线上第十三名比赛方案

一、解决方案及算法

分四个方面去阐述
数据处理
特征工程
选择的模型
集成的方法

数据处理

1、box-cox 变换目标值“price”，解决长尾分布。
2、删除与目标值无关的列，例如“SaleID”，“name”。这里可以挖掘一下“name”的长度作为新的特征。
3、异常点处理，删除训练集特有的数据，例如删除“seller”==1 的值。
4、缺失值处理，分类特征填充众数，连续特征填充平均值。
5、其他特别处理，把取值无变化的列删掉。
6、异常值处理，按照题目要求“power”位于 0～600，因此把“power”>600 的值截断至 600，把"notRepairedDamage"的非数值的值替换为 np.nan，让模型自行处理。

特征工程

1、时间地区类
从“regDate”，“creatDate”可以获得年、月、日等一系列的新特征，然后做差可以获得使用年长和使用天数这些新特征。

“regionCode”没有保留。
因为尝试了一系列方法，并且发现了可能会泄漏“price”，因此最终没保留该特征。

2、分类特征
对可分类的连续特征进行分桶，kilometer 是已经分桶了。
然后对"power"和"model"进行了分桶。

使用分类特征“brand”、“model”、“kilometer”、“bodyType”、“fuelType”与“price”、“days”、“power”进行特征交叉。
交叉主要获得的是后者的总数、方差、最大值、最小值、平均数、众数、峰度等等

这里可以获得非常多的新特征，挑选的时候，直接使用 lightgbm 帮我们去选择特征，一组组的放进去，最终保留了以下特征。（注意：这里是使用 1/4 的训练集进行挑选可以帮助我们更快的锁定真正 Work 的特征）
'model_power_sum','model_power_std',
'model_power_median', 'model_power_max',
'brand_price_max', 'brand_price_median',
'brand_price_sum', 'brand_price_std',
'model_days_sum','model_days_std',
'model_days_median', 'model_days_max',
'model_amount','model_price_max',
'model_price_median','model_price_min',
'model_price_sum', 'model_price_std',
'model_price_mean'

3、连续特征
使用了置信度排名靠前的匿名特征“v_0”、“v_3”与“price”进行交叉，测试方法以上述一样，效果并不理想。
因为都是匿名特征，比较训练集和测试集分布，分析完基本没什么问题，并且它们在 lightgbm 的输出的重要性都是非常高的，所以先暂且全部保留。

4、补充特征工程
主要是对输出重要度非常高的特征进行处理
特征工程一期：
对 14 个匿名特征使用乘法处理得到 14*14 个特征

使用 sklearn 的自动特征选择帮我们去筛选，大概运行了半天的时间。
大致方法如下：

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LinearRegression
sfs = SFS(LGBMRegressor(n_estimators = 1000,objective='mae' ),
          k_features=50,
          forward=True,
          floating=False,
          cv = 0)


sfs.fit(X_data, Y_data)
print(sfs.k_feature_names_)
```

最终筛选得到：
'new3*3', 'new12*14', 'new2*14','new14*14'

特征工程二期：
对 14 个匿名特征使用加法处理得到 14*14 个特征

这次不选择使用自动特征选择了，因为运行实在太慢了，笔记本耗不起。
然后先尝试了全部放进去 lightgbm 训练是否有效，惊喜的发现效果很明显，由于新生成的特征很多，因此要对一部分冗余的特征进行删除。
使用的方法是删除相关性高的变量,把要删除的特征记录下来
大致方法如下：（剔除相关度>0.95 的）

```
corr = X_data.corr(method='spearman')
feature_group = list(itertools.combinations(corr.columns, 2))
print(feature_group)

# 删除相关性高的变量,调试好直接去主函数进行剔除
def filter_corr(corr, cutoff=0.7):
    cols = []
    for i,j in feature_group:
        if corr.loc[i, j] > cutoff:
            print(i,j,corr.loc[i, j])
            i_avg = corr[i][corr[i] != 1].mean()
            j_avg = corr[j][corr[j] != 1].mean()
            if i_avg >= j_avg:
                cols.append(i)
            else:
                cols.append(j)
    return set(cols)

drop_cols = filter_corr(corr, cutoff=0.95)
print(drop_cols)
```

最终获得的应该删除的特征为：
['new14+6', 'new13+6', 'new0+12', 'new9+11', 'v_3', 'new11+10', 'new10+14', 'new12+4', 'new3+4', 'new11+11', 'new13+3', 'new8+1', 'new1+7', 'new11+14', 'new8+13', 'v_8', 'v_0', 'new3+5', 'new2+9', 'new9+2', 'new0+11', 'new13+7', 'new8+11', 'new5+12', 'new10+10', 'new13+8', 'new11+13', 'new7+9', 'v_1', 'new7+4', 'new13+4', 'v_7', 'new5+6', 'new7+3', 'new9+10', 'new11+12', 'new0+5', 'new4+13', 'new8+0', 'new0+7', 'new12+8', 'new10+8', 'new13+14', 'new5+7', 'new2+7', 'v_4', 'v_10', 'new4+8', 'new8+14', 'new5+9', 'new9+13', 'new2+12', 'new5+8', 'new3+12', 'new0+10', 'new9+0', 'new1+11', 'new8+4', 'new11+8', 'new1+1', 'new10+5', 'new8+2', 'new6+1', 'new2+1', 'new1+12', 'new2+5', 'new0+14', 'new4+7', 'new14+9', 'new0+2', 'new4+1', 'new7+11', 'new13+10', 'new6+3', 'new1+10', 'v_9', 'new3+6', 'new12+1', 'new9+3', 'new4+5', 'new12+9', 'new3+8', 'new0+8', 'new1+8', 'new1+6', 'new10+9', 'new5+4', 'new13+1', 'new3+7', 'new6+4', 'new6+7', 'new13+0', 'new1+14', 'new3+11', 'new6+8', 'new0+9', 'new2+14', 'new6+2', 'new12+12', 'new7+12', 'new12+6', 'new12+14', 'new4+10', 'new2+4', 'new6+0', 'new3+9', 'new2+8', 'new6+11', 'new3+10', 'new7+0', 'v_11', 'new1+3', 'new8+3', 'new12+13', 'new1+9', 'new10+13', 'new5+10', 'new2+2', 'new6+9', 'new7+10', 'new0+0', 'new11+7', 'new2+13', 'new11+1', 'new5+11', 'new4+6', 'new12+2', 'new4+4', 'new6+14', 'new0+1', 'new4+14', 'v_5', 'new4+11', 'v_6', 'new0+4', 'new1+5', 'new3+14', 'new2+10', 'new9+4', 'new2+6', 'new14+14', 'new11+6', 'new9+1',

'new3+13', 'new13+13', 'new10+6', 'new2+3', 'new2+11', 'new1+4', 'v_2', 'new5+13', 'new4+2', 'new0+6', 'new7+13', 'new8+9', 'new9+12', 'new0+13', 'new10+12', 'new5+14', 'new6+10', 'new10+7', 'v_13', 'new5+2', 'new6+13', 'new9+14', 'new13+9', 'new14+7', 'new8+12', 'new3+3', 'new6+12', 'v_12', 'new14+4', 'new11+9', 'new12+7', 'new4+9', 'new4+12', 'new1+13', 'new0+3', 'new8+10', 'new13+11', 'new7+8', 'new7+14', 'v_14', 'new10+11', 'new14+8', 'new1+2']]
特征工程三、四期：
这两期的效果不明显，为了不让特征冗余，所以选择不添加这两期的特征，具体的操作可以在 feature 处理的代码中看到。

5、神经网络的特征工程补充说明
以上特征工程处理都是针对于树模型来进行的，接下来，简单说明神经网络的数据预处理。
各位都知道由于 NN 的不可解释性，可以生成大量的我们所不清楚的特征，因此我们对于 NN 的数据预处理只要简单处理异常值以及缺失值。

大部分的方法都包含在以上针对树模型数据处理方法中，重点讲述几个不同点：
在对于“notRepairedDamage”的编码处理，对于二分类的缺失值，往往取其中间值。
在对于其他缺失值的填充，在测试了效果后，发现填充众数的效果比平均数更好，因此均填充众数。

选择的模型

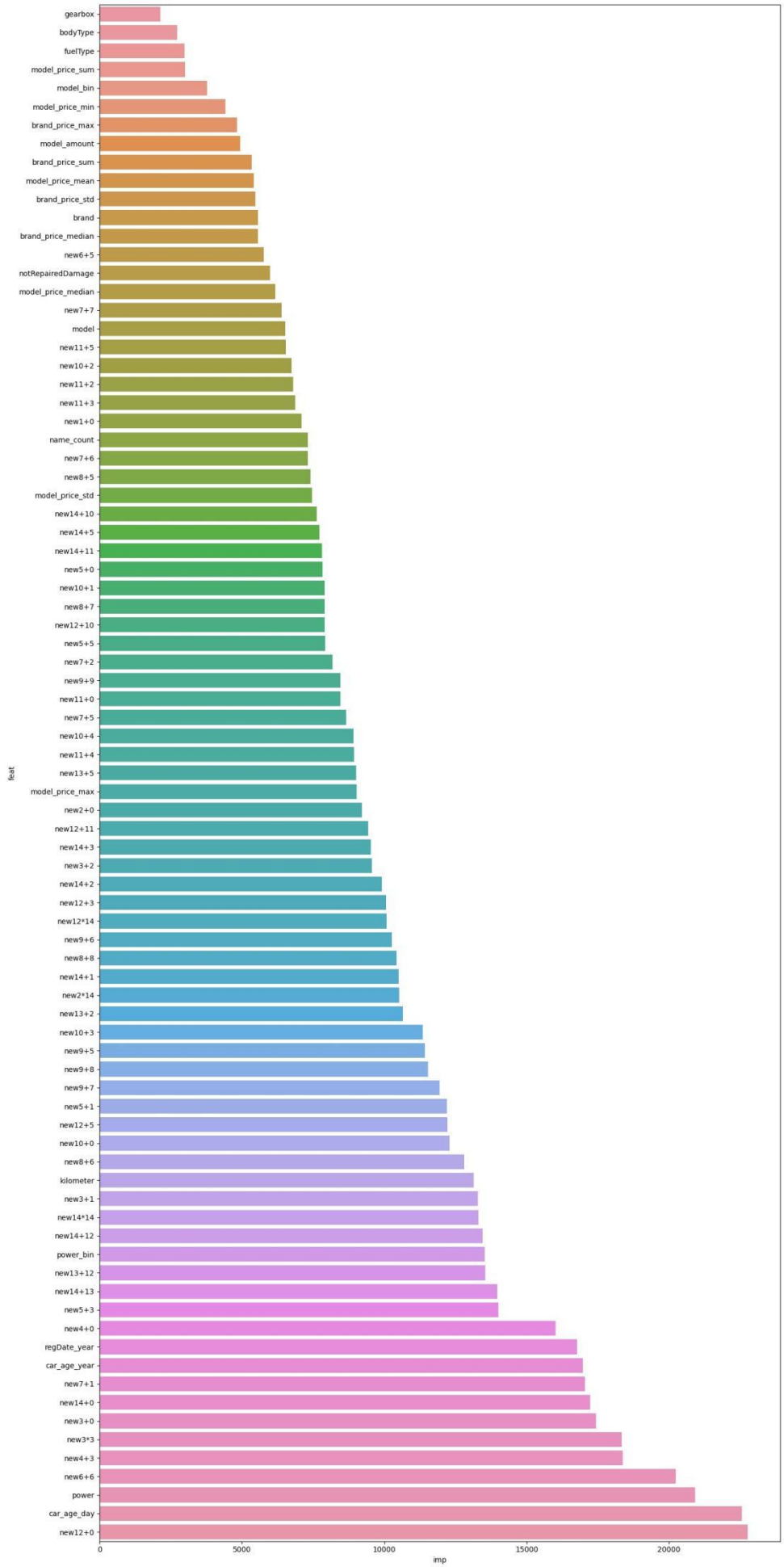
本次比赛，我选择的是 lightgbm+catboost+neural network。
本来也想使用 XGBoost 的，不过因为它需要使用二阶导，因此目标函数没有 MAE，并且尝试了逼近 MAE 的一些自定义函数效果也不理想，因此没有选择使用它。

经过上述的数据预处理以及特征工程：
树模型的输入有 83 个特征；神经网络的输入有 29 个特征。

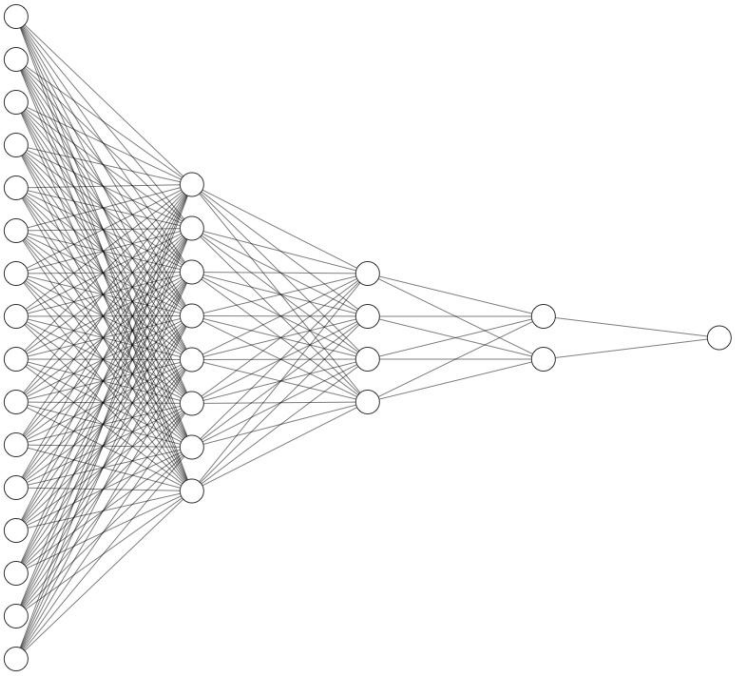
1、lightgbm 和 catboost：
因为它们都是树模型，因此我同时对这两个模型进行分析

第一：lgb 和 cab 的训练收敛速度非常快，比同样参数的 xgb 快非常多。
第二：它们可以处理缺失值，计算取值的增益，择优录取。
第三：调整正则化系数，均使用正则化，防止过拟合。
第四：降低学习率，获得更小 MAE 的验证集预测输出。
第五：调整早停轮数，防止陷入过拟合或欠拟合。
第六：均使用交叉验证，使用十折交叉验证，减小过拟合。
其他参数设置无明显上分迹象，以代码为准，不一一阐述。

以下为 lightgbm 对输入的 83 个特征的重要度排序。


gearbox
bodyType
fuelType
model_price_sum
model_bin
model_price_min
brand_price_max
model_amount
brand_price_sum
model_price_mean
brand_price_std
brand
brand_price_median
new6+5
notRepairedDamage
model_price_median
new7+7
model
new11+5
new10+2
new11+2
new11+3
new1+0
name_count
new7+6
new8+5
model_price_std
new14+10
new14+5
new14+11
new5+0
new10+1
new8+7
new12+10
new5+5
new7+2
new9+9
new11+0
new7+5
new10+4
new11+4
new13+5
model_price_max
new2+0
new12+11
new14+3
new3+2
new14+2
new12+3
new12*14
new9+6
new8+8
new14+1
new2*14
new13+2
new10+3
new9+5
new9+8
new9+7
new5+1
new12+5
new10+0
new8+6
kilometer
new3+1
new14*14
new14+12
power_bin
new13+12
new14+13
new5+3
new4+0
regDate_year
car_age_year
new7+1
new14+0
new3+0
new3*3
new4+3
new6+6
power
car_age_day
new12+0
feat
0
5000
10000
15000
20000
imp

2、neural network：

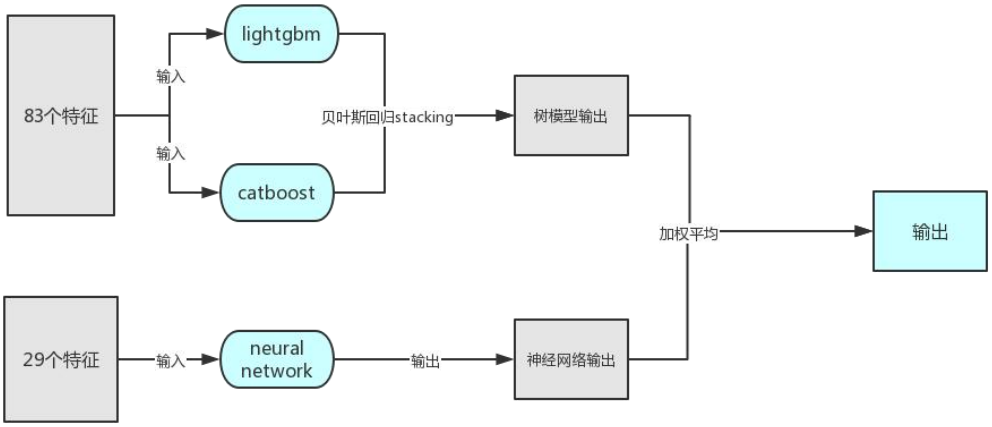


我针对该比赛，自己设计了一个五层的神经网络，大致框架如上图所示，但结点数由于太多只是展示部分结点画图。
以下为全连接层的结点个数设置，具体实现可参考代码。

接下来对神经网络进行具体分析：
第一：训练模型使用小 batchsize，512，虽然在下降方向上可能会出现小偏差，但是对收敛速度的收益大，2000 代以内可以收敛。
第二：神经网络对于特征工程这一类不用操心很多，就能达到与树模型相差无几的精度。
第三：调整正则化系数，使用正则化，防止过拟合。
第四：调整学习率，对训练过程的误差进行分析，选择学习率下降的时机进行调整。
第五：使用交叉验证，使用十折交叉验证，减小过拟合。
第六：选择梯度下降的优化器为 Adam，它是目前综合能力较好的优化器，具备计算高效，对内存需求少等等优点。

集成的方法

由于两个树模型的训练数据一样且结构相似，首先对两个树模型进行 stacking，然后再与神经网络的输出进行 mix。
由于树模型和神经网络是完全不同的架构，它们得到的分数输出相近，预测值差异较大，往往在 MAE 上差异为 200 左右，因此将他们进行 MIX 可以取到一个更好的结果，加权平均选择系数选择 0.5，虽然神经网络的分数确实会比树模型高一点点，但是我们的最高分是多组线上最优输出的结合，因此可以互相弥补优势。



给出的代码是一次输出的结果，如若完美复现线上结果，得多输出几次选取 Top-3 求平均。

二、代码说明

由于后期上分选择了十折交叉验证和非常小的学习率，运行较慢，大家可以先使用五折和较大学习率测试效果 ~

|--data

训练集、测试集，可从比赛官网下载

|--user_data

代码中途生成的一些文件，比赛过程中方便观察

|--prediction_result

输出的提交文本

|--feature

Tree_generation.py ——树模型训练数据的处理程序
NN_generation.py ——神经网络训练数据的处理程序
generation.py ——为以上两个代码的合成版本，生成两份数据

|--model

lgb_model.py ——lightgbm 模型训练代码
cab_model.py ——catboost 模型训练代码
nn_model.py ——神经网络模型训练代码
stack+mix.py ——二层 stack 和三层加权平均代码
model.py ——为以上四个代码的合成版本，输出测试集的预测数据

|--code

requirements.txt ——所使用的依赖
main.py ——主程序，一个代码，包括以上所有的步骤

执行：（进入该目录，执行以下命令即可产生一份预测数据）

python main.py PS：其实 main 是我把 feature 和 model 的代码全都复制扔了进去。