队伍编号	MCB2100593
赛道	A

基于 Stacking 模型的交易周期预测及约束规划调价策略的研究

摘 要

随着机动车数量的增长，二手车交易的市场蓬勃发展。为了促进二手车交易平台健康发展，帮助平台解决预测交易周期以及增大销售利润的问题。本文建立基于 Stacking 技术和贝叶斯优化的模型，将预测模型进行融合，提出了更准确的二手车的交易周期预测策略；建立基于多元非线性规划的调价模型，对合适的车辆的调价时间和幅度进行研究。

针对问题一，主要需要确定影响交易周期的因素并建立合适的预测模型。本文建立了基于 Stacking 技术的 XGBoost、LightGBM 和 CatBoost 融合模型来进行预测，并运用贝叶斯优化进行模型超参数调优。首先，我们对数据进行异常值剔除、正态转换和缺省值填补的预处理，将特征转化为便于统计分析的形式；另外，构造评估方法、降价次数、降价比率等反映车况和降价情况的新特征，通过递归特征消除法进行特征筛选，选取与交易周期有密切关系的 31 个特征并做检验。基于所选特征利用 Stacking 融合技术的模型对交易周期进行预测，在此过程中用 HERO 算法对基模型 XGBoost、LightGBM 和 CatBoost 进行贝叶斯优化，最终得到训练模型。依据此训练模型在验证集上所得预估交易周期的平均绝对误差为 10.45，且优于单一模型评测效果。

针对问题二，主要需要以最小化成本、最大化毛利润为目标，建立调价的决策模型。因此我们需研究决定门店利润的关键因素以及价格对其的影响作用，基于此建立了多元非线性规划模型。设置门店利润为目标函数，具体表达为售车利润与资金占用成本和停车位占用成本的差，将二手车价格对这三部分的影响进行量化表示。另外考虑库存约束及交易约束，将问题转化为有约束的多元非线性规划问题。以一天为决策周期，求解此规划模型即得到目标函数取得最大值时的各车辆价格，与最初定价对比得到调价方案。

关键词：Stacking 技术、XGBoost 模型、LightGBM 模型、CatBoost 模型、HEBO 算法、多元非线性规划

目录

一、问题重述

1.1 问题的背景

随着社会的进步和人民生活水平的不断提高，我国汽车市场不断增大，机动车数量不断增长，人均保有量也随之增加，机动车以“二手车”形式在流通环节，包括二手车收车、二手车拍卖、二手车零售、二手车置换等环节的流通需求越来越大。2018 年，全国二手车交易量为 1382.19 万辆。2019 年二手车的交易量达到 1492.28 万辆，在 2018 年的基础上总量增长了 110 万辆。2020 年，即便受到疫情影响，截至 2020 年 11 月份，我国的二手车市场仍然在 11 个月中达到了 1263.4 万辆，并且在 2020 年的下半年，连续 4 个月我国的二手车交易增长率都达到了 10%以上。

在此背景下，二手车交易平台也随之兴起，成为我国整个汽车销售行业中新的经济增长亮点。但是由于平台体系尚不完善、金融扶持力度不足等问题，二手车平台的发展也面临着挑战。在具体交易过程中，平台一般通过互联网等线上渠道获取用户线索，线下实体门店对外展销和售卖，俗称 O2O 门店模式。不同于一般商品，二手车的收车和售卖具有不确定性，另外由于收购及贮存二手车需要消耗大量成本，对库存车辆进行合理控制是经营者获取利润、避免损失的关键。而合理的库存管理基于对车辆市场的预估，因此基于二手车特征对其成交周期做出准确预估对平台经营具有重要的实际意义。

1.2 问题的提出

问题 1：结合附件 4“门店交易训练数据”对车辆的成交周期（从车辆上架到成交的时间长度，单位：天）进行分析，在初赛所挖掘的成交周期关键因素的基础上，建立交易周期预测模型，并对附件 5“门店交易验证数据”进行预测，并将预测结果保存在附件 6“门店交易模型结果”文件中。

问题 2：为了保障成本最小化、门店销售利润最大化，经营者需要通过调整价格对门店售卖过程中的库存进行管理，即根据在库车辆和新收车辆情况对车辆进行调价，使得热销车辆以更合适的价格成交，保全门店利润，同时也要对滞销车辆进行降价促销，以避免更大的损失。构建门店经营模型并基于此，决策何时对某个车辆进行调价，以及调整多大幅度。

二、问题的分析

2.1 问题一的分析

问题一主要需要我们提取附件 1 和附件 4 中估价训练数据及门店交易数据的车辆特征，对特征进行构造和筛选，基于此预测交易周期。因此我们对各个特征进行 EDA，并基于分析结果对题目进行求解。首先，由于有些特征严重长尾或缺省值过多，不利于预测交易周期，因此我们将这些特征舍弃。其次，我们利用 label encoding 方法将离散特征转化为连续特征进行处理。引入评估方法、降价次数、降价比率等反映车况和降价情况的新特征。通过相关性分析和 XGBoost 算法进行特征筛选，按照从低到高的水平逐次筛除检验模型在验证集上的效果从而得到预测交易周期所需的重要特征。然后利用 Stacking 技术将经过贝叶斯优化的 XGBoost、LightGBM 和 CatBoost 三种模型进行融合，并基于所选特征进行模型训练，并对交易周期进行预估。

除此之外，我们没有直接对成交周期进行预测，而是将预测任务转换为预测最后一次调整价格后，还需要多少天，二手车才能被售出。

2.2 问题二的分析

问题二主要需要我们确定决定门店利润的关键因素以及价格对其影响作用，将其量化表示为非线性规划问题。首先，将门店利润（售车利润与资金占用成本和停车位占用成本的差）最大化设置为目标函数，确定二手车价格对目标函数中其他变量之间的关系。其次，以一天为决策周期，对门店利润进行计算，并给出当天某车是否调价以及调整多大幅度的调价方案。另外，在求解此规划问题时我们需考虑考虑库存约束及交易约束，即库存车辆不超过停车场容量，售出车辆不超过库存车辆。

三、模型的假设

1、假设所调查的二手车交易样本数据都比较准确；

2、假设二手车交易周期不受第三方平台自身因素的影响；

3、假设买家和卖家是在公平、公正的条件下进行买卖，即交易不受数据集特征之外的其余因素影响。

4、假设停车位有限，且一辆车占用一个停车位。

四、符号说明

符号	符号说明
m_t	第t天的收车数量
n_t	第t天的售车数量
p_{ti}	第t天收入的第i辆车的收车价
p_{ti}'	第t天卖出的第i辆车的售车价
M_t	第t天支出的资金占用成本
S_t	第t天支出的停车位占用成本
R_t	第t天售车所得利润
Z_t	第t天门店总利润

五、基于 Stacking 模型的二手车交易周期预测

在本问题中，利用数据 EDA 方法对二手车的 36 种变量数据进行处理，并对二手车信息变量间的相互关系以及变量与二手车估价之间的关系进行探索；在此基础上构造特征并选择 Stacking 模型将经过贝叶斯优化的 XGBoost、LightGBM 和 CatBoost 三种模型融合来进行交易周期预测。

利用 EDA 对已有的数据在尽量少的先验假定下进行探索，通过作图、制表、方程拟合、计算特征量等手段探索数据的结构和规律能够帮助了解数据集，对数据集进行处理并获取重要特征，从而使得数据集的结构和特征集让所研究的预测问题更加可靠。

5.1 数据处理

基于 python 中 pandas 库对附件一中的数据进行读取，对数据进行以下预处理：

1、将数据信息按照类别特征和数值特征进行分类。

2、剔除异常数据，并将所给数据进行类型的转化，方便之后的操作。其中剔除的数据包括一条 target > 10000 的数据以及 target 过低的数据以及 target 比新车价高的数据。剔除原因为：target 过低的数据是一些车况很差的数据，不多见；target 比新车价高的数

据有可能是因为车为绝版车，不多见。

3、假设预测任务中的二手车都会被售出，我们将训练集中无法卖出的二手车样本全部剔除，共计 2000 个样本。

4、对数据进行正态转换。我们采用三种函数对数据进行拟合，比较拟合效果，效果如下图：

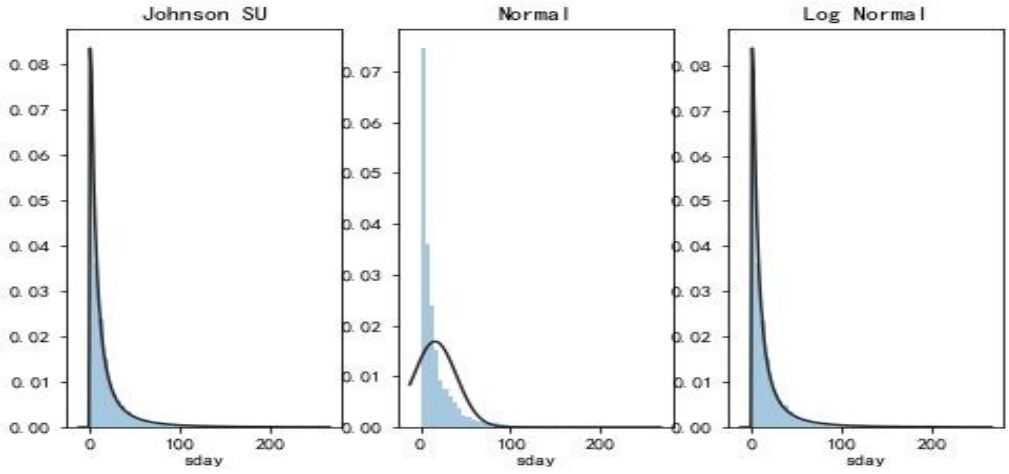

图 1 左图（Johnson SU 函数拟合） 中图（正态函数拟合） 右图（log 函数拟合）

将价格的总体分布画出来后，我们发现数据分布存在右偏，说明存在过大的极端值，因此需要对数据中的过大的价格值进行处理。我们通过上图发现利用 Johnson SU 函数对数据进行转换效果较好。转换后的数据如下图：

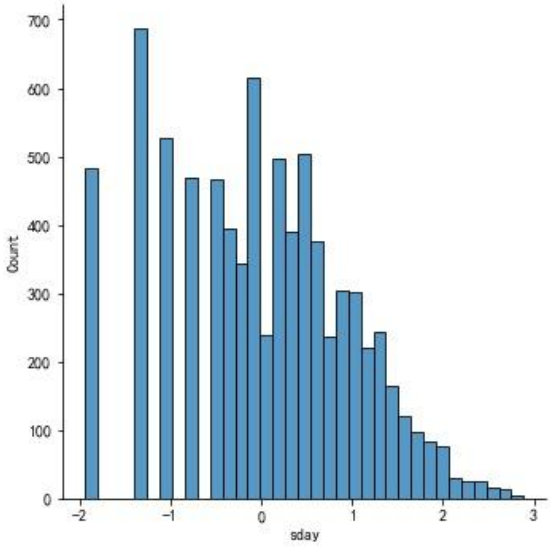

图 2 价格数据正态转换图像

4、根据车型 id 和车系 id 相同的判断准则，对数据中的缺省值，用对应的样本的众数进行填充。

5、对附件 4：门店交易训练数据.txt 中的数据按照展销时间进行排序，并按照 7：3 的比例进行划分，构造训练数据集和验证数据集，训练集用于模型训练，验证集用于模型效果的评估。

5.2 特征构造

取出二手车的数字特征（能够刻画随机变量某些方面的性质特征的量）和类别特征（有限选项内取值的特征），对严重长尾和缺省值过多的特征进行舍弃。具体特征的舍弃如下：

表 1 预测模型中舍弃的特征

舍弃原因	特征
严重长尾	车辆颜色, 燃油类型, 载客人数, Feature_1, Feature_3
缺省值过多	Feature_4, Feature_7, Feature_10, Feature_15

为了处理二手车的离散型特征，可以采用 one-hot encoding 和 label encoding 两种转化方法。由于已知离散特征种类较多，转化后参数维度太大，不利于模型训练与预测，因此本文采用 label encoding 方法。通过提取不同变量下的交易价格数值特征构造所在城市交易价格总值（所在城市_sum）、车辆品牌交易价格均值（品牌_mean）、车辆品牌交易价格最大值（品牌_max）等特征。

另外，引入降价比例、降价比率和降价次数作为反映车辆降价情况的新特征，其中降价比例为售价/最初定价，降价比率为售价/市场新车价；引入两个评估方法作为能够反映车辆性能的新特征：

评估方法 1：使用年限计算法。将新车使用 10 年报废（按照老的规定算）视为 100 分，把 15%作为不折旧的固定部分为残值，其余 85%为浮动折旧值。可分三个阶段：3 年—4 年—3 年来折旧，折旧率分别为 11%、10%和 9%。前三年每年折 11%，总折 28%（85%×33%）。然后加残值，构成了折旧值，计算为：

评估价=市场现行新车售价×[15%（不动残值）+85%（浮动值）×（分阶段折旧率）]+评估值。

评估方法 2：里程分段“54321 法”。具体为：假设一部车有效寿命 30 万公里，将其分为 5 段，每段 6 万公里，每段价值依序为新车价的 5/15、4/15、3/15、2/15、1/15。用此方法评估二手车价格，按照公里数分段，得出每年的折损价格，最后做减到相应用新车价格减掉公里数对应的价格部分即可，详细实现方式见代码。

构造后的特征类别如下：

表 2 关于二手车成交周期的特征

序号	特征	序号	特征
1	年款	17	品牌_count
2	排量	18	品牌_max
3	变速箱	19	品牌_min
4	载客人数	20	品牌_median
5	新车价	21	品牌_sum
6	里程	22	厂商类型_mean
7	注册年份	23	国标码-mean
8	Feature_2	24	年款_mean
9	Feature_5	25	变速箱_mean
10	Feature_13	26	Feature_2_mean
11	Feature_12_num_sqrt	27	Feature_8_mean
12	Feature_12_0	28	评估方法 1
13	Feature_12_1	29	评估方法 2
14	Feature_12_2	30	降价次数
15	所在城市_sum	31	降价比率
16	品牌_mean	32	降价比例

由于不同特征之间的量纲差异过大时，样本之间的相似度评估结果将会受到较大的影响，导致对样本相似度的计算存在偏差，所以采用最小最大化处理，将数据统一映射到[0,1]区间上，归一化公式如下：

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

其中x_{min}指样本中的最小值，x_{max}指样本的最大值。

5.3 特征筛选

首先对构造后得到的连续特征进行相关性分析：计算变量间的 Pearson 系数，正值表示正相关，负值表示负相关，绝对值越大表示线性相关程度越高。

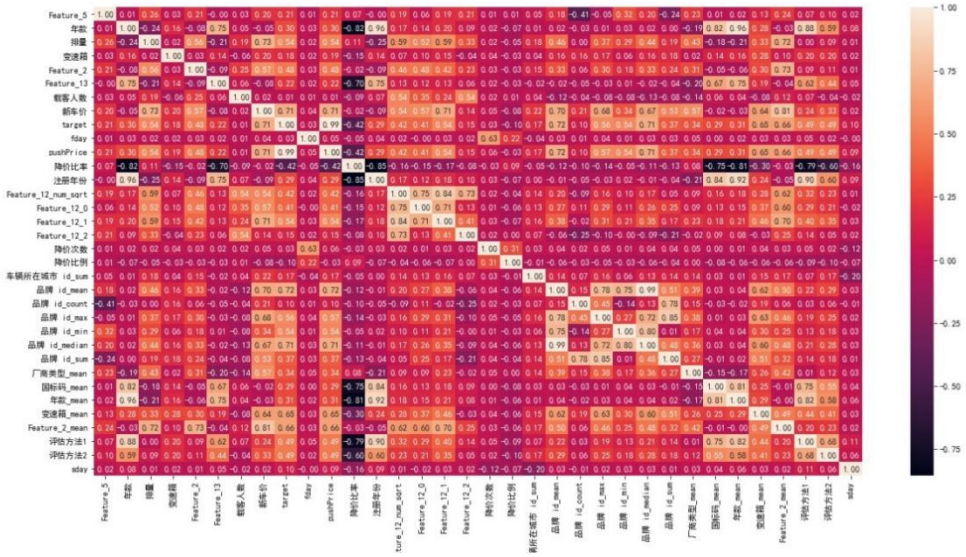

图 3 交易周期连续特征协方差矩阵热力图

根据上图我们可以发现汽车的交易周期与新车价、feature_12_1、品牌 id_mean、品牌 id_median、变速箱_mean、feature_2_mean、评估方法 1 的相关性较强。

然后利用 XGBoost 算法（在 4.4.1 进行介绍）获取特征重要性，依据模型在验证集上的效果对特征按照重要性程度从低到高进行排序，结果如下图：

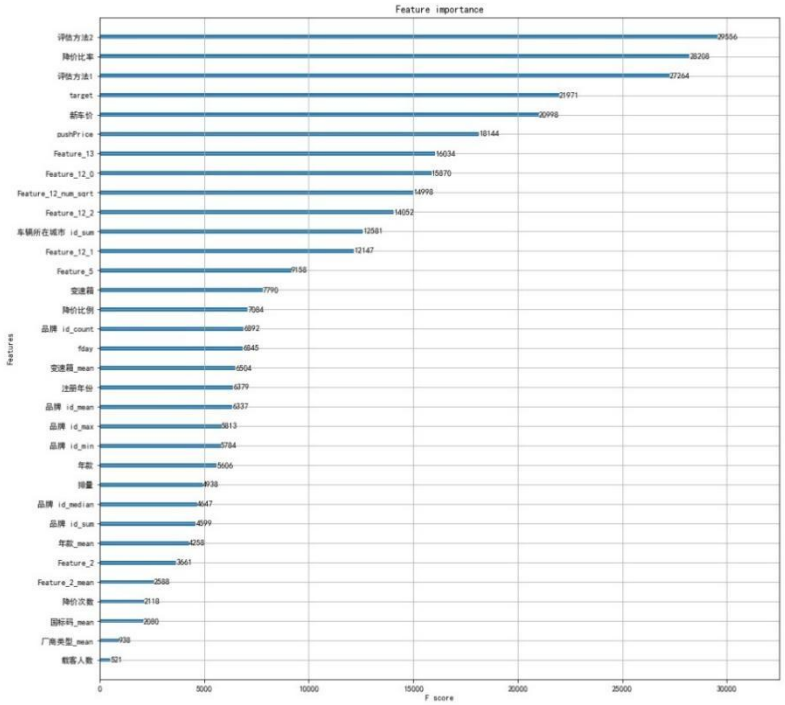

图 4 交易周期特征重要性排序图

由上图我们可以发现评估价格 2 特征的重要性最大，其次为降价比率、评估方法 1、二手车最后一次定价……利用模型训练，我们根据重要性从低到高的顺序删除对应的因素，继而检验我们所选的因素与正确的预测值的差距，数据如下图：

```
Feature Name=评估方法1 Thresh=0.027, n=33, Accuracy: 10.62
Feature Name=target Thresh=0.027, n=32, Accuracy: 10.67
Feature Name=Feature_12_2 Thresh=0.028, n=31, Accuracy: 10.66
Feature Name=pushPrice Thresh=0.028, n=30, Accuracy: 10.66
```

图 5 删除相应因素之后的预测准确率

根据上述的数据，我们可以发现即使删除重要性水平较低的因素，预测价格的准确值也会明显变小，因此我们选择图 4 的所有特征作为预测价格的主要特征。

5.4 模型选择

5.4.1 XGBoost 模型

XGBoost 算法是一种以 CART 决策树模型为基础的集成学习方法，通过构建多棵 CART 决策树来提供预测模型的准确性，最后将每轮训练得到决策树的预测结果求和得到最终的预测值[1]。其第 t 棵 CART 决策树的目标函数定义如下[2]：

$$L^{(t)} = \sum_{i=1}^{n} l\left(y_i, y_i^{'(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{2}$$

其中：

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\|w\|^2 \tag{3}$$

n 为样本总数，$y_i^{'(t-1)}$为第 t-1 个学习器对 i 样本的预测值，$f_t(x_i)$为新加入的第 t 个学习器，$\Omega(f_t)$为正则项。

运用泰勒公式对目标函数进行展开

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2 \tag{4}$$

损失函数的二阶泰勒展开结果为

$$\sum_i L\left(y_i, \hat{y}_i^{t-1} + f_t(x_i)\right) = \sum_i \left[L\left(y_i, \hat{y}_i^{t-1}\right) + L'\left(y_i, \hat{y}_i^{t-1}\right)t(x_i) + \frac{1}{2}L''\left(\left(y_i, \hat{y}_i^{t-1}\right)f_t^2(x_i)\right)\right] \tag{5}$$

用g_i记为第 i 个样本损失函数的一阶导数，h_i记为第 i 个样本损失函数的二阶导数

$$g_i = L'\left(y_i, \hat{y}_i^{t-1}\right) \tag{6}$$

$$h_i = L''\left(y_i, \hat{y}_i^{t-1}\right)$$

在进行特征节点选择时，遍历训练集所有特征变量的取值，用节点分裂前的目标函数值减去分裂后 2 个叶子节点的目标函数值之和，计算增益值，得到树模型最优的切分点，其中增益值计算式为：

$$L_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i+\lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i+\lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i+\lambda}\right] - \gamma \tag{7}$$

5.4.2 LightGBM 模型

LightGBM 是一个高效实现 GBDT 算法的框架。它的原理与 XGBoost 相似，但其在处理数据上的速度是 XGB 模型的 10 倍，内存占用率也仅仅为 XGB 模型的 1/6，且准确率也有提升。LGB 模型采用直方图算法遍历寻找最优的分割点，并且用直方图做差加速从而将遍历速度提升一倍，而且还用带深度限制的 Leaf-wise 的叶子生长算法在保证高效率的同时防止过拟合[3]。

5.4.3 CatBoost 模型

CatBoost 是一种以对称决策树为基学习器，参数较少、支持类别型变量和高准确性的 GBDT 框架。通过采用排序提升（Ordered Boosting）的方式替代传统算法中的梯度估计方法，进而减小梯度估计的偏差，因此能够高效合理地处理类别型特征，解决以往 GBDT 框架的机器学习算法中常出现的梯度偏差和预测偏移问题，从而减少了过拟合的发生，提高了算法的泛化能力[4]。

5.4.4 Stacking 技术

Stacking 是一种集成学习技术，通过元分类器或元回归器聚合多个分类或回归模型。基础层模型基于完整的训练集进行训练，元模型基于基础层模型的输出进行训练[5]。本文采用 Stacking 融合模型预测过程，以 XGBoost、LightGBM、Catboost 三种单一模型为基模型进行训练预测；以 linear regression 作为元模型进行次级训练预测。如下图所示。

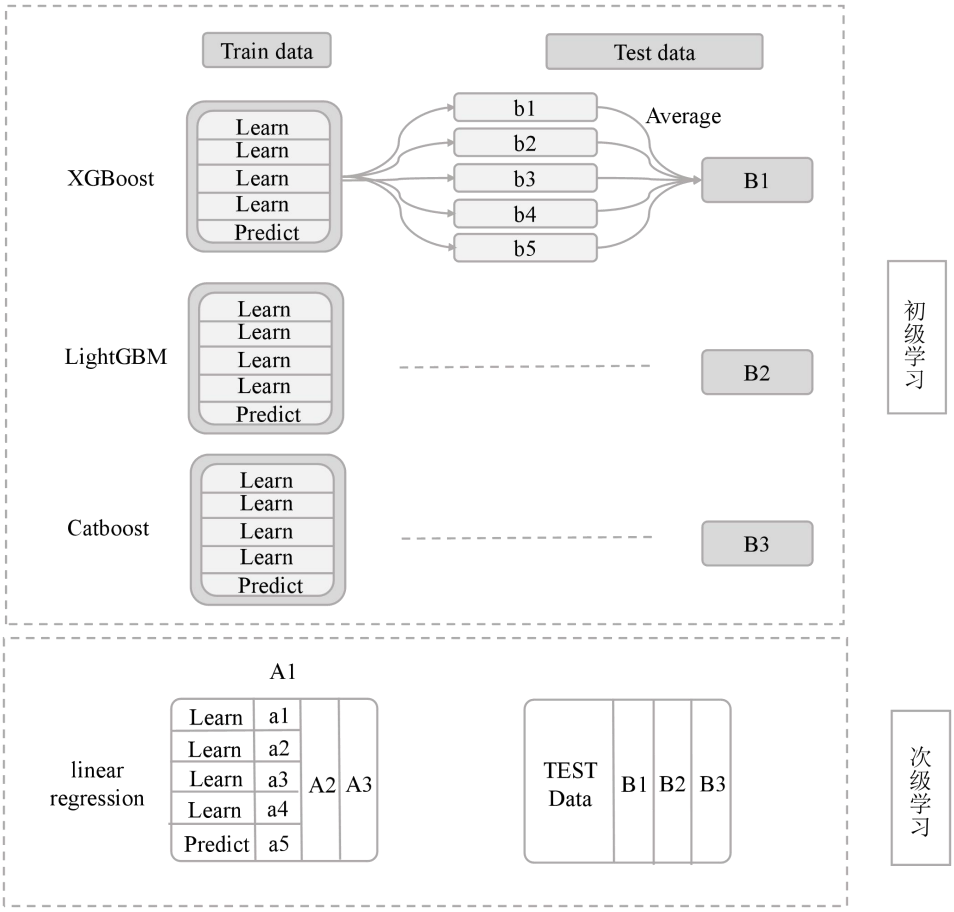

图 6 交叉验证的 Stacking 融合模型图

其中初级学习过程：假设已有数据集$D = \{x_i, y_i\}(= 1, \cdots n)$，其中$x_i$为输入的样本特征，$y_i$为预测标签。将数据集$D$划分为训练集和测试集，基础层模型$M_k$基于五折划分的训练集进行模型训练得到基学习器$L_k$,$k = 1,2,3$。基模型$L_k$分别基于划分测试集和原始测试集进行预测，输出结果$A_k$和$B_k$，$A_k$和$B_k$构成了新的数据样本，其中$k = 1,2,3$。次级学习过程：元模型将初级学习过程的预测输出作为输入数据进行学习得到元学习器L_{new}，利用元学习器对数据学习预测得到预测结果。即：

$$L_{predicted} = L_k(L_1(x_i), L_2(x_i), L_3(x_i)) \tag{8}$$

本文根据所给数据进行了 XGB、LGB、CAT 模型以及 Stacking 技术的处理，结果如下表所示：

表 3 模型预测效果比较

模型	评测标准（$Mape$）
XGBoost	10.62
LightGBM	10.55
CatBoost	10.58

对比分析可以得出，Stacking 融合模型的预测效果均高于单一模型的预测效果，相比于单一模型，Stacking 融合模型拥有更好的预测精度和模型效果。

5.5 基于贝叶斯框架的超参数调优

贝叶斯超参数优化算法的基本思想是根据过去目标的评估结果建立替代函数，以找到目标函数的最小值。在此过程中建立的替代函数比原始目标函数更容易优化，并且通过对采集函数应用特定标准来选择要评估的输入值，相较于遗传算法，优化模型参数的过程更加简单，可以节省大量的时间。贝叶斯参数优化算法流程如下图所示。

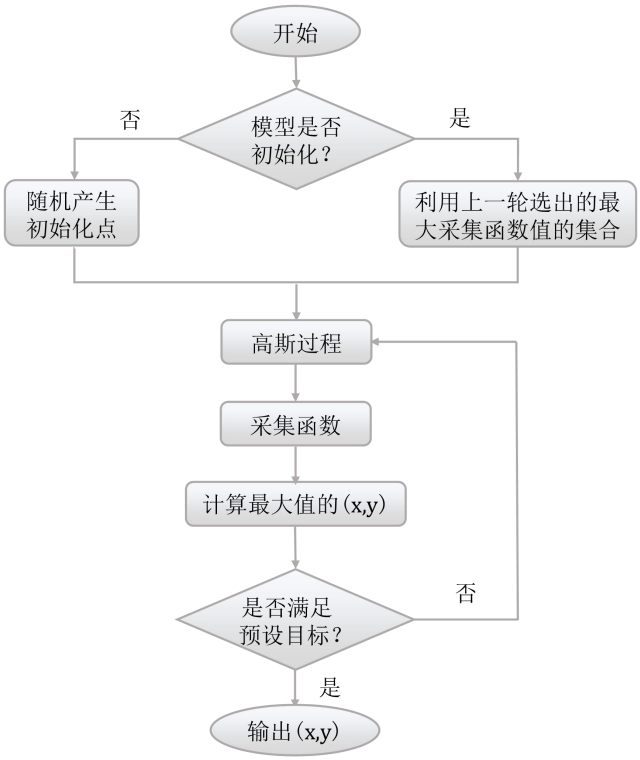

图 7 贝叶斯参数优化流程图

本文采用 HEBO 算法进行贝叶斯优化。

5.5.1 HEBO 算法

相比于普通的贝叶斯优化，HEBO 算法在以下两方面进行优化：带鲁棒性的采集函数、多目标采集函数[6]。

针对采集函数而言，从贝叶斯优化算法流程来看，对于非凸、非凹的黑盒函数，没有全局的解；代理函数和采集函数这两步是分开建模的，没有统一的计算框架让梯度在

这两步间相互传播。为了解决这两个问题，HEBO 算法引入了带鲁棒性的采集函数：

$$\max_{x} \alpha_{rob.}(\cdot) \equiv \max_{x} E_{\epsilon\sim N(0,\sigma_\epsilon^2)I}[\alpha^{\theta+\epsilon}(x|D)] \tag{9}$$

带鲁棒性的采集函数将尝试在一系列代理函数分布中寻找表现良好的候选点。带鲁棒性的采集函数的实现，只需要获取代理函数的均值和方差，而且如果满足以下条件：

$$\bar{\alpha}^{\theta}(x|D) = \alpha^{\theta}(x|D) + \eta\sigma_n \qquad \eta\sim N(0,1) \tag{10}$$

则对于任意的$\rho \in (0,1)$，我们有：

$$\left|\bar{\alpha}^{\theta}(x|D) - E_{\epsilon\sim N(0,\sigma_\epsilon^2)I}[\alpha^{\theta+\epsilon}(x|D)]\right| \le \rho \tag{11}$$

所以针对采集函数而言，引入高斯噪音$\epsilon\sim N(0,\sigma_\epsilon^2)I$，能够使采集函数更加鲁棒。

针对多目标采集函数，HEBO 算法引入多目标采集函数来获取帕累托最优解，防止经常输出相反的候选点，多目标采集函数为：

$$\max(\alpha_{EI}^{\theta}(x|D), \alpha_{PI}^{\theta}(x|D), \alpha_{UBC}^{\theta}(x|D)) \tag{12}$$

$\alpha_{Type}^{\theta}(x|D)$，$Type \in \{EI, PI, UCB\}$，为前面介绍的带鲁棒性的采集函数，HEBO 算法使用 NSGA—II 的进化算法进行求解。

HEBO 算法对数据的输入输出进行变换校准，将数据变换校准和 GPs 核函数联合在一起优化，并引入多目标采集函数，进行更鲁棒的探索，从而找到最精确的最优解。具体参数空间设置见代码。

5.5.2 优化流程

利用贝叶斯优化 Stacking 进行二手车估价预测的执行步骤为：

1、设定待优化参数空间（具体详细见代码），优化目标为均方根误差。

2、利用 HEBO 对 XGBoost、LightGBM、CatBoost 模型进行优化。

3、利用 Stacking 模型将上述三种模型融合。

4、将 Stacking 模型作为训练模型，对二手车估价进行预测。

5.5.3 HEBO 优化结果

1、LGBParams

表 4 LGBParams 的超参数优化结果

超参数	优化结果

learning_rate	0.010035982594961306
subsample	0.35028462644730735
colsample_bytree	0.3796965007412236
reg_lambda	0.0006865914755758266
reg_alpha	0.0026163019185854106
num_leaves	189
max_depth	37
n_estimators	1965
objective	mse
min_child_samples	25

2、XGBParams

表 5 XGBParams 的超参数优化结果

超参数	优化结果
learning_rate	0.010211967985703159
subsample	0.30001115463199896
colsample_bytree	0.48460986059028555
lambda	0.032780563044634864
alpha	0.06981554843421037
max_depth	26
min_child_weight	10
n_estimators	1845

3、CATParams

表 6 CATParams 的超参数优化结果

超参数	优化结果
learning_rate	0.08491056514750207
l2_leaf_reg	1
max_depth	8
n_estimators	2000

min_child_samples	25
loss_function	RMSE

5.6 实验结果

利用上述提及的 Stacking 技术对经过 HEBO 优化的 XGBoost、LightGBM、CatBoost 三种模型的融合所建立的新的模型，对二手车进行交易周期预估。根据题目要求的模型评测准则的算法，我们在验证集上求得对应值为：10.45。

六、基于多元非线性规划的调价规划

6.1 门店经营模型

在车辆售卖过程中，决定门店利润的主要有车辆资金占用成本、停车位占用成本以及售车利润。管理门店时我们进行以天为结算周期，利用预测结果调整销售价格使门店当天利润最大以达到经营目标。

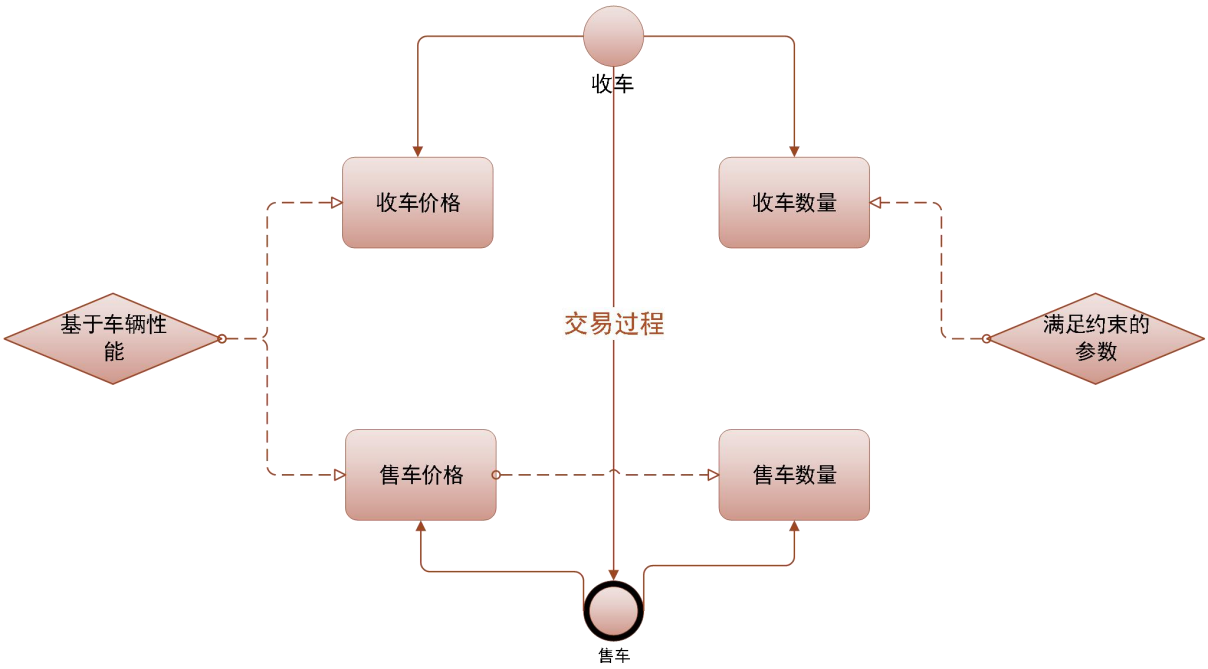

图 8 交易过程结构图

在交易中，利润与成本和售价有关，因此我们需要分析影响成本与售价的主要因素。

其中收车定价和售车定价符合我们初赛模型中的定价模型，其主要因素均为二手车车况，因此两者之间存在正相关关系；日收车数量我们假设为一个满足约束的固定参数；日售车数量基于库存车辆的价格。因此我们分析对于某二手车的售出情况与车辆价格的关系。

根据消费者需求规律，我们可以知道商品价格越高，售出数量越少。对于单个二手车而言，价格越高，售出概率越小。其关系可用下图表示：

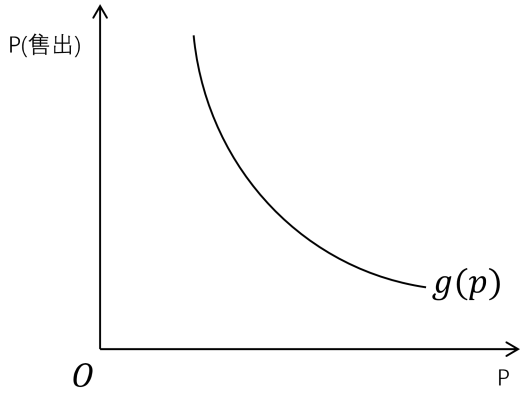

图 9 出售概率与售价关系图

记出售概率为$g(p)$，假设$g(p) > 80\%$即为成功售出，记作 1；未成功售出记作 0。依据数据集我们可得到一个关于售出情况的分类器，并基于此计算每日售出数量。

6.2 基于多元非线性目标的价格规划

6.2.1 经营模型中变量的确定

根据 6.1 所述：收车价和售车价均只与车辆状况有关，所以收车价可以用与定价正相关的函数表示，即$p_{ti} = f(P_{ti}')$。

对于成本而言，我们不妨假设：资金占用成本M与收车价有关，停车位占用成本 S 与停车时间及库存车辆有关。因此第t天的资金占用成本我们表示为

$$M_t = \sum_{i=1}^{n_t} M(p_{ti}) = \sum_{i=1}^{n_t} M(f(P_{ti}')) \tag{13}$$

因此第 t 天的停车位占用成本我们将其表示为

$$S_t = S(\sum_{j=1}^{t} (m_j - n_j)) \tag{14}$$

又因为售车利润为售价与进价之差，所以第t天的售车利润表示为

$$R_t = \sum_{i=1}^{n_t} \left(P_{ti}^{'} - P_{ti}\right) = \sum_{i=1}^{n_t} g(P_{ti}') \tag{15}$$

因此第j天的利润我们可以表示为

$$Z_j = \sum_{i=1}^{n_j} g(P_{ji}') - \sum_{i=1}^{n_j} M\left(f(P_{ji}')\right) - S(\sum_{i=1}^{j}(m_i - n_i)) \tag{16}$$

从评估周期开始到第j天的利润为

$$Z = \sum_{t=1}^{j}\sum_{i=1}^{n_j} f(P_{ti}') - \sum_{t=1}^{j}\sum_{i=1}^{n_j} M(f(P_{ti}'))$$

$$- \sum_{i=1}^{j} S((m_1 - n_1) + \ldots + (m_{i-1} - n_{i-1}) + (m_i - n_i)) \tag{17}$$

根据 6.1 中商品价格与出售情况的关系介绍，第t天第i辆车的出售情况可表示为

$$h(P_{ti}') = \begin{cases} 1, g(P_{ti}') > 80\% \\ 0 \end{cases} \tag{18}$$

在第t天时，我们已知前$t-1$天累计的交易情况，包括收车和售车数、花费成本以及售车利润；除此之外，假设已知第t天新收车辆数m_t。设第 1 天到前$t-1$天的资金占用成本为$\overline{M_{t-1}}$，停车位占用成本为$\overline{S_{t-1}}$，售车利润为$\overline{R_{t-1}}$，库存车辆数为$\overline{c_{t-1}}$，由模型可知其均为已知量。因此第t天时库存车辆数为$\overline{c_{t-1}} + m_t$。由(18)可得出售车辆数为

$$n_t = \sum_{i=1}^{\overline{c_{t-1}}+m_t} h(P_{ti}') \tag{19}$$

6.2.2 经营模型的目标函数和约束

根据以上关系式将(17)进行在第t天时，门店所获累积利润为

$$Z = \overline{R_{t-1}} + \sum_{i=1}^{n_t} f(P_{ti}') - \overline{M_{t-1}} - \sum_{i=1}^{n_t} M(f(P_{ti}')) - \overline{S_{t-1}} - \sum_{t=1}^{j} S(\overline{c_{t-1}} + m_t -$$

$$\sum_{i=1}^{\overline{c_{t-1}}+m_t} h(P_{ti}')) \tag{20}$$

由于常数固定，因此要求在第t天时门店所获累积利润最大，即使得在第t天门店所获利润最大。目标函数如下：

$$\max Z_j = \sum_{i=1}^{n_j} f(P_{ti}') - \sum_{i=1}^{n_j} M(f(P_{ti}')) - \sum_{t=1}^{j} S(\overline{c_{t-1}} + m_t - \sum_{i=1}^{\overline{c_{t-1}}+m_t} h(P_{ti}')) \tag{21}$$

此时问题转化为每一天中每辆车售价为自变量的多元非线性规划问题，该问题中的约束如下：

$$\begin{cases} 0 \le \overline{c_{t-1}} + m_t - n_t \le C_{max} \\ \overline{c_{t-1}} \ge 0 \\ P_{ti}' > 0 \\ \overline{c_{t-1}}, m_t, n_t \in Z^* \end{cases} \tag{22}$$

6.2.3 经营模型的求解以及对解的处理

根据我们上述的模型的建立所得到的目标函数以及约束条件，我们判断其问题属于约束优化问题。因此我们选择采用罚函数法进行求解。

由于这里的模型是抽象的，没有具体的数据支撑，无法得到具体的解。若有数据分析，我们便可以得到第 t 天的能使门店利润最大的每辆车的售价，根据之前的每辆车的售价，将两者进行对比，我们便可以得到每天每辆车的价格的处理方案，从而最小化成本，最大化利润，何时对某个车辆是否进行调价以及调整多大幅度的问题得以解决。

七、模型的评价

7.1 模型的优点

1、本文使用基于 Stacking 技术的 XGBoost、LightGBM 和 CatBoost 融合模型来进行估价，相比于单一模型而言，有利于提高模型的准确性。

2、利用 HEBO 算法来对模型超参数调优，大大提高模型的准确性。

3、使用相关性分析与 XGBoost 算法结合来进行特征筛选，并采用递归特征消除法对重要特征进行检验，以达到准确筛选出相关特征，除去无关及冗余特征的结果，提升算法精度并减少过拟合。

4、门店销售模型得到的能够使得利润最大化的售价，与二手车的初始定价作比较便能得到调价方案，简单明了。

7.2 模型的缺点

1、基于 Stacking 技术的 XGBoost、LightGBM 和 CatBoost 融合模型对周期的预测速度较慢。

2、通过实验发现，剔除边缘样本后，模型预测的效果变好，换而言之，我们的模型对于边缘样本的预测效果稍显不足。

3、多元非线性规划的调价模型由于没有确切数据，求解过程较为复杂。

参考文献

[1] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System.[J]. CoRR, 2016, abs/1603.02754

[2]李原吉,李丹,唐淇.基于 XGBoost 算法的依据车辆信息预估二手车成交价格模型[J].电子测试,2021(21):47-49.DOI:10.16520/j.cnki.1000-8519.2021.21.016.

[3] Wang D , Yang Z , Yi Z . LightGBM: An Effective miRNA Classification Method in Breast Cancer Patients[C]// the 2017 International Conference. 2017.

[4] Prokhorenkova L, Gusev G, Vorobev A, et al. CatBoost: unbiased boosting with categorical features[J]. arXiv preprint arXiv:1706.09516, 2017.

[5]赵梦想,胡敏.基于 Stacking 融合的电商平台二手车定价研究[J].信息与电脑(理论版),2021,33(19):35-38.

[6] Cowen-Riv Er S A I , Lyu W , Wang Z , et al. HEBO: Heteroscedastic Evolutionary Bayesian Optimisation[J]. 2020.

附录

附录 1： 问题一 (jupyter lab 编写)

```
import pandas as pd
import os
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import random
import numpy as np
import math
import warnings
import missingno as msno
from pprint import pprint
from chinese_calendar import is_workday, is_holiday
from pathlib import Path
from sklearn.preprocessing import PowerTransformer
import scipy.stats as st
from collections import defaultdict
import json

# 固定随机种子，稳定模型效果
random.seed(2021)
np.random.seed(2021)
%matplotlib inline
warnings.filterwarnings("ignore")
plt.rc("font",family="SimHei",size="10")
plt.rcParams['axes.unicode_minus']=False
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

```
pd.set_option('max_colwidth',200)
pd.set_option('expand_frame_repr', False)

# 数据集路径
root = Path(os.getcwd()).resolve().parent / "data"
root2 = Path(os.getcwd()).resolve().parent / "pic"
train = root / "Q3_train.csv"
data_df = pd.read_csv(train)

# index2name 存放的是所有特征的名称
index2name = ["carid", "pushDate", "pushPrice", "updatePriceTimeJson", "pullDate",
"withdrawDate", "展销时间",
              "品牌 id", "车系 id", "车型 id", "里程", "车辆颜色", "车辆所在城市
id", "国标码",
              "过户次数", "载客人数", "注册日期", "上牌日期", "国别", "厂商类型", "
年款", "排量", "变速箱", "燃油类型",
              "新车价"]

# category 存放的是类别特征的名称
category = ["品牌 id", "车系 id", "车型 id", "车辆颜色", "车辆所在城市 id", "厂商类型",
"燃油类型"] # 类别类型特征

# numerical 存放的是数值特征的名称
numerical = ["Feature_5", "年款", "排量", "变速箱", "Feature_2", "Feature_13", "载客人数
", "新车价", "target", "fday", "pushPrice"] #  "里程", "厂商类型"实数类型特征,
"Feature_12", "Feature_8"

# cross_num 存放的是要进行特征交叉的特征的名称
cross_num = ["新车价", "Feature_2", "Feature_12_1"]
```

```
# 匿名特征为 Feature_i，此处将 15 个匿名特征加入到 index2name 中
for _ in range(1, 16):
    index2name.append("Feature_"+str(_))
index2name.append("target")
index2name.append("fday")
index2name.append("sday")
index2name.append("成交周期")



sFea = {}
cx = data_df.groupby("车系  id")
for a, b in cx:
    sFea[a] = list(set(b["车型  id"]))

def metric(y_, y):
    ape = np.abs(y_ - y)
    mape = np.mean(ape)
    return mape

data_df = data_df[data_df["pullDate"] == data_df["withdrawDate"]]

data_df["updatePriceTimeJson"] = data_df[["pushDate", "pushPrice",
"updatePriceTimeJson"]].apply(lambda x: {str(x["pushDate"]): x["pushPrice"]} if
len(x["updatePriceTimeJson"]) == 2 else dict({str(x["pushDate"]): x["pushPrice"]},
**json.loads(x["updatePriceTimeJson"])), axis=1)

data_df = data_df[data_df["target"] <= 10000]
data_df.sort_values(by="展销时间", inplace=True)
```

```
data_df["展销时间"] = pd.to_datetime(data_df["展销时间"])
data_df["注册日期"] = pd.to_datetime(data_df["注册日期"])
data_df["上牌日期"] = pd.to_datetime(data_df["上牌日期"])
data_df["pushDate"] = pd.to_datetime(data_df["pushDate"])
data_df["pullDate"] = pd.to_datetime(data_df["pullDate"])

data_df["fday"] = data_df[["pushDate", "updatePriceTimeJson"]].apply(lambda x:
(pd.to_datetime(list(x["updatePriceTimeJson"].keys())[-1]) - x["pushDate"]).days, axis=1)
data_df["sday"] = data_df["成交周期"]    - data_df["fday"]

data_df["成交周期 2"] = data_df["成交周期"]
del data_df["成交周期"]
data_df["成交周期"] = data_df["成交周期 2"]
del data_df["成交周期 2"]

te = pd.read_csv(root / "Q3_processed.csv")
te["展销时间"] = pd.to_datetime(te["展销时间"])
te["注册日期"] = pd.to_datetime(te["注册日期"])
te["上牌日期"] = pd.to_datetime(te["上牌日期"])
te["pushDate"] = pd.to_datetime(te["pushDate"])
te["updatePriceTimeJson"] = te[["pushDate", "pushPrice",
"updatePriceTimeJson"]].apply(lambda x: {str(x["pushDate"]): x["pushPrice"]} if
len(x["updatePriceTimeJson"]) == 2 else dict({str(x["pushDate"]): x["pushPrice"]},
**json.loads(x["updatePriceTimeJson"])), axis=1)

# add the null
# tr = pd.concat([tr, data_df_null], axis=0)
tr = data_df
tr = tr[((tr["新车价"]-tr["target"]) / tr["新车价"]) < 0.969]
tr = tr[tr["新车价"] > tr["target"]]
```

```
# 使用众数填补缺省值
for _ in tr:
    tr[_] = tr[_].fillna(tr[_].value_counts().index[0])

import scipy.stats as st
y = tr["sday"]
plt.figure(figsize=[8,5])
plt.subplot(1,3,1)
plt.title('Johnson SU')
sns.distplot(y, kde=False, fit=st.johnsonsu)
plt.subplot(1,3,2)
plt.title('Normal')
sns.distplot(y, kde=False, fit=st.norm)
plt.subplot(1,3,3)
plt.title('Log Normal')
sns.distplot(y, kde=False, fit=st.lognorm)
plt.savefig(root2 / "Q3_0.jpg")

pt_tr = PowerTransformer(method="yeo-johnson")

tr["sday"] = pt_tr.fit_transform(np.array(tr['sday']).reshape(-1,1))
# tr["成交周期"] = np.log(tr["成交周期"])
plt.figure(figsize=[20, 10])
sns.displot(tr['sday'])
plt.savefig(root2 / "Q3_1.jpg")

# ******************构造新特征********************
class FeatureEngine:
    def __init__(self):
```

```
        self.ID = defaultdict(lambda: defaultdict(int))
        self.sFea = {}
        self.target_cxx = defaultdict(lambda: defaultdict(int))

    @staticmethod
    def add2list(pos, name, stage="train"):
        if stage == "train":
            index2name.insert(pos, name)
            numerical.append(name)

    @staticmethod
    def add2list_cat(pos, name, stage="train"):
        if stage == "train":
            index2name.insert(pos, name)
            category.append(name)

    # strongFea 没啥用，忽略
    def strongFea(self, data_df, stage):
        if stage == "train":
            cx = tr.groupby("车系  id")
            for a, b in cx:
                self.sFea[a] = list(set(b["车型  id"]))

            for cx_idx in self.sFea:
                for cxing_idx in self.sFea[cx_idx]:
                    if len(data_df[(data_df["车系  id"]==cx_idx) & (data_df["车型
id"]==cxing_idx)]) <= 5:
                        self.target_cxx[cx_idx][cxing_idx] = 0
                    else:
                        self.target_cxx[cx_idx][cxing_idx] = float(data_df[(data_df["车
```

```
系 id"]==cx_idx) & (data_df["车型 id"]==cxing_idx)]["target"].mean())


        data_df.insert(len(data_df.columns)-1, "车系车型_mean", data_df.apply(lambda x:
self.target_cxx[x["车系 id"]][x["车型 id"]], axis=1))
        data_df["车系车型_mean"] = data_df["车系车型_mean"].fillna(data_df["车系车
型_mean"].mean())
        self.add2list(len(index2name)-1, "车系车型_mean", stage)


    # 下面一系列的 get 函数是类似的，按照某个特征进行聚类，然后求 mean，count，
max.....
    def getMean(self, data_df, fea_name, stage):
        if stage == "train":
            cs = data_df.loc[:, [fea_name, "target"]]
            cs = cs.groupby(fea_name)
            for k, v in cs:
                self.ID[fea_name+"_mean"][list(set(v[fea_name]))[0]] =
v["target"].mean()
            data_df.insert(len(data_df.columns)-1, fea_name+"_mean",
data_df[fea_name].map(self.ID[fea_name+"_mean"]))
        else:
            data_df.insert(len(data_df.columns)-1, fea_name+"_mean",
data_df[fea_name].map(self.ID[fea_name+"_mean"]))
            data_df[fea_name+"_mean"] =
data_df[fea_name+"_mean"].fillna(data_df[fea_name+"_mean"].mean())
        self.add2list(len(index2name)-1, fea_name+"_mean", stage)



    def getCount(self, data_df, fea_name, stage):
        if stage == "train":
            self.ID[fea_name+"_count"] = dict(data_df[fea_name].value_counts())
```

```
                data_df.insert(len(data_df.columns)-1, fea_name+"_count",
data_df[fea_name].map(self.ID[fea_name+"_count"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_count",
data_df[fea_name].map(self.ID[fea_name+"_count"]))
                data_df[fea_name+"_count"] =
data_df[fea_name+"_count"].fillna(data_df[fea_name+"_count"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_count", stage)


    def getMax(self, data_df, fea_name, stage):
        if stage == "train":
            cs = data_df.loc[:, [fea_name, "target"]]
            cs = cs.groupby(fea_name)
            for k, v in cs:
                self.ID[fea_name+"_max"][list(set(v[fea_name]))[0]] = v["target"].max()
            data_df.insert(len(data_df.columns)-1, fea_name+"_max",
data_df[fea_name].map(self.ID[fea_name+"_max"]))
        else:
            data_df.insert(len(data_df.columns)-1, fea_name+"_max",
data_df[fea_name].map(self.ID[fea_name+"_max"]))
            data_df[fea_name+"_max"] =
data_df[fea_name+"_max"].fillna(data_df[fea_name+"_max"].value_counts().index[0])
        self.add2list(len(index2name)-1, fea_name+"_max", stage)


    def getMin(self, data_df, fea_name, stage):
        if stage == "train":
            cs = data_df.loc[:, [fea_name, "target"]]
            cs = cs.groupby(fea_name)
```

```
                for k, v in cs:
                    self.ID[fea_name+"_min"][list(set(v[fea_name]))[0]] = v["target"].min()
                data_df.insert(len(data_df.columns)-1, fea_name+"_min",
data_df[fea_name].map(self.ID[fea_name+"_min"]))
            else:
                data_df.insert(len(data_df.columns)-1, fea_name+"_min",
data_df[fea_name].map(self.ID[fea_name+"_min"]))
                data_df[fea_name+"_min"] =
data_df[fea_name+"_min"].fillna(data_df[fea_name+"_min"].value_counts().index[0])
            self.add2list(len(index2name)-1, fea_name+"_min", stage)



    def getMedian(self, data_df, fea_name, stage):
        if stage == "train":
            cs = data_df.loc[:, [fea_name, "target"]]
            cs = cs.groupby(fea_name)
            for k, v in cs:
                self.ID[fea_name+"_median"][list(set(v[fea_name]))[0]] =
v["target"].median()
            data_df.insert(len(data_df.columns)-1, fea_name+"_median",
data_df[fea_name].map(self.ID[fea_name+"_median"]))
        else:
            data_df.insert(len(data_df.columns)-1, fea_name+"_median",
data_df[fea_name].map(self.ID[fea_name+"_median"]))
            data_df[fea_name+"_median"] =
data_df[fea_name+"_median"].fillna(data_df[fea_name+"_median"].value_counts().index[0])
        self.add2list(len(index2name)-1, fea_name+"_median", stage)



    def getSum(self, data_df, fea_name, stage):
```

```
        if stage == "train":
            cs = data_df.loc[:, [fea_name, "target"]]
            cs = cs.groupby(fea_name)
            for k, v in cs:
                self.ID[fea_name+"_sum"][list(set(v[fea_name]))[0]] = v["target"].sum()
            data_df.insert(len(data_df.columns)-1, fea_name+"_sum",
data_df[fea_name].map(self.ID[fea_name+"_sum"]))
        else:
            data_df.insert(len(data_df.columns)-1, fea_name+"_sum",
data_df[fea_name].map(self.ID[fea_name+"_sum"]))
            data_df[fea_name+"_sum"] =
data_df[fea_name+"_sum"].fillna(data_df[fea_name+"_sum"].value_counts().index[0])
        self.add2list(len(index2name)-1, fea_name+"_sum", stage)


    def getStd(self, data_df, fea_name, stage):
        if stage == "train":
            cs = data_df.loc[:, [fea_name, "target"]]
            cs = cs.groupby(fea_name)
            for k, v in cs:
                self.ID[fea_name+"_std"][list(set(v[fea_name]))[0]] = v["target"].std()
            data_df.insert(len(data_df.columns)-1, fea_name+"_std",
data_df[fea_name].map(self.ID[fea_name+"_std"]))
        else:
            data_df.insert(len(data_df.columns)-1, fea_name+"_std",
data_df[fea_name].map(self.ID[fea_name+"_std"]))
            data_df[fea_name+"_std"] =
data_df[fea_name+"_std"].fillna(data_df[fea_name+"_std"].value_counts().index[0])
        self.add2list(len(index2name)-1, fea_name+"_std", stage)
```

```
    def newFeature(self, data_df, stage="train"):
        # 构造特征
        pd.set_option('mode.use_inf_as_na', True)

        data_df.insert(len(data_df.columns)-1, "降价比率", (data_df["新车价"] -
data_df["target"]) / data_df["新车价"])
        self.add2list(len(index2name)-1, "降价比率", stage)

        data_df.insert(len(data_df.columns)-1,"展销是否为假期",data_df["展销时间
"].map(lambda x: is_holiday(x)))
        self.add2list_cat(len(index2name)-1, "展销是否为假期", stage)

        data_df.insert(len(data_df.columns)-1,"是否转户", data_df["过户次数
"].map(lambda x: int(x > 0)))
        # self.add2list(len(index2name)-1, "是否转户", stage)
        # self.add2list_cat(len(index2name)-1, "是否转户", stage)

        data_df.insert(len(data_df.columns)-1, "注册年份", data_df["注册日期
"].map(lambda x: x.year))
        self.add2list(len(index2name)-1, "注册年份", stage)

        data_df.insert(len(data_df.columns)-1, "Feature_12_num_sqrt",
data_df["Feature_12"].map(lambda x: eval(x) ** (1./3.)))
        self.add2list(len(index2name)-1, "Feature_12_num_sqrt", stage)

        data_df.insert(len(data_df.columns)-1, "Feature_12_0",
data_df["Feature_12"].map(lambda x: int(x.split('*')[0])))
        self.add2list(len(index2name)-1, "Feature_12_0", stage)

        data_df.insert(len(data_df.columns)-1, "Feature_12_1",
```

```
data_df["Feature_12"].map(lambda x: int(x.split('*')[1])))
            self.add2list(len(index2name)-1, "Feature_12_1", stage)

            data_df.insert(len(data_df.columns)-1, "Feature_12_2",
data_df["Feature_12"].map(lambda x: int(x.split('*')[2])))
            self.add2list(len(index2name)-1, "Feature_12_2", stage)

            data_df.insert(len(data_df.columns)-1, "汽车年龄", (data_df["展销时间"] -
data_df["上牌日期"]).map(lambda x: x.days))
            # self.add2list(len(index2name)-1, "汽车年龄", stage)

            data_df.insert(len(data_df.columns)-1, "降价次数",
data_df["updatePriceTimeJson"].map(lambda x: len(x)-1))
            self.add2list(len(index2name)-1, "降价次数", stage)

            data_df.insert(len(data_df.columns)-1, "降价比例", data_df[["pushPrice",
"updatePriceTimeJson"]].apply(lambda x: (x["pushPrice"] -
float(list(x["updatePriceTimeJson"].values())[-1])) / x["pushPrice"], axis=1))
            self.add2list(len(index2name)-1, "降价比例", stage)


            self.getSum(data_df, "车辆所在城市  id", stage)

            self.getMean(data_df, "品牌  id", stage)
            self.getCount(data_df, "品牌  id", stage)
            self.getMax(data_df, "品牌  id", stage)
            self.getMin(data_df, "品牌  id", stage)
            self.getMedian(data_df, "品牌  id", stage)
            self.getSum(data_df, "品牌  id", stage)
```

```
            # self.getMean(data_df, "是否转户", stage)
            self.getMean(data_df, "厂商类型", stage)
            # self.getMean(data_df, "燃油类型", stage)
            self.getMean(data_df, "国标码", stage)
            self.getMean(data_df, "年款", stage)
            self.getMean(data_df, "变速箱", stage)
            self.getMean(data_df, "Feature_2", stage)
            # self.getMean(data_df, "Feature_6", stage)
            # self.getMean(data_df, "Feature_8", stage)
            # self.getMean(data_df, "Feature_9", stage)
            # self.getMean(data_df, "展销是否为假期", stage)
            # self.getCount(data_df, "车型  id", stage)

            data_df["新车价"] = np.log(data_df["新车价"])
            data_df.insert(len(data_df.columns)-1, "评估方法 1", np.log(data_df.apply(lambda x: x["新车价"] * (0.15 + 0.85*(1 - 0.11*min(3, max(0., x["汽车年龄"] / 365)) - 0.1 * min(4., max(0, x["汽车年龄"] / 365-3)) - 0.09 * min(4., max(0, x["汽车年龄"] / 365-7)))), axis=1)))
            self.add2list(len(index2name)-1, "评估方法 1", stage)
            data_df.insert(len(data_df.columns)-1, "评估方法 2", np.log(data_df.apply(lambda x: x["新车价"] * (15 - 5 * min(6, max(0, x["里程"])) / 6 - 4 * min(6, max(0, x["里程"]-6)) / 6 - 3 * min(6, max(0, x["里程"]-12)) / 6 - 2 * min(6, max(0, x["里程"]-18)) / 6 - min(6, max(0, x["里程"]-24)) / 6) / 15, axis=1)))
            self.add2list(len(index2name)-1, "评估方法 2", stage)
            data_df["评估方法 2"].fillna(data_df["评估方法 1"], inplace=True)

#  生成新特征
FE = FeatureEngine()
FE.newFeature(tr, "train")
FE.newFeature(te, "test")
# FE.crossFeature(tr, "train")
```

```
# FE.crossFeature(te, "test")


#  取出 numerical 特征和 categories 特征
tr_x = tr.iloc[:, :-1]
tr_y = tr[["sday"]]
# te_x = te.iloc[:, :-1]
# te_y = te[["sday"]]
tr_x_num = tr[numerical].astype("float")
te_x_num = te[numerical].astype("float")
tr_x_cat = tr[category]
te_x_cat = te[category]

#  特征归一化：  (x-min) / (max-min)
for _ in tr_x_num:
    if not _ in ["厂商类型", "载客人数", "Feature_8", "是否转户"]:
        min_ = tr_x_num[_].min()
        max_ = tr_x_num[_].max()
        tr_x_num[_] = (tr_x_num[_] - min_) / (max_ - min_)
        te_x_num[_] = (te_x_num[_] - min_) / (max_ - min_)

tr_x_num_array = np.array(tr_x_num)
tr_y_array = np.array(tr_y)
te_x_num_array = np.array(te_x_num)

plt.figure(figsize=(20,10))
tr_num = pd.concat([tr_x_num, tr_y], axis=1)
sns.heatmap(tr_num.corr(), annot=True, fmt='.2f')
plt.savefig(root2 / "Q3_2.jpg")

from xgboost import XGBRegressor
```

```
from lightgbm.sklearn import LGBMRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from catboost import CatBoostRegressor
from pprint import pprint

# 优化结果：模型参数(使用 hebo 进行参数优化，因为一次优化的时间消耗大，因此将优化结果存了下来)
lgbParams = {'learning_rate': 0.010035982594961306, 'subsample': 0.35028462644730735, 'colsample_bytree': 0.3796965007412236, 'reg_lambda': 0.0006865914755758266, 'reg_alpha': 0.0026163019185854106, 'num_leaves': 189, 'max_depth': 37, 'n_estimators': 1965, 'objective': 'mse', 'min_child_samples': 25}
xgbParams = {'learning_rate': 0.010211967985703159, 'subsample': 0.30001115463199896, 'colsample_bytree': 0.48460986059028555, 'lambda': 0.032780563044634864, 'alpha': 0.06981554843421037, 'max_depth': 26, 'min_child_weight': 10, 'n_estimators': 1845}
catParams = {'learning_rate': 0.08491056514750207, 'l2_leaf_reg': 1, 'max_depth': 8, 'n_estimators': 2000, 'min_child_samples': 25, 'loss_function': 'RMSE', 'verbose': False, 'task_type': 'CPU'}

from xgboost import plot_importance
from numpy import sort
from sklearn.feature_selection import SelectFromModel

# Fit model using each importance as a threshold
model_XGB = XGBRegressor(**xgbParams).fit(tr_x_num, tr_y)
thresholds = sort(model_XGB.feature_importances_)
f_i = dict(zip(numerical, model_XGB.feature_importances_))
f_i = sorted(f_i.items(), key=lambda x: x[1])
fig,ax = plt.subplots(figsize=(15,15))
plot_importance(model_XGB, importance_type="weight", ax=ax)
```

```
plt.savefig(root2 / "Q3_3.jpg")


for name, thresh in f_i:
    # select features using threshold
        selection = SelectFromModel(model_XGB, threshold=thresh, prefit=True)
        select_X_train = selection.transform(tr_x_num)
    # train model
        selection_model = XGBRegressor(**xgbParams)
        selection_model.fit(select_X_train, tr_y)
    # eval model
        select_X_test = selection.transform(te_x_num)
        y_pred = selection_model.predict(select_X_test)
        accuracy = metric(pt_tr.inverse_transform(y_pred.reshape(-1, 1)) + te[["fday"]],
np.array(te[["成交周期"]]))
        print("Feature Name=%s Thresh=%.3f, n=%d, Accuracy: %.2f" % (name, thresh,
select_X_train.shape[1], accuracy))


from sklearn.model_selection import KFold
from sklearn.base import BaseEstimator, RegressorMixin, TransformerMixin, clone
import numpy as np


# stacking
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
        def __init__(self, base_models, meta_model, n_folds=5):
                self.base_models = base_models
                self.meta_model = meta_model
                self.n_folds = n_folds


        # 将原来的模型 clone 出来，并且进行实现 fit 功能
        def fit(self, X, y):
```

```
            self.base_models_ = [list() for x in self.base_models]
            self.meta_model_ = clone(self.meta_model)
            kfold = KFold(n_splits=self.n_folds, shuffle=True, random_state=42)

            #对于每个模型，使用交叉验证的方法来训练初级学习器，并且得到次级训练集
            out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
            for i, model in enumerate(self.base_models):
                for train_index, holdout_index in kfold.split(X, y):
                    instance = clone(model)
                    instance.fit(X[train_index], y[train_index])
                    y_pred = instance.predict(X[holdout_index])
                    self.base_models_[i].append(instance)
                    out_of_fold_predictions[holdout_index, i] = y_pred

            # 使用次级训练集来训练次级学习器
            self.meta_model_.fit(out_of_fold_predictions, y)
            return self

        #在上面的 fit 方法当中，我们已经将我们训练出来的初级学习器和次级学习器保存下来了
        #predict 的时候只需要用这些学习器构造我们的次级预测数据集并且进行预测就可以了
        def predict(self, X):
            meta_features = np.column_stack([
                np.column_stack([model.predict(X) for model in base_models]).mean(axis=1)
                for base_models in self.base_models_ ])
            return self.meta_model_.predict(meta_features)

base_models = [
```

```
                    CatBoostRegressor(**catParams),
                    LGBMRegressor(**lgbParams),
                    XGBRegressor(**xgbParams),
                   ]

meta_model = LinearRegression()
stacking_model = StackingAveragedModels(base_models=base_models,
meta_model=meta_model)

stacking_model.fit(tr_x_num_array, tr_y_array.ravel())

y_predict = stacking_model.predict(te_x_num_array)

res = pt_tr.inverse_transform(y_predict.reshape(-1, 1)) + te[["fday"]]

res.insert(0, "carid", te["carid"])

res["fday"] = res["fday"].astype("int")

res.to_csv(root / "Q3.txt", index=False, sep='\t', header=None)
```