趣的读者可以自己尝试对比在同等条件下stacking与加权融合的结果，由于本次赛题为回归且存在异常值的情况下，stacking的模型融合方法是要优于直接加权融合的。

### 8.6 赛题总结

#### 8.6.1 更多方案

##### 异常值二分类模型

在Kaggle的Discussion讨论区，有人发现建模的rmse损失一半以上都是极小异常值-33.219281造成，因此可以考虑将此标签的样本去除进行一个模型训练，同时再使用这个异常值建立一个二分类模型，这也是冠军最后采用的方案，主要是以如下公式综合两部分的预测结果，

train['final'] = train['binpredict'](-33.21928)+(1-train['binpredict'])train['no\_outlier']

这也比较好理解，利用二分类的概率乘以异常值，再加上其在无异常值下的预测结果与非异常值的概率。

##### 改异常值后处理

这也是针对异常值的直接方式，在预测完结果后，直接对其中的最小值进行改值后处理，不过这种方式比较有风险，且没有实际的利用价值，因此不做推荐，可以尝试看看了解一下效果。

##### 用户画像深挖

本章的特征工程都相对简单粗糙，没有足够精细化到比如节假日，以及用户近期行为。

#### 8.6.2 知识点梳理

##### 特征工程

本章主要用到的特征有三类，即RFM、Groupby以及nlp特征，且分别使用字典、pandas.dataframe以及稀疏矩阵进行提取，RFM模型只模拟利用了F与M，对于recent方面的信息有待读者进行思考调研，某种程度上，第二种groupby方案包含了部分R信息，而nlp里面主要采用了CountVector和TF-IDF，需要注意的是三者各自需要的数据结构与类型有所不同，所以导致前期预处理有些许差别。而特征选择方面，尝试了基于pearson相关系数的Filter方法以及基于模型特征重要性的wrapper方法，最后进入建模时也有采用pandas.dataframe与scipy.sparse矩阵两种方式。

##### 参数寻优

本章实战使用了三种参数调优框架，分别是GridSearch、hyperopt以及beyesian，各有特点，为了快速地寻到较为不错的参数组合，参赛者可以只随机采样部分数据进行调试。

##### 建模思路

整体来讲本赛题是一个比较标准的数据挖掘与机器学习建模，训练集与测试集的分布高度耦合，使得参赛者只需要专注于刻画用户的自身消费行为，然后通过机器学习算法进行训练预测。

#### 8.6.3 延伸学习

鉴于篇幅有限，本章还没来得及介绍的还有CatBoost于Word2Vec两种算法，CatBoost也是一种决策树模型，特别的是它可以直接支持离散字段与文本字段的建模计算，省去了大量的预处理时间。而Word2Vec也是nlp领域一个经典的算法，同样是将文本处理成模型可以理解的数值向量。