基于用户协同过滤算法的电影推荐系统

摘　要

随着电子商务的高速发展和普及应用，个性化推荐的推荐系统已成为一个重要研究领域。个性化推荐算法是推荐系统中最核心的技术，在很大程度上决定了电子商务推荐系统性能的优劣。协同过滤是应用最为广泛的一种个性化推荐技术。协同过滤主要分为基于用户的协同过滤和基于项目的协同过滤。

本文研究了基于用户的协同过滤推荐算法及其在电影推荐系统中的应用，设计开发了相应的电影推荐系统中个性化推荐原型系统，并对该算法的推荐质量进行了深入的实验分析。本文也介绍了协同过滤推荐的基本思想。在介绍电影推荐系统推荐技术研究与应用现状的基础上，详细说明了基于用户的协同过滤推荐算法及其具体实现步骤。采用 Java Web 实现了原型系统。对于挖掘结果从算法效率及应用意义上进行比较分析

关键词：电影　基于用户的协同过滤推荐　余弦相似性 相关相似性

目录

第 1 章 绪论

1.1、研究背景

随着互联网技术的迅猛发展，人们逐渐走入了信息过载的时代。面对大量的信息，我们都显得有些无所适从。作为信息需求者，从大量信息中找到自己感兴趣的信息往往是一件非常困难的事情；而对于信息提供者，让自己的信息脱颖而出，受到大家的关注，也是一件非常困难的事情。推荐系统就是解决这一矛盾的重要工具。推荐系统通过建立用户和信息产品之间的关系，利用已有的选择过程或相似性关系，一方面挖掘用户潜在感兴趣的信息，另一方面让信息能够展现在对它感兴趣的用户面前。一个完整的推荐系统通常包含收集用户信息的行为记录模块，分析用户喜好的模型分析模块和推荐算法模块。

我们身边最熟悉的例子要数电子商务网站的推荐系统，几乎每个大型电子商务网站都把个性化推荐作为重要的营销手段之一。更有文献表明早期 Amazon 的 35%销售增量都来自它的推荐系统。除了商品，音乐、电影等也是常见的推荐对象。

在众多的个性化推荐算法中，协同过滤被广泛应用，也是最成功的推荐算法。本课题旨在研究基于用户的协同过滤推荐算法在电子商务个性化商品推荐中的应用。

研究电子商务推荐系统对企业和社会具有很高的经济价值。电子商务个性化推荐系统的关键是建立用户模型。推荐系统的热点问题是推荐技术和推荐算法的研究。推荐算法是整个推荐系统的核心，它的性能决定了最终推荐结果的好坏。为了建立合理的用户模型，满足不同用户对实时性、推荐方式等的要求，产生了一系列的推荐技术和算法。涉及的技术包括基于内容的过滤技术、协同过滤技术、关联规则挖掘技术、分类和聚类技术、神经网络技术等等。

个性化的服务在商家与顾客之间建立起了一条牢固的纽带。顾客越多地使用推荐系统。推荐系统可以更适合顾客的需要，将顾客更多地吸引到自己的网站，与顾客建立长期稳定的关系。从而能有效保留用户，防止用户流失。

个性化推荐技术是电子商务推荐系统中最核心最关键的技术，很大程度上决

定了电子商务推荐系统性能的优劣[4].

1.2、国际发展形势

随着互联网的普及以及电子商务的发展，推荐系统渐渐成为电子商务技术的一个重要研究内容，得到了越来越多的研究者的关注[5]。

国外在这方面的研究起步较早。1997 年 Resnick&Varian 给出了电子商务推荐系统的正式定义[6]。从 1999 年开始，计算机协会 ACM(Association for Computing Machinery)每年召开一次电子商务的研讨会，研究文章中关于电子商务推荐系统的占据了很大比重；该协会下面的数据挖掘特别兴趣组 SIGKDD(Special Interest Group on KDD)和信息检索兴趣小组 SIGIR(Special Interest Group on Information Retrieval)也分别在 1999 年和第 24 届研究发展会议上，开始把推荐系统作为一个专门的研究主题。同时，第十五届人工智能会议、第一届知识管理应用会议等也纷纷将电子商务推荐系统作为研究主题。

第 2 章 基于用户协同过滤推荐技术

2.1 电子商务推荐系统概述

电子商务对传统的商务交易产生了革命性的变化,从而要求“以产品为中心”向“面向客户”、“以客户为中心”的新的商业模式的转变,要求电子商务网站按客户群划分产品,围绕客户进行服务,为客户提供所需要的东西,所以对每个顾客提供个性化的服务成为必要。在这种背景下,推荐系统(Recommender System)应运而生,它是根据用户个人的喜好、习惯来向其推荐信息、商品的程序[7]。电子商务网站可以使用推荐系统分析客户的消费偏好,向每个客户具有针对性地推荐产品,帮助用户从庞大的商品目录中挑选真正适合自己需要的商品,尽可能为每个顾客提供个性化的服务。

个性化推荐(personalized recommendation)技术通过研究不同用户的兴趣，主动为用户推荐最需要的资源，从而更好地解决互联网信息日益庞大与用户需求之间的矛盾。目前，推荐技术被广泛应用到电子商务、数字图书馆、新闻网

站等系统中[8]。各种适用于推荐系统的技术应运而生，如协同过滤技(CF)、bayesian 网技术、聚类分析技术、关联规则技术、神经网络技术和图模型技术等[9]，其中，协同过滤是应用最为广泛的个性化推荐技术[10]。协同过滤推荐又分为基于模型(Model-based)的协同过滤和基于用户的协同过滤。后来，sarwr 教授在 2001 年提出基于项目的协同过滤算法[4]。

推荐技术的分类标准：文献[11]给出了区别推荐技术的两维属性：① 自动化程度(degree of automation)，客户要得到推荐系统的推荐是否需要显式的输入信息；②持久性程度(degree of persistence)，推荐系统产生推荐是基于客户当前的单个会话(session)还是基于客户的多个会话。也有学者认为，除了上述两个特征外，个性化程度(degree of personalization)也是评价推荐技术的重要指标，可以用来反映推荐结果符合用户兴趣爱好的程度。

2.2 协同过滤推荐技术

协同过滤主要是以属性或兴趣相近的用户经验与建议作为提供个性化推荐的基础。透过协同过滤，有助于搜集具有类似偏好或属性的用户，并将其意见提供给同一集群中的用户作为参考，以满足人们通常在决策之前参考他人意见的心态。

协同过滤推荐(collaborative filtering recommendation)是目前研究最多的个性化推荐技术，它基于邻居用户的资料得到目标用户的推荐，推荐的个性化程度高。著名的系统有 GroupLens/ Net Percep2tions,Ringo/ Firefly 及 Tapest ry 等。协同过滤的最大优点是对推荐对象没有特殊要求，能处理非结构化的复杂对象，如音乐、电影。

协同过滤推荐主要分为两类:一是基于内存的协同过滤(memory - based collaborative filtering)，先用相似统计的方法得到具有相似兴趣爱好的邻居用户，所以该方法也称基于用户的协同过滤(user -based collaborative filtering)或基于邻居的协同过滤(neighbor - based collaborative filtering)；二是基于模型的协同过滤(model - based collaborative fil2tering)，先用历史数据得到一个模型，再用此模型进行预测。基于模型的推荐广泛使用的技术包括神经网络等学习技术、潜在语义检索(latent seman2tic indexing)和贝叶斯网络(bayesian networks)，训练一个样本得到模型。Breese 教授认为基于用户的协同推荐比基于模型的协同推荐方法更好[4]。

本文主要研究的是基于用户的协同过滤推荐算法的实现。

基于用户的协同推荐算法随着用户数量的增多，计算量成线性加大，其性能越来越差 ，并且不能对推荐结果提供很好的解释。为此，在 2001 年 Sar2wr 教授提出了第三种协同过滤推荐算法，即基于项目的协同推荐算法(item - based collaborative filter2ing algorithms)。该算法通过先计算已评价项目和待预测项目的相似度，以相似度作为权重，加权各已评价项目的评价分，得到预测项目的预测值，并指出基于项目的推荐算法比基于用户的推荐算法还要好，且能解决基于用户的协同推荐的两个问题。但 Mild 教授从批判的角度重新审视了各种推荐算法，指出基于项目的协同推荐并不一定好，算法准确度与采用的实验规模数据有关，大多数情况下还是基于用户的协同推荐好[12]。

协同过滤的出发点是:兴趣相近的用户可能会对同样的东西感兴趣。所以，只要维护关于用户喜好的数据，从中分析得出具有相似口味的用户，然后就可以根据相似客户的意见来向其进行推荐。另一种可能的出发点是:用户可能较偏爱与其已购买的东西相类似的商品。可以根据用户对各种东西的评价来判断商品之间的相似程度，然后推荐与用户兴趣最接近的那些商品。前一种思路以客户与客户之间的关系为中心，而后一种思路则以项目与项目之间的关系为着眼点。协同过滤推荐的个性化程度高，目前有许多网站采用了基于该技术的推荐系统，如 Amazon.com,CDNow.com, MovieRinder.com 等。

基于“兴趣相近的用户可能会对同样的东西感兴趣”这一出发点，本文主要研究基于用户的协同过滤推荐算法。

第 3 章 基于用户的协同过滤推荐算法

3.1 基于用户协同过滤算法的介绍

基于用户的协同过滤是个性化推荐中应用最为广泛的方法，它是基于邻居用户的兴趣爱好预测目标用户的兴趣偏好。算法先使用统计技术寻找与目标用户有相同喜好的邻居，然后根据目标用户的邻居的偏好产生向目标用户的推荐[13]。

它的基本原理是利用用户访问行为的相似性来互相推荐用户可能感兴趣的资源对当前用户，系统通过其历史访问记录及特定相似度函数，计算出与其访问行为（购买的产品集合、访问的网页集等）最相近的 N 个用户作为用户的最近邻居集，统计的近邻用户访问过而目标用未访问的资源生成候选推荐集，然后计算候选推荐集中每个资源对用户的推荐度，取其中 K 个排在最前面的资源作为

用户的推荐集

。

3.2、基于用户喜好值推荐算法的思路

其基本思想是：通过计算用户对项目评分之间的相似性，搜索目标用户的最近邻居，然后根据最近邻居的评分向目标用户产生推荐[8]。

典型的协同过滤算法是基于用户的。协同过滤推荐算法的实现过程分为 3 步：建立用户矩阵模型、通过用户相似度寻找最近邻居和产生推荐项目[8]。

3.2.1 建立用户模型

如表 2.1，协同过滤算法的输入数据通常表示为一个 m*n 的用户—评价矩阵 R，m 是用户数，n 是项目数，其中 Rij 表示第 i 个用户对第 j 个项目的评分值；

表 2.1 用户-项目评分矩阵

Table 2.1 The ratings matrix of user-project

	$Item_1$	$Item_2$	……	$Item_j$	……	$Item_n$
$User_1$	4	5	…	1	…	0
$User_2$	2	3	…	0	…	4
……	…	…	…	…	…	…
$User_i$	1	0	…	R_{ij}	…	3
……	…	…	…	…	…	…
$User_m$	3	2	…	2	…	3

3.2.2 寻找最近邻居

在这一阶段，主要完成对目标用户最近邻居的查找。通过计算目标用户与其他用户之间的相似度，算出与目标用户最相似的“最近邻居”集。即： 对 目标用户 u 产生一个以相似度 sim(u，v)递减排列 的“ 邻居” 集合。该过程分两步完成： 首先计算用户之间的相似度，可采用皮尔森相关系数、余弦相似性和修正的余弦相似性等度量方法[9] ，其次是根据如下方法选择“ 最近邻居”：(1)选

择相似度大于设定阈值的用户；(2)选择相似度最大的前 k 个用户；(3) 选择相似度大于预定阈值的 k 个用户。

1. 余弦相似性（Cosine）

每一个用户的评分都可以看作为 n 维项目空间上的向量，如果用户对项目没有进行评分，则将用户对该项目的评分设为 0。用户间的相似性通过向量间的余弦夹角度量。设用户 i 和用户 j 在 n 维项目空间上的评分分别表示为向量 $\vec{i}$ 和向量 $\vec{j}$，则用户 i 和用户 j 之间的相似性 $sim(i,j)$ 为：

$$sim(i,j)=\frac{\vec{i}\cdot\vec{j}}{\|\vec{i}\|\cdot\|\vec{j}\|}$$

其中，分子为两个用户评分向量的内积，分母为两个用户向量模的乘积。

2. 相关相似性(Correlation)

设用户 i 和用户 j 共同评分过的项目集合用 I_{ij} 表示，$I_{ij}=I_i\cap I_j$，则用户 i 和用户 j 之间的相似性 $sim(i,j)$ 通过 Pearson 相关系数度量：

$$sim(i,j)=\frac{\sum_{d\in I_{ij}}(R_{i,d}-\overline{R_i})(R_{j,d}-\overline{R_j})}{\sqrt{\sum_{d\in I_{ij}}(R_{i,d}-\overline{R_i})^2}\sqrt{\sum_{d\in I_{ij}}(R_{j,d}-\overline{R_j})^2}}$$

其中，$R_{i,d}$ 表示用户 i 对项目 d 的评分，$\overline{R_i}$、$\overline{R_j}$ 分别表示用户 i 和用户 j 对所打分项目的平均评分。

3.2.2 产生推荐项目：

计算方法如下 ：

$$P_{i,d}=\overline{R_i}+\frac{\sum_{j\in NBSi}sim(i,j)*(R_{j,d}-\overline{R_j})}{\sum_{j\in NBSi}(|sim(i,j)|)}$$

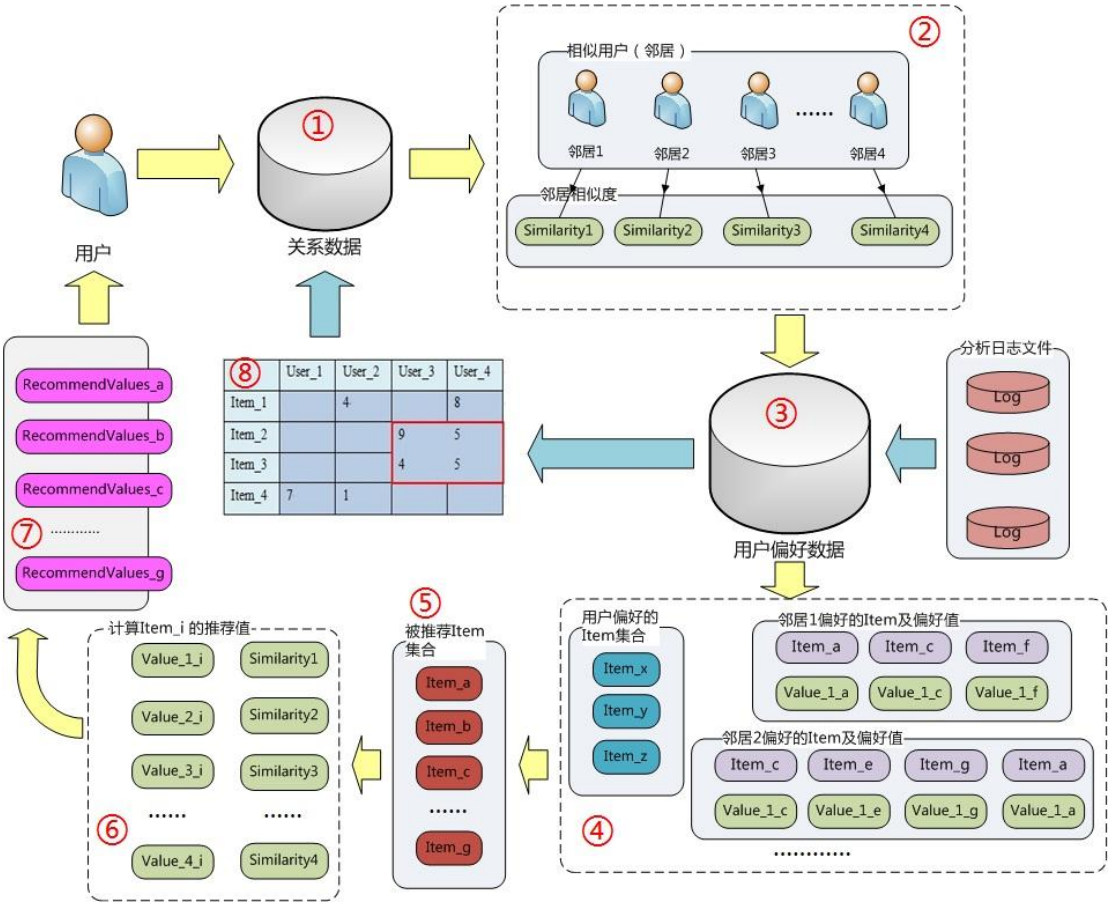
其中 $sim(i,j)$ 表示用户 i 与用户 j 之间的相似性，$R_{j,d}$ 表示最近邻居用户 j 对项目 d 的评分，$\overline{R_i}$ 和 $\overline{R_j}$ 分别表示用户 i 和用户 j 的平均评分，公式(1)的实质是在用户的最近邻居集 NESi 中查找用户，并将目标用户与查找到的用户的相似度的

值作为权值，然后将邻居用户对该项目的评分与此邻居用户的所有评分的差值进行加权平均。

通过上述方法预测出目标用户对未评价项目的评分，然后选择预测评分最高的 TOP-N 项推荐给目标用户。

3.3 算法的实现

1．算法流程图



2.算法代码（见附录）

3.4 电影推荐系统界面实现

电影智能推荐

用户号:1

推荐的电影是:

电影号：313喜好程度:4.5

电影号：748喜好程度:4.5

电影号：288喜好程度:4.0

电影号：300喜好程度:4.0

电影号：307喜好程度:3.0

电影智能推荐

用户号:1

推荐的电影是:

电影号：313喜好程度:4.5

电影号：748喜好程度:4.5

电影号：288喜好程度:4.0

电影号：300喜好程度:4.0

电影号：307喜好程度:3.0

第 4 章 算法推荐质量的实验分析

4.1 算法推荐质量的衡量方法

评价推荐系统推荐质量的度量标准主要包括统计精度度量方法和决策支持精度度量方法两类。统计精度度量方法中的平均绝对偏差 MAE 易于理解，可以直观地对推荐质量进行度量，是最常用的一种推荐质量度量方法。本文采用平均绝对偏差 MAE 作为度量标准，通过计算预测的用户评分与实际的用户评分之间的偏差度量预测准确性，MAE 越小，推荐质量越高。

设预测的用户评分集合表示为{p1, p2,……, pN}，对应的实际用户评分集合表示为{q1,q2,……,qN}，则平均绝对偏差 MAE 定义为：

$$MAE = \frac{\sum_{i=1}^{N} | p_i - q_i |}{N} \tag{5-1}$$

因此想要真正的评价算法的推荐质量，必须要采用真实的数据，这样才能和预测的评分进行对比，并经过反复试验，计算 MAE 的值，并分析试验结果。

4.2 验证推荐方法采用的数据集

本系统采用了 MovieLens 数据集进行试验，并计算用余弦相似性方法计算相似性的平均绝对偏差 MAE 值。

MovieLens 是历史最悠久的推荐系统。它由美国 Minnesota 大学计算机科学与工程学院的 GroupLens 项目组创办，是一个非商业性质的、以研究为目的的实验性站点。

本实验所用的数据集是直接从 GroupLens 网站上下载的，此数据集包含了 943 名用户对 1682 部电影的评价，这些数据是 MovieLens 在 1997 年九月 19 日到 1998 年 4 月 22 日这七个月收集的数据，这个数据集是经过整理的，评分不到 20 项的已经被移除了，而且，已经随机分好了五个试验组，每个试验组包括

完整的数据，包括一个训练集和一个测试集，大约是按照 80%是训练集，20%是测试集分的。其中 u1base 是第一组的训练集，u1test 是第一组的测试集。评分值都为 1 到 5 之间的整数，数值越高，表明用户对该电影的偏爱程度越高。

4.3 实验结果分析

本文实验从数据集中抽取了 500 个用户。本文首先这 500 个用户对未评分项目的评分，再把它们分为两组，选择 300 个用户作为训练集（训练集大小依次变化为 300，200,100 个用户），剩下的 200 个用户作为测试集。然后采用只给出测试用户的 5 个实际评分、10 个实际评分和 20 个实际评分的方式，对两种算法的 MAE 进行比较分析。

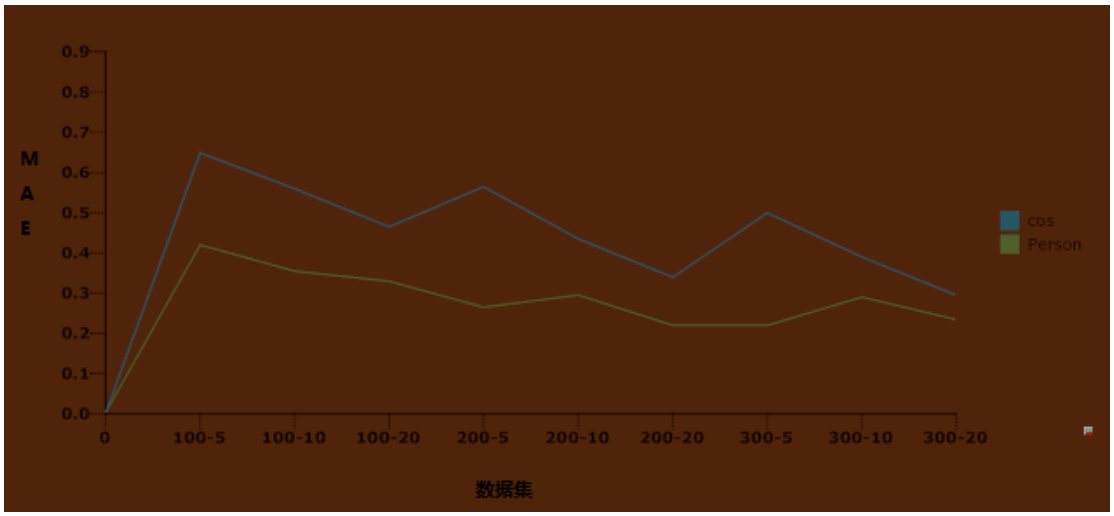
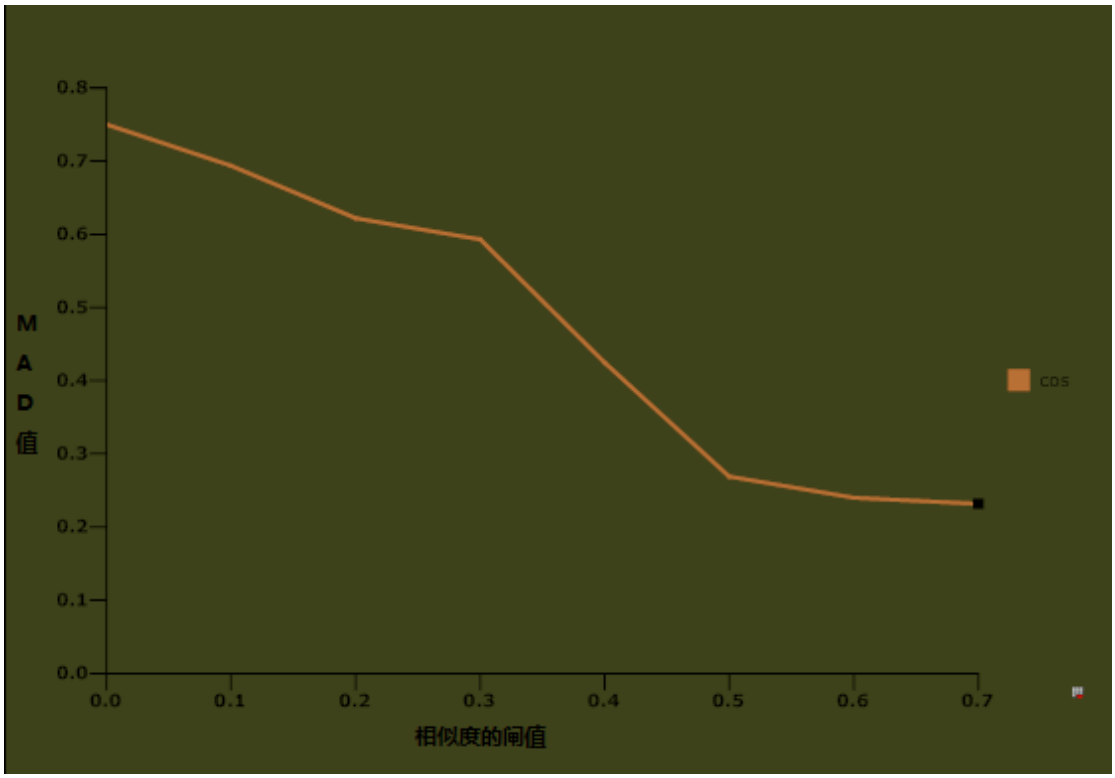


图　　算法的 MAE 比较

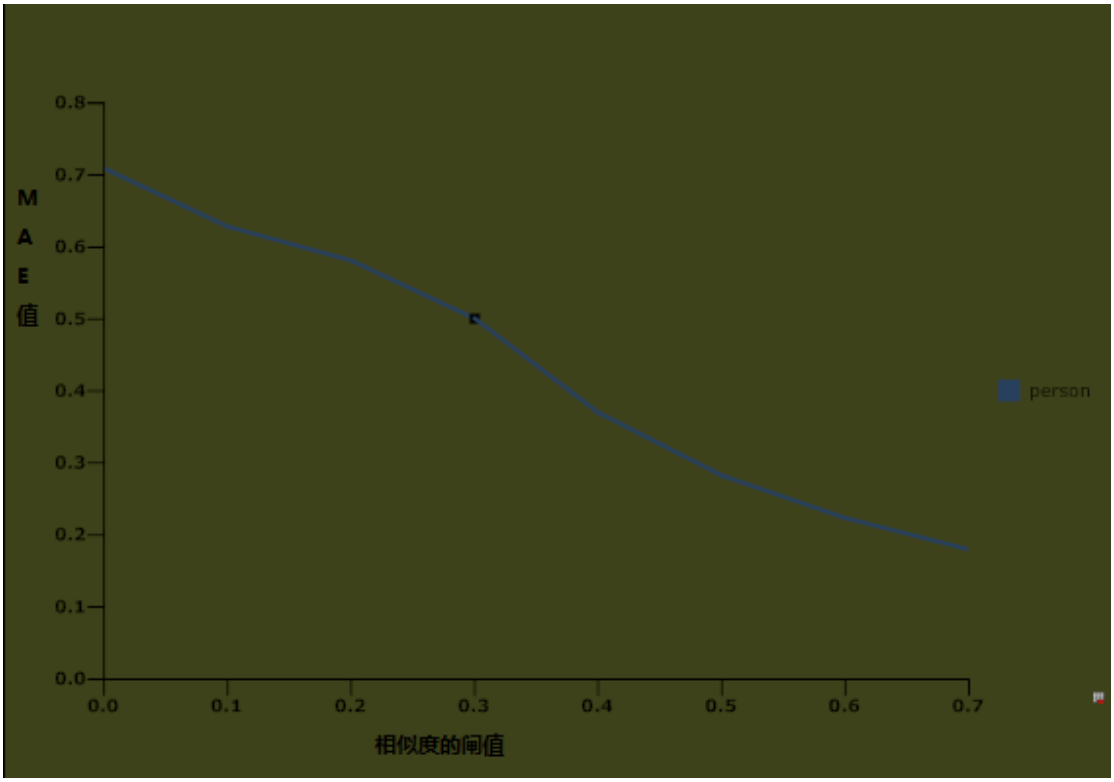
4.3.1 余弦相似性

利用上面介绍的余弦相似性的公式计算用户之间相似性，这个算法看似简单，仍然有很多问题需要注意，如随着数据集的增大，这个算法的运算量呈线性增长，这是一个很重要的问题，对整个 MAE 值的计算有很大影响。



4.3.2 相似相关性

相似相关性的计算就是通过上面介绍的相关相似性的计算公式来实现的。相似相关性会存在极端情况，例如，当用户用户 i 和用户 j 共同评过分的项目为一个项目时，这时利用这个公式计算得到的相似性为 1，而一般情况下相似性的值是小于一的，显然这时的相似性要大于其他用户与目标用户的相似性，若取 j 为 i 的最近邻居，却不一定合理，因为他们共同评过分的那一个项目很可能是巧合，就算品味不同的两个人也可能会喜欢同一件商品的。因此，在利用相关相似性计算最近邻居时，本试验不把与目标用户相似性为 1 的那个用户视为目标用户的最近邻居，只是把这样的情况看做是极端情况。



4.3.3 基于用户评分次数的相似相关性计算方法

本人写了一种基于用户评分次数的相似性的方法，来对比这几种方法。

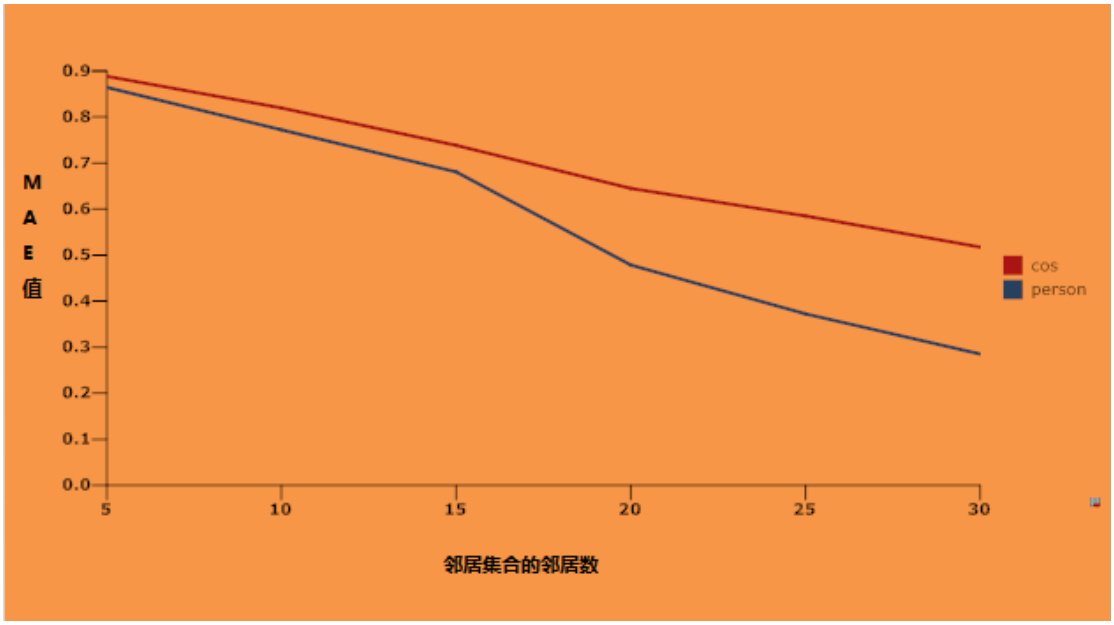
算法的基本思想：从用户间相似性计算公式可以看出，对于相似度较高的用户，所评价的项目数量相差较小，即所评价项目数量相差较大的用户间相似度较低；对于所评价项目数量相近的用户，相似度越高，表明用户共同测评项目数所占的比例越大，且测评分值也越相近，由此可知用户的兴趣爱好越相似[14]。

因为每个用户都评过分，用户一共评过分的次数就作为这个用户的评分次数，对于没有评过分的用户，评分次数为零。在计算用户的最近邻居时，不是在整个用户空间查找最近邻居，而是先获取这个用户的评分次数，那些评分次数在这个次数某个范围内的用户进入候选邻居集，这里波动范围用 a 表示，a 是算法的一个参数，如果目标用户的评分次数为 18，则在计算相似性时，只计算评分次数在(18-a)与(18+a)之间的用户与目标用户的相似性。然后，取最高的几个作为最近邻居集。而那些评分次数不在(18-a)到(18+a)范围内的用户都认为和目标用户相差较大，直接不计算其与目标用户的相似性。

4.3.4 两种算法的对比分析

表 5-1　余弦相似性的 MAE

邻居集的取值	MAE	运行起始时间	运行结束时间
5	0.8801	2:36:42	2:51:50
10	0.8628	9:43:02	9:57:22
15	0.8426	13:34:41	13:49:28



上面的截图是采用数据集 1 得出的结果，这里的数据集 1 是我为了运算速度的考虑在原来的 u1base 数据集的基础上进行一定删减得到的，我把 base 表中 consumerid 大于 300 以及 goodsid 大于 800 的记录都删掉了，test 表也做了同样的操作。第一条记录是最近邻居集取 7 的记录，即目标用户的最近邻居选取 7 个。第二条记录是邻居集取 10 的情况，第三条记录是邻居集取 15 的情况。

对比上图可知，邻居集取的越大，得到的 MAE 值越小，即邻居集取的越大，推荐的质量越好。

一般情况下，通过相似相关性方法计算得到的 MAE 值应该要小于用余弦相似性方法计算相似性得到的 MAE 值，即相关相似性的推荐质量要好于余弦相似性。而基于评分次数的 MAE 值要大于相关相似性的 MAE 值，但相应的时间要

小很多。

由于数据集的庞大，导致整个算法的调试很困难，我也只是尽量的改进算法，以使运行速度尽量加快，实验表明，完全不与数据库直接交互的方法，不一定是最快的，有时候采用与数据库直接交互与从缓存中读数据相结合的方法会更快一些。

第五章　总结

协同过滤推荐(collaborative filtering recommendation)是目前研究最多的个性化推荐技术，推荐的个性化程度高。

协同过滤的出发点：(1)兴趣相近的用户可能会对同样的东西感兴趣，(2)用户可能较偏爱与其已购买的东西相类似的商品。

本系统主要是为了研究学习基于用户的协同过滤推荐算法的实现，并对算法进行深入的分析研究。经过对比试验，在一定范围内，随着最近邻居集数量的增加，推荐的质量会相对较好，在同一数据集的对比试验中，最近邻居集取的个数越多，其 MAE 值会相对变小，而最近邻居集得数量对算法的运算时间一般影响不大。

我觉得关于协同过滤的推荐算法是一个很有趣的课题，但这个需要我们有很大的耐心，因为中间可能会出现很多错，即使是一个小的误区，可能导致结论相差很多，而一般自己写的代码，自己是怎么都看不出问题的，只有通过不断试验，得出一些数据，才有可能发现自己存在的问题，然后再改进。在这个算法的整个研究过程中，由于数据量的庞大，还要不断改进算法，尽量使其运行的更快。

经过本次试验，虽然对基于用户的协同过滤推荐算法的研究没有深入到能提出一些建设性的改进，但对于算法已经有了一定的了解，对以后更深入的研究打下了扎实的基础。

参考文献

[1]基于多目标优化双聚类的数字图书馆协同过滤推荐系统。刘飞飞

[2]基于双聚类模型的协同过滤推荐引擎设计。康美林

[3]基于关联规则的图书销售网站个性化推荐系统设计与实现。王静

[4]高 滢，齐 红，刘亚波，刘大有. 基于用户等级的协同过滤推荐算法[J].吉林大学学报(理学版)，2008，46(3)：489~493.

[5] 周强. 基于用户的协同过滤推荐算法研究[J].南昌高专学报，2006，(3)：88~89.

[6] 高 滢，齐 红，刘亚波，刘大有. 基于用户等级的协同过滤推荐算法[J]. 吉林大学学报(理学版)，2008，46(3)：489~493.

附录

基于用户协同过滤算法（Java）代码

算法代码：

```
public class XieTong {

    /**
     * @param args
     */
    private int tjs=0;   //推荐的电影数
    private int neiborhoodnum=0;  //邻居的数量
    private int Count;     //最大的电影编号
    private int UserNum;   //用户数量
    private List<Person> Users;  //用户对象
    private int c;     //目标用户
    private double[][] Similarity;  //相似用户的相似值
    private double[][] Array;  //用户-评分矩阵
    private int[] Neiborhood;   //目标用户的邻居

public XieTong(List<Person> user,int c,int tjs) { //带参数的构造函数，初始化对象

    super();
    Users=user;
    this.c=c;
    this.tjs=tjs;
    Count=0;
    UserNum=0;
    Iterator<Person> it=Users.iterator();
    int temp=0;
    while(it.hasNext()){
        Person p=new Person();
        p=it.next();
        int uid=p.getUserID();
        int mid=p.getMovieID();
        if(temp!=uid){
            UserNum++;
```

```
            temp=uid;
        }

        if(mid>Count){
            Count=mid;
        }

    }
    Array=new double[UserNum+1][Count+1];
}
public void juzhen(){   //形成用户-评分矩阵
    Iterator<Person> it=Users.iterator();  //用户迭代器
        while(it.hasNext()){
            Person p=new Person();  //一个用户对象
            p=it.next();
            int uid=p.getUserID();    //获得用户的ID
            int mid=p.getMovieID();      //获得电影的ID
            double pfc=p.getPreference();  //获得用户对电影的喜好值
            Array[uid][mid]=pfc;  //形成矩阵


        }

}

public  double[][] Similarity(){ //计算用户相似度

        Similarity=new double[UserNum+1][UserNum+1]; //初始化相似度对象
        double rsx,rsy;  //用户x，y对所看过电影的评价分

        for(int i=1;i<UserNum+1;i++){
            double rcx=0,sx=0;

            for(int m=1;m<Count+1;m++){   //计算用户i的平均评分
                if(Array[i][m]!=0){
                    rcx+=Array[i][m];
                    sx++;
                }
            }
            for(int j=1;j<UserNum+1;j++){  //计算用户j的平均评分
                double rcy=0,sy=0;
                for(int n=1;n<Count+1;n++){
                if(Array[j][n]!=0){
                    rcy+=Array[j][n];
```

```
                    sy++;
                }
            }
                double rxy = 0;
                double rx = 0;
                double ry = 0;
                rsx=rcx/sx;
                rsy=rcy/sy;
                for(int k=1;k<Count+1;k++){   //通过皮尔森相关系数计算用户i,对
于j的相似度

                    if(Array[i][k]!=0&&Array[j][k]!=0&&i!=j){
                        rxy+=(Array[i][k]-rsx)*(Array[j][k]-rsy);


                        rx+=(Array[i][k]-rsx)*(Array[i][k]-rsx);


                        ry+=(Array[j][k]-rsy)*(Array[j][k]-rsy);
                    }



                }
                Similarity[i][j]= rxy/(Math.sqrt(rx)*Math.sqrt(ry));
            }
        }

            return Similarity;
        }
    public  int[] Neiborhood() {  //计算目标用户的邻居
        Neiborhood=new int[11];
        int j=1;
        for(int i=1;i<UserNum+1;i++)  //相似度大于0.5为邻居而且邻居数小于10
            {
            if(Similarity[c][i]>0.5&&neiborhoodnum<=10){
                Neiborhood[j]=i;
                j++;
                neiborhoodnum++;
            }
        }

        return Neiborhood;
    }
```

```
public  Iterator<Map.Entry<Integer, Integer>> Recommender(){     //预测目标用户推荐项目
        double rc=0;
        double s=0;
        Map<Integer,Integer> Recommendations=new HashMap<Integer, Integer>();
        Map<Integer,Integer> Recommendationss=new HashMap<Integer, Integer>();
        double rs,rcs;
        double fr=0,fc=0;

        for(int j=1;j<Count+1;j++){
            if(Array[c][j]!=0){
                rc+=Array[c][j];
                s++;
            }

        }
        rs=(1/s)*rc;

        for(int i=1;i<neiborhoodnum+1;i++)   //通过预测函数计算为目标用户推荐电影的喜好值
            for(int j=1;j<Count+1;j++){
            if(Array[Neiborhood[i]][j]!=0&&Array[c][j]==0){
                for(int k=1;k<neiborhoodnum+1;k++){

    fr+=Similarity[c][Neiborhood[k]]*(Array[Neiborhood[k]][j]-rs);
                fc+=Similarity[c][Neiborhood[k]];
                }
                rcs=rs+fr/fc;
                Recommendations.put(j,(int) rcs);

            }

        }
        List<Map.Entry<Integer, Integer>> list = new ArrayList<Map.Entry<Integer, Integer>>();
        list.addAll(Recommendations.entrySet());
         Collections.sort(list, new Comparator<Map.Entry<Integer, Integer>>() {  //对推荐的电影进行评分降序排序
                public int compare(Map.Entry<Integer, Integer> o1,
                    Map.Entry<Integer, Integer> o2) {
                   //降序

                  return (o2.getValue()-o1.getValue());
```

```
                    //升序

                //   return o2.getValue()-o1.getValue();

                }
            });
        Iterator<Entry<Integer, Integer>> it=list.iterator();
        int i=0;
        while(it.hasNext()){
            Map.Entry<Integer, Integer> map=it.next();
            int movieid=map.getKey();
            int pre=map.getValue();
            Recommendationss.put(movieid, pre);
            i++;
            if(i==tjs)break;
        }
        Set<Map.Entry<Integer, Integer>> TopMovie=
Recommendationss.entrySet();
        return TopMovie.iterator();  //返回目标用户推荐的电影及其评分
    }
}
```