黑龙江大学

本科学生毕业论文

论文题目：　基于协同过滤算法的电影资源个性化推荐系统设计与实现

学　　院：　信息管理与信息系统

年　　级：　2014 级

专　　业：　电子商务

姓　　名：　陈宏骏

学　　号：　20144798

指导教师：　蔡庆平

2018 年　2 月　26 日

摘要

在互联网时代下，信息量呈指数级增长。海量数据以 0、1 方式编码存放在磁盘等设备中非常方便，但是对于人类来说，我们的脑容量有限，在海量数据中挑选出适合自己的信息变得非常困难。简直如大海捞针，即使使用了分类方法和搜索引擎，我们还是要花大量时间去挑选适合自己的信息，并且经常选取了错误的信息。在这样的情况下，个性化推荐系统应运而生。从起初的基于内容的推荐，到现在最常用的协同过滤推荐等推荐算法，我们从繁琐庞大的信息中解脱出来，可以接受大量系统推荐给我们的信息。

在大多数电子商务网站上，都运行着个性化推荐系统，不仅提高了用户购买率，还提高了用户体验。其系统内部实现由于基础数据不同也各有不同，而协同过滤算法被广泛应用到这些系统中，发挥着巨大的作用。

本文首先详细介绍了协同过滤算法与推荐系统概要，分析了各种协同过滤算法的优缺点。然后设计了一个简易的电影资源推荐系统，并使用 python、微信公众号等技术实现了这个推荐系统，最后对这个推荐系统做了简单的测试和结果分析。

关键词

个性化推荐系统；协同过滤；机器学习；数据挖掘；数据分析；python

Abstract

In the Internet age, the amount of information is exponential. Huge amounts of data to 0, 1, encoded in devices such as disk is very convenient, but for humans, our brain capacity is limited, selected for their own information in the huge amounts of data becomes very difficult.It's like looking for a needle in a haystack, and even using a taxonomy and search engine, we still have to spend a lot of time sorting out what's right for us and often picking the wrong information. In this case, the personalized recommendation system comes into being.Based on the recommendations from the content, from the beginning to now, the most commonly used collaborative filtering recommendation recommendation algorithm, such as free from complex huge information, we can accept a large number of system is recommended for our information.

On most e-commerce sites, the personalized recommendation system is running, which not only improves the user purchase rate, but also improves the user experience.The internal implementation of the system is different due to the different basic data, and the collaborative filtering algorithm is widely applied to these systems and plays a huge role.

In this paper, we first introduce the collaborative filtering algorithm and recommendation system profile, and analyze the advantages and disadvantages of various collaborative filtering algorithms.Then design a simple movie recommendation system resources, and using the python, micro letter of public technology implements this recommendation system, finally, the recommendation system has made the simple test and the result analysis.

Key words

Personalized recommendation system；Collaborative filtering；Machine learning；Data mining；Data analysis；Python；

目录

1. 绪论

电子商务网站中，推荐系统广泛存在。典型的，亚马逊商城出现的“您可能需要购买”或者“购买过此类商品的人还购买过”等字样的推荐商品，打开淘宝商城直接呈现在你眼前的商品，都是推荐系统针对用户个人产生的推荐。包括非电商网站也会有非常多的推荐广告，如百度搜索引擎侧边的广告，网上博客侧边的广告。这些广告经常会引起我们的注意，因为它们也产生于推荐系统，是推荐系统为用户“量身制定”的广告。如果没有这些推荐，或者推荐出来的信息都是随便推荐的，那可能起不到广告作用，反而会引起用户的反感，造成用户流失。

1.1 课题背景及意义

二十世纪九十年代以来，电子商务在国内外蓬勃发展，网上浏览信息已经成为人们主要获取信息的途径,网上购物也成为人们的主要消费行为。而网上信息的爆炸式增长，海量数据呈现在用户面前，使用户无所适从，用户如何方便快捷地找到自己需要的商品变得尤为重要[1]。在这种情况下，依赖用户反馈信息的推荐系统应运而生，而且得到了越来越广泛的应用。

首先，推荐系统解决了“信息过载”的额问题，网上信息数量繁多，重复率也极高，如何快速找到适合自己的信息,成为互联网的一大关注点。推荐系统利用用户反馈信息，预测用户的喜好，实时向用户推荐适合的信息，从而提高网络服务的质量，增加企业网站竞争力。

其次,推荐系统可以大规模产生推荐,当今用户对产品和服务的品质要求越来越高，每个用户都希望能够找到符合自己期望的个性化和多样化服务,这时候靠人工实现推荐显然效率低下，而推荐系统能够实现高效、实时、大规模的推荐。

再者，推荐系统可以评估推荐效果，收集用户反馈信息，通过知识挖掘技术，能够获取相应的知识，在客户管理、市场分析等领域非常有用。

总体来说，互联网上的大多数网站为推荐系统的应用提供了平台，而推荐系统的推荐效果也在快速提升。一方面，互联网的发展，特别是电子商务领域，在金融的推动下，电子商务为推荐系统的应用发展提供了广阔的平台，推动推荐算法的发展。另一方面，推荐系统节约了用户浏览查找适合信息的时间，提高了用户的体验。

1.2 国内外研究现状

电子商务推荐系统正式定义是在1997年给出:“它是利用电子商务网站向客户提供商品信息和建议，帮助用户决定应该购买什么商品，模拟销售人员引导用户完成购买过程”，现在这个定义已经被广泛引用。

目前主要有两种类型的推荐系统，一种是网上购物环境下以商品为推荐对象的推荐系统，为用户推荐符合其需求的产品，如书籍、音像、实物等；另一种是以网页为对象的个性化推荐系统,为用户推荐符合其兴趣的网页,目前国内外对后者的研究应用较多，本文实现的推荐系统可分类为前者。

推荐系统研究的最主要热点就是如何提高推荐算法的推荐质量和效率，因为推荐算法是整个推荐系统的核心，它的性能决定了推荐系统的成功与否[2]。协同过滤算法是目前研究最成功的算法，同时它也是最热门的推荐算法。它利用与当前用户具有相似兴趣的其他用户的信息进行推荐。此算法也是本文的重点算法。

虽然协同过滤算法被广泛应用，快速发展，但是它面临着两个基本的挑战：

（1） 算法的可拓展性：随着系统数据量的迅速累积，推荐系统面临着越来越严重的问题，难以满足系统实时性要求。

（2） 推荐系统的推荐质量：推荐系统必须向用户产生有价值的推荐，否则用户就会对系统失去信息。

另外协同过滤算法还存在其他一些问题，不同企业都有相对应的解决方案。

1.3 论文内容结构

本文对协同过滤算法和该算法的优缺点进行介绍和分析，并提出解决方案。并从零实现了一个基于协同过滤算法的电影资源个性化推荐系统。

在第一章中介绍了推荐系统的发展和研究现状。论文第二章研究系统的搭建，分别包含了数据库的填充，前端后台的搭建，以及简单的测试。第三章研究协同过滤算法的细分算法和优缺点，以及使用场景和在系统的实现思想。第四章在搭建好的系统中加入个性化推荐算法，并且解决数据稀疏问题，经过测试完成整个系统。第五章对推荐系统进行分析，并提出优化方案。

2. 协同过滤与相关技术

2.1 推荐算法介绍

2.1.1 基于内容的推荐算法

基于内容的推荐一种传统的推荐方法，它依靠内容的特征将内容分类，并推荐给可能感兴趣的用户。这种方法实现简单，在系统的整个生命周期中都能使用。在后面的章节中将使用此类方法弥补协同过滤算法的数据稀疏问题。

基于内容的推荐是在信息检索领域提出的，该算法可以使用的场景非常多，其基本思想是利用用户喜欢的对象之间的相关性而做出推荐。推荐时首先为每一个推荐对象抽取出一些特征来表示此对象，例如抽取文章的关键词，然后利用一个用户过去的喜好记录，及喜不喜欢这个关键词相关的内容，然后给出该用户的推荐内容。

例如信息分类的电影网站，如果你喜欢看喜剧，则基于内容的推荐算法的推荐结果就是喜剧电影；如果你喜欢看某个明星主演的电影，则它的推荐结果就是这个明星主演的其他电影。

基于内容推荐有以下几个优点：

（1） 推荐容易理解：只考虑单个用户的喜好和内容之间的匹配程度。推荐的内容属于用户关注的范畴。

（2） 用户具有独立性：用户与用户之间的数据不存在关联，用户只和数据相关。

（3） 无新推荐对象冷启动问题：当增加新的内容时，可以迅速提取出该内容的特征，而不会出现数据稀少使算法无法做出推荐。

该算法也存在一些缺点：

（1） 推荐对象的特征提取能力有限：文字内容非常容易提取出关键词，但是如果是影视频或图片则无法提取内容特征。

（2） 无法发现用户的潜在兴趣：该算法只依赖用户过去的兴趣产生推荐，而某些特征的内容用户现在不感兴趣是因为不知道它的存在，基于内容的推荐不会产生此类推荐结果。

（3） 推荐内容重复性较高：提取的特征不能完全代表主体内容，无法识别两个内容的相似程度。

2.1.2 协同过滤算法

协同过滤最早的应用是 1992 年 Xerox 公司在 Palo Alto 的研究中心解决资讯过载的问题。它比基于内容的推荐算法更有效。

传统的基于内容推荐，例如把最近被搜索频率较高的电影推荐给每一个用户；或者从用户浏览的文章中提取出关键词，根据关键词推荐；或者用户最近观看科幻片比较多，便把评分较高的科幻片推荐给用户。这些方法在特定的场景也有显著的效果，但是这些算法都简单的根据单个用户的信息作出预测，而没有考虑到用户与用户之间的相关关系。而协同过滤算法则“考虑的比较周全”。将这部分信息也当做预测的原数据。根据历史数据来建立模型，预测出用户喜欢却还没有得到的信息，推荐给用户，产生价值。现在协同过滤算法应用广泛，特别是电子商务领域的商品推荐，最著名的电子商务推荐系统应属亚马逊商城的图书推荐系统，用户查看自己感兴趣的书籍，马上会在底下看到“买了这本书籍的用户还购买了”的字样。亚马逊是在“对同样一本书有兴趣的读者们兴趣在某种程度上相近”的假设前提下提供这样的推荐，此举也成为亚马逊网络书店为人所津津乐道的一项服务，各网络书店也跟进做这样的推荐服务。

协同过滤的出发点是：兴趣相近的人可能会对同样的东西感兴趣。只要根据用户历史喜好数据，分析出相似爱好的用户，然后就能根据相似用户的意见或者评价进行推荐[3]。

2.1.3 协同过滤算法分类

基于人口统计学推荐。例如基于地理位置推出热门搜索结果，像百度搜索引擎使用的就是这一类，推荐的结果是近期的一些大众热门话题。

基于内容推荐。根据内容的元数据，发现内容之间的相关性，基于用户以前的喜好记录推荐相似的内容给用户[4]。例如推荐相同导演的电影、相同演员的电影之类的。

基于协同过滤算法推荐。这个算法根据内部实现不同又分为以下几种：

（1） 基于用户的协同过滤推荐。方法是尝试寻找与目标用户有相同或者相似喜好记录的邻居用户，然后根据将他们的喜好互相推荐。在本系统中将采用这一类协同过滤推荐算法。

（2） 基于项目的协同过滤推荐。根据所有用户对内容的历史喜好记录，发现物品

之间的相似度，然后根据用户的历史喜好记录将类似物品推荐给该用户。例如喜欢 A 内容的人 10 个人中有 9 个都喜欢 B 内容，则猜测 A 和 B 内容相似，当有新的用户喜欢 B 内容就可以将 A 内容推荐给他。

（3） 基于模型的协同过滤推荐。基于样本中的用户历史喜好记录，训练出一个推荐模型，然后根据实际用户喜好的信息进行预测推荐。

基于用户的协同过滤算法时最早出现的协同过滤算法，也是使用最为广泛的协同过滤算法。它以用户-项目评分矩阵中的行为基础计算用户与用户之间的相似性。协同过滤算法虽然可以简单的分为三个类型，但是它们都有诸多的改进版本。而目前为止，绝大多数的改进都是在基于用户的协同过滤基础上给出。

	UserCF	ItemCF
性能	适用于用户较少的场合，如果用户很多，计算用户相似度矩阵代价很大	适用于物品数明显小于用户数的场合，如果物品很多（网页），计算物品相似度矩阵代价很大
领域	时效性较强，用户个性化兴趣不太明显的领域	长尾物品丰富，用户个性化需求强烈的领域
实时性	用户有新行为，不一定造成推荐结果的立即变化	用户有新行为，一定会导致推荐结果的实时变化
冷启动	在新用户对很少的物品产生行为后，不能立即对他进行个性化推荐，因为用户相似度表是每隔一段时间离线计算的	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
	新物品上线后一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给和对它产生行为的用户兴趣相似的其他用户	但没有办法在不离线更新物品相似度表的情况下将新物品推荐给用户
推荐理由	很难提供令用户信服的推荐解释	利用用户的历史行为给用户做推荐解释，可以令用户比较信服

图 2-1 不同协同过滤算法比较

不同的协同过滤算法各有优缺点，图 2-1 描述了基于用户与基于项目的协同过滤的优缺点比较。

2.1.4 协同过滤优化

在最基本的三种协同过滤算法之外，还有很多扩展的协同过滤算法，可以说协同过滤只是一个主要思想，具体实现就有非常多的版本。在基于模型的协同过滤中，有很多使用机器学习的思想来建模的解决方案，主流可以分为关联算法、聚类算法、分类算法、

回归算法、矩阵分解、神经网络、图模型以及隐语义模型。协同过滤的改进版本也有很多相关书籍专门介绍，在不同的领域会使用不同的改进版。

2.1.5 协同过滤推荐的优缺点

协同过滤算法作为应用最广泛的推荐算法，相比传统的推荐算法，其拥有非常明显的优势。传统的过滤算法使用统计学的方式猜测一个群体的喜好，而协同过滤算法考虑的更加周全，协同过滤算法关心每个用户对信息内容的评价、关心每一个用户的个体差异。

协同过滤算法有以下几个优点：

（1） 发挥历史数据的价值：用户的历史数据在电子商务发展初期并没有引起注意，以前的观念是以产品为中心，“货好不愁卖”。而如今成熟的电子商务网站提倡“用户至上”，亚马逊等电子商务网站及其关系用户的历史信息，并且总是花大成本获取大量高质量数据，研究如何从大数据中挖掘出更大的价值。协同过滤算法便是一个合理应用用户历史数据的算法，它使以前一文不值的用户历史喜好信息发挥出巨大的价值，减少用户浏览商品的时间，并加快了金融的流通。

（2） 体现个体差异：协同过滤算法考虑每一个用的个体差异，使用协同过滤算法推荐给每一个用户的信息基本是不相同的。而这也符合客观事实。人与人之间存在个体差异在一些方面很大甚至会相反。协同过滤算法能够满足人的这一需求。

（3） 不需要关心内容的特征：协同过滤算法不考虑信息的具体内容，对于信息只关心每个用户对它的评分，而不关心信息的类别。这使得算法变得简单，不需要像大型超市一样对商品进行精确的分类。协同过滤算法属于机器学习的一个分支，一定程度上它拥有于人一样的思考方式，依靠用户对信息的评价进行预测。实现也不复杂。

协同过滤算法完全依赖于用户评分，通过构建用户对内容的评分矩阵，使用统计技术查找与目标用户有相似兴趣爱好的用户。其基本思想是用户会对邻居用户感兴趣的信息内容也感兴趣，因此，用户的评分数据收集的质量越高、数量越多，协同过滤算法的推荐质量就越高。但是由于协同过滤算法最大的应用场景电子商务站点的商品数量庞

大，而且不断增加，每个用户不可能会对所有的信息内容都做一次评分，每个用户可能只会对庞大信息库中的少数几条进行评分。通常在1%以下，这样会导致推荐质量下降。

虽然存在诸多缺点，但是对应这些缺点的解决方案也层出不穷。

（1） 用户量增加会导致计算时间变长：在为每个用户寻找邻居时，需要将这个用户的兴趣爱好与其他的每一位用户做一次比较，然后找出最近的“邻居”。如果有用户量为 n，则为每一个用户找到最近邻居的时间复杂度为(n-1)+(n-2)+……+2+1，也就是(n*n)/2，用户量每增加一个时间复杂度将呈指数增长。

（2） 数据稀疏导致推荐结果不满意：用户评价数据非常少，导致系统没有足够的数据为其找到邻居，即使依靠少量数据为其找到邻居用户，质量也可能不会很好，降低了推荐的质量，甚至出现完全不符合用户需求的推荐使用户反感。

（3） 冷启动问题：在系统创建初期，没有用户评分，这是数据稀疏的极端情况。这个时候协同过滤算法没有办法作出推荐。只能依靠其他算法进行推荐或者放弃推荐。

（4） 建立模型耗时：系统中用户与用户的邻居关系可以看做是一个推荐模型，基于模型的协同过滤算法事先采用机器学习的方法对用户数据进行离线学习建立模型，然后使用建立好的模型在线预测用户的喜好，进行推荐。然而这样做无法获取到最新的用户喜好信息，需要定期的进行重新建模。而其他协同过滤算法则在每次需要产生推荐时为用户计算出最近邻居，这样在计算上可能会花费较长的时间。

协同过滤算法基于用户对数据的评价进行预测推荐，这也导致它需要这些数据的支撑，在实际应用中，如果是没有区分用户的系统，或者是体现不出用户对信息内容的喜好的系统，就使用不了协同过滤算法。幸运的是，大多数系统中都能满足以上条件，协同过滤算法算法也得到了广泛的使用。

2.1.6 应用场景

协同过滤算法主要在以下一些场景下得到了深度应用：

（1） 电子商务：在亚马逊、淘宝、京东等大型电商网站上，如果用户需要购买商品，就会发生账号注册和浏览、搜索商品，购买商品后会有评价功能。拥有

这些数据，推荐系统有非常大的用武之地，推荐系统能够刺激用户的购买欲，增加平台收入，能够帮助用户找出所需要的商品，减少用户查找商品的时间，提高用户体验。当数据量大时，甚至能够精准预测用户未来需要购买的物品，而此时，电子商务网站需要做的不仅是盈利，还需要保护用户的信息隐私。

（2） 博客：在浏览博客时，不同人群有着不同的兴趣，如果是一名计算机专业的学生，则他浏览最多的博客应该是技术相关的，而如果系统根据协同过滤算法为其找到了相似博客或者相似用户，则产生出来的推荐结果也会是高质量的。如 CSDN 等博客网站就是用协同过滤算法产生大量推荐内容。用户在浏览博客时，博客页面底下会推荐产生非常多的相似博客链接。

（3） 搜索引擎：在百度和谷歌等搜索引擎系统中，也有用户账号系统，当用户登录百度账号后在百度进行搜索，则系统会采集用户的搜索记录，当用户进入搜索引擎后，页面侧面会出现系统推荐结果，这些结果属于混合算法产生的推荐结果，而不是单靠协同过滤产生的推荐。

（4） 图书馆系统：在学校图书馆系统中，每一个学生或者图书馆用户都会有一个账号，在借阅图书时系统便能采集到学生的借阅信息，从而查找到相似用户或者相似图书产生推荐。

2.2 相关技术介绍

在本文的推荐系统搭建在安装 linux 操作系统的云服务器（Elastic Compute Service 简称 ECS）上。除了核心的推荐算法，还需要以下技术支持：

（1） python：一门被广泛应用的编程语言，有很多不同功能的第三方库可以使用。调用这些第三方库，简单的 python 代码可以实现强大的功能：网页爬虫、神经网络、、图像识别、文章分词及关键字提取、网站后台等等。本系统中使用 python 作为主要开发语言，使用 python 爬取源数据，并使用 python 编写后台逻辑、与数据库交互等。

（2） MySQL：一个免费的关系数据库软件，当拥有大量记录，并且记录与记录之间有复杂的联系时，把这些信息直接存到文件系统是不适合的，存到非关系数据库又体现不出记录之间的复杂关系，比较好的选择就是使用 MySQL，MySQL 是关系数据库中比较流行的一个。在本系统中使用 MySQL 数据库存

储电影资源的信息、用户的信息、用户的行为信息。

（3） 微信公众号：作为一个 C/S 架构的服务，开发一个客户端软件所需要的时间太长，使用 HTML 开发一个网站前端也需要写大量代码，由于本系统对前端界面需求不大，为了节省时间，本文使用腾讯公司的微信公众号作为后台接入点，将微信聊天界面当成用户界面使用，这样避免了编写前端代码，用户也可以使用微信轻松的访问系统。

（4） Web.py：一个使用 python 语言的轻量级网站后台框架，使用这个框架可以简单的实现网站后台，与微信公众号对接，作为公众号的后台，提供用户数据采集功能和推荐计算服务。

除了上述的主要技术，在系统实现阶段还会用到一些其他技术，例如：使用 pythond 的第三方库 requests 下载 HTML 页面内容、使用 Xpath 语法解析 HTML 页面内容、使用 pymysql 库操作 MySQL 等等。

3. 系统分析与设计

3.1 需求分析

本文将利用上一章中介绍的技术搭建一个电影资源个性化推荐系统。本系统对用户提供一个微信公众号，微信用户对公众号添加关注后就能享受到电影个性化推荐服务。

在用户使用推荐系统时，会用到几个功能：

（1） 个性化推荐功能：这是用户所需要的服务，也是推荐系统的必要功能。

（2） 评价功能：用户可以对看过的电影进行一个评价，这份数据将作为推荐算法做抉择的依据。

（3） 搜索功能：一个基本的功能，能让用户准确搜索出电影的详细信息。

除了上述的必要功能，用户可能还会用到一些其他的扩展功能：

（1） 说明书：怎么与公众号进行交互，让系统进行相关的操作，把这个写成说明书给用户看。

（2） 收藏夹：用户对于喜欢的电影可以添加到收藏夹，以免忘记。

在使用该系统时，用户可以主动向公众号发出一些命令，然后公众号完成相应的功能，可能还需要返回结果内容。

推荐功能可以将内容主动推荐给用户，也可以在用户使用系统时显示出推荐结果，也可以让用户主动寻求推荐。由于本系统使用微信公众号，功能有限，所以使用的方法是用户主动寻求推荐，当用户发送“推荐”二字给系统时，系统做出推荐结果返回给用户。

评价功能可以在用户购买商品后进行评价，或者在观看电影后对电影进行评价，但是由于本系统没有购物系统，只是简单的推荐系统，所以这个功能也是用户主动评价，用户可以对电影进行一个 0-10 分的评价。系统将记录这个评分，用来作为个性推荐的依据。

搜索功能则比较简单，用户直接发送电影名，系统将电影详细信息返回给用户。

其他功能对推荐系统的作用不是很大，本文没有做出具体实现。

3.2 系统设计

首先需要准备工作环境，工作环境非常简单，一台电脑作为后台，一个微信公众号作为前端。电脑需要拥有公网 IP，这样微信公众号才能接入到自己编写的后台程序，最好的选择就是购买一个云服务器，配置不需要高。在本系统中使用安装了 CentOS 操作系统的云服务器，微信公众号使用微信申请个人微信公众号。整体结构如下图所示：

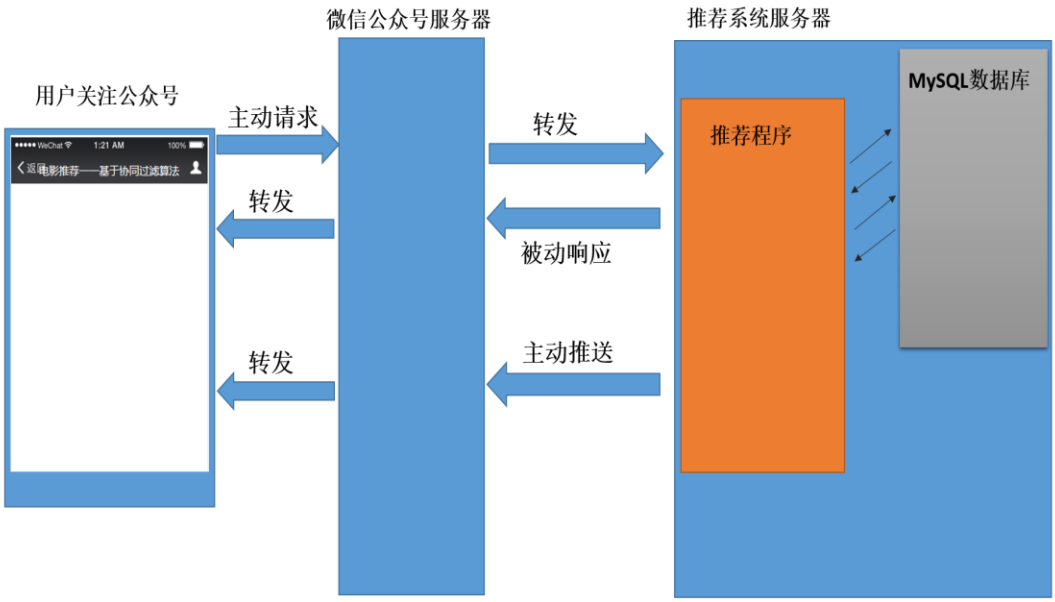


图 3-1 系统结构

在实现推荐系统时，我们需要两个重要的数据资源。一是用户需要的信息资源，在本系统中就是电影资源；二是用户信息资源，用户的信息和用户的历史评价记录。将这些数据存放在 MySQL 数据库里面供推荐程序读取和写入。有了这些基本的数据，我们的算法才能跑起来，它会根据用户的历史评价记录，作出新的推荐。由于算法根据历史数据作出推荐，所以数据的可靠性和数据量能够极大的影响推荐效果。

3.2.1 微信公众号

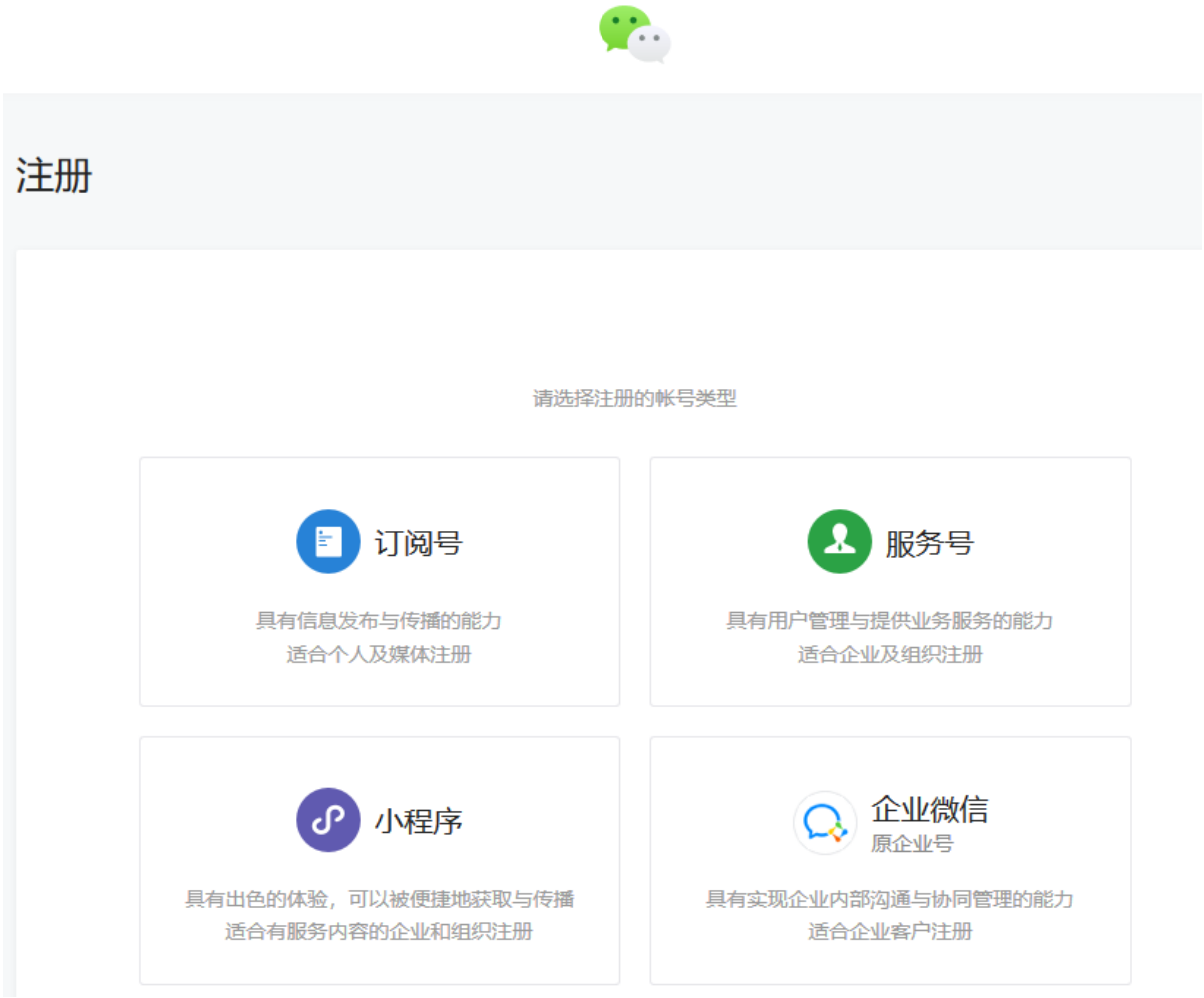


图 3-2 微信公众号申请

在微信公众号官网上，可以注册以下几种账号，如图 3-2。而这里只需要简单的注册订阅号即可。个人订阅号虽然功能权限较少，但是只需要提供个人信息就能申请注册。个人订阅号能够每天群发一条消息到用户，能够在用户主动发送文字、图片或语音时回应消息[5]。在该系统中，这些权限已经足够。

根据注册流程，很容易就能得到一个个人订阅号，每一个微信用户扫描此订阅号的二维码或者直接搜索订阅号名称就能关注此订阅号，但此时的公众号没有任何功能。

3.2.2 接口定义

当订阅号申请完成之后，用任意微信就能搜索关注，如图 3-3。

图 3-3 微信订阅号信息

有了微信公众号后，就相当于有了用户的交互界面。当用户发送文字请求时，微信公众号回复消息。现在给微信公众号定义以下操作接口：

命令 参数 1 参数 2 …

用户只要打开微信公众号发送这样格式的文字，微信公众号就给出相应回复。定义

以下几种操作：

（1）搜索电影信息：

搜索 电影名

（2）评价电影：

评价 电影名 分数

（3）请求个性化推荐：

推荐

（4）简单搜索：

电影名

当微信公众号服务器收到用户发来的请求时，会转发到我们的后台，然后在我们的后台处理后返回结果，最终结果返回到用户界面。

图 3-4 用户交互界面

用户操作如图 3-4 所示，由于现在后端还没有开发，所以现在用户发送请求时得不到返回结果，微信会提示服务出现故障。这些功能的实现将在下一节内说明。现在只说明这

些接口的作用，最简单的，用户可能会主动了解一些电影的信息，为此，提供两个命令：“搜索 电影名”或者直接发送“电影名”。然后公众号返回改电影的详情信息。还要获取用户的喜好信息，所以提供了“评价 电影名 评分”命令，在用户评价电影时将会在后台采集该信息作为协同过滤算法的数据依据。最后提供“推荐”命令让用户获取个性化推荐的结果。由于个人订阅号的权限有限，只能被动响应用户的请求，所以在本系统中所有的功能都靠用户主动发送命令触发。而发送命令只需要打开微信公众号输入文字即可。

3.2.3 前后端交互接口

在微信公众号平台上开启开发者模式，并设置一个令牌，然后在开发者设置的基本设置中，指定一台服务器，并启用服务器配置，如图 3-5 所示，这样设置之后，当用户发送消息时，微信公众号服务器将收到消息并封装成 XML 格式的数据转发到指定的后台服务器的 80 端口，然后等待此服务器回复后转发到用户微信。本节对微信公众号与后台服务的交互接口进行说明。

图 3-5 配置公众号接入后台服务器

3.2.4 文本请求

当用户发送“推荐”两个字到公众号时，公众号服务器会以 POST 的方式发出如下 XML 格式的文本信息给我们指定的服务器的 80 端口。内容如图 3-6 所示，内容包含了公众号 ID、用户微信 ID、发送时间、消息类型、消息内容以及一个消息 ID。

```
<xml><ToUserName><![CDATA[gh_6b3fa33e8d76]]></ToUserName>
<FromUserName><![CDATA[oJj6WwDgpSDqyPj9hVHMDv8olThc]]></FromUserName>
<CreateTime>1523093149</CreateTime>
<MsgType><![CDATA[text]]></MsgType>
<Content><![CDATA[推荐]]></Content>
<MsgId>6541635264140886273</MsgId>
</xml>
```

图 3-6 微信请求格式

当后台服务接收到该信息之后，我们只需要将发送方名字和接收方名字的位置交换，然后将 Content 内容替换成后台程序处理的结果，发送到微信公众号服务器，经过公众号服务器的处理和转发，用户便能收到回复。

在本系统中，只处理用户发送的文字请求，不处理图片和语音。

3.2.5 事件处理

除了文本请求，还有一类通知叫做事件，当有新用户关注公众号或者老用户取消关注时，公众号服务器会封装相应的 XML 内容发送给后台服务器。例如，图 3-7 为新用户关注公众号时，公众号服务器发送给后台服务器的内容。

```
<xml><ToUserName><![CDATA[gh_6b3fa33e8d76]]></ToUserName>
<FromUserName><![CDATA[oJj6WwDgpSDqyPj9hVHMDv8olThc]]></FromUserName>
<CreateTime>1523093975</CreateTime>
<MsgType><![CDATA[event]]></MsgType>
<Event><![CDATA[subscribe]]></Event>
<EventKey><![CDATA[]]></EventKey>
</xml>
```

图 3-7 关注事件

当收到新关注时，系统将发送系统的使用方法给新用户，并记录用户信息；当收到

取消关注通知时，系统将对用户的信息做一些清理工作。

3.2.6 web 服务器

微信公众号开发者配置成功之后，搭建后台服务器。数据库和推荐程序都放在同一台机器上，服务监听 80 端口等待微信公众号服务器发来 POST 请求，然后操作数据库做相应操作后返回结果。

如本章图 3-1 所示架构，我们只需要搭建一个 web 服务器，监听 80 端口，当有用户关注微信公众号、发送文字，发送图片等操作时，微信公众号服务器就会将这些操作封装成 XML 格式的数据发送到我们自己搭建的 web 服务器，web 服务器处理后返回一段 XML 格式的内容给微信公众号服务器，微信公众号服务器再将此回复转成相应的文字或者图片等内容发送给用户。在这个过程中，发送请求的是用户，处理请求的是 web 服务器，微信公众号服务器只是做了一个转发的功能。而数据库和协同过滤算法就放在 web 后台服务器上面，当用户发来请求时作相应的处理后返回信息给用户。

web.py 是一个开源的轻量级 python web 框架[6]。使用 python 就能进行简单的开发，完成这个系统中需要的功能，监听 80 端口，与微信公众号服务器进行简单的交互，及收发 XML 数据。

安装 web.py：

（1） web.py 官网（http://webpy.org/）下载压缩包到本地解压。

（2） 进入解压后的文件夹执行 python setup.py install 安装。

安装完成后，在 python 脚本中使用 import web 语句就能引入 web.py 框架。web.py 是一个框架，可以将它看成一个 web 服务器，在引入它之后，只需要专心编写与微信公众号服务器交互的代码，不需要关心消息如何经过 TCP/IP 协议栈、如何到达对端。

只需要简单的几行代码就能实现一个 web 服务器，如图 3-8，在 urls 中指定 web 的默认访问路径为根目录，指定自定义类型 Main 来处理 HTTP 请求。在 Main 中定义两种请求方式的回调函数以及返回内容。最后运行 web.application()和 run()方法。

```
#!/bin/python
#-*- coding: utf-8 -*-
import web

urls = (
    '/', 'Main',
)
class Main(object):
    def GET(self):
        return "You are welcome."
    def POST(self):
        return "You are welcome."

if __name__ == '__main__':
    app = web.application(urls, globals())
    app.run()
```

图 3-8 使用 web.py

编写完成后保存为 main.py 文件，增加可执行权限。并在命令行运行：

./main.py 80

因为微信公众号服务器只会连接 web 服务器的 80 端口，所以必须监听 80 端口微信公众号服务器才能连接成功，另外 80 端口需要拥有操作系统管理员权限才能监听。

然后在浏览器输入主机 IP 地址或者域名地址，便能得到 web 后台返回的信息，如图 3-9 的例子，web 后台主机 IP 地址为 211.159.176.139。

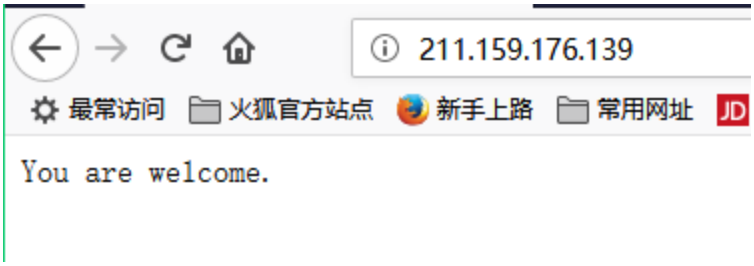


图 3-9 浏览器访问 web 后台

相应地，在 web 后端也能看到请求处理日志，如图 3-10。

```
[chj@VM_69_249_centos web_server]$ sudo ./webpytest.py 80
http://0.0.0.0:80/
60.176.29.94:60059 - - [07/Apr/2018 13:50:24] "HTTP/1.1 GET /" - 200 OK
```

图 3-10 web 后端日志

3.3 数据库设计

3.3.1 电影信息表

```
mysql> desc douban_movie;
+---------------+---------------+------+-----+---------+----------------+
| Field         | Type          | Null | Key | Default | Extra          |
+---------------+---------------+------+-----+---------+----------------+
| id            | int(11)       | NO   | PRI | NULL    | auto_increment |
| title         | varchar(100)  | NO   |     | NULL    |                |
| score         | float         | YES  |     | NULL    |                |
| num           | int(11)       | YES  |     | NULL    |                |
| link          | varchar(200)  | NO   |     | NULL    |                |
| time          | date          | YES  |     | NULL    |                |
| address       | varchar(50)   | YES  |     | NULL    |                |
| other_release | varchar(100)  | YES  |     | NULL    |                |
| actors        | varchar(1000) | YES  |     | NULL    |                |
+---------------+---------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

图 3-11 电影信息表

图 3-11 是一张存放电影信息的 MySQL 表格，每一条记录是一场电影的信息，每一场电影有一个唯一的 ID 字段用来在系统中标识唯一一场电影，因为电影名相同的电影可能不是同一场电影，如图 3-12。

```
mysql> select * from douban_movie where title="西游记";
+------+--------+-------+------+-----------------------------------------+------------+---------+---------------+-----------------------------------------------------------------------+
| id   | title  | score | num  | link                                    | time       | address | other_release | actors                                                                |
+------+--------+-------+------+-----------------------------------------+------------+---------+---------------+-----------------------------------------------------------------------+
|  8297 | 西游记 |   7.4 | 12067 | https://movie.douban.com/subject/3021640/ | 1996-11-18 | 香港    | NULL          | 江华 张卫健 黎耀祥 麦长青 香港                                                |
| 12501 | 西游记 |   5.1 |  1401 | https://movie.douban.com/subject/1979059/ | 2007-07-14 | 日本    | NULL          | 香取慎吾 深津绘里 内村光良 伊藤淳史 鹿賀丈史                                 |
| 13226 | 西游记 |   8.7 | 17637 | https://movie.douban.com/subject/2156775/ | 1999-07-23 | 中国大陆 | NULL          | 沈殊课 苏楠 程玉珠 刘风 张涵予                                               |
| 17295 | 西游记 |   9.5 | 88943 | https://movie.douban.com/subject/2156663/ | 1986-02-00 | 中国大陆 | NULL          | 六小龄童 迟重瑞 马德华 迟亮 徐少华                                           |
| 41537 | 西游记 |   6.3 |  6161 | https://movie.douban.com/subject/4058786/ | 2012-01-30 | 中国大陆 | NULL          | 吴樾 聂远 臧金生 徐锦江 王九胜                                               |
| 45352 | 西游记 |   7.8 |   236 | https://movie.douban.com/subject/5975033/ | NULL       | NULL    | 中国大陆; 20分钟 | 赛赛四 动画 奇幻 冒险                                                        |
+------+--------+-------+------+-----------------------------------------+------------+---------+---------------+-----------------------------------------------------------------------+
6 rows in set (0.00 sec)
```

图 3-12 重名电影

《西游记》就有很多个版本。为了区分开这些重名电影，只能使用电影详情页 URL 来作为关键字，但是 URL 太长了，不方便程序使用，所以使用 ID 值标识每一部电影。

3.3.2 其他表结构

具体的操作还需要依靠数据库来达到大量数据的长期存储。数据库中，除了电影信息表，还需要增加关于用户的表格。当新的用户添加关注时，需要将用户的信息存入数

据库的用户表；当用户评价电影时，需要更新该用户的喜好信息。

针对用户的“搜索”、“评价”、“推荐”三个接口，新增加三个表：用户信息表、用户搜索记录表、用户评分表。在收到新用户关注事件，将用户的信息保存到用户信息表，并赋唯一 ID 标识每个用户，如图 3-13。当用户搜索电影时，保存用户的搜索记录。当用户评价电影时，保存用户的评价记录。保存这些记录都是为“推荐”功能提供数据支持。

三个新表的结构非常简单，如图 3-13，每个表有 3 个字段。

```
mysql> desc user_info;
+------------+------------------+------+-----+---------+----------------+
| Field      | Type             | Null | Key | Default | Extra          |
+------------+------------------+------+-----+---------+----------------+
| id         | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| wx_id      | varchar(100)     | NO   | UNI | NULL    |                |
| start_time | varchar(100)     | YES  |     | NULL    |                |
+------------+------------------+------+-----+---------+----------------+
3 rows in set (0.02 sec)

mysql> desc seek_movie;;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| user_id   | int(20) | NO   |     | NULL    |       |
| movie_id  | int(20) | NO   |     | NULL    |       |
| seek_time | int(20) | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)

ERROR:
No query specified

mysql> desc like_movie;
+----------+---------+------+-----+---------+-------+
| Field    | Type    | Null | Key | Default | Extra |
+----------+---------+------+-----+---------+-------+
| user_id  | int(20) | NO   | PRI | NULL    |       |
| movie_id | int(20) | NO   | PRI | NULL    |       |
| liking   | float   | NO   |     | NULL    |       |
+----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

图 3-13 用户相关表

3.4 推荐算法设计

如果有如图 3-14 中的用户评分数据，针对每一个用户的历史数据，可以为每一个用户绘制一条二维曲线，横坐标为不同的电影，纵坐标为每一场电影的评分，如图 3-15。

	电影A	电影B	电影C	电影D	电影E
用户1	6.5	3	9	7	8
用户2	7	6	8	1	7
用户3	3.5	1	9	2	6.6
用户4	9	8	7	5.8	2
用户5	7.5	3	10	6	8

图 3-14 用户历史评分

从图中可以看出，如果是喜好相似的两个用户，他们的曲线起伏波动会是相近的，甚至可能是重叠的。仔细看这分数据能够看出，用户 1 和用户 5 的曲线是最相近的，所以可以将他们标记为有着共同爱好的用户，也就是互为邻居，当我们为一个用户找到一个或者多个邻居之后，就可以将这个用户的邻居喜欢的内容推荐给该用户。

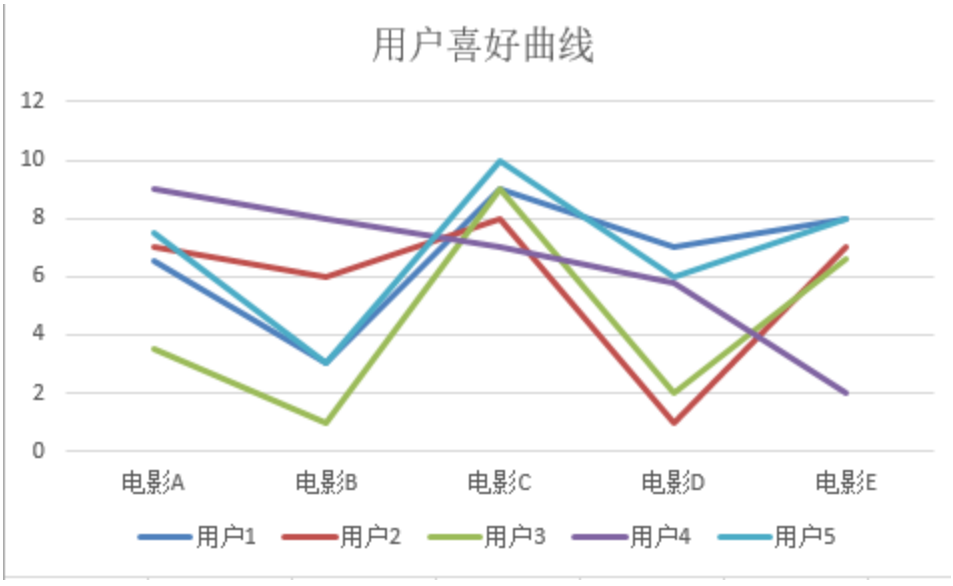


图 3-15 用户喜好曲线

本系统中的协同过滤算法使用这样的原理，不断为每一个用户找到喜好邻居，然后将邻居的喜好推荐给该用户。对于两条曲线，它们之间所围成的面积越小则越相近，也表示这两个用户有共同的兴趣爱好。

上述算法非常简单实用，但是数据库中的用户喜好数据往往是稀疏的，对于同一场电影不可能每个人都对它进行过评分。所以这个算法在满数据的情况下非常实用，但是在数据稀疏的情况下则不是那么有效。当然现实场景中也不需要数据全满，如果数据全满则意味着两个用户看过的电影是相同的，则无法推荐一些用户没有看过的电影给用户。

4. 系统实现

4.1 数据采集

豆瓣网上有大量的电影信息，每次听到电影名时，我总是会上豆瓣网查看电影的评分，然后决定自己要不要去观看。本文的数据来源也豆瓣电影。

在该推荐系统中，要做的就是将用户可能喜欢的电影信息反馈给用户，但是不包含电影本身。提供给用户的信息只能保证用户依靠该信息能够找到相应的电影资源，而系统也不提供下载资源的方法，只提供电影资源的名称和详情 URL 等。

4.1.1 爬虫概要

豆瓣电影的网址为：“https://movie.douban.com/tag/爱情?start=0&type=T”

我们只需要将以上 URL 中的“爱情”分别替换成“科幻”、“喜剧”、“励志”、“文艺”等词就能看到分好类的电影信息的第一页内容。

```
44 tag = urllib.request.quote(u'爱情')
45 url_1 = 'https://movie.douban.com/tag/{}?start=0&type=T'.format(tag)
```

图 4-1 目录页首页 URL

例如，图 4-1 中 url_1 的内容放在浏览器中就能访问到图 4-2 的页面。

图 4-2 目录页内容

而点击目录页首页页面最底下的页码就能翻页浏览该类型所有电影信息。图 4-2 滑到页面底部就是图 4-3 的内容。

图 4-3 目录页页底的页码

爬虫需要做的工作就是代替我们人工去翻页浏览所有电影信息，并将这些信息保存到数据库。我们的爬虫程序会依次访问每一个类型的电影列表，并从第一页翻到最后一页，把每页展示的电影信息存入数据库的表中。

这里我使用 python 的以下第三方库如图 4-4：urllib、requests、lxml、pymysql。

```
4 import urllib.request
5 import requests
6 from lxml import etree
7 import time
8 import random
9 import pymysql
```

图 4-4 python 爬虫使用的库

使用 time 和 random 库只是为了在爬取的每两个页面之间插入一个随机时间间隔。控制访问豆瓣服务器的速度，以防影响到豆瓣服务器正常运行被禁止访问。

核心的函数有以下几个：

使用 page = requests.get(URL)方法能够获取到 URL 指定的 HTML 网页文件存入 page 中；

使用 selector = lxml.etree.HTML(page)方法能够将名为 page 的 HTML 页面转换成一棵标签树存入 selector 中。

使用 text = selector.xpath(“xpath”)函数能够从标签树中提取出自己想得到的 HTML 标签内容（及电影的信息）到 text 中，xpath 为 xpath 表达式，类似于正则表达式，只是 xpath 更适用于提取标签文本（如 XML,HTML）中的内容；

pymysql 库则用来操作 mysql 数据库，建表，插入数据等操作；

urllib 用来格式化 URL。

有了以上这些方法可用，我们只需要编写代码使程序生成豆瓣电影的所有 URL，并依次下载这些页面，分析提取出其中我们需要的信息，就能获得所有的电影信息。图 4-5 的代码生成了所有类型的电影列表页首页 URL，并使用 xpath 语法提取出了每种电影类型的页码数。之后我们只需要依次去遍历这些 URL 并提取信息存入数据库就完成了电影数据的制备。

```
class theme_page(object):
    def __init__(self):
        self.tags = [u"爱情", u"喜剧", u"动画", u'剧情', u'科幻', u'动作', u'经典', u'悬疑', u'青春', u'犯罪', u'惊悚', \
            u"文艺", u"搞笑", u"纪录片", u'励志', u'恐怖', u'战争', u'短片', u'黑色幽默', u'魔幻', u'传记', \
            u'情色', u"感人", u"暴力", u'动画短片', u'家庭', u'音乐', u'童年', u'浪漫', u'黑帮', u'女性', \
            u'同志', u"史诗", u"童话", u'烂片', u'cult']

    def get_total_num(self):
        tags = self.tags
        total_num = []
        list = []
        for tag in tags:
            # print(tag)   #主题名称
            re_url = 'https://movie.douban.com/tag/{}?start=0&type=T'.format(urllib.request.quote(tag))
            # print(re_url)     # 主题链接
            s = requests.get(re_url)
            contents = s.content.decode('utf-8')
            selector = etree.HTML(contents)
            num = selector.xpath('//*[@id="content"]/div/div[1]/div[3]/a[10]/text()')
            total_num.append(int(num[0]))
            # print(num[0])    # 总页数
            list.append({               # list :{'爱情': '393'}, {'喜剧': '392'}, {'动画': '274'},
                '主题' : tag ,
                '总页数' : num[0]
            })
        return list
```

图 4-5 电影列表 URL 的生成

4.1.2 数据库结构

在 MySQL 中，需要建立一个数据库，然后在库中建表，分别用来存放我们的电影信息、用户信息、用户评价信息、用户搜索记录等等。目前我们只需要先建立一个存放电影信息的表，其他表在之后使用到时再建立。将每一条电影信息存为电影信息表中的一条记录。我创建了一个简单的电影信息表，一共有 6 个字段，分别存每场电影的名字、豆瓣评分、评价人数、电影详情页 URL、首映时间、演员表。

```
mysql> desc douban_mov;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| title  | varchar(100)  | NO   |     | NULL    |       |
| score  | varchar(10)   | YES  |     | NULL    |       |
| num    | varchar(100)  | YES  |     | NULL    |       |
| link   | varchar(200)  | YES  |     | NULL    |       |
| time   | varchar(100)  | YES  |     | NULL    |       |
| actors | varchar(1000) | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
6 rows in set (0.09 sec)
```

图 4-6 电影信息表

注意，在我们爬取电影信息时是不进入详情页的，只在列表页就能采到我们所需要的全部信息。这样可以减少我们访问的 URL 数量。并且为了简单，电影信息表的 6 个

字段都是使用 varchar 类型存储，这样爬下来的页面经过简单的提取就能放到数据库中，我们也不用在此做太多复杂的操作，以免一条数据出错导致整个爬虫停止工作。

4.1.3 爬虫主要逻辑

前面的准备工作做完了，我们就可以开始写爬虫了，首先我们遍历所有的 URL，在 4.1.1 中我们生成了各类电影的列表页首页 URL，以及每类电影的总页数。有了这些信息我们就可以推出所有列表页 URL。从浏览器中很容易看出每个列表页有 20 场电影信息（除每种类型最后一个列表页不足 20 页）。而且每两个相邻页的 URL 存在一个关系。例如爱情类电影的首个列表页 URL 是：

https://movie.douban.com/tag/%E7%88%B1%E6%83%85?start=0&type=T

第二页 URL 是：

https://movie.douban.com/tag/%E7%88%B1%E6%83%85?start=20&type=T

很明显的规律，每一页的 start 值会比上一页增加 20，不断增加 start 的值直到最后一页。

根据这个规律我们可以生成所有列表页的 URL，如图 4-7，获取所有类型的列表页 URL，并遍历所有页码。然后依次下载每个 URL 对应的 HTML 页面，存入 s 变量中。将 s 放入 run.search()方法中，在该方法中解析页面内容并存入数据库，该方法的内部实现将在后文中介绍。

```
#让程序run起来
random.seed(100)
page = theme_page()
page_list = page.get_total_num()
print(page_list)
run = douban()
for page_info in page_list:   # 遍历所有主题
    tags = page_info['主题']
    page = page_info['总页数']
    tag = urllib.request.quote(tags)
    threads = []
    for i in range(int(page)):    # 每个主题下的所有页数
        url = 'https://movie.douban.com/tag/{0}?start={1}&type=T'.format(tag, i * 20)
        s = requests.get(url)
        s = s.content.decode('utf-8')
        lists = run.search(s)
        sleep_time_mix = random.randint(30, 70)
        time.sleep(sleep_time_mix)
    sleep_time_max = random.randint(600, 3000)
    time.sleep(sleep_time_max)
```

图 4-7 爬虫主逻辑

在每次下载一个页面并分析完成入库之后，程序会睡眠一个随机时间。然后再循环下载下一个 URL 对应的页面。睡眠随机时间是为了控制程序访问豆瓣服务器的次数，以免给服务器造成负担。

4.1.4 页面分析

在 4.1.3 中遍历了所有列表页的 URL，并且使用 requests.get()方法获取到了每个页面的内容。在 run.search()中提取出我们所需要的 6 个字段，并存入数据库表中。在构造函数中连接数据库，如图 4-8。

```
47 class douban(object):
48     def __init__(self,*args,**kwargs):
49         self.conn = pymysql.connect(host='localhost', port=3306, user='root', password='186386', db='douban', charset='utf8')
50         self.cursor = self.conn.cursor()
51         self.sql_info = "INSERT IGNORE INTO `douban_mov` VALUES(%s,%s,%s,%s,%s,%s)"
52
```

图 4-8 连接数据库

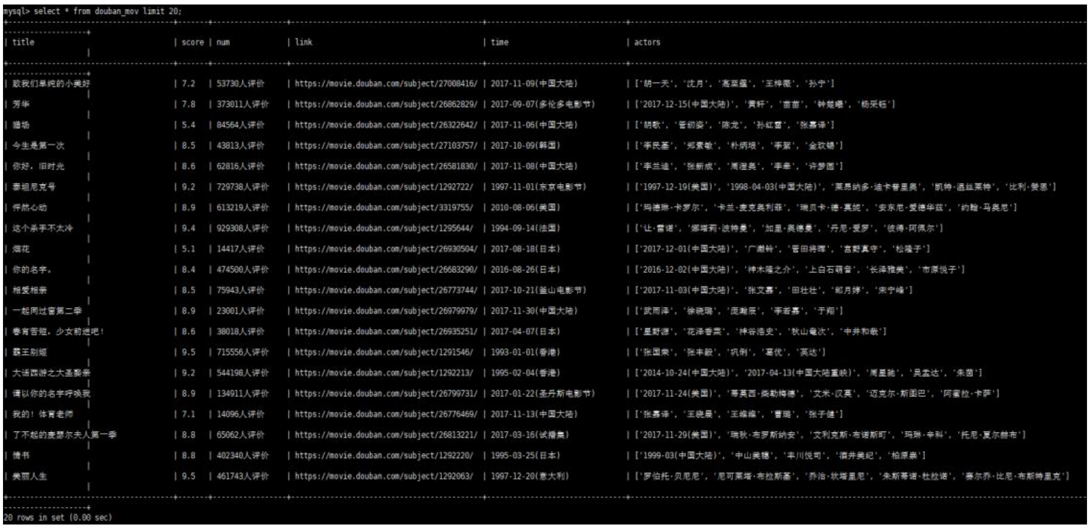
在 search 函数中，使用 xpath 选择器从页面中提取出我们需要的 6 个字段，并将这条电影记录插入到数据库。如图 4-9 所示。

```
53     def search(self,content):
54         '''
55         爬取页面内电影信息
56         '''
57         try:
58             selector = etree.HTML(content)
59             textslist = selector.xpath('//div[contains(@class,"grid-16-8 clearfix")]/div[1]/div[2]/table')
60         except Exception as e:
61             print(e)
62
63         try:
64             for text in textslist:
65                 lists = []
66                 title = text.xpath('tr/td[2]/div/a/text()')
67                 score = text.xpath('tr/td[2]/div/div/span[2]/text()')
68                 num = text.xpath('tr/td[2]/div/div/span[3]/text()')
69                 link = text.xpath('tr/td/a/@href')
70                 content = text.xpath('tr/td[2]/div/p/text()')
71
72                 if title:
73                     title = title[0].strip().replace('\n', "").replace('  ', "").replace('/', "") if title else ''
74                     score = score[0] if score else ''
75                     num = num[0].replace('(', "").replace(')', "") if num else ''
76                     link = link[0] if link else ''
77                     time = content[0].split(' / ')[0] if content else ''
78                     actors = content[0].split(' / ')[1:6] if content else ''
79                     lists.append({
80                         '电影名' : title,
81                         '评分' : score,
82                         '评价人数' : num,
83                         '详情链接' : link,
84                         '上映时间' : time,
85                         '主演' : actors
86                     })
87                     if lists:
88                         lists = lists.pop()
89                     else:
90                         lists = u''
91
92                     print(lists)
93                     try:
94                         self.cursor.execute(self.sql_info,(str(lists['电影名']),str(lists['评分']),str(lists['评价人数']),str(lists['详情链接']), str(lists['上映时间']),str(lists['主演'])))
95                         self.conn.commit()
96                     except Exception as e:
97                         print(e)
98         except Exception as e:
99             pass
100
```

图 4-9 提取电影信息入库

4.1.5 数据处理

爬虫运行后，数据库中的数据就会不断增加。在数据库中能够查看到新插入的数据，如图 4-10 所示。


mysql> select * from douban_mov limit 20;

title	score	num	link	time	actors
致我们单纯的小美好	7.2	53730人评价	https://movie.douban.com/subject/27008416/	2017-11-09(中国大陆)	['胡一天', '沈月', '高至霆', '王梓薇', '孙宁']
芳华	7.8	373011人评价	https://movie.douban.com/subject/26862829/	2017-09-07(多伦多电影节)	['2017-12-15(中国大陆)', '黄轩', '苗苗', '钟楚曦', '杨采钰']
猎场	5.4	84564人评价	https://movie.douban.com/subject/26322642/	2017-11-06(中国大陆)	['胡歌', '菅纫姿', '陈龙', '孙红雷', '张嘉译']
今生是第一次	8.5	43813人评价	https://movie.douban.com/subject/27103757/	2017-10-09(韩国)	['李民基', '郑素敏', '朴炳垠', '李絮', '金玟锡']
你好，旧时光	8.6	62816人评价	https://movie.douban.com/subject/26581830/	2017-11-08(中国大陆)	['李兰迪', '张新成', '周澄奥', '李拳', '许梦圆']
泰坦尼克号	9.2	729738人评价	https://movie.douban.com/subject/1292722/	1997-11-01(东京电影节)	['1997-12-19(美国)', '1998-04-03(中国大陆)', '莱昂纳多·迪卡普里奥', '凯特·温丝莱特', '比利·赞恩']
怦然心动	8.9	613219人评价	https://movie.douban.com/subject/3319755/	2010-08-06(美国)	['玛德琳·卡罗尔', '卡兰·麦克奥利菲', '瑞贝卡·德·莫妮', '安东尼·爱德华兹', '约翰·马奥尼']
这个杀手不太冷	9.4	929308人评价	https://movie.douban.com/subject/1295644/	1994-09-14(法国)	['让·雷诺', '娜塔莉·波特曼', '加里·奥德曼', '丹尼·爱罗', '彼得·阿佩尔']
烟花	5.1	14417人评价	https://movie.douban.com/subject/26930504/	2017-08-18(日本)	['2017-12-01(中国大陆)', '广濑铃', '菅田将晖', '宫野真守', '松隆子']
你的名字。	8.4	474500人评价	https://movie.douban.com/subject/26683290/	2016-08-26(日本)	['2016-12-02(中国大陆)', '神木隆之介', '上白石萌音', '长泽雅美', '市原悦子']
相爱相亲	8.5	75943人评价	https://movie.douban.com/subject/26773744/	2017-10-21(釜山电影节)	['2017-11-03(中国大陆)', '张艾嘉', '田壮壮', '郎月婷', '宋宁峰']
一起同过窗第二季	8.9	23001人评价	https://movie.douban.com/subject/26979979/	2017-11-30(中国大陆)	['武雨泽', '徐晓璐', '庞瀚辰', '李若嘉', '于翔']
春宵苦短，少女前进吧！	8.6	38018人评价	https://movie.douban.com/subject/26935251/	2017-04-07(日本)	['星野源', '花泽香菜', '神谷浩史', '秋山竜次', '中井和哉']
霸王别姬	9.5	715556人评价	https://movie.douban.com/subject/1291546/	1993-01-01(香港)	['张国荣', '张丰毅', '巩俐', '葛优', '英达']
大话西游之大圣娶亲	9.2	544198人评价	https://movie.douban.com/subject/1292213/	1995-02-04(香港)	['2014-10-24(中国大陆)', '2017-04-13(中国大陆重映)', '周星驰', '吴孟达', '朱茵']
请以你的名字呼唤我	8.9	134911人评价	https://movie.douban.com/subject/26799731/	2017-01-22(圣丹斯电影节)	['2017-11-24(美国)', '蒂莫西·柴勒梅德', '艾米·汉莫', '迈克尔·斯图巴', '阿蜜拉·卡萨']
我的！体育老师	7.1	14096人评价	https://movie.douban.com/subject/26776469/	2017-11-13(中国大陆)	['张嘉译', '王晓晨', '王维维', '曹璐', '张子健']
了不起的麦瑟尔夫人第一季	8.8	65062人评价	https://movie.douban.com/subject/26813221/	2017-03-16(试播集)	['2017-11-29(美国)', '瑞秋·布罗斯纳安', '艾利克斯·布诺斯町', '玛琳·辛科', '托尼·夏尔赫布']
情书	8.8	402340人评价	https://movie.douban.com/subject/1292220/	1995-03-25(日本)	['1999-03(中国大陆)', '中山美穗', '丰川悦司', '酒井美纪', '柏原崇']
美丽人生	9.5	461743人评价	https://movie.douban.com/subject/1292063/	1997-12-20(意大利)	['罗伯托·贝尼尼', '尼可莱塔·布拉斯基', '乔治·坎塔里尼', '朱斯蒂诺·杜拉诺', '赛尔乔·比尼·布斯特里克']

20 rows in set (0.00 sec)

图 4-10 爬虫爬取的数据

表中的 6 个字段全是简单的字符串类型，后续程序使用的表和这个表之间还有一些差异，所以还要做数据去重、类型转换、去除两端空格等处理，比较麻烦，为了降低程序的复杂性，这些工作将在程序之前完成。

这个步骤做法很多，可以编写一个简单的脚本进行处理。将这个表的电影信息导入到上一章设计的电影信息表中。

4.2 基础功能实现

4.2.1 后端主要逻辑

当后端服务收到公众号服务器以 POST 方式发来的请求时，判断消息的类型，调用相应的函数作相应的处理，如果出现异常，则返回 success 给公众号服务器，这样用户就不会显示服务器出故障的字样。如图 4-11，调用相应的函数处理相应类型的消息。

当回复时，只需要将发送方和接收方的 name 交换位置即可，如图 4-11 中的 222 和 223 行。

```
216     def POST(self):
217         try:
218             webData = web.data()
219             #print webData
220             content = ""
221             recMsg = receive.parse_xml(webData)
222             toUser = recMsg.FromUserName
223             fromUser = recMsg.ToUserName
224             if isinstance(recMsg, receive.Msg) and recMsg.MsgType == 'text':#主要业务逻辑
225                 content = self.on_text(recMsg)
226             elif isinstance(recMsg, receive.Msg) and recMsg.MsgType == 'image':
227                 content = self.on_image(recMsg)
228             elif isinstance(recMsg, receive.Msg) and recMsg.MsgType == 'event':
229                 content = self.on_event(recMsg)
230             else:
231                 print "暂不支持"
232                 return "success"
233 
234             if content == "":
235                 return "success"
236             replyMsg = reply.TextMsg(toUser, fromUser, content)
237             data = replyMsg.send()
238             return data
239         except Exception, Argment:
240             print "fail, but I pretend to be success."
241             return "success"
242 
```

图 4-11 POST 处理逻辑

在图 4-11 中，receive.parse_xml()方法负责解析收到的 XML 内容，reply.TextMsg()负责组装要响应的 XML 内容并又 send()方法返回最终 XML 内容。具体实现分别放在 receive.py 和 reply.py 文件中。只需要 import receive 和 import reply 就能引用发到这几个函数。具体封装如图 4-12 和图 4-13，图 4-12 中，调用 xml.etree.ElementTree.fromstring()解析 XML，然后使用 find()函数提取关键字段。

```python
#!/bin/python
#coding: utf-8
import xml.etree.ElementTree as ET
def parse_xml(web_data):
    if len(web_data) == 0:
        return None
    xmlData = ET.fromstring(web_data)
    msg_type = xmlData.find('MsgType').text
    if msg_type == 'text':
        return TextMsg(xmlData)
    elif msg_type == 'image':
        return ImageMsg(xmlData)
    elif msg_type == 'event':
        return EventMsg(xmlData)


class Msg(object):
    def __init__(self, xmlData):
        self.ToUserName = xmlData.find('ToUserName').text
        self.FromUserName = xmlData.find('FromUserName').text
        self.CreateTime = xmlData.find('CreateTime').text
        self.MsgType = xmlData.find('MsgType').text

class TextMsg(Msg):
    def __init__(self, xmlData):
        Msg.__init__(self, xmlData)
        self.Content = xmlData.find('Content').text.encode("utf-8")
        self.MsgId = xmlData.find('MsgId').text

class ImageMsg(Msg):
    def __init__(self, xmlData):
        Msg.__init__(self, xmlData)
        self.PicUrl = xmlData.find('PicUrl').text
        self.MediaId = xmlData.find('MediaId').text
        self.MsgId = xmlData.find('MsgId').text

class EventMsg(Msg):
    def __init__(self, xmlData):
        Msg.__init__(self, xmlData)
        self.Event = xmlData.find('Event').text
        self.EventKey = xmlData.find('EventKey').text
```

图 4-12 receive.py 解析 XML

图 4-13 中的组装模块，目前只用回复文字给用户，所以暂时只用得到 TextMsg 类型。

```python
#!/bin/python
#conding=utf-8
import time
class Msg(object):
    def __init__(self):
        pass
    def send(self):
        return "success"

class TextMsg(Msg):
    def __init__(self, toUserName, fromUserName, content):
        self.__dict = dict()
        self.__dict['ToUserName'] = toUserName
        self.__dict['FromUserName'] = fromUserName
        self.__dict['CreateTime'] = int(time.time())
        self.__dict['Content'] = content

    def send(self):
        XmlForm = """
        <xml>
        <ToUserName><![CDATA[{ToUserName}]]></ToUserName>
        <FromUserName><![CDATA[{FromUserName}]]></FromUserName>
        <CreateTime>{CreateTime}</CreateTime>
        <MsgType><![CDATA[text]]></MsgType>
        <Content><![CDATA[{Content}]]></Content>
        </xml>
        """
        return XmlForm.format(**self.__dict)

class ImageMsg(Msg):
    def __init__(self, toUserName, fromUserName, mediaId):
        self.__dict = dict()
        self.__dict['ToUserName'] = toUserName
        self.__dict['FromUserName'] = fromUserName
        self.__dict['CreateTime'] = int(time.time())
        self.__dict['MediaId'] = mediaId
    def send(self):
        XmlForm = """
        <xml>
        <ToUserName><![CDATA[{ToUserName}]]></ToUserName>
        <FromUserName><![CDATA[{FromUserName}]]></FromUserName>
        <CreateTime>{CreateTime}</CreateTime>
        <MsgType><![CDATA[{image}]]></MsgType>
        <Image>
        <MediaId><![CDATA[{MediaId}]]></MediaId>
        </Image>
        </xml>
        """
        return XmlForm.format(**self.__dict)
```

图 4-13 reply.py 组装 XML

图 4-14 中，分别编写了三个处理函数，对于图片消息，回复“暂时不支持”字样；对于事件消息，只关心新用户添加关注事件；对于文字消息，将它当命令解析。如果文字不是“评价”、“搜索”、“推荐”三个命令中的一个，那直接将所有文字当成电影名去

搜索数据库，然后返回搜索结果。

```
    def on_text(self, recMsg):
        content = ""
        user_name = recMsg.FromUserName
        recv_content = recMsg.Content
        self.update_user_info(user_name)#检查用户是否存在，不存在则创建

        recv_msg = self.parse_cmd(recv_content)#解析收到的命令
        exe_cmd = recv_msg[0]
        content = ""
        if exe_cmd == '评价':
            content = self.evaluate(user_name, recv_msg)
        elif exe_cmd == '推荐':
            content = self.recommend(user_name, recv_msg)
        elif exe_cmd == '搜索':
            content = self.search(user_name, recv_msg)
        else:
            content = self.browse(user_name, recv_msg)
        self.db.close()
        return content

    def on_image(self, recMsg):
        content = "感谢您的图片，但是我暂时不能识别图片。。。"
        return content

    def on_event(self, recMsg):
        #print recMsg.Event
        content = ""
        if recMsg.Event == "subscribe":
            db = pymysql.connect(host='localhost', port=3306, user='root', password='186386', db='douban', charset='utf8')
            cursor = db.cursor()
            cmd = 'select * from user_info where wx_id = "{}";'.format(recMsg.FromUserName)
            cursor.execute(cmd)
            results = cursor.fetchall()
            if len(results) == 0:
                cmd = 'insert into user_info(wx_id, start_time) values("{}", "{}");'.format(recMsg.FromUserName, int(time.time()))
                try:
                    cursor.execute(cmd)
                    db.commit()
                except:
                    db.rollback()
            db.close()
            content = "感谢您的关注与支持，评价电影超过一定次数后，本平台将为您提供个性化推荐服务，您可以发送以下内容给我：\n搜索 无问西东\n评价 赛时明月 8.9\n推荐\n"
            return content
        elif recMsg.Event == "unsubscribe":
            return content
        else:
            return content
```

图 4-14 后台处理函数

当用户添加关注时，服务需要在用户表中检查是否有该有用户的记录，没有则添加为新用户，为其分配 ID。然后需要将操作说明发送给用户，让新用户知道如何跟系统交互。测试结果如图 4-15，符合预期的回复。

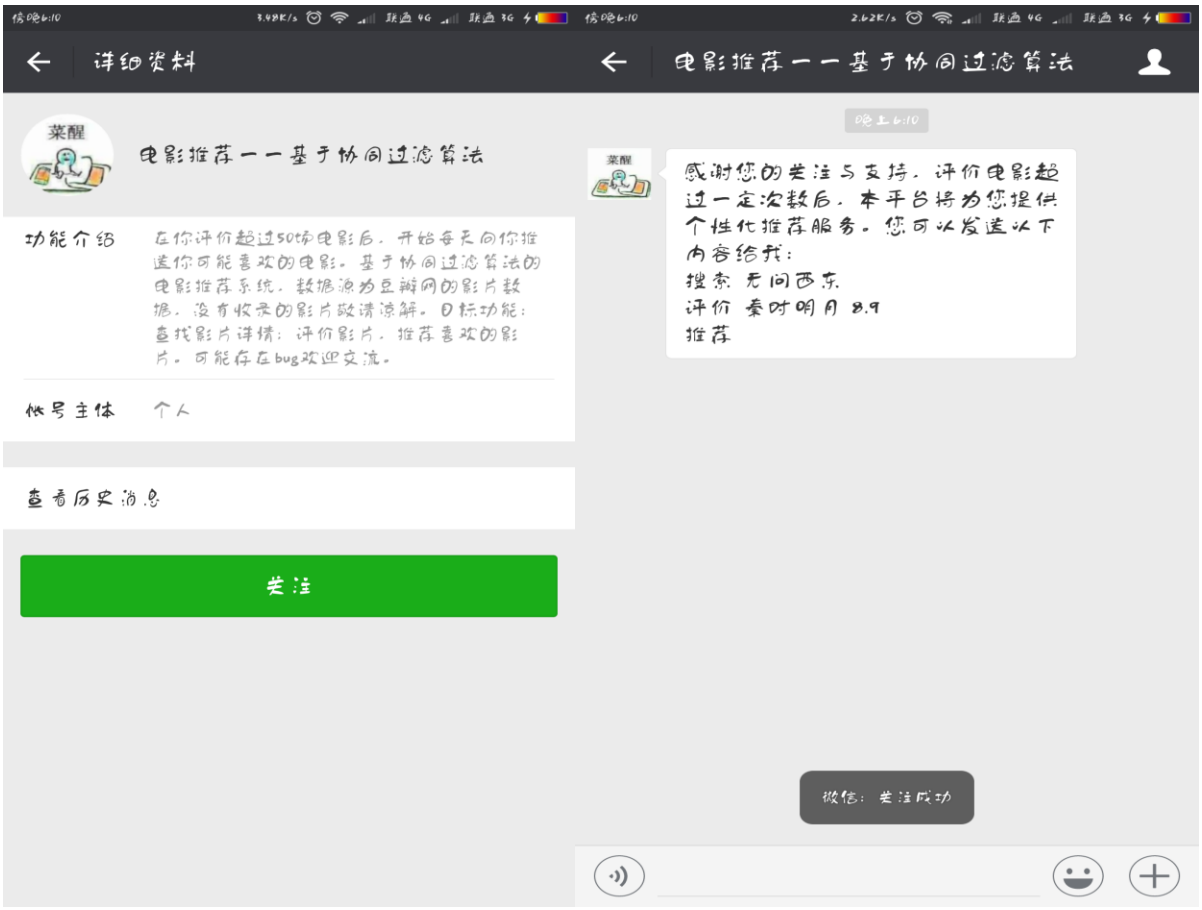


图 4-15 关注提示测试

然后查看数据库中的用户表，发现新用户添加成功。如图 4-16。

```
mysql> select * from user_info;
+----+----------------------------+------------+
| id | wx_id                      | start_time |
+----+----------------------------+------------+
|  3 | oJj6WwDgpSDqyPj9hVHMDv8olThc | 1520249962 |
```

图 4-16 新用户信息

当用户取消关注时，暂时不删除用户的信息，以免该用户再次关注时数据被清空。后期可以修改为保存一段时间后删除，例如用户已经取消关注三年则删除该用户的信息和记录。目前用户量较少则取消关注时不清楚记录。当用户再次关注时检查数据库，发现已经拥有信息则不会重复添加。

4.2.2 搜索功能

接下来编写系统的处理逻辑。当收到公众号服务器以 POST 发来的请求时，依次判断消息类型、消息内容，如果是“搜索”或者是“电影名”，则调用搜索电影功能。如图 4-11 中的 on_search()、on_browse()实现搜索功能，其实 on_search()内部只是简单调用 on_browse()函数，所以搜索功能其实是靠 on_browse()函数实现。如图 4-17。

```
    def search(self, user_name, recv_msg):
        return self.browse(user_name, recv_msg[1:])
    def browse(self, user_name, recv_msg):
        movie_name = recv_msg[0]
        content = ""
        cmd = 'select * from douban_movie where title like "{}";'.format(movie_name)#精准查找
        self.cursor.execute(cmd)
        results = self.cursor.fetchall()
        #print "select "{}" have results num:{}.'.format(movie_name, len(results))
        if len(results):
            for row in results:
                title = row[1].encode("utf-8") if row[1] != None else ""
                score = row[2] if row[2] != None else 0
                num = row[3] if row[3] != None else 0
                link = row[4].encode("utf-8") if row[4] != None else ""
                date_time = row[5] if row[5] != None else ""
                address = row[6].encode("utf-8") if row[6] != None else ""
                other_address = row[7].encode("utf-8") if row[7] != None else ""
                actors = row[8].encode("utf-8") if row[8] != None else ""
                if score:
                    content += '{}\n{}\n{}\n{}\n{}\n评价人数:{}\n评分:{}\n{}\n\n'.format(title, date_time, address, other_address, actors, num, score, link)
                else:
                    content += '{}\n{}\n{}\n{}\n{}\n评价人数:{}\n{}\n\n'.format(title, date_time, address, other_address, actors, num, link)
            cmd = 'select id from user_info where wx_id = "{}";'.format(user_name)#更新查找记录
            self.cursor.execute(cmd)
            results = self.cursor.fetchall()
            user_id = results[0][0]
            cmd = 'select id from douban_movie where title = "{}";'.format(movie_name)
            self.cursor.execute(cmd)
            results = self.cursor.fetchall()
            for row in results:
                movie_id = row[0]
                cmd = 'insert into seek_movie(user_id, movie_id, seek_time) values({}, {}, {})'.format(user_id, movie_id, int(time.time()))
                try:
                    self.cursor.execute(cmd)
                    self.db.commit()
                except:
                    self.db.rollback()
```

图 4-17 精准搜索

在查询数据库后，判断是否找到用户搜索的电影，如果搜索记录不为 0 条，则将电影信息返回给用户，并在搜索记录表内增加搜索记录。如果搜索结果为 0 条记录，则可能是用户输入有误，使用 MySQL 的 like 语句模糊查找，并返回查找结果，如果还没有找到，则返回“该电影不存在”字样，如图 4-18。

```
    def browse(self, user_name, recv_msg):
        movie_name = recv_msg[0]
        content = ""
        cmd = 'select * from douban_movie where title like "{}";'.format(movie_name)#精准查找
        self.cursor.execute(cmd)
        results = self.cursor.fetchall()
        #print 'select "{}" have results num:{}.'.format(movie_name, len(results))
        if len(results):
            pass
        else:#模糊查找
            cmd = 'select * from douban_movie where title like "%{}%" limit 5;'.format(movie_name)
            self.cursor.execute(cmd)
            results = self.cursor.fetchall()
            print 'like "{}" have results num:{}.'.format(movie_name, len(results))
            if len(results) == 0:
                content = "抱歉，暂时没有收录该影片。"#找不到
            else:
                content = "您在找的可能是：\n"#模糊匹配到
                for row in results:
                    title = row[1].encode("utf-8") if row[1] != None else ""
                    score = row[2] if row[2] != None else 0
                    num = row[3] if row[3] != None else 0
                    link = row[4].encode("utf-8") if row[4] != None else ""
                    date_time = row[5] if row[5] != None else ""
                    address = row[6].encode("utf-8") if row[6] != None else ""
                    other_address = row[7].encode("utf-8") if row[7] != None else ""
                    actors = row[8].encode("utf-8") if row[8] != None else ""
                    content += "{}\n{}\n{}\n{}\n{}\n评价人数:{}\n评分:{}\n{}\n\n".format(title, date_time, address, other_address, actors, num, score, link)
        return content
```

图 4-18 模糊搜索

当用户发出搜索请求时，相应的，服务会精准查找数据库中的电影，如果查找到相关电影，则更新查找记录表。如果没有找到，则使用模糊查找并返回结果，不更新查找记录表。如果都没找到，则返回“没有该电影”字样。测试结果如图 4-19。


电影推荐——基于协同过滤算法
天空之城
天空之城
1986-08-02
日本
1992(中国大陆)
日中真弓 横沢启子 初井言榮 寺田农
评价人数:360832
评分:9.0
https://movie.douban.com/subject/1291583/
天空之城
2016-06-05
英国
www.bbc.co.uk/programmes/b07g70jl
迪拉斯·坎贝尔 Hannah Fry 英国 天空之城
评价人数:227
评分:8.6
https://movie.douban.com/subject/26815105/
天空
您在找的可能是:
天空之城
1986-08-02
日本
1992(中国大陆)
日中真弓 横沢启子 初井言榮 寺田农
评价人数:360832
评分:9.0
https://movie.douban.com/subject/1291583/
香草的天空
2001-12-14
汤姆·克鲁斯 佩内洛普·克鲁兹 卡梅隆·迪亚兹 库尔特·拉塞尔 杰森·李
评价人数:38159
评分:7.4
https://movie.douban.com/subject/1299054/
不一样的天空
1993-12-17

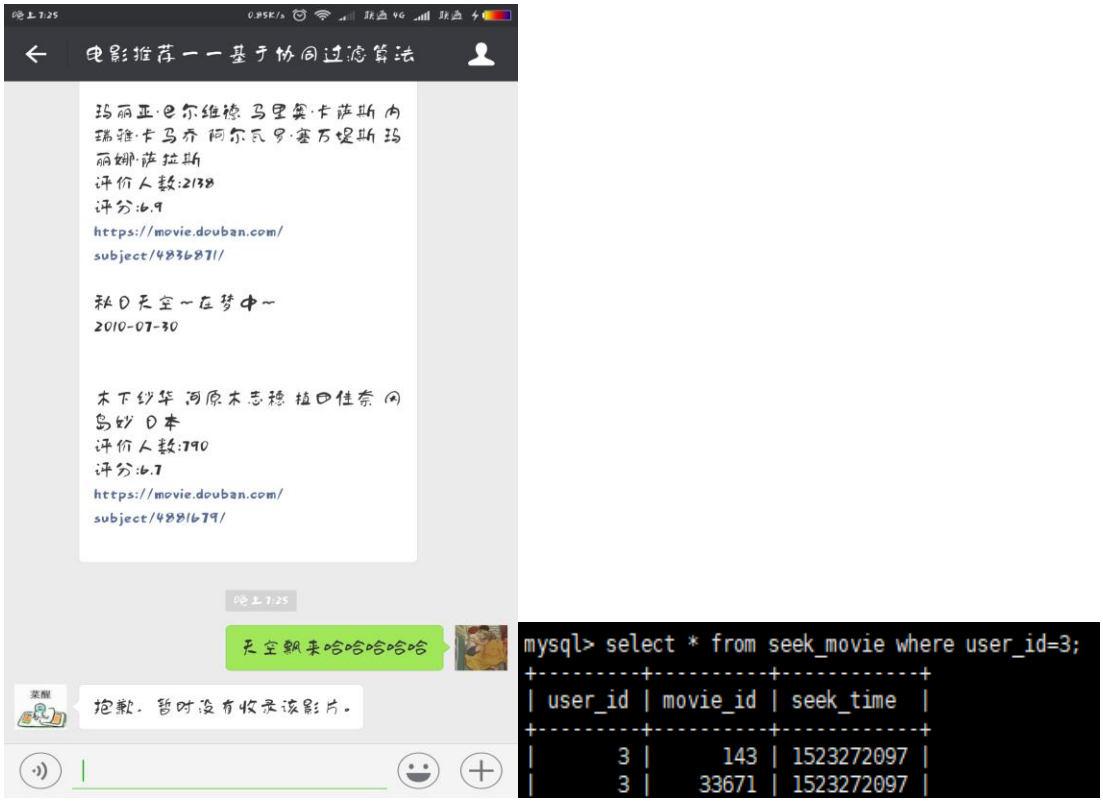

电影推荐——基于协同过滤算法
玛丽亚·B尔维德 马里奥·卡萨斯 内瑞雅·卡马乔 阿尔瓦罗·塞万提斯 玛丽娜·萨拉斯
评价人数:2138
评分:6.9
https://movie.douban.com/subject/4836871/
秋日天空～在梦中～
2010-07-30
木下纱华 河原木志穗 植田佳奈 冈岛钞 日本
评价人数:790
评分:6.7
https://movie.douban.com/subject/4881679/
天空飘来哈哈哈哈哈
抱歉，暂时没有收录该影片。
mysql> select * from seek_movie where user_id=3;
+---------+----------+-----------+
| user_id | movie_id | seek_time |
+---------+----------+-----------+
| 3 | 143 | 1523272097 |
| 3 | 33671 | 1523272097 |

图 4-19 搜索功能测试

图中发现精准搜索时出现了查询到多个结果的情况，查看数据库发现名字叫做《天空之城》的电影有两部，搜索记录也增加了两条。而用户可能想要的只是其中一部。这种情况是无法避免的，因为用户的输入只是一个简单的电影名，如果想更惊喜的搜索，可以让用户输入更多关于电影的信息，以筛选出用户想要的，但是这样使用户输入的复杂度增加，而得到的效果却很小，所以暂时不处理这个问题。

4.2.3 评价功能

为了获取用户对电影的喜好，本系统设置有评分功能，用户对于观看过的电影可以进行一个评价，对电影打一个 0-10 之间的评分。当用户发来评分命令时，系统解析命令后调用 evaluate 方法，记录用户对电影的评分，实现代码如图 4-20。

```
53    def parse_cmd(self, recv_content):
54        recv_msg_buf = recv_content.split(' ')#格式标准化
55        recv_msg = []
56        for buf in recv_msg_buf:
57            if buf != "":
58                recv_msg.append(buf.strip())
59        return recv_msg
60    def evaluate(self, user_name, recv_msg):
61        content = ""
62        movie_name = recv_msg[1]
63        #可能需要处理
64        nice = float(recv_msg[2])
65        if nice > 10:
66            nice = 10
67        elif nice < 0:
68            nice = 0
69        cmd = 'select id from user_info where wx_id = "{}";'.format(user_name)#记录到数据库
70        self.cursor.execute(cmd)
71        results = self.cursor.fetchall()
72        user_id = results[0][0]
73        cmd = 'select id from douban_movie where title = "{}";'.format(movie_name)
74        self.cursor.execute(cmd)
75        results = self.cursor.fetchall()
76        if len(results) == 0:
77            content = "抱歉，电影名输入有误，请重新输入。"
78        else:
79
80            for row in results:
81                movie_id = row[0]
82                cmd = 'select liking from like_movie where user_id={} and movie_id={};'.format(user_id, movie_id)
83                self.cursor.execute(cmd)
84                results = self.cursor.fetchall()
85                if len(results) == 0:
86                    cmd = 'insert into like_movie(user_id, movie_id, liking) values({}, {}, {});'.format(user_id, movie_id, nice)
87                    try:
88                        self.cursor.execute(cmd)
89                        self.db.commit()
90                        content = '评价成功，感谢您的支持。{}:{}分'.format(movie_name, nice)
91                    except:
92                        self.db.rollback()
93                        content = '评价失败，请重新输入。{}:{}分'.format(movie_name, nice)
94                else:
95                    cmd = 'update like_movie set liking={} where user_id={} and movie_id={};'.format(nice, user_id, movie_id)
96                    try:
97                        self.cursor.execute(cmd)
98                        self.db.commit()
99                        content = '更新评分成功。{}:{}分'.format(movie_name, nice)
100                   except:
101                       self.db.rollback()
102                       content = '评价失败，请重新输入。{}:{}分'.format(movie_name, nice)
103       return content
```

图 4-20 评价功能

用户评价电影时，输入评价电影名和分数，对电影评价，这样重名的电影就会被同时评分，但是这可能不是用户想要的结果，用户可能只针对其中的一部电影评分。这种

情况我们是有必要避免的，但是代价仍然是增加复杂度，在用户评价时，告诉用户该电影名对应多部电影，然后让用户再在其中选择一次，这样的复杂度是可以接受的，后续可以增加此处理逻辑，但目前如果用户对一个电影名评分，则相同电影名的电影都会被一同评分。如图 4-21，评价电影时，用户输入的电影名不对不会进行模糊查找，只有在数据库中精准找到的电影名才会评价成功。


```
mysql> select * from like_movie;
+---------+----------+--------+
| user_id | movie_id | liking |
+---------+----------+--------+
|       3 |      143 |    8.5 |
|       3 |      157 |    8.5 |
```

图 4-21 评价功能测试

图中将两部名叫《天空之城》的电影都评 8.5 分，用户可能只想评论其中一部。这里后续可以进行改进完善。

4.3 协同过滤推荐实现

有了第二章的数据和交互系统，以及第三章对于协同过滤算法的研究。本章将在交互系统中实现推荐功能。由于微信订阅号没有主动发消息的接口，所以使用被动推送，当用户发送“推荐”命令时，系统返回个性化推荐结果。

当收到用户的“推荐”命令时，web.py 后台服务将调用相应的 recommend 函数，该函数会完成整个结果计算，最后返回推荐的内容。

4.3.1 读取用户曲线

根据上一章讨论的协同过滤算法，系统使用基于用户的协同过滤算法，首先要做的就是绘制如图 3-15 中的用户评分曲线。根据用户名，可以从数据库中找到该用户 ID，再根据用户 ID 找到该用户评价过的所有电影，有了该用户对的评分数据，则可以绘制用户评分曲线。如图 4-22，通过两次查找数据库，找到目标用户的评分数据，然后将评分数据存入一个字典中，字典的 key 为电影 ID，value 为该用户对该电影的评分。由于字典的 key 和 value 一一对应组成二维对应关系，可以将 key 看成用户评分曲线的横坐标，及自变量；将 value 看成纵坐标，及因变量，返回在 0 到 10 之内变动；将这个字典看做目标用户评价曲线。

```
104     def recommend(self, user_name, recv_msg):
105         content = ""
106         cmd = 'select id from user_info where wx_id="{}";'.format(user_name)
107         self.cursor.execute(cmd)
108         results = self.cursor.fetchall()
109         if len(results) != 1:
110             return content
111         user_id = results[0][0]
112         cmd = 'select * from like_movie where user_id={};'.format(user_id)
113         self.cursor.execute(cmd)
114         results = self.cursor.fetchall()
115         line = {}
116         if len(results):
117             for row in results:
118                 movie_id = row[1]
119                 score = row[2] if row[2] != None else -1
120                 line[movie_id] = score
121
```

图 4-22 目标用户评分数据

以同样的方法，绘制其余每一个用户的评价曲线，并与目标用户对比计算出相似程

度。如图 4-23，调用 compute 函数计算两个用户的相似程度，然后将结果全部存于字典 areas 中，areas 的 key 为其他用户的 ID，值为相似程度。

```
122         cmd = 'select id from user_info where wx_id<>"{}";'.format(user_name)
123         self.cursor.execute(cmd)
124         results = self.cursor.fetchall()
125         areas = {}
126         for other_user in results:#遍历每一个用户
127             cmd = 'select * from like_movie where user_id={};'.format(other_user[0])
128             self.cursor.execute(cmd)
129             results = self.cursor.fetchall()
130             line_other = {}
131             if len(results):
132                 for row in results:
133                     movie_id = row[1]
134                     score = row[2] if row[2] != None else -1
135                     line_other[movie_id] = score
136             tup = self.compute(line, line_other)
137             areas[other_user[0]] = tup
```

图 4-23 遍历所有用户评分数据

4.3.2 用户相似程度

compute 函数接受两个用户的评分数据，返回两个用户的相似程度。只有当两个用户都评论了同一场电影时，才能分析出两个用户的兴趣差值，如果两个用户评价的电影场次是完全不相同的，则认为这两个用户没有共同的兴趣爱好[7]。只有当两个用户评论过相同的电影时，才会计算两个用户的相似程度。同理，连个用户评价的电影场次相同的数量越多，则他们的关注点越相似，则认为他们的相似程度越大；而评论的电影场次差异越大，则关注点不一致，认为他们相似程度越小。

具体算法实现如图 4-24，area 表示两个用户的差异值，该值越大则说明用户差异越大，该值越小则说明用户相似程度越大。但是该值一定是一个非负数。只有当两个用户评价的电影场次完全没有交集时，认为这两个用户的差异无穷大，使用-1 表示无穷大的差异，这是由于数据稀疏造成的。

```
175     def compute(self, line, line_other):
176         area=0.0
177         common = dict.fromkeys([x for x in line if x in line_other])
178         common_num = len(common)
179         x = len(line) - common_num
180         y = len(line_other) - common_num
181         if 0 == common_num:
182             area = -1
183         else:
184             for key in common:
185                 area += (line[key] - line_other[key])**2
186             area = (area*x)/(common_num**2)
187         return (area, y)#area越小越相近,-1为无交集,y为可推荐的数量
```

图 4-24 计算相似程度

对于两个用户评价了同一场电影，则计算他们对这场电影评分的差值的平方记为△。例如，A 用户评价了 10 场电影，B 用户评价了 15 场电影，这些评价中有 7 部电影场次是相同的。则计算这 7 部电影的评分差值的平方△1、△2、△3、△4、△5、△6、△7、。然后取这些值的平均值△arg。

计算出△arg 之后，考虑到 A 和 B 用户一共评价了 7 部相同场次的电影，使 area = △arg/7。通过这个方法来降低△arg 的值，使系统体现出两个用户评价的电影相同场次越多，关注点越相近，相似度越高，差异值越小。

最后，考虑到用户 A 有 3 部电影 B 没有评价，用户 B 有 8 部电影 A 没有评价过。所以如果 A 是个性化推荐的目标用户，则 areaA = area*3；如果 B 是目标用户，则 areaB = area*8。通过这样的方法使差异值增大，使系统体现出两个用户评价过的不相同场次的电影数量越多，关注点越不同，相似度越低，差异值越大。通过对用户曲线差距的平均值△arg 除以一个数再乘以一个数，达到适当调整的效果，解决由数据稀疏造成的问题。

最后整理后得出一个公式放到 compute 函数中作为核心算法。而对于目标用户 A，可推荐的电影就是 B 评价的而 A 未评价的 8 场电影中评分高的。对于 B 用户则相反。

4.3.3 获得最终结果

将目标用户与每个用户对比后，将从这些用户中找出差异值最小的用户，将他视为目标用户的邻居。如果没有找到邻居，则告诉用户没有找到适合的推荐结果。如图 4-25。

```
        neighbor_id = -1
        for (key, val) in areas.items():
            if val[0] == -1:
                del areas[key]
            elif neighbor_id == -1 and val[1] != 0:#首次赋值
                neighbor_id = key
            elif neighbor_id != -1 and val[1] != 0 and val[0] < areas[neighbor_id][0]:#更新
                neighbor_id = key
        if neighbor_id == -1:
            return "抱歉，由于您的评价次数过少，系统暂时推算不出您的兴趣爱好。\n请多多评价，过段时间再试。"
```

图 4-25 寻找邻居用户

找到邻居用户的用户 ID 后，查询数据库得到能够作为推荐的电影 ID 集合。如图 4-26。邻居用户评价过，自己却没有评价过的电影则认为是推荐电影集。

```
        cmd = 'select * from like_movie where user_id={};'.format(neighbor_id)
        self.cursor.execute(cmd)
        results = self.cursor.fetchall()
        movies_id = []
        for row in results:
            if not line.has_key(row[1]):
                movies_id.append(row[1])
        print movies_id
```

图 4-26 寻找推荐电影 ID 集合

最后，根据这些电影 ID，从中找出要推荐的电影信息。而本系统中没有挑选电影，而是将集合中所有电影推荐给用户，如图 4-27。

```
        print movies_id
        if not len(movies_id):
            return "抱歉，由于您的评价次数过少，系统暂时推算不出您的兴趣爱好。\n请多多评价，过段时间再试。"
        for mov_id in movies_id:
            cmd = 'select * from douban_movie where id={};'.format(mov_id)
            self.cursor.execute(cmd)
            result = self.cursor.fetchone()
            title = result[1].encode("utf-8") if result[1] != None else ""
            score = result[2] if result[2] != None else 0
            num = result[3] if result[3] != None else 0
            link = result[4].encode("utf-8") if result[4] != None else ""
            date_time = result[5] if result[5] != None else ""
            address = result[6].encode("utf-8") if result[6] != None else ""
            other_address = result[7].encode("utf-8") if result[7] != None else ""
            actors = result[8].encode("utf-8") if result[8] != None else ""
            if score:
                content += '{}\n{}\n{}\n{}\n{}\n评价人数:{}\n评分:{}\n{}\n\n'.format(title, date_time, address, other_address, actors, num, score, link)
            else:
                content += '{}\n{}\n{}\n{}\n{}\n评价人数:{}\n{}\n\n'.format(title, date_time, address, other_address, actors, num, link)
        return content
```

图 4-27 产生最终推荐内容

这里其实可以做进一步挑选。根据两个用户的相似度和可推荐的集合中电影数量，如果用户差异小（可能差异值在 10 以内），说明有共同爱好，则可以挑选出邻居评分最高的几部电影推荐给目标用户。而如果用户差异值大（例如在 1000 以上），则说明两个用户关注点相同，但是评分相反，则可以将邻居的低评分电影推荐给目标用户。也可以

采用其他的推荐方案。

4.4 解决冷启动问题

在系统中没有评价数据时，协同过滤推荐算法完全做不出推荐。每次寻找邻居都没有数据依据，找不到任何邻居。系统会返回“请评价后再试”字样。在这个尴尬的阶段，需要鼓励用户进行大量评价来填充数据库。另外，在本系统中，先向用户开放了搜索和评价功能一段时候后，才开放推荐功能，所以在系统初期并没有使用推荐算法。

4.4.1 解决方案

在系统中完全没有用户评分时，本系统中每场电影数据都带有一个豆瓣评分。当没有为目标用户找到适合的推荐时，可以先推荐豆瓣评分较高的电影给用户，这是基于大众喜好向用户推荐的结果，可以用来弥补系统初期的冷启动问题。

在代码实现中，只需要在协同过滤算法没有找到适合推荐时调用基于内容的推荐算法即可。并且在返回结果给用户时鼓励用户多作评论以提高推荐质量。

如图 4-28，将原来的返回字样替换成基于内容的推荐算法。

```
148         if neighbor_id == -1:
149             return self.will(line)
150             #return "抱歉，由于您的评价次数过少，系统暂时推算不出您的兴趣爱好。\n请多多评价，过段时间再试。"
151         cmd = 'select * from like_movie where user_id={};'.format(neighbor_id)
152         self.cursor.execute(cmd)
153         results = self.cursor.fetchall()
154         movies_id = []
155         for row in results:
156             if not line.has_key(row[1]):
157                 movies_id.append(row[1])
158         if not len(movies_id):
159             return self.will(line)
160             #return "抱歉，由于您的评价次数过少，系统暂时推算不出您的兴趣爱好。\n请多多评价，过段时间再试。"
```

图 4-28 替换基于内容推荐

4.4.2 代码实现

如图 4-29，添加 will 函数，在协同过滤算法没有找到适合推荐时使用基于内容的推荐随机推荐一部电影给用户。

```
def will(self, line):
    cmd = 'select count(id) from douban_movie;'
    self.cursor.execute(cmd)
    result = self.cursor.fetchone()
    movie_sum = result[0]
    movie_sum -= 100
    rand = random.randint(0, movie_sum)
    cmd = 'select * from douban_movie limit {},100;'.format(rand)
    self.cursor.execute(cmd)
    results = self.cursor.fetchall()
    max_score_addr = 0
    max_score = 0.0
    index = 0
    for row in results:
        if row[8] != None and row[8] > max_score and not line.has_key(row[0]):
            max_score_addr = index
            max_score = row[8]
        index += 1
    title = results[max_score_addr][1].encode("utf-8") if results[max_score_addr][1] != None else ""
    score = results[max_score_addr][8] if results[max_score_addr][8] != None else 0
    num = results[max_score_addr][9] if results[max_score_addr][9] != None else 0
    link = results[max_score_addr][4].encode("utf-8") if results[max_score_addr][4] != None else ""
    date_time = results[max_score_addr][5] if results[max_score_addr][5] != None else ""
    address = results[max_score_addr][6].encode("utf-8") if results[max_score_addr][6] != None else ""
    other_address = results[max_score_addr][7].encode("utf-8") if results[max_score_addr][7] != None else ""
    actors = results[max_score_addr][3].encode("utf-8") if results[max_score_addr][3] != None else ""
    content = '{}\n{}\n{}\n{}\n{}\n评价人数:{}\n评分:{}\n{}\n为提高您的推荐质量，请您多使用评价功能。另外，您评价过的电影不会再次推荐给您。\n'.format(title, date_time, address, other_address, actors, num, score, link)
    return content
```

图 4-29 基于内容推荐算法

用户评价过的电影表明用户已经看过，传入目标用户的评分曲线给该函数，让该函数推荐时过滤掉用户已经评价过的电影。该算法会随机查找 100 场电影，选出评分最高的一场返回给用户。

4.4.3 其他完善方案

由于数据量较多，而用户评价数据稀疏，所以会导致协同过滤算法出现一系列问题。最明显的问题和冷启动一样，目标用户找不到邻居。这时可以使用基于内容等算法作出推荐。

而数据稀疏时，还可能出现找到了最佳邻居，但是最佳邻居的差距值仍然太大，这个时候把邻居的评价的高分电影推荐到目标用户不一定是最佳选择。也就是用户差距值范围不同，推荐的策略也应该做一定的调整。这个部分系统暂时没有实现。在将来的改进中可能会添加，并找到一个合适的算法。

5. 实验结果分析

5.1 推荐功能测试

推荐功能开发完成后，重启后台服务。在微信客户端进行测试功能正确性。

5.1.1 协同过滤推荐

分别让用户 ID 为 3 和 4 两个用户对系统中的电影做评价，评价场次有相同的，也有不同的，评分根据用户对电影的评价填写。然后两个用户分别发送推荐命令，查看后台日志如图 5-1 所示。

```
{3: (6.226666666666667, 8), 38: (8.0, 0)}
182.254.86.144:61518 - - [12/Apr/2018 13:51:20] "HTTP/1.1 POST /" - 200 OK
{4: (8.302222222222222, 6)}
182.254.86.155:50002 - - [12/Apr/2018 13:53:52] "HTTP/1.1 POST /" - 200 OK
```

图 5-1 后台计算邻居

图中显示两个用户的邻居 ID 分别为 4 和 3，也就是两个用户互为邻居，差距值分别为 6.2 和 8.3，可推荐电影数量分别为 8 和 6。另外对于用户 3，其还有一个邻居用户 ID 为 38，但是差距值大于用户 3，并且可推荐电影数为 0，说明用户 38 评价过的电影包含于用户 3 都评价过的电影。可以看到互为邻居的用户对于对方计算出来的差距值不相同，所以使用邻居一词并不准确，应该说是“粉丝”。A 用户对 B 用户的差距值越小，A 用户就会越喜欢 B 用户评分高的电影。

由此，协同过滤算法生效，可以在两个用户的微信客户端看到协同过滤算法的推荐结果，如图 5-2，推荐结果符合预期。

图 5-2 互为邻居的用户推荐结果

5.1.2 基于内容推荐

对于一个没有找到邻居的用户，或者邻居并没有可推荐的电影场次的时候，认为这是冷启动导致的，所以基于内容的推荐算法将启动，并且由于算法是随机找出 100 部影片，并将其中豆瓣评分最高的一部返回给用户，所以用户每次收到的推荐结果都不一样（也有很小的几率出现推荐结果相同）。

让两个没有对任何电影进行过评分的用户发送推荐命令，系统一定会启动基于内容的推荐算法，如图 5-3 所示结果。

图 5-3 冷启动时的推荐结果

系统已经拥有了“搜索”、“评价”、“推荐”三大功能。

对于搜索功能，如果用户直接发送的不是命令，就把输入内容当做电影名直接完成搜索功能返回结果。如果搜索的电影名不存在，则使用模糊搜索返回最多 5 条结果。

对于评价功能，用户必须保证电影名输入正确，否则评价失败。评分范围在 0-10 分，如果用户评分超出 10 分，则取 10 分，如果评分为负数，则取 0 分，评分可以接受小数点保留后一位。如果用户对一部电影多次评价，则取最后一次评分为准。

对于推荐功能，使用基于用户的协同过滤算法，推荐和目标用户有着共同兴趣爱好的邻居用户评价过的电影。如果由于冷启动或数据稀疏没有找到适合推荐，则启用基于内容的推荐算法弥补。

有了这些功能之后，系统大致完成，但是系统中仍有很多不足的地方。本章将对系统中能够改进的点进行说明，并给出解决思路。

5.2 推荐功能优化

在推荐功能测试中发现推荐结果有缺点。当用户评价电影场数较多后，每次推荐结果都是相同的。或者当数据稀疏时，用户得到的推荐结果并不满意。推荐系统并不是完美无缺的，而是在使用过程中发现问题，并不断解决问题。

5.2.1 重复推荐

分析系统中的算法实现，如果系统中的评价数据不发生变化，发现系统为目标用户找到邻居总是同一个人，并且每次推荐的结果总是相同的电影场次。只有当用户的评价数据发生改变后，推荐结果才会发生改变。这个问题最本质的原因是系统没有记住推荐过什么内容给用户，导致用户下一次寻求推荐时，系统忘了已经推荐过的内容，发生重复推荐，直到用户将推荐给他的电影场次一一评价后，推荐内容才会发生改变。

为解决这个问题，系统可以新增一个数据库表格，记录为每个用户推荐过的电影，如果再次发生推荐时发现已经推荐过，则跳过此电影，往后寻找其他推荐。

在推荐电影之后，可以寻求用户对此推荐的满意程度，如果满意度低，则查看数据源进行分析，如果数据中没有恶意评价，说明推荐算法有改进的空间。

5.2.2 响应时间

在系统建立初期，用户量少，计算邻居用户所需要的时间非常短，但随着用户规模增加，每次寻找邻居都需要花费大量时间。为预防这个问题，系统可以新增一份记录数据，记录所有用户的邻居关系，并当评价数据发生改变时或者定期重新计算邻居关系，而不是用户请求时才发生计算。有了邻居数据记录后，每次用户请求推荐只需要简单查询而不需要临时计算，节省了计算时间。另外，如果查询数据库消耗时间占比较大， 可以引入缓存机制，将经常访问的内容复制到内存中，用户请求时可以直接从内存中查询并返回。

5.2.3 避免冷启动

在系统中，使用基于豆瓣评分的推荐算法来弥补协同过滤算法的冷启动问题，但是

系统中每次随机抽取 100 部电影，并找出最高豆瓣评分的返回给用户，这样的推荐可能出现无数种可能，如果随机抽取的 100 部电影评分都比较低，恰好是用户不喜欢的，则推荐系统是失败的。对于这个问题，系统可以新增一个高评分电影表，将豆瓣评分最高的 1000 部电影存于表中，协同过滤算法失败时可以从这个表中依次读取电影推荐，这样做能使基于内容的推荐效果稳定性提高。

5.3 评价功能优化

评价功能非常重要，因为它是协同过滤推荐的基础。没有评分数据就没有协同过滤。本系统中的评分系统依然存在一些问题。

5.3.1 影片名纠纷

在命令设计时，评分命令为“评价 电影名 分数”，这个命令非常简单，但依然存在一些细微问题。

当用户输入电影名错误时，系统找不到相应的电影，则评分失败。如果用户评价的不是电影名，则评分失败是正常的。而如果用户只是输入时输入有误，则评价失败是系统的失误，例如《你的名字。》电影经常被用户误输入为《你的名字》，少了一个句号导致评价失败，系统损失评分数据。这种情况是可以改进的，解决方法有很多。

解决以上问题，在用户输入评价命令后，搜索电影名，如果没有搜到该电影，则可以使用模糊搜索，并将模糊搜索出来的电影名和对应 ID 返回给用户，提示用户可以使用电影 ID 进行评分。而 ID 是数字，所以不容易出现输入有误。

对电影名重复的不同电影，用户评价时，也可以使用以上方法，但用户评价的电影名搜索出多部同名影片时，可以返回这些同名电影的详细信息以及对应 ID 号，提示用户电影名重复，可以使用电影 ID 号进行评分。

这样的改进可以使评价数据更齐全、更精准，相应的提高协同过滤算法的推荐质量。

5.3.2 鼓励评分

鉴于用户评分对协同过滤算法的重要性，可以做适当的提示和奖励，鼓励用户多进行评价，促进数据填充，使协同过滤算法有更多的推测依据。

5.4 其他辅助功能

搜索功能是一个辅助功能，协同过滤算法也可以基于用户的搜索记录。但是基于用户评分的协同过滤比基于搜索记录的协同过滤更严谨。

在系统中，模糊搜索只是简单地使用 MySQL 的 like 语句实现，这样模糊搜索的结果不会太好，而且用户只能搜索电影名，而不能使用导演、演员、上映地点、上映时间等信息进行搜索。关于这些，系统可以搭建一个搜索引擎，专门提供搜索功能，并记录用户搜索历史。

除了搜索功能，系统还能提供其他一些辅助功能提高系统的完整性。

5.4.1 功能说明书

在新用户对公众号添加关注时，系统会返回操作命令的说明。如果用户长时间没使用该系统，则可能忘记了操作方式，于是可以增加一个获取说明书的功能，例如，用户发送“怎么用”，则系统返回完整的公众号说明书给用户。如果微信公众号有添加菜单功能，也可以增加按钮实现功能。

5.4.2 收藏夹

既然有用户系统，则可以为每个用户提供一个收藏夹，让用户可以将电影添加到收藏夹、查看收藏的电影列表、删除收藏夹中的电影等等。收藏夹也可以作为协同过滤算法的元数据。系统实现起来也比较简单，只需要在数据库中增加相应的一张表记录，然后新增收藏夹相关命令。

结论

个性化推荐系统是在互联网、电子商务发展过程中应用知识发现技术、通过频繁的交互针对不同的用户做出个性化的推荐结果。协同过滤算法是目前应用最广泛的个性化推荐技术。

在本文主要研究工作中，基于 CentOS 操作系统、MySQL、微信公众平台、python、协同过滤算法，一步步实现了一个简单实时的电影资源推荐系统，该系统面向微信用户，可以同时为广大用户提供不间断的服务，拥有搜索、评价、个性化推荐等功能。在文章中间的章节对协同过滤算法作出分析评价，研究其实现方法，最后结合本系统环境实现一个基于协同过滤算法的推荐系统。虽然没有分析用户对推荐结果的反馈情况，但是文章最后也提出了很多改进方案。

参考文献

[1] 张恒玮. 基于协同过滤技术的电子商务推荐系统的研究与实现[D].华北电力大学,2012.

[2] 何安. 协同过滤技术在电子商务推荐系统中的应用研究[D].浙江大学,2007.

[3] 王顺东,雨爱华,陈泽军.协同过滤推荐系统分析[J].计算机光盘软件与应用,2012(12):152.

[4] 百变神鼠. 超酷！特色购物网站推荐[J]. 网友世界.2015.

[5] 孙翌,李鲍,高春玲.微信在图书馆移动服务中的应用研究与实践[J].图书情报工作,2014,58(05):35-40.

[6] blueel. Python 六大开源框架对比：Web2py 略胜一筹 [Z]. 网络: blueel,2013.

[7] 房斌. 一个图书馆文档管理系统的设计与实现[D].北京邮电大学,2008.

[8] Robert Layton.Python 数据挖掘入门与实践[M].人民邮电出版社.2016.

[9] Toby Segaran.Programming Collective Intelligence[M].Sebastopol:O'Reilly Media,2007.

[10] 李幼平,尹柱平.基于用户行为与角色的协同过滤推荐算法[J].计算机系统应用,2011,20(11):103-106.

[11] 孔维梁. 协同过滤推荐系统关键问题研究[D].华中师范大学,2013.

[12] 黄立威,刘艳博,李德毅.基于深度学习的推荐系统[J].计算机学报,2017,:1-29.

[13] 徐天伟,宋雅婷,段崇江.基于协同过滤的个性化推荐选课系统研究[J].现代教育技术,2014,24(06):92-98.

[14] 何清,李宁,罗文娟,史忠植.大数据下的机器学习算法综述[J].模式识别与人工智能,2014,27(04):327-336.

[15] 张素智,苏龙飞.基于用户兴趣的协同过滤推荐算法研究[J].郑州轻工业学院学报(自然科学版),2013,28(05):47-49.

致谢

时光飞逝，一转眼在黑大的四年求学生涯即将结束。回想四年前自己还是一个标准中学生，知识范围被圈在中学教材。在这段期间，得到过很多老师、同学和朋友的关怀和帮助。在此感谢期间支持、帮助过我的所有人。

首先，我要深深感谢我的论文指导老师蔡庆平老师。大学期间第一次接触编程技术是在蔡老师的 C 语言课上，往后四年陆陆续续选修了很多蔡老师开的课，每当遇到困难便会与蔡老师进行交流，蔡老师都会细心听完并悉心指导。直到本次毕业论文的选题和写作，蔡老师每次都给我提供了宝贵意见。在这四年的平凡日子中，蔡老师授以我的不仅是知识，还有为人处世的道理。这些都能使我终生受益。再次感谢我的导师蔡老师。

其次，我要感谢我的室友，是你们在这里伴我一起成长。在我实习不在学校期间，每次学校有事情我回不去室友都会帮我，我在能在实习中学习实践也有室友的支持。

最后，感谢我的父母和家人，尊重我的选择，让我从昆明远到黑龙江大学求学，并在期间给我经济和精神上的支持。在我的人生道路上默默支持我。