南开大学
Nankai University

南　　开　　大　　学

大数据计算及应用课程报告

推荐系统

鲁恒泽　　2212878

姚知言　　2211290

刘存　　2213050

2025 年 6 月 8 日

目录

一、 实验要求

任务：预测 Test.txt 文件中各对（u, i）的评分。
数据集：

1. Train.txt，用于训练你的模型。
2. Test.txt，用于测试。
3. ResultForm.txt，这是你的结果文件的格式。数据集的格式在 DataFormatExplanation.txt 中有说明。

在本项目中，我们需要报告 Test.txt 文件中未知对（u, i）的预测评分。可以使用在课程中学到的或从其他资源中学到的任何算法。小组需要撰写一份关于这个项目的报告。报告应包括但不限于以下内容：

- 数据集的基本统计信息（例如，用户数量、评分数量、项目数量等）；
- 算法的详细信息；
- 推荐算法的实验结果（均方根误差、训练时间、空间消耗）；
- 算法的理论分析和/或实验分析。

二、 推荐系统简介

在数字化时代，推荐系统已成为信息过滤和个性化服务的重要工具。它基于用户行为、偏好及物品特征等多维度数据，通过复杂算法为用户提供个性化推荐，帮助用户快速筛选出符合需求和兴趣的内容，提升信息检索效率和用户体验。

推荐系统在电子商务、社交媒体、在线视频、新闻资讯等领域广泛应用。在电商领域，它依据用户购买和浏览行为推荐商品，提高销售量和转化率；在社交媒体中，推荐可能感兴趣的朋友、话题和内容，增强用户互动；在在线视频领域，根据观看历史和喜好推荐视频，提升观看体验和用户粘性。

(一) 数据来源

推荐系统的数据来源是其构建和运行的基础，数据的多样性和质量直接影响推荐结果的准确性和有效性。数据主要来源于用户行为记录和物品特征信息。

用户行为数据是推荐系统的关键数据来源之一，涵盖用户在平台上的各种交互行为，如浏览记录、购买行为、评分、点赞和评论等。浏览记录反映用户对不同内容的关注程度和兴趣方向，购买行为体现用户的消费偏好和实际需求，评分和评论数据揭示用户对物品的主观评价，为推荐系统提供用户画像素材，帮助精准把握用户兴趣和偏好。

物品特征信息也是推荐系统的重要数据来源，包括物品的基本属性、内容描述、分类标签和发布时间等。在电商领域，商品属性如品牌、价格、规格和材质等；在内容推荐领域，如视频或文章，特征信息包括主题、风格、作者和时长等。这些特征信息帮助推荐系统理解物品本质属性和价值，实现基于内容的推荐，为用户提供与过去喜欢物品相似的新内容。

此外，推荐系统还会利用外部数据源来丰富数据维度。社交网络数据提供用户之间的关系信息，帮助发现用户群体间的潜在兴趣传播路径；地理信息数据用于基于位置的推荐，提供与用户地理位置相关的物品或服务；公开数据集和第三方数据服务可提供额外的用户行为模式或物品特征信息，进一步提升推荐的准确性和个性化程度。

(二) 常见算法

1. 基于用户的协同过滤方法

核心原理 首先我们对基于用户的协同过滤的核心原理进行介绍。

基于用户的协同过滤与基于物品的协同过滤思想类似，与基于物品的协同过滤的不同的是，基于用户的协同过滤主要依靠用户和用户之间的相似性来工作。

我们往往认为，喜欢或讨厌相同物品的用户具有一定的相似性。例如，如果用户甲既喜欢物品 A 也喜欢物品 B，而用户乙喜欢物品 A，那么用户乙喜欢物品 B 的概率就相对比喜欢其他物品的概率高一些。

实现流程 接下来我们简要介绍基于用户的协同过滤的实现流程，我们同样假设在数据分析之前，已经通过某种方法获取了用户对物品的评分数据。

以下是实现流程：

1. 基于用户对物品的评分数据，计算用户与用户之间的相似度（可采用余弦相似度、皮尔逊相关系数等方法）。具体而言，对于用户 A 和用户 B，找出他们共同评价过的物品集合，基于这些物品的评分计算两者的相似度，从而构建用户-用户相似度矩阵。

2. 基于用户-用户相似度矩阵进行推荐。对于目标用户，首先找出与其最相似的 K 个用户（称为近邻用户），然后聚合这些近邻用户对未评分物品的评价（可结合相似度加权），计算出目标用户对每个未评分物品的预测评分。如有需要，我们可以选择预测评分最高，且用户尚未进行评分的 Top-N 个物品进行推荐。

一些问题及解决方案 在实际应用中，基于用户的协同过滤也面临一些挑战：

1. 数据稀疏性问题：用户-物品评分矩阵通常非常稀疏，很多用户之间没有共同评价过的物品。解决方案包括：
   - 把二者的相似度记为 0。
   - 通过均值等方式对样本进行填充。
   - 构建更复杂的相关计算。

2. 冷启动问题：新用户由于缺乏历史行为数据难以找到相似用户。解决方案包括：
   - 结合内容信息进行混合推荐。
   - 利用用户统计学信息（性别、年龄、地区等）进行初始化相似推荐。

2. 基于物品的协同过滤算法

核心原理 在这里我们介绍下基于物品的协同过滤的核心原理。

首先我们简要叙述下**协同过滤**的原理，我们知道” 物以类聚、人以群分”，如果两个用户在过去有相似的行为，他们在未来可能也会喜欢相似的东西；如果一个用户喜欢某些物品，那么他也可能喜欢和这些物品相似的其他物品。

而**基于物品的协同过滤**其实核心在于：用户喜欢某一个物品，那么他很有可能喜欢与这个物品类似的物品，我们基于这样的方法进行推荐即可。

实现流程 接下来我们简要介绍下基于物品的协同过滤的流程，首先我们假设已经具有用户对物品的评分，这在实际场景中是较难获取的，因为我们很难让用户手动给出大量的评分，这是不现实的，我们可能需要通过一系列的推算，如基于浏览、收藏、购买等进行合理的计算，为简单起见这里我们不考虑这些因素，即已有用户对物品评分。

接下来简单介绍下实现的流程：

1. 基于用户对物品的评分，我们计算出物品和物品之间的相似度 (如余弦相似度、皮尔逊相关系数等)，具体方法是：我们选择对于物品 a 和物品 b 均进行打分的用户，基于他们给出的得分计算出相似度，即可得到物品-物品的相似度矩阵。
2. 基于物品-物品相似度矩阵，我们可以进行推荐。假设用户 A 看过物品 X，并对其评分较高。我们就可以去查找与物品 X 相似的物品 (如 Y、Z)，并查看这些物品与 X 的相似度，结合用户 A 对 X 的评分和相似度计算一个推荐分数，接下来基于计算出的推荐分数，按照得分从高到低，推荐 Top-N 个物品。

一些问题及解决方案 首先我们注意到：在一些情况下，我们很难保证有用户对两个物品均进行打分，也就是说在现实情况中，我们有很多物品，并不能保证有用户对某两个物品均有打分，这是不现实的，所以我们对于这种情况需要进行一些特殊处理，主要的解决方案有：

1. 相似度为 0 或不计算
2. 引入“填充”或“默认值”
3. 引入“邻域扩展”或“隐式反馈”：也就是说就算两个用户没对同样物品打过分，他们有类似行为也可以看作“相似”

3. 基于 SVD 的推荐算法

基于奇异值分解（SVD）的推荐算法是一种经典的矩阵分解方法，广泛应用于协同过滤推荐系统中。它通过分解用户-物品评分矩阵，提取用户和物品的潜在特征，从而实现个性化推荐。

核心原理 SVD 的核心思想是将用户-物品评分矩阵 R 分解为三个矩阵的乘积：

$$R \approx U\Sigma V^T \tag{1}$$

其中，R 是一个 $m \times n$ 的矩阵，表示 m 个用户对 n 个物品的评分；U 是一个 $m \times k$ 的矩阵，表示用户在 k 维潜在特征空间中的表示；Σ 是一个 $k \times k$ 的对角矩阵，对角线上的元素是奇异值；V 是一个 $n \times k$ 的矩阵，表示物品在 k 维潜在特征空间中的表示。通过选择前 k 个最大的奇异值及其对应的奇异向量，可以近似重构原始评分矩阵，从而实现降维和噪声过滤。

实现流程 基于 SVD 的推荐算法的实现流程主要包括以下步骤：

1. **数据预处理**：对用户-物品评分矩阵 R 进行预处理，如填充缺失值、标准化等。
2. **矩阵分解**：对评分矩阵 R 进行 SVD 分解，得到 U、Σ 和 V。
3. **降维**：选择前 k 个最大的奇异值及其对应的奇异向量，构造降维后的矩阵 U_k、Σ_k 和 V_k。
4. **评分预测**：通过降维后的矩阵重构评分矩阵，预测用户对未评分物品的评分：

$$\hat{R} = U_k \Sigma_k V_k^T \tag{2}$$

5. **推荐生成**：根据预测评分，为每个用户生成推荐列表。

一些问题及解决方案 尽管基于 SVD 的推荐算法在理论上具有良好的性能，但在实际应用中仍面临一些问题：

- **数据稀疏性**：用户-物品评分矩阵通常是高度稀疏的，导致矩阵分解时信息不足。解决方案包括引入正则化项，防止过拟合；采用矩阵填充技术，如基于用户或物品的均值填充。
- **冷启动问题**：新用户或新物品缺乏足够的评分数据，难以进行有效的推荐。解决方案包括为新用户或新物品分配默认特征向量，或结合内容信息进行混合推荐。
- **计算复杂度**：SVD 分解的计算复杂度较高，尤其是对于大规模数据集。解决方案包括采用随机 SVD 算法，通过随机采样减少计算量；利用分布式计算框架，如 Apache Spark，进行并行计算。

基于 SVD 的推荐算法通过矩阵分解提取用户和物品的潜在特征，能够有效处理用户和物品之间的复杂关系，为用户提供个性化的推荐服务。尽管存在一些问题，但通过合理的优化和改进，可以显著提升推荐系统的性能和用户体验。

三、 数据集介绍

本实验使用的数据集主要有两个，分别为：train.txt 和 test.txt，简单介绍及数据格式如下：

1. train.txt：用于训练模型，具体数据格式如下：

   train.txt：

   <user id>|<numbers of rating items>

   <item id> <score>

2. test.txt：用于测试，即需要进行预测的部分，具体数据格式如下：

   test.txt：

   <user id>|<numbers of rating items>

   <item id>

由此可知，train.txt 是我们本次实验的主要数据来源，我们需要基于 train.txt 中的数据，基于已有的不同用户对于不同物品打分情况进行模型训练，test.txt 是我们本次实验构建模型需要去预测的数据，即对于给定的用户对于指定物品的评分预测，由于我们的 test.txt 中的数据并没有真实情况作为对照，所以我们在构建模型时，需要去将 train.txt 进行划分，一部分作为训练数据，另一部分作为我们训练时的测试集，从而便于我们进行模型优化。我们可以基于划分后的数据集进行训练，并使用划分后的测试集进行验证，

(一) 整体数据统计

首先我们针对这两个数据集进行整体上的信息统计，我们可以基于宏观上的统计信息，统计出整体数据集的特征，便于我们进一步优化推荐系统。

首先我们统计出数据集中的一些宏观信息，如表1：

train 中用户 id 范围	1-610
train 中用户数	598
test 中用户 id 范围	1-610
test 中用户数	610
数据集中商品 id 范围	1-193565
train 中商品数	9077
test 中商品数	3618
train 中的评分总数	90854
train 中的平均 score	69.8821

表 1: 宏观数据统计

结合上述数据分析可以，总结如下:

1. 用户 id 最小为 1, 最大为 610, 但是值得注意在数据集 train.txt 中，所有打分用户仅有 598 个，说明存在用户并未进行打分，而在测试 test.txt 中，需要预测的用户为 610, 表示所有用户都需要进行预测，说明我们存在一些 zero-term 的用户样本，我们需要进行考虑。

2. 商品 id 最小为 1, 最大为 193565，同样的，在数据集 train.txt 中，打分的商品有 9077 个商品，同样我们知道在 test.txt 中，存在一些商品没有任何用户对其给出打分。

3. 在 train.txt 中，评分总数为 90854, 平均得分为 69.8821，我们需要基于这 598 名用户给出的 90854 条评分，进行数据集和测试集的划分并进行模型构建。

通过上面的分析,我们可以计算出用户打分-效用矩阵的大小应该是 610*193565 = 118074650, 但 train.txt 中评分总数为 90854，只占整个效用矩阵的约 0.0769%，由此可见效用矩阵非常稀疏，因此在保存效用矩阵时也应该用稀疏的方式进行保存，以节省计算资源。

(二) 具体数据分析

接下来我们针对用户的具体打分情况，给出简单的分析。

1. 用户给分情况分布

我们基于 python 代码统计出用户具体的打分情况，主要如图1。

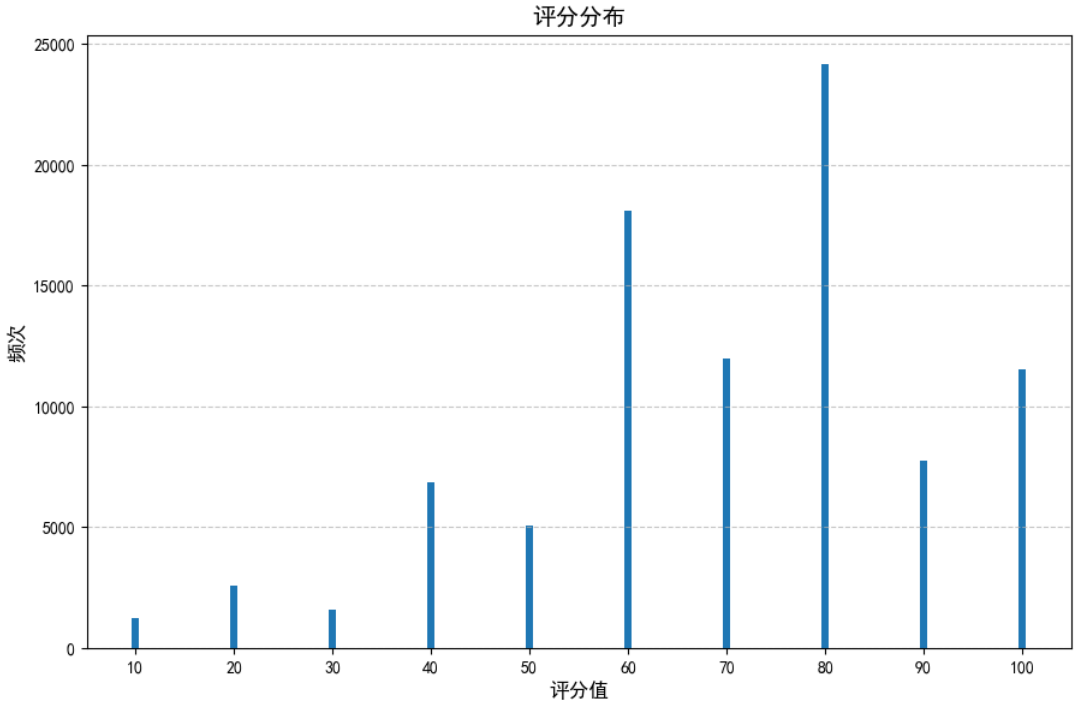

图 1: 用户打分情况分布图

统计得知，评分分布情况如下:

评分	次数	百分比
10	1221	1.34%
20	2552	2.81%
30	1603	1.76%
40	6852	7.54%
50	5067	5.58%
60	18119	19.94%
70	11996	13.20%
80	24177	26.61%
90	7742	8.52%
100	11525	12.69%

表 2: 评分分布统计表

可以知道，绝大多数用户给出的得分均为 60 及以上，占总打分数的约 81%。整体的打分情况并没有呈现出某种特定的分布特征，宏观来看，用户给出得分绝大多数集中在 60 分以上，对于商品给出很低得分情况相对较少，并且没有用户打出 0 分。

2. 用户给分均值分布

接下来我们给出单个用户给分的均值分布分析，如图2。

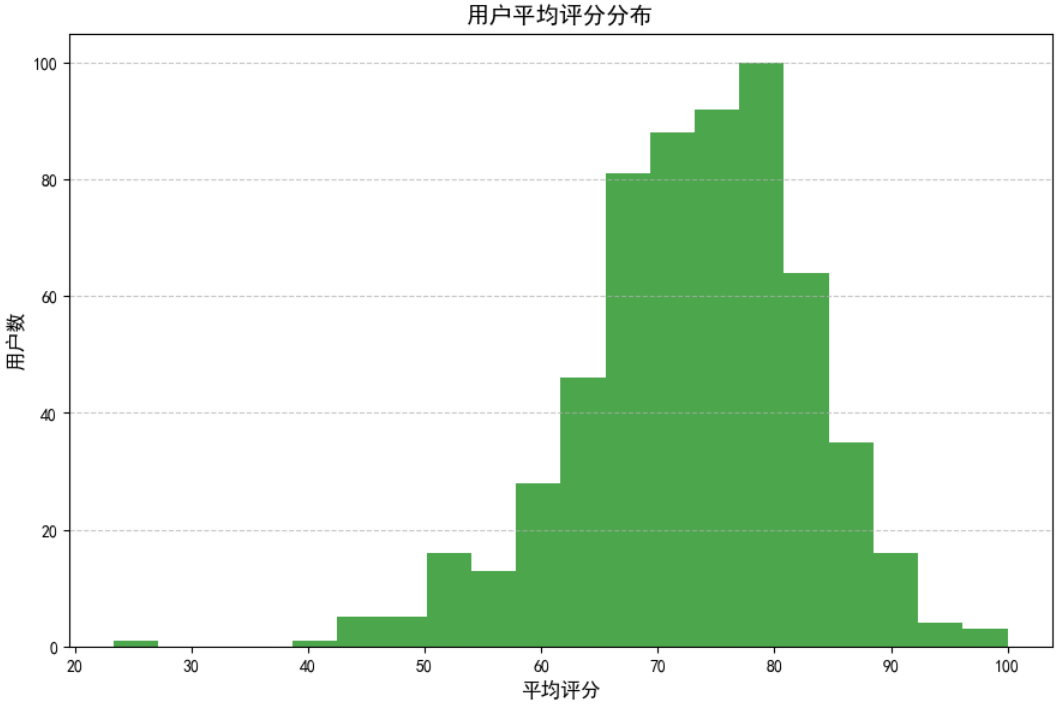

图 2: 单用户打分均值图

统计量化数据为:

用户平均评分: 最低 23.33, 最高 100.00, 平均 72.95

可以看到，绝大多数用户平均打分均位于 60 分以上，即给出的打分相对较高，对于所评分的商品具有较高的评价。

3. 商品得分情况分布

接下来我们由商品角度进行分析，对于商品的得分进行分析，统计图如图3

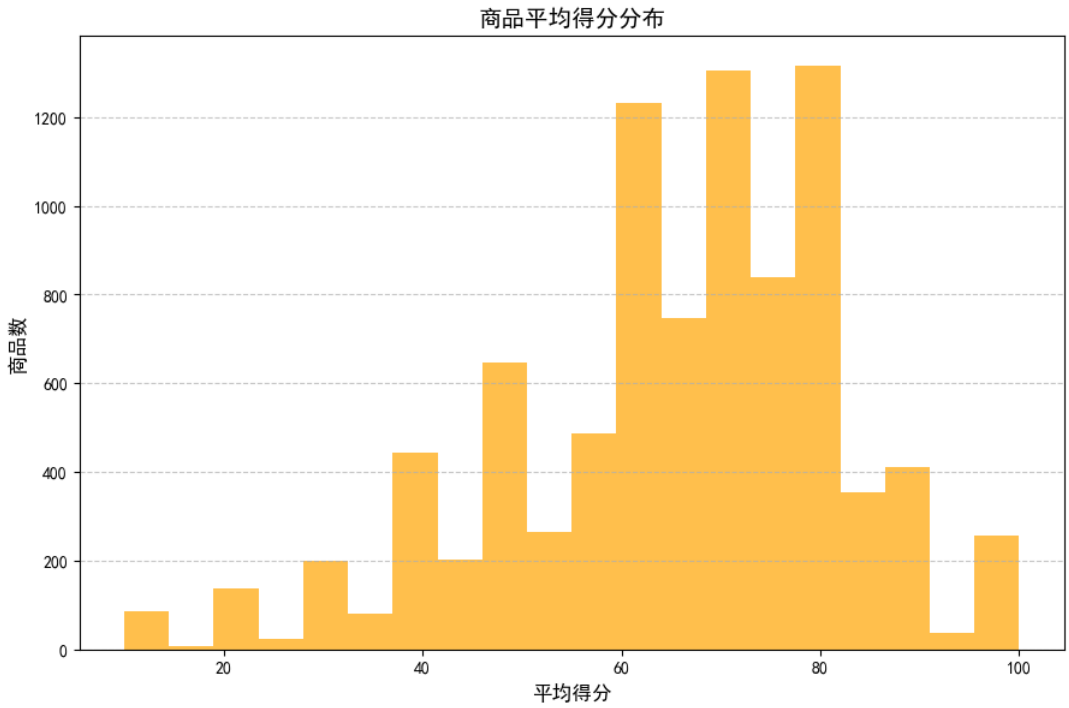

图 3: 商品得分图

统计量化数据为:

商品平均得分: 最低 10.00, 最高 100.00, 平均 65.33

可以看到，绝大多数商品得分均位于 60 分及以上，若以 60 作为及格标准，可以看到绝大多数商品还是得到了及格以上的得分的，用户对于商品的容忍度相对较高，但仍存在一些商品得到了较低的得分，这并不和实际情况相违背，故这些数据相对而言较为真实，不应作为极端数据进行剔除。

四、 算法原理与代码细节

(一) 划分数据集

训练集统计: 用户数: 598 物品数: 9077 评分数: 82698

验证集统计: 用户数: 581 物品数: 2839 评分数: 8156

(二) 基于用户的协同过滤

1. 算法原理

在基于用户的协同过滤中,

基于用户的协同过滤 (User-Based CF) 的核心假设是: **相似用户对物品的评分行为相似**。算法分为两个阶段:

- **训练阶段**: 计算用户之间的相似度矩阵
- **预测阶段**: 基于相似用户的评分预测目标用户对未评分物品的评分

训练阶段 训练阶段的核心是计算用户相似度矩阵 S，其中 S_{uv} 表示用户 u 和用户 v 的相似度。在这一部分中，实现了中心化余弦相似度和皮尔森相关系数两种相似度计算。

- 中心化余弦相似度

  1. 中心化评分：

  $$r'_{u,i} = r_{u,i} - \bar{r}_u \tag{3}$$

  其中 $\bar{r}_u$ 是用户 u 的平均评分：

  $$\bar{r}_u = \frac{1}{|I_u|} \sum_{i \in I_u} r_{u,i} \tag{4}$$

  2. 填充缺失值为 0：

  $$r''_{u,i} = \begin{cases} r'_{u,i} & \text{如果} r_{u,i} \text{ 存在} \\ 0 & \text{否则} \end{cases} \tag{5}$$

  3. 计算余弦相似度：

  $$S_{uv} = \frac{\sum_{i \in I} r''_{u,i} \cdot r''_{v,i}}{\sqrt{\sum_{i \in I} (r''_{u,i})^2} \sqrt{\sum_{i \in I} (r''_{v,i})^2}} \tag{6}$$

  优势：中心化处理能够较好的反映用户的整体偏好，解决用户个体之间喜欢打高分/低分的差异。

  潜在的问题：对未评分处补充 0 会导致未评分物品与该用户打出平均分的物品在比较中不会体现出差异。

- 皮尔逊相关系数

  皮尔逊相关系数直接通过公式计算即可，在计算中仅考虑两用户均进行打分的物品。

  $$S_{uv} = \frac{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{v,i} - \bar{r}_v)^2}} \tag{7}$$

  优势：皮尔逊相关系数仅对两用户共同打分的项目进行比较，不会受到用户打分缺失的影响。

  潜在的问题：很多用户之间并没有均打分的物品，导致相关系数结果为 NaN。

预测阶段 在预测阶段中，基于已有的评分信息，对未知评分信息（如用户 u 对物品 i 的评分 $\hat{r}_{u,i}$）进行预测。

1. 选择近邻用户：
   - 根据上一步构建的相似度矩阵，找到对物品 i 评过分的用户集合 U_i
   - 从中选择与用户 u 最相似的 k 个用户 $N_u(i)$

2. 加权聚合评分：

   根据挑选出来的 k 个用户的归一化评分按照相似度权重修正用户 u 的平均评分，得到最终结果。

   在这里，我们设计了不进行任何额外操作的基础版，以及两个变体版本：

   - 变体 1：删除筛选结果中相似度小于 0 的邻居。

     Idea：若负相关用户对结果无参考价值，通过这种方式可以更加有效的筛选出邻居。
   - 变体 2：对于相似度小于 0 的邻居，对其权重和归一化评分均取相反数进行后续计算。

     Idea：若负相关样本评分可提供反向参考，基础版本因为正样本和负样本可能会互相稀释权重，导致最终权重计算并不是我们想要的，所以做出调整。

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in N_u(i)} S_{uv} \cdot (r_{v,i} - \bar{r}_v)}{\sum_{v \in N_u(i)} |S_{uv}|} \tag{8}$$

   其中：

   - $\bar{r}_u$ 是用户 u 的平均评分
   - $r_{v,i} - \bar{r}_v$ 是用户 v 对物品 i 的归一化评分
   - S_{uv} 是用户 u 和 v 的相似度

3. 评分调整：

   根据我们的数据集可知，评分范围为 10-100，因此在最终评分预测中，我们也将超出范围的预测评分调整到范围中最接近预测值的结果。

$$\hat{r}_{u,i} = \max(10, \min(100, \hat{r}_{u,i})) \tag{9}$$

2. 代码实现

UserBasedCF 类 构建 UserBasedCF 类，管理算法函数，在不同函数中传递变量。

```
class UserBasedCF:
    def __init__(self):
        self.user_similarity = None
        self.train_data = None
        self.user_item_matrix = None
        self.mean_user_rating = None
```

训练函数 训练函数由于为余弦相似度和皮尔逊相关系数分配进行了实现，但主要逻辑相似，对于两函数中的独有实现，我会在下面说明。

首先需要读取先前处理好的训练数据，并构造用户-物品矩阵。

然后计算每个用户的平均评分，根据平均评分对矩阵进行中心化。注意：

- 这里计算平均评分，并对矩阵进行中心化时，并没有使用未打分（NaN）值。
- 余弦相似度通过该方式进行中心化，而皮尔森相似度计算虽然并不需要依托于中心化后的数据，但在此仍需要进行该计算，因为后续预测阶段需要使用中心化的评分。

```
self.train_data = train_data

# 创建用户–物品评分矩阵
self.user_item_matrix = self.train_data.pivot_table(
    index='user', columns='item', values='score'
)

# 计算每个用户的平均评分
self.mean_user_rating = self.user_item_matrix.mean(axis=1)

# 中心化评分
ratings_diff = self.user_item_matrix.sub(self.mean_user_rating, axis=0)
```

接下来，进行相似度的计算。

- 中心化的余弦相似度

  由于调用余弦相似度的计算不能存在 nan 值，所以在此为将 nan 值都处理为 0 计算。

```
# 将NaN替换为0用于相似度计算
ratings_diff_filled = ratings_diff.fillna(0)

# 计算余弦相似度
self.user_similarity = cosine_similarity(ratings_diff_filled)
self.user_similarity = pd.DataFrame(
    self.user_similarity,
    index=self.user_item_matrix.index,
    columns=self.user_item_matrix.index
)
```

- 皮尔逊相关系数

  通过调用 corr 方法完成皮尔逊相关系数的计算。

```
    self.user_similarity = self.user_item_matrix.T.corr(method='pearson')
```

最后，保存原始评分矩阵和中心化评分矩阵，便于后续预测阶段使用。

```
# 保存原始评分矩阵
self.original_ratings = self.user_item_matrix.copy()
# 保存中心化评分矩阵
self.ratings_for_prediction = ratings_diff
```

预测函数 预测函数包括基础班 predict,以及两个变体版本 predict_variant1 和 predict_variant2，其基础思想已在原理部分解释，在后续变体版本需要额外添加逻辑处，将会进行详细说明。

预测函数接收带预测的用户和物品 id，以及 k 值（表示使用 k 个邻居的相似度预测）。首先如果该用户或者物品未出现在训练集中（新用户或新物品），则直接返回全局平均分。

```
def predict(self, user_id, item_id, k=5):
```

```
    if user_id not in self.user_item_matrix.index or item_id not in self.
        user_item_matrix.columns:
        return self.mean_user_rating.mean()  # 返回全局平均分
```

然后，我们首先获取目标用户的平均评分。然后筛选对该物品评过分的用户，如果没有用户评分，则返回用户平均分。

```
    # 获取目标用户的平均评分
    mean_rating = self.mean_user_rating[user_id]

    # 获取对该物品评过分的用户
    rated_users = self.original_ratings.index[
        self.original_ratings[item_id].notna()
    ]

    if len(rated_users) == 0:
        return mean_rating  # 如果没有用户评过分，返回用户平均分
```

然后获取这些用户与目标用户的相似度，选择最相似的 k 个用户。

```
    # 获取目标用户与这些用户的相似度
    sim_scores = self.user_similarity.loc[user_id, rated_users]

    # 获取最相似的k个用户
    top_k_users = sim_scores.nlargest(k).index
    top_k_sim = sim_scores[top_k_users]
```

- 原始方法：直接获取用户归一化评分。

```
    top_k_ratings_diff = self.ratings_for_prediction.loc[top_k_users, item_id
        ]
```

- 变体 1：删除负相关邻居后获取用户归一化评分。

```
    positive_sim_mask = top_k_sim > 0
    top_k_sim = top_k_sim[positive_sim_mask]

    # 如果删除后没有剩余用户，返回用户平均分
    if len(top_k_sim) == 0:
        return mean_rating

    filtered_users = top_k_sim.index
    top_k_ratings_diff = self.ratings_for_prediction.loc[filtered_users,
        item_id]
```

- 变体 2：对负相关邻居的权重和归一化评分取反。

```
    negative_sim_mask = top_k_sim < 0
    top_k_sim[negative_sim_mask] = -top_k_sim[negative_sim_mask]

```

```
    # 获取这些用户对该物品的归一化评分，并对负相关用户取相反数
    top_k_ratings_diff = self.ratings_for_prediction.loc[top_k_users, item_id
        ].copy()
    top_k_ratings_diff[negative_sim_mask] = -top_k_ratings_diff[
        negative_sim_mask]
```

通过这些用户的归一化评分以及相似度（作为权重），得到最终的预测得分。

```
    weighted_sum = (top_k_ratings_diff * top_k_sim).sum()
    if top_k_sim.sum() != 0:
        predicted_rating = mean_rating + weighted_sum / top_k_sim.sum()
    else:
        predicted_rating = mean_rating
```

最终，为确保评分在合理范围内，我们将评分调整到 10-100 之间，并返回。

```
    # 确保评分在合理范围内
    predicted_rating = max(10, min(100, predicted_rating))
    return predicted_rating
```

评价函数 评价函数根据参数选择 k 值和预测函数，对测试集中的每条数据调用预测函数进行预测，然后计算 RMSE（均方根误差）和 MAE（平均绝对误差）以评估模型性能，并计算真值和预测值平均数。最终将预测结果保存到 csv 中（可选）。

```
def report(self, valid_set, output_file="evaluation_report.csv",save = False,
    k=5, predict_func = "normal"):
    report_df = valid_set.copy()

    # 为每条记录生成预测值
    report_df['predicted_score'] = report_df.apply(
        lambda row: self.predict_func[predict_func](row['user'], row['item'],
            k),
        axis=1
    )

    # 计算评估指标
    metrics = {
        'RMSE': np.sqrt(mean_squared_error(report_df['score'], report_df['
            predicted_score'])),
        'MAE': mean_absolute_error(report_df['score'], report_df['
            predicted_score']),
        'Avg_Actual': report_df['score'].mean(),
        'Avg_Predicted': report_df['predicted_score'].mean()
    }

    # 保存到CSV
    if save:
        output_path = output_file
        report_df.to_csv(output_path, index=False)
```

```
        print(f"评估报告已保存到: {output_path}")

    # 打印指标
    print("\n===== 模型评估指标 =====")
    for k, v in metrics.items():
        print(f"{k}: {v:.4f}")

    return metrics
```

(三) 基于物品的协同过滤算法

首先我们回顾下基于物品的协同过滤的算法原理，并介绍下在本次实验中我们的设计思路，再给出具体的实现细节。

1. 算法原理及设计思路

在上文中我们已经介绍了基于物品的协同过滤，其核心即为：我们认为，如果用户喜欢一个物品，那么其有很大概率会喜欢与这个物品相似的物品，那么首先我们需要了解的是：

1. 怎么确定两个物品是否相似，相似度如何?
2. 怎么基于相似性去推荐物品，这里即预测对于某用户未进行打分物品的得分情况?

我们首先需要解决上述两个问题，

1. 首先，对于相似度的计算，由于在本次实验中，我们的数据集中是直接给出用户对物品的打分的，所以我们不需要去进行其他的处理了，可以直接基于评分进行计算，具体即为，我们可以基于评分进行计算相似度，我们基于物品的得分，基于如余弦相似度、皮尔逊相似度等进行相似度计算，即可得到相似度矩阵。
2. 其次，我们需要基于相似度矩阵进行计算得到预测得分，这里我们首先找到相似度矩阵中，和我们需要预测的物品相似度最高的 k 个物品的相似度，然后通过公式：

$$\hat{r}_{ui} = \mu + b_u + b_i + \frac{\sum_{j \in N_k(i)} s_{ij} \cdot (r_{uj} - \mu - b_u - b_j)}{\sum_{j \in N_k(i)} s_{ij}}$$

进行计算，接下来我们分别介绍下其中的变量表示的含义：

(a) μ：表示全局均值，也就是对于所有存在得分的物品，进行平均处理得到的均分。

(b) b_u：表示对于用户的评分偏置，反映其评分高低倾向，也就是如果其给出的得分普遍偏高或者普遍偏低，那么其给出的得分就不完全准确客观，我们需要进行修正。

(c) b_i：表示对于物品的评分偏执，反映其得分高低倾向。

(d) $\frac{\sum_{j \in N_k(i)} s_{ij} \cdot (r_{uj} - \mu - b_u - b_j)}{\sum_{j \in N_k(i)} s_{ij}}$：这个公式表示的是对于相似度最高的前 k 个物品，我们计算得分，并对于相似度做出加权。

我们将这些变量求和，于是得到了我们的预测评分。

于是我们了解到了如何解决这两个问题，接下来我们需要做的就是具体去计算和实现相关的算法，下面我们来详细介绍下算法的步骤：

1. 首先我们需要对于手中的数据进行预处理，从而便于我们读取得到用户对于物品的打分矩阵。

2. 然后我们读取用户对物品的打分，构建矩阵，并基于用户对物品的打分，我们可以计算出物品之间的相似度，也就是当对两个物品评分均很高的时候，我们可以认为这两个物品相似度较高。

3. 计算得到物品之间的相似度矩阵后，我们可以基于相似度矩阵去进行预测，如果用户有过评分，那么我们可以在用户已评分的物品中，找到与被预测物品最相似的 k 个物品，并基于这 k 个物品，计算出残差，并基于相似度进行加权，从而得到公式中的最后一个部分。

更为具体的细节处理我们在下一部分结合代码实现进行介绍。

2. 代码实现

这里我们和基于用户的协同过滤保持一致, 主要分为四个模块: fit、predict、evaluate、report, 接下来我们分别进行介绍:

1. ItemBaseCF 类: 我们在这里保存一些信息，包括用户-物品矩阵，物品-物品相似度矩阵等。

```python
class ItemBasedCF:
    def __init__(self):
        self.item_similarity = None
        self.train_data = None
        self.user_item_matrix = None
        self.mean_item_rating = None
        self.global_mean = 0
        self.user_bias = None
        self.item_bias = None
```

2. 数据预处理、训练函数 fit: 这里我们计算的主要是物品与物品之间的相似度，以及我们在这里计算出用户偏置和物品偏置，从而在后续直接使用即可，首先我们给出具体代码:

```python
def fit(self, train_data, similarity_method='pearson'):
        """使用训练集训练模型（仅对有评分的位置进行计算）"""
        self.train_data = train_data

        # 创建用户–物品评分矩阵
        self.user_item_matrix = self.train_data.pivot_table(
            index='user', columns='item', values='score'
        )

        # 计算每个物品的平均评分（可用于冷启动）
        self.mean_item_rating = self.user_item_matrix.mean(axis=0)

        ## 中心化评分（按列中心化）
        ratings_diff = self.user_item_matrix.sub(self.mean_item_rating, axis
            =1)

        self.global_mean = self.train_data['score'].mean()
```

```
17
18          # 用0填充NaN用于相似度计算
19          ratings_diff_filled = ratings_diff.fillna(self.global_mean)
20
21          # 转置: 物品为行, 用户为列
22          if similarity_method == 'pearson':
23              self.item_similarity = ratings_diff_filled.T.corr(method='pearson
                    ')
24          # 计算物品相似度矩阵 (基于列)
25          elif similarity_method == 'cosine':
26              self.item_similarity = cosine_similarity(ratings_diff_filled.T)
27          self.item_similarity = pd.DataFrame(
28              self.item_similarity,
29              index=self.user_item_matrix.columns,
30              columns=self.user_item_matrix.columns
31          )
32
33          # 用户偏置
34          self.user_bias = self.user_item_matrix.sub(self.mean_item_rating,
                axis=1).mean(axis=1).fillna(0)
35
36          # 物品偏置
37          self.item_bias = self.mean_item_rating - self.global_mean
38
39          print("\n模型训练完成")
40          print(f"用户数量: {len(self.user_item_matrix.index)}")
41          print(f"物品数量: {len(self.user_item_matrix.columns)}")
42
43          self.original_ratings = self.user_item_matrix.copy()
44          self.ratings_for_prediction = ratings_diff
```

这里我们首先基于训练数据集得到用户-物品评分矩阵，然后先求出每个物品的平均评分，其实这里是为了当一个物品刚加入时，新物品刚加入系统，还没有被用户评分过，系统无法判断它适合谁，所以给出一个初始均值，使其能够作为保底预测。

接下来我们进行中心化后，得到相似度，这里的相似度计算我们尝试了两种，分别是**余弦相似度**和**皮尔逊相关系数**。这里我们分别介绍下这两者：

(a) 余弦相似度：

$$\text{sim}_{\cos}(i,j) = \frac{\sum\limits_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum\limits_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} r_{uj}^2}}$$

优点在于：计算简单高效，对于用户评分标准差异不是特别敏感。

问题主要在于：忽略用户评分倾向，比如有的用户普遍打高分，这里并不能进行处理。

(b) 皮尔逊相关系数：

$$\text{sim}_{\text{pearson}}(i,j) = \frac{\sum\limits_{u \in U_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum\limits_{u \in U_{ij}} (r_{ui} - \bar{r}_i)^2} \cdot \sqrt{\sum\limits_{u \in U_{ij}} (r_{uj} - \bar{r}_j)^2}}$$

优点在于：能抵消评分倾向影响（如有的人爱打低分，有的人爱打高分），更能反映“评分模式”的相似性，对不同用户尺度具有标准化能力。

问题主要在于：对极端值敏感（如某个评分远高远低），对于未评分部分不做考虑。

此后我们基于用户平均打分对于全局平均的偏差计算出用户偏置，基于物品平均打分对于全局平均的偏差计算出物品偏置，从而得到两个结果。

3. 预测函数 predict：这里我们在 fit 中已经计算出了物品之间的相似度矩阵，所以我们在这里只需要计算即可，首先我们需要判断该用户是否有评分，如果其没有对任何物品给出评分，那么我们没有办法做出预测，所以我们返回全局均值，

```
def predict(self, user_id, item_id, k=5, abs = False):
    if user_id not in self.user_item_matrix.index or item_id not in self.
        user_item_matrix.columns:
                return self.mean_item_rating.mean()  # 全局平均分
```

接下来我们需要去基于该用户评分物品中，与其最相似的 k 个，然后基于这 k 个物品，基于相似度计算加权。

```
# 获取该用户已评分的物品
user_rated_items = self.original_ratings.loc[user_id].dropna()

if len(user_rated_items) == 0:
    return self.mean_item_rating.get(item_id, self.mean_item_rating.mean())

# 取出相似度
sim_scores = self.item_similarity.loc[item_id, user_rated_items.index]

# 选取最相似的K个物品
top_k_items = sim_scores.nlargest(k).index
top_k_sim = sim_scores[top_k_items]
top_k_ratings = self.ratings_for_prediction.loc[user_id, top_k_items]

# 测试sim先求绝对值
if abs:
    top_k_sim = top_k_sim.abs()

# 添加用户偏置和物品偏置
base_score = self.global_mean
base_score += self.user_bias.get(user_id, 0)
base_score += self.item_bias.get(item_id, 0)

# 不使用偏置
# base_score = self.mean_item_rating.get(item_id, self.global_mean)

weighted_sum = (top_k_sim * top_k_ratings).sum()
if top_k_sim.sum() != 0:
    pred = base_score + weighted_sum / top_k_sim.sum()
else:
```

```
        pred = base_score

    # 限制在合理范围
    pred = max(10, min(100, pred))
    return pred
```

这里我们基于公式:

$$\hat{r}_{ui} = \mu + b_u + b_i + \frac{\sum_{j \in N_k(i)} s_{ij} \cdot (r_{uj} - \mu - b_u - b_j)}{\sum_{j \in N_k(i)} s_{ij}}$$

计算预测评分，并最终对于评分归一化到 10 到 100 之间。由于上文中我们已经介绍了每个部分的含义，这里不再赘述。

4. 评价函数 report：这里我们基于传入的集合，对于每个用户-物品进行预测评分，并计算出 RSME、MAE 等用于评估模型的能力。同时将结果进行输出，便于进行查看。

```
def report(self, valid_set, output_file="itemcf_evaluation_report.csv", k =
    5, abs = False):
    report_df = valid_set.copy()

    report_df['predicted_score'] = report_df.apply(
        lambda row: self.predict(row['user'], row['item']), axis=1
    )

    metrics = {
        'RMSE': np.sqrt(mean_squared_error(report_df['score'], report_df['
            predicted_score'])),
        'MAE': mean_absolute_error(report_df['score'], report_df['
            predicted_score']),
        'Avg_Actual': report_df['score'].mean(),
        'Avg_Predicted': report_df['predicted_score'].mean()
    }

    report_df.to_csv(output_file, index=False)
    print(f"评估报告已保存到: {output_file}")

    print("\n===== 模型评估指标 =====")
    for k, v in metrics.items():
        print(f"{k}: {v:.4f}")

    return metrics
```

(四) 基于 SVD 的推荐算法

与传统的用户协同过滤和物品协同过滤模型相比，SVD 模型通过引入隐因子和偏置项，能够更有效地捕捉用户和物品之间的复杂关系，从而提高预测的准确性和泛化能力。

1. 实现思路

本系统采用的奇异值分解 (SVD) 模型与经典协同过滤方法存在本质差异，其理论创新性体现在以下三个层面：

模型架构差异

- **邻域方法**依赖显式相似度计算：

$$\hat{r}_{ui}^{\text{UserCF}} = \bar{r}_u + \frac{\sum_{v \in \mathcal{N}_k(u)} \text{sim}(u,v)(r_{vi} - \bar{r}_v)}{\sum_{v \in \mathcal{N}_k(u)} |\text{sim}(u,v)|} \tag{10}$$

其中 $\mathcal{N}_k(u)$ 表示用户 u 的 k 近邻集合。

- *SVD* **模型**通过潜在空间映射建立预测函数：

$$\hat{r}_{ui}^{\text{SVD}} = \mu + b_u + b_i + \mathbf{p}_u^\top \mathbf{q}_i \tag{11}$$

其中 $\mathbf{p}_u, \mathbf{q}_i \in \mathbb{R}^k$ 为 k 维隐特征向量。

优化目标差异

传统方法采用局部启发式更新，而 SVD 构建全局优化问题：

$$\min_{\mathbf{P},\mathbf{Q},\mathbf{b}} \sum_{(u,i)\in\mathcal{O}} (r_{ui} - \hat{r}_{ui})^2 + \underbrace{\lambda_1 \|\mathbf{P}\|_F^2 + \lambda_2 \|\mathbf{Q}\|_F^2}_{\text{隐因子正则}} + \underbrace{\lambda_3 \|\mathbf{b}_u\|^2 + \lambda_4 \|\mathbf{b}_i\|^2}_{\text{偏置项正则}} \tag{12}$$

其中 $\mathcal{O}$ 为观测评分集合，正则化项防止过拟合。

特征交互方式

SVD 模型通过矩阵分解捕获高阶交互效应：

$$\mathbf{R} \approx \mathbf{P}\mathbf{Q}^\top = \sum_{d=1}^{k} \sigma_d \mathbf{u}_d \mathbf{v}_d^\top \tag{13}$$

其中 σ_d 为奇异值，$\mathbf{u}_d, \mathbf{v}_d$ 为奇异向量。

2. 代码实现

基于前述理论框架，系统实现的关键代码如下，其技术要点与理论设计的对应关系如下：

模型架构实现

```python
class SVDModel:
    def __init__(self, factors=15):
        # 隐因子维度k=15
        self.P = np.random.randn(n_users, factors) * 0.01  # 用户矩阵初始化
        self.Q = np.random.randn(n_items, factors) * 0.01  # 物品矩阵初始化
        self.bu = np.zeros(n_users)  # 用户偏置项
        self.bi = np.zeros(n_items)  # 物品偏置项
        self.global_mean = np.mean(train_ratings)  # 全局均值
```

对应理论公式 (2) 的实现：

- 用户/物品隐因子矩阵 $\mathbf{P}, \mathbf{Q}$ 采用 $\mathcal{N}(0, 0.01)$ 初始化
- 偏置项 $\mathbf{b}_u, \mathbf{b}_i$ 初始化为零向量

- 全局均值 μ 从训练数据直接计算

优化目标实现

```
def update(self, u, i, r, lambda_p=0.1, lambda_q=0.1, lambda_b=0.01):
    # 计算预测值（对应公式2）
    pred = self.global_mean + self.bu[u] + self.bi[i] + np.dot(self.P[u],
        self.Q[i])

    # 计算误差
    e = r - pred

    # 参数更新（对应公式3）
    self.P[u] += lr * (e * self.Q[i] - lambda_p * self.P[u])
    self.Q[i] += lr * (e * self.P[u] - lambda_q * self.Q[i])
    self.bu[u] += lr * (e - lambda_b * self.bu[u])
    self.bi[i] += lr * (e - lambda_b * self.bi[i])
```

实现特点:

- 分离的正则化系数 $\lambda_p, \lambda_q, \lambda_b$ 对应公式 (3) 的 $\lambda_1, \lambda_2, \lambda_3$
- 采用随机梯度下降 (SGD) 更新参数
- 学习率 lr 按 $\eta_t = \eta_0/\sqrt{t}$ 衰减

特征交互实现

```
def predict_all(self):
    # 矩阵乘积实现高阶交互（对应公式4）
    full_pred = self.global_mean + self.bu[:,None] + self.bi[None,:] + np
        .dot(self.P, self.Q.T)
    return np.clip(full_pred, rating_min, rating_max)
```

创新性实现细节:

动态正则化: 根据训练轮次调整正则化强度

```
lambda_p = 0.1 * (1 + epoch / epochs)  # 随训练轮次增加
```

早停机制: 监控验证集损失

```
if val_loss > best_loss * 1.001:  # 容忍0.1%的波动
    early_stop_counter += 1
```

数值稳定性:

```
pred = np.clip(pred, 1.0, 5.0)  # 限制预测范围
```

五、 实验结果分析

(一) 实验结果展示

我们节选了部分结果在此展示，更详细的文件，我们放在了 best_prediction.txt 中。

```
1|20
260     100.0
1927     93.96577301179968
804     86.71094270583195
3     84.61438574856419
...省略

3|8
2288     50.59646394808111
5048     25.81597614320306
4518     38.48264280986973
```

(二) 基于用户的协同过滤

1. 不同相似度计算在不同 K 值下的对比

在基于用户的协同过滤中，使用了两种相似度计算方法（中心化的余弦相似度，皮尔逊相关系数）。同时，在协同过滤中，选择合理的近邻数量也非常重要。

因此，我们进行了对比实验：在两种相似度计算方法下，设置 K 值为 1 到 20，进行性能评估，结果如图4所示。

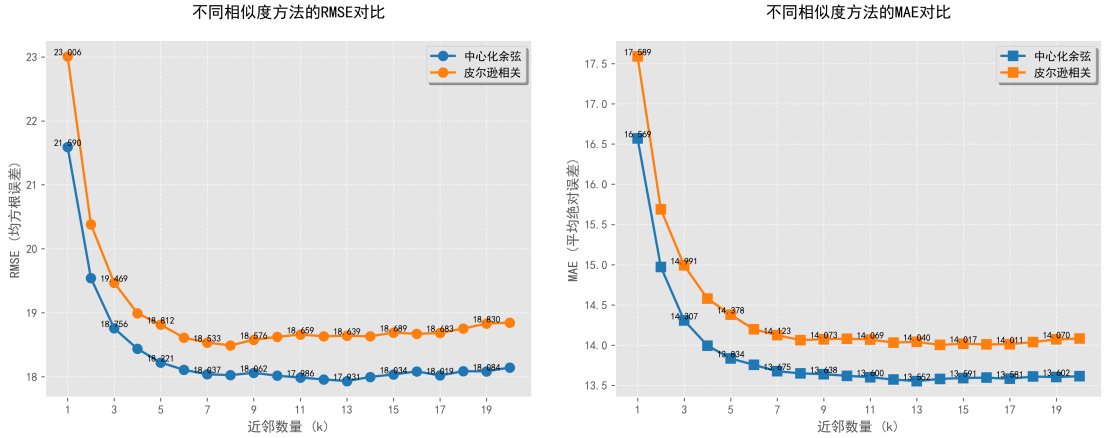

图 4: 不同相似度计算在不同 K 值的对比

从图4中可以发现，中心化余弦相似度的结果全面领先于皮尔逊相关系数，这可能是由于样本过于稀疏导致皮尔逊相关系数产生了过多了 NaN 造成近邻不足，而中心化余弦相似度的近邻较多，利用更多数据达到了更好的拟合效果。而随着 K 值增大，RMSE 和 MAE 均呈现先大幅减少后缓慢增加的趋势，这表明在选择 K 值过多时，会导致相关度较低的邻居引入计算，从而降低模型性能。

在这一组实验中，最终表现最好的是在 K=13 下的中心化余弦相似度方法。其在训练集上的 RMSE 为 17.931，MAE 为 13.552。

2. 不同预测函数变体在不同 K 值下的对比

在基于用户的协同过滤实践中，除了基础的预测逻辑，对负相关用户的不同处理策略可能影响模型性能。我们设计了两种预测函数变体（删除负相关用户、反转负相关用户评分及相似度），

探究其在不同近邻数量（K 值）下的表现。

依托该思想开展对比实验：基于中心化余弦相似度，首先依旧设置 K 值为 1 到 20，对原始预测方法、两种变体方法进行性能评估，结果如图 5 所示。

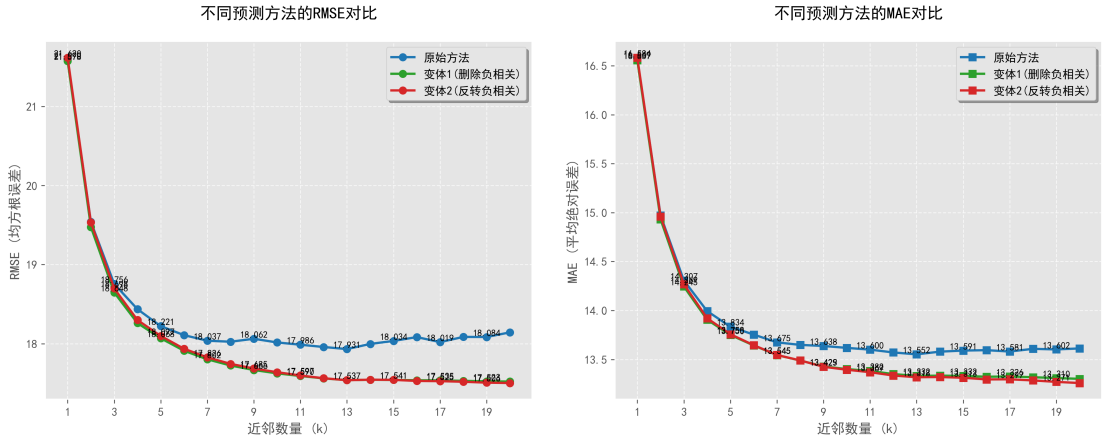

图 5: 不同预测函数变体在不同 K 值的对比

从图中可以看出，随 K 值增大，两种函数变体相较于原始方法均有较为明显的提升，且两种变体性能表现差距不大。

这说明先前未能合理处理负相关样本的干扰对预测准确率有较大的影响，删除负相关样本和反转负相关样本能够很好的提升模型性能。

性能表现最好的点在变体 2 中，K=20 时，RMSE 为 17.5013，MAE 为 13.2588。且似乎随 K 值进一步增大依旧有性能提升的趋势。

因此，我们进行一组 K 值 5-50 的粗粒度分析，如图6所示。

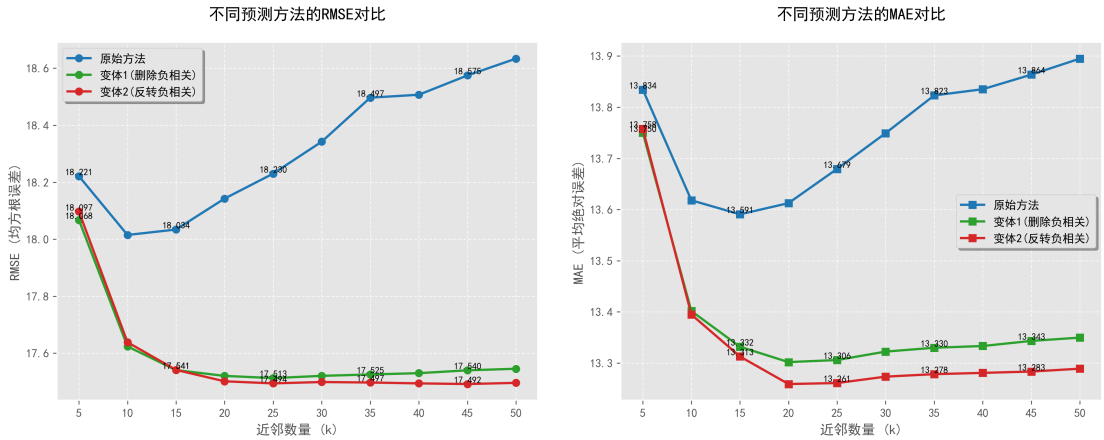

图 6: 不同预测函数变体在不同 K 值的对比（进一步分析）

两个变体方法随 K 值增加，MAE 在 K=20 左右到达最低点后有一定的上升趋势。在 RMSE 上，变体 1 的随 K 值增加也略有上升，而变体 2 似乎逐渐趋于稳定。

从该图中可以看出，随着 K 值增大，相关度为负值的样本随之增加，变体 2 相较于变体 1 的优势逐渐得以体现。因此，负例样本对于结果的估计是非常具有指导意义的。

图中 RMSE 最低点是 K=45 时的变体 2，17.4917；而 MAE 最低点是 K=25 时的变体 2，13.2588。

(三) 基于物品的协同过滤

1. 不同相似度计算方法在不同 K 值下对比

在基于物品的协同过滤中，我们尝试使用**余弦相似度**和**皮尔逊相关系数**进行计算相关性，很显然当我们基于前 k 个物品相似度及得分计算预测得分的时候，这里选择多少个物品是很重要的，所以我们在此进行一起的实验对比。

首先我们较为粗糙的进行设置，这里首先设置 K 的值在 5, 10, 15, 20, 25, 30, 40, 50 中，先初步确定我们的 K 值范围，后续我们再范围内进行更细致的对比分析。如图7

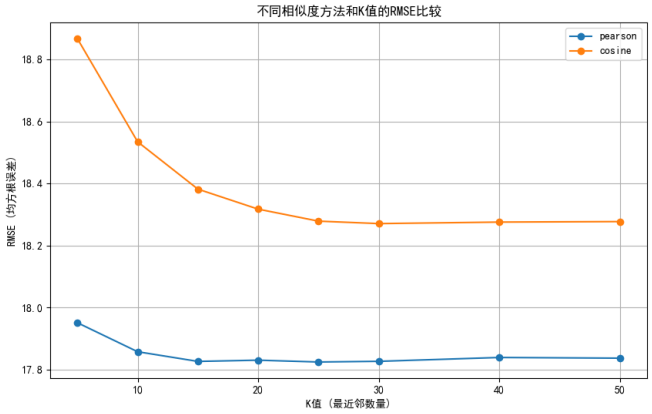

图 7: 不同相似度计算方法下不同 K 值下对比图

接下来我们在 10-30 之间，步长为 1 逐步增大 K 值，并进行更加细致的计算，具体结果如图8

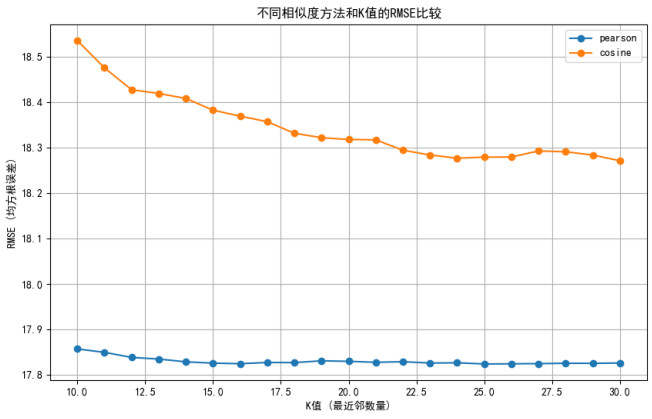

图 8: 不同相似度计算方法下不同 K 值下具体对比图

我们可以看到最佳组合为：相似度方法 =pearson, K=25, RMSE=17.8242

2. 预测分数时，对于相似度分母求和是否取绝对值的分析

在调研公式时，我们发现有部分公式会在取值时先进行绝对值，这样可以避免负相关的情况，我们针对我们的数据集进行实验，并针对不同的 K 值进行分析。主要针对的即为预测分数计算

中:

$$\frac{\sum_{j\in N_k(i)} s_{ij}\cdot(r_{uj}-\mu-b_u-b_j)}{\sum_{j\in N_k(i)} s_{ij}}$$

和

$$\frac{\sum_{j\in N_k(i)} s_{ij}\cdot(r_{uj}-\mu-b_u-b_j)}{\sum_{j\in N_k(i)} |s_{ij}|}$$

中，是否对于分母考虑取绝对值，也就是不考虑正负相关性的问题进行探究。

结果如图9

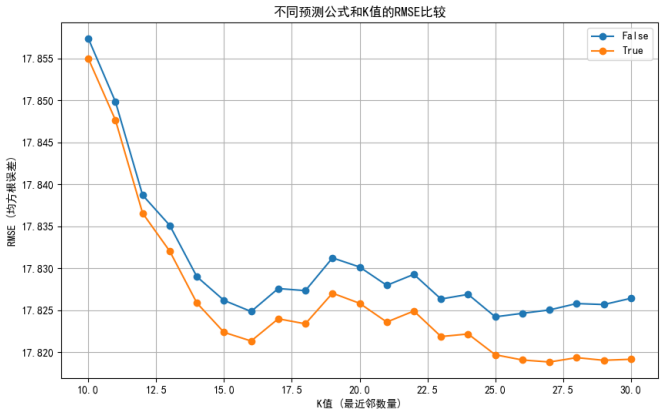

图 9: 不同计算公式下不同 K 值下具体对比图

可以看到在这里如果我们取出绝对值后，整体的 RMSE 更低，说明我们采取绝对值的方法进行计算，效果相对更好，在 K=27 时，达到 RMSE=17.8188，从理论上进行分析，这样的结果也是合理的，因为当我们取绝对值时，不管是正相关程度很高还是负相关程度很高，其本质上都是很重要的，但如果将其进行加和，可能会导致结果偏差甚至失真。

(四) SVD 模型的参数影响

本研究通过控制变量实验系统评估了 SVD 模型中关键超参数的影响机制，实验结果揭示以下规律:

学习率动态衰减机制

固定其他参数（$factors = 15, \lambda = 0.25$），考察学习率 η 的影响。采用指数衰减策略（其中衰减系数 $\alpha = 0.95$）:

$$\eta_t = \eta_0 \times \alpha^t \tag{14}$$

我们发现 $\eta = 0.005$ 时取得最优收敛性，此时验证集 RMSE 最低位 17.23。

表 3: 学习率影响

α	最终 RMSE	收敛轮次
0.01	17.7057	1
0.0075	17.3833	2
0.005	17.2395	3
0.002	17.3667	6
0.0015	17.4087	9
0.001	17.4301	16
0.0005	17.5001	90

正则化强度优化本研究采用分级正则化策略，分别控制用户偏置 (λ_{ub})、物品偏置 (λ_{ib})、用户隐因子矩阵 (λ_P) 和物品隐因子矩阵 (λ_Q) 的正则化强度。通过网格搜索得到的优化配置及影响机制如下:

表 4: 分级正则化参数优化结果

λ_{ub}	λ_{ib}	λ_P	λ_Q	Val. RMSE
0.5	0.5	0.4	0.4	18.88
0.20	0.20	0.15	0.15	17.45
0.25	0.25	0.20	0.20	17.32
0.30	0.30	0.25	0.25	17.89
0.15	0.25	0.10	0.20	17.67

实验发现:

- 偏置项正则化需强于隐因子矩阵（$\lambda_{ub}, \lambda_{ib} > \lambda_P, \lambda_Q$），最优比值为 1.25:1
- 当 $\lambda_{P/Q} > 0.25$ 时出现明显欠拟合
- 用户侧正则化敏感性高于物品侧，λ_{ub} 变化对 RMSE 影响幅度达 $\pm 3.7\%$

最优参数组合的理论解释:

$$\mathcal{L}\text{reg} = \underbrace{\lambda ub|\mathbf{b}u|^2 + \lambda ib|\mathbf{b}i|^2}_{\text{偏置正则}} + \underbrace{\lambda_P|\mathbf{P}|F^2 + \lambda Q|\mathbf{Q}|F^2}_{\text{矩阵正则}} \tag{15}$$

其中偏置项需要更强正则化以防止个体偏差主导预测结果，这与 Koren 提出的"asymmetric regularization" 原则一致。实验表明，当 $\lambda_{ub} = \lambda_{ib} = 0.25$，$\lambda_P = \lambda_Q = 0.2$ 时，验证集 RMSE 达到最低值 17.32。

隐因子维度选择

隐因子维度 k 的影响呈现非线性特征:

$$\text{RMSE}(k) = a - b\log(k) + \epsilon \quad (k \le k_{\text{opt}}) \tag{16}$$

实验数据表明:

- 当 $k = 12$ 时达到最优（RMSE=17.35），继续增大 k 导致过拟合，但是降低 K 也会导致效果不好。
- 计算复杂度 $\mathcal{O}(k^2)$ 增长，$k = 100$ 时训练时间达 $k = 15$ 的 7.3 倍

表 5: 隐因子维度影响

factor	最终 RMSE	收敛轮次
15	17.45	15
12	17.35	22
10	17.42	29
8	17.37	25

（五） 训练时间与空间消耗

我们对三种推荐算法（UserCF、ItemCF 和 SVD）的训练时间和内存消耗进行了对比测试，具体性能数据如下：

UserCF 性能

```
[UserCF] 时间: 0.3279s, 内存峰值: 177.17MB
```

该算法计算用户相似度矩阵时表现出高效性（0.33 秒），但内存消耗较高（177MB）。这种特性源于用户规模通常小于物品规模，但用户-用户相似度矩阵仍需较大存储空间。

ItemCF 性能

```
[ItemCF] 时间: 14.2280s, 内存峰值: 887.81MB
```

物品协同过滤表现出显著更高的计算成本（14.23 秒）和内存需求（888MB）。这是因为：

- 物品数量通常远多于用户（如电商场景）
- 物品-物品相似度矩阵的维度平方级增长
- 需要维护更庞大的共现矩阵

SVD 性能

```
═══ 性能报告 ═══
训练用时: 83.50秒
测试预测用时: 0.30秒
内存峰值: 29.63 MB
各阶段内存使用(MB):
初始化: 81.73 -> 140.58
加载数据: 140.58 -> 147.59
训练后: 147.59 -> 153.01
预测后: 153.01 -> 153.52
```

矩阵分解模型展现出独特优势：

- **内存效率最优**（峰值仅 29.6MB）
- **预测阶段极快**（0.3 秒）
- 训练时间较长（83.5 秒）源于迭代计算特征向量

通过数据可知：

- **实时性要求高**选 UserCF（最快训练）

- **内存受限**选 SVD（最省内存）
- ItemCF 适合物品数较少的场景

六、 总结

在本次实验中，我们按照从简单到复杂的顺序，较为系统地实现了推荐系统中的几种主流算法，包括基于用户的协同过滤算法、基于物品的协同过滤算法以及基于 SVD 的隐语义模型。通过对不同算法实现效果的分析与比对，我们不仅进一步加深了对推荐系统的理解，还提升了对相关知识的掌握程度。此外，实验过程也锻炼了小组成员之间的团结协作能力，以及动手实践的工程和科研能力。我们逐步认识到数据预处理和分析在处理大数据问题时的重要性，并且发现通过保存必要的参数，能够有效加速程序运行，节省算力资源。在本学期的学习过程中，我们初步掌握了利用常用大数据处理技术解决实际问题的能力，这为我们未来的学习和研究提供了一种重要的思维方法和有力工具，使我们获益匪浅。