華中科技大学

课 程 实 验 报 告

课程名称： 大数据分析

专业班级： 本硕博 2301 班

学　　号： **U202315763**

姓　　名： 王家乐

指导教师： 崔金华

报告日期： **2025.5.27**

计算机科学与技术学院

目录

实验五 推荐系统算法及其实现

1.1 实验目的

1、了解推荐系统的多种推荐算法并理解其原理。
2、实现 User-User 的协同过滤算法并对用户进行推荐。
3、实现基于内容的推荐算法并对用户进行推荐。
4、对两个算法进行电影预测评分对比
5、推荐算法中，利用 minhash 算法对效用矩阵进行降维处理

1.2 实验内容

给定 Recommend-data.zip 电影数据集，包含电影评分文件（ratings.csv，原始电影评分数据，仅供参考），训练集 train_set.csv 和测试集 test_set.csv（原始电影评分拆分出来的两个数据集），movies.csv 电影信息数据。

根据给定的数据集，完成以下推荐系统任务：

1）**基于用户-用户的协同过滤推荐算法。**对训练集中的评分数据构造用户-电影效用矩阵，使用 pearson 相似度计算方法计算用户之间的相似度，也即相似度矩阵。实现基于协同过滤的推荐系统算法，完成对单个指定的 userID 用户进行推荐，找到与其最相似的 k 个用户，用这 k 个用户的评分情况对当前用户的所有未评分电影进行评分预测，选取评分最高的 n 个电影进行推荐。此外，利用测试集数据，计算推荐算法的 SSE（误差平方和），在测试集中包含 100 条用户-电影评分记录，对测试集中的每个用户-电影需要计算其预测评分，再和真实评分进行对比，获得 SSE。

2）**基于用户-用户的协同过滤推荐优化方法。**此方法采用 minhash 算法对效用矩阵进行降维处理，从而得到相似度矩阵。注意 minhash 采用 jaccard 方法计算相似度，需要对效用矩阵进行 01 处理，这儿将 0.5-2.5 的评分置为 0，3.0-5.0 的评分置为 1。

3）**基于内容的推荐算法。**将数据集 movies.csv 中的电影类别作为特征值，计算这些特征值的 tf-idf 值，得到关于电影与特征值的 n（电影个数）*m（特征值个数）的 tf-idf 特征矩阵。根据得到的 tf-idf 特征矩阵，用余弦相似度的计算方法，得到电影之间的相似度矩阵。实现基于内容的推荐系统算法，完成对某个用户-电影进行预测评分时，获取当前用户的已经完成的所有电影的打分，通过电

影相似度矩阵获得已打分电影与当前预测电影的相似度，按照下列方式进行打分计算：

$$\text{score} = \frac{\sum_{i=1}^{n} score^{'}(i) * sim(i)}{\sum_{i=1}^{n} sim(i)}$$

其中，选取相似度大于零的值进行计算，如果已打分电影与当前预测用户-电影相似度大于零，加入计算集合，否则丢弃。相似度为负数的，强制设置为 0，表示无相关。假设计算集合中一共有 n 个电影，score 为我们预测的计算结果，score'(i)为计算集合中第 i 个电影的分数，sim(i)为第 i 个电影与当前用户-电影的相似度。如果 n 为零，则 score 为该用户所有已打分电影的平均值。要求能够对指定的 userID 用户进行电影推荐，推荐电影为预测评分排名前 k 的电影。userID 与 k 值可以根据需求做更改。此外，对测试集中对应的用户-电影进行预测评分，输出每一条预测评分，并与真实评分进行对比，计算获得 SSE 值。

4）基于内容的推荐优化方法。该方法使用 minhash 算法对基于内容推荐算法的相似度计算进行降维，把最小哈希的模块作为一种近似度的计算方式，从而得到相似度矩阵。注意 minhash 采用 jaccard 方法计算相似度，特征矩阵应为 01 矩阵，因此特征矩阵选取采用方式为，如果该电影存在某特征值，则特征值为 1，不存在则为 0，从而得到 01 特征矩阵。

备注：协同过滤算法和基于内容推荐算法都会涉及到相似度的计算，最小哈希算法在牺牲一定准确度的情况下对相似度进行计算，其能够有效的降低维数，尤其是对大规模稀疏 01 矩阵。同学们可以使用哈希函数或者随机数映射来计算哈希签名。哈希签名可以计算物品之间的相似度。最终降维后的维数等于我们定义映射函数的数量，我们设置的映射函数越少，整体计算量就越少，但是准确率就越低。大家可以分析不同映射函数数量下，最终结果的准确率有什么差别。

1.3 实验过程

1.3.1 编程思路

1. 基于用户-用户的协同过滤推荐算法

（1）数据加载：从 data/ 目录中加载 movies.csv, train_set.csv, test_set.csv 三个数据文件：movies.csv：电影信息；train_set.csv：训练集用户评分；test_set.csv：测试集用户评分。

（2）构建用户评分数据结构：user_ratings：{user_id: {movie_id: rating}}；user_avg：{user_id: 平均评分}。计算用户的平均评分：对于用户 u，其评分集合为 $R_u = \{r_{u,i}\}$，则平均评分为：$\overline{r}_u = \frac{1}{|R_u|}\sum_{i \in R_u} r_{u,i}$ 。

（3）用户评分归一化：对每个用户的评分进行中心化归一化处理（即每个评分减去用户的平均评分，$r'_{u,i} = r_{u,i} - \overline{r}_u$）。

（4）计算用户之间的相似度：对每对用户 u、v，计算他们对共同电影的评分余弦相似度。设共同评分电影集合为I_{uv}，则 $\mathrm{sim(u,v)} = \frac{\sum_{i \in I_{uv}} r'_{u,i} \cdot r'_{v,i}}{\sqrt{\sum_{i \in I_u} (r'_{u,i})^2} \cdot \sqrt{\sum_{i \in I_v} (r'_{v,i})^2}}$ 。

（5）构建用户相似度矩阵：计算所有用户两两之间的相似度，保存在 sim_matrix 中；每个用户按相似度排序其邻居。

（6）评分预测：对用户 u 和电影 i，使用其最相似的 k 个用户的评分预测该电影的评分。设邻居集合为$N_k(u)$，预测评分为：$\widehat{r_{u,i}} = \overline{r_u} + \frac{\sum_{v \in N_k(u)} sim(u,v) \cdot (r_{v,i} - \overline{r_v})}{\sum_{v \in N_k(u)} |sim(u,v)|}$ 。

（7）评估模型性能：对测试集中每个用户-电影对，计算平方误差；累加求出 SSE（$\mathrm{SSE} = \sum_{(u,i) \in \mathrm{TestSet}} (\widehat{r_{u,i}} - r_{u,i})^2$）。

2. 基于用户-用户的协同过滤推荐优化方法

（1）数据加载：从 data/ 目录中加载 movies.csv, train_set.csv, test_set.csv 三个数据文件：movies.csv：电影信息；train_set.csv：训练集用户评分；test_set.csv：测试集用户评分。

（2）构建用户评分数据结构：user_ratings：{user_id: {movie_id: rating}}；user_avg：{user_id: 平均评分}；binary_user_ratings：{user_id: 0/1}。评分二值化（用于后续 MinHash，规则：评分>=3.0 记作 1，否则记作 0）。

（3）生成 MinHash 签名：对每个用户的喜欢的电影集合（即 binary_rating == 1）生成 MinHash 签名，使用多个哈希函数$h_i(x) = (a_i \cdot x + b_i)\ mod\ p$，每个用户生成一个长度为 num_hashes 的签名向量，记录每个哈希函数作用下的最小哈希值，对每个用户 u，其签名为：$Sig_u = [\min_{x \in S_u} h_1(x), \ldots\ldots, \min_{x \in S_u} h_k(x)]$ 。

（4）计算用户相似度矩阵：使用 Jaccard 相似度即两个签名向量中相同位置相等的数量比上签名长度，$\mathrm{sim(u,v)} = \frac{1}{\mathrm{k}}\sum_{\mathrm{i}=1}^{\mathrm{k}} \phi[\, Sig_u[i] = Sig_v[i]\,]$ 。

（5）对于用户 u 未评分的电影 i，设邻居集合为N_u，则预测评分为：$\widehat{r_{u,i}} = \overline{r_u} + \frac{\sum_{v \in N_u} sim(u,v) \cdot (r_{v,i} - \overline{r_v})}{\sum_{v \in N_u} |sim(u,v)|}$。

（6）评估模型性能：对测试集中每个用户-电影对，计算平方误差；累加求出 SSE（$\mathrm{SSE} = \sum_{(\mathrm{u,i}) \in \mathrm{TestSet}} (\widehat{r_{u,i}} - r_{u,i})^2$）。

3. 基于内容的推荐算法

（1）数据加载：从 data/ 目录中加载 movies.csv, train_set.csv, test_set.csv 三个数据文件：movies.csv：电影信息；train_set.csv：训练集用户评分；test_set.csv：测试集用户评分。

（2）构建用户评分数据结构：user_ratings：{user_id: {movie_id: rating}}；user_avg：{user_id: 平均评分}。计算用户的平均评分：对于用户 u，其评分集合为 $\mathrm{R_u} = \{\mathrm{r_{u,i}}\}$，则平均评分为：$\overline{\mathrm{r}}_\mathrm{u} = \frac{1}{|\mathrm{R_u}|} \sum_{\mathrm{i} \in \mathrm{R_u}} \mathrm{r_{u,i}}$ 。

（3）构建 TF-IDF 向量：把 genres 按 | 分词（比如 "Action|Thriller" -> ["Action", "Thriller"]）；TF：$\mathrm{tf(t,d)} = \frac{f_{t,d}}{\sum_\mathrm{k} f_{t,d}}$，$f_{t,d}$ 是电影 d 中类型 t 出现的次数；IDF：$\mathrm{idf(t)} = \log(\frac{\mathrm{N}}{\mathrm{df(t)+1}})$，N 是总电影数，df(t)是类型 t 出现的电影数；TF-IDF 向量：$TF - IDF_{d,t} = tf(t,d) \cdot idf(t)$。

（4）计算余弦相似度：$\mathrm{sim(i,j)} = \frac{\vec{\mathrm{v_i}} \cdot \vec{\mathrm{v_j}}}{||\vec{\mathrm{v_i}}|| \cdot ||\vec{\mathrm{v_j}}||}$，$\vec{v_i}$ $\vec{v_j}$ 是电影 i、j 的 TF-IDF 向量，使用嵌套的循环的暴力计算（可替换为 sklearn 的 cosine_similarity）。

（5）评分预测：根据用户之前看过的电影及其评分，通过与目标电影的相似度加权平均，预测用户对未看电影的评分。$\widehat{r_{u,m}} = \frac{\sum_{i \in I_u} sim(i,m) \cdot r_{u,i}}{\sum_{i \in I_u} sim(i,m)}$，$I_u$为用户 u 看过的电影集合，sim(i,m)为电影 i 与目标电影 m 的相似度。

（6）模型评估：对测试集中每个用户-电影对，计算平方误差；累加求出 SSE（$\mathrm{SSE} = \sum_{(\mathrm{u,m}) \in \mathrm{TestSet}} (\widehat{\mathrm{r_{u,m}}} - \mathrm{r_{u,m}})^2$）。

4. 基于内容的推荐优化方法

（1）数据加载：从 data/ 目录中加载 movies.csv, train_set.csv, test_set.csv 三个数据文件：movies.csv：电影信息；train_set.csv：训练集用户评分；test_set.csv：测试集用户评分。

（2）构建用户评分数据结构：user_ratings：{user_id: {movie_id: rating}}；user_avg：{user_id: 平均评分}。计算用户的平均评分：对于用户 u，其评分集合为 $R_u = \{r_{u,i}\}$，则平均评分为：$\bar{r}_u = \frac{1}{|R_u|}\sum_{i \in R_u} r_{u,i}$ 。

（3）生成 MinHash 签名：对每个电影的特征集合生成 MinHash 签名，使用多个哈希函数$h_i(x) = (a_i \cdot x + b_i)\ mod\ p$，每个用户生成一个长度为 num_hashes 的签名向量，记录每个哈希函数作用下的最小哈希值，对每个电影 m，其签名为：$Sig_m = [\min_{f \in F_m} h_1(f), \ldots\ldots, \min_{f \in F_m} h_k(f)]$ 。

（4）计算电影相似度矩阵：使用 Jaccard 相似度即两个签名向量中相同位置相等的数量比上签名长度，$\mathrm{sim(A,B)} = \frac{1}{k}\sum_{i=1}^{k} \phi[Sig_A[i] = Sig_B[i]]$ 。

5）评分预测：根据用户之前看过的电影及其评分，通过与目标电影的相似度加权平均，预测用户对未看电影的评分。$\widehat{r_{u,m}} = \frac{\sum_{i \in I_u} sim(i,m) \cdot r_{u,i}}{\sum_{i \in I_u} sim(i,m)}$ ，I_u为用户 u 看过的电影集合，sim(i,m)为电影 i 与目标电影 m 的相似度。

（6）模型评估：对测试集中每个用户-电影对，计算平方误差；累加求出 SSE（$\mathrm{SSE} = \sum_{(u,m) \in TestSet} (\widehat{r_{u,m}} - r_{u,m})^2$）。

1.3.2 遇到的问题及解决方式

1. 数据稀疏性问题：

在构建用户-电影评分矩阵时发现，大部分用户只对少数电影进行了评分，导致评分矩阵非常稀疏，影响了相似度计算的准确性。

解决方式：

尝试对评分矩阵进行填补处理，但最终决定采用只计算有共同评分项的用户对或电影对，从而在一定程度上缓解了稀疏性影响。此外，使用最小哈希（MinHash）方法也有效减少了稀疏数据带来的计算开销。

2. MinHash **算法实现复杂度高：**

初次实现 MinHash 算法时，对哈希函数的构造方式不明确，导致签名结果不准确，进而影响了 Jaccard 相似度的计算。

解决方式：

通过查阅相关资料，明确了哈希函数的形式 h(x) = (a*x + b) % c，并设置多组互异的 a、b 参数，保证哈希函数的独立性。同时验证每组 MinHash 结果的合理性。

3. TF-IDF **向量构造及归一化：**

在基于内容的推荐算法中，对 genres 字段进行 TF-IDF 编码时发现，不同电影的类别数量不一，导致向量维度不一致。

解决方式：

统一所有电影的类别集合作为特征空间，并对每部电影建立与该特征空间对应的一致长度的 TF-IDF 向量，填充为 0 的项表示该电影不具备该特征。

4. 相似度矩阵计算耗时严重：

在计算所有用户之间、或所有电影之间的相似度矩阵时，算法时间复杂度高，尤其是使用余弦相似度计算时耗时更明显。

解决方式：

尝试利用向量化工具（如 NumPy）进行矩阵级计算，同时限制相似度矩阵中只保留 Top-K 相似项，从而加快整体处理速度。

1.3.3 实验测试与结果分析

1. 基于用户-用户的协同过滤推荐算法

代码运行结果如下：

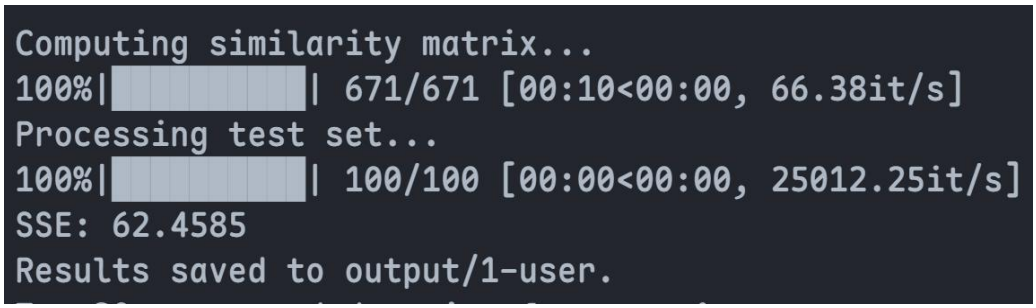


图 1-1 代码运行结果

为用户 1 选取最相近的 30 个用户推荐 10 部电影：

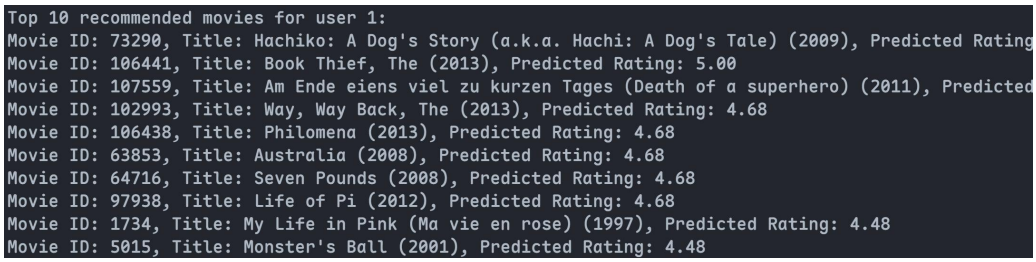


图 1-2 用户 1 推荐结果

对于测试集，取相似用户数量为 30，SSE 为 62.4585：

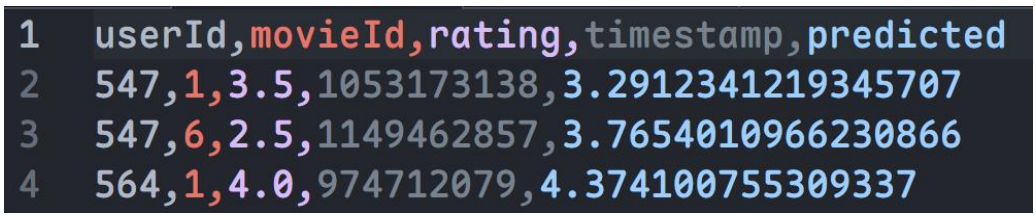


图 1-3 测试集结果

2. 基于用户-用户的协同过滤推荐优化方法

代码运行结果如下：

```
✓ import math ···

Creating MinHash signatures...
100%|██████████| 671/671 [00:02<00:00, 315.61it/s]
Computing similarity matrix...
100%|██████████| 671/671 [00:02<00:00, 226.92it/s]
Processing test set...
100%|██████████| 100/100 [00:00<00:00, 24881.68it/s]
SSE: 59.8800
Results saved to output/3-user-minhash.
```

图 2-1 代码运行结果

对于测试集，取相似用户数量为 30，SSE 为 59.8800，预期评分与实际评分较为接近：

```
1   userId,movieId,rating,timestamp,predicted
2   547,1,3.5,1053173138,3.7437278691832683
3   547,6,2.5,1149462857,3.548257726273467
4   564,1,4.0,974712079,3.9234362604982196
5   564,2,4.0,974839862,3.551982851018221
6   624,1,5.0,1019126661,3.504680404022951
7   624,2,3.0,1019125424,3.076440892046393
8   15,1,2.0,997938310,2.9347803969017052
9   15,2,2.0,1134521380,3.060658431020622
```

图 2-2 测试集结果

3. 基于内容的推荐算法

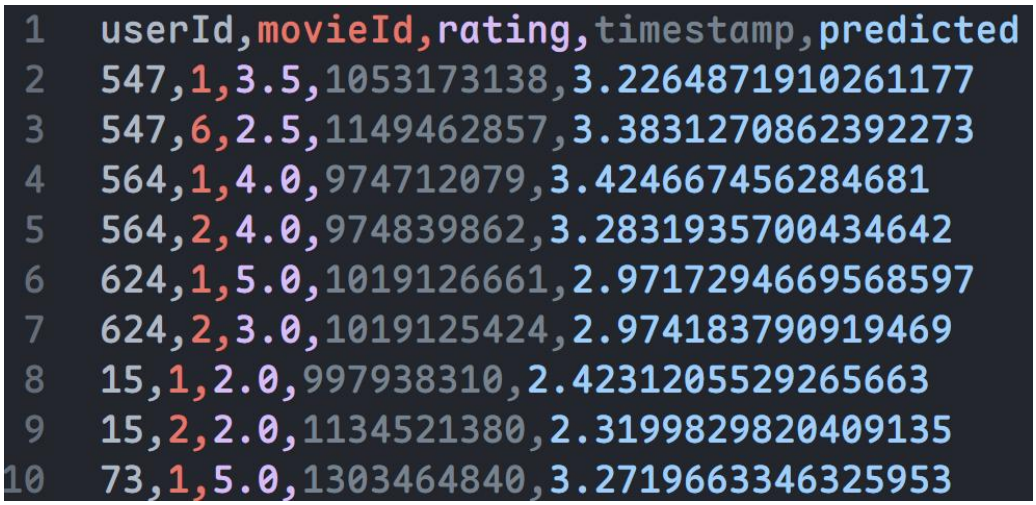
代码运行结果如下：

```
✓ import math ···

Computing TF-IDF matrix...
100%|██████████| 9125/9125 [00:00<00:00, 621880.67it/s]
Computing cosine similarity matrix...
100%|██████████| 9125/9125 [02:06<00:00, 72.11it/s]
Processing test set...
100%|██████████| 100/100 [00:00<00:00, 2038.31it/s]
SSE: 67.1192
Results saved to output/2-content.
```

图 3-1 代码运行结果

对于测试集，SSE 为 67.1192，预期评分与实际评分较为接近：

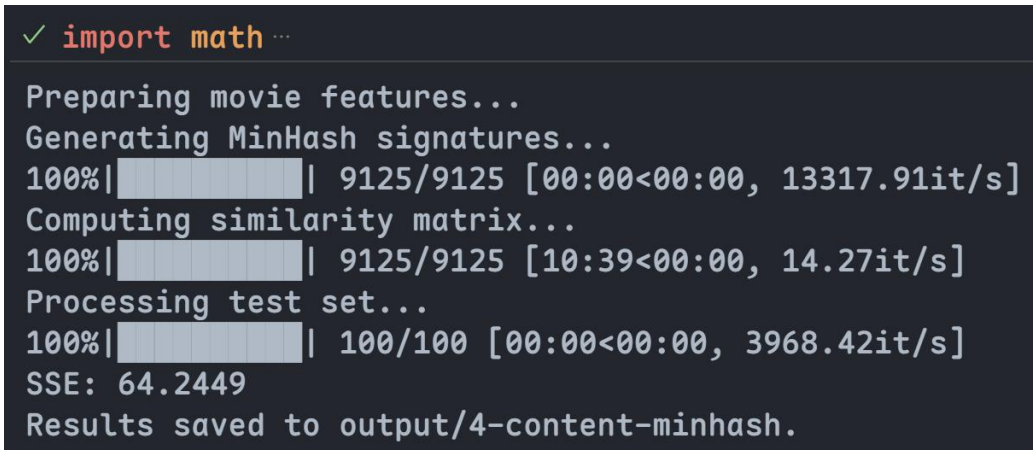


```
userId,movieId,rating,timestamp,predicted
547,1,3.5,1053173138,3.2264871910261177
547,6,2.5,1149462857,3.3831270862392273
564,1,4.0,974712079,3.424667456284681
564,2,4.0,974839862,3.2831935700434642
624,1,5.0,1019126661,2.9717294669568597
624,2,3.0,1019125424,2.974183790919469
15,1,2.0,997938310,2.4231205529265663
15,2,2.0,1134521380,2.3199829820409135
73,1,5.0,1303464840,3.2719663346325953
```

图 3-2 测试集结果

4. 基于内容的推荐优化算法

代码运行结果如下：

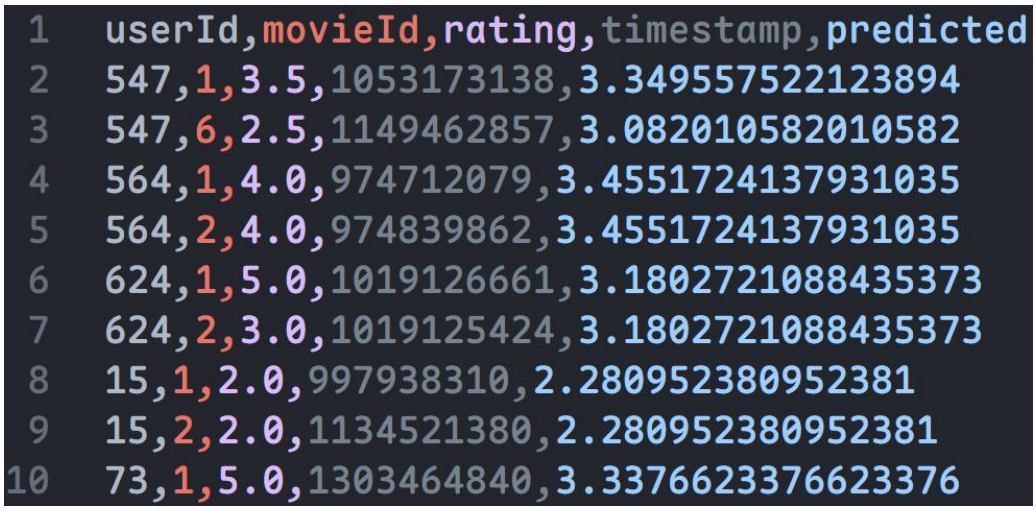


```
✓ import math ···

Preparing movie features...
Generating MinHash signatures...
100%|██████████| 9125/9125 [00:00<00:00, 13317.91it/s]
Computing similarity matrix...
100%|██████████| 9125/9125 [10:39<00:00, 14.27it/s]
Processing test set...
100%|██████████| 100/100 [00:00<00:00, 3968.42it/s]
SSE: 64.2449
Results saved to output/4-content-minhash.
```

图 4-1 代码运行结果

对于测试集，SSE 为 64.2449，预期评分与实际评分较为接近：



```
userId,movieId,rating,timestamp,predicted
547,1,3.5,1053173138,3.349557522123894
547,6,2.5,1149462857,3.082010582010582
564,1,4.0,974712079,3.4551724137931035
564,2,4.0,974839862,3.4551724137931035
624,1,5.0,1019126661,3.1802721088435373
624,2,3.0,1019125424,3.1802721088435373
15,1,2.0,997938310,2.280952380952381
15,2,2.0,1134521380,2.280952380952381
73,1,5.0,1303464840,3.3376623376623376
```

图 4-2 测试集结果

5. 总结

对四种算法进行实现和测试，结合 100 条用户-电影评分的测试数据，得到了以下实验结果：

算法	SSE	分析
基于用户协同过滤	62.4585	挖掘用户间兴趣相似性
基于用户协同过滤（MinHash 优化）	59.8800	降低了计算开销
基于内容推荐	67.1192	推荐结果更具个性化
基于内容推荐（MinHash 优化）	62.2499	对新电影推荐更快

表 5-1 总结

1.4 实验总结

本次实验内容是推荐系统的实现，分为基于用户的协同过滤算法和基于内容的推荐算法，分别以用户间的相似性和电影间的相似性为出发点，来对目标用户的喜好进行预测，这也是实际中推荐系统的重要方法。

但是在实现过程中，我也发现了这两种算法存在的问题。对于基于用户的协同过滤算法，如果用户的相似用户中看过某部电影的人较少，那么在不对公式进行修正的情况下，对于这部电影的评分就会被这少数人极大影响，从而造成结果的失真。而基于内容的推荐算法存在的问题是，电影的分类并不能很好地作为用户评分的参考标准，即用户评分时不会以分类作为单一标准，还可能有电影情节、特效制作等方面的标准，仅以电影分类为预测标准是有失偏颇的。因此，在真实的推荐系统中，我们不仅需要增加推荐的相关指标，也需要将基于用户、基于内容两种方式进行结合，以期找到更精准的推荐结果。

本次实验的主要收获为：掌握了协同过滤与基于内容推荐两类主流推荐算法的实现细节与差异；了解了 MinHash 算法对高维稀疏矩阵降维处理的实际效果；学会了使用多种相似度度量方法（余弦相似度、Jaccard 相似度、Pearson 相关系数）；提升了数据结构组织、向量化编程、哈希函数构造等实际编程能力。