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： **ACM1901**

学　　号： **U201914965**

姓　　名： 卫云泽

指导教师： 崔金华

报告日期： **2021 年 12 月 31 日**

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
1.1 实验目的.....1
1.2 实验内容.....1
1.3 实验过程.....3
1.3.1 编程思路.....3
1.3.2 遇到的问题及解决方式.....10
1.3.3 实验测试与结果分析.....11
1.4 实验总结.....13

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、在学有余力的情况下，**加入 minhash 算法**对效用矩阵**进行降维处理**

1.2 实验内容

给定 MovieLens 数据集，包含电影评分，电影标签等文件，其中电影评分文件分为训练集 train_set 和测试集 test_set 两部分

基础版必做一：**基于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。**对单个用户进行推荐时，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。**

在测试集中包含 100 条用户-电影评分记录，用于计算推荐算法中预测评分的准确性，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，误差计算使用 SSE 误差平方和**。

选做部分提示：此算法的进阶版采用 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 jarcard 方法计算相似度，需要对效用矩阵进行 01 处理，也即**将 0.5-2.5 的评分置为 0，3.0-5.0 的评分置为 1**。

基础版必做二：**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**。

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：

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

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。

推荐算法准确值的判断：**对给出的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，误差计算使用 SSE 误差平方和**。

选做部分提示：进阶版采用 minhash 算法对特征矩阵进行降维处理，从而得到相似度矩阵，注意 minhash 采用 jarcard 方法计算相似度，特征矩阵应为 01 矩阵。因此进阶版的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**。

选做（进阶）部分：

本次大作业的进阶部分是在基础版本完成的基础上大家可以尝试做的部分。进阶部分的主要内容是使用**迷你哈希（MinHash）算法对协同过滤算法和基于内容推荐算法的相似度计算进行降维**。同学可以把迷你哈希的模块作为一种近似度的计算方式。

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，迷你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**。

对基于用户的协同过滤推荐算法和基于内容的推荐算法进行推荐效果对比和分析，选做的完成后再进行一次对比分析。

1.3 实验过程

课程项目实现了电影推荐系统，并基于 MovieLens 数据集开展了一系列测试。实验中使用的推荐算法包括基于用户的协同过滤算法和基于内容的推荐算法。在完成了基本推荐功能后，还使用迷你哈希（minhash）对数据进行降维，分别对两种算法实现的推荐系统进行了优化。

1.3.1 编程思路

1. 基于用户的协同过滤推荐算法

数据集分析与数据读取

基于用户的协同过滤推荐系统基于“ratings.csv”展开，该数据集使用 id 表示用户和电影，给出了部分用户对部分电影的评分，评分的范围在 0.5 至 5 之间，以 0.5 作为步进，此外还给出了用户对电影评分的时间戳信息，该信息在本算法中为无用信息。数据集“ratings.csv”被进一步划分为训练集“train_set.csv”和测试集“test_set.csv”，数据集规模如表 1-1 所示。

表 1-1 原始数据集、训练集和测试集规模

	ratings.csv	train_set.csv	test_set.csv
数据集大小	100004	99904	100
用户数	671	671	50
电影数	9066	9066	15

进一步分析数据集，发现数据集中用户 id 是连续的，范围是 1~671；电影 id 是不连续的，范围是 1~ 163949，因此在数据处理过程中可能需要对电影进行重编号。使用 pandas 读取 csv 文件，并将 DataFrame 转换成透视表结构，形成用户-电影效用矩阵。训练集与测试集读取的部分代码如下。

```
1. data = pd.read_csv("train_set.csv")
2. um = data.pivot_table(index=['movieId'], columns=['userId'], values='rating')
3. testdata = pd.read_csv("test_set.csv")
4. testum = testdata.pivot_table(index=['movieId'], columns=['userId'], values='ra
   ting')   # Utility Matrix
```

代码中使用 DataFrame 的 pivot_table 方法，将原始数据转换成为以‘movieId’为 index、以‘userId’为 columns、以‘rating’为 value 的透视表，即用户-电影的效用矩阵。训练集生成的效用矩阵概览如图 1-1 所示。可见效用矩阵中含有大量 NaN（非数），即用户并没有对该电影进行打分，效用矩阵非常稀疏。

userId	1	2	3	4	5	6	7	...	665	666	667	668	669	670	671
movieId								...							
1	NaN	NaN	NaN	NaN	NaN	NaN	3.0	...	NaN	NaN	NaN	NaN	NaN	4.0	5.0
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	4.0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	3.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
161944	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
162376	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
162542	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
162672	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
163949	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

图 1-1 训练集透视表（效用矩阵）

获取 pearson 相似度

对于用户x,y，皮尔森相关系数定义如下：

$$sim_{pearson}\frac{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})(r_{ys}-\overline{r_y})}{\sqrt{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})^2}\sqrt{\sum_{s\in S_{xy}}(r_{yx}-\overline{r_y})^2}} \tag{1}$$

式中S_{xy}是用户x,y都有打分的电影集合，r_{xs},r_{ys}分别表示用户x和用户y对电影s的打分值，$\overline{r_x},\overline{r_y}$分别表示用户$x$和用户$y$对$S_{xy}$中电影打分的均值。求其他用户与指定用户的相似度的实现的核心代码如下。

```
1.  n_users = len(um.columns)
2.  sim = {}
3.  for i in range(1, n_users + 1):
4.      sim[i] = um[id].corr(um[i], method="pearson")   # pearson 相关系数
5.      if math.isnan(sim[i]):  # 无法计算的相关系数设置为 0
6.          sim[i] = 0
7.  sim = sorted(sim.items(), key=lambda it: it[1], reverse=True)
8.  topk = [sortedres[i] for i in range(k)]
```

对于给定用户id，遍历所有用户。对于用户i，使用 pandas.Series 的 corr 方法求出用户id和i之间的 pearson 相关系数。经查阅资料与实际测试，corr 方法中实现的 pearson 相关系数的计算方法与（1）式一致，且对非数 NaN 具有良好的容忍性。由于可能存在两用户打分的电影没有交集、或 pearson 相关系数求解过程中分母为 0 的情况，代码第 4 行获得的相关系数同样可能是一个非数 NaN，为方便后续推荐过程，实验中统一对 NaN 的相关系数置 0，即认为二者不相关。在获得用户id和所有用户的相似度之后，对相似度进行降序排序，如代码第 7 行。由于使用 dict 类型存放用户的相似度，排序时需要指定排序关键字为(key,value)中的 value 值，即 key=**lambda** it: it[1]，同时使用 reverse=True 设置排序顺序为降序。

至此，我们已经求出用户id和其他所有用户的相似度排名，在排名中截取前k个即为和用户id最相似的k个用户。

获得用户的推荐电影

基于与用户x最相似的k个用户对电影的打分情况，可以计算出用户x未曾观看的电影的预测评分，用户x对电影i的预测评分计算公式如下。

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} r_{yi}}{\sum_{y \in N} s_{xy}} \tag{2}$$

式中集合N是与用户x最相近的k个用户的集合，s_{xy}是用户x和用户y的相似度，r_{yi}是用户y对电影i的打分。公式基于加权的思想，即认为与用户x相似度高的用户的评分更为重要，因此权重更高，可以直接使用相似度表示权重。

获取用户所有未观看电影的预测评分，并给出前n个推荐电影的代码实现如下。

```
for movie, rate in um[userid].items():
    if math.isnan(rate):  # 只推荐没有评价过的电影
        weight = 0  # 权重之和，即相似度之和
        mark_weightedsum = 0  # 乘以权重的打分之和
        for user, r in topk:
            if not math.isnan(um[user][movie]):
                mark_weightedsum += um[user][movie] * r
                weight += r
        if weight != 0:
            mark_pred = mark_weightedsum / weight  # 求加权平均
            predictmarks.append((movie, mark_pred))
predictmarks.sort(key=lambda d: d[1], reverse=True)
print("Top{} Predictmarks: {}".format(n, predictmarks[:n]))  # 打印推荐的电影
```

代码段实现了对用户$userid$的电影推荐，首先遍历用户$userid$的所有电影评分，对于未评分（评分为 NaN）的电影执行分数预测。电影分数预测需要遍历与用户$userid$最相似的k个用户（存放于 topk 中），分别计算带权分数和以及相似度之和，并根据公式（2）求出分数的加权平均作为预测分数。最后对所有预测的电影按照分数从高到低排序，取前n个打印输出，即为对该用户的电影推荐。

基于迷你哈希的优化

基于皮尔森相关系数的用户相似度求解充分利用了用户对不同电影打分的信息，获得的相似度较为精准，由于数据维数较大，该方法同时也具有运算量较大、耗时较长的缺点。基于降维的思想，用户相似度的计算可使用迷你哈希+Jaccard 相关系数的算法近似求得。

基于迷你哈希优化的第一步是对用户-电影效用矩阵进行 0-1 化处理，即将评分不高于 2.5 的电影置 0，评分高于 2.5 的电影置 1。训练集数据的 0-1 化过程可使用如下代码较为简洁地实现，其中 um 为 0-1 化的效用矩阵。

```
data = pd.read_csv("train_set.csv")
data.loc[data['rating'] <= 2.5, 'rating'] = 0
```

```
data.loc[data['rating'] > 2.5, 'rating'] = 1
um = data.pivot_table(index=['movieId'], columns=['userId'], values='rating')
```

获得 0-1 化效用矩阵后，进一步构造迷你哈希矩阵，即实现数据的降维。迷你哈希方法的思路如下：首先寻找一个哈希函数将效用矩阵按行“打乱”，即为原效用矩阵的每一行映射一个新的行号，再在映射后的矩阵中自上而下寻找每一个用户的第一个不为 0 的电影，即该用户打分较高的电影，并用这个电影 id 作为该用户本轮的签名。选取不同的哈希函数，多次执行上述过程，即可为每个用户获得一组签名，构成迷你哈希矩阵。获得迷你哈希矩阵的核心代码如下。

```
for hashi in range(n_hashes):
    hash_func = list(um.index.values)
    np.random.shuffle(hash_func)
    hashed_um = um.reindex(hash_func)      # 随机哈希之后的新效用矩阵
    tobeSigned = queue.Queue()             # 记录需要获得签名的用户—队列优化
    for i in range(1, n_users+1):          # 初始化
        tobeSigned.put(i)
    Signature_matrix_i = np.zeros(n_users + 1)     # 记录本轮签名结果
    for movie in hash_func:
        tobeSigned_new = queue.Queue()             # 本轮留下的尚未获得签名的
        while not tobeSigned.empty():
            user_i = tobeSigned.get()
            if hashed_um.loc[movie][user_i] == 1:
                Signature_matrix_i[user_i] = movie
            else:
                tobeSigned_new.put(user_i)
        if tobeSigned_new.empty():
            break
        tobeSigned = tobeSigned_new
    Signature_matrix[hashi] = Signature_matrix_i
```

其中 n_hashes 是哈希过程的执行次数，每一轮哈希开始时，初始化本轮哈希函数为效用矩阵的行索引向量 um.index.values，并使用 np.random.shuffle 作为随机哈希函数对行索引向量进行打乱处理，获得哈希后的索引，再根据此索引对效用矩阵的索引顺序进行修改，获得执行哈希操作后的效用矩阵，如代码 2-4 行。

接下来是寻找每一位用户本次的哈希签名的过程，由于用户只需寻找到第一个不为 0 的电影，因此获得签名的用户不必再继续搜索，可以使用队列结构记录当前还未找到签名的用户：遍历按照电影 index 按行遍历执行哈希操作的效用矩阵，对于当前队列中的用户，判断用户是否对该电影进行了较高评分的打分（即矩阵中对应值为 1），若已经打分，则该电影就是该用户本轮的哈希签名，记录在 Signature_matrix_i[user_i]中，否则将用户移入下一轮待搜索队列，迭代搜索直到队列为空，此时所有用户都获得了一份哈希签名，形成了本次的用户哈希签名向量 Signature_matrix_i，将该向量加入到哈希签名矩阵中。整个过程迭代

n_hashes 次即可获得最终的哈希签名矩阵 Signature_matrix，见代码 5-20 行。

在获得哈希签名矩阵之后，即可使用该矩阵作为降维后的数据进行用户相似度计算，用以替换基于皮尔森相关系数的用户相似度计算过程，替换代码如下。

```
1.  n_users = len(um.columns)
2.  sim = {}
3.  for userid in range(1, n_users + 1):
4.      common = np.sum(Signature_matrix[id] == Signature_matrix[userid])
5.      sim[userid] = common / n_hashes
6.  sim = sorted(sim.items(), key=lambda it: it[1], reverse=True)
```

此段代码实现了基于迷你哈希使用 Jaccard 相关系数计算用户相似度的过程，代码第 4 行求出了哈希签名矩阵中两用户的签名相同的次数，值得注意的是，此处的比较是按行一一比较，而不是简单的交集运算，即两用户在同一次哈希过程中获得相同的签名才算一次签名相同；代码第 5 行计算了 Jaccard 相似度，即签名相同次数除以总签名次数。使用此处求出的相似度代替先前的皮尔森相似度，执行后续电影评分预测操作，即可完成电影推荐任务。

2. 基于内容的推荐算法

数据集分析与数据预处理

基于内容的推荐算法从电影的内容出发，为用户推荐与其打分较高的电影相似的电影，因此需要额外使用描述电影内容（类别）的数据集“movies.csv”。

“movies.csv”包含三列信息，分别是电影的 movieId，电影名称和电影类别，本实验只关注 movieId 和电影类别的对应关系。其中电影类别使用字符串描述，如“Fantasy”等，同一电影的多个类别使用“|”作为分隔符。

读入数据后，首先对数据每一行的类别使用“|”进行分割，获得 movieId 及其对应类别的字符串列表，使用 set 容器统计出现过的所有电影类别，发现所有电影的类别仅有 20 种，如图 1-2 所示，类别中的(no genres listed)表示“没有分类”，可以视作一种默认类别。

```
{'Documentary', 'Animation', 'IMAX', 'Fantasy', 'Horror', 'Musical', 'Comedy',
 'Thriller', 'Crime', 'Action', 'Western', '(no genres listed)', 'Adventure',
 'Drama', 'Sci-Fi', 'Children', 'War', 'Film-Noir', 'Romance', 'Mystery'}
```

图 1-2 电影的 20 种类别

根据电影类别数据填写电影-类别矩阵，电影对应的类别标记为 1，没有的类别标记为 0，形成的稀疏矩阵如图 1-3 所示

至此，完成了数据集的读取与预处理，电影推荐过程基于电影-类别矩阵展开。

获取 TF-IDF 特征值

基于内容的基础推荐算法基于 TF-IDF 算法和余弦相似度展开。获得电影-类别矩阵之后，首先需要求出电影的 TF-IDF 值。

movieId	Documentary	Animation	IMAX	...	Film-Noir	Romance	Mystery
				...			
1	0.0	1.0	0.0	...	0.0	0.0	0.0
2	0.0	0.0	0.0	...	0.0	0.0	0.0
3	0.0	0.0	0.0	...	0.0	1.0	0.0
4	0.0	0.0	0.0	...	0.0	1.0	0.0
5	0.0	0.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...
162672	0.0	0.0	0.0	...	0.0	1.0	0.0
163056	0.0	0.0	0.0	...	0.0	0.0	0.0
163949	1.0	0.0	0.0	...	0.0	0.0	0.0
164977	0.0	0.0	0.0	...	0.0	0.0	0.0
164979	1.0	0.0	0.0	...	0.0	0.0	0.0

图 1-3 电影-类别矩阵

词频-逆文档率（Term Frequency–Inverse Document Frequency, TF-IDF）是一种用于信息检索和数据挖掘的加权基数，它可以评估一个分词对于一份文本的重要程度。

对于相同重要程度的词语而言，其在较短样本中的词频会低于较长样本的，因此通常使用归一化词频来评估一个词在文本中的重要性。归一化词频算法如下：

$$tf_{i,j} = \frac{w_{i,j}}{N_j} \tag{3}$$

其中$tf_{i,j}$是词语w_i在样本s_j中的归一化词频，$w_{i,j}$是词语w_i在样本s_j中出现的次数，N_j是样本s_j的词语总数。

逆文档率是衡量一个词语总体重要性的指标，它可以用来降低各个文本中都含有的词语的词频权重。词语w_i的逆文档率计算公式如下：

$$idf_i = log\left(\frac{S}{|k: w_i \in s_k|}\right) \tag{4}$$

其中idf_i是词语w_i的逆文档率，S是样本总数，$|k: w_i \in s_k|$表示包含w_i的样本数目。

结合$tf_{i,j}$和idf_i，词语w_i在样本s_j中的 TF-IDF 权值可表示为：

$$tfidf_{i,j} = tf_{i,j} \times idf_i = \frac{w_{i,j}}{N_j} \times log\left(\frac{S}{|k: w_i \in s_k|}\right) \tag{5}$$

实验中获取 TF-IDF 特征值的思路是首先计算各个类别的 IDF 值，再在电影-类别矩阵上直接计算得到 TF-IDF 矩阵。求 IDF 值的代码实现如下。

```
1.  genres_idf = {}
2.  for genres in movie_matrix.columns:
3.      count = 0
4.      for i in range(n_movies):
5.          if movie_matrix.iloc[i][genres] != 0:
6.              count += 1
7.      genres_idf[genres] = math.log10(n_movies / count)
```

根据（4）式，代码 3-6 行统计了某一特征在所有电影中的出现频次，第 7 行计算了电影总数除以出现频次的以 10 为底的对数值，即为该类别的 IDF 值。

获得了所有特征的 IDF 值之后，遍历所有电影的每一个类别，求出该电影这个类别的 TF-IDF 特征值，代码实现如下。

```
for i in range(n_movies):
    genres_sum = np.sum(movie_matrix.iloc[i])
    for index, value in movie_matrix.iloc[i].items():
        if value != 0:
            movie_matrix.iloc[i][index] /= genres_sum
            movie_matrix.iloc[i][index] *= genres_idf[index]
```

由于一个电影的类别只会出现一次，因此求词频的过程可简化为 1 除以该电影的类别总数，再使用获得的 TF 值乘以对应类别的 IDF 值即可获得 TF-IDF 值，如代码 5-6 行所示。代码段执行完毕后，原电影-类别矩阵即转换为 TF-IDF 特征矩阵。

获得用户的推荐电影

基于电影内容的推荐算法以被推荐用户打过分的电影作为“训练集”，根据 TF-IDF 特征矩阵、使用余弦相似度方法求出其他未打分电影与“训练集”电影的相似度，进而求出电影的预测分数，给出预测分数最高的n个电影作为推荐电影。因此，推荐过程的第一步是获得用于训练的电影列表（train_list）和待求电影列表（test_list），实现代码如下。

```
for movie, rating_train in movielist.items():
    if math.isnan(rating_train):
        test_list.append(movie)  # 分数是非数的电影是未打分电影，即待求电影
    else:
        train_list.append(movie) # 分数不为非数的电影是已打分电影，即训练电影
```

在完成训练、待求电影列表的划分之后，遍历待求电影列表中的电影，基于 TF-IDF 特征矩阵使用余弦相似度计算该待求电影与其他所有用于训练的电影之间的相似度，再结合训练电影的分数求出待求电影的预测分数，计算公式如下。

$$\text{score} = \frac{\sum_{i=1}^{n} score'(i) * sim(i)}{\sum_{i=1}^{n} sim(i)} \tag{6}$$

式中$score'(i)$是第i个训练电影的分数，$sim(i)$是第i个电影和该电影的余弦相似度。电影分数预测的代码实现如下。

```
for movie_test in test_list:  # 遍历待求电影列表
    score_sim_sum = 0
    sim_sum = 0
    for movie_train in train_list:
        sim = cos(movie_matrix.loc[movie_train], movie_matrix.loc[movie_test])
        score_sim_sum += sim * movielist[movie_train]  # 带权分数之和
        sim_sum += sim  # 相似度（权重）之和
    if sim_sum > 0:
```

```
9.          ratings[movie_test] = score_sim_sum / sim_sum  # 加权分数
10. sortedratings = sorted(ratings.items(), key=lambda it: it[1], reverse=True)
11. topk = [sortedratings[i] for i in range(k)]
12. print("top{}:{} ".format(k, topk))
```

代码 4-9 行实现了对某个电影的预测分数的计算。求出所有待求电影分数后对分数从大到小进行排序，截取前k个电影作为推荐电影打印输出。至此，已实现了基于内容的电影推荐。

基于迷你哈希的优化

基于内容的推荐算法的迷你哈希的降维优化过程与基于用户的协同过滤推荐算法的迷你哈希优化过程相似。由于哈希映射是针对矩阵的行号执行的，因此在执行哈希操作之前需要先获得电影-类别矩阵的转秩，即矩阵的每一行代表一个类别，再使用随机哈希函数进行行的打乱处理，每次选取第一个为 1 的类别作为电影的哈希签名。由于电影类别总共仅有 20 个，故哈希次数需要减少至个位数，如执行 3 次哈希过程即可获得不错的结果。

1.3.2 遇到的问题及解决方式

nan 非数问题：

在基于用户的协同过滤推荐算法的初版代码测试中，出现了推荐的前n个电影分数部分为非数（nan）的问题，且推荐的前n个电影分数也并非按照分数由高到低排序，出现错误的部分推荐结果如图 1-4 所示。

```
Top100 Predictmarks: [(1, nan), (2, nan), (3, nan), (5, nan), (6, nan), (7, nan),
 (8, 4.0), (9, nan), (16, 7.429829560932368), (11, 3.917372106990247), (12,
 3.7081375952677726), (4, 2.9999999999999996), (15, 2.5), (14, 2.342944181399815)
```

图 1-4 出现非数的错误电影推荐结果（部分）

解决方式：

通过对推荐系统的每一步中间计算结果的打印输出，发现用户的相似度的计算结果中即包含大量非数。造成该现象的原因是 Pandas 的 corr 方法对非数的容忍性，即无法计算的相似度会以非数 nan 的形式展现，而不会产生程序运行错误。因此需要对无法计算的用户相似度进行特殊处理，如赋 0（认为无法计算相似度的用户之间不存在相关关系），即可解决非数问题，正确的推荐结果（部分）如图 1-5 所示。

```
Top100 Predictmarks: [(80, 5.0), (262, 5.0), (352, 5.0), (373, 5.0), (417, 5.0),
 (446, 5.0), (451, 5.0), (492, 5.0), (559, 5.0), (759, 5.0), (766, 5.0), (820, 5.0),
 (907, 5.0), (927, 5.0), (935, 5.0), (961, 5.0), (970, 5.0), (984, 5.0), (994, 5.0),
```

图 1-5 正确的电影推荐结果（部分）

电影相似度矩阵规模过大问题：

在基于内容的电影推荐算法的初版代码中，企图直接求出并电影相似度矩阵

并保存为文件，在预测时读取文件中的相应条目即可，认为预先的计算可以大幅缩短预测耗时。但实际由于电影数目过多，电影之间的相似度条目数目又是电影数目的平方的数量级，约为一亿项，经过数小时的计算，最终获得了一个 800 多 MB 的 csv 文件，如图 1-6 所示。读取该文件中指定条目的耗时已经超出了计算该条目的耗时。

图 1-6 完整的电影相似度矩阵文件

解决方式：

放弃预计算相似度矩阵的思路，采用实时求解的做法即用即计算，每一次只求出需要的电影相似度即可，实际的计算耗时远小于读取相似度矩阵文件对应项目的耗时。

1.3.3 实验测试与结果分析

1. 参数调优

基于用户的协同过滤推荐算法

在基于用户的协同过滤推荐算法中，主要参数为k，即相似度最高的用户个数。在测试中发现，当 k 取值较小（如不大于 60）时，由于相似用户样本量过少，出现部分电影无法给出预测评分的现象，因此k应当取较大值进行测试，SSE 和测试用时随k的变化趋势图如图 1-7 所示。

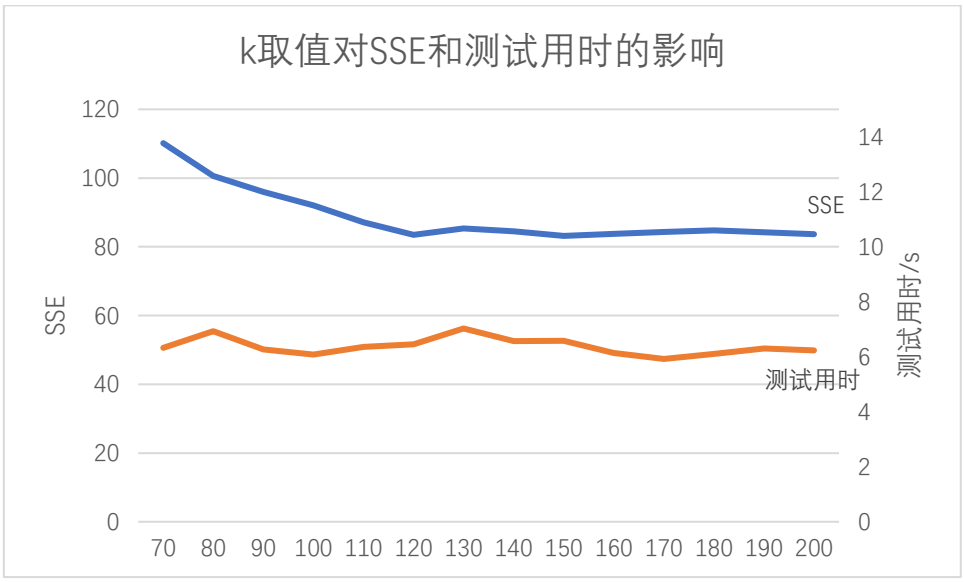

图 1-7 k 取值对 SSE 和测试用时的影响

从图 1-7 可知，SSE 随k值增大而减小，最终趋于稳定，其原因是k取值过小时样本量较小，噪声数据对结果的影响较大；k取值较大时相似度较低的用户对结果的影响受到加权算法的抑制而降低，故结果趋于稳定。从图可知，k取 120 以上 SSE 的变化幅度较小，故k的较合适的取值为 120。此外，从图中还可

发现测试用时与k取值无关，其原因是算法的时间瓶颈在于用户相似度的计算，而不在预测过程。

使用迷你哈希优化的算法引入了另一个参数：n_hashes，即使用的哈希函数数目。由于实验中的参数k的较优值的选取并非在极值点，故本实验中不采用网格搜索方法进行参数调优。在k取值 120 时，选取不同量级的哈希函数数目的测试结果如表 1-2 所示，考虑到迷你哈希使用随机哈希映射而导致结果具有一定波动性，表 1-2 及后文涉及迷你哈希的实验结果均为 5 次均值结果。

表 1-2 哈希函数数目对测试结果的影响

n_hashes	10	100	1000
SSE	86.25	79.71	75.82

从表 1-2 可知，更多的哈希函数会带来更好的测试结果，与实验预期相符，原因是哈希函数数目越大，数据的损失就越小，结果越准确。但选取过多的哈希函数会导致预处理时间的增长，因此选取 100 作为哈希函数数目的较合适取值。

基于内容的推荐算法

在基于内容的推荐算法中，对电影的预测评分基于用户所有其他已打分电影展开，故不存在可调节参数。使用迷你哈希优化的算法通用含有参数n_hashes，考虑到电影的类别一共仅 20 个维度，故n_hashes的取值需要限制在 20 以内才具有优化意义。预测结果SSE和预测时间随n_hashes的变化趋势如图 1-8 所示。

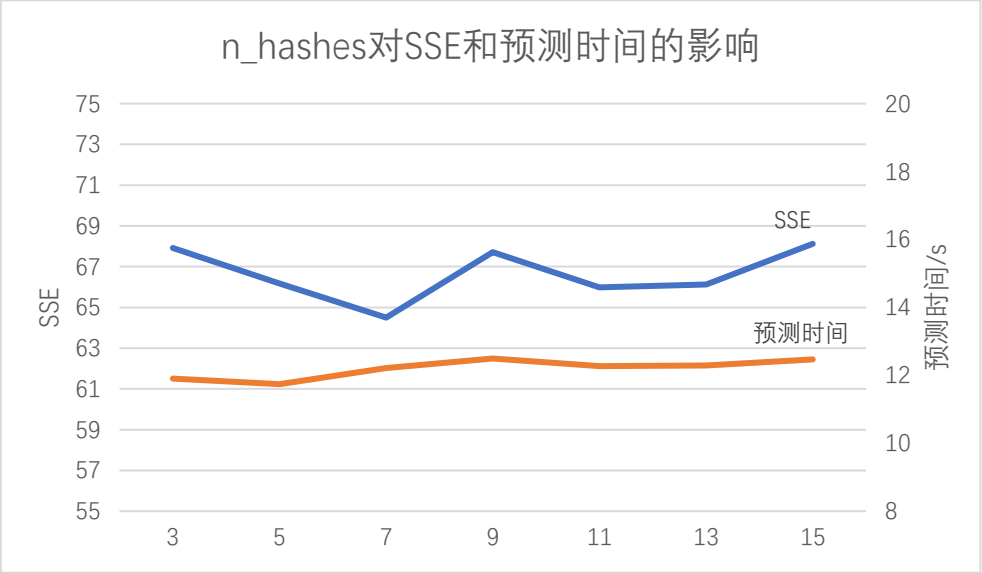

图 1-8 n_hashes 对 SSE 和预测时间的影响

从图 1-8 可知，迷你哈希函数数量取 7 时具有较好的预测效果，原因是选取的哈希函数过少会损失过多的精度，但大部分电影的类别数目实际较少，故迷你哈希函数数量取值过大时反而会使数据失真。预测时间对哈希函数数目的增长并不铭感，与实验预期不一致，分析其原因在于哈希过程本身并不是算法的瓶颈。

2. 不同算法预测效果对比

实验中实现的四种算法（包括两种优化算法）的预测效果如表 1-3 所示，其中基于用户的协同过滤推荐算法k取值 120，优化算法的迷你哈希函数数目为 100；基于内容的推荐算法优化算法中迷你哈希函数数目取值为 7。

表 1-3 四种算法的预测效果

	SSE	测试用时/s
基于用户的协同过滤推荐算法	83.49	6.45
基于用户的协同过滤+迷你哈希优化	79.71	4.81
基于内容的推荐算法	67.06	24.69
基于内容的推荐算法+迷你哈希优化	64.50	12.22

从表 1-3 的实验结果可获得如下对比结论：

1. 分别对比表中 1、3 和 2、4 两行可知，无论是否使用迷你哈希优化，基于内容的推荐算法对电影分数的预测效果都要优于基于用户的协同过滤推荐算法的预测效果，但测试用时更长。预测结果表明，电影推荐领域的用户相似度要弱于电影之间的相似度，即喜好相似的用户对不同类别电影的打分未必一致，但对某一用户而言，对于其喜好类型的电影打分更趋于一致。由于电影数量规模远大于用户数量，因此求电影相似度的过程比求用户相似度的过程更加耗时。
2. 分别对比表 1、2 和 3、4 两行可知，使用迷你哈希+Jaccard 相似度优化的算法测试用时均少于不优化的算法，但迷你哈希对基于内容的推荐算法的优化效果比对基于用户的协同过滤推荐算法的效果要更好，测试用时大幅降低，降幅高达 50.5%。使用迷你哈希优化的预测效果 SSE 同样均小于不优化的算法，结果与预期不符，其原因有待进一步的研究和思考。

1.4 实验总结

在本次大数据分析实验结课项目中，我使用了四种算法实现了电影推荐系统。在实验中，我首先了解了基于用户、基于内容推荐概念，学习了皮尔森相关系数、余弦相似度等基本原理，并在此基础上实现了基于用户的协同过滤推荐系统和基于内容的推荐系统。

在基础推荐系统实现后，我还学习了迷你哈希算法的降维原理，并基于 Jaccard 相关系数对两种推荐系统进行了优化，降低了推荐过程的耗时。此外，在系统实现中我还遇到了许多细节问题，如非数问题等，这些问题的解决也增强了我对较大型系统的细节把控意识。总之我在本次实验中收获颇丰，我相信大数据

分析这课程会成为我打开数据科学殿堂的“敲门砖”,它展现给我的只是冰山一角，其中更多的奥妙还在等待着我慢慢挖掘。

最后，我还想要感谢崔金华老师和助教学长在课程中对我的帮助与指导，同时也要感谢大数据分析这门课,丰富的课程内容与实验安排带给我了巨大的收获，衷心地祝愿这门课程会越办越好。