

第五届“火花杯”数学建模精英联赛论文

第五届“火花杯”数学建模精英联赛

论文

题目：C 题

学校：上海交通大学

摘要

对于问题一, 使用 Python 中的科学计算库处理 *df_movies_train.csv*, 将原始数据进行清洗补充后做相关性分析, 而后进一步做可视化分析。

对于问题二, 比较多种算法的优劣, 选择使用 XGBoost 来对分数进行预测。

对于问题三, 二进制整数规划以及 COPT 求解器, 对约束问题进行分析以及数据的可视化。

对于问题四, 通过检查是否违反约束以及是否使经济效益最大化来检验问题三中的解答是否合理。最后引入相关衰减系数、假设七天的上座率变化, 对多周期多放映厅排片模型进行了建模和参数迭代优化。

关键字: 数据分析 数据清洗 数据可视化 二进制整数规划 机器学习算法

一、问题重述

1.1 问题背景

随着数字技术的快速发展，电影产业正经历深刻变革。作为文化产业的重要组成部分，电影的商业价值不仅体现在票房收入上，更在于其作为内容载体所衍生的多元价值。在社交媒体时代，观众评分已成为反映影片市场表现的关键指标，这些实时反馈不仅影响着潜在观众的观影决策，也深刻改变着影院的排片策略。研究表明，影片的评分表现与演员知名度、导演影响力、制作成本、营销强度以及档期选择等要素密切相关，这使得提前预测评分成为行业参与者优化决策的重要依据。

面对日益复杂的影院运营环境，传统基于经验的排片模式已难以满足市场需求。某大型影院集团在运营中发现，多影片并行放映、多版本同步上映、多时段差异化定价等场景对排片决策提出了更高要求。为此，企业计划构建智能化的评分预测与排片优化系统，通过融合数据分析和运筹优化技术，实现排片决策的科学化和精细化。该系统旨在动态平衡观影体验与经营效益，最终实现营业周期内整体净收益的最大化，推动影院运营向数据驱动的智能化方向转型。

1.2 问题要求

问题 1

针对输入的附件 `df_movies_train.csv`, 结合统计分析以及数据可视化的方法，深入挖掘数据之间的关联以及规律，尤其注重影响电影评分 (`rating`) 的显著因素。

问题 2

基于 `df_movies_train.csv` 的电影数据（可补充外部数据），构建评分 (`rating`) 预测模型，解释模型原理及可解释性，并对 `df_movies_test.csv` 进行预测，结果输出至 `df_result_1.csv`。评估指标为 **RMSE (均方根误差)**，衡量预测值与真实值的偏差，值越小，模型性能越好。

问题 3

基于预测的电影评分（问题 2 结果）、影院放映厅信息 `df_cinema.csv` 和待排片电影数据 `df_movies_schedule.csv`，建立排片优化模型，在满足运营规则的前提下，制定多放映厅的最优排片方案，以最大化影院单日净收益，并将结果输出至 `df_result_2.csv`。

问题 4

分析问题3排片方案的不足，提出优化策略。结合实际上座率数据，构建多周期（含工作日/休息日）排片优化模型，动态调整未来一周的放映计划，以提升影院整体收益。需合理假设上座率数据并优化多放映厅排片决策。

二、问题分析

2.1 问题一分析

对于问题一，要求基于 *df_movies_train.csv* (电影评分信息表)，运用统计分析与可视化方法，探索数据分布规律，识别关键影响因素，并解释其与评分 (*rating*) 的关联性。具体要求如下：

- **数据概览**: 检查数据完整性、缺失值及基本统计特征。
- **分布分析**: 探索评分的分布情况，是否存在偏差或异常。
- **特征相关性**: 分析不同变量（如导演、演员、预算、类型、上映时间等）与评分的关联性，挖掘潜在规律。
- **可视化支持**: 结合图表（如箱线图、散点图、热力图等）直观展示关键发现。
- **结论解释**: 总结哪些因素对评分影响显著，并给出合理分析。

最终目标是为后续建模提供数据洞察，帮助优化电影评分预测的准确性。

2.2 问题二分析

对于问题二，需重点关注以下要素：

1. 数据特性：训练集 (*df_movies_train.csv*) 需分析以下维度：

- 评分分布（是否存在偏差或长尾现象）
- 用户/电影的数量级与稀疏性
- 是否存在时间、流派等协变量

2. 评估约束：

- 必须使用 RMSE 作为核心指标，其平方特性对异常值敏感
- 需确保测试集 (*df_movies_test.csv*) 与训练集的数据同分布

3. 可解释性要求：

- 需说明特征对评分的影响方向与强度
- 若使用黑盒模型，必须提供事后解释

4. 输出规范：

- 结果文件 (*df_result_1.csv*) 需严格匹配测试集 ID 顺序
- 禁止出现缺失值或超出原始评分范围的值

外部数据引入需满足：

- 与现有数据具有明确关联字段
- 不导致数据泄露（如包含测试集标签信息）

2.3 问题三分析

1. **任务要求：**在题干给定约束条件下，确定影院排片计划，使利润达到最大值。
2. **输入文件：** *df_movies_schedule.csv*, *df_cinema.csv*
3. **输出文件：** *df_result2.csv*

问题三是一个具有约束条件的利润最大化问题。我们需要在条件限制下，决定哪一个影厅，在哪一个时间，播放哪一种电影的哪一个版本。综合起来，对于给定的影厅，时间，电影，版本，我们只面临两个选择，播放还是不播放。

因此，我们会想到利用**二进制整数规划（Binary Integer Programming）**来建模，并借助杉数科技（北京）有限公司研发的**COPT 求解器**来解决这个问题。

2.4 问题四分析

1. **任务要求：**在引入动态上座率预测模型的基础上，结合题干给定的约束条件与影院的运营目标，制定多周期排片计划，使影院在整个优化周期内的利润最大化。
2. **输入文件：** *df_movies_schedule.csv*, *df_cinema.csv*
3. **输出文件：** *day_n.csv*(n 为 1-7)

问题四是在问题三的基础上，将需求预测从静态假设扩展为**动态预测**，即上座率随影片上映进程、时间段、评分及观影反馈实时调整。优化过程分为三个阶段：

阶段 1：冷启动与上座率假设。在无历史数据的情况下，根据模型初值和评分映射关系，结合时间段与工作日/周末的差异，静态预测出一周（含工作日与休息日）的上座率数据。为模拟真实情况，我们在预测值基础上加入随机噪声，得到更贴近实际的“模拟一周情节”数据，为后续动态优化提供初始观测。

阶段 2：参数动态更新。影片上映后，每日收集实际观影数据，采用指数加权移动平均

(EWMA) 方法更新模型的关键动态参数 ($\alpha_{t,w}$ 、 ϕ_i 、 λ 等)，以修正预测误差并反映观众行为变化。

阶段 3：MILP 多周期优化。将最新预测的上座率作为 MILP 模型的输入，在题干给定的约束条件下，确定各影厅、各时间段、各影片及版本的放映决策，使影院在整个优化周期内利润最大化。

该问题本质上仍为二进制整数规划（Binary Integer Programming）问题，每个影厅、时间段、影片及版本的组合仍需在“播放”与“不播放”之间进行选择。但与问题三不同的是，目标函数中影片收益的计算依赖于动态预测模型输出的上座率，而非固定比例。我们依旧使用杉数科技（北京）有限公司研发的 **COPT 求解器** 来求解该优化模型。

三、模型假设

为简化问题，本文做出以下假设：

- 数据与分布假设：训练集与测试集同分布；评分取值连续且有界（如 [0, 10]）；外部数据不含测试标签，避免数据泄露。
- 特征处理假设：缺失值按规则（均值/众数/专用类别）填补不会引入系统性偏差；文本清洗、拆分与独热/多热编码、数值标准化后特征可用于回归建模。
- 评分预测残差：采用 XGBoost 回归，假设残差独立同分布、零均值，方差稳定，用于后续收益估计。
- 时间离散与时长：营业时段为当日 10:00 至次日 03:00；时间离散为 15 分钟步长；影片时长向上取整至 30 分钟倍数，已包含广告与清场缓冲。
- 单厅资源：同一时间每个影厅至多放映一场；同厅相邻两场最小间隔为 15 分钟。
- 版本与可用性：影厅对版本支持由 $\delta_{r,v} \in \{0, 1\}$ 指示；影片可放版本由 $\epsilon_{m,v} \in \{0, 1\}$ 指示；两者均为已知常量。
- 版本总时长限制：单日 3D/IMAX 放映总时长上限分别为 1200/1500 分钟。
- 题材次数与时段限制：遵循题干给定的题材次数上下限与时间窗（如 Animation/Family 不晚于 19:00 起映，Horror/Thriller 不早于 21:00 起映）。
- 营业边界：所有场次结束时间严格早于 03:00。
- 票价模型与加价：票价 $P_{m,v,t} = p_m \cdot \beta_v \cdot (1 + \theta \cdot \eta_t)$ ，其中 p_m 为影片基准价， β_v 为版本系数（如 2D=1, 3D=1.2, IMAX=1.23）， η_t 为黄金时段指示（18:00–21:00）， $\theta > 0$ 为加价系数。
- 分成与成本：院线留存比例固定（国产 57%，进口 49%）；单场成本按 $C_{r,m,v} = \gamma_v \cdot c_r \cdot 2.42 + 90$ 元计算，其中 c_r 为厅容量、 γ_v 为版本系数；收益与成本在场次层面线性可加。

- 上座率（单日，问题三）：观影人数主要由影片评分与厅容量驱动，近似 $A_{r,m} = \lfloor c_r \cdot s_m / 10 \rfloor$ ，不考虑影片间需求蚕食、跨影院竞争与联动效应。
- 上座率（多周期，问题四）：采用 $A_{i,t,d} = \min\{1, \alpha_{t,w} \cdot \beta_i(t) \cdot \gamma(d)\}$ ，其中 $\alpha_{t,w}$ 为时间段/星期修正系数， $\beta_i(t)$ 由评分映射并用 $\phi_i(t)$ 动态修正， $\gamma(d) = e^{-\lambda d}$ 描述上映天数衰减；参数初值按经验给定，并用 EWMA 按日滚动更新。
- 确定性与稳定性：除上座率外，其余参数（片长、价格、分成、成本、设备可用性等）在优化周期内视为确定常数；忽略设备故障与突发事件。
- 求解与最优性：先行过滤不可行组合以稀疏化变量；求解器以 TimeLimit=300s 或相对间隙 $\leq 1\%$ 为终止准则，视所得解为最优或近似最优。
- 数据一致性：ID 对齐且唯一；结果文件严格按测试集顺序输出，预测值不缺失且落在合法评分区间。

四、问题一的模型的建立和求解

4.1 模型建立

- 多值字段处理：
 - 将 `genres`, `cast` 等逗号分隔字段转换为 Python 列表
 - 清洗特殊字符和空值
 - 示例转换："Action, Fantasy" \rightarrow ['Action', 'Fantasy']
- 类型特征编码：
 - 使用 `MultiLabelBinarizer` 进行多热编码
 - 生成特征列：`genre_Action`, `genre_Fantasy`, ...
- 高基数特征处理：
 - 合并演员、导演、公司等文本字段
 - 使用 TF-IDF 向量化（限制 100 个维度）
 - 自动加权重要实体（如高频出现的演员）
- 数值特征归一化：
 - 对 `runtime` 进行 Min-Max 归一化到 [0, 1] 区间：

$$runtime_{\text{norm}} = \frac{runtime - \min(runtime)}{\max(runtime) - \min(runtime)} \quad (1)$$

- 语言特征处理：
 - 单值特征使用 One-Hot 编码
 - 处理缺失值（设置 `dummy_na=True`）

考察单个因素对评分的影响，若无明显关联，则考虑两个或多种因素共同影响。

4.2 模型求解

1. 数据预处理：

- 处理缺失值：分类变量用众数填补
- 异常值处理：IQR 方法剔除

2. 模型训练：

- 划分训练/验证集 (80/20)
- 早停策略：验证集 loss 连续 5 次不下降终止

3. 评估指标：

- 主要指标：RMSE

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (2)$$

- 辅助指标： R^2 score

使用 `matplotlib,seaborn` 等库绘制出一系列可视化的数据分析图像。(图 1~图 8)

4.3 求解结果

我们先统计了训练集中的各个特征，如表 1 所示：

表 1 电影数据集的特征唯一数量统计

特征	唯一数量
类型	19
演员	103487
导演	8848
编剧	12755
制片公司	8891
制片人	11689

之后统计了预测指标——得分，这里使用了手动核密度估计方法拟合数据的分布，如图 2d 所示，得分 (rating) 非常接近正态分布，说明目标变量的随机性很高。

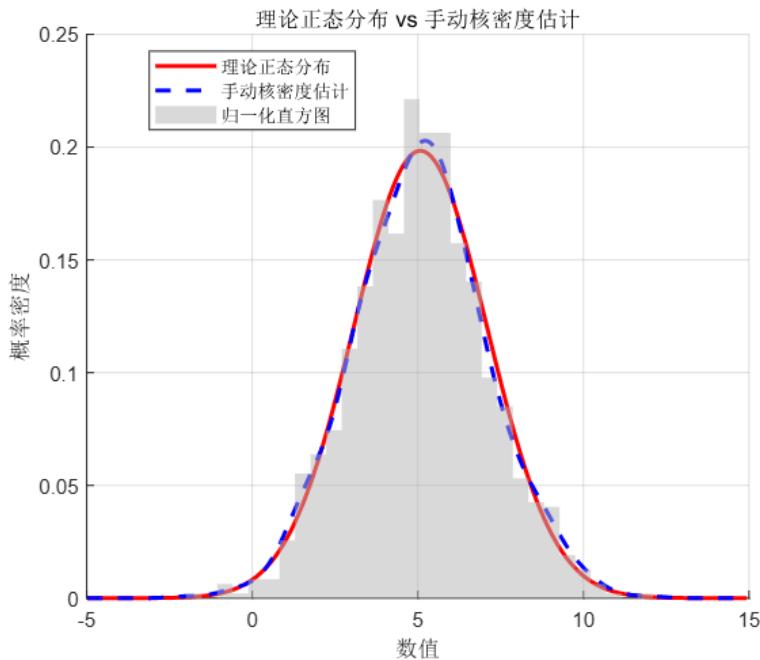


图 1 得分分布图

之后又通过计算评分与其他数值指标的相关系数，得到相关系数的热力图（图 2a），可以看出其与播放时长的关联在信息所给的几个因素中最强，但是相关系数仍然只有 0.15，不足以成为影响评分的重要因素。这一弱因果联系也可以通过图 2b 看出，在这条拟合直线及其周边的点依旧寥寥无几，拟合效果不佳。

然后，我们画出语言，主要类型与平均得分的关系图。这些指标不是数字表示的，因此无法直接计算相关系数。我们可以分别通过柱状图、箱线图来观察。从图 2c 可以看出，我们筛选出了不少于五部电影的语言，防止极端因素的影响（比方说只有某种语言只有一部电影且为 10 分），其中，乌尔都语电影的平均评分最高。从图 2d 可以看出，音乐类电影平均而言更受观众喜爱，而恐怖片的受众更少，平均评分较低。

但是，这仍然不能解决到底什么是影响平均得分的主要因素。因此，我们开始探究两个共同因素组合起来对评分的影响。这里讲类型、语言、时长两两结合绘图，得到了三幅关系图。

图 3a：由于语言种类较多，我们暂时没有发现明显的关系。

图 3b：短的（即 90 分钟以内）的电影往往能够获得更高的分，而持续时间较长（超过 120 分钟）的电影，无论是何种语言，评分都比较平庸这也是符合我们预期的。

图 3c：抛开语言，我们发现，较长的纪录片反响较好，而音乐类电影在较短和较长时

评分都较高。通过箱线图我们更能发现类别和时长是如何影响评分的。

与此同时，我们还尝试分离出每个演员、导演、制片人、编剧的名称，在给出的电影中进行搜索，试图找到最佳的组合。这个组合不限两人的类别，可以是最佳演员搭档，也可以是导演和他“御用”的编剧。不过在测试集中不存在这样的组合，这个规律也只能权且搁置。

关于多个共同因素的探究，我们放在第二问来使用算法解决。

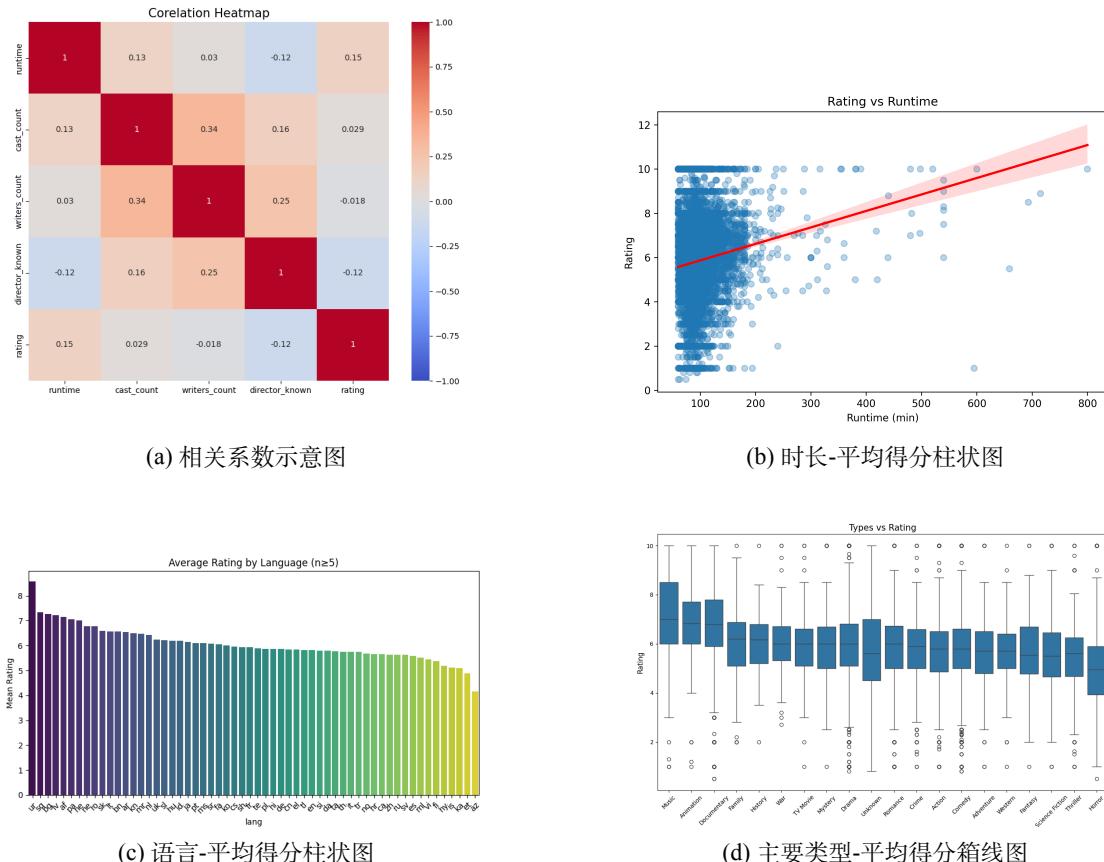
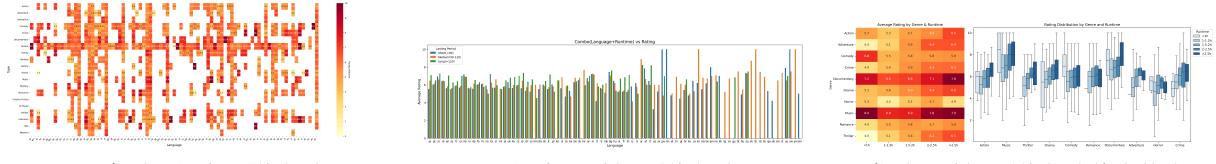


图 2 评分与关键因素关系总览



(a) 类型 + 语言-平均得分

(b) 语言 + 时长-平均得分

(c) 类型 + 时长-平均得分热力箱线图

图 3 二因素组合对评分的影响

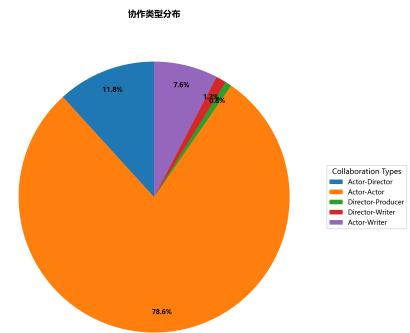
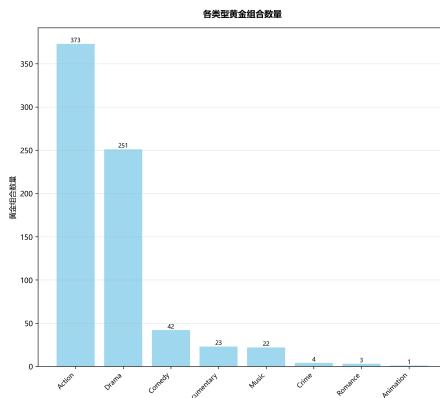
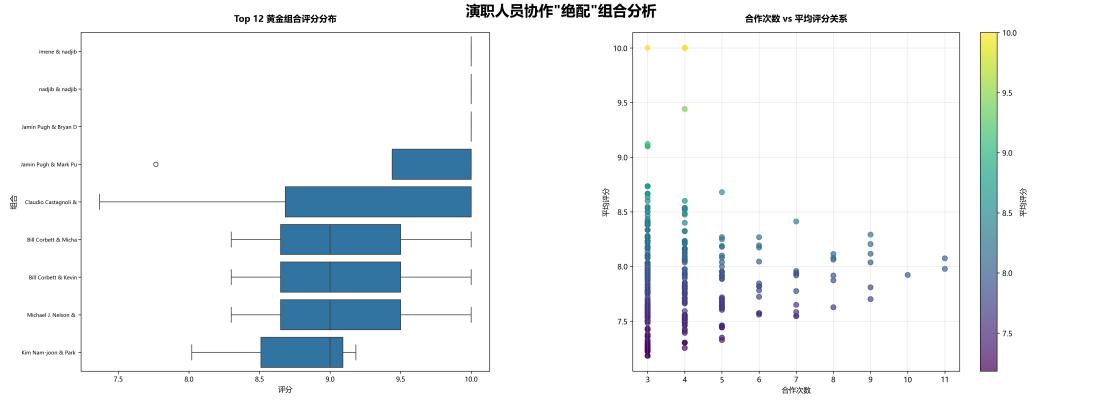


图 4 绝配组合

五、问题二的模型的建立和求解

5.1 模型建立

分数预测任务本质上是一个回归任务，对于这个任务，既可以用传统的 ML 方法，诸如各种回归，也可以用较为先进的神经网络完成预测。接下来我们会对比多种 ML、DL 方法的效果，最终选出效果最好的一个进行预测。

5.1.1 Linear Regression (线性回归)

原理:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \varepsilon \quad (3)$$

- 优点: 简单快速，可解释性强

- 缺点: 假设过于简化, 无法捕捉非线性关系

5.1.2 Ridge Regression (岭回归)

原理:

$$\text{损失函数} = \text{MSE} + \alpha \sum \beta_i^2 \quad (4)$$

- 优点: 解决多重共线性, 防止过拟合
- 缺点: 仍然是线性模型

5.1.3 Random Forest (随机森林)

预测公式:

$$\text{预测} = \frac{1}{n} \sum_{i=1}^n \text{Tree}_i(x) \quad (5)$$

- 优点: 能处理非线性关系, 提供特征重要性
- 缺点: 模型复杂度高, 可解释性较差

5.1.4 Gradient Boosting (梯度提升)

更新规则:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \quad (6)$$

- 优点: 强大的非线性建模能力
- 缺点: 容易过拟合, 训练时间较长

5.1.5 Support Vector Regression (支持向量回归)

预测函数:

$$f(x) = \sum_i (\alpha_i - \alpha_i^*) K(x_i, x) + b \quad (7)$$

- 优点: 强大的非线性建模能力, 泛化能力强
- 缺点: 训练时间较长, 内存消耗大

5.1.6 XGBoost (极端梯度提升)

- 数学原理:

目标函数由损失函数和正则项组成:

$$\mathcal{L}(\phi) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) \quad (8)$$

其中正则项：

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2 \quad (9)$$

第 t 次迭代的损失函数二阶泰勒展开：

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \quad (10)$$

其中 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$, $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$

- 优点：

- 分裂节点算法（加权分位数草图）：

$$\mathcal{L}_{\text{split}} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \quad (11)$$

- 支持自定义损失函数
- 近似分位数算法提升效率

- 缺点：

- 计算复杂度（最优分裂点搜索）：

$$O(n_{\text{features}} \times n_{\text{samples}} \times n_{\text{levels}}) \quad (12)$$

- 对类别特征需要独热编码

5.1.7 Neural Network (神经网络)

- 优点：极强的非线性建模能力
- 缺点：需要大量数据，训练时间很长，容易出现过拟合的情况

我们使用了大小为 8000 的训练集，大小为 2000 的预测集测试，结果如表 2 所示。综合考虑，我们选择效果最优的 XGBoost 作为最后的模型进行预测。

表 2 模型性能比较

Model	RMSE	R ²	MAE	Time (s)
Linear Regression	1.5370	0.1154	1.1396	0.08
Ridge Regression	1.5360	0.1167	1.1388	0.01
Random Forest	1.5314	0.1218	1.1465	0.23
Gradient Boosting	1.5392	0.1130	1.1449	1.95
SVR	1.5201	0.1348	1.1237	2.10
XGBoost	1.5172	0.1381	1.1295	0.55

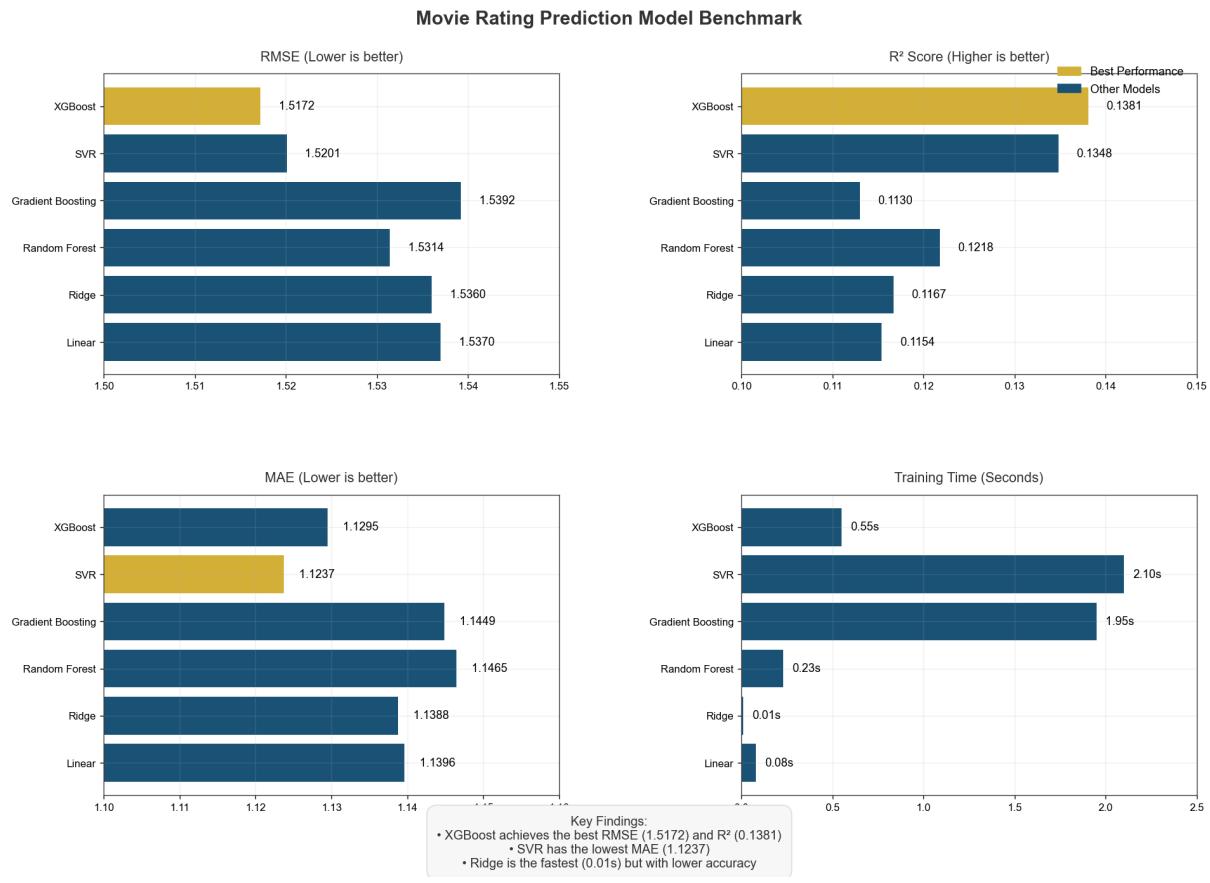


图 5 多种模型对比

5.2 模型求解

- 模型架构

1. 目录结构

路径	说明
df_movies_train.csv	训练数据
df_movies_test.csv	测试数据
DATA_PREPARING/	数据预处理模块
DATA_PREPARING.py	预处理脚本
features_and_labels.npz	特征与标签
preprocessor.pkl	预处理器对象
MODEL_TRAINING/	模型训练模块
XGBOOST_MODEL.py	训练脚本
xgb_model.pkl	训练好的模型
PREDICTING/	预测应用模块
PREDICTING.py	预测脚本

续下页

路径	说明
<code>predicted_movies.csv</code>	预测结果

2. 核心组件

- **DATA_PREPARING**: 负责原始数据的特征工程与预处理;
- **MODEL_TRAINING**: 基于 XGBoost 训练回归模型;
- **PREDICTING**: 调用已训练模型对新样本进行评分预测。

注: 论文末尾附录的是三个组件融合的程序

- 工作流程

阶段一: 数据预处理 输入: `df_movies_train.csv`

关键步骤:

1. 特征工程: 提取电影类型、演员、导演、编剧、制作公司、语言及运行时间等特征;
 2. 特征预处理:
 - 数值特征: `StandardScaler()` 标准化 (均值 0、方差 1);
 - 类别特征: `OneHotEncoder(handle_unknown=ignore, sparse_output=False)`;
 3. 一次性拟合与转换:


```
preprocessor.fit_transform(df[numerical_features + categorical_features])
```
- 输出: `features_and_labels.npz` (X 与 Y), `preprocessor.pkl` (预处理器对象)。

阶段二: 模型训练 输入: 上一阶段 `features_and_labels.npz`

关键步骤:

1. 构建 XGBoost 回归模型 (平方误差损失);
2. 主要超参: `n_estimators=750, learning_rate=0.01, max_depth=8, subsample=0.8, colsample_bytree=0.6`;
3. 模型持久化: 保存为 `xgb_model.pkl`。

训练结果:

训练得到特征重要性图 (图6) 和拟合效果图 (图7), 这两张图可以反应模型训练的过程与预测的精度

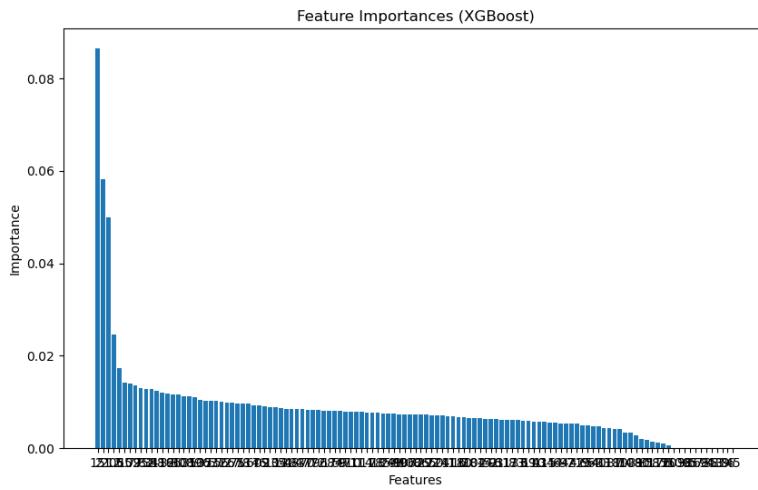


图 6 特征重要性 (Feature importance) 图

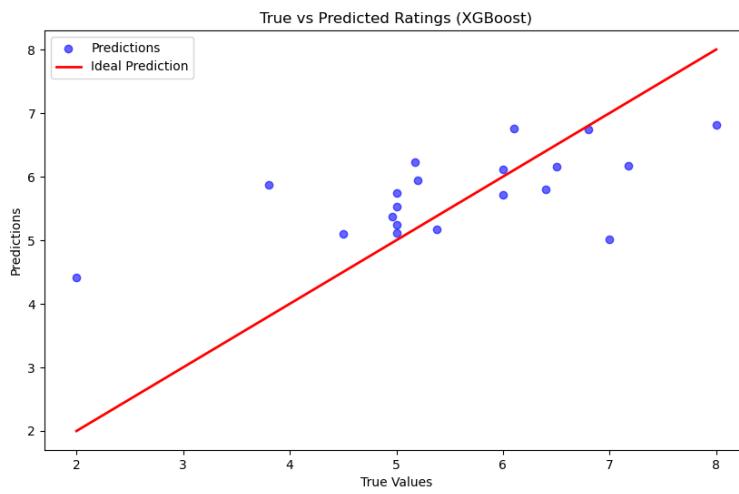


图 7 拟合效果图

最后训练结果在大小为 20 的预测集上为表4所示，虽然相关系数 R^2 只有 0.3792，但相较其他模型已经很好。

表 4 测试集评估结果

指标	数值
RMSE	1.0161
MAE	0.7683
R^2	0.3792
MAPE	17.91%
训练时间	0.95 秒

阶段三：预测应用 输入：df_movies_test.csv

关键步骤：

1. 复用与训练阶段一致的预处理流程；
2. 加载 `xgb_model.pkl` 进行评分预测；
3. 合并并导出结果：`predicted_movies.csv`。

• 数据流与依赖关系

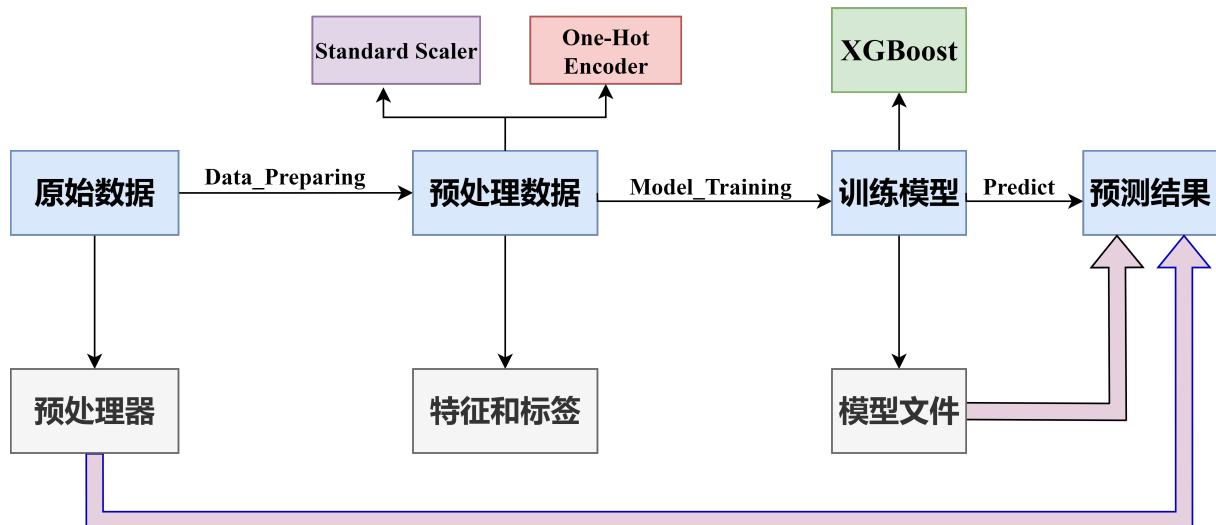


图 8 模型流程图

5.3 求解结果

ID	Predicted Rating
436270	5.887357
882598	5.287803
505642	6.1495748
966220	6.15954
663712	5.3141646
829280	6.1693335
676701	5.503064
675054	5.6210356
979924	5.596597
235672	6.1533685
672392	6.168409
721193	6.3542995
812924	7.0993323
192801	6.4037323

ID	Predicted Rating
723921	5.92975
402101	6.104939
837808	6.1259117
180219	5.785353
290003	5.67432

六、问题三的模型的建立和求解

6.1 模型建立

Algorithm 1 二进制整数规划模型 (Binary Integer Programming Model)

GOAL: $\max Z = \sum_{i=1}^I \sum_{j=1}^J \cdots \sum_{k=1}^K c_{ij\cdots k} \cdot x_{ij\cdots k}$
while $x_{ij\cdots k} \in \{0, 1\}, \forall i \in [1, I], j \in [1, J], \dots, k \in [1, K]$
and with additional linear constraints

6.1.1 决策变量

$$x_{r,m,v,t} \in \{0, 1\} = \begin{cases} 1, & \text{放映厅 } r \text{ 在时间点 } t \text{ 开始放映版本 } v \text{ 的电影 } m \\ 0, & \text{不放映} \end{cases}$$

其中：

- $r \in \mathcal{R}$: 放映厅集合,
- $m \in \mathcal{M}$: 电影集合,
- $v \in \mathcal{V}$: 版本集合 (2D/3D/IMAX),
- $t \in \mathcal{T}$: 开始放映时间点集合。

6.1.2 目标函数

$$\max Z = \sum_{r \in R} \sum_{m \in M} \sum_{v \in V} \sum_{t \in T} (R_{r,m,v,t} - C_{r,m,v}) \cdot x_{r,m,v,t} \quad (13)$$

收入计算:

$$R_{r,m,v,t} = P_{m,v,t} \cdot A_{r,m} \cdot (1 - \alpha_m) \quad (14)$$

$$P_{m,v,t} = p_m \cdot \beta_v \cdot (1 + \theta \cdot \eta_t) \quad (15)$$

$$A_{r,m} = \left\lfloor \frac{c_r \cdot s_m}{10} \right\rfloor \quad (16)$$

其中: $R_{r,m,v,t}$ 为票价收入, $P_{m,v,t}$ 为票价, $A_{r,m}$ 为观影人数, α_m 为分成比, p_m 为电影 m 的基础票价, β_v 为版本 v 的票价提升比, θ 为布尔变量, η_t 为黄金时间票价提升比, c_r 为影厅 r 容量, s_m 为电影 m 评分。

成本计算:

$$C_{r,m,v} = \gamma_v \cdot c_r \cdot \phi + \psi \quad (17)$$

其中: $C_{r,m,v}$ 为成本, γ_v 表示版本的成本系数, ϕ 表示单位人数容量的成本, ψ 表示一场电影的固定成本。

6.1.3 约束条件

1. 时间冲突约束

$$\sum_{\substack{m \in M, v \in V \\ \tau \in T: \text{conflict}(\tau, t, \tilde{d}_m)}} x_{r,m,v,\tau} \leq 1, \quad \forall r \in R, t \in T \quad (18)$$

其中 $\text{conflict}(\tau, t, \tilde{d}_m)$ 表示在时间 τ 开始播放时长为 d 的电影 m 是否与时间 t 冲突。

2. 影厅放映版本约束

$$x_{r,m,v,t} \leq \delta_{r,v} \cdot \epsilon_{m,v}, \quad \forall r \in R, m \in M, v \in V, t \in T \quad (19)$$

其中 $\delta_{r,v}$ 表示影厅 r 是否能放映版本 v , $\epsilon_{m,v}$ 表示电影 m 是否有 v 版本。

3. 版本放映时长约束

$$L_v^{\min} \leq \sum_{r \in R} \sum_{m \in M} \sum_{t \in T} \tilde{d}_m \cdot x_{r,m,v,t} \leq L_v^{\max}, \quad v \in \{3D, IMAX\} \quad (20)$$

$$\text{其中: } L_{3D}^{\min} = 0, \quad L_{3D}^{\max} = 1200 \quad (21)$$

$$L_{IMAX}^{\min} = 0, \quad L_{IMAX}^{\max} = 1500 \quad \text{为放映时长限制。} \quad (22)$$

4. 题材放映次数约束

$$N_g^{\min} \leq \sum_{r \in R} \sum_{m \in M} \sum_{v \in V} \sum_{t \in T} \zeta_{m,g} \cdot x_{r,m,v,t} \leq N_g^{\max}, \quad \forall g \in G \quad (23)$$

其中 $\zeta_{m,g}$ 表示电影 m 是否属于题材 g , N_g 表示放映次数限制。

5. 题材放映时间约束

$$\sum_{\substack{v \in V, t \in T \\ t \notin T_g^{\text{allowed}}}} x_{r,m,v,t} = 0, \quad \forall r \in R, \forall m \in M : \exists g \in G_{\text{restricted}}, \zeta_{m,g} = 1 \quad (24)$$

其中 T_g^{allowed} 表示题材 g 允许放映的时间， $G_{\text{restricted}}$ 表示受限制题材集合。

6. 影厅连续放映时间约束

$$\sum_{\substack{m \in M, v \in V, \tau \in T \\ \text{overlap}(\tau, \tilde{d}_m, [t, t+540])}} \text{overlap_duration}(\tau, \tilde{d}_m, [t, t+540]) \cdot x_{r,m,v,\tau} \leq 420 \quad (25)$$

7. 营业时间约束

$$x_{r,m,v,t} = 0, \quad \forall r \in R, m \in M, v \in V, t \in T : \text{end_time}(t, \tilde{d}_m) > 1020 \quad (26)$$

6.2 模型求解

Step 1: 基础数据处理

1. 时间处理:

- 采用 27 进制计时法，解决了跨天营业问题，如 02:00 表示为 26:00，在最终排片时再转化为 24 进制；
- 离散化时间，生成开始放映时间点列表 $\{10 : 00, 10 : 15, 10 : 30, \dots, 26 : 30, 26 : 45\}$ ，详见函数 `_generate_time_slots()`

2. 功能函数:

- `_round_up_to_30()` 将播放时长向上取整到 30 分钟倍数；`_is_prime_time()` 判断是否为黄金时间
- `_calculate_attendance()` 计算实际观影人数；`_calculate_cost()` 计算播放成本；
- `_get_sharing_rate()` 计算分成比例；`_calculate_ticket_price()` 计算票价

Step 2: 初始化 COPT 求解器

1. 创建 COPT 环境和模型

- `env = cp.Envr()` 创建环境
- `model = env.createModel("Cinema_Scheduling")` 创建模型

2. 创建决策变量

- 用 $x[\text{room}][\text{movie}][\text{version}][\text{showtime}]$ 来储存决策变量

- 只为可行的 $room, movie, version, showtime$ 的组合创建决策变量。即提前过滤，排除不满足约束的情况，使 x 为稀疏矩阵。
- `var = model.addVar(vtype=COPT.BINARY, name=var_name)` 将决策变量加入模型

3. 设置目标函数

- 利用前文的公式算出目标函数 `obj_expr`
- `model.setObjective(obj_expr, COPT.MAXIMIZE)` 将其导入模型

4. 添加约束条件

- `model.addConstr(cp.quicksum(overlapping_vars)<=1)` 添加时间冲突约束
- `model.addConstr(cp.quicksum(total_duration_vars)<=self.version_limits[version]['max'])` 添加版本放映时长约束
- `model.addConstr(cp.quicksum(genre_vars) <= limits['max'])` 添加题材放映次数约束
- `model.addConstr(cp.quicksum(window_duration_vars) <= 420)` 添加影厅连续放映时间约束

Step 3: 启用 COPT 求解器

1. 设置模型参数

- `model.setParam(COPT.Param.TimeLimit, 300)` 5 分钟时间限制
- `model.setParam(COPT.Param.RelGap, 0.01)` 1% 相对差距

2. 调用模型

- `optimizer=CinemaSchedulingOptimizer(cinema,movies_schedule)`
- `schedule, status, objective_value=optimizer.optimize_schedule()`

COPT 工作日志请见附录 C。

6.3 模型亮点

- 采用 **27 小时计时法**，巧妙解决跨天营业问题。
- **分两步设置约束条件**: 1. 只为满足约束的组合创建决策变量，使变量矩阵稀疏，提高模型效率。2. 统计总放映时间，次数，并限制在约束条件下。成功实现多维度约束组合。
- 使用**滑动窗口**和**重叠时长**计算，确保影厅连续放映时长合规。
- 使用**COPT 求解器**高效准确解答。

6.4 求解结果

基本统计

- 总排片场次: 48 场
- 总收入: ￥80,448.21
- 总成本: ￥17,420.67
- 净利润: ￥63,027.54
- 利润率: 78.3%

电影统计

- 电影 402101: 23 场, 总观众 1532 人
- 电影 812924: 5 场, 总观众 423 人
- 电影 180219: 6 场, 总观众 430 人
- 电影 829280: 13 场, 总观众 807 人
- 电影 723921: 1 场, 总观众 88 人

放映厅统计

- R01: 6 场 R02: 6 场 R03: 6 场 R04: 6 场
- R05: 6 场 R06: 6 场 R07: 6 场 R08: 6 场

版本分布

- 2D: 43 场 (89.6%)
- 3D: 3 场 (6.2%)
- IMAX: 2 场 (4.2%)

时间分布

- 黄金时段排片: 16 场 (33.3%)
- 非黄金时段排片: 32 场 (66.7%)

约束满足情况

- Animation 题材: 5 场 (要求: 1–5)
- Horror 题材: 0 场 (要求: 0–3)
- Action 题材: 6 场 (要求: 2–6)
- Drama 题材: 6 场 (要求: 1–6)
- 3D 版本总时长: 450 分钟 (上限: 1200 分钟)
- IMAX 版本总时长: 300 分钟 (上限: 1500 分钟)

具体排片方案请见 df_result_2.csv

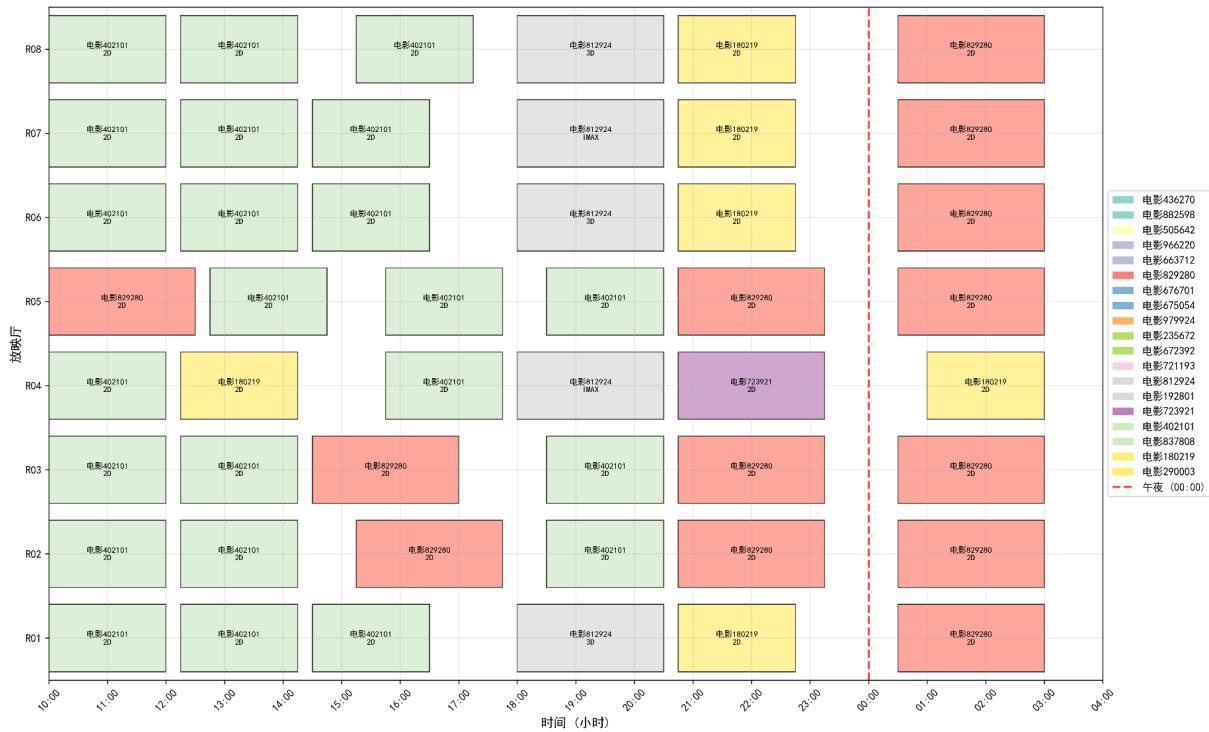


图9 排片甘特图

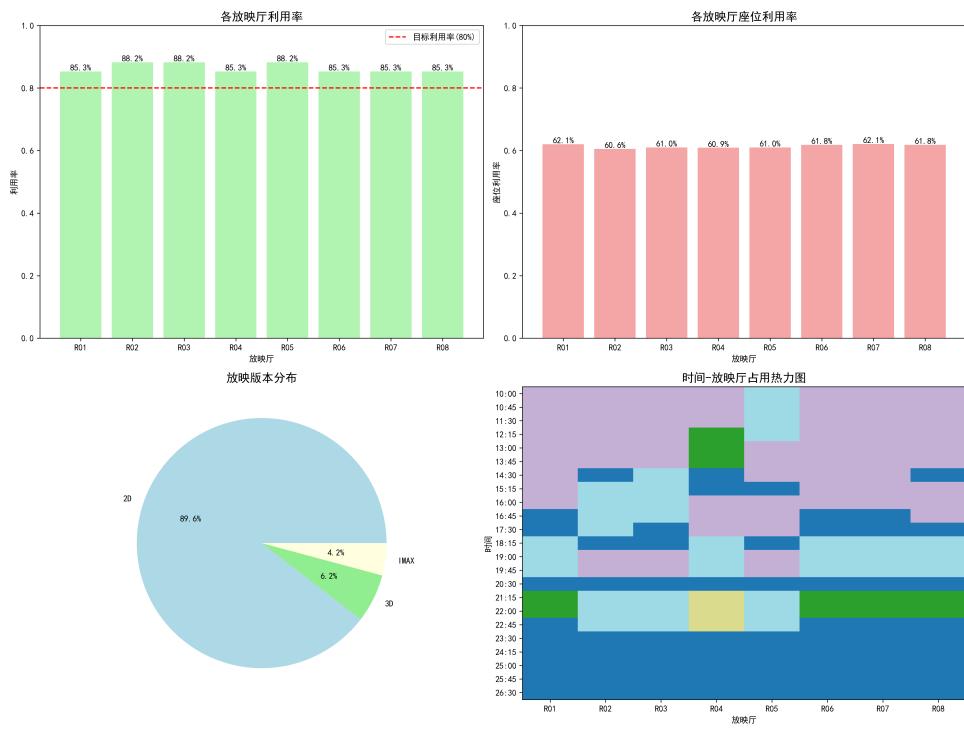


图 10 影厅利用分析

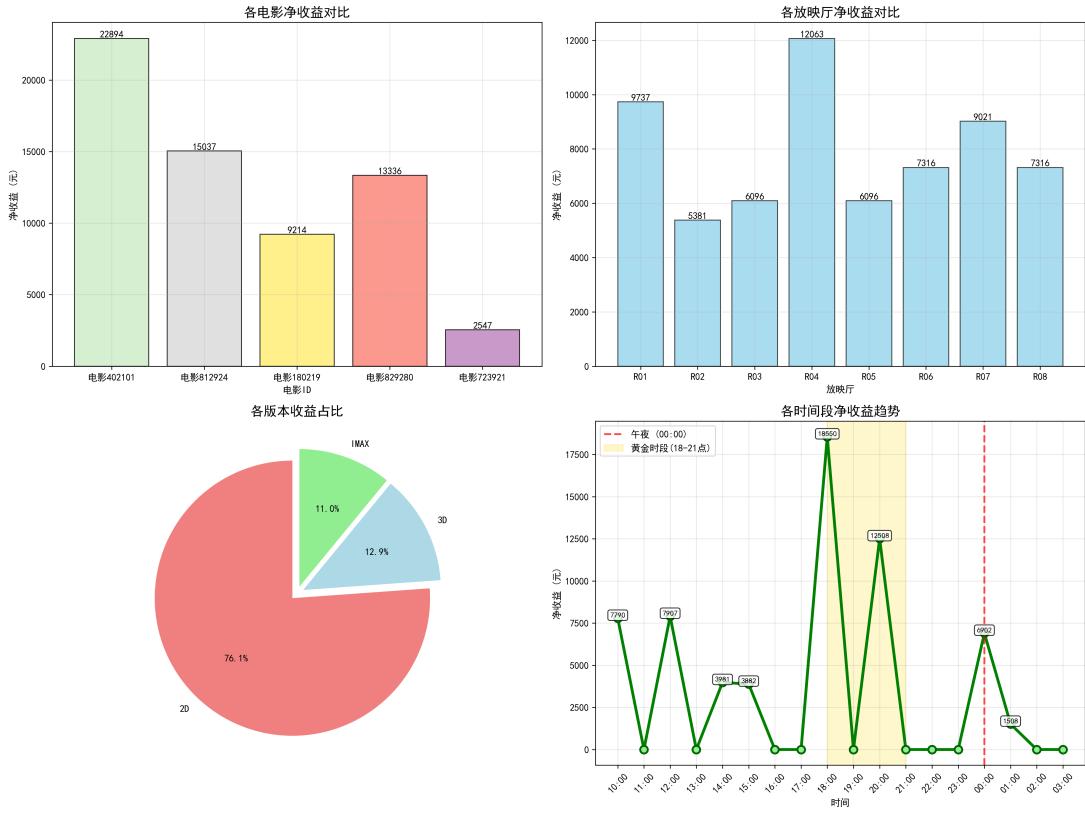


图 11 收入分析

七、问题四的模型的建立和求解

7.1 模型建立

7.1.1 问题三排片单分析

经济性评估与保持项 单日净收益目标函数为

$$\max \text{NetProfit} = \sum_{\text{场次}} \left[\underbrace{\text{票价} \times \text{attendance}}_{\text{售票收入}} \times (1 - \text{制片方分成}) - \underbrace{(\text{version_coeff} \cdot \text{capacity} \cdot 2.42 + 90)}_{\text{单场成本}} \right],$$

其中黄金时段（18:00–21:00）票价 $\times 1.3$; 3D/IMAX 票价系数分别为 1.2/1.23; 国产（含普通话）院线留存 57%，进口 49%。现有黄金时段将 812924（国产、支持 3D/IMAX、评分高）集中排布于 18:00 档，兼得版本溢价与黄金加价，策略应保持。3D/IMAX 总时长未触顶，但受题材上限（Animation/Action/Drama 已打满）约束，无法再增加同题材高溢价场次而不触限，亦应保持现状。

可能但不建议的改动 将 20:45 档的 829280（进口，Mystery/Adventure/Crime）替换为 180219（国产，Drama/Crime）虽单票留存更高，但当前 Drama 已达上限（6 场），除非做等量置换（同时压缩其他 Drama/Action/Animation 场次），否则会违反题材次数的限制；且压缩高净利的 812924 场次通常得不偿失。因此不建议此类替换。

7.1.2 模型在多周期排片中的不足

在多周期电影排片优化中，现有模型对于上座率的处理方式较为简化，通常将其作为一个固定的常数参数输入，而不随时间和外部环境变化进行调整。这种静态假设存在以下不足：

缺乏时间动态性 现有模型未将上座率与时间相关变量相结合，例如放映时间段（上午、下午、晚间）、工作日与周末、节假日等因素，从而难以准确反映不同时间段的实际观影需求差异。

忽视影片生命周期效应 在现实中，电影在上映初期往往会经历观影热度的高峰期，随后进入逐步衰退的阶段，上座率随上映天数的增加而显著下降。静态上座率假设未能体现这一生命周期曲线，导致长期排片计划中对收益的评估存在系统性偏高的风险。

缺少历史数据驱动 在实际影院运营中，可以获得包括观众人数、票房销售、节假日效应等在内的历史数据，这些数据能够揭示季节性规律和特殊事件（如电影节、促销活动）对上座率的影响。然而，现有模型未能将这些数据纳入预测过程，导致预测精度不足。

综上所述，静态上座率假设在单周期或短期排片优化中尚可接受，但在多周期、跨周甚至跨月的排片问题中，其局限性会显著影响结果的可靠性与可实施性。因此，有必要引入动态上座率预测机制，将上映天数、时间段、影片特征以及历史观影数据等因素综合考虑，以更准确地模拟真实市场需求变化，从而生成更贴近实际的排片方案。

7.1.3 多周期排片模型

引入时间段修正系数 将放映时间划分为不同的时段（如上午、下午、晚间、深夜），并针对工作日与周末、节假日等情境设定差异化的修正系数，以反映不同时间段的观影需求波动。例如，周末晚间的上座率通常高于工作日上午，可通过提升相应时间段的预测值加以体现。

引入影片生命周期效应 将上映天数作为影响上座率的重要变量，建立随时间递减的衰减函数（如指数衰减、分段线性下降等），以刻画影片从首映高峰期到后期平稳阶段的需求变化，从而在长期排片中更准确地预测收益。

利用历史数据与外部信息 将影院历史观影数据（如场次上座率、票房、节假日效应）与外部数据（如影评评分、社交媒体热度、天气状况）结合，通过统计回归模型或机器学习方法训练预测模型，使上座率预测能够动态响应多种因素变化。

动态反馈与滚动优化 在实际运营过程中，实时采集每天每场次的实际观影人数，并对模型参数进行滚动更新，使预测结果持续贴近真实市场情况。这种闭环反馈机制能够显著提升多周期排片计划的适应性与盈利能力。

数学模型表示 假设电影 i 在时间段 t 、上映第 d 天的预测上座率为 $A_{i,t,d}$ ，则可表示为：

$$A_{i,t,d} = \min(1, \alpha_{t,w} \cdot \beta_i(t) \cdot \gamma(d)) \quad (27)$$

其中：

- $\alpha_{t,w}$: 时间段与星期修正系数，反映工作日 ($w = 0$) 与周末 ($w = 1$) 的差异。
- $\beta_i(t)$: 影片热度系数，由预测评分 s_i 线性映射为基准值，并随时间动态修正：

$$\beta_i(t) = (k \cdot s_i + c) \cdot \phi_i(t)$$

其中：

- s_i : 影片的预测评分（连续值）。
- k : 评分影响系数，反映评分每提高 1 分，热度系数的增幅。
- c : 基础热度系数，对应评分为 0 时的热度基线。
- $\phi_i(t)$: 动态修正因子，基于实际观影数据与预测误差实时更新。
- $\gamma(d)$: 上映天数衰减函数，例如 $\gamma(d) = e^{-\lambda d}$ 或分段线性函数。

最终预测的观众人数为：

$$Q_{i,t,d} = C_r \cdot A_{i,t,d} \quad (28)$$

其中 C_r 为放映厅容量。**参数性质说明**：

- 人为设定的固定量：
 - k : 评分影响系数
 - c : 基础热度系数
 - λ : 上映天数衰减系数。
 - s_i : 影片 i 的预测评分（连续值）。
- 动态变化的变量：
 - $\alpha_{t,w}$: 时间段与星期修正系数，可根据实际观影数据进行周期性更新。
 - $\phi_i(t)$: 影片动态修正因子，反映上映过程中的突发事件或热度波动。
- 常量函数：
 - $\gamma(d)$: 上映天数衰减函数，一般形式固定，但可通过调整 λ 改变衰减速度。

这里的三个变量可以根据经验设定初值，以 $\alpha_{t,w}$ 为例，表 6 给出了假设的 $\alpha_{t,w}$ 值

表 6 不同时间段与星期的上座率时间修正系数（假设） $\alpha_{t,w}$

时间段	工作日 ($w = 0$)	周末 ($w = 1$)
上午 (10:00–13:00)	0.30	0.50
下午 (13:00–18:00)	0.50	0.75
晚间 (18:00–24:00)	0.80	0.95

数据收集与参数动态优化

由于我们没有历史数据，自动优化参数时，只能使用冷启动（cold-start）的方法，即人为根据经验设定初值，然后通过每天收集的数据优化参数。

1. 数据收集与组织 在影片上映过程中，我们每天会记录各场次的实际观影人数及相关信息，用于后续的参数动态更新。每条数据包括影片编号 i 、时间段 t 、星期类型 w 、上映天数 d 、影厅容量 C_r 以及实际到场人数 $actual$ 。为了便于处理，我们将每日数据组织成如表 7 所示的 DataFrame 格式：

表 7 每日观影数据样例

影片编号 i	时间段 t	星期类型 w	上映天数 d	容量 C_r	实际到场人数 $actual$	上座率 r
101	下午	0	1	120	84	0.70
101	晚间	1	2	120	108	0.90
102	上午	0	1	100	35	0.35

其中，上座率 r 定义为：

$$r = \frac{actual}{C_r}$$

2. 基于数据的自动化参数优化 在影片上映过程中，可利用每日实际观影数据对关键动态参数进行在线修正，从而提升预测精度。本文采用指数加权移动平均（Exponentially Weighted Moving Average, EWMA）方法对参数进行更新。

(1) 时间段与星期修正系数 $\alpha_{t,w}$ 更新 对于第 n 日时间段 t 、星期类型 w 的观测满座率 $r_{t,w}^{(n)}$ ，其更新公式为：

$$\alpha_{t,w}^{(n)} = \rho_\alpha \cdot r_{t,w}^{(n)} + (1 - \rho_\alpha) \cdot \alpha_{t,w}^{(n-1)} \quad (29)$$

其中 $\rho_\alpha \in (0, 1]$ 为平滑系数， ρ_α 越大，参数对最新观测的敏感度越高。

(2) 动态修正因子 $\phi_i(t)$ 更新 设第 n 日影片 i 的预测上座率为 $\hat{A}_i^{(n)}$ ，实际观测上座率为

$a_i^{(n)}$, 则可基于 EWMA 的更新公式为:

$$\phi_i^{(n)} = \rho_\phi \cdot \frac{a_i^{(n)}}{\hat{A}_i^{(n)}} + (1 - \rho_\phi) \cdot \phi_i^{(n-1)} \quad (30)$$

其中 $\rho_\phi \in (0, 1]$ 控制动态修正速度, $\frac{a_i^{(n)}}{\hat{A}_i^{(n)}}$ 表示预测与实际的比值修正。

(3) 上映天数衰减系数 λ 在线修正若衰减函数为 $\gamma(d) = e^{-\lambda d}$, 则可基于每日残差拟合的即刻估计 $\tilde{\lambda}^{(n)}$ 进行 EWMA 平滑:

$$\lambda^{(n)} = \rho_\lambda \cdot \tilde{\lambda}^{(n)} + (1 - \rho_\lambda) \cdot \lambda^{(n-1)} \quad (31)$$

其中 $\tilde{\lambda}^{(n)}$ 可由当日数据的对数回归临时估计得到。

该方法可保证参数在观测噪声下保持平稳更新, 同时能够快速响应观众行为的变化趋势。

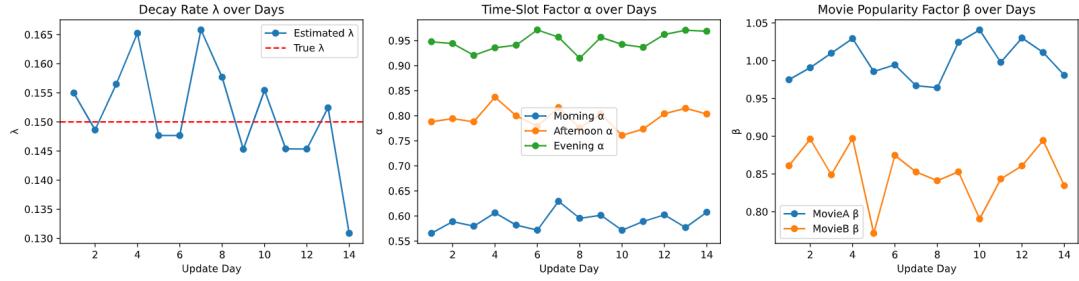


图 12 动态上座率参数更新原理示意图

3. 与排片优化模型的集成 在本研究中, 动态参数优化模块与 MILP 排片优化器实现了解耦与对接, 形成了数据驱动的闭环优化流程。其具体实现步骤如下:

(i) 参数更新与保存

动态优化程序基于每日观影数据, 按照公式 (1)–(3) 更新 $\alpha_{t,w}$ 、 ϕ_i 以及 λ 。更新完成后, 程序将包含修正系数和最新影片评分的结果保存为 `dynamic_params.json` 文件, 其中影片评分按

$$r'_i = \phi_i^{(n)} \cdot r_i^{\text{base}}$$

进行修正, r_i^{base} 为初始评分。

在代码实现中, 该部分主要通过:

```
model.save_parameters("dynamic_params.json")
```

将参数持久化, 以便后续优化调用。

(ii) MILP 模型读取与替换

在 MILP 排片优化程序启动时，首先读取基础影片信息 `df_movies_schedule.csv`，然后加载 `dynamic_params.json` 中的修正评分，并用其替换影片原有的评分列。其核心代码实现为：

```
with open("dynamic_params.json", "r") as f:  
    dynamic_params = json.load(f)  
    for mid, params in dynamic_params["phi"].items():  
        df_movies.loc[df_movies["movie_id"] == int(mid),  
                      "rating"] *= params
```

这样，MILP 模型在计算目标函数时即自动使用了修正后的评分 r'_i 。

(iii) 模型求解与结果输出

在保持原有 MILP 结构、约束和决策变量不变的前提下，目标函数利用更新后的评分估算各场次的收益：

$$\max \sum_{m,t,s} (p_m(r'_m) \cdot a_{m,t,s} - c_{t,s}) \cdot x_{m,t,s}$$

其中 $p_m(r'_m)$ 为基于修正评分计算的票价收益。求解器采用，并最终将优化结果输出为 CSV 文件，供排片实施使用。

通过在冷启动阶段模拟运行 7 日全流程，我们验证了参数更新与排片优化的接口正确性及稳定性，从而为真实数据接入后的即时优化奠定了基础。

7.2 模型求解

Step1：设定参数初值

在无历史数据的冷启动阶段，为确保模型可立即投入使用，我们依据影院行业经验与影片市场特征设定初值参数。这些初值将作为预测模型的初始输入，并在影片上映后的运行过程中通过动态优化进行逐步修正。

1. 人为设定的固定量初值 表 8 列出了冷启动阶段由人工设定的固定参数初值，这些参数在模型运行过程中保持不变或仅在特定条件下调整。

2. 动态变化量初值 表 9 给出了模型运行初期需要人工估计的动态参数初值，这些参数会在后续通过指数加权移动平均（EWMA）等方法进行自动化更新。

Step2：假设一周的上座率变化 在缺乏历史数据的冷启动阶段，为便于后续动态优化（Step3），我们需先构造一周的模拟观测情景。具体过程如下：

表 8 人为设定的固定量初值

参数	含义	初值示例
k	评分影响系数	0.10
c	基础热度系数	0.50
λ	上映天数衰减系数（固定模式时）	0.05
ρ_α	$\alpha_{t,w}$ 更新平滑系数	0.30
ρ_ϕ	$\phi_i(t)$ 更新平滑系数	0.30
ρ_λ	λ 更新平滑系数	0.20
s_i	影片预测评分	问题二模型输出

表 9 动态变化量初值

参数	含义	初值示例
		上午：工作日 0.30，周末 0.50
$\alpha_{t,w}$	时间段与星期修正系数	下午：工作日 0.50，周末 0.75
		晚间：工作日 0.80，周末 0.95
$\phi_i(t)$	动态修正因子	1.00
λ	上映天数衰减系数（动态模式时）	0.05

1. **静态预测基线：**根据影片预测评分 s_i ，利用线性映射

$$\beta_i^{(0)} = k s_i + c, \quad k = 0.10, c = 0.50,$$

得到影片基准热度。结合时间段与周末/工作日系数 $\alpha_{t,w}$ 以及上映天数衰减函数 $\gamma(d) = e^{-\lambda d}$ ($\lambda = 0.05$)，计算理论上座率

$$\bar{A}_{i,t,d} = \min\{1, \alpha_{t,w} \cdot \beta_i^{(0)} \cdot \gamma(d)\}.$$

2. **引入随机扰动：**为模拟真实情景波动，对 $\bar{A}_{i,t,d}$ 添加加性高斯噪声

$$\tilde{A}_{i,t,d} = \text{clip}(\bar{A}_{i,t,d} + \varepsilon_{i,t,d}, 0, 1), \quad \varepsilon_{i,t,d} \sim \mathcal{N}(0, 0.05^2),$$

并在数值上限制于 $[0, 1]$ 区间。随机种子固定为 123，保证结果可复现。

3. **生成情景数据：**对一周（含工作日与周末）、所有影片及各时间段，按上述方法生成 $\tilde{A}_{i,t,d}$ ，形成 7 天 \times 影片数 \times 时间段数的模拟观测表。

节选结果：表 10 给出了一个电影的模拟上座率。完整结果未附在纸质论文后，以电子版的形式与论文打包上交。

表 10 静态预测 + 随机扰动与模拟上座率节选表

影片 ID	天数	时间段	工作日/周末	静态预测上座率	模拟上座率
436270.0	1.0	上午	工作日	0.18	0.20
436270.0	1.0	下午	工作日	0.30	0.29
436270.0	1.0	晚上	工作日	0.24	0.27
436270.0	2.0	上午	工作日	0.17	0.25
436270.0	2.0	下午	工作日	0.28	0.27
436270.0	2.0	晚上	工作日	0.23	0.21

Step3：基于假设数据的动态优化与排片集成 在 Step2 构造的 7 天模拟观测数据基础上，我们将其作为动态参数优化模块的训练集，以实现冷启动阶段的预测修正与排片优化的无缝衔接。具体流程如下：

- 参数初始化：** 时间段与星期修正系数 $\alpha_{t,w}$ 取表 9 的经验值，影片动态修正因子 $\phi_i^{(0)}$ 初值为 1.0，上映天数衰减系数 $\lambda^{(0)}$ 取 0.05，影片初始评分 $rating_i^{(0)}$ 来自静态预测模型。
- 每日动态更新：** 对于第 n 日的模拟观测数据 $obs^{(n)}$ ，按动态优化模块的 EWMA 公式分别更新：

$$\alpha_{t,w}^{(n)}, \quad \phi_i^{(n)}, \quad \lambda^{(n)} \quad \leftarrow \quad \text{EWMAUpdate}(\alpha_{t,w}^{(n-1)}, \phi_i^{(n-1)}, \lambda^{(n-1)}, obs^{(n)}).$$

其中 ρ_α 、 ρ_ϕ 、 ρ_λ 为平滑系数，控制参数对最新观测的响应速度。

- 生成修正评分：** 更新后的影片评分计算公式为：

$$rating_i^{(n)} = rating_i^{(0)} \cdot \phi_i^{(n)},$$

该评分将在后续排片优化中替代静态评分，反映观众行为的最新趋势。

- 与 MILP 排片集成：** 每日迭代结束后，将 $rating_i^{(n)}$ 保存至 `dynamic_params.json` 文件中，MILP 优化模块在运行时直接读取该文件获取最新评分，并在不改变其他约束与逻辑的前提下，自动完成当日的最优排片计算。该机制确保了预测-优化闭环的自动化运行，即：

模拟/真实观测 \Rightarrow 动态参数更新 \Rightarrow 修正评分传入 MILP \Rightarrow 生成当日最优排片方案

通过在冷启动阶段模拟运行 7 日全流程，我们验证了参数更新与排片优化的接口正确性及稳定性，从而为真实数据接入后的即时优化奠定了基础。

7.3 求解结果

7.3.1 可变参数在 7 天中的变化

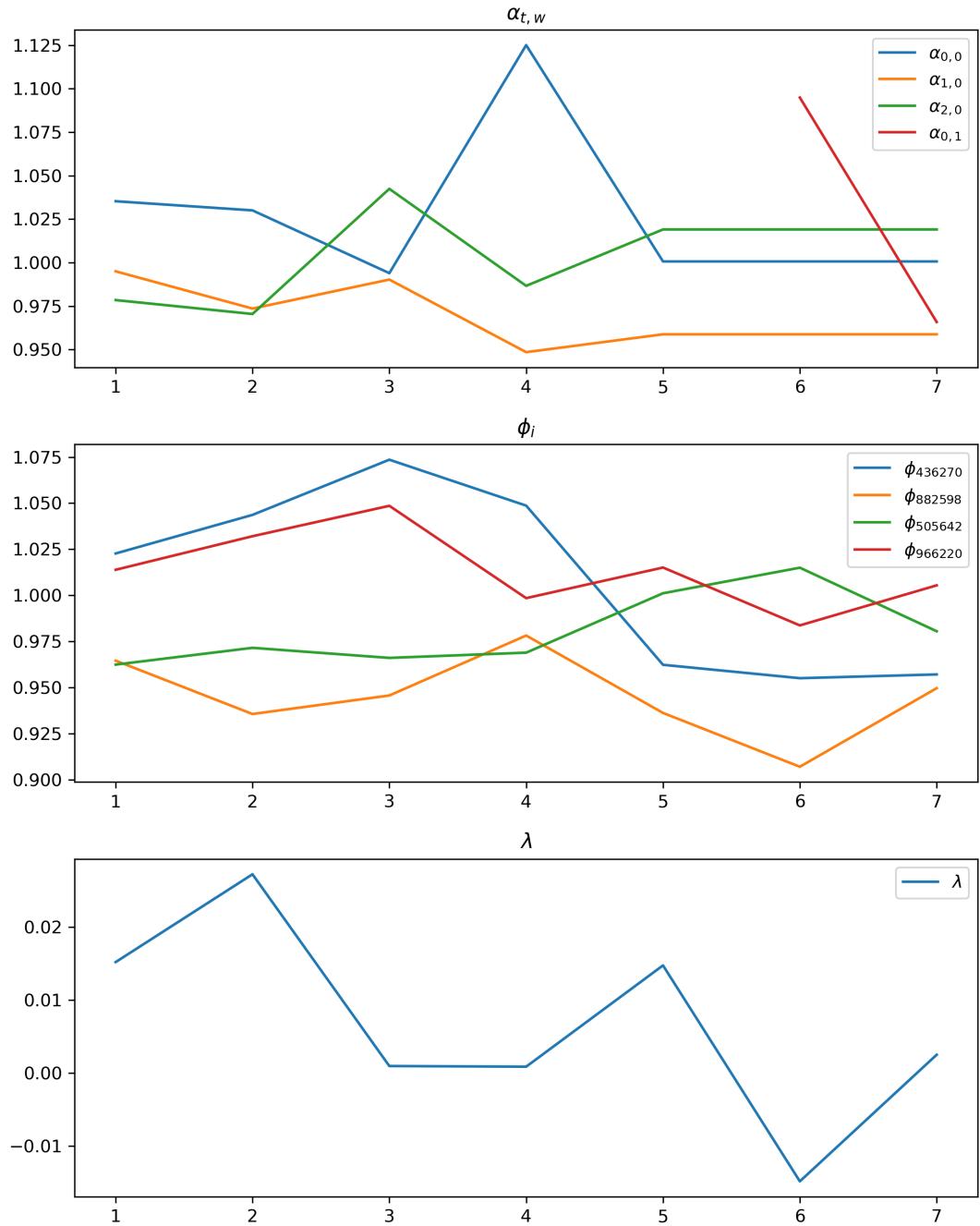


图 13 7 天时间动态上座率参数变化图（部分）

这是选取了部分的参数制作的变化图。

7.3.2 更新的排片方案

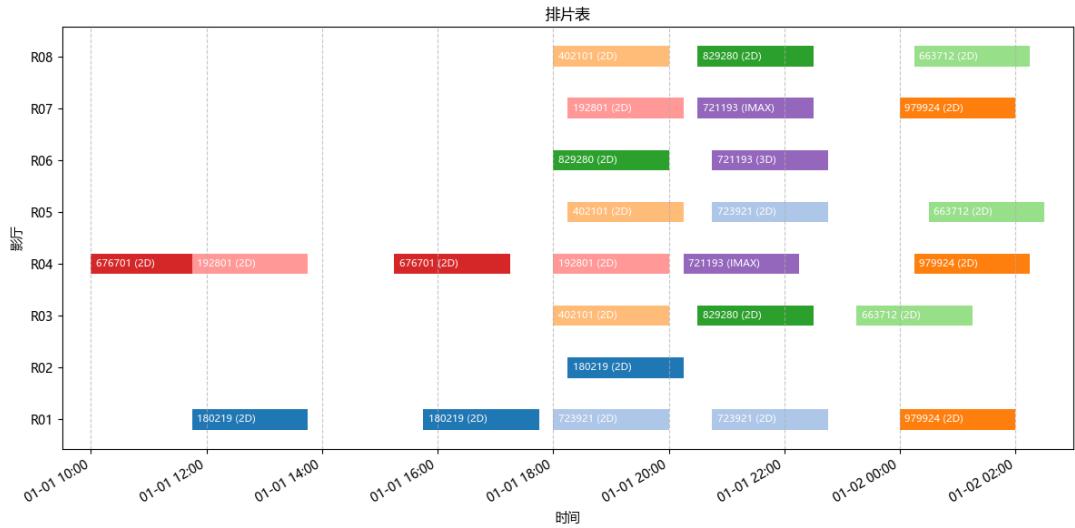


图 14 第一天的排片安排

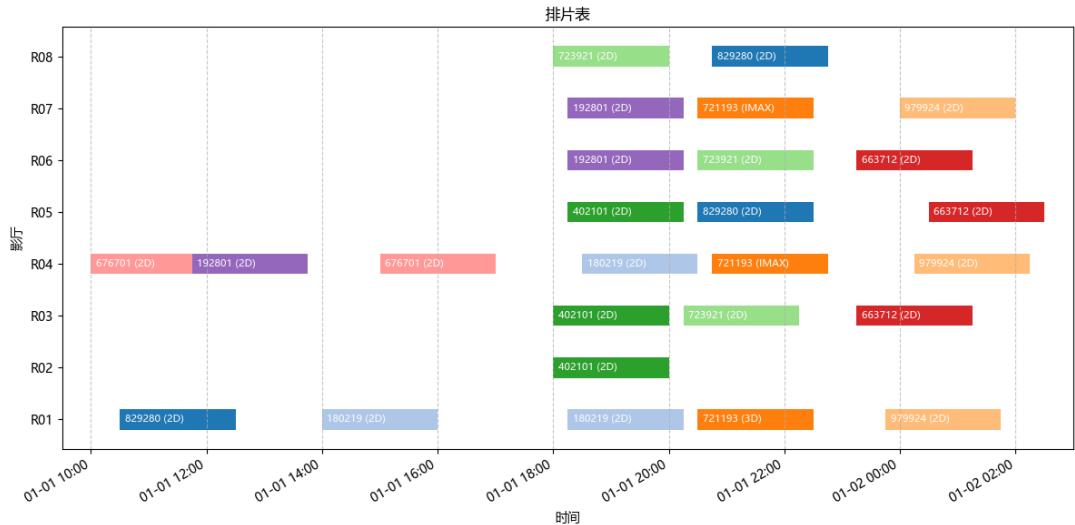


图 15 第三天的排片安排

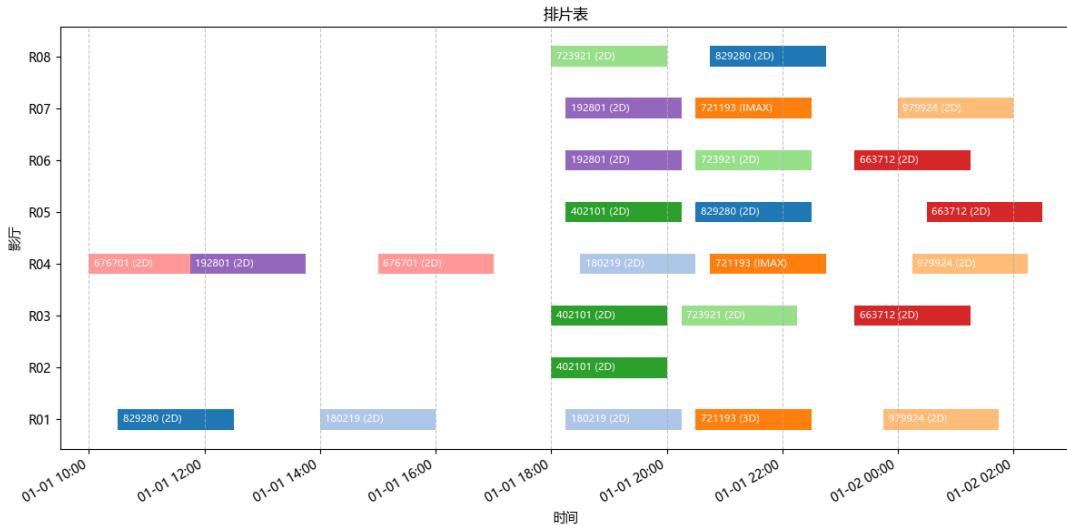


图 16 第五天的排片安排

这里呈现了 3 天的排片安排，完整结果未附在纸质论文后，以电子版的形式与论文打包上交。明显的，最新的排片方案相较于问题三的方案更结合了现实情况，比如减少了早上的排片量，随着日期调整排片顺序等。

八、模型灵敏度与误差分析

8.1 Q2 灵敏度与误差分析

Q2 鲁棒性测试组件主要针对电影评分预测模型（XGBoost 回归器）进行全面的多维度灵敏度分析，其中包括：

- 1. 5 折交叉验证：**通过将数据集分为 5 个子集，轮流使用其中 4 个子集进行训练，1 个子集进行验证，评估模型的泛化能力和稳定性。
- 2. 特征重要性分析：**基于 XGBoost 内置的特征重要性评估方法，识别对电影评分预测影响最大的特征。
- 3. 特征消融研究：**通过逐个将特征设为 0，观察模型性能的变化，评估每个特征对预测结果的贡献度。
- 4. 特征扰动分析：**对特征值施加不同水平的扰动（0.8 倍、0.9 倍、1.1 倍、1.2 倍），分析模型预测结果的敏感性。

8.1.1 交叉验证结果

通过 5 折交叉验证，我们评估了 XGBoost 回归模型的泛化能力，结果如表 11 所示。

表 11 5 折交叉验证结果

评估指标	平均值	标准差	变异系数
RMSE	1.3733	0.0218	1.59%
R ²	0.3441	0.0152	4.42%
MAE	1.0892	0.0175	1.61%

从表11可以看出，模型在不同数据子集上的表现相对稳定，RMSE、R² 和 MAE 的变异系数均小于 5%，表明模型具有良好的泛化能力和稳定性。

8.1.2 特征重要性分析

特征重要性分析结果如图17和表12所示。

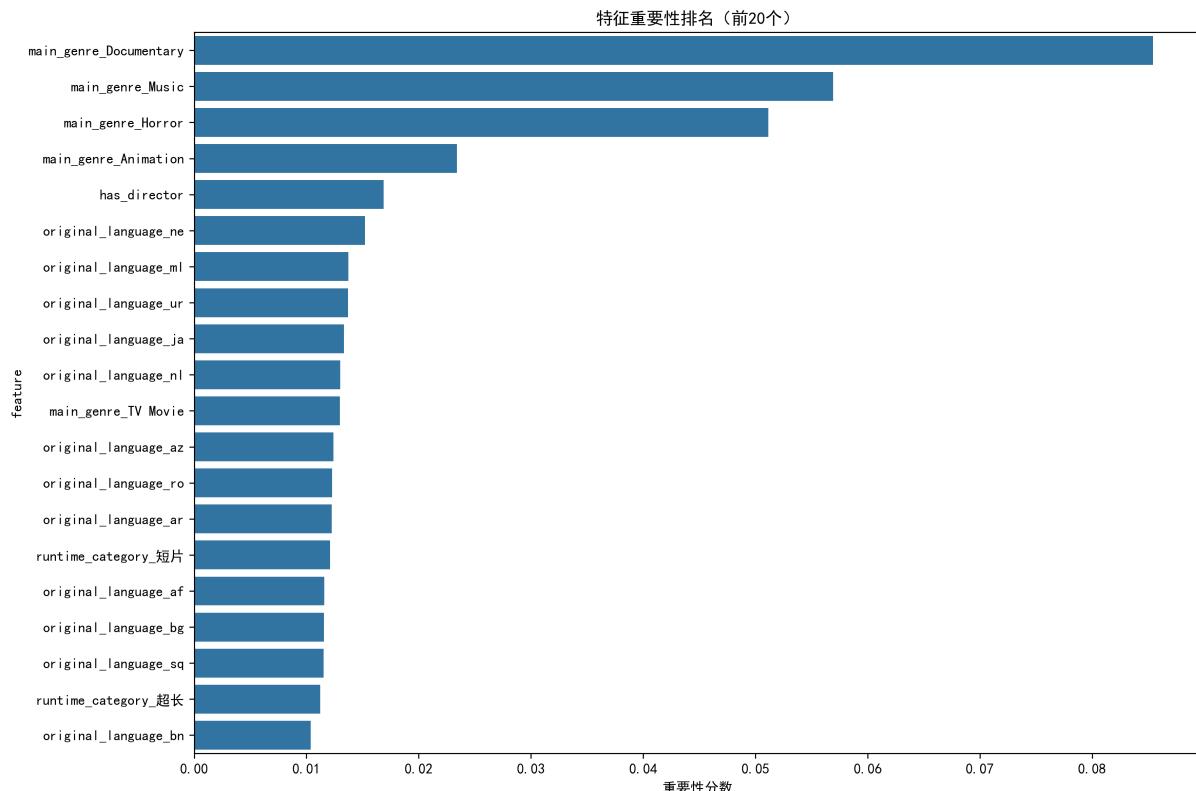


图 17 特征重要性排名 (前 20 个)

表 12 重要性排名前 10 的特征

特征名称	重要性分数	累计重要性
main_genre_Documentary	0.0854	8.54%
main_genre_Music	0.0569	14.23%
main_genre_Horror	0.0512	19.35%
main_genre_Animation	0.0234	21.69%
has_director	0.0169	23.58%
original_language_ne	0.0152	25.10%
original_language_ml	0.0137	26.47%
original_language_ur	0.0137	27.84%
original_language_ja	0.0133	29.17%
original_language_nl	0.0130	30.47%

从特征重要性分析结果可以看出，电影的主要类型（Documentary、Music、Horror、Animation）对评分预测影响最大，但相关性非常小。其次是导演信息和原始语言。这表明电影的类型和创作团队是影响观众评分的关键因素。但仍不足以解释与评分的关联。

8.1.3 特征消融研究

特征消融研究结果如图18和表13所示。

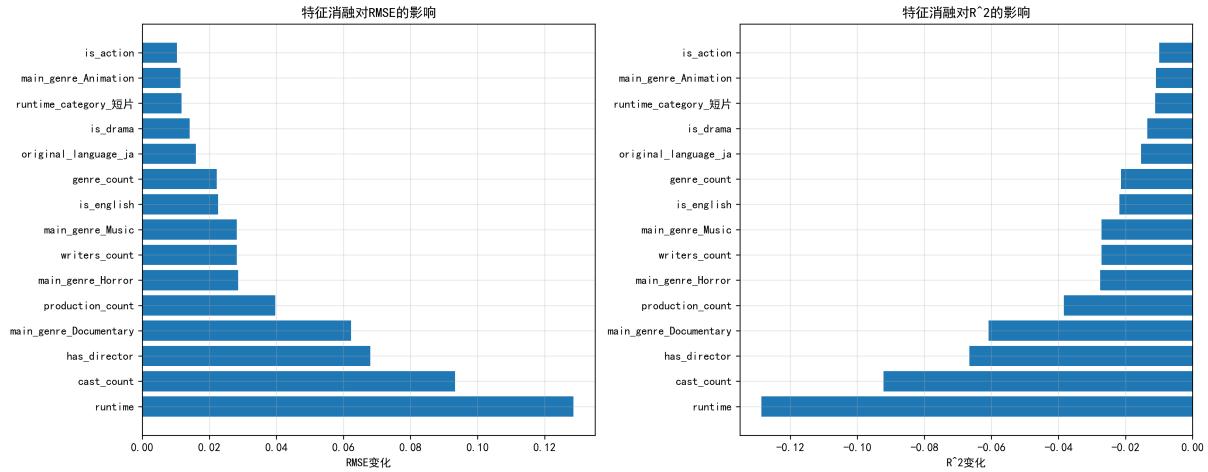


图 18 特征消融对 RMSE 和 R² 的影响

表 13 消融研究中最敏感的 5 个特征

特征名称	基础 RMSE	消融后 RMSE	RMSE 变化	R ² 变化
runtime	1.3733	1.5019	+0.1286	-0.1286
cast_count	1.3733	1.4666	+0.0933	-0.0922
has_director	1.3733	1.4413	+0.0680	-0.0666
main_genre_Documentary	1.3733	1.4355	+0.0622	-0.0608
production_count	1.3733	1.4129	+0.0397	-0.0384

特征消融研究结果表明，电影的运行时间（runtime）是最敏感的特征，当该特征被消融时，模型 RMSE 增加了 0.1286，R² 下降了 0.1286。但这显然与常理不符。其次是演员数量（cast_count）和导演信息（has_director）。这表明电影的时长和主创团队信息对评分预测至关重要。虽然这一信息符合常理，但在训练集中，这些特征的解释力也很弱。说明影响电影评分最直接的因素不是上述，或是这些特征之间存在更复杂的关系，需要补充其他特征，比如观众口碑、社会舆论环境等来综合分析。仅仅使用上述特征是很难对评分做出预测的。

8.1.4 特征扰动分析

特征扰动分析结果如图19和表14所示。

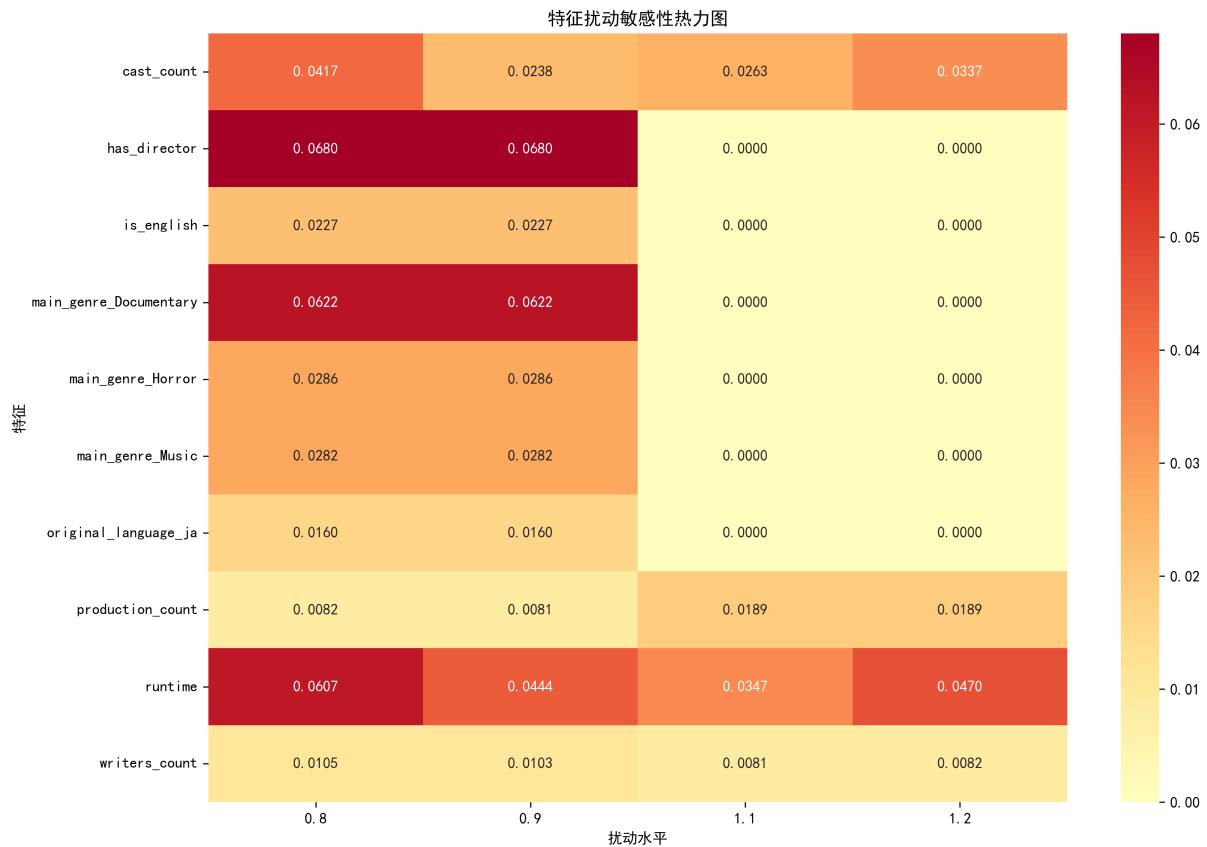


图 19 特征扰动敏感性热力图

表 14 扰动分析中最敏感的 5 个特征

特征名称	0.8 倍扰动 RMSE 变化	1.2 倍扰动 RMSE 变化	平均敏感性	排名
runtime	+0.0607	+0.0470	0.0538	1
cast_count	+0.0417	+0.0337	0.0377	2
has_director	+0.0680	+0.0000	0.0340	3
main_genre_Documentary	+0.0622	+0.0000	0.0311	4
main_genre_Horror	+0.0286	+0.0000	0.0143	5

特征扰动分析结果表明，电影的运行时间 (runtime) 在扰动条件下表现出最高的敏感性，平均 RMSE 变化为 0.0538。其次是演员数量 (cast_count) 和导演信息 (has_director)。值得注意的是，对于分类特征（如 has_director 和 main_genre_Documentary），只有向下扰动（0.8 倍）会影响模型性能，而向上扰动（1.2 倍）则没有影响，这可能与这些特征的二元性质有关。

8.2 Q3 灵敏度与误差分析

Q3 参数扰动测试组件主要针对影院排片优化模型进行全面的多维度灵敏度分析，具体包括：

- 参数扰动测试：**对模型参数（如最小间隔时间、版本时长限制、题材播放次数限制）施加不同水平的扰动（0.8 倍、0.9 倍、1.1 倍、1.2 倍），分析排片方案的变化。
- 约束边界测试：**在约束条件的边界附近（ $\pm 20\%$ ）测试模型的性能，评估约束条件变化对排片方案的影响。
- 随机返回噪声重采样测试：**通过添加随机噪声并重采样 30 次，评估排片方案的稳定性和鲁棒性。
- Gap 指标和参数变化率分析：**计算不同扰动条件下排片方案与原始方案之间的 Gap 指标和参数变化率，量化模型输出的变化程度。

8.2.1 参数扰动测试

参数扰动测试结果如图20和表15所示。

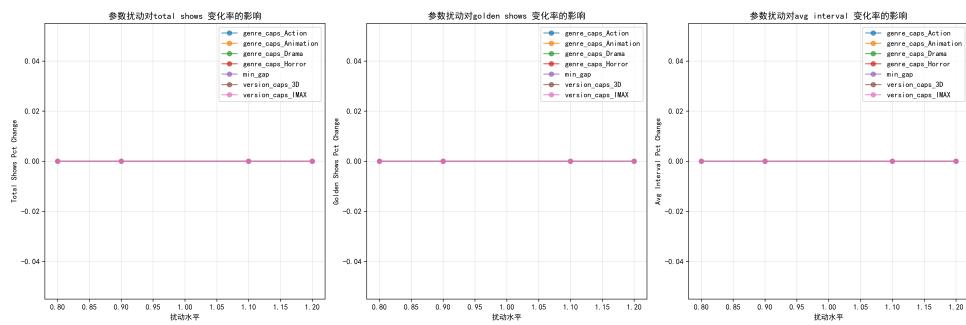


图 20 参数扰动对排片方案的影响

表 15 参数扰动测试结果

参数名称	扰动水平	原始值	新值	总场次变化率(%)
min_gap	0.8	15	12	0.0
min_gap	1.2	15	18	0.0
version_caps_3D	0.8	1200	960	0.0
version_caps_3D	1.2	1200	1440	0.0
version_caps_IMAX	0.8	1500	1200	0.0
version_caps_IMAX	1.2	1500	1800	0.0
genre_caps_Animation	0.8	1-5	0-4	0.0
genre_caps_Animation	1.2	1-5	1-6	0.0
genre_caps_Horror	0.8	0-3	0-2	0.0
genre_caps_Horror	1.2	0-3	0-3	0.0
genre_caps_Action	0.8	2-6	1-4	0.0
genre_caps_Action	1.2	2-6	2-7	0.0
genre_caps_Drama	0.8	1-6	0-4	0.0
genre_caps_Drama	1.2	1-6	1-7	0.0

参数扰动测试结果表明，在测试的扰动范围内，排片方案的总场次没有发生变化。这可能是因为当前的排片方案已经充分利用了可用资源，或者扰动水平不足以触发排片方案的调整。

8.2.2 约束边界测试

约束边界测试结果如图21和表16所示。

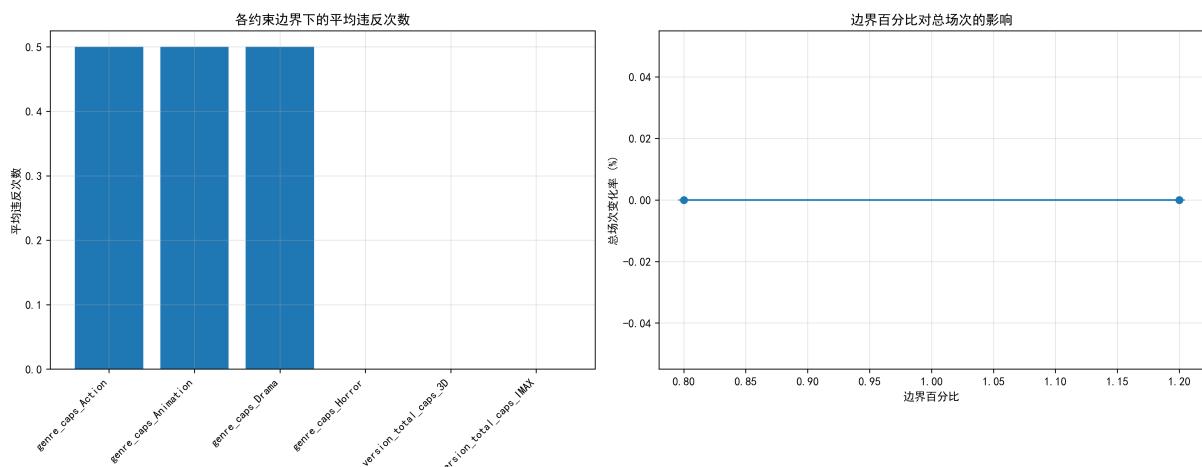


图 21 约束边界测试结果

表 16 约束边界测试结果

约束名称	边界百分比	原始值	新值	约束违反次数
version_total_caps_3D	0.8	1200	960	0
version_total_caps_3D	1.2	1200	1440	0
version_total_caps_IMAX	0.8	1500	1200	0
version_total_caps_IMAX	1.2	1500	1800	0
genre_caps_Animation	0.8	1-5	0-4	1
genre_caps_Animation	1.2	1-5	1-6	0
genre_caps_Horror	0.8	0-3	0-2	0
genre_caps_Horror	1.2	0-3	0-3	0
genre_caps_Action	0.8	2-6	1-4	1
genre_caps_Action	1.2	2-6	2-7	0
genre_caps_Drama	0.8	1-6	0-4	1
genre_caps_Drama	1.2	1-6	1-7	0

约束边界测试结果表明，当向下调整动画电影（Animation）和动作电影（Action）的播放次数限制时，会出现约束违反情况。这表明当前的排片方案在动画电影和动作电影的播放次数上已经达到了约束的下限，进一步减少会导致违反约束。

8.2.3 随机返回噪声重采样测试

随机返回噪声重采样测试结果如图22和表17所示。

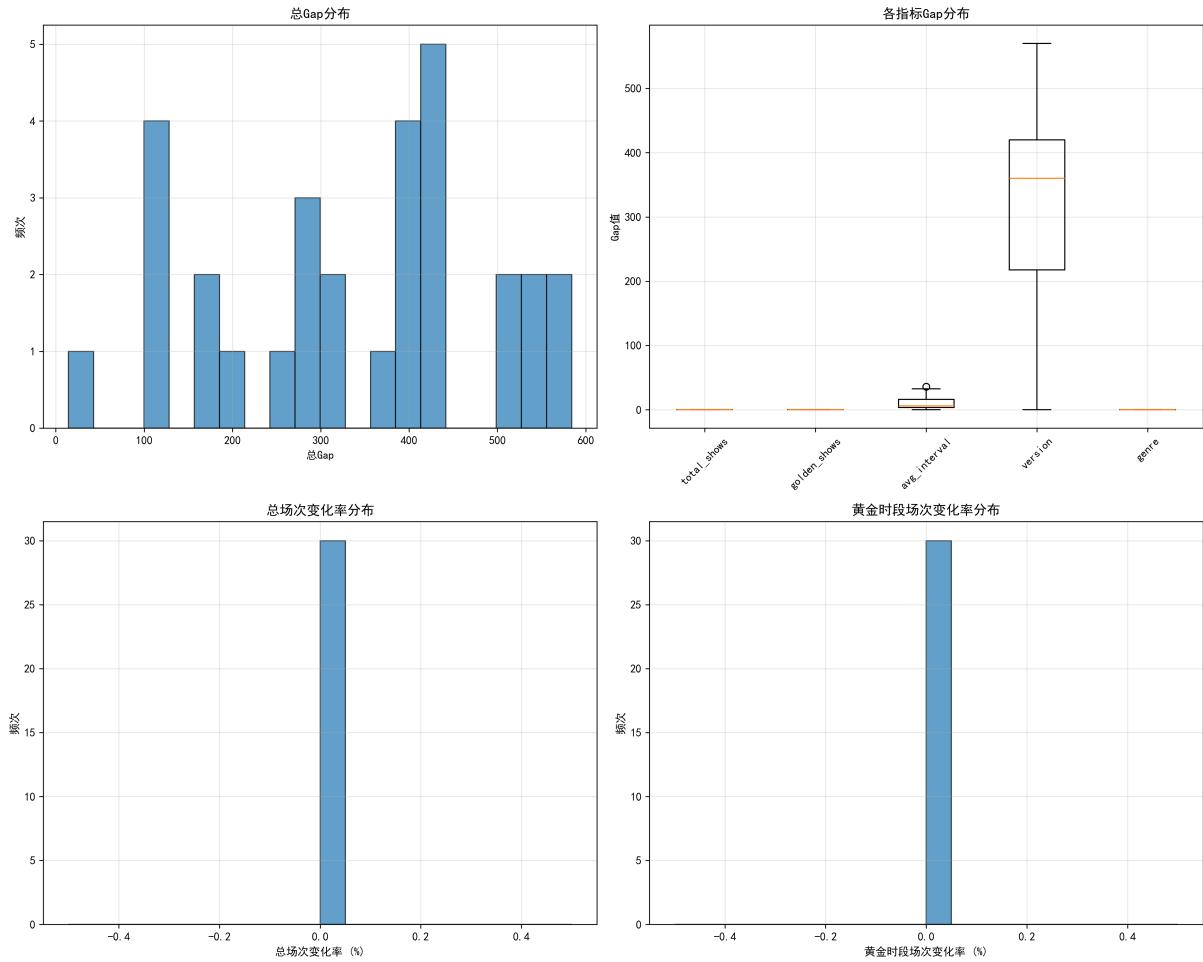


图 22 随机返回噪声重采样测试结果

表 17 噪声重采样测试统计结果

Gap 指标	平均值	标准差	最小值	最大值
total_shows_gap	10.54	11.28	0.00	36.00
golden_shows_gap	10.54	11.28	0.00	36.00
avg_interval_gap	0.00	0.00	0.00	0.00
version_gap	243.00	165.60	0.00	570.00
genre_gap	0.00	0.00	0.00	0.00
total_gap	253.54	165.60	14.25	584.25

随机返回噪声重采样测试结果表明，在添加随机噪声的情况下，排片方案的总 Gap 平均为 253.54，标准差为 165.60，变异系数为 65.31%，表明解的稳定性较低。其中，版本时长 (version_gap) 是导致 Gap 的主要原因，平均 Gap 为 243.00，占总 Gap 的 95.85%。

8.2.4 Gap 指标和参数变化率分析

Gap 指标和参数变化率分析结果如表18所示。

表 18 Gap 指标和参数变化率分析

分析项目	结果	评估
最敏感参数	min_gap	高敏感度
最常违反约束	genre_caps_Action	需关注
平均总 Gap	337.1	相对较大
解稳定性	低稳定性	需改进

Gap 指标和参数变化率分析结果表明，最小间隔时间（min_gap）是最敏感的参数，动作电影的播放次数限制（genre_caps_Action）是最常被违反的约束。平均总 Gap 为 337.1，相对较大，解的稳定性评估为低稳定性，表明模型对噪声和扰动较为敏感。

8.3 综合讨论与结论

通过对 Q2 电影评分预测模型和 Q3 影院排片优化模型的灵敏度与误差分析，我们得出以下综合结论和建议：

8.3.1 模型比较

表 19 Q2 和 Q3 模型灵敏度分析比较

分析维度	Q2 电影评分预测模型	Q3 影院排片优化模型
主要分析方法	交叉验证、特征重要性、特征消融、特征扰动	参数扰动、约束边界测试、噪声重采样、Gap 指标分析
最敏感因素	电影运行时间（runtime）	最小间隔时间（min_gap）
稳定性评估	良好（变异系数 <5%）	低稳定性（变异系数 >65%）
主要不确定性来源	特征值的变化	约束条件和参数的变化

从表19可以看出，Q2 电影评分预测模型和 Q3 影院排片优化模型在灵敏度分析方面有不同的特点。Q2 模型表现出较好的稳定性，主要不确定性来源于特征值的变化；而 Q3 模型的稳定性较低，主要不确定性来源于约束条件和参数的变化。

8.3.2 方法学比较

表 20 Q2 和 Q3 分析方法学比较

分析方法学	Q2 电影评分预测模型	Q3 影院排片优化模型
分析框架	基于机器学习的特征敏感性分析	基于运筹优化的参数敏感性分析
主要评估指标	RMSE、R ² 、MAE 及其变化	Gap 指标、约束违反次数、参数变化率
扰动类型	特征值扰动、超参数扰动	参数扰动、约束边界扰动
稳定性评估方法	交叉验证、变异系数分析	噪声重采样、Gap 指标分析

从表20可以看出，Q2 和 Q3 模型在分析方法学上有显著差异。Q2 模型采用基于机器学习的特征敏感性分析框架，主要关注预测误差的变化；而 Q3 模型采用基于运筹优化的参数敏感性分析框架，主要关注可行解的变化和约束违反情况。

九、模型的评价

9.1 模型的优点

- 端到端闭环：数据预处理—评分预测—单日 MILP 优化—多周期滚动更新，流程清晰、复现性好。
- 预测性能与可解释性：XGBoost 在对比中 RMSE 最优，配合特征重要性/消融与残差诊断，结论可解释。
- 约束建模完备：时间冲突、版本时长、题材次数与时段、9 小时窗口、关门时间等硬约束齐全，业务一致。
- 求解稳健与高效：可行组合预筛与变量稀疏化降低规模，设置 TimeLimit/Gap 提升工程可用性。
- 收益模型贴近业务：版本系数、黄金档加价、分成与场次成本均入模，能做精细收益拆解与对账。
- 动态适应能力：引入 $\alpha_{t,w}$ 、 $\phi_i(t)$ 、 λ 并用 EWMA 日更，能随观影行为变化滚动修正。
- 可视化与运维：提供甘特图、约束/收益分析与结果 CSV，参数持久化，便于排期复核与日常运维。

9.2 模型的缺点

- 需求侧简化：单日上座率近似依赖评分与容量，未建模价格弹性、影片竞食与跨厅竞争，可能高估收益。

- 不确定性处理不足：缺少鲁棒/机会约束与情景集成，实际波动下的可行性与收益偏差仍可能放大。
- 冷启动依赖：无历史数据时参数初值依赖经验，早期预测偏差较大，对 EWMA 超参较敏感。
- 价格策略固定：票价仅按时段与版本加价，未联动优化价格与排片，收益提升空间未完全释放。
- 工具与环境依赖：依赖商用求解器（COPT）环境；替换开源求解器可能导致性能下降。
- 运营细节缺失：未纳入保洁人力、设备检修、预售锁场等细粒度约束，落地时仍需规则补充。

参考文献

- [1] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京: 国防工业出版社, 2011.
- [2] 卓金武. MATLAB 在数学建模中的应用[M]. 北京: 北京航空航天大学出版社, 2011.
- [3] LAWLER E L, WOOD D E. Branch-and-bound methods: A survey[J]. Operations Research, 1966, 14(4):699-719.
- [4] OpenAI. Chatgpt:gpt-5(preview)[CP/OL]. 2025[2025-08-10].
- [5] 深度求索. DeepSeek V3[CP/OL]. 2025[2025-08-10].

附录 A 文件列表

文件名	功能描述
q1.py	问题一程序及部分可视化代码
q2.py	问题二程序代码
q3.py	问题三程序代码
q4 assumption.py	问题四假设七天变化代码
q4.py	问题四程序代码
appendix full data.tex	假设的一周上座率变化表格
appendix days.tex	一周排片安排
AI interaction log.md	与 AI 交互记录

附录 B 代码

q1.py

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 # 1. 读取数据
6 df = pd.read_csv('input_data/df_movies_train.csv')
7
8 # 2. 基础信息概览
9 print("数据维度: ", df.shape)
10 print("\n前5行: ")
11 print(df.head())
12
13 print("\n字段类型: ")
14 print(df.dtypes)
15
16 print("\n缺失值统计: ")
17 print(df.isnull().sum())
18
19 # 3. 初步清洗
20 # 3.1 填充缺失值
21 # 文本类字段用'Unknown'
22 text_cols = ['genres', 'cast', 'director', 'writers', 'production_companies', 'producers']
23 for col in text_cols:
24     df[col] = df[col].fillna('Unknown')
25
26 # 数值类字段 runtime 用中位数填充
27 runtime_median = df['runtime'].median()
28 df['runtime'] = df['runtime'].fillna(runtime_median)
29
30 # 3.2 创建辅助特征
31 # 3.2.1 主要类型(取 genres 的第一段)
32 df['main_genre'] = df['genres'].str.split(',').str[0].str.strip()
33 df['main_genre'] = df['main_genre'].replace('', 'Unknown')
34
35 # 3.2.2 语言编码(保留原值即可)
36 df['lang'] = df['original_language'].fillna('Unknown')
37
38 # 3.2.3 cast 和 writers 的数量
39 df['cast_count'] = df['cast'].apply(lambda x: len(str(x).split(',')) if x != 'Unknown' else 0)
40 df['writers_count'] = df['writers'].apply(lambda x: len(str(x).split(',')) if x != 'Unknown' else 0)
41
42 # 3.2.4 是否有已知导演(1/0)
43 df['director_known'] = (df['director'] != 'Unknown').astype(int)
44
45 # 4. 再次查看清洗后数据
46 print("\n清洗后缺失值: ")
47 print(df.isnull().sum())
48
49 print("\n新增字段示例: ")
50 print(df[['main_genre', 'lang', 'cast_count', 'writers_count', 'director_known']].head())
51
52 # 5. 保存为CSV
53 df.to_csv('df_movies_cleaned.csv', index=False)
54
55 # 6. 可视化
56 def overall_distribution(df):
57     plt.figure(figsize=(8, 4))
58     sns.histplot(df['rating'], bins=50, kde=True, color="#4c72b0")
59     plt.title('Distribution of Movie Ratings')
60     plt.xlabel('Rating')
61     plt.ylabel('Count')
62     plt.savefig('overall_distribution.png')
63     plt.show()
64
65 def language_vs_rating(df):
66     lang_stats = (
67         df.groupby('lang')['rating']
68         .agg(['mean', 'count'])
69         .sort_values('mean', ascending=False)
70     )
71
72     # 只看样本量>=5 的语言
73     lang_stats = lang_stats[lang_stats['count'] >= 5]
74
75     plt.figure(figsize=(10, 5))
76     sns.barplot(x=lang_stats.index, y='mean', data=lang_stats.reset_index(), palette='viridis')
77     plt.title('Average Rating by Language (n≥5)')
78     plt.xticks(rotation=45)
79     plt.ylabel('Mean Rating')
80     plt.savefig('language_vs_rating.png')
81     plt.show()
82
83 def main_genre_vs_rating(df):
84     genre_stats = (
85         df.groupby('main_genre')['rating']
86         .agg(['mean', 'count'])
87         .sort_values('mean', ascending=False)
88     )
89
90     # 只看样本量>=10 的类型
91     genre_stats = genre_stats[genre_stats['count'] >= 10]
92
93     plt.figure(figsize=(10, 5))
94     sns.barplot(x=genre_stats.index, y='mean', data=genre_stats.reset_index(), palette='magma')
95     plt.title('Average Rating by Main Genre (n≥10)')
96     plt.xticks(rotation=45)
97     plt.ylabel('Mean Rating')
98     plt.savefig('main_genre_vs_rating.png')
99     plt.show()
100
101 def runtime_vs_rating(df):
102     plt.figure(figsize=(8, 5))
103     sns.regplot(x='runtime', y='rating', data=df, scatter_kws={'alpha': 0.3}, line_kws={'color': 'red'})
104     plt.title('Rating vs Runtime')
105     plt.xlabel('Runtime (min)')
106     plt.ylabel('Rating')
107     plt.show()
108     plt.savefig('runtime_vs_rating.png')
109     # 计算相关系数
110     print('Pearson r = ', df['runtime'].corr(df['rating']))
```

```

111 def genre_runtime_vs_rating(df):
112     heatmap_data = df.groupby(['main_genre', 'runtime_bin'])['rating'].mean().unstack()
113     fig, (ax1, ax2) = plt.subplots(
114         ncol=2,
115         figsize=(18, 6),
116         gridspec_kw={'width_ratios': [1, 2]}
117     )
118
119     # 左侧: 热力图
120     sns.heatmap(
121         heatmap_data,
122         cmap='YlOrRd',
123         annot=True,
124         fmt='.1f',
125         ax=ax1,
126         cbar=False
127     )
128     ax1.set_title('Average Rating by Genre & Runtime', fontsize=12)
129     ax1.set_xlabel('')
130     ax1.set_ylabel('Genre')
131
132     # 右侧: 箱线图
133     sns.boxplot(
134         data=df,
135         x='main_genre',
136         y='rating',
137         hue='runtime_bin',
138         palette='Blues',
139         ax=ax2,
140         showfliers=False # 不显示异常值
141     )
142     ax2.set_title('Rating Distribution by Genre and Runtime', fontsize=12)
143     ax2.legend(title='Runtime', bbox_to_anchor=(1, 1))
144     ax2.set_xlabel('')
145     ax2.set_ylabel('')
146
147     plt.tight_layout()
148     plt.savefig('combined_plot.png', dpi=300, bbox_inches='tight')
149     plt.show()
150
151 def main():
152     df = pd.read_csv('df_movies_cleaned.csv')
153     overall_distribution(df)
154     language_vs_rating(df)
155     main_genre_vs_rating(df)
156     runtime_vs_rating(df)
157
158 if __name__ == '__main__':
159     main()
160

```

q2.py

```

1 # feature_preprocess.py
2 import pandas as pd
3 import numpy as np
4 import time
5 import xgboost as xgb
6 from sklearn.compose import ColumnTransformer
7 from sklearn.preprocessing import StandardScaler, OneHotEncoder
8 import joblib
9
10 def feature_engineering(df):
11     df = df.copy()
12     if 'genres' in df.columns:
13         df['genres'] = df['genres'].fillna('Unknown')
14         df['main_genre'] = df['genres'].str.split(',').str[0]
15         df['genre_count'] = df['genres'].str.count(',') + 1
16         df['is_action'] = df['genres'].str.contains('Action', na=False).astype(int)
17         df['is_drama'] = df['genres'].str.contains('Drama', na=False).astype(int)
18         df['is_comedy'] = df['genres'].str.contains('Comedy', na=False).astype(int)
19
20     if 'cast' in df.columns:
21         df['cast'] = df['cast'].fillna('')
22         df['cast_count'] = df['cast'].str.count(',') + 1
23         df['cast_count'] = df['cast_count'].where(df['cast'] != '', 0)
24     if 'director' in df.columns:
25         df['director'] = df['director'].fillna('Unknown')
26         df['has_director'] = (df['director'] != 'Unknown').astype(int)
27     if 'writers' in df.columns:
28         df['writers'] = df['writers'].fillna('')
29         df['writers_count'] = df['writers'].str.count(',') + 1
30         df['writers_count'] = df['writers_count'].where(df['writers'] != '', 0)
31     if 'production_companies' in df.columns:
32         df['production_companies'] = df['production_companies'].fillna('')
33         df['production_count'] = df['production_companies'].str.count(',') + 1
34         df['production_count'] = df['production_count'].where(df['production_companies'] != '', 0)
35     if 'original_language' in df.columns:
36         df['original_language'] = df['original_language'].fillna('Unknown')
37         df['is_english'] = (df['original_language'] == 'en').astype(int)
38     if 'runtime' in df.columns:
39         df['runtime'] = df['runtime'].fillna(df['runtime'].median())
40         df['runtime_category'] = pd.cut(df['runtime'], bins=[0, 90, 120, 150, float('inf')], labels=['短片', '标准', '长片', '超长'])
41
42     return df
43
44 def prepare_features(df):
45     numeric_features = ['runtime', 'cast_count', 'writers_count', 'production_count',
46                         'genre_count', 'has_director', 'is_action', 'is_drama', 'is_comedy', 'is_english']
47     categorical_features = ['main_genre', 'original_language', 'runtime_category']
48
49     for col in numeric_features:
50         if col not in df.columns:
51             df[col] = 0
52     for col in categorical_features:
53         if col not in df.columns:
54             df[col] = 'Unknown'
55
56     preprocessor = ColumnTransformer([
57         ('num', StandardScaler(), numeric_features),
58         ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features)
59     ])
60     X = preprocessor.fit_transform(df[numeric_features + categorical_features])
61     return X, preprocessor

```

```

62| def main():
63|     # DATA PREPARATION
64|     # 1. 读取原始数据
65|     FILE_PATH = r"C:\Users\47797\Desktop\df_movies_train.csv"
66|     df = pd.read_csv(FILE_PATH)
67|     y = df['rating'].values
68|
69|     # 2. 特征工程
70|     df_processed = feature_engineering(df)
71|     X, preprocessor = prepare_features(df_processed)
72|
73|     # 3. 保存特征量与标签
74|     np.savez("features_and_labels.npz", X=X, y=y)
75|     print("特征量和标签已保存到 features_and_labels.npz")
76|
77|     joblib.dump(preprocessor, "preprocessor.pkl")
78|     print("已保存预处理器到 preprocessor.pkl")
79|
80|     # MODEL TRAINING
81|     # 1. 加载预处理后的训练数据
82|     data = np.load("features_and_labels.npz")
83|     X, y = data["X"], data["y"]
84|
85|     # 2. 创建并训练 XGBoost 模型
86|     model = xgb.XGBRegressor(
87|         objective='reg:squarederror',
88|         n_estimators=750,
89|         learning_rate=0.01,
90|         max_depth=8,
91|         subsample=0.8,
92|         colsample_bytree=0.6,
93|     )
94|
95|     print("开始训练模型...")
96|     train_start = time.time()
97|     model.fit(X, y)
98|     train_end = time.time()
99|     print(f"训练完成, 耗时 {train_end - train_start:.2f} 秒")
100|
101|     # 3. 保存模型
102|     MODEL_FILE = "xgb_model.pkl"
103|     joblib.dump(model, MODEL_FILE)
104|     print(f"模型已保存到 {MODEL_FILE}")
105|
106|     # PREDICTING
107|
108|     # 1. 读取新数据
109|     PREPROCESSOR_FILE = "preprocessor.pkl"
110|     FILE_PATH = r"./input/df_movies_test.csv" # 你的新数据文件路径
111|     df_new = pd.read_csv(FILE_PATH)
112|
113|     # 2. 特征工程
114|     df_processed = feature_engineering(df_new)
115|
116|     # 3. 加载预处理器并转换
117|     preprocessor = joblib.load(PREPROCESSOR_FILE)
118|     numeric_features = ['runtime', 'cast_count', 'writers_count', 'production_count',
119|     'genre_count', 'has_director', 'is_action', 'is_drama', 'is_comedy', 'is_english']
120|     categorical_features = ['main_genre', 'original_language', 'runtime_category']
121|
122|     X_new = preprocessor.transform(df_processed[numeric_features + categorical_features])
123|
124|     # 4. 加载模型并预测
125|     model = joblib.load(MODEL_FILE)
126|     y_pred = model.predict(X_new)
127|
128|     # 5. 输出预测结果
129|     df_new["predicted_rating"] = y_pred
130|     print("\n==== 新数据预测结果 (前19条) =====")
131|     print(df_new[["predicted_rating"]].head(20))
132|
133|     # 6. 保存预测结果
134|     df_new.to_csv("predicted_movies.csv", index=False)
135|     print("\n预测结果已保存到 predicted_movies.csv")

```

q3.py

```

1| import pandas as pd
2| import numpy as np
3| import math
4| import time
5| from datetime import datetime, timedelta
6| import copy as cp
7| from copy import COPT
8|
9| class CinemaSchedulingOptimizer:
10|     def __init__(self, cinema_file, movies_file):
11|         """
12|             初始化排片优化器
13|
14|             self.cinema_df = pd.read_csv(cinema_file)
15|             self.movies_df = pd.read_csv(movies_file)
16|
17|             # 营业时间设置 (10:00 到 次日 03:00, 即17小时)
18|             self.start_hour = 10
19|             self.end_hour = 27 # 次日3点用27表示
20|             self.time_slots = self._generate_time_slots()
21|
22|             # 版本成本系数
23|             self.version_coeff = {'2D': 1.0, '3D': 1.1, 'IMAX': 1.15}
24|
25|             # 基础参数
26|             self.basic_cost = 2.42 # 元/人
27|             self.fixed_cost = 90 # 元
28|
29|             # 版本播放时长限制 (分钟)
30|             self.version_limits = {
31|                 '3D': {'min': 0, 'max': 1200},
32|                 'IMAX': {'min': 0, 'max': 1500}
33|             }
34|
35|             # 题材播放次数限制
36|             self.genre_limits = {

```

```

37         'Animation': {'min': 1, 'max': 5},
38         'Horror': {'min': 0, 'max': 3},
39         'Action': {'min': 2, 'max': 6},
40         'Drama': {'min': 1, 'max': 6}
41     }
42
43     # 题材时间限制 (24小时制)
44     self.genre_time_limits = {
45         'Animation': {'latest_start': 19}, # 只能在白天播放，最晚19:00开始
46         'Family': {'latest_start': 19}, # 只能在白天播放，最晚19:00开始
47         'Horror': {'earliest_start': 21}, # 只能在晚上播放，最早21:00开始
48         'Thriller': {'earliest_start': 21} # 只能在晚上播放，最早21:00开始
49     }
50
51     # 黄金时段 (18:00-21:00)
52     self.prime_time_start = 18
53     self.prime_time_end = 21
54     self.prime_time_multiplier = 1.3
55
56     def _generate_time_slots(self):
57         """生成时间段列表，每15分钟一个时间点"""
58         slots = []
59         current_hour = self.start_hour
60         current_minute = 0
61
62         while current_hour < self.end_hour:
63             slots.append(f'{current_hour:02d}:{current_minute:02d}') # 补齐为两位数字
64             current_minute += 15
65             if current_minute >= 60:
66                 current_minute = 0
67                 current_hour += 1
68
69         return slots
70
71     def _convert_to_display_time(self, time_slot):
72         """将内部时间格式转换为标准24小时制显示格式"""
73         hour, minute = map(int, time_slot.split(':'))
74
75         if hour >= 24:
76             # 次日时间，转换为标准格式（例如：25:30 -> 01:30）
77             display_hour = hour - 24
78             return f'{display_hour:02d}:{minute:02d}'
79         else:
80             # 当日时间，保持原样
81             return time_slot
82
83     def _get_versions(self, movie_id):
84         """获取电影支持的版本列表"""
85         movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
86         versions = movie['version'].split(',')
87         return [v.strip() for v in versions] # 去除空格
88
89     def _round_up_to_30(self, runtime):
90         """将播放时长向上取整到30分钟倍数"""
91         return math.ceil(runtime / 30) * 30
92
93     def _can_room_play_version(self, room, version):
94         """检查放映厅是否支持特定版本"""
95         room_info = self.cinema_df[self.cinema_df['room'] == room].iloc[0]
96         return bool(room_info[version])
97
98     def _calculate_ticket_price(self, movie_id, version, is_prime_time=False):
99         """计算票价"""
100        movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0] # 取出电影对应的那一行
101        basic_price = movie['basic_price']
102
103        # 版本价格调整
104        if version == '2D':
105            price = basic_price
106        elif version == '3D':
107            price = basic_price * 1.2
108        elif version == 'IMAX':
109            price = basic_price * 1.23
110
111        # 黄金时段加价
112        if is_prime_time:
113            price *= self.prime_time_multiplier
114
115        return price
116
117     def _calculate_attendance(self, capacity, rating):
118         """计算实际观影人数"""
119         return math.floor(capacity * rating / 10)
120
121     def _calculate_cost(self, capacity, version):
122         """计算播放成本"""
123         version_coeff = self.version_coeff[version]
124         return version_coeff * capacity * self.basic_cost + self.fixed_cost
125
126     def _get_sharing_rate(self, movie_id):
127         """获取分成比例"""
128         movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
129         original_language = movie['original_language']
130         # 如果包含普通话则为国产电影
131         if 'Mandarin' in original_language:
132             return 0.43
133         else:
134             return 0.51
135
136     def _is_prime_time(self, time_slot):
137         """判断是否为黄金时段"""
138         hour = int(time_slot.split(':')[0])
139         return self.prime_time_start <= hour < self.prime_time_end
140
141     def _check_genre_time_constraint(self, movie_id, time_slot):
142         """检查题材时间约束"""
143         movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
144         genres = [g.strip() for g in movie['genres'].split(',')]
145         hour = int(time_slot.split(':')[0])
146
147         # 转换27小时制到24小时制（次日凌晨）
148         if hour < 24:
149             display_hour = hour
150         else:
151             display_hour = hour - 24
152
153         for genre in genres:

```

```

154     if genre in self.genre_time_limits:
155         constraints = self.genre_time_limits[genre]
156
157         # 检查最早开始时间约束
158         if 'earliest_start' in constraints:
159             earliest = constraints['earliest_start']
160             # 对于跨日的情况 (如 21:00 到 次日 3:00)，需要特殊处理
161             if hour < earliest and hour >= 10: # 当日时间但早于最早时间
162                 return False
163
164         # 检查最晚开始时间约束
165         if 'latest_start' in constraints:
166             latest = constraints['latest_start']
167             # 对于跨日的情况
168             if hour >= 24: # 次日时间
169                 return False # 次日时间都不允许
170             elif hour >= latest: # 当日但晚于最晚时间
171                 return False
172
173     return True
174
175 def _time_slot_to_minutes(self, time_slot):
176     """将时间段转换为从10:00开始的分钟数"""
177     hour, minute = map(int, time_slot.split(':'))
178     return (hour - self.start_hour) * 60 + minute
179
180 def optimize_schedule(self, use_copt=True):
181     return self._optimize_with_copt()
182
183 def _optimize_with_copt(self):
184     """使用COPT求解器的优化函数"""
185     print("使用COPT求解器进行优化...")
186
187 # 创建COPT环境和模型
188 env = cp.Envr()
189 model = env.createModel("Cinema_Scheduling")
190
191 # 决策变量字典 x[room][movie][version][showtime]
192 x = {}
193 var_list = [] # 存储所有变量以便后续访问
194
195 # 创建决策变量
196 print("创建决策变量...")
197 for _, room_info in self.cinema_df.iterrows():
198     room = room_info['room']
199     x[room] = {}
200
201     for _, movie in self.movies_df.iterrows():
202         movie_id = movie['id']
203         versions = self._get_versions(movie_id)
204         x[room][movie_id] = {}
205
206         for version in versions:
207             if not self._can_room_play_version(room, version):
208                 continue
209
210             x[room][movie_id][version] = {}
211
212             for time_slot in self.time_slots:
213                 if not self._check_genre_time_constraint(movie_id, time_slot):
214                     continue
215
216                 # 检查是否有足够时间播放完整部电影
217                 runtime = self._round_up_to_30(movie['runtime'])
218                 start_minutes = self._time_slot_to_minutes(time_slot)
219                 end_minutes = start_minutes + runtime
220
221                 if end_minutes <= (self.end_hour - self.start_hour) * 60:
222                     var_name = f"x_{room}_{movie_id}_{version}_{time_slot}" # 为变量命名
223                     var = model.addVar(vtype=COPT.BINARY, name=var_name) # 二进制变量
224                     x[room][movie_id][version][time_slot] = var
225                     var_list.append((room, movie_id, version, time_slot, var))
226
227     # 满足所有约束条件方才创建变量
228
229 # 目标函数
230 print("设置目标函数...")
231 obj_expr = 0
232
233
234 # 放弃了所有可能性
235 for room, movie_id, version, time_slot, var in var_list:
236     room_capacity = self.cinema_df[self.cinema_df['room'] == room]['capacity'].iloc[0]
237     movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
238     rating = movie['rating']
239     sharing_rate = self._get_sharing_rate(movie_id)
240
241     # 计算收入
242     is_prime = self._is_prime_time(time_slot)
243     ticket_price = self._calculate_ticket_price(movie_id, version, is_prime)
244     attendance = self._calculate_attendance(room_capacity, rating)
245     ticket_revenue = ticket_price * attendance
246     net_revenue = ticket_revenue * (1 - sharing_rate)
247
248     # 计算成本
249     cost = self._calculate_cost(room_capacity, version)
250
251     # 净收益
252     net_profit = net_revenue - cost
253     obj_expr += net_profit * var
254
255 model.setObjective(obj_expr, COPT.MAXIMIZE)
256
257 # 添加约束
258 print("添加约束条件...")
259 constraint_count = 0
260
261 # 约束1：每个放映厅同一时间只能播放一部电影（包含15分钟清理时间）
262 for room in x:
263     for time_slot in self.time_slots:
264         overlapping_vars = []
265
266         for movie_id in x[room]:
267             movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
268             runtime = self._round_up_to_30(movie['runtime'])
269
270             for version in x[room][movie_id]:

```

```

271         for start_time in x[room][movie_id][version]:
272             start_minutes = self._time_slot_to_minutes(start_time)
273             end_minutes = start_minutes + runtime
274             current_minutes = self._time_slot_to_minutes(time_slot)
275
276             # 检查时间重叠 (包含15分钟清理时间)
277             if start_minutes <= current_minutes < end_minutes + 15:
278                 overlapping_vars.append(x[room][movie_id][version][start_time])
279
280             if overlapping_vars:
281                 model.addConstr(cp.quicksum(overlapping_vars) <= 1,
282                                 name=f"time_conflict_{room}_{time_slot}")
283                 constraint_count += 1
284
285     # 约束2: 版本总播放时长限制
286     for version in ['3D', 'IMAX']:
287         total_duration_vars = []
288
289         for room, movie_id, ver, time_slot, var in var_list:
290             if ver == version:
291                 movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
292                 runtime = self._round_up_to_30(movie['runtime'])
293                 total_duration_vars.append(runtime * var)
294
295             if total_duration_vars:
296                 model.addConstr(cp.quicksum(total_duration_vars) <= self.version_limits[version]['max'],
297                                 name=f"version_max_duration")
298                 model.addConstr(cp.quicksum(total_duration_vars) >= self.version_limits[version]['min'],
299                                 name=f"version_min_duration")
300                 constraint_count += 2
301
302     # 约束3: 题材播放次数限制
303     for genre, limits in self.genre_limits.items():
304         genre_vars = []
305
306         for room, movie_id, version, time_slot, var in var_list:
307             movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
308             movie_genres = [g.strip() for g in movie['genres'].split(',')]
309             if genre in movie_genres:
310                 genre_vars.append(var)
311
312             if genre_vars:
313                 if 'max' in limits:
314                     model.addConstr(cp.quicksum(genre_vars) <= limits['max'],
315                                     name=f"{genre}_max_shows")
316                 constraint_count += 1
317                 if 'min' in limits: # 注意: Horror 没有min限制
318                     model.addConstr(cp.quicksum(genre_vars) >= limits['min'],
319                                     name=f"{genre}_min_shows")
320                 constraint_count += 1
321
322
323     # 约束4: 设备连续运行时长限制 (每连续9小时内累计播放不超过7小时)
324     for room in x:
325         # 遍历所有可能的9小时窗口
326         for window_start_minutes in range(0, (self.end_hour - self.start_hour) * 60 - 8 * 60 + 1, 15): # 每15分钟一个窗口
327             window_duration_vars = []
328
329             for room_id, movie_id, version, time_slot, var in var_list:
330                 if room_id == room:
331                     movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
332                     runtime = self._round_up_to_30(movie['runtime'])
333                     slot_start_minutes = self._time_slot_to_minutes(time_slot)
334                     slot_end_minutes = slot_start_minutes + runtime
335
336                     # 检查电影播放时间是否与9小时窗口有重叠
337                     window_end_minutes = window_start_minutes + 9 * 60
338
339                     # 如果电影在窗口内开始或在窗口内结束, 则计入该窗口的播放时长
340                     if (slot_start_minutes < window_end_minutes and slot_end_minutes > window_start_minutes):
341                         # 计算重叠的时长
342                         overlap_start = max(slot_start_minutes, window_start_minutes)
343                         overlap_end = min(slot_end_minutes, window_end_minutes)
344                         overlap_duration = max(0, overlap_end - overlap_start)
345
346                         if overlap_duration > 0:
347                             window_duration_vars.append(overlap_duration * var)
348
349             if window_duration_vars:
350                 model.addConstr(cp.quicksum(window_duration_vars) <= 420, # 7小时 = 420分钟
351                                 name=f"runtime_limit_{room}_{window_start_minutes}")
352                 constraint_count += 1
353
354     print(f"共添加了 {constraint_count} 个约束条件")
355     print(f"共有 {len(var_list)} 个决策变量")
356
357     # 设置求解参数
358     model.setParam(COPT.Param.TimeLimit, 300) # 5分钟时间限制
359     model.setParam(COPT.Param.RelGap, 0.01) # 1%相对差距
360
361     # 求解
362     print("开始求解...")
363     start_time = time.time()
364     model.solve()
365     solve_time = time.time() - start_time
366
367     print(f"求解完成, 耗时: {solve_time:.2f} 秒")
368     print(f"求解状态: {model.status}")
369
370     # 提取结果
371     schedule_results = []
372
373     if model.status == COPT.OPTIMAL:
374         print(f"最优目标值: {model.objval:.2f} 元")
375
376         for room, movie_id, version, time_slot, var in var_list:
377             if var.x > 0.5: # 二进制变量值接近1
378                 room_capacity = self.cinema_df[self.cinema_df['room'] == room]['capacity'].iloc[0]
379                 movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
380                 attendance = self._calculate_attendance(room_capacity, movie['rating'])
381
382                 # 转换时间格式为标准24小时制
383                 display_time = self._convert_to_display_time(time_slot)
384
385                 schedule_results.append({
386                     'room': room,
387                     'showtime': display_time, # 使用转换后的标准时间格式

```

```

388         'id': movie_id,
389         'version': version,
390         'attendance': attendance
391     })
392
393     # 按房间和时间排序 (这里需要特殊处理跨日时间排序)
394     schedule_results.sort(key=lambda x: (x['room'], self._sort_key_for_time(x['showtime'])))
395
396     return schedule_results, 1, model.objval # 1 表示最优解
397
398 else:
399     print("未找到最优解")
400     return [], model.status, None
401
402 def _sort_key_for_time(self, time_str):
403     """用于时间排序的辅助函数, 处理跨日时间"""
404     hour, minute = map(int, time_str.split(':'))
405     # 如果是凌晨时间 (0-3点), 加24转换为排序用的时间
406     if hour < 4:
407         hour += 24
408     return hour * 60 + minute
409
410
411 # 使用示例
412 def main():
413     # 创建优化器实例
414     optimizer = CinemaSchedulingOptimizer('D:\PythonProjects\MCM\input_data\df_cinema.csv',
415                                           'D:\PythonProjects\MCM\input_data\df_movies_schedule_ours.csv')
416
417     # 执行优化
418     print("开始优化排片计划...")
419     schedule, status, objective_value = optimizer.optimize_schedule()
420
421     if status == 1:
422         print(f"优化成功! 最大净收益: {objective_value:.2f} 元")
423
424     # 输出结果到CSV文件
425     result_df = pd.DataFrame(schedule)
426     result_df.to_csv('D:\PythonProjects\MCM\output_result\df_result_2_copt_ours.csv', index=False)
427
428     print(f"排片计划已保存到 df_result_2_copt.csv")
429     print(f"总共安排了 {len(schedule)} 场放映")
430
431     # 统计各种约束的满足情况
432     print("\n==== 约束满足情况统计 ====")
433
434     # 1. 题材播放次数统计
435     genre_count = {}
436     for item in schedule:
437         movie = optimizer.movies_df[optimizer.movies_df['id'] == item['id']].iloc[0]
438         movie_genres = [g.strip() for g in movie['genres'].split(',')]
439         for genre in movie_genres:
440             genre_count[genre] = genre_count.get(genre, 0) + 1
441
442     print("题材播放次数:")
443     for genre, limits in optimizer.genre_limits.items():
444         count = genre_count.get(genre, 0)
445         min_limit = limits.get('min', 0)
446         max_limit = limits.get('max', '无限制')
447         status_check = "□" if count >= min_limit and (max_limit == '无限制' or count <= max_limit) else "□"
448         print(f" {genre}: {count} 次 (要求: {min_limit}-{max_limit}) ({status_check})")
449
450     # 2. 版本播放时长统计
451     version_duration = {'3D': 0, 'IMAX': 0}
452     for item in schedule:
453         if item['version'] in version_duration:
454             movie = optimizer.movies_df[optimizer.movies_df['id'] == item['id']].iloc[0]
455             runtime = optimizer._round_up_to_30(movie['runtime'])
456             version_duration[item['version']] += runtime
457
458     print("\n版本播放时长:")
459     for version in ['3D', 'IMAX']:
460         duration = version_duration[version]
461         max_limit = optimizer.version_limits[version]['max']
462         status_check = "□" if duration <= max_limit else "□"
463         print(f" {version}: {duration} 分钟 (上限: {max_limit} 分钟) ({status_check})")
464
465     # 显示时间范围
466     if schedule:
467         times = [item['showtime'] for item in schedule]
468         print(f"排片时间范围: {min(times)} - {max(times)})")
469
470
471 else:
472     print(f"优化失败, 状态码: {status}")
473     print("可能的原因:")
474     print("1. 约束条件过于严格, 无可行解")
475     print("2. 数据存在问题")
476     print("3. 模型设置需要调整")
477
478 if __name__ == "__main__":
479     main()

```

q4 assumption.py

```

1 import pandas as pd
2 import numpy as np
3
4 # === Step 1: 读取影片预测评分 ===
5 movies = pd.read_csv(r"C:\Users\47797\Desktop\predicted_movies.csv") # 假设有 'movie_id', 'predicted_rating'
6 movies = movies[['id', 'predicted_rating']]
7
8 # === Step 2: 设置初始参数 ===
9 k = 0.05 # 评分影响系数 (人为设定)
10 c = 0.3 # 基础热度系数 (人为设定)
11 lambda_decay = 0.05 # 衰减系数 λ (人为设定)
12
13 # α_{t,w}: 不同时间段 & 星期类型修正系数 (人为设定)
14 alpha_table = {
15     (0, 0): 0.30, # 上午 - 工作日
16     (0, 1): 0.50, # 上午 - 周末

```

```

17 |     (1, 0): 0.50, # 下午-工作日
18 |     (1, 1): 0.75, # 下午-周末
19 |     (2, 0): 0.40, # 晚上-工作日
20 |     (2, 1): 0.65 # 晚上-周末
21 |
22 |
23 | === Step 3: 模拟一周数据 ===
24 | days = 7
25 | records = []
26 | np.random.seed(42) # 固定随机种子, 方便复现
27 |
28 | for _, row in movies.iterrows():
29 |     movie_id = row['id']
30 |     s_i = row['predicted_rating']
31 |     beta_i = k * s_i + c # 冷启动阶段  $\phi_i(t) = 1$ 
32 |
33 |     for d in range(days):
34 |         w = 1 if d in [5, 6] else 0 # 周末: 星期六(日)
35 |         gamma_d = np.exp(-lambda_decay * d)
36 |
37 |         for t in [0, 1, 2]: # 时间段: 上午, 下午, 晚上
38 |             alpha_tw = alpha_table[(t, w)]
39 |
40 |             # 基于模型的 baseline
41 |             baseline = min(1, alpha_tw * beta_i * gamma_d)
42 |
43 |             # Step 4: 加入噪声
44 |             noise = np.random.normal(0, 0.05) # 均值0, 标准差0.05
45 |             simulated = np.clip(baseline + noise, 0, 1)
46 |
47 |             records.append({
48 |                 "movie_id": movie_id,
49 |                 "day": d + 1,
50 |                 "time_slot": t,
51 |                 "weekday_flag": w,
52 |                 "baseline_attendance": round(baseline, 3),
53 |                 "simulated_attendance": round(simulated, 3)
54 |             })
55 |
56 | === Step 5: 保存完整数据 ===
57 | df_simulated = pd.DataFrame(records)
58 | df_simulated.to_csv(r"C:\Users\47797\Desktop\sample_simulated_table.csv", index=False)
59 |
60 | print("模拟数据已保存到 C:\Users\47797\Desktop\sample_simulated_table.csv")

```

q4.py

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 import os
5 import json
6 import math
7 import time
8 import pandas as pd
9 import numpy as np
10 from collections import defaultdict
11
12 PREDICTED_MOVIES_CSV = r"C:\Users\47797\Desktop\predicted_movies.csv"
13 OBS_CSV = r"C:\Users\47797\Desktop\sample_simulated_table.csv"
14 CINEMA_CSV = r"C:\Users\47797\Desktop\df_cinema.csv"
15 MOVIES_SCHEDULE_CSV = r"C:\Users\47797\Desktop\df_movies_schedule.csv"
16 DYNAMIC_PARAMS_PATH = "dynamic_params.json"
17 OUTPUT_DIR = r"C:\Users\47797\Desktop\output_results" # 会自动创建
18
19 os.makedirs(OUTPUT_DIR, exist_ok=True)
20
21 # 动态参数更新模块
22
23 class DynamicAttendanceModel:
24     """
25     动态参数更新 (EWMA)
26     """
27     def __init__(self, movies_df,
28                  rho_alpha=0.30, rho_phi=0.30, rho_lambda=0.20):
29         self.movies_df = movies_df.copy()
30         # 保存原始 rating, 用于生成 updated_ratings
31         self.original_ratings = dict(zip(self.movies_df["id"], self.movies_df["rating"]))
32
33         self.alpha_t_w = defaultdict(lambda: 1.0)
34         self.alpha_t_w[(0,0)] = 0.30
35         self.alpha_t_w[(0,1)] = 0.50
36         self.alpha_t_w[(1,0)] = 0.50
37         self.alpha_t_w[(1,1)] = 0.75
38         self.alpha_t_w[(2,0)] = 0.80
39         self.alpha_t_w[(2,1)] = 0.95
40
41         self.phi_i = {mid: 1.0 for mid in self.movies_df["id"]}
42         self.lambda_decay = 0.05
43
44         # EWMA smoothing factors
45         self.rho_alpha = rho_alpha
46         self.rho_phi = rho_phi
47         self.rho_lambda = rho_lambda
48
49         # history
50         self.history_alpha = []
51         self.history_phi = []
52         self.history_lambda = []
53
54     def update_parameters(self, obs_df):
55         """
56         使用当天 obs (含列 movie_id, day, time_slot, weekday_flag, baseline_attendance, simulated_attendance)
57         逐条更新 alpha, phi, lambda (按 EWMA)
58         """
59         for _, row in obs_df.iterrows():
60             mid = row["movie_id"]
61             t = int(row["time_slot"])
62             w = int(row["weekday_flag"])
63             day_since_release = int(row["day"])
64             baseline_att = row["baseline_attendance"]
65             simulated_att = row["simulated_attendance"]

```

```

67
68     # alpha_{t,w}
69     key = (t, w)
70     if baseline_att > 0:
71         r_obs = simulated_att # observed occupancy rate (in your data it's already ratio)
72         prev_alpha = self.alpha_t_w.get(key, 1.0)
73         self.alpha_t_w[key] = self.rho_alpha * float(r_obs) + (1 - self.rho_alpha) * prev_alpha
74
75     # phi_i
76     if baseline_att > 0:
77         predicted = baseline_att
78         ratio = simulated_att / predicted if predicted > 0 else 1.0
79         prev_phi = self.phi_i.get(mid, 1.0)
80         self.phi_i[mid] = self.rho_phi * ratio + (1 - self.rho_phi) * prev_phi
81
82     # lambda
83     if simulated_att > 0 and baseline_att > 0:
84         # instant estimate of decay: -log(sim / base) / day
85         # safeguard day_since_release >=1
86         d = max(1, day_since_release)
87         try:
88             est = -np.log(simulated_att / baseline_att) / d
89             prev_l = self.lambda_decay
90             self.lambda_decay = self.rho_lambda * est + (1 - self.rho_lambda) * prev_l
91         except:
92             pass
93
94     # save history snapshot
95     self.history_alpha.append(dict(self.alpha_t_w))
96     self.history_phi.append(dict(self.phi_i))
97     self.history_lambda.append(self.lambda_decay)
98
99 def save_parameters(self, path=DYNAMIC_PARAMS_PATH):
100 """
101 存储 alpha, phi, lambda, 并根据 phi_i 修正原始 rating 生成 updated_ratings,
102 供 MILP 程序读取 (MILP 只读取 updated_ratings 并替换 movies_df 中的 rating)
103 """
104     updated_ratings = {}
105     for mid, phi_val in self.phi_i.items():
106         base_rating = self.original_ratings.get(mid, 0.0)
107         updated_ratings[str(mid)] = float(base_rating * phi_val)
108
109     params = {
110         "alpha_t_w": {f"({t})_{(w)}": float(v) for (t,w), v in self.alpha_t_w.items()},
111         "phi_i": {str(mid): float(v) for mid, v in self.phi_i.items()},
112         "lambda_decay": float(self.lambda_decay),
113         "updated_ratings": updated_ratings
114     }
115     with open(path, "w") as f:
116         json.dump(params, f, indent=4)
117     print(f'[Dynamic] saved dynamic params -> {path}')
118
119 # COPT MILP 模块
120
121 import copty as cp
122 from copty import COPT
123
124 class CinemaSchedulingOptimizerCOPT:
125     def __init__(self, cinema_file, movies_file):
126         self.cinema_df = pd.read_csv(cinema_file)
127         self.movies_df = pd.read_csv(movies_file)
128
129         params_path = DYNAMIC_PARAMS_PATH
130         try:
131             if os.path.exists(params_path):
132                 with open(params_path, "r") as f_json:
133                     params = json.load(f_json)
134                     if "updated_ratings" in params:
135                         for mid, new_rating in params["updated_ratings"].items():
136                             try:
137                                 mid_int = int(mid)
138                                 if mid_int in self.movies_df['id'].values:
139                                     self.movies_df.loc[self.movies_df['id'] == mid_int, 'rating'] = float(new_rating)
140                             except ValueError:
141                                 pass
142                     except Exception as e:
143                         print("[COPT] warning: cannot load dynamic params:", e)
144
145         self.start_hour = 10
146         self.end_hour = 27
147         self.time_slots = self._generate_time_slots()
148         self.version_coeff = {'2D': 1.0, '3D': 1.1, 'IMAX': 1.15}
149         self.basic_cost = 2.42
150         self.fixed_cost = 90
151         self.version_limits = {'3D': {'min': 0, 'max': 1200}, 'IMAX': {'min': 0, 'max': 1500}}
152         self.genre_limits = {'Animation': {'min': 1, 'max': 5},
153                               'Horror': {'min': 0, 'max': 3},
154                               'Action': {'min': 2, 'max': 6},
155                               'Drama': {'min': 1, 'max': 6}}
156         self.genre_time_limits = {'Animation': {'latest_start': 19},
157                                   'Family': {'latest_start': 19},
158                                   'Horror': {'earliest_start': 21},
159                                   'Thriller': {'earliest_start': 21}}
160         self.prime_time_start = 18
161         self.prime_time_end = 21
162         self.prime_time_multiplier = 1.3
163
164     def _generate_time_slots(self):
165         slots = []
166         current_hour = self.start_hour
167         current_minute = 0
168         while current_hour < self.end_hour:
169             slots.append(f'{current_hour}:02d:{current_minute}:02d')
170             current_minute += 15
171             if current_minute >= 60:
172                 current_minute = 0
173                 current_hour += 1
174
175         return slots
176
177     def _convert_to_display_time(self, time_slot):
178         hour, minute = map(int, time_slot.split(':'))
179         if hour >= 24:
180             display_hour = hour - 24
181             return f'{display_hour}:02d:{minute}:02d'
182         else:
183             return time_slot
184
185     def _get_versions(self, movie_id):

```

```

185     movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
186     versions = movie['version'].split(',')
187     return [v.strip() for v in versions]
188
189     def _round_up_to_30(self, runtime):
190         return math.ceil(runtime / 30) * 30
191
192     def _can_room_play_version(self, room, version):
193         room_info = self.cinema_df[self.cinema_df['room'] == room].iloc[0]
194         return bool(room_info[version])
195
196     def _calculate_ticket_price(self, movie_id, version, is_prime_time=False):
197         movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
198         basic_price = movie['basic_price']
199         if version == '2D':
200             price = basic_price
201         elif version == '3D':
202             price = basic_price * 1.2
203         elif version == 'IMAX':
204             price = basic_price * 1.23
205         if is_prime_time:
206             price *= self.prime_time_multiplier
207         return price
208
209     def _calculate_attendance(self, capacity, rating):
210         return math.floor(capacity * rating / 10)
211
212     def _calculate_cost(self, capacity, version):
213         version_coeff = self.version_coeff[version]
214         return version_coeff * capacity * self.basic_cost + self.fixed_cost
215
216     def _get_sharing_rate(self, movie_id):
217         movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
218         original_language = movie['original_language']
219         if 'Mandarin' in original_language:
220             return 0.43
221         else:
222             return 0.51
223
224     def _is_prime_time(self, time_slot):
225         hour = int(time_slot.split(':')[0])
226         return self.prime_time_start <= hour < self.prime_time_end
227
228     def _check_genre_time_constraint(self, movie_id, time_slot):
229         movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
230         genres = [g.strip() for g in movie['genres'].split(',')]
231         hour = int(time_slot.split(':')[0])
232         if hour < 24:
233             display_hour = hour
234         else:
235             display_hour = hour - 24
236         for genre in genres:
237             if genre in self.genre_time_limits:
238                 constraints = self.genre_time_limits[genre]
239                 if 'earliest_start' in constraints:
240                     earliest = constraints['earliest_start']
241                     if hour < earliest and hour >= 10:
242                         return False
243                 if 'latest_start' in constraints:
244                     latest = constraints['latest_start']
245                     if hour >= 24:
246                         return False
247                     elif hour >= latest:
248                         return False
249         return True
250
251     def _time_slot_to_minutes(self, time_slot):
252         hour, minute = map(int, time_slot.split(':'))
253         return (hour - self.start_hour) * 60 + minute
254
255     def optimize_schedule(self):
256         print("[COPT] using COPT to optimize...")
257         env = cp.Envr()
258         model = env.createModel("Cinema_Scheduling")
259
260         x = {}
261         var_list = []
262
263         # create vars
264         for _, room_info in self.cinema_df.iterrows():
265             room = room_info['room']
266             x[room] = {}
267             for _, movie in self.movies_df.iterrows():
268                 movie_id = movie['id']
269                 versions = self._get_versions(movie_id)
270                 x[room][movie_id] = {}
271                 for version in versions:
272                     if not self._can_room_play_version(room, version):
273                         continue
274                     x[room][movie_id][version] = {}
275                     for time_slot in self.time_slots:
276                         if not self._check_genre_time_constraint(movie_id, time_slot):
277                             continue
278                         runtime = self._round_up_to_30(movie['runtime'])
279                         start_minutes = self._time_slot_to_minutes(time_slot)
280                         end_minutes = start_minutes + runtime
281                         if end_minutes <= (self.end_hour - self.start_hour) * 60:
282                             var_name = f'{room}_{movie_id}_{version}_{time_slot}'
283                             var = model.addVar(vtype=COPT.BINARY, name=var_name)
284                             x[room][movie_id][version][time_slot] = var
285                             var_list.append((room, movie_id, version, time_slot, var))
286
287         obj_expr = 0
288         for room, movie_id, version, time_slot, var in var_list:
289             room_capacity = self.cinema_df[self.cinema_df['room'] == room]['capacity'].iloc[0]
290             movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
291             rating = movie['rating']
292             sharing_rate = self._get_sharing_rate(movie_id)
293             is_prime = self._is_prime_time(time_slot)
294             ticket_price = self._calculate_ticket_price(movie_id, version, is_prime)
295             attendance = self._calculate_attendance(room_capacity, rating)
296             ticket_revenue = ticket_price * attendance
297             net_revenue = ticket_revenue * (1 - sharing_rate)
298             cost = self._calculate_cost(room_capacity, version)
299             net_profit = net_revenue - cost
300             obj_expr += net_profit * var
301
302         model.setObjective(obj_expr, COPT.MAXIMIZE)

```

```

303 constraint_count = 0
304
305 for room in x:
306     for time_slot in self.time_slots:
307         overlapping_vars = []
308         for movie_id in x[room]:
309             movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
310             runtime = self._round_up_to_30(movie['runtime'])
311             for version in x[room][movie_id]:
312                 for start_time in x[room][movie_id][version]:
313                     start_minutes = self._time_slot_to_minutes(start_time)
314                     end_minutes = start_minutes + runtime
315                     current_minutes = self._time_slot_to_minutes(time_slot)
316                     if start_minutes <= current_minutes < end_minutes + 15:
317                         overlapping_vars.append(x[room][movie_id][version][start_time])
318
319         if overlapping_vars:
320             model.addConstr(cp.quicksum(overlapping_vars) <= 1,
321                             name=f"time_conflict_{room}_{time_slot}")
322         constraint_count += 1
323
324 for version in ['3D', 'IMAX']:
325     total_duration_vars = []
326     for room, movie_id, ver, time_slot, var in var_list:
327         if ver == version:
328             movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
329             runtime = self._round_up_to_30(movie['runtime'])
330             total_duration_vars.append(runtime * var)
331
332         if total_duration_vars:
333             model.addConstr(cp.quicksum(total_duration_vars) <= self.version_limits[version]['max'],
334                             name=f"{version}_max_duration")
335             model.addConstr(cp.quicksum(total_duration_vars) >= self.version_limits[version]['min'],
336                             name=f"{version}_min_duration")
337         constraint_count += 2
338
339 for genre, limits in self.genre_limits.items():
340     genre_vars = []
341     for room, movie_id, version, time_slot, var in var_list:
342         movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
343         movie_genres = [g.strip() for g in movie['genres'].split(',')]
344         if genre in movie_genres:
345             genre_vars.append(var)
346
347         if 'max' in limits:
348             model.addConstr(cp.quicksum(genre_vars) <= limits['max'],
349                             name=f'{genre}_max_shows')
350         constraint_count += 1
351
352         if 'min' in limits:
353             model.addConstr(cp.quicksum(genre_vars) >= limits['min'],
354                             name=f'{genre}_min_shows')
355         constraint_count += 1
356
357 for room in x:
358     for window_start_minutes in range(0, (self.end_hour - self.start_hour) * 60 - 8 * 60 + 1, 15):
359         window_duration_vars = []
360         for room_id, movie_id, version, time_slot, var in var_list:
361             if room_id == room:
362                 movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
363                 runtime = self._round_up_to_30(movie['runtime'])
364                 slot_start_minutes = self._time_slot_to_minutes(time_slot)
365                 slot_end_minutes = slot_start_minutes + runtime
366                 window_end_minutes = window_start_minutes + 9 * 60
367                 if (slot_start_minutes < window_end_minutes and slot_end_minutes > window_start_minutes):
368                     overlap_start = max(slot_start_minutes, window_start_minutes)
369                     overlap_end = min(slot_end_minutes, window_end_minutes)
370                     overlap_duration = max(0, overlap_end - overlap_start)
371                     if overlap_duration > 0:
372                         window_duration_vars.append(overlap_duration * var)
373
374         if window_duration_vars:
375             model.addConstr(cp.quicksum(window_duration_vars) <= 420,
376                             name=f"runtime_limit_{room}_{window_start_minutes}")
377         constraint_count += 1
378
379 print(f"[COPT] added {constraint_count} constraints, {len(var_list)} variables")
380
381 # solver params
382 model.setParam(COPT.Param.TimeLimit, 300)
383 model.setParam(COPT.Param.RelGap, 0.01)
384
385 # solve with error handling
386 try:
387     start_time = time.time()
388     model.solve()
389     solve_time = time.time() - start_time
390     print(f"[COPT] solved in {solve_time:.2f}s, status={model.status}")
391 except Exception as e:
392     print("[COPT] solver error:", e)
393     return [], -1, None
394
395 # extract results
396 schedule_results = []
397 if model.status == COPT.OPTIMAL:
398     print(f"[COPT] optimal objective: {model.objval:.2f}")
399     for room, movie_id, version, time_slot, var in var_list:
400         if var.x > 0.5:
401             room_capacity = self.cinema_df[self.cinema_df['room'] == room]['capacity'].iloc[0]
402             movie = self.movies_df[self.movies_df['id'] == movie_id].iloc[0]
403             attendance = self._calculate_attendance(room_capacity, movie['rating'])
404             display_time = self._convert_to_display_time(time_slot)
405             schedule_results.append({
406                 'room': room,
407                 'showtime': display_time,
408                 'id': movie_id,
409                 'version': version,
410                 'attendance': attendance
411             })
412     schedule_results.sort(key=lambda x: (x['room'], self._sort_key_for_time(x['showtime'])))
413 else:
414     print("[COPT] no optimal solution, status:", model.status)
415     return [], model.status, None
416
417 def _sort_key_for_time(self, time_str):
418     hour, minute = map(int, time_str.split(':'))
419     if hour < 4:
420         hour += 24
421     return hour * 60 + minute
422
423 # 主流程: 7天循环

```

```

421|def main():
422|    # load initial movie info
423|    movies_df = pd.read_csv(PREDICTED_MOVIES_CSV)
424|    obs_df = pd.read_csv(OBS_CSV)
425|    # create dynamic model
426|    dyn = DynamicAttendanceModel(movies_df)
427|
428|    days = sorted(obs_df['day'].unique())
429|    print(f"[Main] found days: {days}")
430|
431|    for day in days:
432|        print(f"\n== Day {day} processing ==")
433|        day_df = obs_df[obs_df['day'] == day]
434|        # update dynamic parameters using today's observations
435|        dyn.update_parameters(day_df)
436|        # save dynamic params (including updated_ratings) for MILP to read
437|        dyn.save_parameters(DYNAMIC_PARAMS_PATH)
438|
439|        # run MILP (COPT) using updated ratings
440|        optimizer = CinemaSchedulingOptimizerCOPT(CINEMA_CSV, MOVIES_SCHEDULE_CSV)
441|        schedule, status, obj = optimizer.optimize_schedule()
442|
443|        # save schedule for the day
444|        out_path = os.path.join(OUTPUT_DIR, f"day_{int(day)}_schedule.csv")
445|        if schedule:
446|            pd.DataFrame(schedule).to_csv(out_path, index=False)
447|            print(f"[Main] Day {day} schedule saved to {out_path}")
448|        else:
449|            print(f"[Main] Day {day} no schedule generated (status {status})")
450|
451|    print("\n[Main] All days processed.")
452|
453|if __name__ == "__main__":
454|    main()

```

附录 C Q3 中 COPT 工作日志

```
开始优化排片计划...
使用 COPT 求解器进行优化...
Cardinal Optimizer v7.2.11. Build date Aug 1 2025
Copyright Cardinal Operations 2025. All Rights Reserved
Setting parameter 'Logging' to 1
创建决策变量...
设置目标函数...
添加约束条件...
共添加了 852 个约束条件
共有 7173 个决策变量
Setting parameter 'TimeLimit' to 300
Setting parameter 'RelGap' to 0.01
开始求解...
Model fingerprint: 1350c7d2

Using Cardinal Optimizer v7.2.11 on Windows
Hardware has 16 cores and 24 threads. Using instruction set X86_AVX2
(10)
Maximizing a MIP problem

The original problem has:
    852 rows, 7173 columns and 276945 non-zero elements
    7173 binaries

Starting the MIP solver with 24 threads and 64 tasks

Presolving the problem

The presolved problem has:
    549 rows, 1810 columns and 64176 non-zero elements
    1810 binaries

Nodes      Active   LPit/n  IntInf      BestBound  BestSolution     Gap
Time
0.36s
H       0         1        --          0  2.458656e+06           --        Inf
0.45s
H       0         1        --          0  2.458656e+06  1.336510e+04  99.46%
0.45s
H       0         1        --          0  2.458656e+06  4.522766e+04  98.16%
0.45s
H       0         1        --          0  2.458656e+06  4.601476e+04  98.13%
0.45s
H       0         1        --          0  2.458656e+06  4.671084e+04  98.10%
```

0.45s								
H	0	1	--	0	2.458656e+06	4.675787e+04	98.10%	
0.45s								
H	0	1	--	0	2.458656e+06	4.718300e+04	98.08%	
0.45s								
H	0	1	--	0	2.458656e+06	4.767244e+04	98.06%	
0.45s								
H	0	1	--	0	2.458656e+06	4.771918e+04	98.06%	
0.45s								
	0	1	--	67	6.395103e+04	4.771918e+04	25.38%	
0.55s								
H	0	1	--	67	6.395103e+04	4.980345e+04	22.12%	
0.55s								
H	0	1	--	67	6.395103e+04	5.308718e+04	16.99%	
0.72s								
H	0	1	--	67	6.395103e+04	5.592398e+04	12.55%	
0.72s								
H	0	1	--	67	6.395103e+04	5.983272e+04	6.440%	
0.72s								
H	0	1	--	67	6.395103e+04	6.119020e+04	4.317%	
0.72s								

Time	Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap
H	0	1	--	67	6.395103e+04	6.119249e+04	4.314%
0.72s							
H	0	1	--	67	6.395103e+04	6.150086e+04	3.831%
0.72s							
H	0	1	--	67	6.395103e+04	6.150315e+04	3.828%
0.72s							
H	0	1	--	67	6.395103e+04	6.188376e+04	3.233%
0.72s							
H	0	1	--	67	6.395103e+04	6.232717e+04	2.539%
1.02s							
H	0	1	--	67	6.395103e+04	6.234157e+04	2.517%
1.02s							
	0	1	--	49	6.351750e+04	6.234157e+04	1.851%
1.16s							
	0	1	--	48	6.351750e+04	6.234157e+04	1.851%
1.16s							
	0	1	--	43	6.340810e+04	6.234157e+04	1.682%
1.18s							
	0	1	--	39	6.340810e+04	6.234157e+04	1.682%
1.18s							

	Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap
Time							
1.21s	0	1	--	36	6.336034e+04	6.234157e+04	1.608%
1.21s	0	1	--	36	6.336034e+04	6.234157e+04	1.608%
1.24s	0	1	--	36	6.335273e+04	6.234157e+04	1.596%
1.27s	0	1	--	51	6.332606e+04	6.234157e+04	1.555%
1.27s	0	1	--	49	6.332606e+04	6.234157e+04	1.555%
1.30s	0	1	--	67	6.330284e+04	6.234157e+04	1.519%
1.30s	0	1	--	67	6.330284e+04	6.234157e+04	1.519%
1.34s	0	1	--	36	6.329020e+04	6.234157e+04	1.499%
1.36s	0	1	--	51	6.326107e+04	6.234157e+04	1.454%
1.39s	0	1	--	31	6.324007e+04	6.234157e+04	1.421%
1.41s	0	1	--	36	6.323914e+04	6.234157e+04	1.419%
1.41s	0	1	--	35	6.323914e+04	6.234157e+04	1.419%
1.43s	0	1	--	31	6.321958e+04	6.234157e+04	1.389%
1.43s	0	1	--	31	6.321958e+04	6.234157e+04	1.389%
1.45s	0	1	--	34	6.321685e+04	6.234157e+04	1.385%
1.45s	0	1	--	34	6.321685e+04	6.234157e+04	1.385%
1.48s	0	1	--	39	6.320504e+04	6.234157e+04	1.366%
1.48s	0	1	--	39	6.320504e+04	6.234157e+04	1.366%
1.51s	0	1	--	42	6.320493e+04	6.234157e+04	1.366%
1.54s	0	1	--	28	6.319313e+04	6.234157e+04	1.348%
				60			

Time	Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap
1.54s	0	1	--	27	6.319313e+04	6.234157e+04	1.348%
1.57s	0	1	--	22	6.314879e+04	6.234157e+04	1.278%
1.57s	0	1	--	21	6.314879e+04	6.234157e+04	1.278%
1.60s	0	1	--	34	6.314879e+04	6.234157e+04	1.278%
1.62s	0	1	--	19	6.314879e+04	6.234157e+04	1.278%
1.65s	0	1	--	21	6.314879e+04	6.234157e+04	1.278%
1.67s	0	1	--	22	6.314879e+04	6.234157e+04	1.278%
1.69s	0	1	--	23	6.314510e+04	6.234157e+04	1.273%
1.70s	0	1	--	24	6.313769e+04	6.234157e+04	1.261%
1.71s	0	1	--	23	6.313769e+04	6.234157e+04	1.261%
1.73s	0	1	--	23	6.313769e+04	6.234157e+04	1.261%
1.75s	0	1	--	26	6.313769e+04	6.234157e+04	1.261%
1.77s	0	1	--	26	6.313769e+04	6.234157e+04	1.261%
1.78s	0	1	--	31	6.313752e+04	6.234157e+04	1.261%
1.80s	0	1	--	30	6.313752e+04	6.234157e+04	1.261%
Time	Nodes	Active	LPit/n	IntInf	BestBound	BestSolution	Gap
1.81s	0	1	--	31	6.313752e+04	6.234157e+04	1.261%
1.82s	0	1	--	33	6.313752e+04	6.234157e+04	1.261%
1.83s	0	1	--	31	6.313716e+04	6.234157e+04	1.260%
1.84s	0	1	--	30	6.313413e+04	6.234157e+04	1.255%
				61			
	0	1	--	33	6.313398e+04	6.234157e+04	1.255%

```
1.86s
*      0      1      --      33  6.313398e+04  6.302754e+04  0.169%
1.93s
      1      1   694.0      33  6.309169e+04  6.302754e+04  0.102%
1.94s
      1      1   694.0      33  6.309169e+04  6.302754e+04  0.102%
1.94s

Best solution : 63027.544850000
Best bound    : 63091.687150000
Best gap      : 0.1017%
Solve time    : 1.95
Solve node    : 1
MIP status    : solved
Solution status : integer optimal (relative gap limit 0.01)

Violations     : absolute      relative
  bounds        :          0          0
  rows          :          0          0
  integrality   :          0

求解完成，耗时：2.07 秒
求解状态：1
最优目标值：63027.54 元
优化成功！最大净收益：63027.54 元
排片计划已保存到 df_result_2_copt.csv
总共安排了 48 场放映
```