

**模式识别大作业**

题 目 电影推荐系统

学 院 信息科学与工程

专 业 控制科学与工程

组 员 谌鈫

指导教师 赵海涛

**完成日期： 2018 年 10 月26日**

随着互联网的发展，人们正处于一个信息爆炸的时代。相比于过去的信息匮乏，面对现阶段海量的信息数据，对信息的筛选和过滤成为了衡量一个系统好坏的重要指标。一个具有良好用户体验的系统，会将海量信息进行筛选、过滤，将用户最关注最感兴趣的信息展现在用户面前。这大大增加了系统工作的效率，也节省了用户筛选信息的时间。

**基于协同过滤的推荐**

协同过滤是一种在推荐系统中广泛采用的推荐方法。这种算法基于一个“物以类聚，人以群分”的假设，喜欢相同物品的用户更有可能具有相同的兴趣。基于协同过滤的推荐系统一般应用于有用户评分的系统之中，通过分数去刻画用户对于物品的喜好。协同过滤被视为利用集体智慧的典范，不需要对项目进行特殊处理，而是通过用户建立物品与物品之间的联系。

第一步：建立用户电影矩阵模型

协同过滤算法的输入数据一般用一个m\*n的用户评价矩阵Matrix表示，m是用户数，n是电影数，Matrix[ij]表示第i个用户对第j个电影的评价

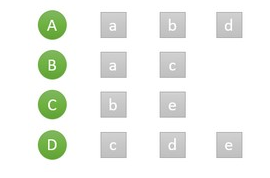


第二步：发现兴趣相似的用户

这一阶段主要完成目标用户最近的查找，通过计算目标用户与其他用户之间的相识度，得到与用户最近的邻居集，度量用户间的相似性，设N(u)为用户u喜欢的电影集合，N(v)为用户v喜欢的电影集合，u和v的相识度可通过以下式子进行计算：

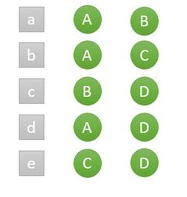
1. 采用Jaccard公式：
2. 采用余弦相似度：

这里选择余弦进行相识度度量计算，假设目前共有4个用户（A,B,C,D）,5部电影（a,b,c,d,e）,用户与电影关系如下图所示：

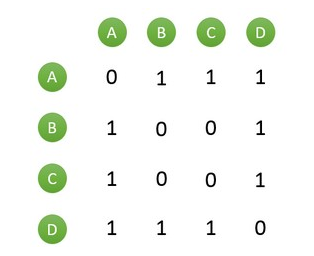


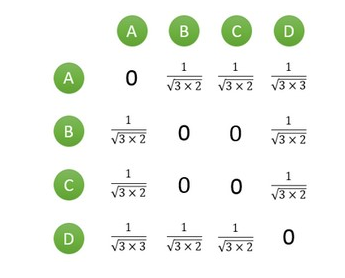


这种方法的时间复杂度是O(U^2),所以非常耗时。而在上表可以看到“用户-电影”表是一个稀疏矩阵，即N(u)^N(v)=0,如果换一下思路，可以先计算N(u)^N(v)！=0的用户，然后计算sqrt(N(u)\*N(v)),为此可以首先建立“电影-用户”的倒位表



设C[uv]=N(u)N(v),在倒排索引中假定用户u和用户v同时属于倒排索引中K部电影对应的用户列表，就有C[uv]=K,例如上图所示只有电影a中同时出来了用户有A和用户B，则矩阵中赋值1：





用户间的相似度计算就得到了，可以很直观的找到与目标用户兴趣相似的用户。

第三步：产生推荐项目

需要从矩阵中找到与目标用户最相似的K个用户，用集合S(u,K)表示，将S中用户喜欢的电影全部提取出来，并除去u已喜欢的电影，对每个候选电影i，用户对它的兴趣程度用以下公式计算：

表示用户V对电影i的喜欢程度，此处举例全部为1

假设给A推荐电影，选取K=3，对用户A，电影c,e没有看过，因此可以将这两部电影推荐给用户A,根据UerCF算法用户A对电c,e的兴趣计算p(A,c)和p(A,e)



即用户A对电影c和e的喜欢程度可能一样，在真实的推荐系统中考虑到用户的评分，最后根据得分排序取前K个即为推荐电影

实现代码如下：

|  |
| --- |
| # coding = utf-8 |
|  |  |
|  | # 基于用户的协同过滤推荐算法实现 |
|  | import random |
|  |  |
|  | import math |
|  | from operator import itemgetter |
|  |  |
|  |  |
|  | class UserBasedCF(): |
|  | # 初始化相关参数 |
|  | def \_\_init\_\_(self): |
|  | # 找到与目标用户兴趣相似的20个用户，为其推荐10部电影 |
|  | self.n\_sim\_user = 20 |
|  | self.n\_rec\_movie = 10 |
|  |  |
|  | # 将数据集划分为训练集和测试集 |
|  | self.trainSet = {} |
|  | self.testSet = {} |
|  |  |
|  | # 用户相似度矩阵 |
|  | self.user\_sim\_matrix = {} |
|  | self.movie\_count = 0 |
|  |  |
|  | print('Similar user number = %d' % self.n\_sim\_user) |
|  | print('Recommneded movie number = %d' % self.n\_rec\_movie) |
|  |  |
|  |  |
|  | # 读文件得到“用户-电影”数据 |
|  | def get\_dataset(self, filename, pivot=0.75): |
|  | trainSet\_len = 0 |
|  | testSet\_len = 0 |
|  | for line in self.load\_file(filename): |
|  | user, movie, rating, timestamp = line.split(',') |
|  | if random.random() < pivot: |
|  | self.trainSet.setdefault(user, {}) |
|  | self.trainSet[user][movie] = rating |
|  | trainSet\_len += 1 |
|  | else: |
|  | self.testSet.setdefault(user, {}) |
|  | self.testSet[user][movie] = rating |
|  | testSet\_len += 1 |
|  | print('Split trainingSet and testSet success!') |
|  | print('TrainSet = %s' % trainSet\_len) |
|  | print('TestSet = %s' % testSet\_len) |
|  |  |
|  |  |
|  | # 读文件，返回文件的每一行 |
|  | def load\_file(self, filename): |
|  | with open(filename, 'r') as f: |
|  | for i, line in enumerate(f): |
|  | if i == 0: # 去掉文件第一行的title |
|  | continue |
|  | yield line.strip('\r\n') |
|  | print('Load %s success!' % filename) |
|  |  |
|  |  |
|  | # 计算用户之间的相似度 |
|  | def calc\_user\_sim(self): |
|  | # 构建“电影-用户”倒排索引 |
|  | # key = movieID, value = list of userIDs who have seen this movie |
|  | print('Building movie-user table ...') |
|  | movie\_user = {} |
|  | for user, movies in self.trainSet.items(): |
|  | for movie in movies: |
|  | if movie not in movie\_user: |
|  | movie\_user[movie] = set() |
|  | movie\_user[movie].add(user) |
|  | print('Build movie-user table success!') |
|  |  |
|  | self.movie\_count = len(movie\_user) |
|  | print('Total movie number = %d' % self.movie\_count) |
|  |  |
|  | print('Build user co-rated movies matrix ...') |
|  | for movie, users in movie\_user.items(): |
|  | for u in users: |
|  | for v in users: |
|  | if u == v: |
|  | continue |
|  | self.user\_sim\_matrix.setdefault(u, {}) |
|  | self.user\_sim\_matrix[u].setdefault(v, 0) |
|  | self.user\_sim\_matrix[u][v] += 1 |
|  | print('Build user co-rated movies matrix success!') |
|  |  |
|  | # 计算相似性 |
|  | print('Calculating user similarity matrix ...') |
|  | for u, related\_users in self.user\_sim\_matrix.items(): |
|  | for v, count in related\_users.items(): |
|  | self.user\_sim\_matrix[u][v] = count / math.sqrt(len(self.trainSet[u]) \* len(self.trainSet[v])) |
|  | print('Calculate user similarity matrix success!') |
|  |  |
|  |  |
|  | # 针对目标用户U，找到其最相似的K个用户，产生N个推荐 |
|  | def recommend(self, user): |
|  | K = self.n\_sim\_user |
|  | N = self.n\_rec\_movie |
|  | rank = {} |
|  | watched\_movies = self.trainSet[user] |
|  |  |
|  | # v=similar user, wuv=similar factor |
|  | for v, wuv in sorted(self.user\_sim\_matrix[user].items(), key=itemgetter(1), reverse=True)[0:K]: |
|  | for movie in self.trainSet[v]: |
|  | if movie in watched\_movies: |
|  | continue |
|  | rank.setdefault(movie, 0) |
|  | rank[movie] += wuv |
|  | return sorted(rank.items(), key=itemgetter(1), reverse=True)[0:N] |
|  |  |
|  |  |
|  | # 产生推荐并通过准确率、召回率和覆盖率进行评估 |
|  | def evaluate(self): |
|  | print("Evaluation start ...") |
|  | N = self.n\_rec\_movie |
|  | # 准确率和召回率 |
|  | hit = 0 |
|  | rec\_count = 0 |
|  | test\_count = 0 |
|  | # 覆盖率 |
|  | all\_rec\_movies = set() |
|  |  |
|  | for i, user, in enumerate(self.trainSet): |
|  | test\_movies = self.testSet.get(user, {}) |
|  | rec\_movies = self.recommend(user) |
|  | for movie, w in rec\_movies: |
|  | if movie in test\_movies: |
|  | hit += 1 |
|  | all\_rec\_movies.add(movie) |
|  | rec\_count += N |
|  | test\_count += len(test\_movies) |
|  |  |
|  | precision = hit / (1.0 \* rec\_count) |
|  | recall = hit / (1.0 \* test\_count) |
|  | coverage = len(all\_rec\_movies) / (1.0 \* self.movie\_count) |
|  | print('precisioin=%.4f\trecall=%.4f\tcoverage=%.4f' % (precision, recall, coverage)) |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | rating\_file = 'D:\\学习资料\\推荐系统\\ml-latest-small\\ratings.csv' |
|  | userCF = UserBasedCF() |
|  | userCF.get\_dataset(rating\_file) |
|  | userCF.calc\_user\_sim() |
|  | userCF.evaluate() |
|  |  |
|  |  |

