一、 问题重述

1.1 引言

社交媒体的不断普及，人们获取信息的方式日益丰富。在视频平台上，视频博主通过发表视频吸引用户关注，而用户的各种互动行为，将会作为内容推广的部分参考指标。博主的内容受用户的反馈影响，平台推送机制根据用户互动操作运作，从而用户与博主之间就形成了一个反馈循环。为了更好地发现互动行为与博主之间的关系，人们希望通过对用户互动数据的深度分析来改进推送机制，并且基于用户历史行为预测用户未来的操作。

1.2 问题提出

问题 1 通过对用户与博主的历史互动数据进行分析，可以有效揭示用户行为特征，这对优化平台推送机制以及提升用户交互有着重要依据。基于附件中提供的用户数据，建立数学模型来预测 2024 年 7 月 21 日博主新增的关注数，并且列出新增关注最多的五位博主以及关注数。

问题 2 通过附件 1 的用户历史互动数据以及附件 2 的 2024 年 7 月 22 日互动数据，预测用户在 7 月 22 日可能关注的博主，并且预测哪些博主会被用户关注。

问题 3 视频平台的推送机制依赖于用户与博主之间的互动数据（如点赞，评论，关注），建立数学模型，预测 2024 年 7 月 21 日用户是否在线，假若用户在线，预测该用户会与哪些博主发生互动，给出互动数最高的三位博主。

问题 4 平台需要考虑不同用户的时间习惯来实现更精确的推送。在问题 3 的基础上，预测用户在 2024 年 7 月 23 日的在线时段，并根据每个时段的互动数，给出互动最多的三位博主和时段。

二、 问题分析

在问题分析部分，对每个问题进行解析与模型选择的分析。问题 1 要基于用户与博主的历史互动数据，预测博主的新增关注数，在此采用随机森林模型学习用户基于日期的特征数据，通过学习用户互动与关注之间的联系进行预测。问题 2 则要预测用户在指定日期的关注行为，通过提取用户互动行为特征，使用 LightGBM 模型对用户行为进行学习，继而预测关注。问题 3 要预测用户是否在线以及可能产生互动的博主，首先通过 LightGBM 模型学习用户活跃度预测用户在线概率，再通过 LightGBM 模型对用户的短期互动行为-博主等特征对进行学习，预测用户与博主的互动数。问题 4 则在问题 3 的基础上，进一步延伸到预测用户在指定日期的在线情况和互动时段

2.1 问题 1 分析：预测博主新增关注数

问题 1 要求通过历史互动数据来预测 2024 年 7 月 21 日博主的新增关注数。题目所给出的数据包括了用户与博主之间的所有互动行为，例如观看、点赞、关注、评论。这些行为在一定时间内可以通过时间窗口来提取短期内的用户特征数据进行预测。为了解决这个问题，使用随机森林模型进行预测。随机森林具有通过特征洞察数据关系的能力，所以他能够有效地预测博主在某日的新增关注数。随机森林对原有数据进行有放回取样生成训练集，使用训练集构建决策树来达到学习的效果，并根据最后每棵决策树的平均值（投票或打分）来预测新增关注数。

2.2 问题 2 分析：预测用户新关注博主

问题 2 要求根据历史互动数据，预测 2024 年 7 月 22 日的用户关注行为，可以视为一个二分类问题。对于该题可以使用 LightGBM 模型解决，LightGBM 的基本单元也是决策树，但是其基于梯度上升的思想设计使其训练速度具有优势，并且能够处理高维数据。通过对历史互动数据进行数据分析，我们可以提取出大量与用户、博主、用户-博主对有关的特征，其中用户-博主对特征里有大量反映了用户对某博主的感兴趣程度的数据，通过 LightGBM 模型学习特征数据，可以在模型内隐性的产生用户与博主的关联度，通过对正、负样本的学习，最终给出每个博主的关注概率。根据关注概率，若关注概率超过阈值，则会视为用户将关注该博主。

2.3 问题 3 分析：预测用户在线情况以及互动博主

问题 3 要求预测用户在 2024 年 7 月 21 日是否在线，并预测在线时用户可能与哪些博主互动，可以视为一个二分类问题。在解决该问题时，可以将问题分成两个分任务，任务 1 是预测用户在线情况，任务 2 是预测用户会与哪些博主互动。其中任务 1 是一个二分类问题，通过用户历史数据来预测用户是否在线，同样可以使用 LightGBM 模型来解决。通过设计用户短期内活跃数据（活跃天数、距上次活动天数等）特征，构建训练和验证集进行训练，最终对目标用户做出预测。对于任务 2，用户与博主的互动次数预测是一个回归问题，但是 LightGBM 在不同参数下可以实现回归和分类，因此我们可以使用 LightGBM 回归器来预测用户与博主的互动次数。

2.4 问题 4 分析：预测用户在线情况及互动时段

问题 4 要求基于用户历史互动数据，预测用户在 2024 年 7 月 23 日的在线情况以及在线时可能与哪些博主互动。在解决该问题时，同样可以将问题分成两个分任务，任务 1 是预测用户在线情况，任务 2 是预测用户会与哪些博主互动。其中任务 1 是一个二分类问题，通过用户历史数据来预测用户是否在线，可以使用问题三已经训练好的 LightGBM 模型来解决。任务 2 是一个预测问题，通过用户历史数据来预测用户可能与哪些博主互动，但是介于本题时间颗粒度较小，LightGBM 难以高效的学习时间序列特征，因此基于历史互动偏好来统计预测用户可能的互动。

三、 模型假设与符号说明

1. 平台用户和博主数量固定，不存在平台新用户/博主的加入和账号注销行为。
2. 用户和博主的互动关系建立后不再变化，即平台中用户不存在取消点赞、删除评论、取消关注的行为。
3. 历史互动数据完整且准确，可以真实反映出用户和博主之间的联系。
4. 假设用户未来行为可以基于历史数据有效预测，过去的互动可以反映用户对博主的兴趣程度，不存在较多不符合趋势的行为模式

表 1　符号说明

符号	说明
U	用户集合
B	博主集合
$T_{threshold}$	预测概率阈值
X	特征向量或特征矩阵
Y	标签或目标变量
P	概率或预测值
N	数量或计数
f	模型函数

四、 问题一的模型建立与求解

问题一要求基于用户历史交互数据来预测 2024 年 7 月 21 日博主的新增关注数。对于该问题，我们可以将博主的新增关注数视为一个回归问题，使用随机森林模型来进行预测。

4.1 数据准备与特征构建

首先，我们将用户的历史行为数据转化为一组可以描述用户与博主之间的关系的特征数据，以便于随机森林学习。在该题中，我们使用滑动时间窗口，将前 $k=3$ 天的互动数据作为预测第四天的特征数据。并且基于用户-行为-博主三者之间的简单关系，构造特征向量 $X_{b,D_{pred}}$，定义为:

$$X_{b,D_{pred}} = [\Sigma_{d=D_{pred}-k}^{D_{pred}-1} N_{b,d}^{watch}, \Sigma_{d=D_{pred}-k}^{D_{pred}-1} N_{b,d}^{like}, \Sigma_{d=D_{pred}-k}^{D_{pred}-1} N_{b,d}^{comment}, \Sigma_{d=D_{pred}-k}^{D_{pred}-1} N_{b,d}^{follow}]$$

其中，$D_{pred}-K$ 到 $D_{pred}-1$ 表示该次预测时间窗口的范围。

然后，为了训练预测模型，我们基于构造预测特征的方式构造训练样本，与之不同的是在此需要的目标变量是实际的新增关注数。训练数据的日期范围设定为可以恰好从历史数据中构建完整 k 天特征的日期，即从 2024.7.11+k 天到 2024.7.20 日。

对于历史日期 d_{train}（2024.7.14 2024.7.20），构建训练样本 $(X_{b,d_{train}}, Y_{b,d_{train}})$:

$$X_{b,d_{train}} = [\sum_{i=1}^{k} N_{b,d_{train}-i}^{watch}, \sum_{i=1}^{k} N_{b,d_{train}-i}^{like}, \sum_{i=1}^{k} N_{b,d_{train}-i}^{comment}, \sum_{i=1}^{k} N_{b,d_{train}-i}^{follow}]$$

$$Y_{b,d_{train}} = N_{b,d_{train}}^{follow}$$

将所有在时间窗内的样本组合起来，形成数据集 (X_{train}, Y_{train})。

4.2 选择预测模型

随机森林模型是由多棵决策树构成的集成学习模型。每棵决策树都是通过对训练数据进行有放回抽样得到的子集来训练的。每棵树在训练时会随机选择特征进行

分裂，最终通过投票或平均值来得到预测结果。它能够有效地处理特征之间的联系，并且对于噪声有较强的鲁棒性。

模型的数学表达式可以表示为：

$$P_{b,D_{pred}}^{follow} = f(X_{b,D_{pred}})$$

其中 f 代表训练好的随机森林回归模型。该模型简要结构如下图所示：

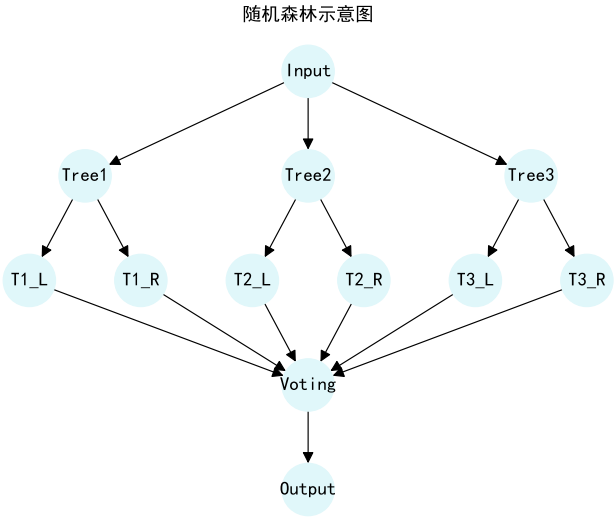

图 1　随机森林模型简要结构

4.3 模型参数调优

为了获取最优的模型性能，使用 RandomizedSearchCV 对模型参数进行调优。我们选择了以下参数进行调优：

- n_estimators: 森林中树的数量。
- max_depth: 树的最大深度。
- min_samples_split: 节点分裂所需的最小样本数。
- min_samples_leaf: 叶子节点所需的最小样本数。
- max_features: 每棵树分裂时考虑的特征数量。

参数搜索空间根据经验设定。采用 KFold 交叉验证策略（例如，5 折），以负平均绝对误差 (neg_mean_absolute_error) 作为评估指标，寻找使交叉验证得分最高的参数组合。

4.4 模型的求解

1. 训练模型：使用 RandomizedSearchCV 获得最优参数后，使用训练集训练模型得到最终的预测模型。
2. 构建预测数据集：根据4.1中的方法，构建用于预测 2024 年 7 月 21 日新增关注数的特征向量集合 X_{pred}。确保包含所有在历史数据中出现过的博主。
3. 预测：使用训练好的最终模型对 X_{pred} 进行预测，得到初步的预测结果 $\hat{P}_{b,D_{pred}}^{follow}$。
4. 结果后处理：对初步预测结果进行后处理，将预测值四舍五入取整，并确保预测结果非负，得到最终的预测新增关注数 $P_{b,D_{pred}}^{follow}$。

5. 排名生成：将每个博主的 ID 与其对应的预测新增关注数关联起来，按预测关注数降序排序。

根据以上流程，得到 2024 年 7 月 21 日新增关注数的预测结果，可以看到结果如下。

表 2　2024 年 7 月 21 日新增关注数预测结果

排名	1	2	3	4	5
博主 ID	B21	B5	B15	B60	B13
新增关注数	503	498	378	372	293

4.5 模型分析

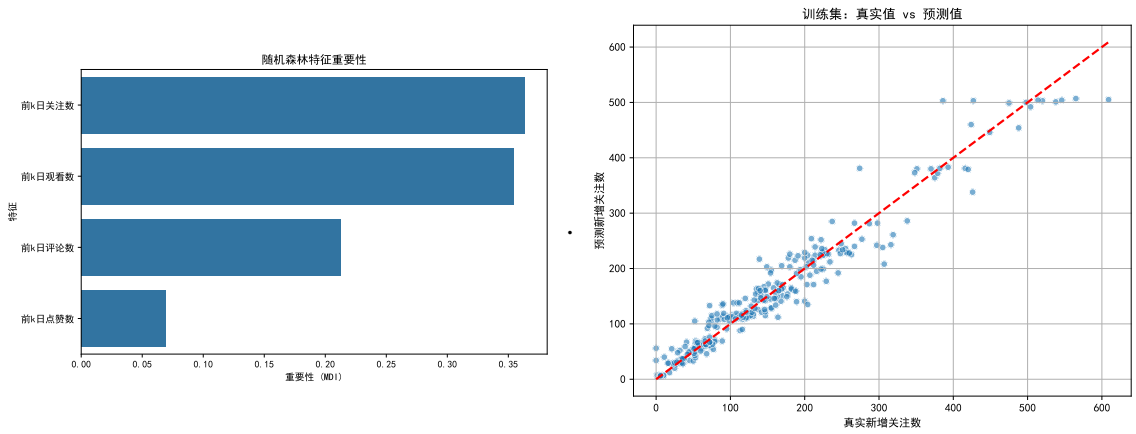

图 2　随机森林模型的特征重要性和训练集预测值散点图

通过以上两张图可以发现，随机森林模型对于前 k 日关注数和观看数有着较高重视程度，说明博主的新增关注数与历史关注数和观看数有着较强的线性关系。而且模型的训练集预测值散点图显示，模型在训练集上的拟合效果良好，说明模型能够较好地捕捉到博主新增关注数的变化趋势。

此外我们也使用训练集对最终模型进行性能评估，使用均方根误差 (RMSE)、平均绝对误差 (MAE)、决定系数 (R^2) 来衡量模型的性能。结果如下：

表 3　模型评估结果

指标	结果
RMSE	26.6882
MAE	18.2211
R^2	0.9491

可以看出，模型在训练集上的表现良好，R^2 值接近 1，说明模型能够较好地拟合训练数据。

五、 问题二的模型建立与求解

问题二要求基于用户历史互动数据，预测 2024 年 7 月 22 日用户可能关注的博主。对于该题，我们可以视为一个二分类问题，通过构建用户-博主对特征，使用 LightGBM 模型进行学习分类。

5.1 数据准备与标签构建

为了训练分类模型，我们通过构造带有标签（是否新关注）的用户-博主对特征数据，使用 LightGBM 模型进行学习分类。在标签数据构造中，我们构造了如下两个样本：

- 正样本 (label=1): 在特定日期（如训练日的 2024.7.19 或验证日的 2024.7.20）发生的新关注行为对应的用户-博主对。新关注的定义是该用户在该日期之前（包括该日期之前的所有历史数据）没有关注过该博主，但在该日期发生了关注行为。
- 负样本 (label=0): 在特定日期（如训练日的 2024.7.19 或验证日的 2024.7.20）没有发生关注行为的用户-博主对。

5.2 特征构建

为每个用户-博主对构建特征向量 $X_{u,b}$，以捕捉影响关注行为的因素。特征可以分为四类：

1. 用户级别特征：基于用户在历史数据截止日期前的总互动数据计算，反映用户的活跃度、偏好等。例如：总互动次数、关注博主数量、观看/点赞/评论/关注次数、活跃天数、关注率、距离上次活跃天数等。
2. 博主特征：基于博主在历史数据截止日期前的总互动数据计算，反映博主的活跃度、偏好等。例如：总互动次数、关注博主数量、观看/点赞/评论/关注次数、活跃天数、关注率、距离上次活跃天数等。
3. 用户-博主交互特征：基于特定用户与特定博主在历史数据截止日期前的互动数据计算，反映用户对该博主的兴趣和粘性。例如：总互动次数、观看/点赞/评论次数、首次/最后一次互动距今的天数、互动天数、互动频率等。
4. 当天互动特征：基于预测日（或训练/验证目标日）当天用户与博主的互动数据计算，反映用户在预测日对该博主的即时兴趣。例如：当天总互动次数、当天是否发生了观看/点赞/评论行为（二值特征）。

特征计算时，历史数据截止日期根据训练、验证或预测阶段的不同而变化（训练集使用 7.18 之前历史，验证集使用 7.19 之前历史，预测集使用 7.20 之前历史）。当天互动数据则使用训练日（7.19）、验证日（7.20）或预测日（7.22）的数据。对计算出的特征进行缺失值填充（如计数类特征填充 0，天数类特征填充-1 表示未发生）。

5.3 选择分类模型

本题选择 LightGBM (Light Gradient Boosting Machine) 分类模型。LightGBM 是一种高效的梯度提升决策树框架，具有训练速度快、内存占用低、精度高等优点，特别适合处理大规模数据和类别特征。模型的目标是预测用户新关注博主的概率。

模型的数学表达式可以表示为：

$$P(L_{u,b,D_{pred}} = 1 | X_{u,b,D_{pred}}) = f_{cls}(X_{u,b,D_{pred}})$$

其中 f_{cls} 代表训练好的 LightGBM 分类模型。

5.4 数据处理与特征工程

根据5.1和5.2中的方法，构造训练集和验证集：

- 训练集：
  - 标签日期: 2024.7.19
  - 历史数据截止日期: 2024.7.18
  - 当天互动数据: 2024.7.19 的互动数据
  - 构建 (X_{train}, Y_{train})
- 验证集：
  - 标签日期: 2024.7.20
  - 历史数据截止日期: 2024.7.19
  - 当天互动数据: 2024.7.20 的互动数据
  - 构建 (X_{val}, Y_{val})

确保训练集和验证集的特征列一致，将 $UserID$ 和 $BloggerID$ 的标记为类别特征。

此外，特征工程是模型效果的关键。需要提取能够反映用户关注倾向和用户对特定博主兴趣的特征。

- 用户特征：
  - 用户历史总互动次数。
  - 用户历史观看/点赞/评论/关注次数及比例。
  - 用户历史互动过的不同博主数量。
  - 用户历史关注总数。
  - 用户活跃天数。
  - 用户近 N 天（如近 3 天、近 7 天）的互动次数/类型分布。
- 博主特征：
  - 博主历史被互动总次数。
  - 博主历史被观看/点赞/评论/关注次数及比例。
  - 博主历史互动过的不同用户数量。
  - 博主历史被关注总数。
  - 博主近 N 天被互动次数/类型分布。
- 用户-博主交互特征：(这是最重要的特征类别)
  - 用户与该博主历史总互动次数。
  - 用户与该博主历史观看/点赞/评论次数。(注意：不包含关注次数，因为关注是我们要预测的，且假设关注后不再变化)
  - 用户与该博主首次互动时间距今时长。
  - 用户与该博主最后一次互动时间距今时长。
  - 用户与该博主历史互动频率（例如，互动次数/互动天数）。
  - 近期互动特征：
    - 训练时 (预测 7.20): 用户在 7.20 当天与该博主的互动次数。用户在 7.20 当天与该博主的互动类型（是否有观看、是否有点赞、是否有评论）。
    - 预测时 (预测 7.22): 用户在 7.22 当天与该博主的互动次数。用户在 7.22 当天与该博主的互动类型（是否有观看、是否有点赞、是否有评论）。

5.5 模型的训练

使用构建好的训练集 (X_{train}, y_{train}) 与特征训练 LightGBM 模型。在训练过程中，使用验证集 (X_{val}, y_{val}) 进行评估，并采用早停 (early stopping) 策略。以 AUC (Area Under the ROC Curve) 作为评估指标，当验证集上的 AUC 在一定数量的迭代次数内没有提升时，停止训练，并使用最佳迭代次数的模型。这有助于防止模型过拟合。

5.6 模型的求解

1. 准备预测数据与特征：
    - 从附件 2 中筛选出指定用户在 2024 年 7 月 22 日的互动数据。
    - 在附件 1 中找出这些用户在 2024 年 7 月 22 日之前（即截止到 2024 年 7 月 20 日）已经关注过的博主。
    - 构建预测候选对：从目标用户在 2024 年 7 月 22 日有互动的博主中，移除他们已经关注过的博主。这些剩余的用户-博主对是潜在的新关注对象。
    - 构建预测特征：为每个预测候选对构建特征向量 $X_{u,b,D_{pred}}$。历史数据截止日期为 2024.7.20，当天互动数据使用 2024.7.22 的互动数据。确保预测特征的列与训练特征一致。
2. 预测：使用训练好的 LightGBM 模型对预测特征进行预测，得到每个候选对新关注的概率 $P(L_{u,b,D_{pred}} = 1|X_{u,b,D_{pred}})$.
3. 确定预测结果：设定一个概率阈值 $T_{threshold}$ (在本题目中根据允许的最低召回率确定)。如果一个用户-博主对的预测概率大于 $T_{threshold}$，则认为该用户在 2024 年 7 月 22 日会新关注该博主。

根据模型的预测结果，指定的几位用户在 2024 年 7 月 22 日预测新关注的博主 ID 如表 2 所示：

表 4　用户在 2024 年 7 月 22 日预测新关注的博主 ID

用户 ID	U7	U6749	U5769	U14990	U52010
新关注博主 ID	B7, B27	B17	/	B24	B13

六、 问题三的模型建立与求解

问题三可以分成两个子任务，任务 1 是预测用户在线情况，任务 2 是预测用户会与哪些博主互动。对于任务 1，用户在线情况是一个二分类问题，用户是否在线与用户活跃天数、互动频率、最后活跃时间等特征有关，本题中依旧选择 LightGBM 分类器进行预测。对于任务 2，用户与博主的互动次数预测是一个回归问题，但是 LightGBM 在不同参数下可以实现回归和分类，因此我们可以使用 LightGBM 回归器来预测用户与博主的互动次数。

6.1 特征工程

在特征工程上，基于5.2中的特征构建方法进行改进，具体改进如下：

- 引入时序特征：除了计算基于历史范围的聚合特征外，还基于不同时间窗口（例如近 1 天、近 3 天、近 7 天）的聚合特征。其中包括：
    - 用户层面：

    - 用户总互动次数/各互动行为次数
    - 活跃天数/最后活跃天数差/平均每日互动
  - 用户-博主对层面：
    - 用户与博主总互动次数/各互动行为次数
    - 最后互动天数差

6.2 模型的训练与求解

本题仍然选择 LightGBM 分类器进行训练，模型的训练与求解方法与问题 2 基本相同，在此仅介绍不同部分。

在训练互动预测模型时，模型需要学习根据历史数据预测特定用户与特定博主的互动次数，因此需要使用时序特征，并且选择了适合训练参数使模型能够学习时间联系。互动预测模型的预测过程如下：

1. 检查任务 1 中用户是否标记为在线，如果非在线状态则跳过互动预测。
2. 生成候选的用户-博主对：对于每一个被预测为在线的用户，模型将预测用户与互动过的博主组成的候选对，因为用户最有可能在预测当天继续与他们之前互动过的博主互动。
3. 为候选对计算基于 7.11-7.20 的数据历史特征
4. 使用训练好的互动模型对这些候选对进行预测，得到每个用户与每个候选博主可能产生的互动次数。
5. 按照预测的互动次数对候选对进行排序，选择互动次数最高的三位博主。

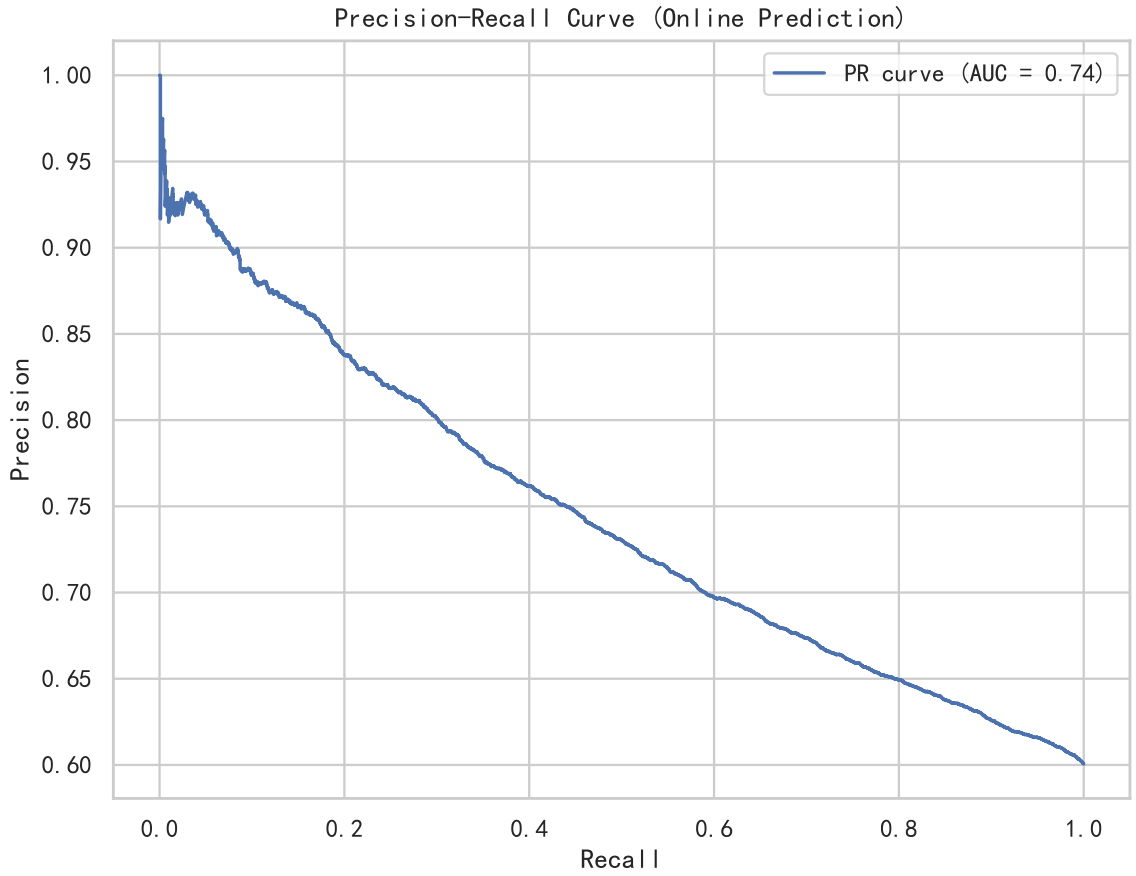

图 3　在线预测模型的 PR 曲线

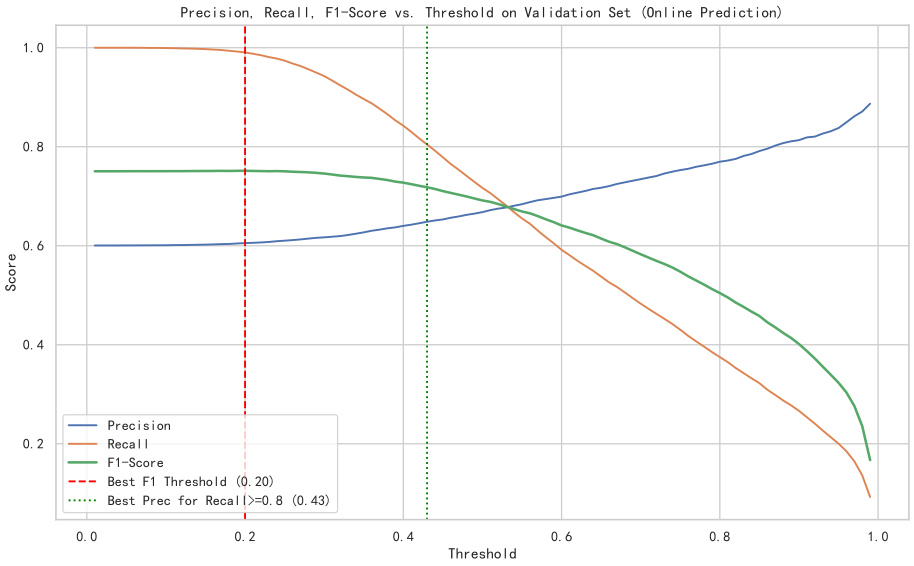

图 4　在线预测模型的 F1-Score 曲线

在二分类问题中，PR 曲线和 F1-Score 曲线是反映模型性能的直观指标。PR 曲线是精确率（Precision）和召回率（Recall）之间的关系图，F1-Score 是精确率和召回率的调和平均数。我们可以通过调整模型的阈值来平衡精确率和召回率。PR 曲线反映了不同召回率下的精确率表现，而 F1-Score 曲线则展示了不同阈值下的 AUC（PR 曲线下的面积）值和 F1-Score 的变化。我们可以通过这两个图表来评估模型的性能，并选择合适的阈值。

在训练过程中，我们可以看到模型的 PR 曲线和 F1-Score 曲线是可以体现出模型具有区分度的。随着阈值的增加，模型的准确率逐渐增加，但是召回率逐渐降低，因此我们可以设定一个最低可以接受的召回率（图4绿线所示），作为阈值选择的依据。

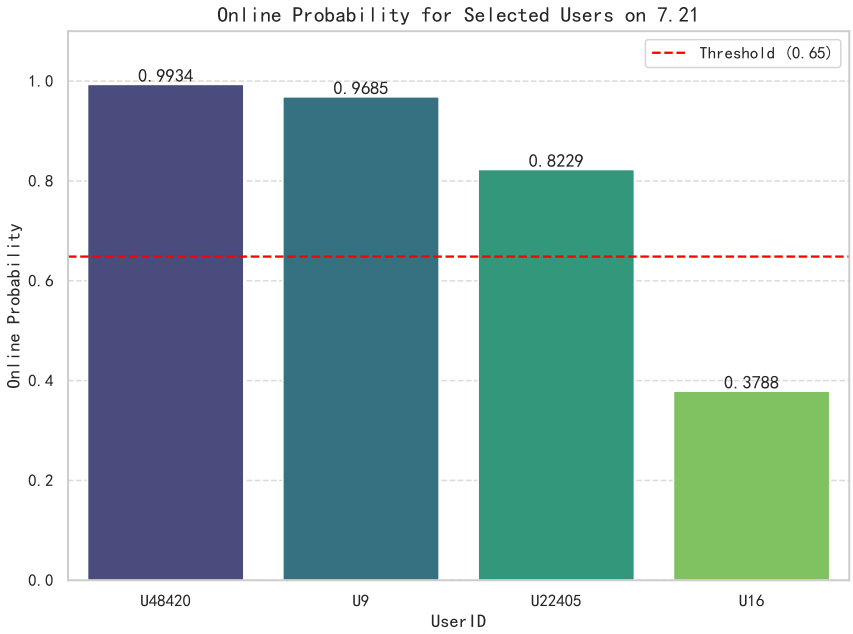

图 5　用户在线概率

最后我们可以得到上图所示的用户在线概率，并且可以看到用户 U16 在 2024 年 7 月 22 日在线的概率很低，因此可以认为用户 U16 在 2024 年 7 月 22 日不会在线，并且 U16 也的在线概率也明显低于阈值 $T_{threshold}$。最终答案为下表所示：

表 5　预测在 2024 年 7 月 21 日用户的预测互动的博主 ID

用户 ID	博主 ID 1	博主 ID 2	博主 ID 3
U9	B13	B7	B24
U22405	B52	B42	B47
U16	/	/	/
U48420	B5	B2	B21

6.3 模型分析

由于时间限制，我们未能对 LightGBM 回归模型在独立的测试集上进行充分的性能评估。因此我们从标签分布以及特征分布和相关性进行模型分析，从而大概的反映模型学习的特征。

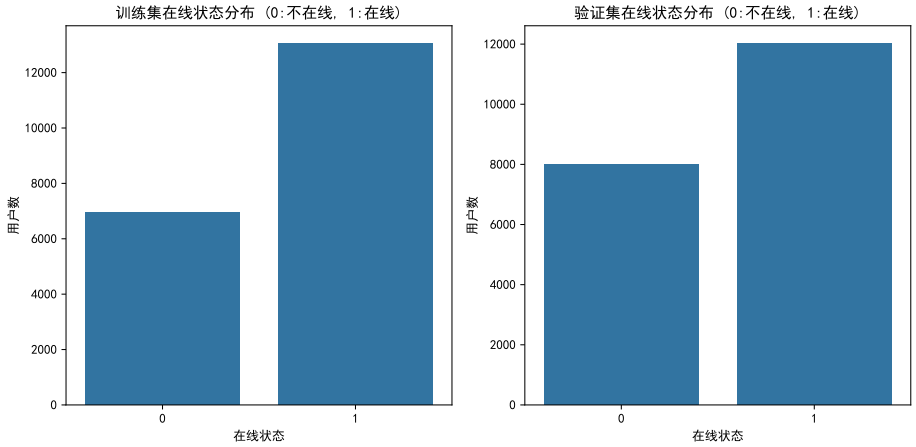

图 6　用户在线预测模型的标签分布

通过上图可以发现，模型预测的在线用户数远大于离线用户数，结合互联网发展情况和电子产品普及程度来看，该结果是符合现实的。需要注意的是，此处评估结果反映模型在训练集上的拟合能力，其泛化能力尚待在独立的测试集上进行验证。

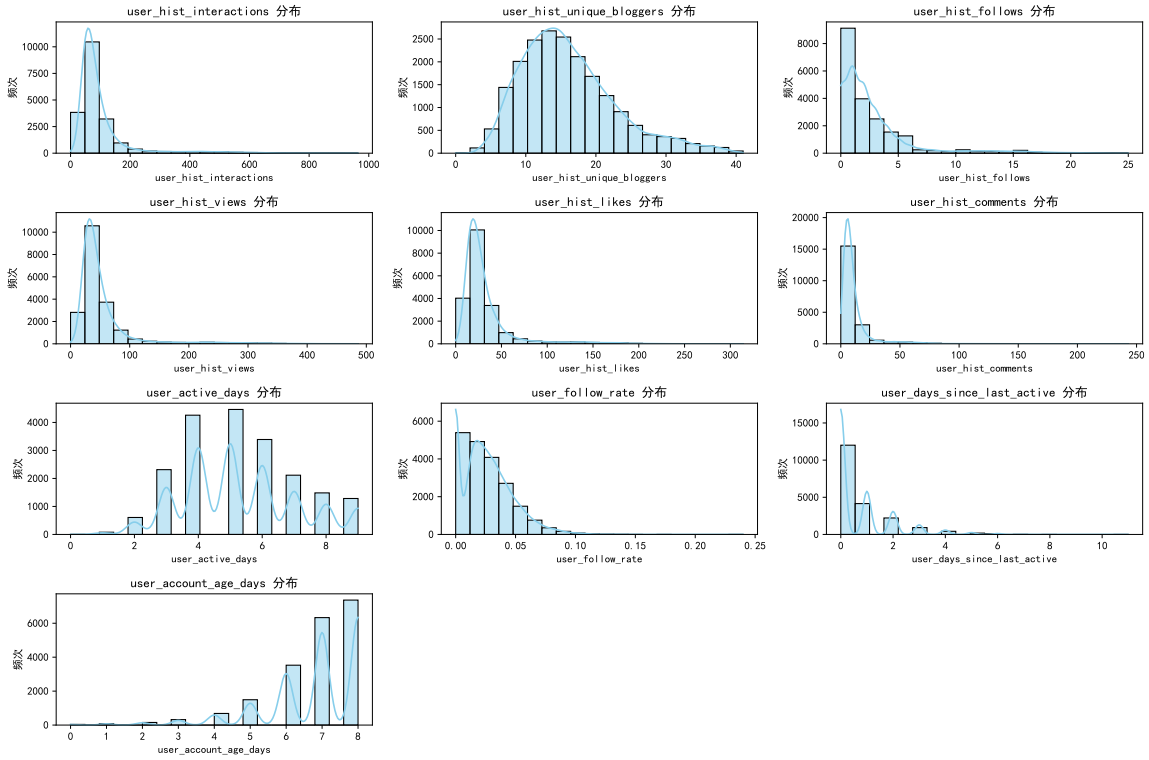

图 7　用户特征分布热图

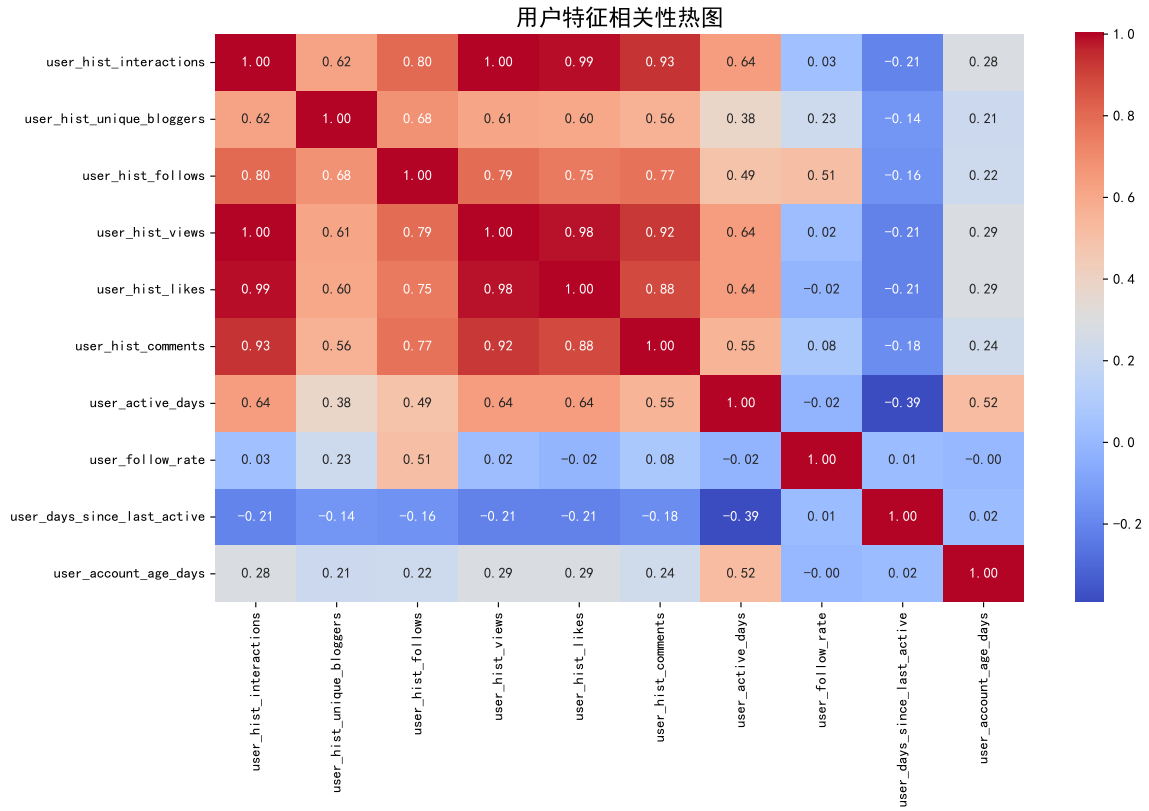

图 8　用户特征相关性热图

通过图7、图8可以发现，互动行为（浏览、点赞、评论）之间高度相关，并且

与互动总数、关注数、互动过的唯一博主数以及活跃天数有较强的正相关性。但是也存在一些负相关性，例如互动次数与关注数、互动过的唯一博主数以及活跃天数有较强的负相关性。

七、 问题四的模型建立与求解

问题四要求预测用户在 2024 年 7 月 23 日的在线时段，并根据每个时段的互动数，给出互动最多的三位博主和时段。因为 LightGBM 模型是基于静态数据进行分类回归的，虽然我们可以将时序特征绑定在别的特征上，但是其本身难以学习到与时间特征强相关的内容。所以在此我们预测互动时段转为基于历史行为频率来分析。我们将其分为两个子任务，任务 1 是预测用户在线情况，任务 2 是基于历史行为分析时段互动偏好并给出 Top 3。

7.1 任务 1：数据预处理与特征工程

数据预处理和特征工程方法与问题 3 的在线预测子任务基本一致。使用附件 1 的全部历史数据（2024.7.11-2024.7.20）作为训练和验证的基础。

1. 构建训练集和验证集: 沿用问题 3 的训练集（预测 7.20 在线，特征基于 7.11-7.19）和验证集（预测 7.19 在线，特征基于 7.11-7.18）。
2. 构建特征: 使用问题 3 中生成的用户级别特征，包括基于整个历史和不同时间窗口（1 天、3 天、7 天）的时序特征。对缺失值进行填充。

7.2 任务 1：模型训练与阈值选择

使用问题 3 训练好的 LightGBM 分类模型，并在验证集上评估性能，选择最佳预测概率阈值 $T_{online\_threshold}$。沿用问题 3 中通过评估指标（如 F1、PR 曲线）确定的阈值选择策略和最终阈值。

7.3 任务 2：选择预测模型

对于任务 2 中，需要找出在 2024 年 7 月 23 日用户在线时，用户会与哪些博主互动，给出 Top 3 的博主-时段组合。因此，本题目不使用 LightGBM 作为预测模型，因为时序特征的处理能力有限，难以高效学习到小时颗粒度下的用户-博主-时段的交互模式。所以在，本任务中直接分析用户在历史数据中表现出的活跃时段偏好以及互动偏好，来预测用户在 2024 年 7 月 23 日的在线时段和互动博主。

7.4 任务 2：预测方法

对于在子任务 1 中预测为在线的每个用户：

1. 识别历史活跃时段：统计用户在附件 1 历史数据（2024.7.11-7.20）中每个小时的行为总数 $H_{u,h}^{hist}$。用户历史行为过的小时段被认为是潜在的活跃时段。
2. 统计历史时段内互动：对于用户历史行为过的每个小时段 h，统计用户在该时段内与每个博主 b 发生的互动总数 $I_{u,b,h}^{hist}$（点赞 + 评论 + 关注之和）。
3. 生成候选组合：将用户历史行为过的所有小时段，与用户历史在这些小时段内互动过的所有博主进行组合，形成 (用户 ID, 博主 ID, 时段) 候选三元组。每个三元组的“预测互动数”即为用户历史在该时段内与该博主的互动总数 $I_{u,b,h}^{hist}$。
4. 选取 Top 3：从所有候选三元组中，找出预测互动数 $I_{u,b,h}^{hist}$ 最高的 3 个组合。每个组合对应一个博主 ID 和一个时段。如果某个用户历史互动总数（点赞 + 评论 + 关注）不足 3 个，则给出所有预测互动数大于 0 的组合。

7.5 模型的求解

按照题目的预测要求，我们按照以下步骤进行预测：

Step1： 通过问题 3 中的在线预测模型，预测 2024 年 7 月 23 日用户在线情况。

Step2： 对于每个预测为在线的用户，使用任务 2 中的方法进行预测，得到每个用户-博主-时段组合的预测互动数。

Step3： 对每个用户，按照预测互动数从高到低排序，选择 Top 3 组合。

最终求解得到的结果如下表所示：

表 6　用户在 2024 年 7 月 23 日各时段互动的博主

用户 ID	U10	U1951	U1833	U26447
博主 ID1	B2	/	B19	B13
时段 1	02:00-03:00	/	01:00-02:00	02:00-03:00
博主 ID2	B2	/	B15	B45
时段 2	15:00-16:00	/	03:00-04:00	15:00-16:00
博主 ID3	B23	/	B13	B45
时段 3	16:00-17:00	/	11:00-12:00	20:00-21:00

7.6 模型分析

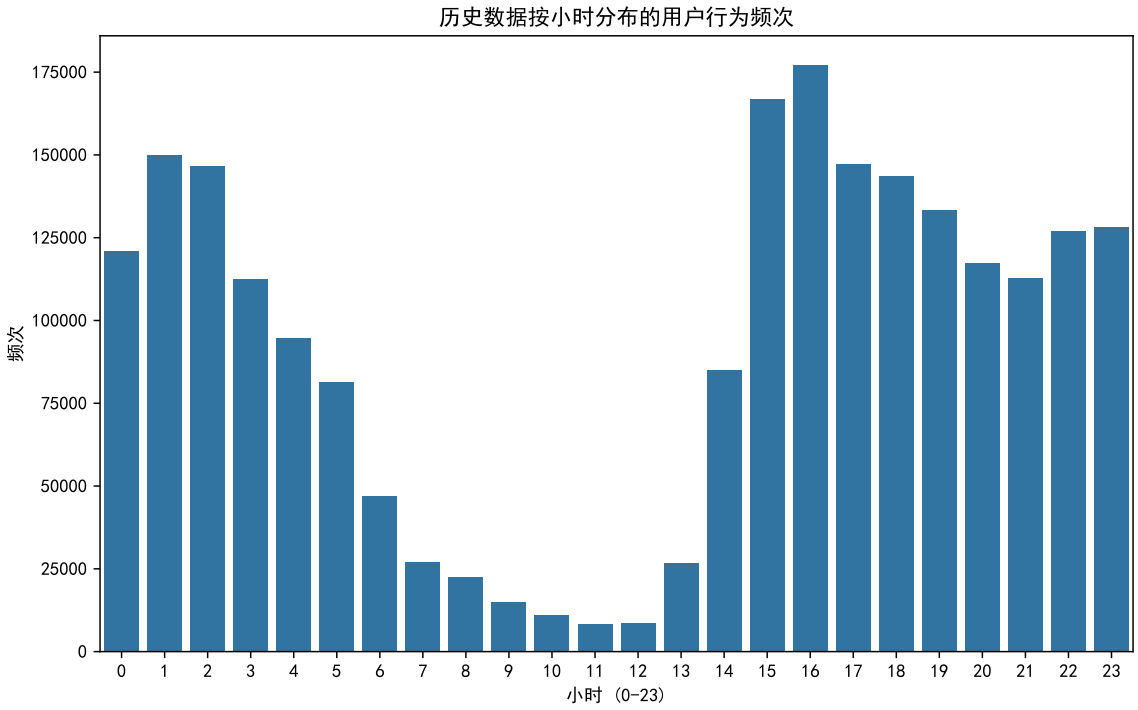

图 9　按小时分布的用户行为频次

通过上图可以发现，用户在 2024 年 7 月 22 日的互动次数基本在 6 到 14 点间是谷底，而在 15 点到次日 5 点则有大量行为发生，因此可以推测大部分用户均习惯在

晚上或凌晨在线。结合问题四的结果来看，绝大部分操作都集中在行为频次较高的时段，由此可以推断问题四的预测结果的合理性。

八、 模型评价与改进

8.1 模型的优点

通过问题一的随机森林模型，我们可以发现，随机森林模型对于日关注数的敏感度较高，且模型的预测值与实际值的散点图显示模型具有较好的拟合能力。因此可以认为随机森林模型在应对日关注数预测问题等回归问题时具有较好的预测能力。

通过问题二、问题三的 LightGBM 模型，我们可以发现，LightGBM 模型更擅长于处理高维数据，且模型的预测值与实际值的散点图显示模型具有较好的拟合能力，并且模型的训练速度具有优势。验证集与训练集的标签分布有较高的相似度，且标签分布符合实际情况，因此可以认为 LightGBM 模型在应对用户在线预测问题等二分类问题时具有较好的预测能力。

8.2 模型的缺点

通过问题三可以发现，LightGBM 模型在预测在线情况时，F1-Score 并不能直接作为阈值的参考，需要结合预测情况和召回率百分比来选择阈值。并且模型的 AUC 值相对于第二问较低，说明模型在遇到时序数据时，模型的预测质量降低。

而 LightGBM 模型在处理与时间序列强相关的问题时，模型的预测效果较差，这说明模型在静态学习能力较强，而在时序学习能力较弱，需要大量特征提高模型的时序学习能力。并且模型面对时间颗粒度较高的场景，需要大量数据进行学习，训练效果难以保证。

8.3 模型的改进

使用前置模型预处理数据，将稠密的数据稀疏化，使得模型能够更好地学习到数据的特征。通常可以结合机器学习或深度学习模型来提取数据特征，从而提高模型的预测能力。

可以通过结合 cuda 并行计算技术，提高 LightGBM 的训练速度，使得可以使用更多数据进行训练，从而提高模型的预测能力。

参考文献

[1] 谢军飞, 张海清, 李代伟, 等. 基于 Lightgbm 和 XGBoost 的优化深度森林算法 [J]. 南京大学学报 (自然科学版), 2023, 59(5): 833-840

[2] Microstrong, 深入理解 LightGBM, https://zhuanlan.zhihu.com/p/99069186, 2025.05.04.

[3] 韩信子, 图解机器学习 | LightGBM 模型详解, https://www.showmeai.tech/article-detail/195, 2025.05.04.

[4] 梦迪, 翟. “Weibo Recommendation System Based on Semantic Analysis.”Computer Science and Application 06 (2016): 531-538.

[5] 司守奎，孙玺菁. 数学建模算法与应用. 北京：国防工业出版社，2012：P57

附录 A　问题一随机森林实现

```python
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV, KFold
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score,
) # 导入评估指标
from scipy.stats import randint, uniform # 用于定义随机搜索的参数分布
import warnings
import os # 引入os模块检查文件是否存在

# 忽略可能的警告
warnings.filterwarnings("ignore")

# --- 配置 ---
file_path = "program/a1.csv" # 确保文件路径正确
prediction_date = datetime(2024, 7, 21)
historical_start_date = datetime(2024, 7, 11)
historical_end_date = datetime(2024, 7, 20)
lookback_days = 3 # 使用前 k 天的数据作为特征，这里 k=3

# --- 检查文件是否存在 ---
if not os.path.exists(file_path):
    print(f"错误: 文件未找到 - {file_path}")
    print("请确保 '附件1.csv' 文件与脚本在同一目录下。")
    exit()

# --- 1. 数据加载与预处理 ---
print(f"正在加载数据: {file_path}...")
try:
    # 尝试更高效的读取方式，指定dtype
    dtype_spec = {
        "UserID": "category",
        "UserBehaviour": "int8",
        "BloggerID": "category",
        "Time": "object", # 先读object，再转datetime
    }
    df = pd.read_csv(file_path, dtype=dtype_spec, low_memory=False)
    print("数据加载完成.")

    # 转换时间列
    print("正在转换时间格式...")
    df["Time"] = pd.to_datetime(df["Time"])
    df["Date"] = df["Time"].dt.date
```

```
    print("时间格式转换完成.")

    # 过滤历史数据范围
    print(
        f"正在过滤数据范围至 {historical_start_date.date()} -
            {historical_end_date.date()}..."
    )
    df_history = df[
        (df["Date"] >= historical_start_date.date())
        & (df["Date"] <= historical_end_date.date())
    ].copy()
    print(f"过滤完成，剩余 {len(df_history)} 条数据.")

    # 释放原始大DataFrame内存
    del df

except Exception as e:
    print(f"数据加载或初步处理失败: {e}")
    exit() # 如果数据加载失败，直接退出

# 获取所有独特的博主ID，后续预测需要为所有这些博主进行
all_blogger_ids = df_history["BloggerID"].unique()
print(f"共有 {len(all_blogger_ids)} 位博主在历史数据中出现.")

# --- 2. 计算每日互动计数 ---
print("正在计算每日博主互动计数...")
# 使用pivot_table更方便地获取各种行为的每日计数
daily_interactions = (
    pd.pivot_table(
        df_history,
        values="UserID", # 任意一个值列即可，我们只关心计数
        index=["BloggerID", "Date"],
        columns="UserBehaviour",
        aggfunc="count",
        fill_value=0, # 关键：用0填充没有发生某种行为的日期/博主组合
    )
    .rename(
        columns={
            1: "watch_count",
            2: "like_count",
            3: "comment_count",
            4: "follow_count",
        }
    )
    .reset_index()
)

# 确保所有博主在所有历史日期都有记录，即使计数为0
# 创建所有博主和历史日期的组合
```

```
all_dates_in_history = pd.date_range(
    start=historical_start_date.date(), end=historical_end_date.date(), freq="D"
).date
all_blogger_date_combinations = pd.MultiIndex.from_product(
    [all_blogger_ids, all_dates_in_history], names=["BloggerID", "Date"]
)
daily_interactions = (
    daily_interactions.set_index(["BloggerID", "Date"])
    .reindex(all_blogger_date_combinations, fill_value=0)
    .reset_index()
)

print("每日互动计数计算完成.")
# print(daily_interactions.head())

# --- 3. 构建训练数据集 ---
print(f"正在构建训练数据集 (使用前 {lookback_days} 天数据预测当天)...")

X_train = []
y_train = []
train_samples_meta = [] # 存储样本对应的博主和日期

# 训练的目标日期范围: 从 historical_start_date + lookback_days 到 historical_end_date
train_start_date = historical_start_date + timedelta(days=lookback_days)
train_end_date = historical_end_date

# 确保训练数据日期范围有效
if train_start_date > historical_end_date:
    print(
        f"错误: 历史数据范围 ({historical_start_date.date()} -
            {historical_end_date.date()}) 太短, 无法构建训练集 (需要至少 {lookback_days +
            1} 天)."
    )
    exit()

current_train_date = train_start_date
while current_train_date <= train_end_date:
    lookback_end = current_train_date - timedelta(days=1)
    lookback_start = current_train_date - timedelta(days=lookback_days)

    # 提取当前训练日期的目标数据
    target_day_data = daily_interactions[
        daily_interactions["Date"] == current_train_date.date()
    ]

    # 提取当前训练日期的特征数据 (前 lookback_days 的汇总)
    feature_window_data = daily_interactions[
        (daily_interactions["Date"] >= lookback_start.date())
        & (daily_interactions["Date"] <= lookback_end.date())
```

```
    ]

    # 按博主ID汇总特征窗口内的数据
    features_aggregated = (
        feature_window_data.groupby("BloggerID")[
            ["watch_count", "like_count", "comment_count", "follow_count"]
        ]
        .sum()
        .reset_index()
    )
    features_aggregated.rename(
        columns={
            "watch_count": "sum_watch_prev",
            "like_count": "sum_like_prev",
            "comment_count": "sum_comment_prev",
            "follow_count": "sum_follow_prev",
        },
        inplace=True,
    )

    # 迭代所有博主，为每个博主在当前训练日期构建样本
    for blogger_id in all_blogger_ids:
        # 获取特征 (处理可能没有数据的博主)
        blogger_features = features_aggregated[
            features_aggregated["BloggerID"] == blogger_id
        ]
        if blogger_features.empty:
            features_row = [0, 0, 0, 0] # 如果前lookback_days没有数据，特征为0
        else:
            features_row = (
                blogger_features[
                    [
                        "sum_watch_prev",
                        "sum_like_prev",
                        "sum_comment_prev",
                        "sum_follow_prev",
                    ]
                ]
                .iloc[0]
                .tolist()
            )

        # 获取目标 (处理当天没有关注的博主)
        blogger_target = target_day_data[target_day_data["BloggerID"] == blogger_id]
        target_value = (
            blogger_target["follow_count"].iloc[0] if not blogger_target.empty else 0
        ) # 如果当天没数据，目标为0

        X_train.append(features_row)
```

```
            y_train.append(target_value)
            train_samples_meta.append(
                (blogger_id, current_train_date.date())
            ) # 记录样本元数据

        current_train_date += timedelta(days=1)

X_train = np.array(X_train)
y_train = np.array(y_train)

print(f"训练数据集构建完成，共 {len(X_train)} 个样本.")
# print("X_train sample:", X_train[:5])
# print("y_train sample:", y_train[:5])

# --- 4. 参数调优 ---
print("\n正在进行参数调优 (RandomizedSearchCV)...")

# 定义参数搜索空间
param_distributions = {
    "n_estimators": randint(50, 301), # 树的数量在 50 到 300 之间随机整数
    "max_depth": [None]
    + list(
        randint(10, 51).rvs(size=20)
    ), # 最大深度可以是 None，或者 10 到 50 之间的一些随机整数 (减少样本数量)
    "min_samples_split": randint(2, 21), # 分裂所需的最小样本数在 2 到 20 之间随机整数
    "min_samples_leaf": randint(1, 11), # 叶节点所需的最小样本数在 1 到 10 之间随机整数
    "max_features": ["sqrt", "log2", 1.0], # 寻找最佳分裂时考虑的特征数量
}

# 交叉验证策略
# 使用 KFold with shuffle。cv=5 表示 5折交叉验证。
cv_strategy = KFold(n_splits=5, shuffle=True, random_state=42)

# 初始化 RandomizedSearchCV
# n_iter: 随机采样的参数组合数量。这里设置为 50 次尝试，可以根据计算资源调整。
# scoring: 使用负平均绝对误差作为评估指标，越高越好 (因为是负的误差)。
# random_state: 保证结果可复现。
# n_jobs=-1: 利用所有核心进行并行计算。
random_search = RandomizedSearchCV(
    estimator=RandomForestRegressor(random_state=42), # 传入一个基础模型实例
    param_distributions=param_distributions,
    n_iter=50, # 尝试 50 个不同的参数组合，这个值可以根据计算资源调整
    cv=cv_strategy,
    scoring="neg_mean_absolute_error",
    verbose=1, # 打印进度信息
    random_state=42,
    n_jobs=-1,
)
```

```
# 在训练数据上运行随机搜索
random_search.fit(X_train, y_train)

print("\n参数调优完成.")
print("最佳参数组合:", random_search.best_params_)
print("最佳交叉验证得分 (负MAE):", random_search.best_score_) # 这是负MAE，值越大越好

# --- 5. 使用最优参数训练最终模型 ---
print("\n正在使用最优参数训练最终模型...")
final_model = random_search.best_estimator_ # 获取带有最佳参数的训练好的模型实例
# 注意: best_estimator_ 已经在训练数据上拟合过了，可以直接用于预测

print("最终模型训练完成 (使用最佳参数).")

# --- 6. 模型在训练集上的评价 ---
print("\n正在评价最终模型在训练集上的性能...")

# 使用训练好的模型对训练集进行预测
y_train_pred = final_model.predict(X_train)

# 确保预测结果非负并四舍五入取整，以便与真实值对比
y_train_pred_rounded = np.round(y_train_pred).astype(int)
y_train_pred_rounded[y_train_pred_rounded < 0] = 0

# 计算评估指标
mae_train = mean_absolute_error(y_train, y_train_pred_rounded)
mse_train = mean_squared_error(y_train, y_train_pred_rounded)
rmse_train = np.sqrt(mse_train)
r2_train = r2_score(y_train, y_train_pred_rounded)

print(f"训练集评估结果:")
print(f" 平均绝对误差 (MAE): {mae_train:.4f}")
print(f" 均方误差 (MSE): {mse_train:.4f}")
print(f" 均方根误差 (RMSE): {rmse_train:.4f}")
print(f" 决定系数 (R²): {r2_train:.4f}")

print("模型评价完成.")

# --- 7. 构建预测数据集 ---
print(
    f"\n正在构建预测数据集 (使用 {prediction_date.date()-timedelta(days=lookback_days)}
        - {prediction_date.date()-timedelta(days=1)} 数据预测
        {prediction_date.date()})..."
)

# 预测窗口: 7.18 到 7.20 (如果 lookback_days=3)
prediction_feature_start_date = prediction_date - timedelta(days=lookback_days)
prediction_feature_end_date = prediction_date - timedelta(days=1)
```

```python
# 确保预测特征窗口在历史数据范围内
if (
    prediction_feature_start_date.date() < historical_start_date.date()
    or prediction_feature_end_date.date() > historical_end_date.date()
):
    print(
        f"错误: 预测特征窗口 ({prediction_feature_start_date.date()} -
            {prediction_feature_end_date.date()}) 超出历史数据范围
            ({historical_start_date.date()} - {historical_end_date.date()})."
    )
    exit()


# 提取预测日期的特征数据（前 lookback_days 的汇总）
prediction_feature_window_data = daily_interactions[
    (daily_interactions["Date"] >= prediction_feature_start_date.date())
    & (daily_interactions["Date"] <= prediction_feature_end_date.date())
]

X_pred = []
predict_blogger_ids = []

# 确保为所有在历史数据中出现过的博主生成预测特征
for blogger_id in all_blogger_ids:
    blogger_features = prediction_feature_window_data[
        prediction_feature_window_data["BloggerID"] == blogger_id
    ]

    if blogger_features.empty:
        features_row = [0, 0, 0, 0] # 如果前lookback_days没有数据，特征为0
    else:
        # 汇总前 lookback_days 的数据
        sum_features = (
            blogger_features[
                ["watch_count", "like_count", "comment_count", "follow_count"]
            ]
            .sum()
            .tolist()
        )
        features_row = sum_features

    X_pred.append(features_row)
    predict_blogger_ids.append(blogger_id)

X_pred = np.array(X_pred)

print(f"预测数据集构建完成，共 {len(X_pred)} 个样本.")
# print("X_pred sample:", X_pred[:5])
```

```
# --- 8. 进行预测 ---
print("正在使用最终模型进行预测...")
predictions = final_model.predict(X_pred)
print("预测完成.")

# --- 9. 后处理与结果输出 ---
print("正在处理预测结果并生成排名...")

# 确保预测结果非负并四舍五入取整
predicted_follows = np.round(predictions).astype(int)
predicted_follows[predicted_follows < 0] = 0

# 构建结果DataFrame
results_df = pd.DataFrame(
    {"BloggerID": predict_blogger_ids, "PredictedFollows_20240721": predicted_follows}
)

# 按预测关注数降序排序
results_df = results_df.sort_values(by="PredictedFollows_20240721", ascending=False)

# 选取前5位博主
top_5_bloggers = results_df.head(5)

print("\n预测完成，2024.7.21 当日新增关注数最多的5位博主：")
# 按照表1格式输出
print("表1：2024.7.21 当日新增关注数最多的5位博主")
print("-" * 40)
# 使用 to_markdown 或 to_string 打印为表格格式
try:
    # 如果安装了 tabulate 库
    from tabulate import tabulate

    print(tabulate(top_5_bloggers, headers="keys", tablefmt="github", showindex=False))
except ImportError:
    # 否则使用 to_string
    print(
        top_5_bloggers.rename(
            columns={"PredictedFollows_20240721": "新增关注数"}
        ).to_string(index=False)
    )

print("-" * 40)
```

附录 B　特征提取

```
def create_historical_features(df_hist_subset, reference_date):
# 确保 Date 列是 datetime 类型
df_hist_subset["Date"] = pd.to_datetime(df_hist_subset["Date"])
```

```
# 筛选出截止日期之前的数据
# 使用 .copy() 避免 SettingWithCopyWarning
hist_ref = df_hist_subset[df_hist_subset["Date"] <= reference_date].copy()

# 如果 hist_ref 为空，直接返回空DataFrame
if hist_ref.empty:
    print(
        f"Warning: No historical data found up to {reference_date.date()}. Returning
            empty dataframes."
    )
    # 返回带有必要列名的空DataFrame，以便后续merge不会报错
    user_agg = pd.DataFrame(columns=["UserID"])
    blogger_agg = pd.DataFrame(columns=["BloggerID"])
    user_blogger_hist_agg = pd.DataFrame(columns=["UserID", "BloggerID"])
    return user_agg, blogger_agg, user_blogger_hist_agg

user_agg = (
    hist_ref.groupby("UserID")
    .agg(
        user_hist_interactions=(
            "BloggerID",
            "count",
        ), # Total interactions (all behaviours) in full history
        user_hist_unique_bloggers=("BloggerID", "nunique"),
        user_hist_follows=("UserBehaviour", lambda x: (x == 4).sum()),
        user_hist_views=("UserBehaviour", lambda x: (x == 1).sum()),
        user_hist_likes=("UserBehaviour", lambda x: (x == 2).sum()),
        user_hist_comments=("UserBehaviour", lambda x: (x == 3).sum()),
        user_active_days=("Date", "nunique"), # Active days in full history
        user_first_active_date=("Date", "min"),
        user_last_active_date=("Date", "max"),
    )
    .reset_index()
)
user_agg["user_follow_rate"] = user_agg["user_hist_follows"] / user_agg[
    "user_hist_interactions"
].replace(0, 1)
user_agg["user_days_since_last_active"] = (
    reference_date - user_agg["user_last_active_date"]
).dt.days
user_agg["user_account_age_days"] = (
    reference_date - user_agg["user_first_active_date"]
).dt.days

blogger_agg = (
    hist_ref.groupby("BloggerID")
    .agg(
        blogger_hist_interactions=(
```

```
            "UserID",
            "count",
        ), # Total interactions received in full history
        blogger_hist_unique_users=("UserID", "nunique"),
        blogger_hist_follows=(
            "UserBehaviour",
            lambda x: (x == 4).sum(),
        ), # Follows received
        blogger_hist_views=(
            "UserBehaviour",
            lambda x: (x == 1).sum(),
        ), # Views received
        blogger_hist_likes=(
            "UserBehaviour",
            lambda x: (x == 2).sum(),
        ), # Likes received
        blogger_hist_comments=(
            "UserBehaviour",
            lambda x: (x == 3).sum(),
        ), # Comments received
        blogger_active_days=(
            "Date",
            "nunique",
        ), # Active days (had interactions) in full history
        blogger_first_active_date=("Date", "min"),
        blogger_last_active_date=("Date", "max"),
    )
    .reset_index()
)
blogger_agg["blogger_follow_rate"] = blogger_agg[
    "blogger_hist_follows"
] / blogger_agg["blogger_hist_interactions"].replace(0, 1)
blogger_agg["blogger_days_since_last_active"] = (
    reference_date - blogger_agg["blogger_last_active_date"]
).dt.days
blogger_agg["blogger_account_age_days"] = (
    reference_date - blogger_agg["blogger_first_active_date"]
).dt.days

gc.collect()

# --- 计算用户-博主交互特征 (整个历史期间 up to reference_date) ---
# print("Calculating user-blogger interaction features...") # 移除这个打印
user_blogger_hist_agg = (
    hist_ref.groupby(["UserID", "BloggerID"])
    .agg(
        ub_hist_interactions=(
            "Time",
            "count",
```

```
        ), # Total interactions in full history
        ub_hist_views=("UserBehaviour", lambda x: (x == 1).sum()),
        ub_hist_likes=("UserBehaviour", lambda x: (x == 2).sum()),
        ub_hist_comments=("UserBehaviour", lambda x: (x == 3).sum()),
        ub_hist_follows=(
            "UserBehaviour",
            lambda x: (x == 4).sum(),
        ), # Add ub follows
        ub_first_interaction_date=("Date", "min"),
        ub_last_interaction_date=("Date", "max"),
        ub_interaction_days=("Date", "nunique"), # Interaction days in full history
    )
    .reset_index()
)
user_blogger_hist_agg["ub_days_since_first_interaction"] = (
    reference_date - user_blogger_hist_agg["ub_first_interaction_date"]
).dt.days
user_blogger_hist_agg["ub_days_since_last_interaction"] = (
    reference_date - user_blogger_hist_agg["ub_last_interaction_date"]
).dt.days
user_blogger_hist_agg["ub_interaction_frequency"] = user_blogger_hist_agg[
    "ub_hist_interactions"
] / user_blogger_hist_agg["ub_interaction_days"].replace(0, 1)
user_blogger_hist_agg["ub_follow_rate"] = user_blogger_hist_agg[
    "ub_hist_follows"
] / user_blogger_hist_agg["ub_hist_interactions"].replace(
    0, 1
)  # Add ub follow rate

gc.collect()

# --- 构建时序特征 ---
time_windows = [
    1,
    3,
    7,
]

for window in time_windows:
    window_start_date = reference_date - pd.Timedelta(days=window - 1)
    # Filter data for the current window (within the already filtered hist_ref)
    df_window = hist_ref[hist_ref["Date"] >= window_start_date].copy()

    # Get list of users present in this window
    users_in_window = df_window["UserID"].unique()

    if not df_window.empty:
        # Total interactions (Like, Comment, Follow) in window
        interaction_behaviors_window = df_window[
```

```
        df_window["UserBehaviour"].isin([2, 3, 4])
    ].copy()
    user_interactions_window = (
        interaction_behaviors_window.groupby("UserID")
        .size()
        .reset_index(name=f"user_total_interactions_last_{window}d")
    )
    user_agg = pd.merge(
        user_agg, user_interactions_window, on="UserID", how="left"
    )

    for behaviour_type, behaviour_name in zip(
        [1, 2, 3, 4], ["view", "like", "comment", "follow"]
    ):
        user_behaviour_count = (
            df_window[df_window["UserBehaviour"] == behaviour_type]
            .groupby("UserID")
            .size()
            .reset_index(name=f"user_{behaviour_name}_count_last_{window}d")
        )
        user_agg = pd.merge(
            user_agg, user_behaviour_count, on="UserID", how="left"
        )

    user_active_days_window = (
        df_window.groupby("UserID")["Date"]
        .nunique()
        .reset_index(name=f"user_active_days_last_{window}d")
    )
    user_agg = pd.merge(
        user_agg, user_active_days_window, on="UserID", how="left"
    )

    user_last_active_date_window = (
        df_window.groupby("UserID")["Date"]
        .max()
        .reset_index(name=f"user_last_active_date_last_{window}d")
    )
    user_agg = pd.merge(
        user_agg, user_last_active_date_window, on="UserID", how="left"
    )
    user_agg[f"user_days_since_last_active_last_{window}d"] = (
        reference_date - user_agg[f"user_last_active_date_last_{window}d"]
    ).dt.days
    user_agg = user_agg.drop(f"user_last_active_date_last_{window}d", axis=1)

    total_interactions_col = f"user_total_interactions_last_{window}d"
    active_days_col = f"user_active_days_last_{window}d"
    days_since_col = f"user_days_since_last_active_last_{window}d"
```

```
        user_agg[total_interactions_col] = user_agg[total_interactions_col].fillna(
            0
        )
        for behaviour_name in ["view", "like", "comment", "follow"]:
            col_name = f"user_{behaviour_name}_count_last_{window}d"
            user_agg[col_name] = user_agg[col_name].fillna(0)
        user_agg[active_days_col] = user_agg[active_days_col].fillna(0)
        user_agg[days_since_col] = user_agg[days_since_col].fillna(window)
        user_agg[f"user_avg_daily_interactions_last_{window}d"] = user_agg[
            total_interactions_col
        ] / user_agg[active_days_col].replace(
            0, np.nan
        )  # Use np.nan to avoid ZeroDivisionError
        user_agg[f"user_avg_daily_interactions_last_{window}d"] = user_agg[
            f"user_avg_daily_interactions_last_{window}d"
        ].fillna(
            0
        )

    else:
        user_agg[f"user_total_interactions_last_{window}d"] = 0
        for behaviour_name in ["view", "like", "comment", "follow"]:
            user_agg[f"user_{behaviour_name}_count_last_{window}d"] = 0
        user_agg[f"user_active_days_last_{window}d"] = 0
        user_agg[f"user_days_since_last_active_last_{window}d"] = window
        user_agg[f"user_avg_daily_interactions_last_{window}d"] = 0

    gc.collect()

for window in time_windows:
    window_start_date = reference_date - pd.Timedelta(days=window - 1)
    # Filter data for the current window (within the already filtered hist_ref)
    df_window = hist_ref[hist_ref["Date"] >= window_start_date].copy()

    ub_in_window = df_window[["UserID", "BloggerID"]].drop_duplicates()

    if not df_window.empty:
        ub_interactions_window = (
            df_window.groupby(["UserID", "BloggerID"])
            .size()
            .reset_index(name=f"ub_total_interactions_last_{window}d")
        )
        user_blogger_hist_agg = pd.merge(
            user_blogger_hist_agg,
            ub_interactions_window,
            on=["UserID", "BloggerID"],
            how="left",
        )
```

```
        for behaviour_type, behaviour_name in zip(
            [1, 2, 3, 4], ["view", "like", "comment", "follow"]
        ):
            ub_behaviour_count = (
                df_window[df_window["UserBehaviour"] == behaviour_type]
                .groupby(["UserID", "BloggerID"])
                .size()
                .reset_index(name=f"ub_{behaviour_name}_count_last_{window}d")
            )
            user_blogger_hist_agg = pd.merge(
                user_blogger_hist_agg,
                ub_behaviour_count,
                on=["UserID", "BloggerID"],
                how="left",
            )

        ub_last_interaction_date_window = (
            df_window.groupby(["UserID", "BloggerID"])["Date"]
            .max()
            .reset_index(name=f"ub_last_interaction_date_last_{window}d")
        )
        user_blogger_hist_agg = pd.merge(
            user_blogger_hist_agg,
            ub_last_interaction_date_window,
            on=["UserID", "BloggerID"],
            how="left",
        )
        user_blogger_hist_agg[f"ub_days_since_last_interaction_last_{window}d"] = (
            reference_date
            - user_blogger_hist_agg[f"ub_last_interaction_date_last_{window}d"]
        ).dt.days
        user_blogger_hist_agg = user_blogger_hist_agg.drop(
            f"ub_last_interaction_date_last_{window}d", axis=1
        )

        total_interactions_col = f"ub_total_interactions_last_{window}d"
        user_blogger_hist_agg[total_interactions_col] = user_blogger_hist_agg[
            total_interactions_col
        ].fillna(0)
        for behaviour_name in ["view", "like", "comment", "follow"]:
            col_name = f"ub_{behaviour_name}_count_last_{window}d"
            user_blogger_hist_agg[col_name] = user_blogger_hist_agg[
                col_name
            ].fillna(0)
        days_since_col = f"ub_days_since_last_interaction_last_{window}d"
        user_blogger_hist_agg[days_since_col] = user_blogger_hist_agg[
            days_since_col
        ].fillna(window)
```

```
    else:
        user_blogger_hist_agg[f"ub_total_interactions_last_{window}d"] = 0
        for behaviour_name in ["view", "like", "comment", "follow"]:
            user_blogger_hist_agg[f"ub_{behaviour_name}_count_last_{window}d"] = 0
        user_blogger_hist_agg[f"ub_days_since_last_interaction_last_{window}d"] = (
            window
        )

    gc.collect()

# 删除不再需要的原始日期列 (已经在计算 days_since 时使用了)
date_cols_to_drop = [
    "user_first_active_date",
    "user_last_active_date",
    "blogger_first_active_date",
    "blogger_last_active_date",
    "ub_first_interaction_date",
    "ub_last_interaction_date",
]
user_agg = user_agg.drop(
    columns=[col for col in date_cols_to_drop if col in user_agg.columns],
    errors="ignore",
)
blogger_agg = blogger_agg.drop(
    columns=[col for col in date_cols_to_drop if col in blogger_agg.columns],
    errors="ignore",
)
user_blogger_hist_agg = user_blogger_hist_agg.drop(
    columns=[
        col for col in date_cols_to_drop if col in user_blogger_hist_agg.columns
    ],
    errors="ignore",
)

gc.collect()

return user_agg, blogger_agg, user_blogger_hist_agg
```

附录 C　问题二、三 LightGBM 实现

```
# --- 2. 子任务 1: 预测用户 2024.7.21 是否在线 ---

print("\n--- Building Training and Validation Data for Online Prediction ---")
print(
    "Processing data for online prediction training (Predicting 7.20 based on
        7.11-7.19)..."
)
train_pred_date_online = pd.to_datetime("2024-07-20")
```

```
train_hist_end_online = train_pred_date_online - timedelta(days=1) # 7.19
user_features_train_online_base, _, _ = create_historical_features(
    df_hist, train_hist_end_online
)
# 找到在 7.20 活跃的用户 (训练标签)
active_on_train_date_online = df_hist[df_hist["Date"] == train_pred_date_online][
    "UserID"
].unique()
# 获取所有在历史数据中出现过的用户 + 7.20 活跃的用户，确保标签集包含所有可能的用户
all_users_for_train_label = np.union1d(
    user_features_train_online_base["UserID"].unique(), active_on_train_date_online
)
user_features_train_online = pd.DataFrame({"UserID": all_users_for_train_label})
user_features_train_online = pd.merge(
    user_features_train_online,
    user_features_train_online_base,
    on="UserID",
    how="left",
)  # Left merge to keep all users in label set
user_features_train_online["is_online"] = (
    user_features_train_online["UserID"].isin(active_on_train_date_online).astype(int)
)

# 验证集: 预测 7.19 是否在线，特征基于 7.11-7.18
print(
    "Processing data for online prediction validation (Predicting 7.19 based on
        7.11-7.18)..."
)
val_pred_date_online = pd.to_datetime("2024-07-19")
val_hist_end_online = val_pred_date_online - timedelta(days=1) # 7.18
user_features_val_online_base, _, _ = create_historical_features(
    df_hist, val_hist_end_online
)
# 找到在 7.19 活跃的用户 (验证标签)
active_on_val_date_online = df_hist[df_hist["Date"] == val_pred_date_online][
    "UserID"
].unique()
# 获取所有在历史数据中出现过的用户 + 7.19 活跃的用户，确保标签集包含所有可能的用户
all_users_for_val_label = np.union1d(
    user_features_val_online_base["UserID"].unique(), active_on_val_date_online
)
user_features_val_online = pd.DataFrame({"UserID": all_users_for_val_label})
user_features_val_online = pd.merge(
    user_features_val_online, user_features_val_online_base, on="UserID", how="left"
)  # Left merge to keep all users in label set
user_features_val_online["is_online"] = (
    user_features_val_online["UserID"].isin(active_on_val_date_online).astype(int)
)
gc.collect()
```

```
# --- 修正点：区分不同类型的特征进行填充 ---
# 在线预测数据的填充（训练集和验证集）
print(
    "Filling missing values for online training and validation data (distinguishing
        feature types)..."
)

large_days_value = 11

# 填充训练数据
print("Filling NaNs for online training data...")
# 找到所有 days_since_last_active 相关的列名
user_days_since_cols_train = [
    col for col in user_features_train_online.columns if "days_since_last_active" in col
]
# 对 days_since 列使用较大的值填充
for col in user_days_since_cols_train:
    user_features_train_online[col] = user_features_train_online[col].fillna(
        large_days_value
    )
# 对其他数值列使用 0 填充
numeric_cols_train = user_features_train_online.select_dtypes(include=np.number).columns
other_numeric_cols_train = numeric_cols_train.difference(
    user_days_since_cols_train
).difference(["is_online"])
user_features_train_online[other_numeric_cols_train] = user_features_train_online[
    other_numeric_cols_train
].fillna(0)

# 填充验证数据
print("Filling NaNs for online validation data...")
# 找到所有 days_since_last_active 相关的列名
user_days_since_cols_val = [
    col for col in user_features_val_online.columns if "days_since_last_active" in col
]
# 对 days_since 列使用较大的值填充
for col in user_days_since_cols_val:
    user_features_val_online[col] = user_features_val_online[col].fillna(
        large_days_value
    )
# 对其他数值列使用 0 填充
numeric_cols_val = user_features_val_online.select_dtypes(include=np.number).columns
other_numeric_cols_val = numeric_cols_val.difference(
    user_days_since_cols_val
).difference(["is_online"])
user_features_val_online[other_numeric_cols_val] = user_features_val_online[
    other_numeric_cols_val
].fillna(0)
```

```
X_train_online = user_features_train_online.drop(columns=["UserID", "is_online"])
y_train_online = user_features_train_online["is_online"]
X_val_online = user_features_val_online.drop(columns=["UserID", "is_online"])
y_val_online = user_features_val_online["is_online"]

X_val_online = X_val_online[X_train_online.columns]

print(
    f"Online prediction training data shape: {X_train_online.shape}, labels shape:
        {y_train_online.shape}"
)
print(
    f"Online prediction validation data shape: {X_val_online.shape}, labels shape:
        {y_val_online.shape}"
)
gc.collect()
```

附录 D　问题四互动时段预测实现

```
if not predicted_online_users_p4:
    print("No target users predicted to be online on 7.23. Result table contains empty
        values.")
else:
    df_online_hist = df_hist[df_hist['UserID'].isin(predicted_online_users_p4)].copy()

    if df_online_hist.empty:
        print("Predicted online users have no historical data (7.11-7.20). Cannot
            predict time slots and bloggers.")
    else:
        print("Analyzing historical behavior for predicted online users...")

        # Step 3.1: 获取每个用户 Top3 小时段（按行为频率）
        hourly_counts = df_online_hist.groupby(['UserID',
            'Hour']).size().reset_index(name='count')
        hourly_counts['prob'] =
            hourly_counts.groupby('UserID')['count'].transform(lambda x: x / x.sum())
        top_hours_online_users = hourly_counts.groupby('UserID',
            group_keys=False).apply(lambda g: g.nlargest(3,
            'prob')).reset_index(drop=True)
        top_hours_online_users = top_hours_online_users[['UserID', 'Hour']]
        print(f"Identified Top 3 hours for {top_hours_online_users['UserID'].nunique()}
            predicted online users.")

        # Step 3.2: 统计每个 用户-小时-博主 的互动数（点赞+评论+关注）
        interaction_behaviors_p4 = [2, 3, 4]
        interaction_hist_online = df_online_hist[df_online_hist['UserBehaviour']
        .isin(interaction_behaviors_p4)].copy()
```

```
if not interaction_hist_online.empty:
    ub_hourly_interactions_online = interaction_hist_online.groupby(['UserID',
        'Hour', 'BloggerID']).size().reset_index(name='interaction_count')
else:
    print("No interaction data (like, comment, follow) for predicted online
        users in history. Cannot predict bloggers.")
    ub_hourly_interactions_online = pd.DataFrame(columns=['UserID', 'Hour',
        'BloggerID', 'interaction_count'])
print(f"Calculated interaction counts for {len(ub_hourly_interactions_online)}
    predicted online user-hour-blogger pairs.")

merged_interactions_online = pd.merge(
    top_hours_online_users,
    ub_hourly_interactions_online,
    on=['UserID', 'Hour'],
    how='left'
)
top_bloggers_per_hour_online = merged_interactions_online.groupby(['UserID',
    'Hour'], group_keys=False).apply(
    lambda g: g.nlargest(1, 'interaction_count', keep='first')
).reset_index(drop=True)

print("Populating result table for predicted online users...")

user_hour_blogger_dict = {}
if not top_bloggers_per_hour_online.empty:
    for user_id, hour, blogger_id, interaction_count in
        top_bloggers_per_hour_online[['UserID', 'Hour', 'BloggerID',
        'interaction_count']].itertuples(index=False):
        if user_id not in user_hour_blogger_dict:
            user_hour_blogger_dict[user_id] = []
        user_hour_blogger_dict[user_id].append((hour, blogger_id,
            interaction_count))

    for user_id in user_hour_blogger_dict:
        user_hour_blogger_dict[user_id].sort(key=lambda x: x[0]) # Sort by hour

    for user_id, results_list in user_hour_blogger_dict.items():
        if user_id in result_table_p4.index:
            for i in range(min(3, len(results_list))):
                hour, blogger_id, interaction_count = results_list[i]

                if pd.notna(blogger_id) and pd.notna(interaction_count) and
                    interaction_count > 0:
                    # Use .loc with the user_id LABEL (index)
                    result_table_p4.loc[user_id, f'博主ID {i+1}'] =
                        str(blogger_id) # Ensure it's a string
                    result_table_p4.loc[user_id, f'时段{i+1}'] =
                        f"{int(hour):02}:00-{int(hour)+1:02}:00" # Format hour
```

```
                            with leading zero
                else:
                    print(f"Warning: User {user_id} has prediction results but is not in
                        the target user list {target_users_p4}. Skipping.") # Should not
                        happen if logic is correct

        else:
            print("No top bloggers found for any predicted online user based on
                historical interactions.")
```