OpenSODA挑战赛T3：基于机器学习的OpenRank 指标拟合与优化

队伍：vision

项目B站链接：https://www.bilibili.com/video/BV1VN4y1Z7wQ/?spm_id_from=333.999.top_right_bar_window_default_collection.content.click&vd_source=efe075d2ac8f2a60dda4195a8241736d

README

- **项目结构**

```
├─data.zip            # 数据文件
|   main.ipynb     # 代码文件
|   requirements.txt          # 环境依赖
|   决赛-T3-vision.pdf       # 实验报告
|   README         # README文件
```

- **环境配置**

```
Windows 10
Python 3.8
Anaconda 4.10.1
Jupyter Notebook
```

- **运行方法**

1.创建conda环境

```
conda create -n opensoda python=3.8
conda activate opensoda
```

2.安装依赖

```
pip install -r requirements.txt
```

3.运行`main.ipynb`

创建好环境后，使用`Python`+`Anaconda`+`Jupyter Notebook`运行即可

一、项目简介

本项目是OpenSODA挑战赛的T3题目：基于机器学习的 OpenRank 指标拟合与优化

1.1 背景介绍

OpenDigger 项目中包含大量的各类指标数据，其中最核心的指标之一是项目的全域 OpenRank 指标。项目全域 OpenRank 指标是在整个 GitHub 上的开发者与仓库的协作关系网络之上，基于 OpenRank 算法得到的一个具有时序信息的网络指标，可以有效体现一个项目在开源生态中的协作影响力。除此以外，OpenDigger 项目还输出了大量的统计型指标，如 CHAOSS 指标体系中的各类指标。

相较而言，CHAOSS 指标体系中的统计型指标会更加细致的关注项目在各个方面可以采集到的统计数据，而全域 OpenRank 指标则更关注结果，即并不关心项目中的一些细节的统计数据，只关心最终有多少有影响力的开发者在项目中深度参与和活跃。因此我们可以用统计型指标来拟合 OpenRank 指标，从而解决如下一些问题：

- 对于拟合网络型指标，到底哪些统计型指标是更加重要的，这对于项目制定有可执行性的优化方案非常有益。
- 对于其他无法简单获得全域数据的情况，如 Gitee、GitLab 上的项目，可以通过统计指标来估计其 OpenRank 指标，从而可以与 GitHub 上的项目进行横向的比较。

1.2 任务介绍

该赛题的任务是使用 CHAOSS 统计型数据指标来拟合 OpenRank 网络指标，检验目标是拟合精度。

访问 OpenDigger 项目 GitHub README，可以看到目前已经发布的所有指标详情，可以直接通过 URL 访问不同项目的历史上每个月的指标数据。

使用 2022 全年全域 OpenRank 全球 Top300 项目为样本，使用其 2023 年之前的所有数据为训练集，2023 年 1 - 3 月的数据为验证集。

1.3 任务目标

该任务的核心目标是使用 CHAOSS 统计型指标拟合 OpenRank 网络指标，参赛者可自由选择拟合算法来进行实验，但预期得到如下结果：

- 使用上述 300 个项目的数据进行训练从而拟合 2023 年 1 - 3 月数据，误差率作为最终评判结果。
- 拟合算法需要具有一定的可解释性，即哪些统计型指标相对而言更加重要。
- 额外的考量可能带来更多加分，如：
- - 根据项目状态与阶段不同，可能拟合算法不同，即项目阶段不同，统计型指标的重要性会发生变化。这里的阶段可以是以 OpenRank 阈值为标准，或以其他合理的指标阈值为标准。
    - 统计型指标与 OpenRank 指标的拟合可能存在时间上的不同步，即统计指标的变化可能会延迟反映在 OpenRank 指标上

二、算法介绍

2.1 算法描述

2.1.1 时间序列预测

本项目实质是一个时间序列预测问题，关于时序预测问题，已经有很多类型的算法可以解决，主要有以下几种：

- **传统时序建模**

例如自回归模型（Autoregressive model, AR）、移动平均模型（Moving Average model，MA）、自回归滑动平均模型（Autoregressive moving average model，ARMA）、差分整合移动平均自回归模型（Autoregressive integrated moving average model，ARIMA）

传统时序建模对多变量时序预测效果一般

- **机器学习方法**

这类方法以 XGBoost[1]与LightGBM[2]为代表，一般就是把时序问题转换为监督学习，通过特征工程和机器学习方法去预测；这种模型可以解决绝大多数的复杂的时序预测模型。支持复杂的数据建模，支持多变量协同回归，支持非线性问题。

- **深度学习方法**

这类方法以RNN、CNN、Transformer等方法为主，例如RNN方法DeepAR[3]，CNN方法An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling[4]，Tranformer方法Informer[5]

2.1.1 主体算法

由于拟合算法需要具有一定的可解释性，即哪些统计型指标相对而言更加重要，因此主体算法使用机器学习算法，本次实验选择了性能较为优秀的机器学习算法XGBoost与LightGBM

- **XGBoost的优点如下：**

① 精度更高
GBDT 只用到一阶泰勒展开，而 XGBoost 对损失函数进行了二阶泰勒展开。XGBoost 引入二阶导一方面是为了增加精度，另一方面也是为了能够自定义损失函数，二阶泰勒展开可以近似大量损失函数；

② 灵活性更强
GBDT 以 CART 作为基分类器，XGBoost 不仅支持 CART 还支持线性分类器，使用线性分类器的 XGBoost 相当于带L_1和L_2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。此外，XGBoost 工具支持自定义损失函数，只需函数支持一阶和二阶求导；

③ 正则化
XGBoost 在目标函数中加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、叶子节点权重的L_2范式。正则项降低了模型的方差，使学习出来的模型更加简单，有助于防止过拟合，这也是XGBoost优于传统GBDT的一个特性。

④ Shrinkage（缩减）
相当于学习速率。XGBoost 在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。传统GBDT的实现也有学习速率；

⑤ 列抽样
XGBoost 借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算。这也是XGBoost异于传统GBDT的一个特性；

⑥ 缺失值处理
对于特征的值有缺失的样本，XGBoost 采用的稀疏感知算法可以自动学习出它的分裂方向；

⑦ XGBoost工具支持并行
XGBoost的并行是在特征粒度上的。XGBoost在训练之前，预先对数据进行了排序，然后保存为block结构，后面的迭代中重复地使用这个结构，显著减小计算量。在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，因而各个特征的增益计算可以多线程并行计算。

⑧ 可并行的近似算法
树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以XGBoost还提出了一种可并行的近似算法，用于高效地生成候选的分割点。

⑨ 剪枝
xgboost先从顶到底，建立所有可以建立的子树，再从底到顶进行反向剪枝。比起GBM，这样不容易陷入局部最优解

⑩ 内置交叉验证
xgboost允许在每一轮boosting迭代中使用交叉验证。

- **XGboost 的不足之处：**

① 每轮迭代时，都需要遍历整个训练数据多次。如果把整个训练数据装进内存则会限制训练数据的大小；如果不装进内存，反复地读写训练数据又会消耗非常大的时间。

② 预排序方法的时间和空间的消耗都很大

- **LightGBM的优点如下：**

为了避免上述 XGBoost 的缺陷，并且能够在不损害准确率的条件下加快模型的训练速度，LightGBM 在传统的 GBDT 算法上进行了如下优化：

① 基于 Histogram 的决策树算法。

② 单边梯度采样 Gradient-based One-Side Sampling(GOSS)：减少大量小梯度的数据实例，在计算信息增益的时候只利用剩下的具有高梯度的数据，从而节省时间空间开销

③ 互斥特征捆绑 Exclusive Feature Bundling(EFB)：使用 EFB 可以将许多互斥的特征绑定为一个特征，这样达到了降维的目的。

④ 带深度限制的 Leaf-wise 的叶子生长策略：大多数 GBDT 工具使用低效的按层生长 (level-wise) 的决策树生长策略，因为它不加区分的对待同一层的叶子，带来了很多没必要的开销。实际上很多叶子的分裂增益较低，没必要进行搜索和分裂。LightGBM 使用了带有深度限制的按叶子生长 (leaf-wise) 算法。

⑤ 直接支持类别特征(Categorical Feature)

⑥ 支持高效并行

⑦ Cache 命中率优化

- **LightGBM的不足之处**

① Boosting族是迭代算法，每一次迭代都根据上一次迭代的预测结果对样本进行权重调整，所以随着迭代不断进行，误差会越来越小，模型的偏差（bias）会不断降低。由于LightGBM是基于偏差的算法，所以会对噪点较为敏感；

② 在寻找最优解时，依据的是最优切分变量，没有将最优解是全部特征的综合这一理念考虑进去；

2.1.2 算法流程

本次项目主要的算法流程如下：

- 首先进行数据预处理，包括指标分类处理、时间处理、数据整合、异常值填充与训练集验证集划分
- 接着进行特征工程，包括查看基本信息、特征总体分布图、目标变量openrank的分布情况、输入变量与目标变量之间的关系、相关性热力图与相关性表、降维方法的初步探究以及考虑滞后特征
- 然后是模型建立与训练，包括模型选择、参数设置、交叉验证、评价指标、可解释性的探究、降维的进一步探究以及根据项目状态与阶段分类拟合的问题
- 实验结果与分析部分将会展示上述实验的结果并进行分析，包括实验环境设置、实验结果、结果分析与讨论

- 最后将给出结论

注：项目代码（带注释）将会附加在最后

2.2 实现细节

本节将会详细阐述项目使用算法的实现细节以及整个项目流程

2.2.1 数据集概述

- **背景介绍**

2020 年，由于新冠疫情的影响，越来越多的人开始远程工作，GitHub 也成为了许多开发者的首选平台。在 2020 年，GitHub 上有数百万个开源项目，涵盖了各种技术领域，其中包括了很多热门的项目。

GitHub Top 300 是指使用 2022 全年全域 OpenRank 全球 Top300 项目，这些项目通常拥有大量的贡献者和用户。这些项目涵盖了各种语言、各种应用场景，是当前最受欢迎的开源项目。

- **数据集概述**

本次实验仅需要使用top300_metrics数据集，即Top 300 仓库的 OpenDigger 的指标数据，包括 OpenRank 值，指标简短说明如下（参考 OpenSODA挑战赛官方数据集）,详细说明可参照 OpenDegger 官方文档：https://github.com/X-lab2017/open-digger/

文件名称	描述
active_dates_and_times.json	项目每天的活跃度
activity_details.json	每人每天的活跃度
activity.json	项目每月的活跃度
attention.json	每月的关注度
bus_factor_detail.json	每人每月的巴士系数
bus_factor.json	项目每月的巴士系统
change_request_age.json	PR请求的生命时长（没有关闭的PR默认到23年3月份）
change_request_resolution_duration.json	PR请求从创建到结束的
change_request_response_time.json	PR请求从创建到首次响应的时长
change_requests_reviews.json	PR审阅者的数量
change_requests.json	PR的数量
code_change_lines_add.json	代码添加的行数
code_change_lines_remove.json	代码减少的行数
code_change_lines_sum.json	代码总变更数
contributor_email_suffixes.json	贡献者邮箱后缀

inactive_contributors.json	不活跃开发者数
issue_age.json	issue的生命时长（没有关闭的PR默认到23年3月份）
issue_comments.json	issue的评论数量
issue_resolution_duration.json	issue的开启到关闭的时长
issue_response_time.json	issue从开始到首次响应的时长
issues_and_change_request_active.json	issue和request的数量
issues_closed.json	issue关闭的数量
issues_new.json	issue创建的数量
new_contributors_detail.json	新增加的贡献者名单
new_contributors.json	新增加的贡献者数量
openrank.json	openrank值
participants.json	项目参与者人数
stars.json	start的数量
technical_fork.json	fork数量

- **数据集格式**

top300_metrics格式如下：

```
--top_300_metrics
    --organization_name
        --project_name
            --metric_name.json
```

其中每个组织下可能有一个或多个项目，每个项目都有所有指标数据

2.2.2 数据预处理

- **指标分类处理**

首先本次实现是用CHAOSS 统计型数据指标来拟合 OpenRank 网络指标，因此只需要考虑CHAOSS指标和OpenRank指标；

指标数据中，有一些指标的数据是可以直接处理使用的，类似`activity.json`中一个时间点一个数据的格式；

{"2015-01":30.04,"2015-02":25.96,"2015-03":28.2,"2015-04":31.81,"2015-05":31.18,"2015-06":21.5,

这些指标如下：

```
"openrank",
"technical_fork",
"new_contributors",
"inactive_contributors",
"bus_factor",
"issues_new",
"issues_closed",
"code_change_lines_add",
"code_change_lines_remove",
"code_change_lines_sum",
"change_requests",
"change_requests_accepted",
"change_requests_reviews"
```

但是还有一些是无法直接使用的，例如`issue_response_time.json`中有指标数据的一些统计信息，如均值：

{"avg":{"2015-01":1.67,"2015-02":2.18,"2015-03":1.54,"2015-04":5.45,"2015-05":1,

水平：

"levels":{"2015-01":[55,1,0,1],"2015-02":[40,3,0,1],"2015-03":[38,1,1,1]

与多个分位数`quantile_0`、`quantile_1`、`quantile_2`、`quantile_3`、`quantile_4`

本次实验取均值avg；

并且`active_dates_and_times`指标数据每个数据点形式是列表，不方便后序处理，需要对列表取均值

{"2015-01":[0,0,0,0,0,0,0,1,1,3,4,2,2,1,4,3,7,7,2,6,0,4,0,0,0,0,0,0,0,0,0,0,3,1,0,1,1,3,3,7,0,3,1,1,1,1,1,1,1,6,0,0,0,1,

这些指标如下：

```
"active_dates_and_times",
 "issue_response_time",
 "issue_resolution_duration",
 "issue_age",
 "change_request_response_time",
 "change_request_resolution_duration",
 "change_request_age"
```

- **时间处理**

由于本次实验是在时间序列上进行预测，因此时序数据后续需要作为特征参与学习，因而对时间数据采用以下处理方式：

原始的时间数据格式：`yyyy-mm`，如`2015-01`

将时间数据处理为距离最早时间的月份间隔，例如`2015-01`是最早时间，那么就记为0，`2015-05`就是4，以此列推，具体如下：

```
0      1
1      2
2      3
3      4
4      5
      ..
86    95
87    96
88    97
89    98
90    99
Name: Time, Length: 21572, dtype: int64
```

- **数据整合**

最终的数据要将每个项目的数据进行合并，合并方法是直接拼接

- **异常值填充**

对于NaN这类异常值用0进行填充

- **训练集验证集划分**

由于使用其 2023 年之前的所有数据为训练集，2023 年 1 - 3 月的数据为验证集，因此按照时间划分训练集验证集

对于输入特征变量X，将数据原始时间列time和待拟合指标openrank去除；对于目标变量y，即为待拟合指标openrank

2.2.3 特征工程

- **基本信息：**

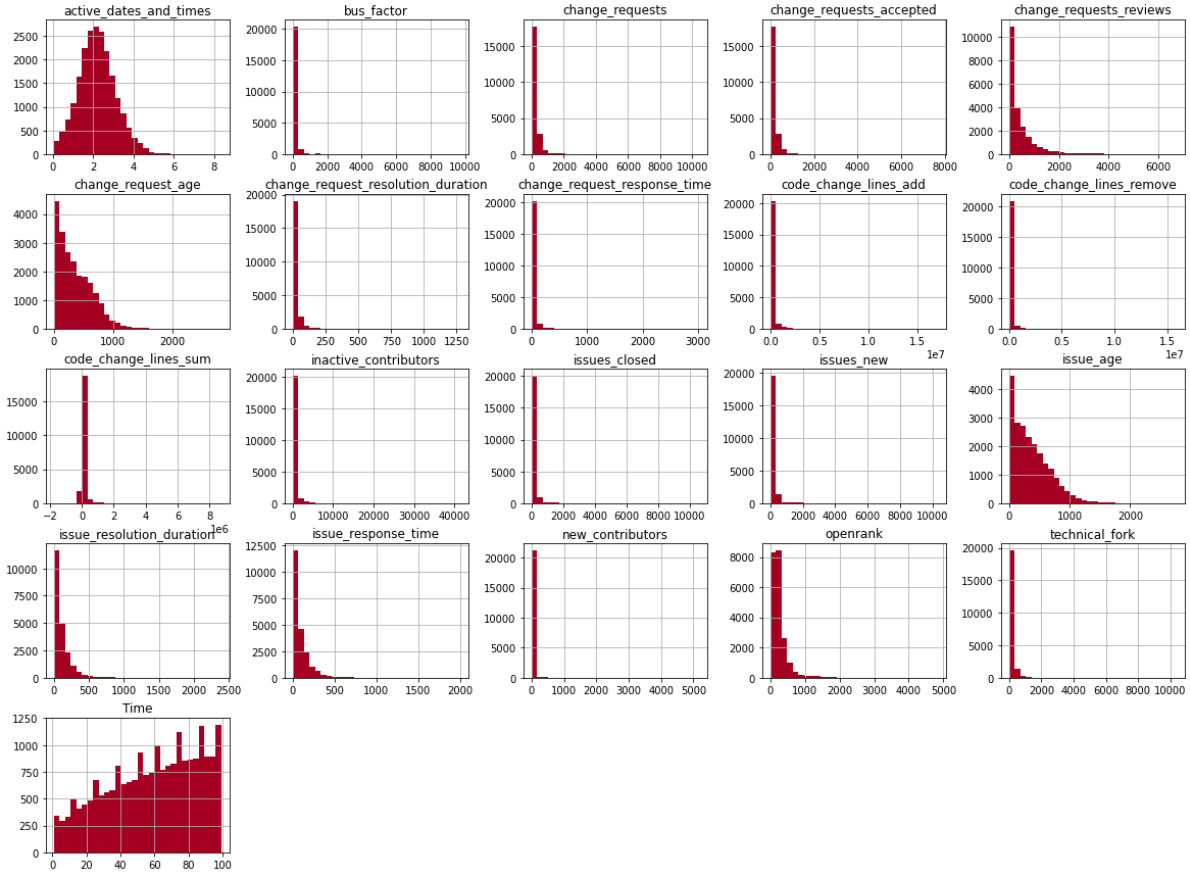
总数据量	特征维度	训练集数据量	测试集数据量
21572	21	20678	894

所有特征：

```
'active_dates_and_times',
'bus_factor',
'change_requests',
'change_requests_accepted',
'change_requests_reviews',
'change_request_age',
'change_request_resolution_duration',
'change_request_response_time',
'code_change_lines_add',
'code_change_lines_remove',
'code_change_lines_sum',
'inactive_contributors',
'issues_closed',
'issues_new',
'issue_age',
'issue_resolution_duration',
'issue_response_time',
'new_contributors',
'openrank',
'technical_fork',
```

```
'Time'
```

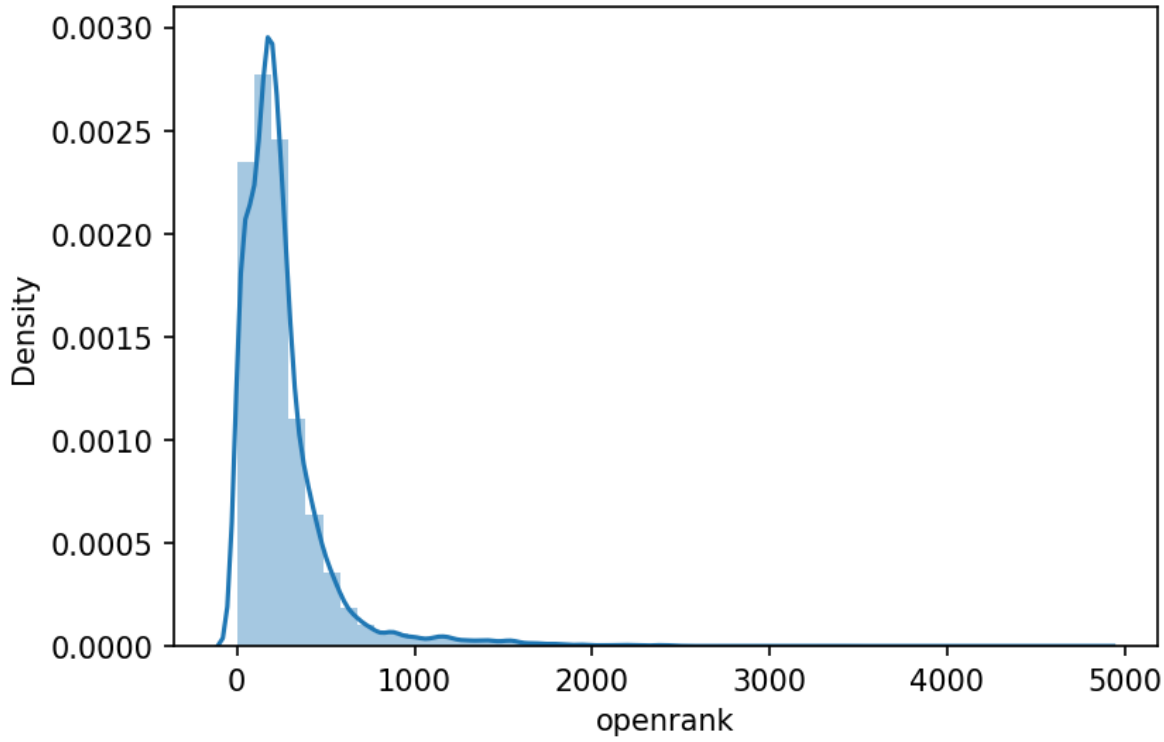
- **特征总体分布图**



- **目标变量openrank的分布情况**

```
count    21572.000000
mean       251.546959
std        258.252390
min          0.000000
25%        106.840000
50%        194.940000
75%        301.190000
max       4829.380000
Name: openrank, dtype: float64
```

分布图



可以发现该分布是一个近似正态分布，也是一个长尾分布，虽然最大值和最小值之间相差较多，但是主要还是集中于0-500之间，后续会有按照openrank阈值确定不同阶段使用不同算法进行拟合的探究

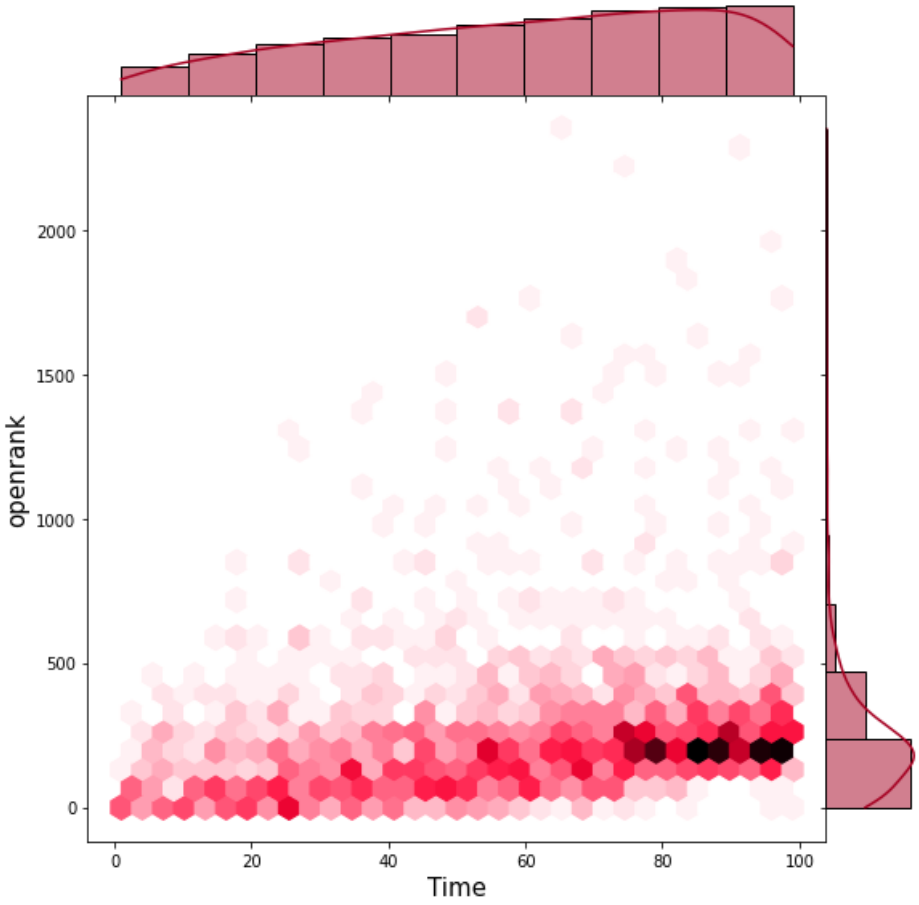
- **输入变量与目标变量之间的关系**

由于变量过多，这里仅选取一些变量进行探究

① 时序数据和Openrank值的关系

绘制时序变量Time和openrank的关系图

注：由于将数据量整体绘制会较为密集，影响观感，因此每隔10条数据采样绘制

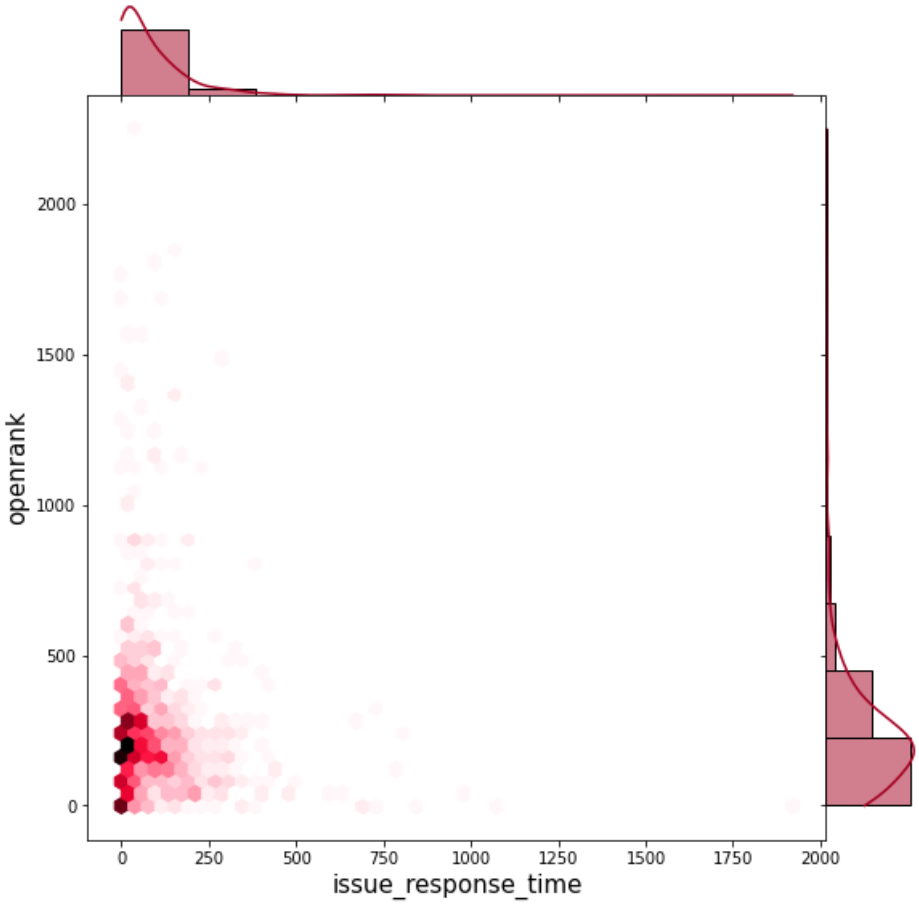


可以发现随着时间增长，更高的openrank值出现的总体比例/次数也在增长，原因是本次实验选取的是2022年Top300的项目，意味着大部分项目在2023年大概率不会处于下坡的状态，因此随着时间的推移，大部分的项目在开源生态中的协作影响力都是增长的

② issue从开始到首次响应的时长issue_response_time和Openrank值的关系

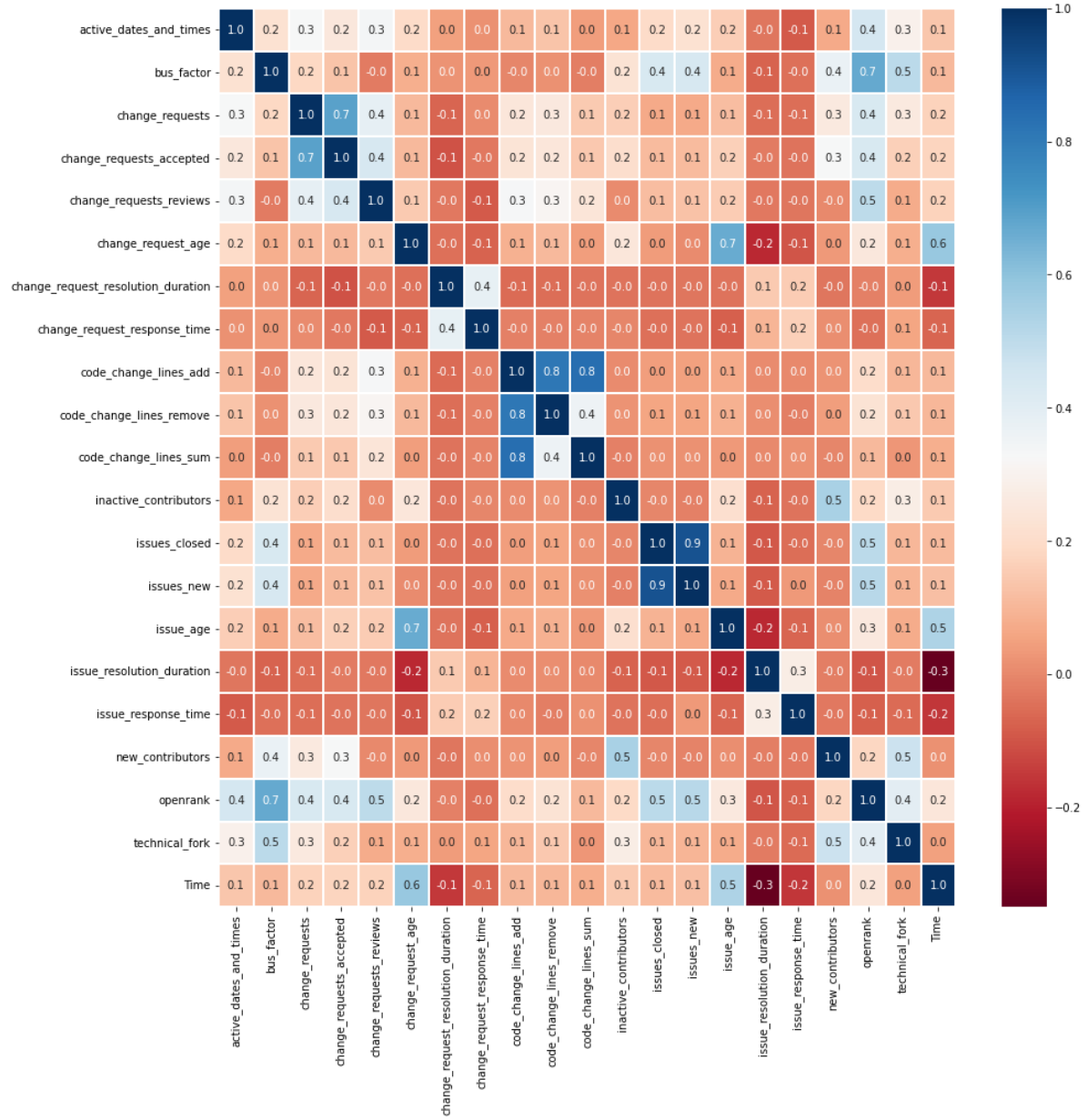
绘制时序变量Time和openrank的关系图

注：由于将数据量整体绘制会较为密集，影响观感，因此每隔20条数据采样绘制



可以发现随着issue_response_time的降低，更高的openrank值出现的总体比例/次数也在增长，可见openrank与issue_response_time有明显的负相关关系（即issue_response_time降低，openrank升高）

- **相关性热力图**



同时查看openrank和其他变量的相关性表（降序）

```
openrank                                1.000000
bus_factor                              0.673038
issues_new                              0.511024
issues_closed                           0.506355
change_requests_reviews                 0.504181
active_dates_and_times                  0.442392
change_requests                         0.435062
change_requests_accepted                0.433368
technical_fork                          0.397825
issue_age                               0.256412
change_request_age                      0.246644
Time                                    0.242044
code_change_lines_remove                0.228507
code_change_lines_add                   0.201539
inactive_contributors                   0.190880
new_contributors                        0.182152
code_change_lines_sum                   0.108429
change_request_resolution_duration     -0.015687
change_request_response_time           -0.046676
issue_response_time                    -0.079464
issue_resolution_duration              -0.096710
Name: openrank, dtype: float64
```

相关性热力图和相关性表可以给出一个直观的相关性关系，可以发现:

① openrank值和bus_factor的相关性最高，其次是issue_new，issue_closed和change_requests_reviews

② openrank值和change_request_resolution_duration，change_request_response_time, issue_response_time以及issue_resolution_duration这几个变量都是负相关的

③ 由相关性热力图上可以发现有一些变量之间存在较高的相关性，比如：issue_age和change_request_age等，这些变量之间的高相关性即为多重共线性，可能会对后续模型产生影响

- **降维（一）**

多重共线性

定义：多重共线性(Multicollinearity)是指线性回归模型中的解释变量之间由于存在精确相关关系或高度相关关系而使模型估计失真或难以估计准确

由2.2.3节可以知道，数据可能存在多重共线性，下面要验证是否存在多重共线性

通常根据方差扩大因子和特征值两类指标来判断数据是否存在多重共线性，这里使用特征值判断

判断方法是：计算$X'X$的特征值，假设最大特征值和最小特征值分别为$\lambda_{max}, \lambda_{min}$

则X的条件数为$\lambda_{max}/\lambda_{min}$

判断多重共线性的方法是看条件数是否大于某一个阈值，条件数越大多重共线性的程度越深

计算出的特征值和条件数如下：

```
eigenvalues:[5.63260693e+15 7.42864447e+14 9.16954902e+10 1.53105590e+10
 4.76855219e+09 3.49378864e+09 2.98036592e+09 1.41436661e+09
 1.01450620e+09 7.06296430e+08 5.86956270e+08 5.27934196e+08
 3.43773036e+08 1.98743854e+08 1.69985188e+08 1.27092454e+08
 4.97176130e+07 3.22354330e+07 1.60658407e+07 1.93160303e+04
 7.85229889e-03]
Condition_Number:846947151.4823266
```

显然条件数很大，数据存在多重共线性

解决方法

可以使用主成分分析（Principal Component Analysis，PCA）进行降维，主成分分析的原理是通过正交变换将一组可能存在相关性的变量转换为一组线性不相关的变量，转换后的这组变量叫主成分，从而解决多重共线性的问题；

更为简单的方法是使用不受多重共线性影响的机器学习模型建模，注意到多重共线性主要是线性层面的影响，如果模型的非线性建模能力较强，则自变量之间的相关性对模型性能没有影响，典型的比如XGBoost、LightGBM，这些模型的非线性建模能力较强，不受数据多重共线性的影响。

冗余特征

但是及时使用不受多重共线性影响的机器学习模型进行建模，数据中仍然存在冗余特征，即这些特征的信息量较少，与待拟合指标的相关性较低，重要性较低，去掉这些特征对模型拟合的结果几乎没有影响，那么去掉这些冗余特征可以加速模型的训练速度。

这些冗余特征的进一步探究等到初步训练结束后根据可解释性，会在报告的后续章节进一步探究。

- **滞后特征**

考虑滞后特征是时间序列预测中常见的特征工程，滞后性即当前时间步的特征变量对目标变量的影响，不一定在当前时间步反应出来，可能会有滞后影响，因此添加滞后变量相当于考虑这一点。本实验中，统计型指标与 OpenRank 指标的拟合可能存在时间上的不同步，即统计指标的变化可能会延迟反映在 OpenRank 指标上，因此将统计型指标的特征进行滞后可以更好地建模统计型指标和OpenRank指标的关系。

添加滞后特征的方法：将某个时间步的特征值向后移动一个或多个时间步并加入新时间步的特征

比如：使用一阶滞后特征，查看issues_new和其滞后特征issues_new_Lag

	issues_new	issues_new_Lag
0	60.0	0.0
1	45.0	60.0
2	43.0	45.0
3	83.0	43.0
4	74.0	83.0
...	...	...
21564	53.0	49.0
21565	81.0	53.0
21566	77.0	81.0
21567	79.0	77.0
21568	65.0	79.0

20678 rows × 2 columns

2.2.4 模型建立与训练

- **模型选择**

选择XGBoost与LightGBM两种模型

- **参数设置**

初始参数与调参的过程暂且不展示，调参方法可以选择控制参数变量逐一手动调参，或者网格化搜索（这次实验的数据量不算小，网格化搜索花费的时间远大于手动调参）

直接给出最终的调参结果：

XGBoost

```
'colsample_bytree': 1,
'colsample_bylevel': 1,
'learning_rate': 0.06,
'max_depth': 9,
'alpha': 10,
'subsample':1,
'min_child_weight':4,
'gamma':0.2,
'reg_alpha':0.1,
'reg_lambda':0.3,
'scale_pos_weight':1
```

详细的参数解释参考XGBoost官方文档

LightGBM

```
'colsample_bytree': 1,
'learning_rate': 0.06,
'max_depth': 9,
'alpha': 10,
'subsample':1,
'min_child_weight':4,
'reg_alpha':0.1,
'reg_lambda':0.3,
'scale_pos_weight':1,
"verbose": -1
```

详细的参数解释参考LightGBM官方文档

- **交叉验证**

XGBoost和LightGBM都提供交叉验证的方法，具体而言就是将训练集进一步划分为训练集和验证集，这里使用5折交叉验证，训练轮次分别为100,200,500,1000轮，并且如果10轮性能不提升就提前结束。

交叉验证过程(部分)
XGBoost

	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	337.048695	1.724691	337.503398	6.988901
1	317.852997	1.615686	318.785467	6.622925
2	299.796624	1.515855	301.324411	6.364285
3	282.828976	1.431963	284.786951	6.088244
4	266.874567	1.362039	269.250920	5.901852
..	...	...	...	...
995	2.233032	0.238008	55.466037	18.297090
996	2.225733	0.238796	55.465884	18.297102
997	2.218138	0.239567	55.465612	18.297204
998	2.211589	0.238959	55.465543	18.297178
999	2.205117	0.239397	55.465065	18.296765

[1000 rows x 4 columns]

LightGBM

```
{'rmse-mean': [244.6411062896056,
  232.97645700951847,
  222.1525221410267,
  211.94690069889694,
  202.4080645530158,
  193.44757658082673,
  184.9808698645916,
  177.0999371378634,
  169.61351123681644,
  162.70487194945878,
  156.28868801149946,
  150.1931518909448,
  144.5786361214999,
  139.3533204628824,
  134.28171727874286,
```


训练结果在第三节进行展示

- **评价指标**

常用回归问题评价指标如下：

均方误差（Mean Square Error, MSE）

均方根误差（Root Mean Square Error, RMSE)

平均绝对误差（Mean Absolute Error, MAE）

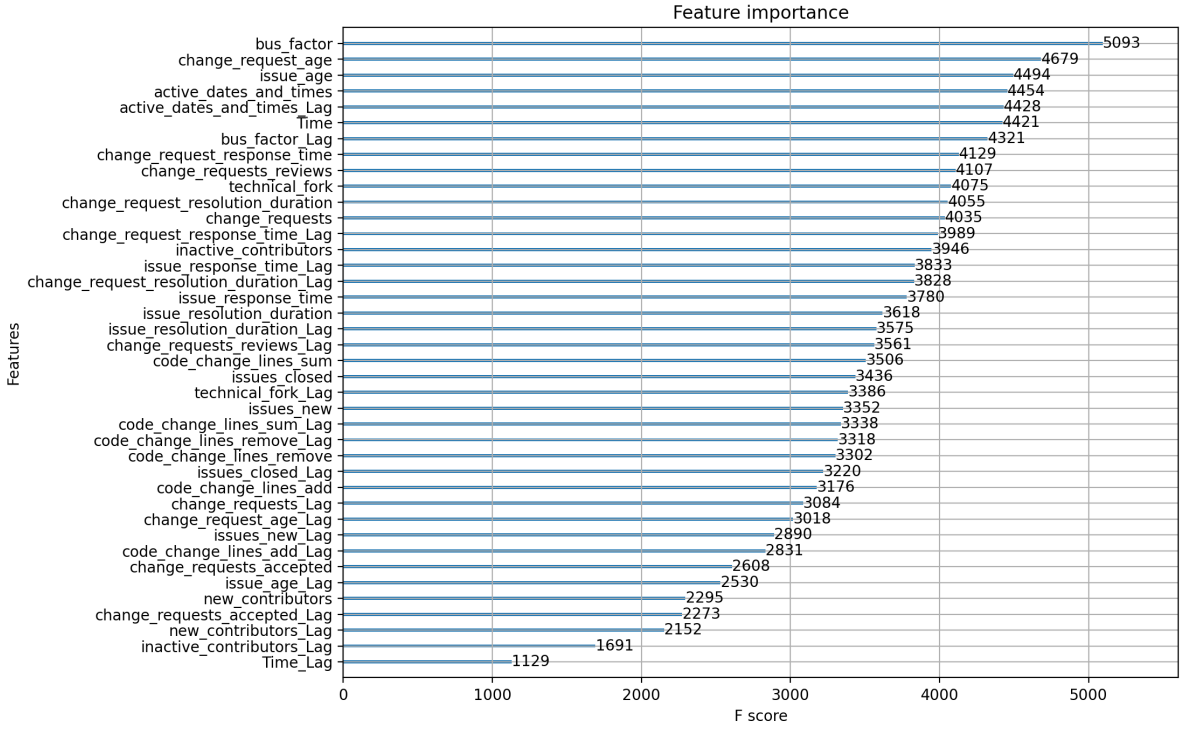
拟合优度R^2（R square)

几个指标各有局限性，因此综合考虑较为合理

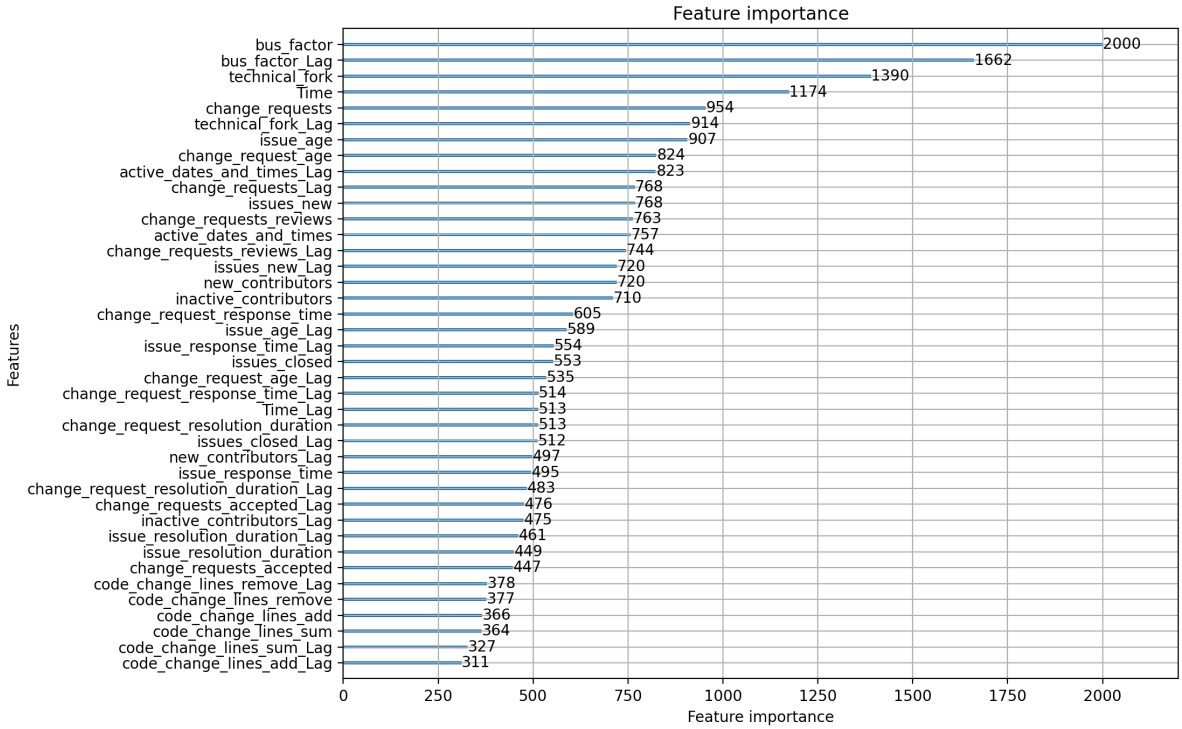
2.2.5 可解释性

拟合算法需要具有一定的可解释性，即哪些统计型指标相对而言更加重要。机器学习算法的可解释性有以下两个方法：

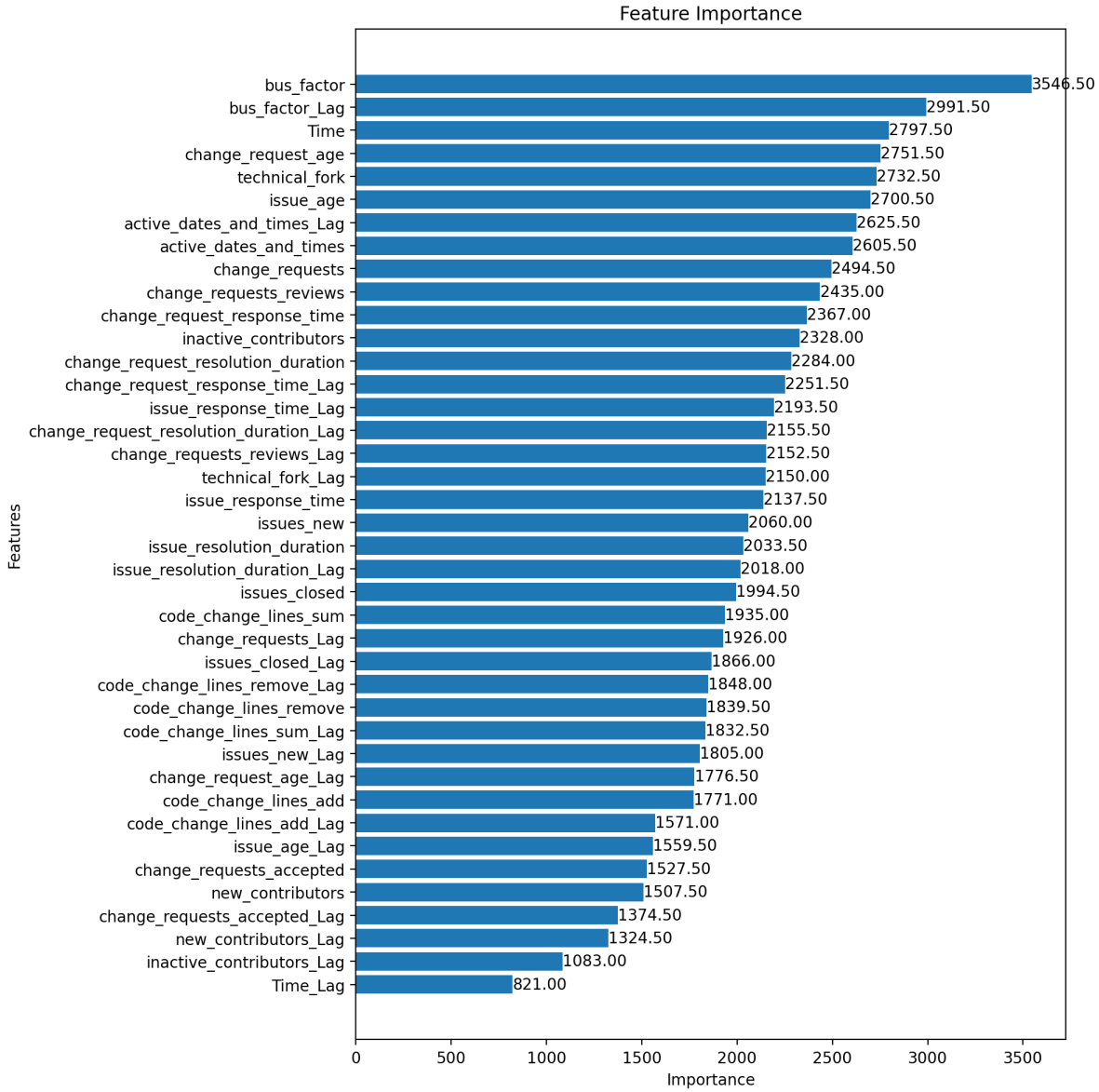
- **XGBoost自带的特征重要性**



LightGBM自带的特征重要性



两张图得出的特征重要性有一定的区别，将两张图的特征重要性进行平均，绘制平均后的特征重要性：



可以发现，bus_factor和该特征的滞后特征最为重要，时序特征是次要重要的特征，验证了时间序列预测时序特征的重要性；紧随其后的是change_request_age与technical_fork，其余重要性排名前十的特征为：issue_age，active_dates_and_times_Lag，active_dates_and_times，change_requests，change_requests_reviews

较为不重要的特征为：Time_Lag（很容易发现时序特征本身的滞后特征对待拟合指标不会有较大影响），inactive_contributors_Lag，new_contributors_Lag，change_requests_accepted_Lag，new_contributors，可以发现的是，new_contributors及其滞后特征均对OpenRank值的拟合没有较大作用。

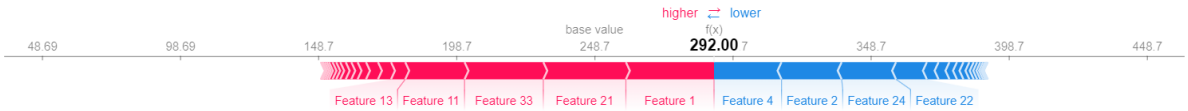
- **机器学习可解释性分析库：SHAP**

SHAP 属于模型事后解释的方法，它的核心思想是计算特征对模型输出的边际贡献，再从全局和局部两个层面对“黑盒模型”进行解释。SHAP构建一个加性的解释模型，所有的特征都视为“贡献者”。对于每个预测样本，模型都产生一个预测值，SHAP value就是该样本中每个特征所分配到的数值。

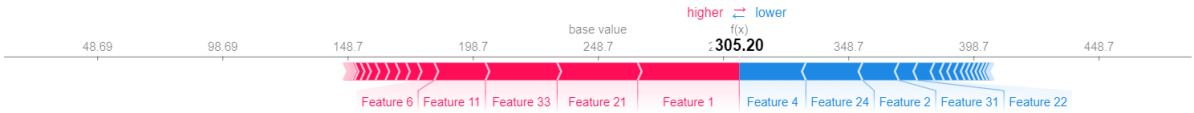
force_plot

单个样本每个特征的SHAP value贡献度

- **XGBoost**

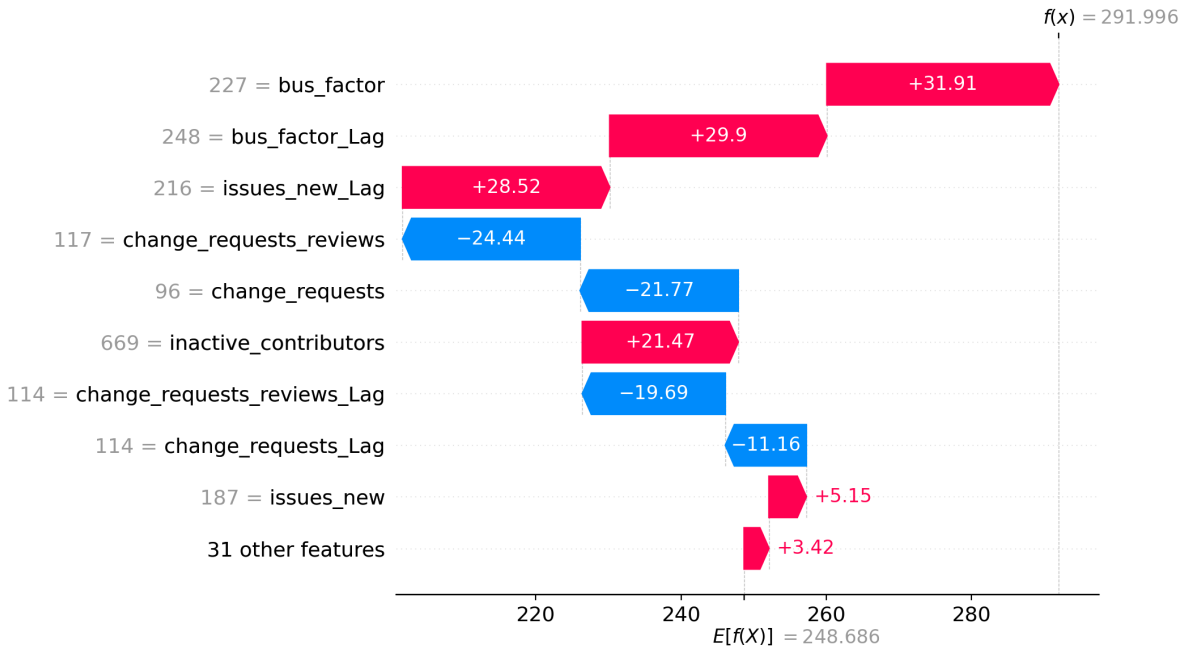


- **LightGBM**

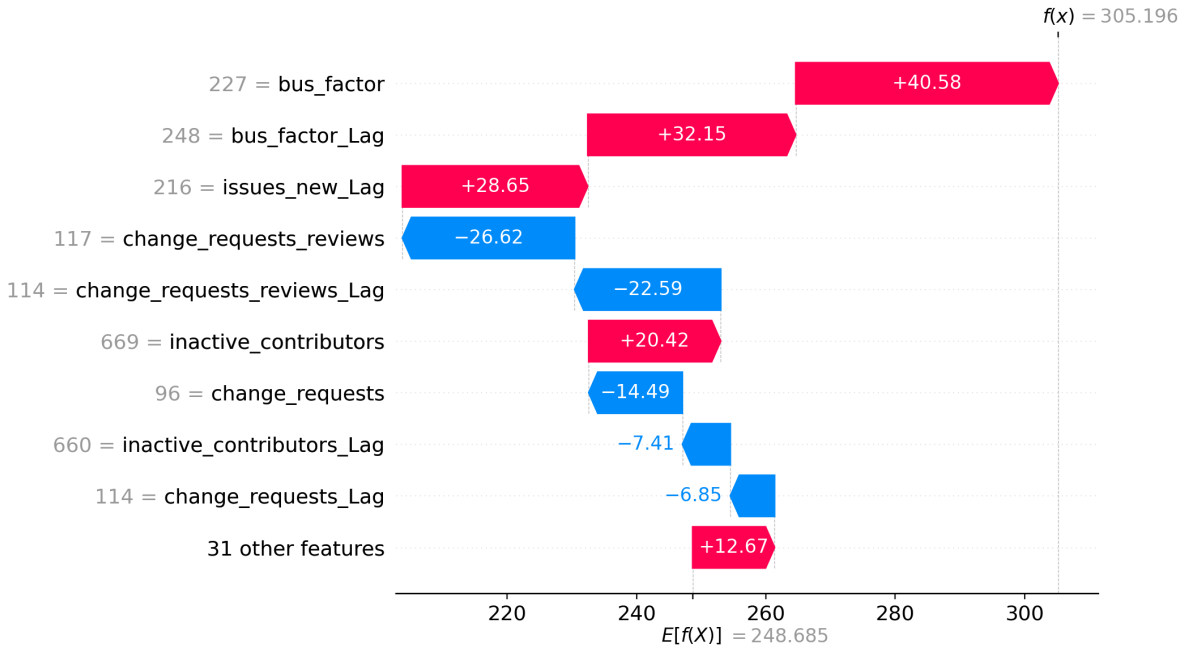


柱状图版本

- **XGBoost**



- **LightGBM**

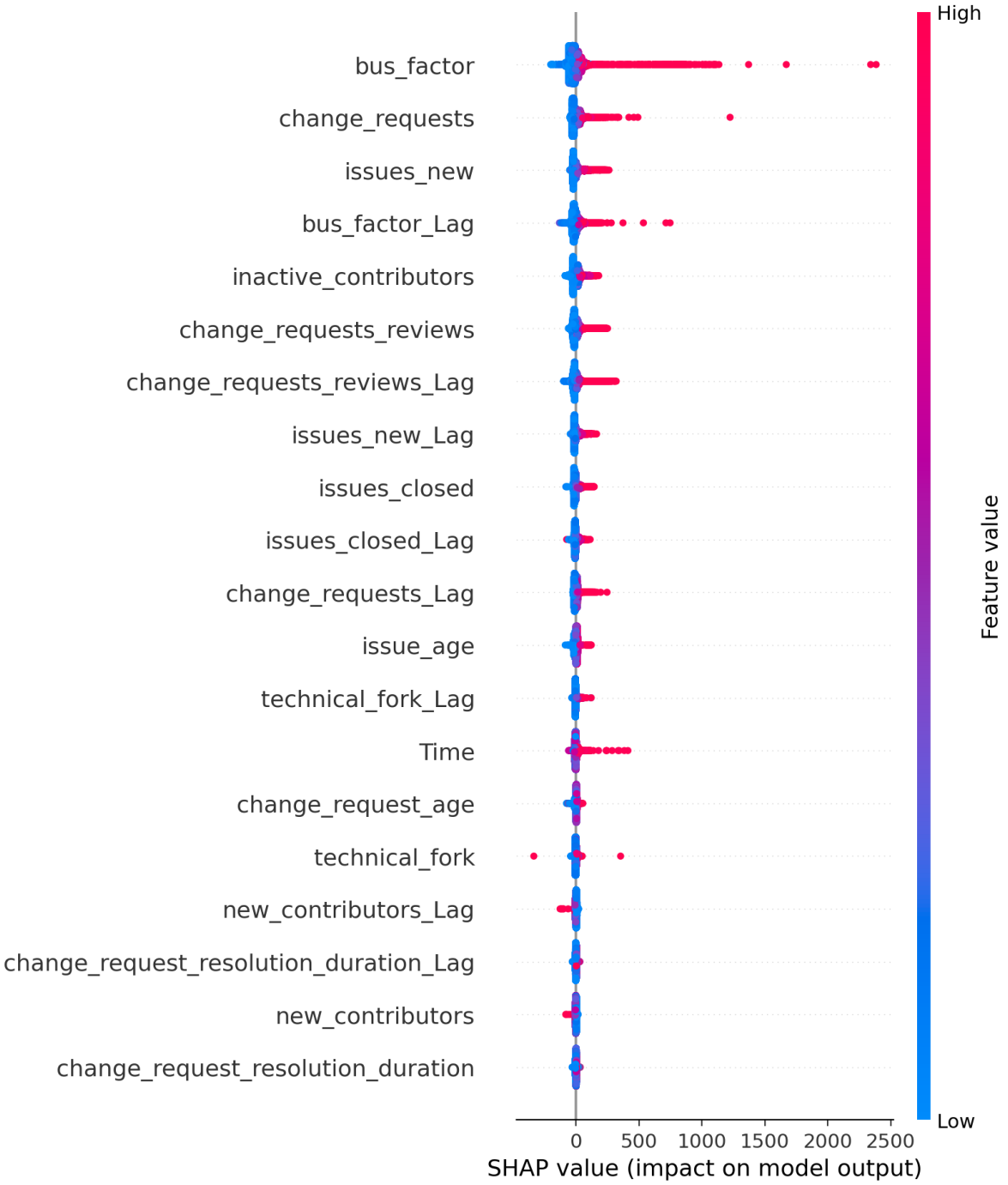


summary_plot

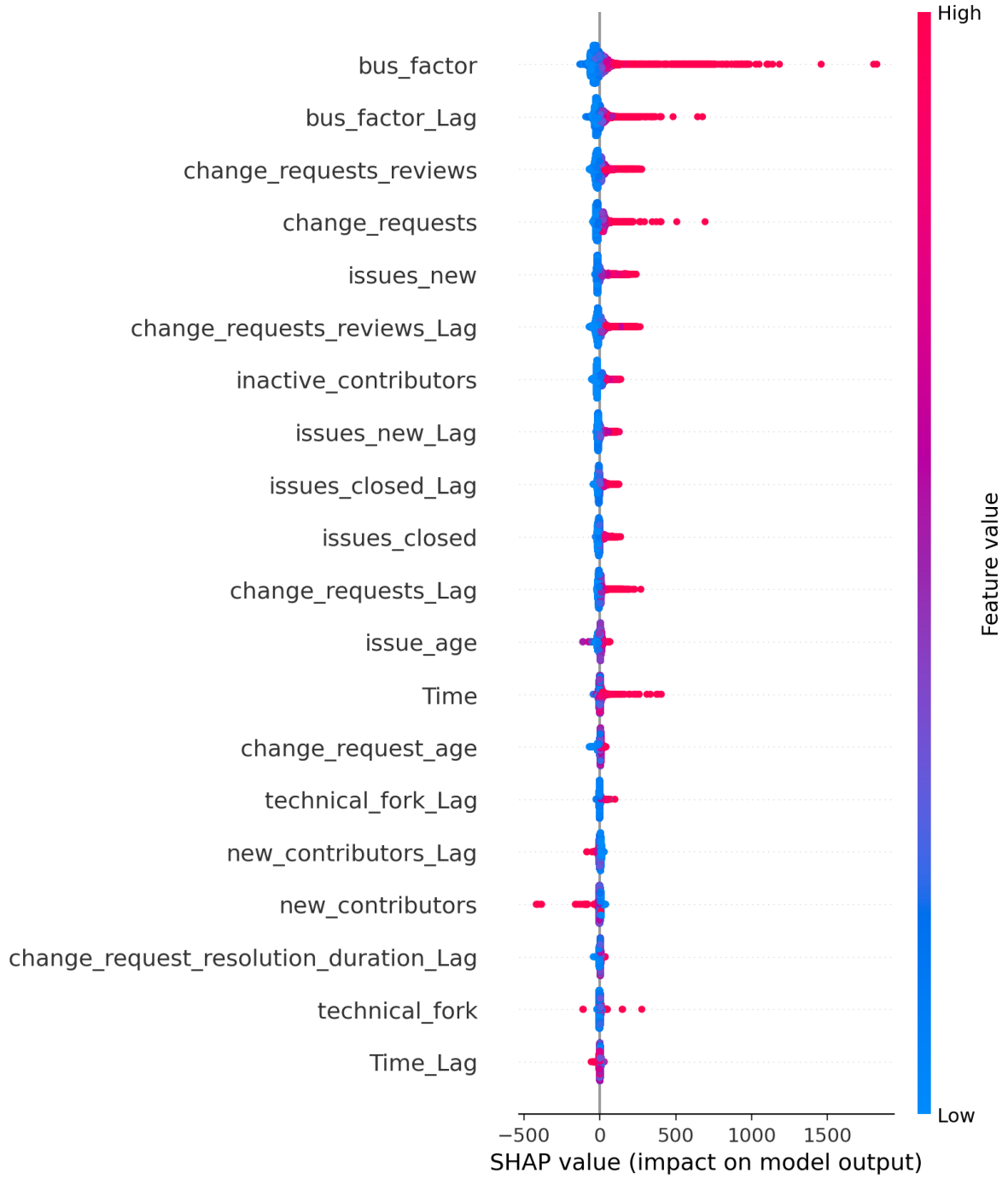
为每个样本绘制SHAP值，每一行代表一个特征，横坐标为SHAP值。一个点代表一个样本，颜色表示特征值(红色高，蓝色低)

注：这里只展示SHAP value较大的特征

- **XGBoost**

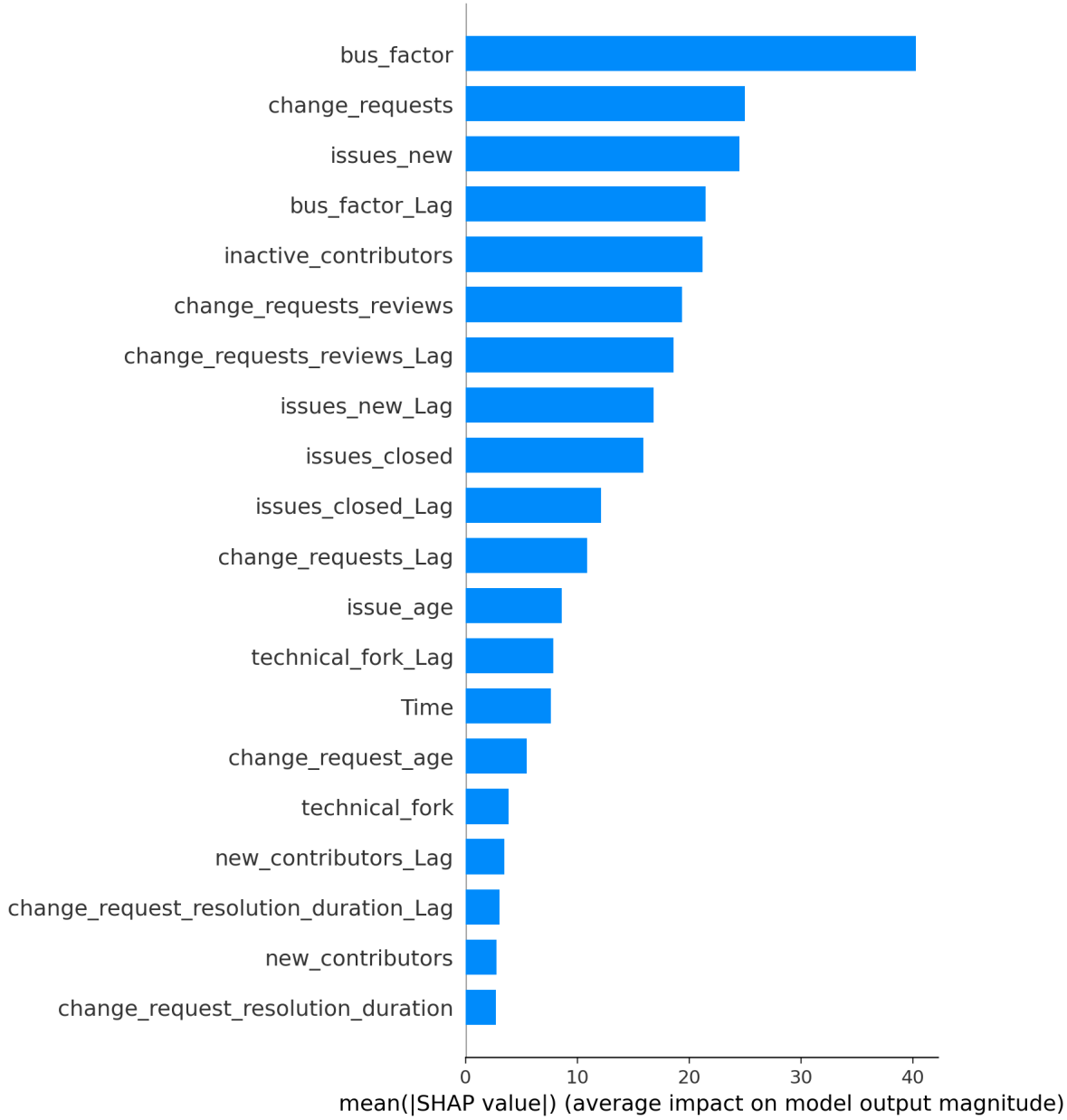

bus_factor
change_requests
issues_new
bus_factor_Lag
inactive_contributors
change_requests_reviews
change_requests_reviews_Lag
issues_new_Lag
issues_closed
issues_closed_Lag
change_requests_Lag
issue_age
technical_fork_Lag
Time
change_request_age
technical_fork
new_contributors_Lag
change_request_resolution_duration_Lag
new_contributors
change_request_resolution_duration
0
500
1000
1500
2000
2500
SHAP value (impact on model output)
High
Low
Feature value

- **LightGBM**

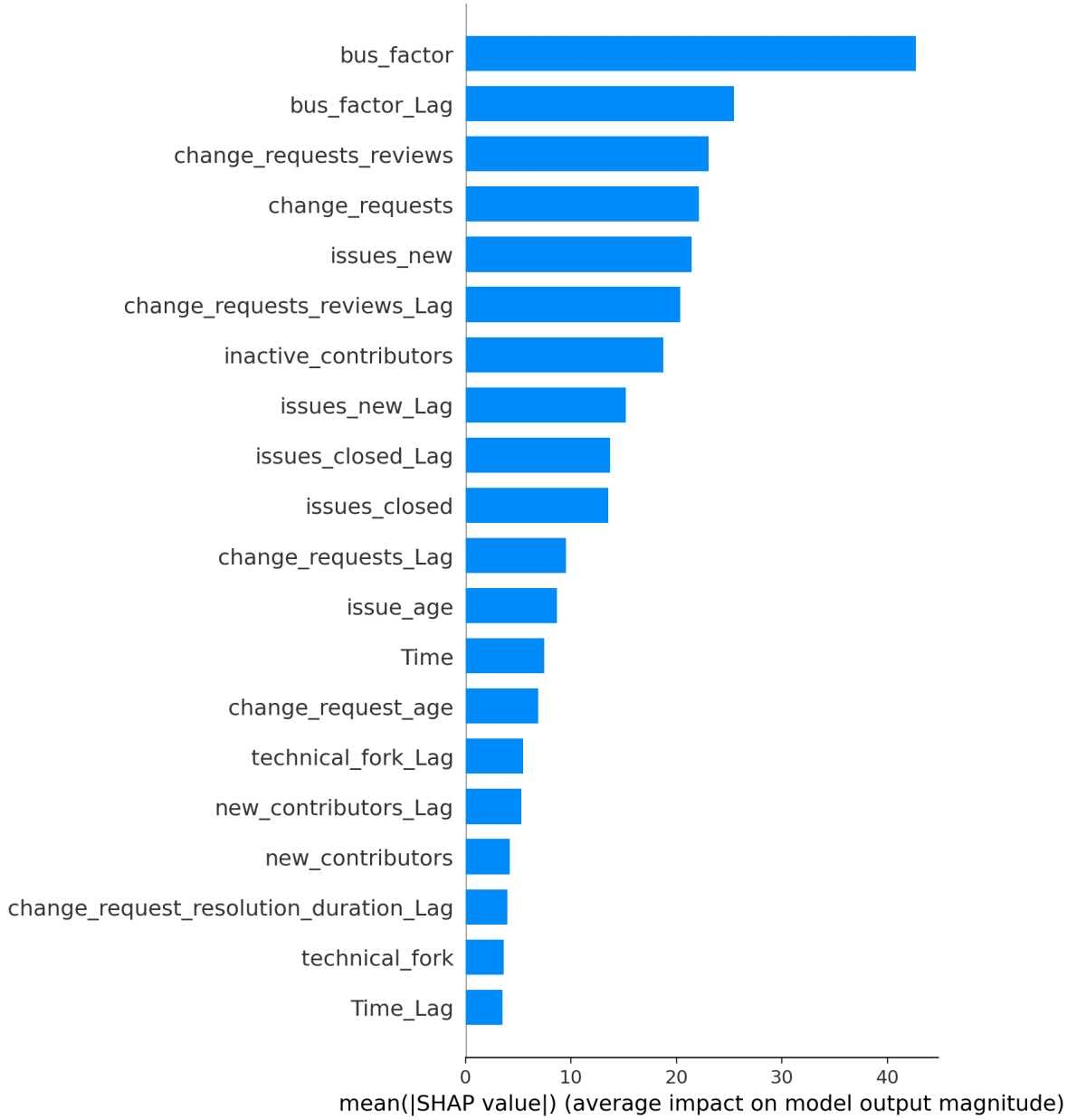


柱状图版本：

- **XGBoost**


bus_factor
change_requests
issues_new
bus_factor_Lag
inactive_contributors
change_requests_reviews
change_requests_reviews_Lag
issues_new_Lag
issues_closed
issues_closed_Lag
change_requests_Lag
issue_age
technical_fork_Lag
Time
change_request_age
technical_fork
new_contributors_Lag
change_request_resolution_duration_Lag
new_contributors
change_request_resolution_duration
0
10
20
30
40
mean(|SHAP value|) (average impact on model output magnitude)

- **LightGBM**



对比两张图可以发现，bus_factor整体的SHAP value最高，这与相关性分析和XGBoost、LightGBM自带的特征重要性分析结果相同，其次XGBoost是change_requests和issues_new，LightGBM是bus_factor_Lag和change_requests_reviews，可以发现的是这个结果与特征重要性有相似的部分，但同时也有不同点，原因是SHAP value评估的是特征的边际贡献，与重要性较为相似但不完全是重要性。

2.2.6 降维（二）

先前提到了使用不受多重共线性影响的机器学习模型进行建模，这里根据模型可解释性进行真正意义上的降维。

综合考虑特征重要性与SHAP value，去除一些重要性与SHAP value较小的冗余特征，对模型性能几乎没有影响，并且降低了模型复杂度，显著提升了运行速度

实验过程如下：

XGBoost

① 第一次剔除如下指标

```
"Time_Lag",
"inactive_contributors_Lag",
"new_contributors_Lag",
"change_requests_accepted_Lag",
"issue_age_Lag",
"new_contributors",
"change_requests_accepted",
"code_change_lines_add_Lag",
"code_change_lines_remove",
"code_change_lines_remove_Lag",
"code_change_lines_sum_Lag",
"code_change_lines_sum",
"change_request_resolution_duration_Lag"
```

② 第二次剔除如下指标

```
"Time_Lag",
"inactive_contributors_Lag",
"new_contributors_Lag",
"change_requests_accepted_Lag",
"issue_age_Lag",
"new_contributors",
"change_requests_accepted",
"code_change_lines_add_Lag",
"code_change_lines_remove",
"code_change_lines_remove_Lag",
"code_change_lines_sum_Lag",
"code_change_lines_sum",
"change_request_resolution_duration_Lag",
```

```
"technical_fork"
```

LightGBM

① 第一次剔除如下指标

```
"Time_Lag",
"inactive_contributors_Lag",
"new_contributors_Lag",
"change_requests_accepted_Lag",
"issue_age_Lag"
```

② 第二次剔除如下指标

```
"Time_Lag",
"inactive_contributors_Lag",
"new_contributors_Lag",
"change_requests_accepted_Lag",
"issue_age_Lag",
"new_contributors",
"change_requests_accepted",
"code_change_lines_add_Lag",
"code_change_lines_remove"
```

结果在第三节展示

2.2.7 根据项目状态与阶段分类拟合

根据项目状态与阶段不同，可能拟合算法不同，即项目阶段不同，统计型指标的重要性会发生变化。这里的阶段可以是以 OpenRank 阈值为标准，或以其他合理的指标阈值为标准。

这里选择根据OpenRank值，进行项目分段：

```
count    21572.000000
mean       251.546959
std        258.252390
min          0.000000
25%        106.840000
50%        194.940000
75%        301.190000
max       4829.380000
Name: openrank, dtype: float64
```

① 首先考虑将项目分为两阶段，分界阈值为OpenRank 50%分位数

② 也可以考虑将项目分为四阶段，即分别用25%，50%，75%分位数作为分界阈值

分段后，将分段的数据单独取出进行训练与验证（数据量减小），结果与分析见第三节：

三、实验结果与分析

3.1 实验环境与设置

实验环境

```
Windows 10

Python 3.7.11

Anaconda 4.10.3

Jupyter Notebook
```

需要额外安装的package

```
numpy==1.20.3
pandas==1.3.4
matplotlib==3.5.3
tqdm==4.65.0
scikit-learn==1.0.1
seaborn==0.11.2
xgboost==1.6.2
lightgbm==3.2.1
```

3.2 实验结果

3.2.1 不同训练轮次下模型结果

Method	Epoch	Time	RMSE	MAE	explained_variance_score	R2_score
XGBoost	100	13.5s	103.9063	59.4101	0.8541	0.8533
XGBoost	200	25.8s	102.9162	58.4126	0.8568	0.8561
XGBoost	500	1m10.4s	101.8802	56.6220	0.8599	0.8590
XGBoost	1000	2m12.5s	101.4421	56.1374	0.8611	0.8602
LightGBM	100	2.5s	112.3649	65.9922	0.8353	0.8285
LightGBM	200	4.4s	112.3095	66.4775	0.8390	0.8287
LightGBM	500	6.5s	110.8115	65.1004	0.8431	0.8332
LightGBM	1000	12.4s	110.4581	64.4310	0.8441	0.8343

Table 1: Experiment1 Results

注：所有数据均为调参后结果

3.2.2 降维前后模型结果

Method	Time	RMSE	MAE	explained_variance_score	R2_score
XGBoost	2m21s	101.4421	56.1374	0.8611	0.8602
XGBoost(reduce1)	1m42.1s	100.8122	56.7699	0.8623	0.8619
XGBoost(reduce2)	1m40.3s	103.0021	56.9462	0.8562	0.8559
LightGBM	12.4s	110.4581	64.4310	0.8441	0.8343
LightGBM(reduce1)	9.7s	105.2504	60.4104	0.8495	0.8495
LightGBM(reduce2)	9.0s	110.9163	61.5408	0.8329	0.8328

Table 2: Experiment2 Results

注：reduce1,reduce2分别代表2.2.6中的两次降维

3.2.3 分阶段模型结果

二阶段

Method	Time	RMSE	MAE	explained_variance_score	R2_score
XGBoost(stage1)	51.8s	24.8538	18.5117	0.5980	0.5933
XGBoost(stage2)	1m16.3s	103.2180	62.2905	0.8781	0.8780
LightGBM(stage1)	9.1s	24.6575	18.6517	0.6011	0.5997
LightGBM(stage2)	8.5s	106.2637	64.6637	0.8708	0.8707

Table 3: Experiment3 Results

四阶段

Method	Time	RMSE	MAE	explained_variance_score	R2_score
XGBoost(stage1)	16.5s	37.6717	30.8186	0.4190	0.0046
XGBoost(stage2)	23.0s	19.3378	15.7841	0.2474	0.2395
XGBoost(stage3)	25.7s	26.5872	21.4081	0.2916	0.2875
XGBoost(stage4)	11.4s	139.2765	89.2171	0.8510	0.8510
LightGBM(stage1)	6.4s	35.3139	29.9013	0.4012	0.1253
LightGBM(stage2)	7.7s	20.6992	17.0574	0.1532	0.1286
LightGBM(stage3)	7.4s	27.1576	21.4908	0.2566	0.2566
LightGBM(stage4)	6.1s	143.4238	91.0808	0.8426	0.8420

Table 4: Experiment4 Results

3.3 结果分析与讨论

3.3.1 不同训练轮次下模型结果分析

① XGBoost在训练轮次不断增加的过程中，性能逐渐有略微的提升，1000轮次的结果最佳，explained_variance_score达到0.8611，R2_score达到0.8602，在验证集上总体拟合程度较好。

② XGBoost总体在性能上略优于LightGBM，然而LightGBM在运行速度上远快于XGBoost。在性能损失可接受范围内，特别是较大的数据量，则选择速度更快的LightGBM在实际应用中较为合理；如果追求更好的性能，那么选择XGBoost更为合理。

③ 模型均具备可解释性，在2.2.5中已进行展示。

3.3.2 降维前后模型结果分析

① 降维后两个模型的性能均先上升后下降，在explained_variance_score指标与R2_score指标上，两个模型随着降维中剔除特征的数量增强均先增加后下降，这是由于剔除冗余特征减少冗余信息，更有利于模型的拟合；并且减少特征维度可以减少过拟合风险，提高模型泛化能力，故而模型性能提升；而进一步删除特征可能删除了包含有用信息的重要特征导致模型性能下降。

② 降维后两个模型的运行时间都有下降，XGBoost有显著下降，LightGBM略有下降，可见降维对降低模型复杂度，减少运行时间的作用。如果数据量进一步加大，那么这样的效果会更加明显。

3.3.3 分阶段模型结果分析

结果发现两阶段，处于前半段的数据，四阶段处于前3个阶段的数据，即OpenRank值较小的数据，用原先的机器学习算法拟合效果并不是特别好；而两阶段处于后半段的数据，四阶段处于最后一个阶段的数据，即OpenRank值较大的数据，用原先的机器学习算法拟合效果较好。

本次实验暂时并未分析出这样结果出现的原因，后续进一步的原因分析和解决方案会在OpenSODA比赛后续阶段进行探究。

四、结论

本次实验对时序数据进行了预处理、特征工程，其中特别是添加了滞后特征，使用机器学习算法对数据进行了训练以及后续的拟合，结果尚可：

最佳结果为使用XGBoost运行1000轮的结果，而LightGBM相比XGBoost结果略有下降，但运行时间显著降低。

接着使用特征重要性以及SHAP value从而探究了模型的可解释性，发现bus_factor，Time等特征对于模型的拟合结果更为重要，Time_Lag，new_contributors等特征对模型的拟合结果不是特别重要，并根据上述可解释性进一步降维，降低了模型复杂度，进一步提升了性能，并提升了运行速度；

根据项目状态与阶段不同，使用不同拟合算法部分，仅依据结果发现了OpenRank值较小的项目数据使用XGBoost等机器学习算法进行拟合效果较差，而OpenRank值较大的项目数据使用XGBoost等机器学习算法进行拟合效果较好，但尚未得到有效解决方案，这一部分也是后续OpenSODA比赛的重心。

五、参考文献

[1] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

[2] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, *30*.

[3] Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181-1191.

[4] Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

[5] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021, May). Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 35, No. 12, pp. 11106-11115).

六、附录：源代码（带注释）

注：代码为项目中的main.ipynb文件

- 引入相关包

```
# 引入相关包
import numpy as np
import pandas as pd
import os
import json
from tqdm import tqdm
from sklearn.decomposition import PCA
from sklearn import preprocessing
import seaborn as sns
import matplotlib.pyplot as plt
import xgboost as xgb
import lightgbm as lgb
from sklearn.metrics import
mean_squared_error,explained_variance_score,r2_score,mean_
absolute_error
np.set_printoptions(suppress=True)
```

数据预处理

- 定义指标分类

```
metrics_unprocessed=["active_dates_and_times",
                     "issue_response_time",
                     "issue_resolution_duration",
                     "issue_age",
                     "change_request_response_time",
                     "change_request_resolution_duration",
                     "change_request_age"
                     ]

metrics_processed=["openrank",
                   "technical_fork",
```

```
                    "new_contributors",
                    "inactive_contributors",
                    "bus_factor",
                    "issues_new",
                    "issues_closed",
                    "code_change_lines_add",
                    "code_change_lines_remove",
                    "code_change_lines_sum",
                    "change_requests",
                    "change_requests_accepted",
                    "change_requests_reviews"
                ]
```

- 一些数据预处理函数

```
# 输入数据路径
data_path="./data/top_300_metrics/"

# 根据指标分类获取指标数据
def get_value(file,metric):
    with open(file) as f:
        data = json.load(f)

        # 如果指标是有统计信息的,取均值
        if metric in metrics_unprocessed:
            data=data['avg']
        if "2021-10-raw" in data:
            data.pop("2021-10-raw")
    f.close()
    return data

# 对时间进行编码
def time_encoding(df):
    df["time"] = pd.to_datetime(df["time"], format="%Y-
%m")
    df["Time"] = (df["time"].dt.year -
df["time"].dt.year.min()) * 12 + df["time"].dt.month
    return df
```

```
# 判断是否是CHAOSS指标
def is_CHAOSS_metric(metric):
    return (metric in metrics_unprocessed) or (metric in
metrics_processed)

# 初始化数据表
def init_metric_table(project_path):
    time=[]
    active_dates_and_times=[]

    # 先选择active_dates_and_times这一个指标初始化数据表,之后再将
其他指标数据加入表中
    with
open(os.path.join(project_path,metrics_unprocessed[0]+".js
on")) as f:
        data=json.load(f)
        for key,value in data.items():
            time.append(key)

 active_dates_and_times.append(np.average(np.array(value))
)

 metric_table=pd.DataFrame({"time":time,"active_dates_and_
times":active_dates_and_times})
    return metric_table

# 获取整个数据表
def get_project_metric_table(project_path):
    metrics_table=init_metric_table(project_path)
    for file in os.listdir(project_path):
        file_path=os.path.join(project_path,file)
        metric=file[:-5]
        if metric==metrics_unprocessed[0] or not
is_CHAOSS_metric(metric):
            continue

        # 获取指标数据
        data=get_value(file_path,metric)
```

```
        # 按照时间顺序排列,并填充缺失值

 metrics_table[metric]=metrics_table["time"].map(data).fil
lna(0)
    return metrics_table
```

- 读取数据并进行数据预处理

```
metric_table=[]     # 数据表

# 对每个组织
for organization in tqdm(os.listdir(data_path)):
    org_path=os.path.join(data_path,organization)

    # 对组织的每个项目
    for project in os.listdir(org_path):
        project_path=os.path.join(org_path,project)

 proj_metric_table=get_project_metric_table(project_path)
        metric_table.append(proj_metric_table)
        # corr_proj=proj_metric_table.corr()
```

- 进一步处理并划分训练集验证集

```
# 合并所有项目的数据
data=pd.concat(metric_table).fillna(0)

# 重新设置索引
data = data.reset_index(drop=True)

# 将时间编码
data=time_encoding(data)
Data=data.drop(columns=["time"])

# 划分训练集验证集
test_data=data[data['time']>="2023-01"]
train_data=data[data['time']<"2023-01"]
```

```
X_train=train_data.drop(columns=["openrank","time"])
y_train=train_data["openrank"]
X_test=test_data.drop(columns=["openrank","time"])
y_test=test_data["openrank"]

print("Data Shape:{}, Train data shape:{}, Test data
Shape:{}".format(Data.shape,X_train.shape,X_test.shape))
```

特征工程

- 查看数据所有特征

```
Data.columns
```

- 查看时间编码

```
Data['Time']
```

- 定义一些绘图函数

```
# 分布图
def displot(x):
    fig = plt.figure(dpi=150)
    sns.distplot(x)

# 另一种分布图
def hisplot(x):
    fig = plt.figure(dpi=150)
    sns.histplot(x)

# 相关性热力图
def heatmap(X):
    f,ax = plt.subplots(figsize=(15, 15))
```

```
    sns.heatmap(X.corr(), annot=True,
linecolor='white',linewidths=0.1,cmap="RdBu", fmt=
'.1f',ax=ax)

# 直方分布图
def hist(Data):
    Data.hist(bins=30, figsize=(20,15),color='#A50021')

# 双变量的相关性图
def jointplot(x,y,data):
    fig = plt.figure(dpi=150)
    sns.jointplot(x=x, y=y, data=data,
kind="hex",color="#A50021",ratio=8, space=0, height=8,
marginal_kws={'bins':10,'kde':True})
    plt.xlabel(x, fontsize=15)
    plt.ylabel(y, fontsize=15)
    plt.show()
```

- 待拟合变量OpenRank值的分布情况

```
Data['openrank'].describe()
```

- 两种分布图

```
displot(Data['openrank'])
```

```
hisplot(Data['openrank'])
```

- 所有特征的分布直方图

```
hist(Data)
```

- 相关性热力图

```
heatmap(Data)
```

- openrank和其他变量的相关性表（降序）

```
Data.corr()["openrank"].sort_values(ascending=False)
```

- 输入变量与目标变量之间的关系

① 时序数据和OpenRank值的关系

```
jointplot("Time","openrank",Data.iloc[::10])
```

② issue从开始到首次响应的时长issue_response_time和OpenRank值的关系

```
jointplot("issue_response_time","openrank",Data[::20])
```

- 多重共线性判断

```
# 计算X'X特征值
eigenvalues=np.linalg.eigvals(Data.T @ Data)

# 计算条件数
Condition_Number =
np.sqrt(np.abs(np.max(eigenvalues)/np.min(eigenvalues)))

print("eigenvalues:{}".format(eigenvalues))
print("Condition_Number:{}".format(Condition_Number))
```

- 滞后特征

```
# 添加一阶滞后特征
def add_lag_feature(X,lag):
    lag=1
    for column in X.columns:
        X[column + "_Lag"] =
X[column].shift(lag).fillna(0)
    return X

X_train=add_lag_feature(X_train,lag=1)
X_test=add_lag_feature(X_test,lag=1)
```

- 查看issues_new和其滞后特征issues_new_Lag

```
X_train[["issues_new", "issues_new_Lag"]]
```

模型建立与训练

- 定义XGBoost模型与训练验证过程

```
# 定义XGBoost回归模型
def train_xgboost(X_train,X_test,y_train,y_test,epoch):

    # 模型建立
    model = xgb.XGBRegressor()

    # 参数设置
    params = {'colsample_bytree': 1,
              'colsample_bylevel': 1,
              'learning_rate': 0.06,
              'max_depth': 9,
              'alpha': 10,
              'subsample':1,
              'min_child_weight':4,
              'gamma':0.2,
              'reg_alpha':0.1,
              'reg_lambda':0.3,
```

```
                'scale_pos_weight':1}

    # 交叉验证
    dtrain = xgb.DMatrix(X_train, label=y_train)
    cv_results = xgb.cv(params, dtrain,
num_boost_round=epoch, nfold=5, metrics='rmse',
early_stopping_rounds=10)

    # 获取最佳迭代轮数,训练模型
    best_num_boost_rounds = cv_results.shape[0]  # 最佳的迭
代轮数
    best_model = xgb.train(params, dtrain,
num_boost_round=best_num_boost_rounds)

    # 将测试集数据转换为DMatrix格式
    dtest = xgb.DMatrix(X_test)

    # 使用最佳模型进行预测
    y_pred = best_model.predict(dtest)

    # 评估
    mse = mean_squared_error(y_test, y_pred)
    rmse=np.sqrt(mse)
    score=explained_variance_score(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    # 打印评估结果
    print("RMSE:{}".format(rmse))
    print("MAE:{}".format(mae))
    print("explained_variance_score:{}".format(score))
    print("R2 score:{}".format(r2))

    return best_model,cv_results
```

- 定义LightGBM模型与训练验证过程

```
```

```
def train_lightgbm(X_train, X_test, y_train,
y_test,epoch):
    # 定义LightGBM回归模型
    model = lgb.LGBMRegressor(objective='regression')

    # 参数设置
    params = {'colsample_bytree': 1,
            'learning_rate': 0.06,
            'max_depth': 9,
            'alpha': 10,
            'subsample':1,
            'min_child_weight':4,
            'reg_alpha':0.1,
            'reg_lambda':0.3,
            'scale_pos_weight':1,
            "verbose": -1
            }

    # 创建LightGBM的数据对象
    dtrain = lgb.Dataset(X_train, label=y_train)

    # 交叉验证
    cv_results = lgb.cv(params, dtrain,
num_boost_round=epoch, nfold=5, metrics='rmse',
stratified=False)

    # 使用最佳迭代轮数训练模型
    best_num_boost_rounds = len(cv_results['valid rmse-
mean'])
    best_model = lgb.train(params, dtrain,
num_boost_round=best_num_boost_rounds)

    # 使用最佳模型进行预测
    y_pred = best_model.predict(X_test)

    # 评估
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    score = explained_variance_score(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
```

```
    mae = mean_absolute_error(y_test, y_pred)

    # 打印评估结果
    print("RMSE: {}".format(rmse))
    print("MAE: {}".format(mae))
    print("explained_variance_score: {}".format(score))
    print("R2 score: {}".format(r2))

    return best_model,cv_results
```

- 进行训练与测试

注：轮次可进行调整，本次实验选择epoch=100,200,500,1000分别进行训练

XGBoost

```
best_model_xgb,cv_results=train_xgboost(X_train,X_test,y_train,y_test,epoch=1000)
```

查看交叉验证过程

```
print(cv_results)
```

LightGBM

```
best_model_lgb,cv_results=train_lightgbm(X_train,X_test,y_train,y_test,epoch=1000)
```

查看交叉验证过程

```
cv_results
```

模型可解释性

- 绘制特征重要性图

XGBoost

```
# 获取特征重要性
plt.figure(dpi=200)
plt.rcParams['figure.dpi'] = 200
fig, ax = plt.subplots(figsize=(10, 8))

# 绘制特征重要性
xgb.plot_importance(best_model_xgb, ax=ax)
plt.show()
```

LightGBM

```
# 获取特征重要性
plt.figure(dpi=200)
plt.rcParams['figure.dpi'] = 200
fig, ax = plt.subplots(figsize=(10, 8))

# 绘制特征重要性
lgb.plot_importance(best_model_lgb, ax=ax)
plt.show()
```

- 绘制平均后的特征重要性

```
# 降序查看特征重要性
xgb_feature_importance=best_model_xgb.get_score(importance
_type='weight')
sorted_importance = {feature: importance for feature,
importance in sorted(xgb_feature_importance.items(),
key=lambda x: x[1], reverse=True)}
for feature, importance in sorted_importance.items():
    print(f'{feature}: {importance}')
```

```
# 降序查看特征重要性
lgb_feature_importance=best_model_lgb.feature_importance()
lgb_feature_name = best_model_lgb.feature_name()
sorted_importance = sorted(zip(lgb_feature_name,
lgb_feature_importance), key=lambda x: x[1], reverse=True)

for name, importance in sorted_importance:
    print(f'{name}: {importance}')
```

```
# 合并特征重要性并取平均
combined_importance = {}
for feature, importance in
dict(sorted_importance).items():
    combined_importance[feature] = (importance +
xgb_feature_importance.get(feature, 0)) / 2

# 按照特征重要性降序排序
sorted_importance_combined =
sorted(combined_importance.items(), key=lambda x: x[1],
reverse=True)

# 查看特征重要性
for feature, importance in sorted_importance_combined:
    print(f'{feature}: {importance}')
```

```
# 提取特征名称和对应的重要性值
feature_names = [feature for feature, _ in
sorted_importance_combined]
importance_values = [importance for _, importance in
sorted_importance_combined]

# 绘制横向柱状图
plt.figure(figsize=(10, 10))
bars=plt.barh(feature_names, importance_values)
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Feature Importance')
plt.gca().invert_yaxis()
```

```
# 添加数值标注
for i, bar in enumerate(bars):
    plt.text(bar.get_width() + 0.1, bar.get_y() +
bar.get_height() / 2, f'{importance_values[i]:.2f}',
ha='left', va='center')
plt.tight_layout()
plt.show()
```

- SHAP机器学习可解释性

XGBoost

```
# 导入并初始化shap
import shap
shap.initjs()

# shap解释器
explainer_xgb = shap.TreeExplainer(best_model_xgb)

# shap value
shap_values_xgb = explainer_xgb(X_train)
```

绘制两种summary_plot

```
fig, ax = plt.subplots(figsize=(20, 15))
shap.summary_plot(shap_values_xgb, X_train,
plot_type="bar")
```

```
fig, ax = plt.subplots(figsize=(20, 15))
shap.summary_plot(shap_values_xgb, X_train)
```

绘制单个样本每个特征的SHAP value贡献度（force_plot和waterfall）

```
shap.force_plot(explainer_xgb.expected_value,
shap_values_xgb.values[20000])
```

```
shap.plots.waterfall(shap_values_xgb[20000])
```

LightGBM

```
# 导入并初始化shap
import shap
shap.initjs()

# 设置模型的目标
best_model_lgb.params['objective'] ='reg:linear'

# shap解释器
explainer_lgb = shap.TreeExplainer(best_model_lgb)

# shap value
shap_values_lgb = explainer_lgb(X_train)
```

绘制两种summary_plot

```
fig, ax = plt.subplots(dpi=200)
shap.summary_plot(shap_values_lgb, X_train,
plot_type="bar")
```

```
fig, ax = plt.subplots(dpi=200)
shap.summary_plot(shap_values_lgb, X_train)
```

绘制单个样本每个特征的SHAP value贡献度（force_plot和waterfall）

```
shap.force_plot(explainer_lgb.expected_value,
shap_values_lgb.values[20000])
```

```
fig, ax = plt.subplots(dpi=200)
shap.plots.waterfall(shap_values_lgb[20000])
```

降维

- 降维：特征选择

XGBoost

两次剔除特征

```
# 要剔除的特征
remove_columns=["Time_Lag",
                "inactive_contributors_Lag",
                "new_contributors_Lag",
                "change_requests_accepted_Lag",
                "issue_age_Lag",
                "new_contributors",
                "change_requests_accepted",
                "code_change_lines_add_Lag",
                "code_change_lines_remove",
                "code_change_lines_remove_Lag",
                "code_change_lines_sum_Lag",
                "code_change_lines_sum",
                "change_request_resolution_duration_Lag"
                ]

X_train_reduce=X_train.drop(columns=remove_columns)
X_test_reduce=X_test.drop(columns=remove_columns)

best_model=train_xgboost(X_train_reduce,X_test_reduce,y_train,y_test,epoch=1000)
```

```
# 要剔除的特征
remove_columns=["Time_Lag",
                "inactive_contributors_Lag",
                "new_contributors_Lag",
                "change_requests_accepted_Lag",
                "issue_age_Lag",
                "new_contributors",
                "change_requests_accepted",
                "code_change_lines_add_Lag",
                "code_change_lines_remove",
                "code_change_lines_remove_Lag",
                "code_change_lines_sum_Lag",
                "code_change_lines_sum",
                "change_request_resolution_duration_Lag",
                "technical_fork"
                ]
```

```
X_train_reduce=X_train.drop(columns=remove_columns)
X_test_reduce=X_test.drop(columns=remove_columns)

best_model=train_xgboost(X_train_reduce,X_test_reduce,y_train,y_test,epoch=1000)
```

LightGBM

两次剔除特征

```
# 要剔除的特征
remove_columns=["Time_Lag",
                "inactive_contributors_Lag",
                "new_contributors_Lag",
                "change_requests_accepted_Lag",
                "issue_age_Lag"
                ]

X_train_reduce=X_train.drop(columns=remove_columns)
X_test_reduce=X_test.drop(columns=remove_columns)

best_model=train_lightgbm(X_train_reduce,X_test_reduce,y_train,y_test,epoch=1000)
```

```
# 要剔除的特征
remove_columns=["Time_Lag",
                "inactive_contributors_Lag",
                "new_contributors_Lag",
                "change_requests_accepted_Lag",
                "issue_age_Lag",
                "new_contributors",
                "change_requests_accepted",
                "code_change_lines_add_Lag",
                "code_change_lines_remove"
                ]

X_train_reduce=X_train.drop(columns=remove_columns)
X_test_reduce=X_test.drop(columns=remove_columns)
```

```
best_model=train_lightgbm(X_train_reduce,X_test_reduce,y_train,y_test,epoch=1000)
```

根据项目状态与阶段分类拟合

- 根据openrank阈值确定项目阶段

```
# 分位数
quantiles = y_train.describe().loc[['25%', '50%', '75%']]

# 四阶段划分
def Stage4(openrank):
    q25,q50,q75=quantiles
    if openrank<=q25:
        stage=1
    elif openrank<=q50:
        stage=2
    elif openrank<=q75:
        stage=3
    else:
        stage=4
    return stage

# 两阶段划分
def Stage2(openrank):
    q25,q50,q75=quantiles
    if openrank<=q50:
        stage=1
    else:
        stage=2

    return stage

# 根据划分函数与阶段数划分数据
def get_train_test_data(data,Stage,stage):
    lag=1

    # 获取划分函数和阶段数对应的数据
```

```
    data_stage=data[data["openrank"].apply(Stage) ==
stage]

    # 划分训练集与验证集
    test_data_stage=data_stage[data_stage['time']>="2023-
01"]
    train_data_stage=data_stage[data_stage['time']<"2023-
01"]
    X_train_stage=train_data_stage.drop(columns=
["openrank","time"]).fillna(0)
    y_train_stage=train_data_stage["openrank"]
    X_test_stage=test_data_stage.drop(columns=
["openrank","time"]).fillna(0)
    y_test_stage=test_data_stage["openrank"]

    # 滞后特征
    for column in X_train_stage.columns:
        X_train_stage[column + "_Lag"] =
X_train_stage[column].shift(lag).fillna(0)
        X_test_stage[column + "_Lag"] =
X_test_stage[column].shift(lag).fillna(0)

    # 降维

 X_train_stage=X_train_stage.drop(columns=remove_columns)
    X_test_stage=X_test_stage.drop(columns=remove_columns)
    return
X_train_stage,X_test_stage,y_train_stage,y_test_stage
```

XGBoost

两阶段数据

```
X_train1,X_test1,y_train1,y_test1=get_train_test_data(data
,Stage2,stage=1)
X_train2,X_test2,y_train2,y_test2=get_train_test_data(data
,Stage2,stage=2)
```

两阶段模型训练验证

```
best_model1,cv_results=train_xgboost(X_train1,X_test1,y_train1,y_test1,epoch=1000)
```

```
best_model2,cv_results=train_xgboost(X_train2,X_test2,y_train2,y_test2,epoch=1000)
```

四阶段数据

```
X_train1,X_test1,y_train1,y_test1=get_train_test_data(data,Stage4,stage=1)
X_train2,X_test2,y_train2,y_test2=get_train_test_data(data,Stage4,stage=2)
X_train3,X_test3,y_train3,y_test3=get_train_test_data(data,Stage4,stage=3)
X_train4,X_test4,y_train4,y_test4=get_train_test_data(data,Stage4,stage=4)
```

四阶段模型训练验证

```
best_model1,cv_results=train_xgboost(X_train1,X_test1,y_train1,y_test1,epoch=1000)
```

```
best_model2,cv_results=train_xgboost(X_train2,X_test2,y_train2,y_test2,epoch=1000)
```

```
best_model3,cv_results=train_xgboost(X_train3,X_test3,y_train3,y_test3,epoch=1000)
```

```
best_model4,cv_results=train_xgboost(X_train4,X_test4,y_train4,y_test4,epoch=1000)
```

LightGBM

两阶段数据

```
X_train1,X_test1,y_train1,y_test1=get_train_test_data(data
,Stage2,stage=1)
X_train2,X_test2,y_train2,y_test2=get_train_test_data(data
,Stage2,stage=2)
```

两阶段模型训练验证

```
best_model1,cv_results=train_lightgbm(X_train1,X_test1,y_t
rain1,y_test1,epoch=1000)
```

```
best_model2,cv_results=train_lightgbm(X_train2,X_test2,y_t
rain2,y_test2,epoch=1000)
```

四阶段数据

```
X_train1,X_test1,y_train1,y_test1=get_train_test_data(data
,Stage4,stage=1)
X_train2,X_test2,y_train2,y_test2=get_train_test_data(data
,Stage4,stage=2)
X_train3,X_test3,y_train3,y_test3=get_train_test_data(data
,Stage4,stage=3)
X_train4,X_test4,y_train4,y_test4=get_train_test_data(data
,Stage4,stage=4)
```

四阶段模型训练验证

```
best_model1,cv_results=train_lightgbm(X_train1,X_test1,y_t
rain1,y_test1,epoch=1000)
```

```
best_model2,cv_results=train_lightgbm(X_train2,X_test2,y_t
rain2,y_test2,epoch=1000)
```

```
best_model3,cv_results=train_lightgbm(X_train3,X_test3,y_t
rain3,y_test3,epoch=1000)
```

```
best_model4,cv_results=train_lightgbm(X_train4,X_test4,y_t
rain4,y_test4,epoch=1000)
```