



江 蘇 大 學

JIANGSU UNIVERSITY

本 科 毕 业 论 文

基于用户的协同过滤推荐的实现

Implementation of Collaborative Filtering Recommendation Based on
User

学院名称： _____ 江苏大学

专业班级： _____ 计算机科学与技术 1201

学生姓名： _____ 何佳宇

指导教师姓名： _____ 邢玉萍

指导教师职称： _____ 讲师

2016 年 5 月

基于用户的协同过滤推荐的实现

专业班级：计算机科学与技术 1201 学生姓名：何佳宇

指导教师：邢玉萍

职称：讲师

摘要 在这个信息爆炸的年代,用户想要搜索他们想去了解的信息的同时总会被推送好多无用的信息。这大大的干扰着用户去获取他们的实际所需。所以推荐系统应运而生,现如今好多电子商务的系统都自带对每一个用户个性化推荐的功能。这样用户在无从下手的时候可以根据系统推荐而获得一些启发甚至系统推荐直接就是他们所需求的。本文针对现在个性化推荐中使用最多的算法之一——协同过滤算法做一定的研究,并且在传统过滤算法的基础上做一定的改进,旨在提高当前推荐算法对用户推荐的准确度和尽可能的去缓解现在大数据环境下推荐的种种问题。本文的主要研究内容包含了以下几个方面:

(1) 本文在传统的过滤算法的基础上加上了对用户特征的考虑,将用户和电影的数据导入红黑树的数据结构中,使用红黑树的高效查找的特性缩短查询时间。在获取指定用户的相似用户的过程中通过用户的特有 ID 在红黑树中查找到该用户的特征的相关信息再将用户的特征相似度作为权值,加在传统计算相似用户的算法上,使得算法在获取相似用户的过程中只在特征相似的用户中去查找推荐,符合个性化推荐的思想。

(2) 我们通过将实验过程中采用的算法思想所获得结果和传统的算法所获得结果进行比对,通过他们在选取相同相似用户个数的条件下获得的 MAE (平均绝对误差) 进行比对。

(3) 由于在获取最相似用户的时候将用户特征的相似度作为权重加在原先的过滤算法中,用户特征作为一个影响因素我们需要对这个权重所占的比重获取最佳的取值。只有合理的选取权重大小才能获取最好的推荐效果而不能随随便便就去定义一个权重。所以我们需要在实验中不断地尝试去获取最佳取值。

(4) 我们实验所用的数据源是 movielens 中的数据。我们在这个数据集上对我们的算法进行实验并验证。结果表明我们在原先算法上的改进比原先算法的推荐效果要更好。同时也可以对以后的改进方向上提供一定的思路。

关键词: 用户特征 协同过滤 最邻近用户 推荐

Implementation of Collaborative Filtering Recommendation Based on User

Abstract In the age of data explosion, people will receive much useless information when they search the information they really want to know. This situation will interfere with the users to obtain their actual needs. In this context, the recommendation system was born. Now a lot of electronic commerce system comes with a personalized recommendation for each user. This function can help users get some inspiration and even recommend things that users really need. This article aimed to do some research on the one of the most widely used in personalized recommendation algorithm——collaborative filtering algorithm, and put forward some improvements on the basis of the original filtering algorithm to improve the accuracy of the current recommendation algorithm and to ease the problems under the large data environment. The main work of this paper is as follows:

(1) In this paper, we add user feature factor into traditional collaborative filtering recommendation algorithm. Adding user data and movie data into the red black tree to shorten the query time. We use the user ID to find the information we need in red black tree and calculate the similarity of user property. Take this similarity as a weight add into traditional collaborative filtering recommendation algorithm. With this similarity we can just recommend items people may like in the range of similar users. Meet the ideas of personalized recommendation.

(2) During the experimental process we compare the result of our algorithm to the traditional one. Mainly by compare the MAE (mean absolute error) they get in the selection of the same number of users.

(3) In obtaining the most similar users, we need to add the similarity of user's properties as the weight into the algorithm we use in this paper. We have to obtain the most suitable value of the weight to make sure that we can get the best result. Not just to send it a value. So we need to constantly try to get the best value in the experiment.

(4) The data source used in our experiment is from the Movielens. We test and validate our algorithm on this dataset. The results show that our algorithm is better than the original algorithm in improving the effectiveness of the algorithm. At the same time, it also can provide some ideas for the future improvement direction.

Key words: User Property Collaborative Filtering Neighbour Recommendation

目录

第一章 绪论.....	1
1.1 课题研究背景及意义.....	1
1.2 推荐系统研究现状.....	2
1.3 课题研究内容.....	2
1.4 可行性分析.....	3
1.5 论文框架.....	3
第二章 协同过滤推荐算法综述.....	5
2.1 协同过滤推荐算法简介.....	5
2.2 协同过滤推荐的基本思想.....	6
2.3 推荐系统、推荐算法、实验环境介绍.....	6
2.3.1 推荐系统介绍.....	6
2.3.2 主要推荐算法介绍.....	7
2.3.3 实验环境.....	8
第三章 系统框架搭建.....	10
3.1 数据集的准备.....	10
3.2 算法搭建流程.....	10
3.2.1 相关数据结构的建立.....	10
3.2.2 获取指定用户的相似用户.....	11
3.2.3 偏好的预判和推荐.....	13
3.2.4 算法小结.....	14
3.3 本章小结.....	15
第四章 算法推荐质量实验分析.....	16
4.1 实验数据集和算法评价方法.....	16
4.2 实验方案及结果分析.....	17
4.3 本章小结.....	21
第五章 总结与展望.....	22
5.1 总结.....	22
5.2 展望.....	23
致 谢.....	24
参考文献.....	25
附 录.....	27

第一章绪论

1.1 课题研究背景及意义

在互联网飞速发展的今天。我们每天接收到的信息和以往完全无法比较。互联网在满足了人们了解世界的欲望的同时，也导致了一个不好的现象。就是无论你身处这个世界的哪一个角落，你都能不断地收到这个世界给你发来的各种各样的信息，无论对你有用与否，这些信息都会源源不断的推送给你。所以现在人们很难在这海量的信息当中筛选出真正对他们有用的。导致那些信息推送无法达到应有的效果。

在这种情况下，搜索系统开始走上了历史舞台。用户可以主动去搜寻自己想要的信息而不是只能去被动接受。但是搜索引擎在帮助用户快速获取所需信息同时也反馈给了用户一大堆没用的数据。而且每一个用户去搜索同一个事物会返回相同的结果，无法提供个性化的推荐是现在搜索引擎共有的短板。所以在不断研究之下。个性化的推荐系统也如雨后春笋般不断出现并获得了迅猛的发展^[1]。

个性化的推荐系统可以将用户在网络上留下的各种信息提取出来做处理分析，而后根据每一个用户不同的特性对他们做不同的推荐。这个特点就让这个系统和搜索引擎就有了本质的不同。由推荐系统帮助用户过滤他们不怎么感兴趣的内容，而后呈现出他们感兴趣程度最高的信息。这个举措对现在大数据环境下的用户而言无疑是一个福音。这样的举措不仅能使用户获得良好的用户体验，并且由于这个系统和用户的操作密切相关，效果越好用户使用的就越频繁，用户使用的越频繁系统就可以对用户兴趣做出更加准确的预测，这是一个双赢的结果。

推荐系统中个性化推荐已经在很多的领域得到了发展。最典型的就是电子商务中的使用。一个用户的消费记录和浏览记录被系统记录到数据库中，然后系统会根据这些信息对用户进行推荐。这个思想在电影，音乐上都有不同的体现。例如各种音乐系统中的猜你喜欢就是个性化推荐的真实表现。由于个性化推荐的不断兴起，学术界对这个体系的研究也日益深入。作为推荐系统核心的推荐算法必须具有很强的可扩展性，一个好的推荐算法就是一个推荐系统的活招牌。所以需要不断地去思考，去研究，不断地去提出更有效，更准确的推荐算法^[2]。

1.2 推荐系统研究现状

推荐系统采用的协同过滤推荐现在主要分为两个方向——基于用户的协同过滤推荐和基于物的协同过滤推荐。基于用户的协同过滤是在采用当前用户和其他用户的历史数据建立模型之后预测当前用户对其他物品的感兴趣程度而后进行的推荐。基于物的推荐则是其他用户对这个物品的评价和其他已知的评价来预测当前用户对这个物品的评分来进行的推荐。

虽然现在的推荐算法在表面看来发展的很不错，但是深入研究之后还是存在着很多的问题没有很好的解决方案。协同过滤算法是建立在用户历史评分的基础上的，协同过滤在用户拥有历史数据时可以显示出很好的推荐效果，但是我们无法避免一种情况，那就是随时随地都会有着新用户的不断加入，针对这些群体的推荐就没有一个很成熟的算法去处理。而且在如今这么大数据量的情况下，即使用户做了很多评价，但是放在整个系统的数据库中一比就会发现这个用户所拥有的数据十分的稀疏，一个用户接触到的物品在物品总数中所占的比重小的可以忽略不计。当然不成熟不代表我们没有去研究。比如在面临新用户时基于内容的推荐可以通过用户的邻居用户做推荐。而针对数据的稀疏问题有人也提出了像 BP 神经网络、稀疏矩阵、十字链表等方法来缓解这种问题^[3]。

在研究协同过滤推荐本身的缺陷之外很多人也在算法的准确性上做了很多的努力。我们知道人类社会是一个不断发展的过程，算法同样，一个算法需要时间去不断地深入研究才能获得更好的结果。本文使用用户特征对新用户的推荐问题作了一定的处理，也对传统的推荐算法做了一定的改进并使用数据集来检测算法的可用性。

1.3 课题研究内容

本文针对基于用户的协同过滤算法进行进一步的探讨。在传统算法的基础上对这个算法加上了更多因素的考虑，达到提高推荐算法的推荐效果和对以后提高推荐算法的研究方向上提供一点意见的目的。本文主要做了如下事项：

(1) 本文在传统的过滤算法的基础上加上了对用户特征的考虑，将用户和电影的数据导入红黑树这个数据结构中，使用红黑树的高效查找的特性缩短特定项的查询时间。在获取指定用户的相似用户的过程中通过用户特有的 ID 在红黑树中查找到该用户的特征的相关信息再将用户的特征相似度作为权值，加在传统计算相似用户的算法上，使得算法在获取相似用户的过程中只在特征相似的用户中去查找推荐，符合个性化推荐的思想^[2]。

(2) 我们通过将实验过程中采用的算法思想所获得实验结果和传统的算法所获得结果

进行比对，主要通过他们在选取相同相似用户个数的条件下获得的 MAE（平均绝对误差）进行比对，从比对的结果中发现我们采用的改进算法可以产生更好的预测评分，这也表示在推荐的时候我们的方法可以比传统的推荐算法有更好的推荐效果。

（3）由于在获取最相似用户的时候将用户特征的相似度作为权重加在原先的过滤算法中，用户特征作为一个影响因素我们需要对这个权重所占的比重获取最佳的取值。只有合理的选取权重大小才能获取最好的推荐效果而不能随随便便就去定义一个权重。所以我们需要在实验中不断地尝试去获取最佳取值^[4]。

（4）我们实验所用的数据源是 Movielens 中的数据。我们在这个数据集上对我们的算法进行实验并验证。结果表明我们在原先算法上的改进比原先算法的推荐效果要更好。同时也可以对以后的改进方向上提供一定的思路。

1.4 可行性分析

经济可行性：本文需要用到的协同过滤推荐所需的数据集可以在网上免费获得，无需去网上购买或者线下支付，这可以省去一大笔的开销。我们使用的开发软件是 VS2013，我们只需要去购买一个正版软件，只需付一个版权费。其他的没有多大的限制。所以在经济上是完全可行的。

技术可行性：本文只是针对一个算法的研究，没有其他系统那么花哨的界面要求。我们做的是在底层上对数据的相关处理。我们需要掌握的只是一门可以从文件中获取数据并且可以进行了逻辑分析的语言。

环境可行性：本次毕业设计完全是在计算机上开发并检测，所以对环境无任何影响。

法律可行性：协同过滤推荐算法在电子商务上的运用十分广泛。我们在购买商品的同时涉及到钱财的流通，我们需要防范有人以这种方式实施诈骗或者有人通过这些商务网站购买非法物品。我们需要使用协同过滤对相关的信息进行获取和分析来降低这一类事件的发生。

1.5 论文框架

本文共包含了五个章节的内容，章节内容如下所示，论文的结构如图 1.1 所示。

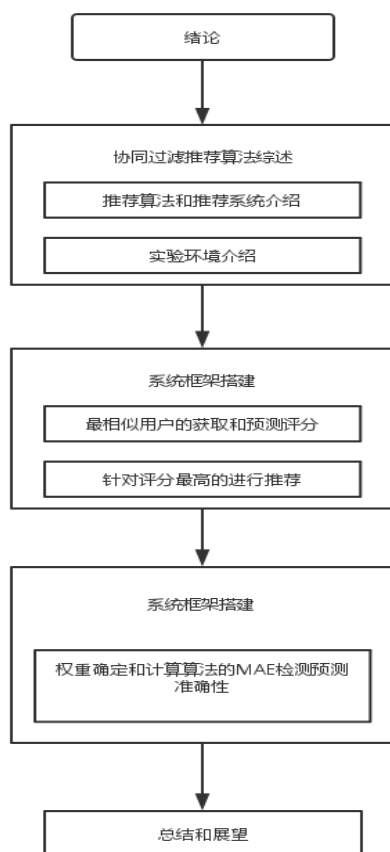


图 1.1 论文结构

第一章：绪论。论述本课题研究的时代背景和现实意义。而后介绍了推荐系统的定义和我们为什么要开展本文的课题，最后对论文的内容和框架做了一个概述。

第二章：协同过滤推荐算法综述。对协同过滤算法的简介和主要算法的介绍。对我们实现本文系统的实验环境的介绍。

第三章：系统框架搭建。算法搭建的整个流程。从分析数据集的数据结构到建立程序可以识别的数据结构而后对实验中不断改良的算法做介绍直至得出最后准确度最高的算法。

第四章：算法推荐质量效果分析。针对第三章中协同过滤算法产生的结果做分析处理，判断是否如预期一般可以在传统协同过滤算法上更进一步。

第五章：总结及展望。论证结束之后对整个过程的总结。提出一些存在的问题并对今后研究方向的一些猜测。

第二章 协同过滤推荐算法综述

2.1 协同过滤推荐算法简介

协同过滤早年是为了给邮件筛选和分类使用的。而且最初的协同过滤不是建立在如今这种数据集的基础上的,而是采用人工输入的方法不断的去训练一个系统来达到过滤无用信息的目的。最初的使用是在 Xerox 公司 PaloAlto 的研究中心。这个研究中心的员工每日都需要处理大量的电子邮件,但是当时没有很好的机器操作的方法来减少员工的工作量,所以为了提高工作效率,这个研究中心便着手开始开发世界上第一个协同过滤的系统。这个系统采用的思想也把协同过滤这个理念明确的表现了出来。这个系统主要流程分为以下几个步骤:

(1) 用户可以根据自身选取自己喜欢的邮件的类型。

(2) 随后他们每个人向系统发出一个需求信息,系统在接收到这个信息后,经过相关的处理返回给用户很多关于用户兴趣爱好的邮件。

(3) 然后每个人从系统返回的邮件中挑选至少三个邮件作为以后系统过滤邮件的依据。然后用户再接收邮件时系统会根据以往的偏好将过滤之后的邮件发到每一个用户的邮箱。

由于这是协同过滤的第一次尝试,难免有着很大的局限性。用户需要明确指出个人喜好而不是由系统去判断,而且无法广泛被采用,只能服务于这个邮件筛选。

但是一个有用的东西如果没有人开个先河将他实现的话或许需要很长时间才能发展起来,但是一旦有了原型之后,我们就可以对其原理进行研究,而后将其使用在其他可以用到的领域。事实也是如此。在这个原型出现之后,相继出现了新闻、电影、音乐的自动协同过滤推荐系统,这些系统使用用户在系统上留下的信息进行模型的搭建,给用户整个系统上用户可能感兴趣的相关项目的推荐。

现在的推荐系统都有各自独特的推荐算法。虽然在内容可能有很大的相似,但是每一个都有其不同的侧重点。比如新闻的推荐会采用保留用户所有时间段的兴趣的方法来对用户进行推荐,而不会将用户的对某一个话题的感兴趣程度在时间维度上做一个衰减处理。而音乐推荐的话可能就是获取实时的喜好,针对用户现在听的推荐用户当下最火的几首类似的歌曲。

不过现实生活中协同过滤算法发展的最好的领域还是在电子商务领域。用户在一个购物网站购买了物品之后,就可以获得相对于初进去时相对精确的推荐,在添加了好友之后,系统还会根据好友之间的相似度对用户做出相关的推荐。而且在购买一个物品之后我们还会看到其他的热门物品和买了这个物品的用户还喜欢的商品。这种情况就时刻表明生活中随处都

存在着协同过滤的影子。

2.2 协同过滤推荐的基本思想

基于用户的协同过滤算法主要是基于用户的历史评分和最相似用户来进行用户对未知项目的评分预测。感觉就像是对人进行分类，我们将每一个人归类到一个团体，而后给你推荐在这个团体中你没有接触过的但是别人接触过的并且有很好评价的物品。体现了一类人一类爱好的思想。由于需要用户的评价来判断，所以必须需要用户对项目进行显式的评分。再让系统根据这些被记录下来的评分去分析一个用户的喜好。再根据这个被分析出来的喜好找出这个用户的最相似用户集。根据最相似用户的评分去预测目标用户的评分。而后选取其中的 Top-N 项对目标用户进行推荐^[5]。

	Item1	Item2	Item3	Item4	Item5	Item6
User1	3	4	2	4	5	1
User2	5	4	3	5	3	5
User3	1	3	4	2	3	4
User4	4	4	5	3	3	2

图 2.1 评分表

上图 2.1 表示的是用户与物品的评分对应表，这些都是用户被记录的历史评分数据，使用数字 1-5 表示用户从讨厌到喜欢的程度，这些数据被保留在系统中。而后系统通过每一个用户的历史评价采用协同过滤算法获取每一个用户和其他用户之间的相似度，而后在最相似用户的群体中选取目标用户预测评分最高的进行推荐。协同过滤可以直接从用户的历史数据中获取出所需的信息，从而分析出用户的喜好，不在依赖用户和物品本身。而且随着用户可用历史信息的不增多，我们的算法可以获得更多的数据进行分析从而获得更好的推荐效果。推荐的个性化效果也会不断地显示出来。

2.3 推荐系统、推荐算法、实验环境介绍

2.3.1 推荐系统介绍

推荐系统有着这样的定义：以电子商务网站为媒介，向每一个用户提供店铺中的商品详细信息并模拟导购员对用户进行店铺内商品的推荐并帮助用户完成整个购买过程的系统^[1]。

推荐系统主要分为三大块：用户模块、推荐项目模块、推荐算法模块，详细关系如图 2.2 所示。在这个系统中系统将用户模块中的相关信息进行提取并将其与实际需求进行比对。通

过对应的计算公式进行信息的过滤，提取出用户最有可能感兴趣的项目并对其做最终推荐。

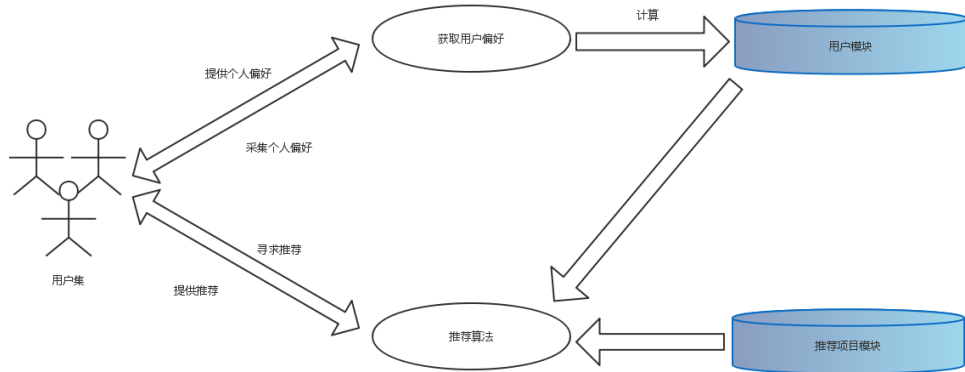


图 2.2 推荐系统模型

随着现代数学的迅猛发展，我们不再是只在理论的层面上对一个事物进行构想，而是可以建立合理的理论体系来支撑起整个系统框架。我们已经将这个模型所需的一切因素以数学公式的形式表现了出来。正因为这个理论体系的成功搭建才使得我们的推荐系统在诞生之后发展的如此迅速。

这个模型理论的依据如下：我们将 A 作为用户群，将 B 作为可推荐的项目群，我们需要做的就是对任意一个属于 B 中的单体 b 使用算法中的公式 $P()$ 做属于用户群 A 中任意单体 a 的预测评分，来获得每一个用户的预测评分集 R。R 中存放 B 中每一项对应的预测评分。如式(2-1)所示

$$a \in A, b \in B, R(b) = P(a, b) \quad (2-1)$$

最终我们需要做的就是从 R 获取其中的 Top-N 项推荐给用户。

以上的说明可以明确的指出推荐系统就是由用户模块，推荐项目模块和推荐算法这三大块组成。本文的研究也是以这三大块为基础建立的。

2.3.2 主要推荐算法介绍

一款推荐系统的好坏完全取决于推荐算法，推荐算法就是整个推荐系统的核心。基于这个原因推荐算法吸引了来自世界各地的关注，往往一个好的推荐算法的诞生必将会给推荐领域带来一场全新的技术革新。就当下的推荐算法而言，发展的比较好的有以下几款^[6]：

基于内容的推荐算法：这个算法起源与信息检索领域。核心思想是从整个可推荐的物品集中查找和用户以往评分较高的物品相类似的推荐给用户。这个算法主要研究用户的历史数据，通过用户历史数据分析出用户的偏好，在可以推荐的领域中查找预测评分最高的推荐给

用户。这个算法被大量使用在像音乐推荐这样的系统中。比如 KuGou 中就有猜你喜欢这个功能。用户将他们喜欢的歌曲放入收藏，而后推荐系统对这些收藏的数据进行对应的操作，而后在整个曲库中查找用户可能会喜欢的歌曲反馈给用户。

协同过滤推荐算法：就目前来说这个推荐算法可以说是最成功的推荐算法之一。在现实生活中这个算法被广泛的运用在不同的推荐领域。如今的电子商务上表现的尤为突出。这个算法建立的前提是需要一个数据集。我们不再是简单的挖掘单个用户的历史行为，而是将这个区间内的所有用户数据全部作为分析的依据，在这些用户中找到与指定用户最相似的用户，将最相似用户喜欢的但是目标用户没有听说过的项目做预测评分，然后针对预测评分最高的几项做推荐，这样用户可以通过其他用户看到更多有意思的事物。

基于关联规则的推荐算法：这个算法使用物品之间的关联作为判断的基准。将用户买过的商品作为规则的线索，推荐的物品集作为整个规则体系。通过用户购买商品留下的历史记录找出用户购买商品之间的关联。这个在在日常生活中随处可见。最实际的例子就是商场中物品的摆放，在各种干货的旁边同样有着各种饮料供用户选择。很早以前在孩子尿布旁边放置啤酒是一样的，这是一种消费策略，同样也是关联规则的实际体现。

基于知识的推荐算法：此算法有别于以往的推荐算法，它在整个网络环境中去获取它要用到的用户数据和项目数据。通过语义的匹配和这方面相关的推理算法获得我们所需的推荐结果，而且它的推荐策略也不局限于一种，他会根据实际情况做出相应的调整来给每一种情况尽量好的解决方案。

基于网络结构的推荐算法：这是一种全新的推荐思维，这个方法没有局限在必须获取到用户详细信息和必须结合推荐项目的详细信息做推荐，而是直接把每一个用户和每一个物品抽象为节点。一个用户在选择一个项目之后，我们就将这两个事物之间建立联系。通过这种关联体系形成的复杂的关系网。通过研究这个关系网来查找出我们需要的对用户进行推荐。

现实生活中这些的算法都存在着优劣，要是单个运用基本上每一种只能适用于某一个领域而无法作为通用的方法被广泛的采用而且采用单一方案的产生的结果也不怎么尽如人意。所以我们一般会将几种算法结合，取长补短，来获取最好的推荐效果。

2.3.3 实验环境

本文实验是在 CPU 为 AMD A8-5550M APU with Radeon(tm) HD Graphics 2.10GHz, 内存为 4GB, 系统为 Windows 8.1 64 位系统的实验机上进行的。软件采用 Visual Studio 2013 并使用底层语言 C++ 开发。

Visual Studio 2013 是 Microsoft 于 2013 年在全球开发者大会推出的一款强大的开发工具，这款软件除了拥有一般开发工具所拥有的功能之外。更加有效的是可以部分的支持于 2011 年推出的 C++11 的新特性。在使用 C++ 开发的过程中我们可以使用这些新特性来很大程度减少工作量。软件内置的 STL（标准模板库）可以为用户提供强大的容器，用户可以直接来调用而不用再次花费时间去重新建立一个同样的结构。在此基础上的开发将会变得更加方便。现在的 Visual Studio 2013 在很多的开发领域被广泛使用。搭建软件的底层框架，或者建立服务器的底层框架，在游戏引擎的开发和游戏大作的开发都有其身影。它将很多强大的功能不断的集成，不断地进行改善，为开发者提供了很多的帮助。

第三章 系统框架搭建

3.1 数据集的准备

数据集是整个算法为之构建的基础。所以在进行算法实现之前，我们需要理清楚所需研究的数据集的结构，本文研究的数据集是由 Minnesota 大学收集并免费公布的。这个数据集中包含了五个不同的数据集，分别为 .base 文件的训练集和 .test 文件的训练集。每一个基本上由 90 多个用户对大约 1300 部电影超过 10 万条评分，评分数据处于 1~5 的范围中，代表着一个用户对一部电影的喜好程度（越高则越喜欢），每个用户至少对其中的 20 部电影参与了评分。除了评分数据之外，这个数据集还包含了每一个用户的基础信息（个人 ID，年龄，性别，职业，邮编）和每一部电影的相关信息（电影 ID，电影名，发行时间，网址）。同样的我们为了检测算法的通用性也将包含 6400 个用户对大约 3900 部电影超过 100 万条的评分数据纳入到本文中做检测。

3.2 算法搭建流程

3.2.1 相关数据结构的建立

鉴于在获取最终结果的过程中需要对用户的相关信息不断地进行查找，为了提高查找效率，将用户和电影的信息以各自的 ID 为 Key 值存放到对应的红黑树（一种高度平衡的二叉树，查找效率为 $\log n$ ）中。将用户的评分数据存入一个评分矩阵，通过用户 ID 和电影 ID 查找到每个用户对应于每一部电影的评分。每一个用户对应一个评价过的电影列表。

用户信息存放结构如图 3.1 所示（代码见附录中用户信息读取）：

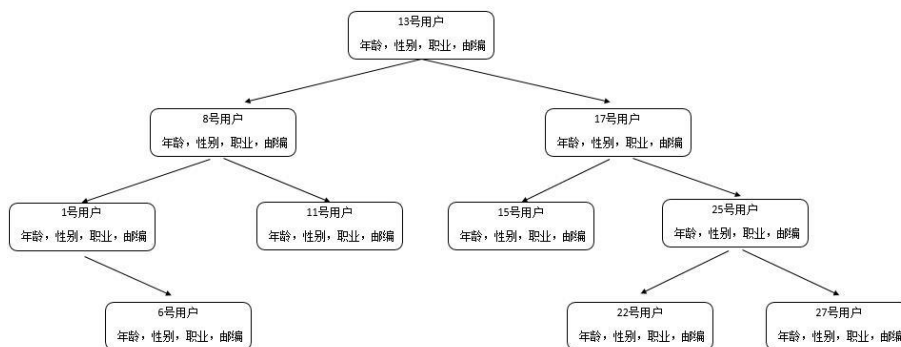


图 3.1 用户信息的红黑树

电影信息存放结构如图 3.2 所示（代码见附录中电影信息读取）：

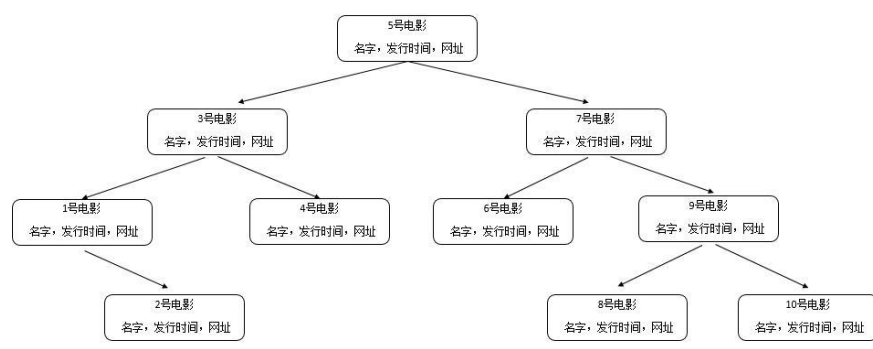


图 3.2 电影信息的红黑树

用户和它对应的电影列表如图 3.3 所示（代码见附录中评分信息读取）：

用户 编号	电影列表									
1	2	23	25	56	89	456	560	589	690	...
2	53	65	89	455	560	582	780	972	1055	...
3	5	12	25	54	58	59	102	420	681	...
4	450	685	788	801	854	865	874	895	1244	...
5	102	235	632	785	965	1246	1258	1297	1299	...
6	6	49	52	235	854	972	988	998	1208	...
7	24	26	59	95	97	164	189	289	784	...

图 3.3 用户和电影列表的关系

评分数据如图 3.4 所示（代码见附录中评分信息读取）：

User/Ite m	I1	I2	...	Ij	...	In
U1	R1,1	R1,2	...	R1,j	...	R1,n
U2	R2,1	R2,2	...	R2,j	...	R2,n
...
Ui	Ri,1	Ri,2	...	Ri,j	...	Ri,n
...
Un	Rn,1	Rn,2	...	Rn,j	...	Rn,n

图 3.4 评分矩阵

3.2.2 获取指定用户的相似用户

每一个用户都有着其独一无二的特征，本文在获取指定用户相似用户的过程中使用改进的余弦相关性计算同时也添加了对用户特征的思考。在采用这种方法之前，我也对照了以前采用的用于相似用户计算的方法。

比如最粗糙的只考虑用户参与个数之间关系的相似度计算的 Jaccard 公式 (3-1):

$$\text{similarity1}(i, j) = \frac{|R_i \cap R_j|}{|R_i \cup R_j|} \quad (3-1)$$

$\text{similarity1}(i, j)$ 代表用户 i 和用户 j 之间的相似性, R_i 和 R_j 代表用户 i 和 j 已参与过的项目集合。两者的交集除以两者的并集这样获得的用户相似度十分粗糙。在忽略了很多关键要素的同时也无法很好的为用户推荐所需的信息。

所以为了更精确的获得所需的信息,有人提出了使用余弦相关的相似度计算公式(3-2):

$$\text{similarity2}(i, j) = \frac{\sum_{d \in I_{i,j}} (R_{i,d} - \bar{R}_i)(R_{j,d} - \bar{R}_j)}{\sqrt{\sum_{d \in I_{i,j}} (R_{i,d} - \bar{R}_i)^2} \sqrt{\sum_{d \in I_{i,j}} (R_{j,d} - \bar{R}_j)^2}} \quad (3-2)$$

$I_{i,j}$ 表示用户 i 和 j 共同参与过评分的项目的集合。 $R_{i,d}$ 和 $R_{j,d}$ 分别代表用户 i 和 j 对项目 d 的评分, \bar{R}_i 和 \bar{R}_j 代表用户 i 和 j 评价过的项目的平均评分。

这样计算出的用户相似度相对与公式 (1) 而言更符合实际需求,同时对后续的推荐算法的发展起到了很好的推动作用。在这样的基础上我们可以为相似用户的获取加入更多的因素来使得结果更符合实际所需。

比如我们可以针对公式 (2) 简单的扩充一下用户计算均方差的区间就可以进一步的提高用户之间相似度的准确性。可以得到经过改良的余弦相关计算公式 (3-3) 如下:

$$\text{similarity3}(i, j) = \frac{\sum_{d \in I_{i,j}} (R_{i,d} - \bar{R}_i)(R_{j,d} - \bar{R}_j)}{\sqrt{\sum_{d \in I_i} (R_{i,d} - \bar{R}_i)^2} \sqrt{\sum_{d \in I_j} (R_{j,d} - \bar{R}_j)^2}} \quad (3-3)$$

I_i 和 I_j 为用户 i 和用户 j 参与过评价的集合。

这里我们将获取的单个用户的均方差的范围扩大,对每个用户而言,他的均方差会把所有之前的评分全部考虑进去,对用户的相似度的准确性有更好的说服力。

本文在公式 (3) 采用的相似度的计算的基础上,将用户的特征也作为一个因素加以考虑,使得相似用户之间的相关性更大,最终的结果也更加的准确。我们基于在同一个年龄段的用户或者是身处同一个社会地位的用户可能会有更加相似的兴趣爱好这样的一个假设上得到了相似度计算的公式 (3-5)。

在计算的同时我们也需要获取用户的特征相似度将其作为权值加到公式 (3-5) 中。获取用户特征相似度公式 (3-4) 如下:

$$\text{similarity4}(i, j) = \frac{|P_i \cap P_j|}{N} \quad (3-4)$$

P_i 和 P_j 代表用户 i 和 j 自身的特征, N 代表特征的总数。

上面公式计算的是两个用户相同的特征个数在总特征中所占的比重。使用公式(3-4)获得用户特征相似度对公式(3-3)进行加权以获得最终加上用户特征的用户之间的相似度的计算公式如下(代码见附录中用户相似度计算)^[2]:

$$\text{similarity5}(i, j) = P \times \text{similarity4} + (1 - P) \times \text{similarity3} \quad (3-5)$$

P 代表用户特征相似度在整个相似度计算中所占的比重, 经测试 $P=0.08$ 的时候, 可以获得最好的结果。

以上的过程全是建立在用户已经存在的情况下对用户的推荐。现实生活中我们同样会遇到这种情况, 我们的系统时刻都会出现一个新用户, 这个新用户没有任何的历史数据, 我们就无法为这个用户使用上面的方法进行推荐。这个就是很常见的“冷启动”问题。本文针对这个也做了相关的处理。

我们提取出用户在注册时填入的用户信息, 将其作为判断的依据, 而后根据公式(3-4)选取用户的相似用户, 取出相似用户中评分较高的几项作为推荐反馈给指定用户。若是简单的针对用户推荐所有用户中平均评分最高的项目, 会导致失去个性化推荐的效果, 这会导致所有用户的最终的推荐结果都是一样的, 那么用户在注册阶段所做的一切全都是无用功。会大大的降低用户体验。

3.2.3 偏好的预判和推荐

在获取到指定用户的相似用户后, 接下来要做的便是对该用户进行评分的预判, 针对相似用户的感兴趣列表中指定用户未曾涉及的项目, 通过评分预测公式(3-6)获取到该用户对未评分的项目的预测评分。选取其中的 Top-N 项进行推荐处理。

根据最相似用户获取到相似用户评价过, 但是指定用户没有涉及的项目集合 T 。针对这个集合 T 中的每一项 d 做指定用户 i 对其的评分预测。公式如下:

$$\text{predict1}(i, d) = \sum_{d \in T, j \in R} (\text{similarity5}(i, j) \times P(j, d)) \quad (3-6)$$

$P(j, d)$ 表示的是用户 j 对项目 d 的实际评分, R 表示指定用户的相似用户集合。

通过利用相似用户的评分来预测指定用户对某一项的评分, 而后选取其中的 Top-N 项进行推荐。这个算法表面看起来的确实现了这样的功能, 但是仔细想想, 我们会发现一个很大

的问题。如果有一部电影很多人看过，但是对这部电影的普遍评价都不高，但是在经过上面运算之后它的值可以达到一个很高的值。同样会在最后处理的时候推荐给用户，这个会导致推荐了一部烂片给用户，就达不到我们实际所想实现的效果。大大降低了这款推荐系统的用户体验。所以我们需要对这种情况的出现进行防范和处理。所以我们不能使用上面的方法去预测用户的评分，而应该去将已有的信息尽量的加以利用。我们可以在获得预测评分的过程中加上对指定用户对以往看过电影的平均评价。而后对其进行加权处理，就可以获得更加精准的预测。公式如下（代码见附录中针对相似用户的项目的评分预测）^[5]：

$$\text{predict2}(i, d) = \bar{P}_i + \frac{\sum_{j \in T} \text{similarity5}(i, j) \times (P_{j,d} - \bar{P}_j)}{\sum_{j \in T} (|\text{similarity5}(i, j)|)} \quad (3-7)$$

\bar{P}_i 和 \bar{P}_j 代表用户 i 和 j 的平均评价。 $P_{j,d}$ 代表邻居用户 j 对项目 d 的评分， T 代表相似用户的集合。在用户 i 的相似用户集 R 中获得用户之间的相似度，将相似度作为权值，加上邻居用户的评价和指定用户对以往项目的平均评价，多方位结合之下获得的预测评分有更强的说服力。

在获得用户对数据集 T 中每一个项目的预测评分之后对其进行排序之后选取其中的 Top- N 项进行最终的推荐。

3.2.4 算法小结

本文中的算法在初始的简单通过用户之间相似的个数获取用户之间的相似度到最后加上了用户特征，评价平均值等因素，在推荐结果的准确性上提供了一层又一层的保障。但是光想想是无法测试我们这个推荐算法的效果到底如何。我们需要一个统一的标准去检测我们的推荐是否可以提供良好的结果。如今的推荐准确度有着很多有用的衡量标准，本文中采用便是平均绝对误差 MAE 来对推荐的结果做测评，这个标准简单而且结果十分精确。

计算平均绝对误差的公式如下（代码见附录中平均绝对误差的获取）^[2, 5]：

$$\text{MAE} = \frac{\sum_{i \in Q} |P_i - R_i|}{N} \quad (3-8)$$

Q 代表用户评价过的物品集合， P_i 代表用户的预测评分，使用公式（3-7）可获得。 R_i 表示用户实际的评价， N 代表着 Q 这个数据集中所拥有元素的个数。一般而言平均绝对误差越小则表示预测的越精确，产生的推荐效果也越理想。

3.3 本章小结

基于用户的协同过滤算法实质上就是基于现有的用户评价集对指定的用户做出准确的喜好预测，然后对其进行在他关注范围之外的项目的推荐。不断扩大这个用户的涉猎范围就可以将这个用户的特征不断地进行挖掘就可以对其进行更加精确的推荐。本文实现的算法是根据指定用户的自身特征选取的相似用户，所以一个用户他的个性越明显，我们就能获得更多的可用信息，在选择相似用户的过程中就可以不断地缩小选择范围来不断地提高我们预测的精确度。由此产生的推荐效果也会越来越理想。对一个可用信息很少的用户进行推荐的话，最终推荐效果不理想的概率也会随之大大增加。

第四章 算法推荐质量实验分析

4.1 实验数据集和算法评价方法

本文中所采用的数据集是常用的 MovieLens，这个数据集包含了用户信息，电影信息和用户和电影之间的评分信息。其中有两种评分数据，分别是包含了 u1 到 u5 文件的测试集和训练集文件和来自 6400 个用户对大约 3900 部电影超过 100 万条的评分记录，在这些文件中每一个用户都至少对 20 部电影做出了评价，评价采用的是计分制，使用数字（1-5）表示用户对这部电影的喜好程度^[7]。

其中用户信息如表 4.1 所示（结构体定义代码见附录）：

表 4-1 用户信息

UserID	用户编号
Age	年龄
Gender	性别
Occupation	职业
Zip-code	邮编

电影信息如表 4.2 所示（结构体定义代码见附录）：

表 4-2 电影信息

MovieID	电影编号
Name	名称
Time	发行时间
Address	网址

用户评分信息如表 4.3 所示（结构体定义代码见附录）：

表 4-3 用户评分信息

UserID	用户编号
MovieID	电影编号
Rating	评价
Timestamp	时间戳

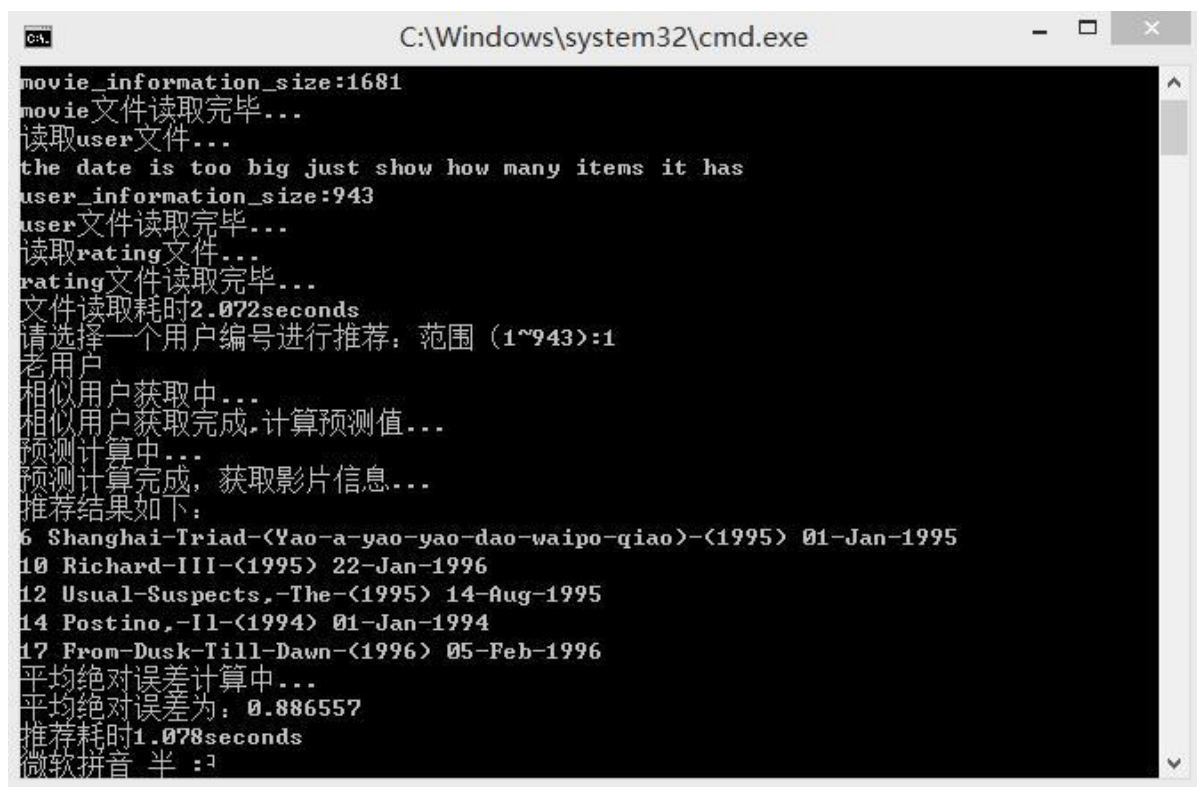
在将这三个信息源中的数据全部导入到相对应的数据结构中后。针对指定用户做预测评分较高的几项的推荐。在推荐的同时我们需要对我们的推荐结果的准确性做出评判，本文中我们使用公式（3-8）计算的平均绝对误差 MAE 作为推荐结果准确度的评判标准。平均绝对误差越小代表着我们的算法在推荐上的准确度越高。

4.2 实验方案及结果分析

本文使用这个算法针对指定的用户实施推荐,推荐结束之后输出这个算法针对这个用户进行推荐的平均绝对误差。文件中的 u1-u5 中的 .base 文件中的评价信息作为训练集, .test 文件中的数据作为测试集。通过训练集中的数据建立我们的系统分析模型。用户的信息存放在 u.user 文件中,这个文件包含了用户的编号,年龄,性别等相关特征。电影信息存放于 u.item 文件中,其中包含了每一部电影的编号,名字,发行时间等信息。训练集和测试集中的数据呈 4:1 的比例。我们选择训练集中的用户对其进行推荐。并计算最终推荐结果的平均绝对误差来对推荐结果做评判。

除了以上的数据之外,我还加入了另外一个数据集,它包含了 6400 个用户对大约 3900 部电影超过 100 万条评分数据,将这个数据集的结果和小数据量的做比对。

实验最终结果显示界面如图 4.4 和 4.5 所示:



```
GA. C:\Windows\system32\cmd.exe
movie_information_size:1681
movie文件读取完毕...
读取user文件...
the date is too big just show how many items it has
user_information_size:943
user文件读取完毕...
读取rating文件...
rating文件读取完毕...
文件读取耗时2.072seconds
请选择一个用户编号进行推荐:范围(1~943):1
老用户
相似用户获取中...
相似用户获取完成,计算预测值...
预测计算中...
预测计算完成,获取影片信息...
推荐结果如下:
6 Shanghai-Triad-(Yao-a-yao-yao-dao-waipo-qiao)-(1995) 01-Jan-1995
10 Richard-III-(1995) 22-Jan-1996
12 Usual-Suspects,-The-(1995) 14-Aug-1995
14 Postino,-Il-(1994) 01-Jan-1994
17 From-Dusk-Till-Dawn-(1996) 05-Feb-1996
平均绝对误差计算中...
平均绝对误差为:0.886557
推荐耗时1.078seconds
微软拼音 半 :1
```

图 4.1 小数据量运行界面

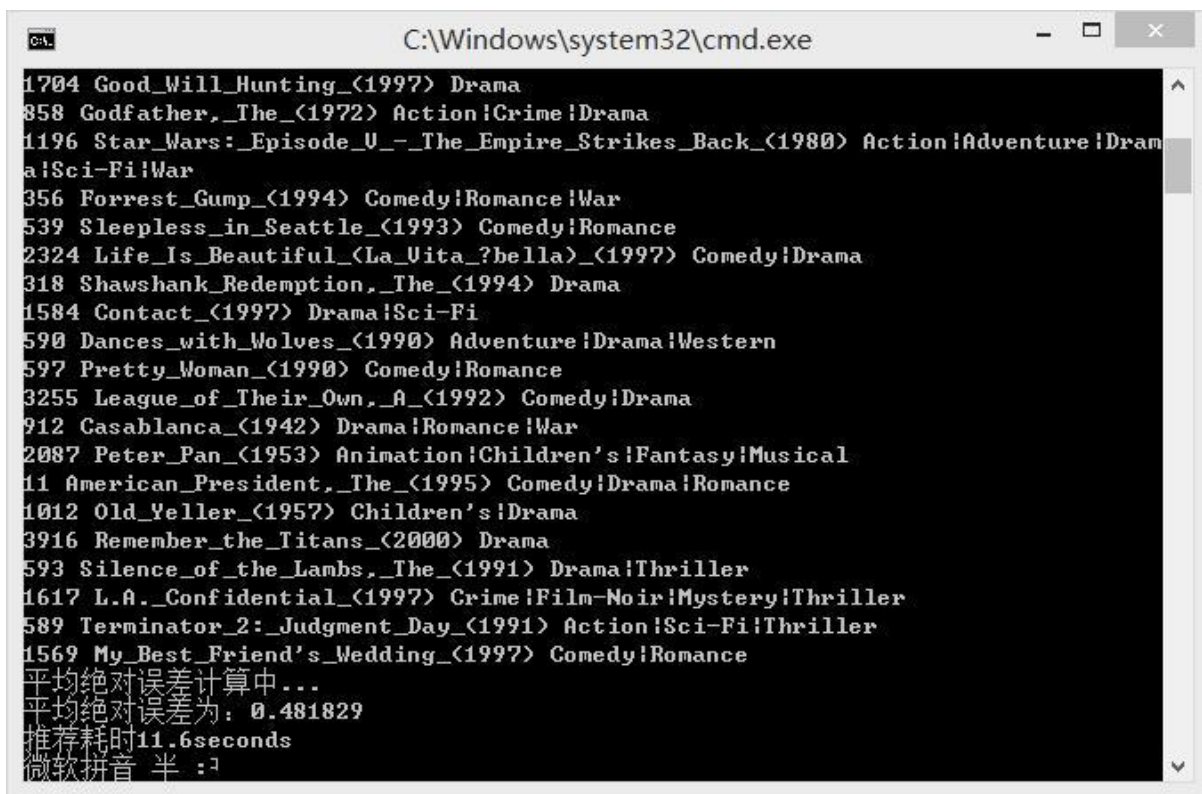


图 4.2 大数据量运行界面

使用本文算法的前提是要确定用户特征所占权重的大小,权重测试数据如下图 4.6 所示:

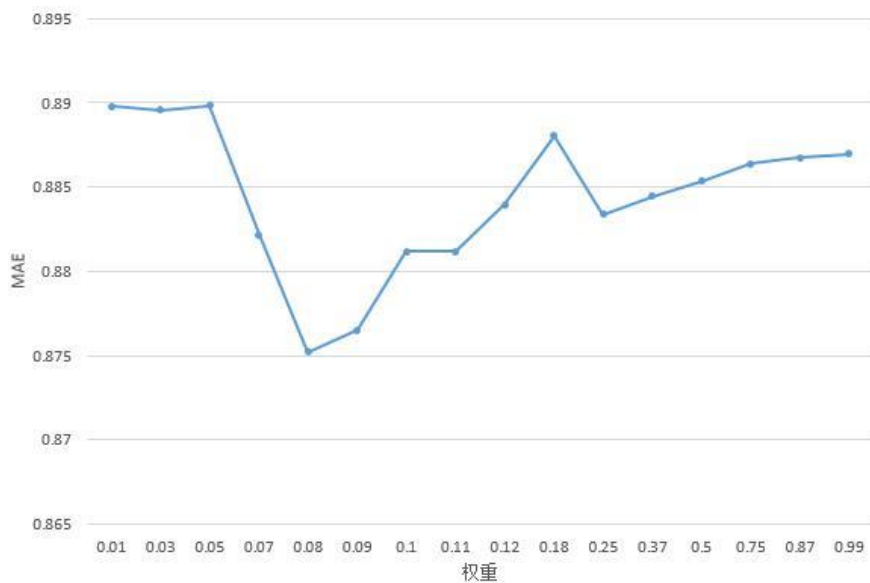


图 4.3 用户特征权重的确定

由上图可以看出我们在将用户特征权值置为 0.08 时效果最好。同时我们使用本文优化的协同过滤算法通过改变相似用户的个数获得的 MAE 与传统的协同过滤算法获得的 MAE 进行比对的理想结果应该如下图 4.7 所示:

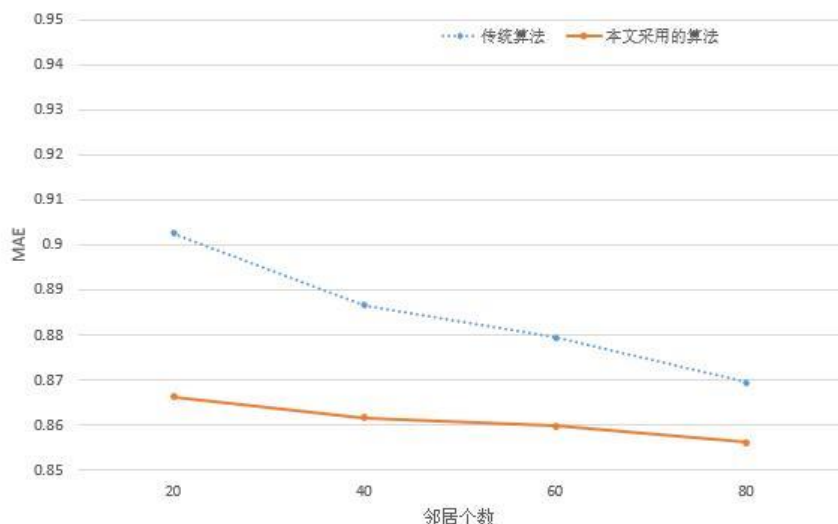


图 4.4 实验结果

但是在实际的实验中我们发现并不是所想就会有所得。不同数据量在相同邻居数下获得的 MAE 呈现不同的走向，真实数据如图 4.8 所示：

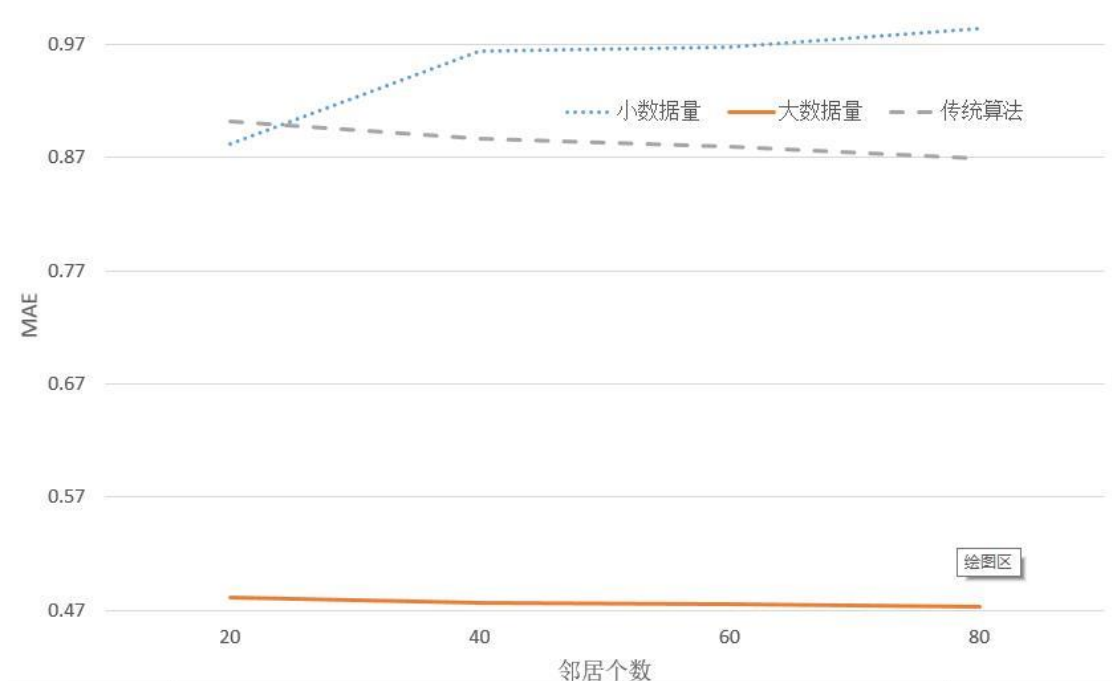


图 4.5 不同数据量相同邻居数的 MAE 比对

由上表中的数据可以看出在选取同等邻居用户的情况下，本文基于用户特征的协同过滤算法在大数据量的情况下要比传统的协同过滤算法的平均绝对误差要小，证明了我们这个思想的可行性和正确性。并且可以看出在大数量的情况下随着邻居用户增多本文采用的将用户特征作为权重加上的算法的平均绝对误差比之传统的要呈现更好的推荐效果，更说明了本文实现的算法比之传统的算法而言有着更加好的推荐效果。也说明我们的过滤算法需要大量的

数据来搭建一个稳固的模型。

但是实验在运行时还是存在着相关的缺陷，比如在选取最相似用户的个数时，我发现 20 是一个分界线，20 以内的数随机性很大，准确性没有很好的规律性，但是在继续往上增加的时候，我们的算法计算出的平均绝对误差就会呈现出我们想要的效果——随着用户数的增加预测结果的准确性也会不断的提高。

上述步骤只是对已有用户进行推荐，现实生活中出现的新用户就无法使用上述方法。针对这个问题，我们同样可以根据用户特征使用公式（3-4）来选取指定用户的相似用户，但是相对于其他的用户而言，这个新用户只有用户的特征可以提供一定程度的参考。我们可以为用户的每一个特征提供一定的权重，这样就可以为新用户选取出最适合的相似用户再根据这些相似用户对新用户做出推荐。程序流程如图 4.9 所示（代码见附录中对未注册和注册的新用户的推荐）：

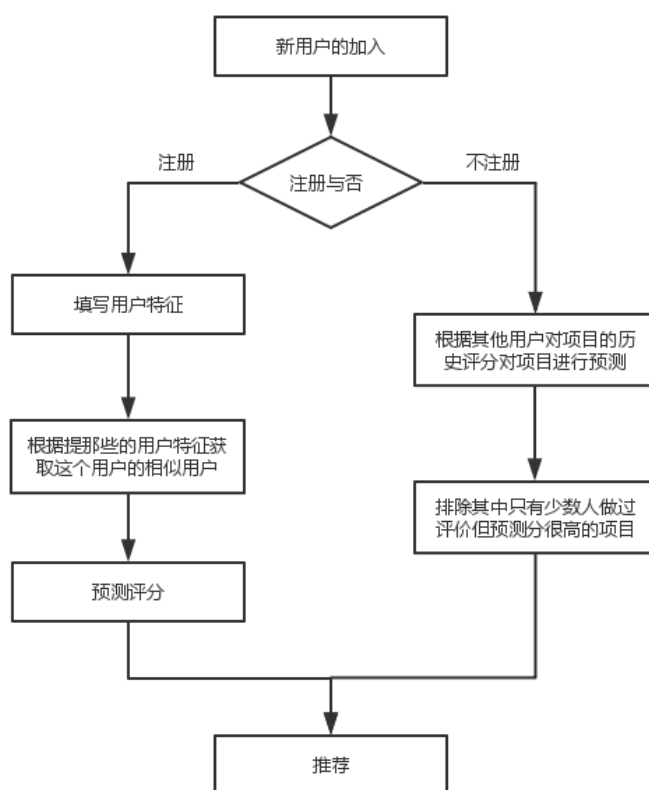


图 4.6 冷启动时的用户推荐流程

我们可以在图 4.9 看出在新用户进入系统的时候程序如何去处理对新用户的推荐，而不会使得系统在面对这种情况下不知道如何去处理。模拟了实际需求，在解决冷启动问题的同时也把系统功能进一步的完善了。

4.3 本章小结

本文采用了一个基于传统协同过滤算法的基础上加上了对用户特征的考虑的算法思想。通过用户的特征选取进一步的缩减相似用户的选取范围。真正的强调了每一个用户的个性化的推荐。切实的提高推荐的准确度。

本章节完成的主要任务如下：

(1) 介绍了传统的协同过滤算法和添加了余弦相似性的相似度计算和本文最终采用的在余弦相似性相关的基础上加上用户特征作为权值的改进算法。

(2) 给出了在加入用户特征的余弦相关性协同过滤推荐算法的流程描述。对评分信息和用户特征的数据建模，用户之间相似度计算，获取指定用户的相似用户，对指定用户进行推荐项目的预测评分到最后的生成推荐。

(3) 给出了本文采用算法和传统协同过滤算法之间的效果比较。

(4) 加上对冷启动的考虑并实现对新用户的推荐。

(5) 通过实验验证了本文推荐算法的有效性。

本章节发现的问题：

在历史评价很少的情况下，我们所获取到的平均绝对误差并不如我们预期的那样随着用户数的增加而减小，因为随着邻居数的增大我们获取到的数据变得越来越稀疏。真正有用的信息占用的比率也会越来越小，导致推荐效果越来越差。所以基于用户的协同过滤算法需要建立在大数据的基础上才可以很好的运作。

第五章 总结与展望

5.1 总结

在如今这个信息爆炸的年代，推荐系统中的推荐算法的好坏直接决定了这个系统是否可以长久存在。协同过滤算法在现如今是个性化推荐算法中发展的最为成熟的算法之一。主要是针对于像如今迅猛发展的电子商务。但是这个算法也存在着很大的局限性。每一个推荐算法基本上只能服务于一种数据集，无法得到一种通用的算法原型。而且现在随着网络信息的越发庞杂，在个性化推荐的过程中我们需要考虑的因素也变得越来越复杂。所以需要不断地去改进思路去应对现如今的变化形式。

本文针对电影数据对指定用户的推荐进行了深入的研究。在传统的协同过滤算法的实现上加入了一些更改的方案来获得更加准确的推荐结果。我们在传统的推荐算法的基础上加入了用户的特征分析，对每一个用户的个性真正的去了解，去分析，去计算出指定用户最相似的邻居群。提高推荐算法的推荐精度。

本文首先介绍了这个课题研究的背景和意义。并介绍了需要用到推荐算法的领域和当下推荐算法的发展形势。可以让我们了解到现在科技环境下的协同过滤发展的重要性。对协同过滤算法的基本思想和基本流程进行了相关的介绍。而后就是本文如何针对 movielens 的数据集进行协同过滤算法在传统算法的基础上的改进实现和对这两种算法获得结果进行比对。

本文采用了基于用户特征的推荐算法。这个算法中我们不再是简单的对获取到的最相似用户的评分项中指定用户没有接触过的做预测，然后取出评分最高的几项作推荐。而是在获取最相似用户的过程中加入了用户特征这个变量的考虑，这个可以使我们获得的最相似用户起码在兴趣范围上不会和指定用户存在很大的误差。加入了用户特征之后我们也可以在对新用户进行推荐时可以有一个很好的评判标准，而不是一味地在忽略用户个性的同时推荐其他用户评价过的好电影。

在系统创建的过程中我们主要说明了这个系统所用到的数据集和算法建立在什么形式的数据结构上。通过算法建立过程中对相似用户提取算法和指定用户评分预测算法的不断改善来不断提高我们推荐的准确率。然后针对本文采用的推荐算法和传统的推荐算法之间进行比对得出我们这个算法是否在原先的基础上向前更进一步。

总的来说本文主要完成以下工作：

- (1) 首先介绍推荐算法的适用领域和在当下发挥的作用。对推荐算法整个流程做了一

个介绍，再回顾以往的推荐算法来引出本文要采用的改进之后的想法。

(2) 提出本文采用的在传统的算法基础上加入用户特征因素的改进算法。在以往只对用户历史评分作判断的基础上加上用户特征。在加上之后发现用户的相似度的计算同样有助于对新用户的推荐。在遇到新用户时也可以对该用户的在注册时所填入的信息对其查找相似用户来对其进行个性化推荐。不会在只有历史评分的情况下无法解决冷启动的问题。

(3) 最后我们对这个实验的最后结果做详细的比较分析。介绍了我们搭建系统的平台和环境配置。对数据集的信息做详细的介绍。并且将实验中所采取的方案和传统的方案进行对比可以知道这个算法改进之后是否真的可以做出更好的推荐。经过实验检测我们发现在加入了用户特征因素之后我们的推荐系统做出的推荐相对于传统的算法而言有着更好的用户体验。

5.2 展望

本文主要采用了基于用户特征的协同过滤的推荐算法。在提高了推荐准确度的同时随着研究的不断深入也有着一些其他的问题需要解决。针对现在推荐算法在各个领域之中的不同表现。可以继续深入的点还是存在很多的。

对于未来的研究方向可以有以下几点：

(1) 在本文中提出的基于用户特征的算法可以加上对用户评价时间的判断，但是由于数据集中存在着同一个用户同一时刻对多部电影进行评价，没有很好的处理这种情况的办法，而且若对时间做评分衰减，这会导致那些喜欢看老片的用户可能无法找到他们真正中意的，所以个性化推荐在这上面还是有着很多的研究需要去完成^[4,10]。

(2) 用户特征其实可以对其进行更多的划分，比如每一个特征占据你整个特性的比重，可能有些系统是以年龄为主要的评判标准，而有的就可能以地域为评判标准，这些我们都需要去考虑，这样才能实现真正的个性化^[8,9]。

(3) 推荐算法除了满足用户个性化需求外，还存在着一些内部的问题，那就是现在我们所提取出的数据是否真的可以真正的代表一个用户个性评判的标准。这一切我们都需要去考虑。没有一个算法是完美的，所以未来我们还需要为这个做更多的研究。

致 谢

首先，我要感谢我的毕业指导教师邢玉萍老师。在整个毕业设计的过程中，她从算法思想，算法的流程再到算法实现之后论文的撰写和修正给我很多的指导和帮助。她带着我去切实的了解一个算法，见证了这个算法的发展和在现实中的地位。是我研究算法的第一步。虽然在最后交了一份答卷，但是不敢说这个算法没有任何的问题。但是却让我对算法的研究产生了很大的兴趣。

其次，我要感谢我的同学们，在和他们讨论的过程中我了解到了很多我在算法视线中忽视的问题。他们在我实现算法时给我提供了很多的意见，让我了解到这个算法的博大精深之外也使我的动手能力得到了很大提高。

最后，我还要感谢我的家人，没有他们在我灰心丧气时候的不断鼓励，我可能无法交上这一份满意的答卷。是他们的默默付出，才使得我能一路走到今天。

参考文献

- [1] 王丽娜, 张学恒, 王伟晨. 基于协同过滤算法的智能推荐系统研究[J]. 辽宁工业大学学报(社会科学版), 2015, 03:24-26.
- [2] 党博, 姜久雷. 基于评分差异度和用户偏好的协同过滤算法[J]. 计算机应用, 2016, 04:1050-1053.
- [3] 李映, 李玉龙, 王阳萍. 一种改进的混合协同过滤推荐算法[J]. 电子科技, 2016, 04:45-48.
- [4] 吴雄峰, 贾年. 基于用户特征和时间效应的协同过滤算法[J]. 现代计算机(专业版), 2016, 10:21-24.
- [5] 韩亚楠, 曹菡, 刘亮亮. 基于评分矩阵填充与用户兴趣的协同过滤推荐算法[J]. 计算机工程, 2016, 01:36-40.
- [6] Xiao-LinXu, Guang-LinXu. Improved Collaborative Filtering Recommendation Based on Classification and User Trust[J]. Journal of Electronic Science and Technology, 2016, 01:25-31.
- [7] 查九, 李振博, 徐桂琼. 基于组合相似度的优化协同过滤算法[J]. 计算机应用与软件, 2014, 12:323-328.
- [8] 吕果, 李法运. 基于改进协同过滤的移动个性化推荐服务研究[J]. 情报探索, 2014, 02:101 - 105.
- [9] ZHAOFeng, XIONGYan, LIANGXiao, GONGXudong, LUQiwei. Privacy-Preserving Collaborative Filtering Based on Time-Drifting Characteristic[J]. Chinese Journal of Electronics, 2016, 01:20-25.
- [10] 范家兵, 王鹏, 周渭博, 燕京京. 在推荐系统中利用时间因素的方法[J]. 计算机应用, 2015, 05:1324-1327.
- [11] 龚安, 高云, 高洪福. 一种基于项目属性评分的协同过滤推荐算法[J]. 计算机工程与科学, 2015, 12:2366-2371.
- [12] 李梁, 张海宁, 李宗博, 陈佳瑜. 融合用户属性的协同过滤推荐算法在政府采购中的应用[J]. 重庆理工大学学报(自然科学), 2015, 01:76-81.
- [13] 苏芳芳, 王成, 陈维斌, 张玉侠. 相似度最优加权协同过滤推荐模型[J]. 小型微型计算机

系统, 2016, 04:682-686.

[14] 吴应良, 姚怀栋, 李成安. 一种引入间接信任关系的改进协同过滤推荐算法[J]. 现代图书情报技术, 2015, 09:38-45.

[15] 黄绍川. 基于聚类算法的电子商务日志挖掘商业智能研究[J]. 中国商贸, 2014, 01:124-125.

[16] YingXia, ZhongzhaoZhang, LinMa, YaoWang. Semi-Supervised Clustering Fingerprint Positioning Algorithm Based on Distance Constraints[J]. Journal of Harbin Institute of Technology, 2015, 06:55-61.

附 录

1. 用户信息结构体

```
//用户信息
//u.user
struct user_information_2{
    int Age;//年龄
    char Gender;//性别
    std::string Occupation;//职业
    std::string Zip_code;//邮编
};
```

2. 电影信息结构体

```
//电影信息
//u.item
struct movie_information_2{
    std::string Title; //名字
    std::string Time; //时间
    std::string Address;//地址
};
```

3. 用户信息的读取和数据结构的建立

```
void user_file_read(char *filename)
{
    //u.user
    std::ifstream fin(filename);
    if (!fin){
        cout << "user file open failed." << endl;
        return;
    }
    int temp_id;
    int temp_age;//年龄
    char temp_Gender;//性别
    std::string temp_Occupation;//职业
    std::string temp_Zip_code;//邮编
    while (!fin.eof()){
        fin >> temp_id >> temp_age >> temp_Gender >> temp_Occupation >> temp_Zip_code;
        mapUser.insert(std::make_pair(temp_id,
user_information_2{ temp_age,temp_Gender,temp_Occupation, temp_Zip_code }));
    }
    fin.close();
}
```

4.电影信息的读取和数据结构的建立

```
void movie_file_read(char *filename)
{
    //u.item
    std::ifstream fin(filename);
    if (!fin){
        cout << "movie file open failed." << endl;
        return;
    }
    int temp_id;
    std::string temp_Title; //名字
    std::string temp_Time; //时间
    std::string temp_Address; //地址
    while (!fin.eof()){
        fin >> temp_id >> temp_Title >> temp_Time >> temp_Address;
        mapMovie.insert(std::make_pair(temp_id, movie_information_2{ temp_Title,
temp_Time, temp_Address }));
    }
    fin.close();
}
```

5.评分数据的读取和数据结构的建立

```
void rating_file_read(char *filename)
{
    FILE* fp = fopen(filename, "r");
    if (NULL == fp){
        return;
    }
    int len = fread(buffer, 1, MAXS, fp);
    buffer[len] = '\0';
    fclose(fp);
    fp = NULL;
    //字符串的处理，将数据导入所需的数据结构中
    int i=0;
    number[i] = 0;
    for (int j = 0; j < len; ++j){
        if (buffer[j] == ' ' || buffer[j] == '\n' || buffer[j] == '\t'){
            number[++i] = 0;
        }
        else{
            number[i] = number[i] * 10 + buffer[j] - '0';
        }
    }
    for (int p = 0; p < MAXN; p+=4){
```



```

        if (number[p] == 0) {
            break;
        }
        //评分矩阵生成
        rating[number[p]][number[p + 1]] = number[p + 2];
        //用户电影列表生成
        relation_for_use[number[p]].insert(number[p + 1]);
        //数组中数据为四个一组，分别为用户 ID，电影 ID，评价，时间戳
    }
}

```

6. 获取一个用户的平均评分

//获得某个用户的平均评价

```

double get_user_average_rate(int userID)
{
    double average_rate_for_choose = 0;
    int num = 0;
    for (int i = 1; i <= (int)mapMovie.size(); ++i) {
        if (rating[userID][i] != 0) {
            average_rate_for_choose += rating[userID][i];
            ++num;
        }
    }
    return average_rate_for_choose /= num;
}

```

7. 两个用户相似度计算:

//使用皮尔森相关系数计算用户相似度(修改的余弦相关)

```

double get_similar_percentage_2(int userID1, int userID2)
{
    double average_1 = 0; //用户 1 的平均评价
    double average_2 = 0; //用户 2 的平均评价
    int num_1 = 0; //用户 1 评价过的总数
    int num_2 = 0; //用户 2 评价过的总数
    for (int i = 0; i <= (int)mapMovie.size(); ++i) {
        if (rating[userID1][i] != 0) {
            average_1 += rating[userID1][i];
            ++num_1;
        }
        if (rating[userID2][i] != 0) {
            average_2 += rating[userID2][i];
            ++num_2;
        }
    }
}

```

```

/*cout << average_1 << " " << num_1 << endl;
cout << average_2 << " " << num_2 << endl;*/
average_1 /= num_1;
average_2 /= num_2;
double Denominator_1 = 0;//分母 1
double Denominator_2 = 0;//分母 2
double Molecule = 0;//分子
for (int i = 0; i <= (int)mapMovie.size(); ++i){
    if (rating[userID1][i] != 0 && rating[userID2][i] != 0){
        Molecule += (rating[userID1][i] - average_1)*(rating[userID2][i] - average_2);
    }
    if (rating[userID1][i] != 0){
        Denominator_1 += (rating[userID1][i] - average_1)*(rating[userID1][i] - average_1);
    }
    if (rating[userID2][i] != 0){
        Denominator_2 += (rating[userID2][i] - average_2)*(rating[userID2][i] - average_2);
    }
}
//改进的相似度
if ((Denominator_1 >= -0.01&&Denominator_1 <= 0.01) || (Denominator_2 >=
-0.01&&Denominator_2 <= 0.01)){
    return 0;
}
double similarity_2 = Molecule / (sqrt(Denominator_1)*sqrt(Denominator_2));
//用户特征值的加入
auto iter_1 = mapUser.find(userID1);
auto iter_2 = mapUser.find(userID2);
//特征相同的个数
int similar_attribute_num = 0;
if (iter_1->second.Gender == iter_2->second.Gender){
    similar_attribute_num++;
}
if (iter_1->second.Age == iter_2->second.Age){
    similar_attribute_num++;
}
if (iter_1->second.Occupation == iter_2->second.Occupation){
    similar_attribute_num++;
}
//特征所占比例为 0.08
similarity_2=PERCENTAGE*(double)similar_attribute_num/
(double) ATTRIBUTE_NUMBER+ (1 - PERCENTAGE) *similarity_2;
return similarity_2;
}

```

8.对自身已评价项目评分预测代码:

//计算预测评价

```
void calculate_predict(std::multiset<neighbours> test, int choose_user_ID, std::set<int> MovieIDs,
double all_similarity, double average_rate_for_choose)
```

```
{
    //进入公式运算
    for (auto iter_1 = MovieIDs.begin(); iter_1 != MovieIDs.end(); ++iter_1){
        double Molecule = 0;//分子
        int num = 0;//用于计算是否一个人都没看过，若如此则不入考虑
        //在邻居用户中查找这个电影的信息而后对其进行计算
        for (auto iter_2 = test.begin(); iter_2 != test.end(); ++iter_2){
            //获取邻居的电影列表
            auto iter_3 = relation_for_use.find(iter_2->user_ID);
            if (iter_3 == relation_for_use.end()){
                continue;
            }
            //在邻居用户的电影列表中查找是否存在该项
            auto iter_4 = iter_3->second.find(*iter_1);
            if (iter_4 == iter_3->second.end()){
                ++num;
                continue;
            }
            double get_neighbour_average = get_user_average_rate(iter_2->user_ID);
            double rate = get_user_rate_for_movie(iter_2->user_ID, *iter_1);
            Molecule += iter_2->similarity*(rate - get_neighbour_average);
        }
        //若是其他用户全都没看过则不入考虑
        if (num == NEED_NEIGHBOUR){
            continue;
        }
        else{
            //预测评分的计算
            double predict = average_rate_for_choose + Molecule / all_similarity;
            for_accuracy_2[*iter_1] = predict;
        }
    }
}
```

9.获取平均绝对误差:

//计算平均绝对误差（改良）

```
void calculate_accuracy_2(std::multiset<neighbours> test, int choose_user_ID,
std::set<int> MovieIDs, double all_similarity, double average_rate_for_choose)
```

```
{
    for_accuracy_2.clear();
```

```

    cout << "平均绝对误差计算中..." << endl;
    calculate_predict(test, choose_user_ID, MovieIDs, all_similarity,
average_rate_for_choose);
    //平均绝对误差(越小越好)
    double realaccuracy = 0;
    for (auto iter = for_accuracy_2.begin(); iter != for_accuracy_2.end(); ++iter) {
        //实际评价
        double realrate = get_user_rate_for_movie_2(choose_user_ID, iter->first);
        //cout << iter->first << " " << realrate << endl;
        realaccuracy += fabs(realrate - iter->second);
    }
    /*cout << "for_accuracy" << endl;
    for (std::map<int, rate_num>::iterator iter = for_accuracy.begin(); iter !=
for_accuracy.end(); ++iter) {
        cout << iter->first << " " << iter->second.rate << " " << iter->second.num <<
endl;
    }
    cout << realaccuracy << " " << for_accuracy.size() << endl;*/
    //平均误差
    realaccuracy = realaccuracy / for_accuracy_2.size();
    cout << "平均绝对误差为: " << realaccuracy << endl;
}

```

10. 获取老用户的相似用户:

//获得需要的用户(为老用户推荐)

```

void get_similar_neighbours(int choose_user_ID)
{
    auto iter = relation_for_use_1.find(choose_user_ID);
    cout << "相似用户获取中..." << endl;
    std::set<int> tempset(iter->second); //获取用户的感兴趣 movieID
    std::multiset<neighbours> multisetneighbour;
    neighbours temp_neighbour;
    //计算每一个用户和目标用户的相似度并排序
    for (auto s_iter = relation_for_use_1.begin(); s_iter !=
relation_for_use_1.end(); ++s_iter) {
        if (s_iter->first != choose_user_ID) {
            //改进后的相似度计算
            temp_neighbour.similarity = get_similar_percentage_2(choose_user_ID,
s_iter->first);
            temp_neighbour.user_ID = s_iter->first;
            multisetneighbour.insert(temp_neighbour);
        }
    }
    //只需要相似度最高的几个用户
}

```

```

    auto t_iter = multisetneighbour.begin();
    for (int i = 0; i < NEED_NEIGHBOUR; ++i) {
        ++t_iter;
    }
    //删除不必要的数
    multisetneighbour.erase(t_iter, multisetneighbour.end());
    cout << "相似用户获取完成, 计算预测值..." << endl;
    double all_similarity = 0;
    for (auto iter = multisetneighbour.begin(); iter != multisetneighbour.end();
        ++iter) {
        all_similarity += iter->similarity;
    }
    //存放指定用户的平均评价
    double average_rate_for_choose = get_user_average_rate_2(choose_user_ID);
    calculate_similarity(multisetneighbour, choose_user_ID, tempset,
all_similarity, average_rate_for_choose);
    //计算平均绝对误差(改良)
    calculate_accuracy_2(multisetneighbour, choose_user_ID, tempset,
all_similarity, average_rate_for_choose);
}

```

11. 针对未注册的新用户的推荐:

//为新用户推荐(未注册)

```

void get_similar_neighbour_for_new(int choose_user_ID)
{
    //计算已评价用户对电影的评价的平均值排序之后对用户实施推荐
    std::vector<PAIR> new_user_recommmed;
    double allrating = 0;
    int num = 0;
    for (int i = 1; i <= (int)mapMovie.size(); ++i) {
        for (int j = 1; j <= (int)mapUser.size(); ++j) {
            if (rating_1[j][i] > 0) {
                allrating += rating_1[j][i];
                num++;
            }
        }
    }
    if (num <= LINE) {
        new_user_recommmed.push_back(PAIR(i, 0));
    }
    else {
        new_user_recommmed.push_back(PAIR(i, allrating / num));
        allrating = 0;
        num = 0;
    }
}

```

```

}
//对获得的数据进行排序
std::sort(new_user_recommnd.begin(), new_user_recommnd.end(), cmp_by_value);
//获取实际所需数据
auto iter = new_user_recommnd.begin();
for (int i = 0; i < MOVIE_NUMS_FOR_NEW; ++i) {
    ++iter;
}
//删除不必要的数据
new_user_recommnd.erase(iter, new_user_recommnd.end());
show_recommend(new_user_recommnd);
}

```

12.为注册的新用户推荐:

//为新用户推荐(已注册)

```

void get_similar_neighbour_for_new_2(int choose_user_ID)
{
    char new_Gender;//性别
    cout << "性别(M:man, F:womean):  ";
    std::cin >> new_Gender;
    int new_Age;//年龄
    cout << "年龄区间如下" << endl;
    cout << "1:1-18" << endl;
    cout << "18:18-24" << endl;
    cout << "25:25-34" << endl;
    cout << "35:35-44" << endl;
    cout << "45:45-49" << endl;
    cout << "50:50-55" << endl;
    cout << "56:56+" << endl;
    cout << "年龄: ";
    std::cin >> new_Age;
    std::string new_Occupation;
    int new_Occupation_num;//职业
    cout << "职业区间如下" << endl;
    puts("1:administrator");
    puts("2:artist");
    puts("3:doctor");
    puts("4:educator");
    puts("5:engineer");
    puts("6:entertainment");
    puts("7:executive");
    puts("8:healthcare");
    puts("9:homemaker");
    puts("10:lawyer");
}

```

```
puts("11:librarian");
puts("12:marketing");
puts("13:none");
puts("14:other");
puts("15:programmer");
puts("16:retired");
puts("17:salesman");
puts("18:scientist");
puts("19:student");
puts("20:technician");
puts("21:writer");
cout << "职业: ";
std::cin >> new_Occupation_num;
switch (new_Occupation_num){
case 1:
    new_Occupation = "administrator";
    break;
case 2:
    new_Occupation = "artist";
    break;
case 3:
    new_Occupation = "doctor";
    break;
case 4:
    new_Occupation = "educator";
    break;
case 5:
    new_Occupation = "engineer";
    break;
case 6:
    new_Occupation = "entertainment";
    break;
case 7:
    new_Occupation = "executive";
    break;
case 8:
    new_Occupation = "healthcare";
    break;
case 9:
    new_Occupation = "homemaker";
    break;
case 10:
    new_Occupation = "lawyer";
    break;
```

```
case 11:
    new_Occupation = "librarian";
    break;
case 12:
    new_Occupation = "marketing";
    break;
case 13:
    new_Occupation = "none";
    break;
case 14:
    new_Occupation = "other";
    break;
case 15:
    new_Occupation = "programmer";
    break;
case 16:
    new_Occupation = "retired";
    break;
case 17:
    new_Occupation = "salesman";
    break;
case 18:
    new_Occupation = "scientist";
    break;
case 19:
    new_Occupation = "student";
    break;
case 20:
    new_Occupation = "technician";
    break;
case 21:
    new_Occupation = "writer";
    break;
default:
    break;
}
//存放用户相似度和 ID
std::multiset<neighbours> multisetneighbour;
int num = 0;
//根据用户特征提取出相似用户
for (auto iter = mapUser.begin(); iter != mapUser.end(); ++iter) {
    neighbours temp_neighbour;
    //计算每一个用户和目标用户的相似度并排序
    num = 0;
```



```

    if (iter->second.Gender == new_Gender) {
        ++num;
    }
    if (iter->second.Age == new_Age) {
        ++num;
    }
    if (iter->second.Occupation == new_Occupation) {
        ++num;
    }
    if (num == 0) {
        continue;
    }
    else{
        temp_neighbour.similarity = (double)num / (double)ATTRIBUTE_NUMBER;
        temp_neighbour.user_ID = iter->first;
        multisetneighbour.insert(temp_neighbour);
    }
}
auto iter_once = multisetneighbour.begin();
for (int i = 0; i < NEED_NEIGHBOUR; ++i) {
    ++iter_once;
}
//获取到最近邻的信息
multisetneighbour.erase(iter_once, multisetneighbour.end());
for (auto iter = multisetneighbour.begin(); iter != multisetneighbour.end();
++iter) {
    cout << iter->user_ID << " " << iter->similarity << endl;
}
//计算已评价用户对电影的评价的平均值排序之后对用户实施推荐
std::vector<PAIR> new_user_recommmed;
double allrating = 0;
num = 0;
for (int i = 1; i <= (int)mapMovie.size(); ++i) {
    for (auto iter = multisetneighbour.begin(); iter != multisetneighbour.end();
++iter) {
        if (rating_1[iter->user_ID][i] > 0) {
            allrating += rating_1[iter->user_ID][i];
            num++;
        }
    }
    new_user_recommmed.push_back(PAIR(i, allrating / num));
    allrating = 0;
    num = 0;
}

```

```

//对获得的数据进行排序
std::sort(new_user_recommmed.begin(), new_user_recommmed.end(), cmp_by_value);
//获取实际所需数据
auto iter = new_user_recommmed.begin();
for (int i = 0; i < MOVIE_NUMS_FOR_NEW; ++i){
    ++iter;
}
//删除不必要的数据
new_user_recommmed.erase(iter, new_user_recommmed.end());
//推荐
show_recommend(new_user_recommmed);
}

```

13. 对相似用户项目的评分预测函数:

```

//根据相似用户群体获取 movie 预测值，并计算一下准确值
void calculate_similarity(std::multiset<neighbours> test, int choose_user_ID,
std::set<int> MovieIDs, double all_similarity, double average_rate_for_choose)
{
    cout << "预测计算中..." << endl;
    //电影预测
    std::map<int, double> movieID_estimation;
    //指定用户没有评价过的项目
    std::set<int> movie_without_rate;
    //获取用户没有评价过的项目集合
    for (auto iter = test.begin(); iter != test.end(); ++iter){
        //查找到最相似用户获取他们的感兴趣电影信息
        //auto s_iter = relation_for_use.find(iter->user_ID);
        auto s_iter = relation_for_use_1.find(iter->user_ID);
        //从最相似用户的电影列表里查找到指定用户没接触过的信息
        for (auto t_iter = s_iter->second.begin(); t_iter != s_iter->second.end();
        ++t_iter){
            //在指定用户的电影列表中查找是否存在该电影
            auto f_iter = MovieIDs.find(*t_iter);
            //若该电影在指定用户的表中不存在则添加进待处理的表中
            if (f_iter == MovieIDs.end()){
                //填充为评价的电影列表
                movie_without_rate.insert(*t_iter);
            }
        }
    }

    for (auto iter_1 = movie_without_rate.begin(); iter_1 != movie_without_rate.
end(); ++iter_1){
        double Molecule = 0;//分子

```

```

int num = 0; //用于计算是否一个人都没看过，若如此则不入考虑
//在邻居用户中查找这个电影的信息而后对其进行计算
for (auto iter_2 = test.begin(); iter_2 != test.end(); ++iter_2) {
    //获取邻居的电影列表
    //auto iter_3 = relation_for_use.find(iter_2->user_ID);
    auto iter_3 = relation_for_use_1.find(iter_2->user_ID);
    if (iter_3 == relation_for_use_1.end()) {
        continue;
    }
    //在邻居用户的电影列表中查找是否存在该项
    auto iter_4 = iter_3->second.find(*iter_1);
    if (iter_4 == iter_3->second.end()) {
        ++num;
        continue;
    }
    double get_neighbour_average = get_user_average_rate(iter_2->user_ID);
    double rate = get_user_rate_for_movie(iter_2->user_ID, *iter_1);
    Molecule += iter_2->similarity*(rate - get_neighbour_average);
}
//若是其他用户全都没看过则不入考虑
if (num == NEED_NEIGHBOUR) {
    continue;
}
else {
    //预测评分的计算
    double predict = average_rate_for_choose + Molecule / all_similarity;
    movieID_estimation[*iter_1] = predict;
}
}

cout << "预测计算完成，获取影片信息..." << endl;
sort_similarity(movieID_estimation);
}

```

14.对用户相似度进行排序:

//对相似度从大到小排序

```

void sort_similarity(std::map<int, double> movieID_estimation)
{
    //将数据放入 vector 中并排序
    std::vector<PAIR> tempneed(movieID_estimation.begin(),
movieID_estimation.end());
    std::sort(tempneed.begin(), tempneed.end(), cmp_by_value);
    //获取实际所需数据
    auto iter = tempneed.begin();
}

```

```
    for (int i = 0; i < MOVIE_FOR_OLD; ++i) {  
        ++iter;  
    }  
    //删除不必要的数据  
    tempneed.erase(iter, tempneed.end());  
    show_recommend(tempneed);  
}
```

15.最终推荐代码:

//显示最终推荐结果

```
void show_recommend(std::vector<PAIR> ID_NEED)  
{  
    cout << "推荐结果如下: " << endl;  
    for (auto iter = ID_NEED.begin(); iter != ID_NEED.end(); ++iter) {  
        auto t_iter = mapMovie.find(iter->first);  
        if (t_iter != mapMovie.end()) {  
            cout << t_iter->first << " "  
                << t_iter->second.Title << " "  
                << t_iter->second.Time << " "  
                << endl;  
        }  
        else {  
            cout << "no have this movie" << endl;  
        }  
    }  
}
```