

华中科技大学

# 项目开发报告

题目： 基于协同过滤算法的电影推荐系统

课程名称： 机器学习

专业班级： CS1705 班

学 号： U201714707

姓 名： 练炳诚

指导教师： 李玉华

报告日期： 2020 年 7 月 30 日

计算机科学与技术学院

# 项目开发报告

## 1.1 项目目的

通过代码实现基于用户的协同过滤（UserCF）、基于物品的协同过滤（ItemCF）、带惩罚机制的 UserCF（UserCF-IIF）、带惩罚机制的 ItemCF（ItemCF-IUF）等推荐算法，并在 MovieLens 数据集上进行测试，计算各推荐算法的评测指标：准确率、召回率、覆盖率、流行度，对比几种推荐算法的效果。具体做了以下工作：

- （1）通过协同过滤算法实现电影推荐；
- （2）改进基本的协同过滤算法；
- （4）通过评测指标比较几种推荐算法的效果。

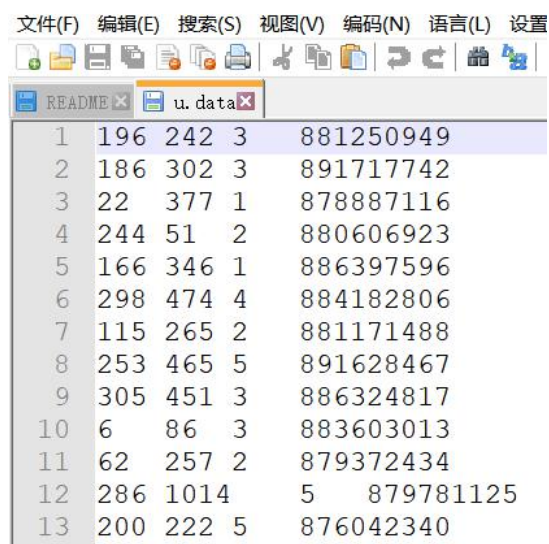
数据集介绍：MovieLens 数据集包含多个用户对多部电影的评级数据，也包括电影元数据信息和用户属性信息。这个数据集经常用来做推荐系统，机器学习算法的测试数据集。尤其在推荐系统领域，很多著名论文都是基于这个数据集的。

数据集的下载地址：<http://files.grouplens.org/datasets/movielens/>

## 1.2 问题分析

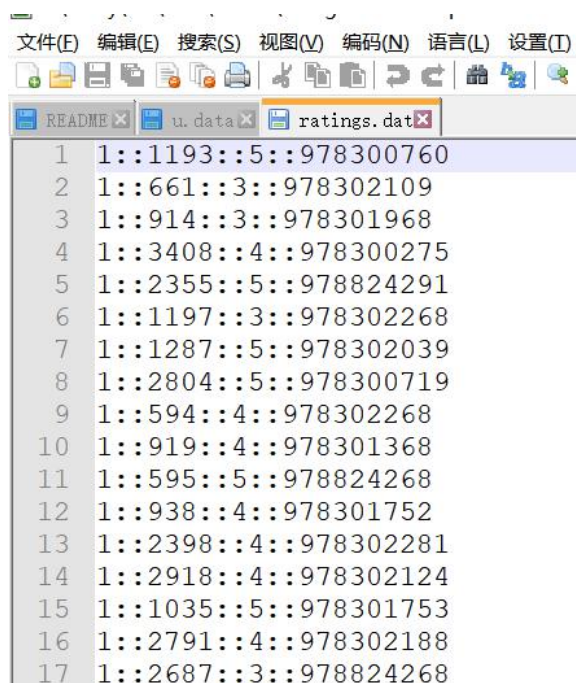
首先是要获取数据集，并划分训练集和测试集，选取 MovieLens 中的 ml-100k 和 ml-1m 两个数据集，分别为含 100,000 条评分记录和含 1000,000 条评分记录的数据。

解压数据包后，阅读文档可知数据包中有已经划分好的数据，ml-100k 的完整数据存储在 u.data 中，ml-1m 的完整数据存储在 ratings.dat 中，两者的存储格式不同，u.data 中每一行是以'\t'分隔的 4 项数字，分别为用户编号、电影编号、评分(1~5 间的整数)、时间戳，ratings.dat 中分隔符是“::”字符串，如图 1-1 和图 1-2 所示。



1	196	242	3	881250949
2	186	302	3	891717742
3	22	377	1	878887116
4	244	51	2	880606923
5	166	346	1	886397596
6	298	474	4	884182806
7	115	265	2	881171488
8	253	465	5	891628467
9	305	451	3	886324817
10	6	86	3	883603013
11	62	257	2	879372434
12	286	1014	5	879781125
13	200	222	5	876042340

图 1-1 u.data 中的数据



1	1::1193::5::978300760
2	1::661::3::978302109
3	1::914::3::978301968
4	1::3408::4::978300275
5	1::2355::5::978824291
6	1::1197::3::978302268
7	1::1287::5::978302039
8	1::2804::5::978300719
9	1::594::4::978302268
10	1::919::4::978301368
11	1::595::5::978824268
12	1::938::4::978301752
13	1::2398::4::978302281
14	1::2918::4::978302124
15	1::1035::5::978301753
16	1::2791::4::978302188
17	1::2687::3::978824268

图 1-2 ratings.dat 中的数据

为更加灵活的划分数据集，项目中不采用已划分的数据，而是自己进行划分。考虑到协同过滤算法中的模型仅与训练集的划分有关，故每次划分后可将模型保存下来，下次运行时若划分的参数与上次协同，可直接读取模型。

完成数据集划分后，要实现两种不同思路的 CF 算法和相应的改进算法，并使用训练集进行训练，最后用测试集进行测试，计算出以下四个推荐系统的评测指标：

(1) 精确率 (precision)：描述最终的推荐列表中有多少比例是发生过的

用户—物品评分记录；

(2) 召回率 (recall)：描述有多少比例的用户—物品评分记录包含在最终的推荐列表中；

(3) 覆盖率 (coverage)：反映了推荐算法发掘长尾的能力，覆盖率越高，说明推荐算法越能够将长尾中的物品推荐给用户；

(4) 流行度 (popularity)：反映了推荐列表中物品的平均流行度。如果推荐出的物品都很热门，说明推荐结果偏向于流行物品，否则说明推荐结果比较新颖。

为了与基础推荐算法进行比较，还要实现随机推荐算法 Random 和最受欢迎推荐算法 MostPopular。

最后分析评估各种推荐算法的性能。

## 1.3 设计与分析

### 1.3.1 基于用户的协同过滤算法：UserCF

协同过滤算法是只基于用户的历史行为对用户未来的行为进行预测的一种算法。对于协同过滤的算法，最著名的、在业界得到最广泛应用的算法是基于邻域的方法，而基于邻域的方法主要包含基于用户的协同过滤算法和基于物品的协同过滤算法。

基于用户的协同过滤算法主要包括以下两个步骤：

(1) 找到和目标用户兴趣相似的用户集合；

(2) 找到集合中用户所喜欢的，且目标用户没有使用过的物品并推荐给目标用户。

在电影推荐中，喜欢程度是由用户评分决定的，步骤（1）的关键就是计算用户间的相似度，这里主要利用行为的相似度计算兴趣的相似度。对用户  $u$  和用户  $v$ ，令  $N(u)$  和  $N(v)$  分别表示两个用户的评分过的电影集合，采用余弦相似度公式如下：

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)| |N(v)|}} \quad (1.1)$$

例如用户  $u$  评分过电影  $\{a,b,c\}$ ，用户  $v$  评分过电影  $\{a,d\}$ ，则  $w_{uv}$  计算如下：

$$w_{uv} = \frac{|\{a,b,c\} \cap \{a,d\}|}{\sqrt{3 \times 2}} = \frac{1}{\sqrt{6}}$$

该算法的时间复杂度是  $O(|U|*|U|)$ ，一般情况下用户—物品矩阵是十分稀疏的，当用户数较大时，这个计算过程会比较耗时，为加快计算速度，没有共同评价的用户的相似度为 0，我们只需要计算出有共同评价电影用户之间的相似度即可。

为此，可以先建立电影—用户的倒排索引，列出每个电影的评价用户列表，对这些列表进行扫描，既可统计每个用户评价的电影数量  $|N(u_i)|$ ，也可以找到具有共同评价电影的用户，建立用户—用户矩阵  $N[u][v]$ ，记录用户  $u$  和用户  $v$  共同评价的电影数。最后，余弦相似度计算只针对每个电影列表中的用户，其余的相似度为 0。算法的代码描述如下：

```
# 建立电影-用户倒排索引, key=movieID, value=list of userIDs
movie2users = collections.defaultdict(set)
movie_popular = defaultdict(int)
for user, movies in trainset.items():
    for movie in movies:
        movie2users[movie].add(user)

# 记录用于用户相似度评估的电影数量
movie_count = len(movie2users)

# 对用户共同评分的进行计数
usersim_mat = {}
for movie, users in movie2users.items():
    for user1 in users:
        # 初始化为 0
        usersim_mat.setdefault(user1, defaultdict(int))
        for user2 in users:
            if user1 == user2:
                continue
            else:
                usersim_mat[user1][user2] += 1

# 计算用户余弦相似度矩阵
for user1, related_users in usersim_mat.items():
    len_user1 = len(trainset[user1])
    for user2, count in related_users.items():
        len_user2 = len(trainset[user2])
        # 用户 1 和用户 2 的相似度 = 共同评价电影数 / sqrt(用户 1 评价电影数 * 用户 2 评价电影数)
        usersim_mat[user1][user2] = count / math.sqrt(len_user1 * len_user2)
```

在计算得到用户余弦相似度矩阵后，UserCF 算法会给目标用户推荐和他兴趣相似度最高的  $K$  个用户评分最高的  $N$  部电影，如下公式度量了用户  $u$  对电影  $i$  的感兴趣程度：

$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{uv} r_{vi} \quad (1.2)$$

其中， $w_{uv}$  表示用户  $u$  和用户  $v$  之间的余弦相似度， $r_{vi}$  表示用户  $v$  对电影  $i$  的评分，即用户  $u$  对电影  $i$  的感兴趣程度由与他相似度最高的  $K$  个用户对电影  $i$  的评分乘上相似因子后累加得到。

采用该计算公式计算与用户  $u$  最相似的  $K$  个用户，并推荐  $N$  个电影的代码如下：

```
watched_movies = self.trainset[user]
# 选出与用户 user 评分习惯最相似的 K 个用户和相似因子
for similar_user, similarity_factor in sorted(self.user_sim_mat[user].items(),
                                             key=itemgetter(1), reverse=True)[0:K]:
    for movie, rating in self.trainset[similar_user].items():
        if movie in watched_movies:
            continue
        # 用户 1 对该电影的感兴趣程度 += 用户 1 和用户 2 的相似因子 * 用户 2 的评分
        predict_score[movie] += similarity_factor * rating
# 返回感兴趣程度最高的 N 部电影作为推荐
return [movie for movie, _ in sorted(predict_score.items(), key=itemgetter(1), reverse=True)[0:N]]
```

### 1.3.2 基于物品的协同过滤算法：ItemCF

UserCF 有一些缺点。首先，随着用户数量的增多，计算用户相似度矩阵将越来越困难，其运算的时间复杂度和空间复杂度的增长近似于平方关系。其次，基于 UserCF 很难对推荐结果作出解释。

基于物品的协同过滤算法 ItemCF 的主要思想是分析用户的行为记录和计算物品之间的相似度，物品 A 和物品 B 具有很大的相似度是因为喜欢物品 A 的用户大都喜欢物品 B。该算法可以利用用户的历史行为给推荐结果提供推荐解释，比如给用户推荐《数据挖掘》是因为用户之前看过《机器学习》。

ItemCF 的主要步骤如下：

- (1) 计算物品之间的相似度；

(2) 根据物品的相似度和用户的历史行为给用户进行推荐。

步骤 (1) 的关键是计算物品之间的相似度, 设  $N(i)$  是喜欢物品  $i$  的用户数,  $N(j)$  是喜欢物品  $j$  的用户数,  $|N(i) \cap N(j)|$  是同时喜欢物品  $i$  和物品  $j$  的用户数, 则物品  $i$  和物品  $j$  的相似度计算公式如下:

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|} \quad (1.3)$$

这个公式存在一个问题, 即若物品  $j$  很热门, 有很多人喜欢, 则  $w_{ij}$  就会很大接近于 1。因此, 该公式会造成任何物品都会和热门的物品有很大的相似度, 故对式(1.3)改进如下:

$$w_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)| |N(j)|}} \quad (1.4)$$

式(1.4)中惩罚了物品  $j$  的权重, 减轻了热门物品会和很多物品相似的可能性。

和 UserCF 类似, ItemCF 在计算物品相似度时, 列出每个用户评价的电影列表, 对这些列表进行扫描, 即可统计每个电影的评分用户数量  $|N(i)|$ , 也能找到被同一用户进行评价的电影, 建立电影—电影矩阵  $N[i][j]$ , 记录对电影  $i$  和电影  $j$  都进行评价的用户数。最后, 相似度的计算只针对被同一用户评价过的电影, 其余的相似度为 0。算法的代码描述如下:

```
# 对同一用户评价的电影进行计数
movie_sim_mat = {}

for user, movies in trainset.items():
    for movie1 in movies:
        # 初始化为 0
        movie_sim_mat.setdefault(movie1, defaultdict(int))
        for movie2 in movies:
            if movie1 == movie2:
                continue
            else:
                movie_sim_mat[movie1][movie2] += 1

# 计算物品相似度矩阵
for movie1, related_items in movie_sim_mat.items():
    len_movie1 = movie_popular[movie1]
    for movie2, count in related_items.items():
        len_movie2 = movie_popular[movie2]
        # 物品相似度 = 两电影被同一用户评价数 / sqrt(电影 1 的评价数 * 电影 2 的评价数)
        movie_sim_mat[movie1][movie2] = count / math.sqrt(len_movie1 * len_movie2)
```

在完成物品相似度计算后,可采用如下公式评估用户  $u$  对电影  $j$  的兴趣程度:

$$p(u, j) = \sum_{i \in N(u) \cap S(j, K)} w_{ji} r_{ui} \quad (1.5)$$

其中,  $N(u)$  是用户评分过的电影集合,  $S(j, K)$  是和电影  $j$  最相似的  $K$  个电影的集合,  $w_{ji}$  是电影  $j$  和电影  $i$  的相似度,  $r_{ui}$  是用户  $u$  对电影  $i$  的评分。该公式的含义是, 和用户历史评分电影越相似的电影, 越有可能在用户的推荐列表中获取比较高的排名。

为用户推荐  $N$  部电影的代码如下:

```
# 选出最相关的 K 部电影
for movie, rating in watched_movies.items():
    for related_movie, similarity_factor in sorted(self.movie_sim_mat[movie].items
    ()),
                                                    key=itemgetter(1), reverse=True)
    [0:K]:
        if related_movie in watched_movies:
            continue
        # 用户对电影 1 的感兴趣程度 += 电影 1 和电影 2 的相似因子 * 电影 2 的评分
        predict_score[related_movie] += similarity_factor * rating
    # 返回评分最高的 N 部电影作为推荐
    return [movie for movie, _ in sorted(predict_score.items(), key=itemgetter(1), reverse=True)[0:N]]
```

### 1.3.3 两种 CF 算法的改进: UserCF-IIF 和 ItemCF-IUF

前面了两种算法中, 均未考虑活跃用户和热门电影对相似度的影响。在 UserCF 中, 如果物品很热门, 如用户  $u$  和用户  $v$  都买过牙刷, 这不能说明他们的兴趣相似, 但如果用户  $u$  和用户  $v$  都买过足球, 则说明他们的兴趣爱好相似, 因为很有可能他们都踢足球。

为减弱热门物品对用户相似度的影响, 改进式(1.1)如下:

$$w_{uv} = \frac{\sum_{i \in N(u) \cap N(v)} \frac{1}{\log 1 + |N(i)|}}{\sqrt{|N(u)| |N(v)|}} \quad (1.6)$$

该公式通过  $\frac{1}{\log 1 + |N(i)|}$  惩罚了中热门物品相似度的贡献。经过这样改进的

UserCF 算法记作 UserCF-IIF(UserCF-Inverse Item Frequency)算法。



在 ItemCF 中，两个物品产生相似度是因为它们共同出现在很多用户的兴趣列表中。若一个用户非常活跃，他对大量的物品都“感兴趣”，就会为这些物品的相似度产生贡献。如电影评论家会看很多电影并进行评价，这样诸多电影就因为电影评论家的观看产生了相似性。但是，电影评论家观看这些电影并非是真的感兴趣，而是职业需求，所以电影评论家的观看历史对电影相似性的贡献应该远远小于普通的观影用户。

为解决活跃用户对物品相似度的影响，可采用与式(1.6)一样的思想改进式(1.4)，加入惩罚机制降低活跃用户的贡献，改进如下：

$$w_{ij} = \frac{\sum_{u \in N(i) \cap N(j)} \frac{1}{\log 1 + |N(u)|}}{\sqrt{|N(i)| |N(j)|}} \quad (1.7)$$

该公式通过  $\frac{1}{\log 1 + |N(u)|}$  惩罚了活跃用户对电影相似度的贡献，经过这样改

进的 ItemCF 算法记作 ItemCF-IUF(ItemCF-Inverse User Frequency)算法。

### 1.3.4 非个性化推荐算法：Random 和 MostPopular

为对比前述个性推荐算法的性能，需要设计基本的非个性化推荐算法。随机推荐算法 Randomc 从电影列表中随机选取  $N$  部电影向用户进行推荐。最受欢迎推荐算法 MostPopular 从所有电影列表中选取受欢迎度前  $N$  的电影向用户推荐。电影  $i$  的受欢迎度的计算公式如下：

$$p(i) = \sum_{u \in N(i)} r_{ui} \quad (1.8)$$

其中， $r_{ui}$  是用户  $u$  对电影  $i$  的评分，为简单起见，取 1 即可。

### 1.3.6 评测指标

为评估几种推荐算法的性能，需要计算 1.2 节中介绍的 4 个评测指标。令  $R(u)$  是为用户  $u$  推荐的电影集合， $T(u)$  是在测试集上用户评价过的电影的集合， $I$  是电影集合。

(1) 精确率 (precision)：描述最终的推荐列表中有多少比例是发生过的用户—物品评分记录，计算公式为：

$$Precision = \frac{\sum_u |R(u) \cap T(u)|}{\sum_u |R(u)|} \quad (1.9)$$

(2) 召回率 (recall)：描述有多少比例的用户—物品评分记录包含在最终的推荐列表中，计算公式为：

$$Recall = \frac{\sum_u |R(u) \cap T(u)|}{\sum_u |T(u)|} \quad (1.10)$$

(3) 覆盖率 (cpverage)：描述所推荐的物品数在总物品中占的比例，反映了推荐算法发掘长尾的能力，覆盖率越高，说明推荐算法越能够将长尾中的物品推荐给用户，计算公式为：

$$Coverage = \frac{|\bigcup_{u \in U} R(u)|}{|I|} \quad (1.11)$$

(4) 流行度 (popularity)：反映了推荐列表中物品的平均流行度。如果推荐出的物品都很热门，说明推荐结果偏向于流行物品，否则说明推荐结果比较新颖，计算公式为：

$$Popularity = \log(1 + \sum_{u \in U} r_{ui}) \quad (1.12)$$

其中，由于物品的流行度分布满足长尾分布，在取对数后，流行度的均值更加稳定。

### 1.3.5 程序流程设计

程序整体流程为：读取数据集，按指定比例分割为训练集和测试集，若该比例划分的数据集此前已保存，则直接读取；否则，按比例划分数据集，并将该比例下的训练集和测试集保存。然后根据指定的推荐算法和参数，使用训练集进行训练，将计算得到的模型（协同过滤中为相似度矩阵）存储到本地文件。最后指定几个用户进行电影推荐，并使用测试集的数据测试模型，计算评测指标，进行打印。流程图如图 1-3 所示。

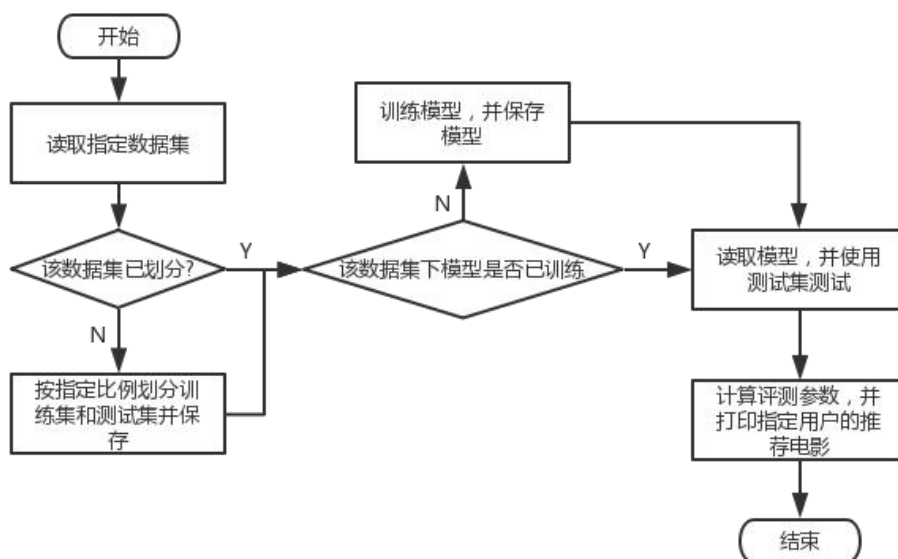


图 1-3 程序流程图

## 1.4 结果分析

以下测试均是在 ml-100k 数据集下进行，训练集占 10%，测试集占 90%。

首先测试对用户进行推荐电影的功能，使用 UserCF 算法，参数 K 设定为 20，N 设定为 10，选取用户 1，100，233，666，888 作为推荐对象，结果如图 1-4 所示。

```

recommend for userid = 1:
['168', '4', '222', '474', '318', '655', '423', '732', '568', '385']

recommend for userid = 100:
['327', '301', '748', '307', '303', '331', '332', '322', '343', '304']

recommend for userid = 233:
['132', '22', '199', '496', '211', '526', '181', '427', '474', '79']

recommend for userid = 666:
['50', '654', '185', '199', '98', '89', '131', '47', '1', '521']

recommend for userid = 888:
['275', '14', '191', '1', '173', '302', '127', '151', '357', '285']
  
```

图 1-4 UserCF 算法电影推荐结果

切换到 ItemCF 算法，在同样的数据集上推荐结果如图 1-5 所示。

```

recommend for userid = 1:
['168', '423', '385', '568', '405', '403', '550', '4', '222', '393']

recommend for userid = 100:
['307', '748', '301', '245', '332', '50', '312', '327', '678', '322']

recommend for userid = 233:
['172', '210', '195', '132', '173', '79', '186', '238', '191', '28']

recommend for userid = 666:
['98', '89', '50', '654', '405', '1', '117', '22', '228', '7']

recommend for userid = 888:
['174', '98', '216', '50', '405', '25', '168', '282', '121', '238']

```

图 1-5 ItemCF 算法推荐结果

可以发现两种推荐算法的推荐结果不同。

为比较个性化推荐算法带来的性能提示，先测试非个性化的基础推荐算法的性能，如表 1-1。

表 1-1 两种基础算法在 MovieLens 数据集下的性能

	准确率	召回率	覆盖率	流行度
Random	4.90%	4.60%	99.64%	3.0472
MositPopular	10.54%	9.90%	4.07%	5.9578

可以看到，两种基础算法体现了两个极端，Random 算法是随机进行推荐，故准确度和召回率最低，但其覆盖率最高；MostPopular 算法总是推荐欢迎度前 N 个电影，故其准确率和召回率较 Random 稍高，但其覆盖率很低，因为推荐的电影是固定的 N 部，且其流行度较高。

UserCF 算法的性在参数 K 不同取值下的结果如表 1-2。

表 1-2 UserCF 算法在不同 K 参数下的性能

K	准确率	召回率	覆盖率	流行度
5	16.22%	15.24%	34.89%	5.2858
10	18.25%	17.14%	27.65%	5.4057
20	19.68%	18.49%	22.20%	5.4927
40	19.69%	18.50%	17.95%	5.5623
80	19.30%	18.13%	13.17%	5.6195
160	18.63%	17.50%	10.83%	5.6642

可以看到，在  $K=40$  左右时 UserCF 算法达到最佳性能。同时，UserCF 算法的准确率和召回率都远高于 MostPopular 算法，体现了个性推荐算法的好处。随着  $K$  的增大，UserCF 算法的覆盖率逐渐降低，流行度逐渐升高，说明  $K$  越大，系统越倾向于推荐流行度高的电影，覆盖率随流行度的升高而降低。

在  $K=40$  时，对比 UserCF 算法和 UserCF-IIF 算法如表 1-3。

**表 1-3 UserCF 算法和 UserCF-IIF 算法性能对比**

	准确率	召回率	覆盖率	流行度
UserCF	19.69%	18.50%	17.95%	5.5623
UserCF-IIF	19.85%	18.65%	18.19%	5.5482

可以发现，加入惩罚机制的 UserCF-IIF 算法的性能略高于 UserCF 算法，这说明在计算用户的相似度时考虑电影的欢迎度对推荐结果的质量有帮助。

ItemCF 算法的性能在参数  $K$  不同取值下的结果如表 1-4。

**表 1-4 ItemCF 算法在不同  $K$  参数下的性能**

$K$	准确率	召回率	覆盖率	流行度
5	18.14%	17.04%	21.54%	5.5132
10	18.25%	17.14%	15.92%	5.6006
20	17.89%	16.80%	13.23%	5.6202
40	16.96%	15.93%	12.03%	5.6173
80	16.41%	15.41%	11.37%	5.6233
160	16.04%	15.07%	10.71%	5.6206

可以看到，在  $K=10$  左右，ItemCF 达到最佳性能。此外，与 UserCF 不同，参数  $K$  对流行度的影响不完全是正相关的，当  $K$  增大到一定程度时，流行度不再有明显变化，随着  $K$  的增大，覆盖率逐渐减小，这是因为推荐的电影逐渐趋于一个种类的  $N$  部电影。

在  $K=40$  时，对比 ItemCF 算法和 ItemCF-IUF 算法如表 1-5。

表 1-5 ItemCF 算法和 ItemCF-IUF 算法性能对比

	准确率	召回率	覆盖率	流行度
ItemCF	18.25%	17.14%	15.92%	5.6006
ItemCF-IUF	18.97%	17.82%	15.32%	5.6277

表 1-6 汇总了 6 种推荐算法的性能。

表 1-6 6 种推荐算法的性能对比

	准确率	召回率	覆盖率	流行度
UserCF	19.69%	18.50%	17.95%	5.5623
UserCF-IIF	19.85%	18.65%	18.19%	5.5482
ItemCF	18.25%	17.14%	15.92%	5.6006
ItemCF-IUF	18.97%	17.82%	15.32%	5.6277
Random	4.90%	4.60%	99.64%	3.0472
MositPopular	10.54%	9.90%	4.07%	5.9578

个性化推荐算法相对非个性化推荐算法的性能都有大幅提升，在相同数据集下，UserCF 算法的性能略高于 ItemCF 算法。UserCF 算法和 ItemCF 算法的四项评测指标随 K 值的对比分别如图 1-6、图 1-7、图 1-8、图 1-9 所示。

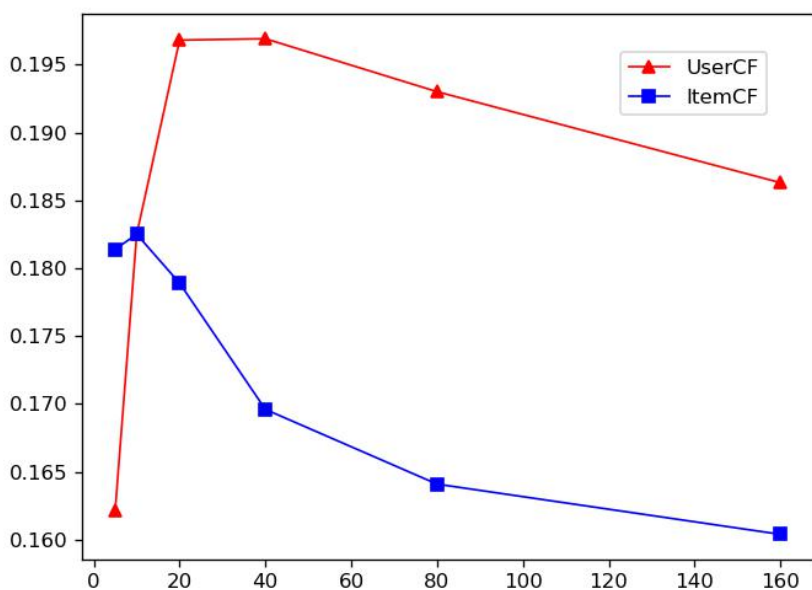


图 1-6 UserCF 和 ItemCF 算法在不同 K 值下的准确率曲线

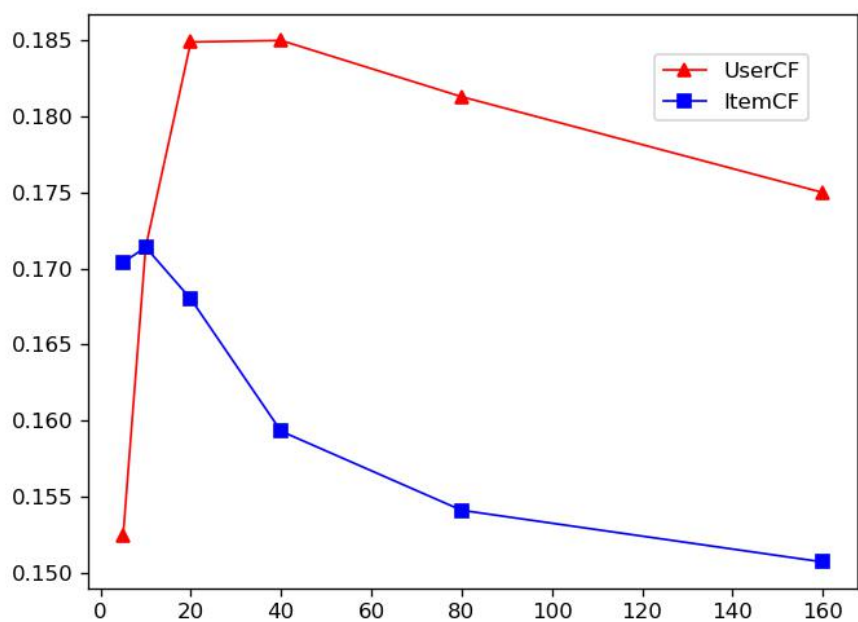


图 1-7 UserCF 和 ItemCF 算法在不同 K 值下的召回率曲线

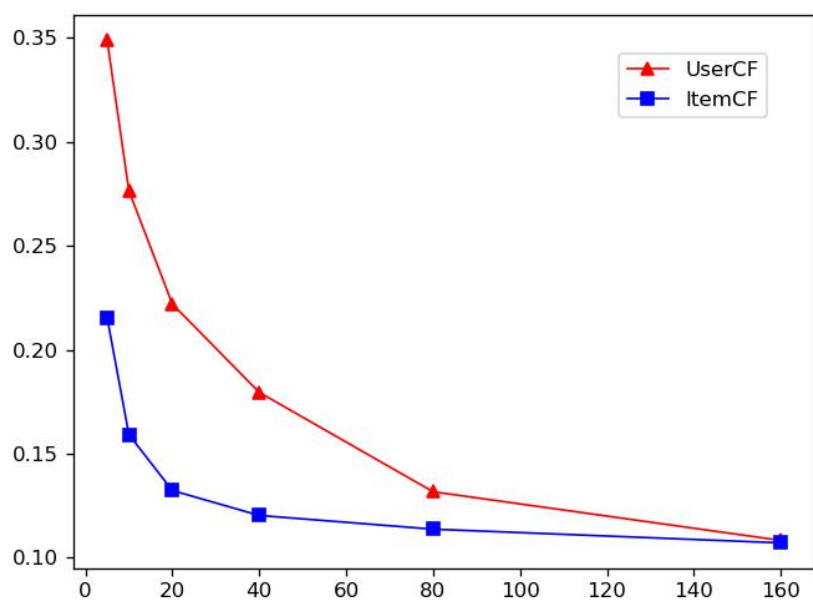


图 1-8 UserCF 和 ItemCF 算法在不同 K 值下的覆盖率曲线

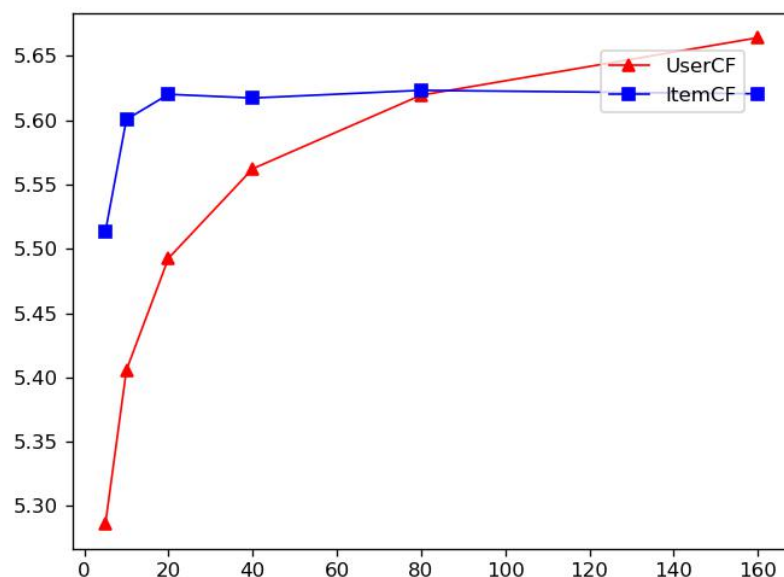


图 1-9 UserCF 和 ItemCF 算法在不同 K 值下的流行度曲线

从对比图中可观察到：

- (1) 两种算法的性能都对 K 值的选取有要求，要选取合适的 K 值；
- (2) 两种算法的覆盖率随 K 值增大逐渐减小，如此前分析，UserCF 算法中覆盖率减小的原因是推荐的电影趋于流行度高的电影，ItemCF 算法中覆盖率减小的原因是推荐的电影逐渐趋于同一种类的电影；
- (3) ItemCF 算法的流行度在达到合适 K 值前随 K 值增加而提升，到达 K 值之后趋于稳定，而 UserCF 算法的流行度随着 K 值的增加仍逐渐增加，只是增加的速率减缓。

## 1.5 思考与总结

本次课程设计的选题初衷来自于近期阅读的项亮撰写的《推荐系统实践》，从该本书中了解了协同过滤算法在推荐系统的应用，这使我产生了浓厚的兴趣。

总的来说，协同过滤算法是基于领域的算法，UserCF 的推荐结果着重于反映和用户兴趣相似的小群体的热点，而 ItemCF 的推荐结果着重于维系用户的历史兴趣，即 UserCF 的推荐更社会化，ItemCF 的推荐更个性化。考虑到这一因素，ItemCF 更加适用于电影推荐系统。UserCF 算法的缺点是用户数量较大时，算法



的执行效率较低，因为会建立庞大的相似性矩阵，但有的情形下 UserCF 算法比 ItemCF 算法更适用。例如对新闻推荐，新闻的更新速率远远大于用户的更新速率，而且对于新用户来说，完全推荐欢迎度高的热门新闻，而 ItemCF 算法对新用户的历史兴趣识别是困难的。

虽然在项目中实现了协同过滤算法，并进行了优化和比较，但协同过滤算法也存在着一些问题。比如著名的“哈利波特”问题，虽然在优化后的协同过滤算法中加入了惩罚机制，但对《哈利波特》这种非常火爆的书籍，即使加入惩罚机制，仍然不能很好地限制它的受欢迎度对相似性带来的贡献。针对这一问题，可以引入物品的内容数据以进行相似性计算。

另一方面，协同过滤算法存在冷启动问题，即一个新的用户没有任何历史记录时，或者一个新的物品添加进来时，又或者一个新的推荐系统建立之初，无法计算其相似度。这时非个性化推荐的作用就体现出来了，可以给用户进行 Random 推荐，使用户做出选择，产生初始的数据。

协同过滤算法是基于统计方法的，没有学习过程，模型即为一个用户选择的数据集，还有一些基于机器学习方法的算法，如 LFM（Latent Factor Model）算法，这是一种监督学习的算法，有良好的理论基础，它可以发掘物品的隐含语义模型，并通过这些隐含语义计算推荐度。在之后的学习过程中，我打算继续实现其他的推荐算法，并与协同过滤算法进行比较。