申请上海交通大学工程硕士学位论文

基于机器学习的社区文章推荐系统研究

论文作者：刘炯

学　　号：1150332031

交大导师：姚天昉

企业导师：吕栋

院　　系：计算机科学与工程系

工程领域：计算机技术

上海交通大学电子信息与电气工程学院

2018 年 6 月

Thesis Submitted to Shanghai Jiao Tong University

for the Degree of Engineering Master

A STUDY OF

COMMUNITY RECOMMENDATION SYSTEM

BASED ON MACHINE LEARNING

M.D. Candidate : Liu Jiong

Student ID : 1150332031

Supervisor(I) : Yao Tianfang

Supervisor(II) : Lv Dong

Department : Computer Science & Engineering

Speciality : Computer Technology

School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai,P.R.China

Jun, 2018

基于机器学习的社区文章推荐系统研究

摘 要

随着互联网技术的飞速发展，我们逐步迈入大数据时代。近年来，不少社区、论坛等互联网社交、问答的平台迅速崛起。由于互联网上数据量飞速增长，使得用户无从快速地选择所需要的信息。因此，帮助用户快速地搜索到符合用户喜好的信息变得尤为重要。信息推荐技术是实现这一需求的重要手段。通过挖掘、分析用户的历史行为数据、以及社区里的评论文本数据，推荐系统可以快速定位到用户所需要的信息，为用户提供可信的推荐结果。这样一来，用户检索信息的时间就可以大大减少了，而且用户的产品体验也改善了。因此，推荐功能也就起到了改善用户产品体验满意程度的作用，也为用户带来了更方便、更快捷的生活。

协同过滤推荐方法是最早的推荐算法之一。其算法原理是：通过参考用户的历史行为和文章的不同属性来预测用户可能偏好的文章，选取预测得分较高的文章推荐给用户。但是，传统的推荐算法往往存在一些问题。其主要方面归纳如下：（1）数据稀疏性问题;（2）冷启动(Cold Start)问题;（3）同质化问题。

为了解决传统的推荐算法所存在的问题，本文的研究工作提出了一种基于传统协同过滤的改进方法。首先，使用了皮尔森相似度对用户打分进行了归一化，剔除了零得分的文章和由于用户使用习惯差异引入的噪声。。另外，基于隐语义模型的推荐算法解决了在推荐上的稀疏性问题。除此之外，用户历史行为数据只能捕捉到一部分用户的行为习惯，对于没有游览、点击行为的用户便遗漏了信息。因此，本文还提出了使用组合推荐策略的方法。结合协同过滤算法和 LDA 主题模型相融合的推荐算法，一方面从用户评论数据中挖掘出用户的兴趣偏好，根据 LDA 主题模型中用户在不同主题上的概率用来计算用户的相似度。另一方面，改进的协同过滤算法用来计算文章的相似度。结合这两方

面的信息，从不同维度上搜集更全面的用户信息用于推荐服务。这样的组合策略大大减轻了推荐中冷启动问题和同质化问题的影响。通过对实验结果分析得知，基于组合模型的推荐算法在准确率、召回率和多样性方面的优于传统推荐算法。

为了实验验证组合模型的推荐效果，我们采集某企业的真实数据作为数据集，通过对比多种算法，并在多种评估指标上进行了实验。实验结果表明：基于融合模型的推荐算法在上述数据集上取得了良好的成绩。其准确性、查全率和广泛性都获得了提高。因此，该算法明显优于其他传统的推荐算法，从而证明了融合组合模型的推荐算法具有可行性、合理性和有效性。

关键词 协同过滤，隐语义模型，TFIDF，Word2Vec，LDA 主题模型, 神经网络

A STUDY OF COMMUNITY RECOMMENDATION SYSTEM BASED ON MACHINE LEARNING

ABSTRACT

With the rapid development of Internet technology, we are gradually entering the era of big data. In recent years, many communities, BBS, Internet social, question and answer platform etc. are rising rapidly. Because of the rapid growth of the amount of data on the Internet, which makes users fail to select the information needed quickly. Therefore, It becomes quite urgent to help users to search the information which meets users' preferences quickly .Information recommendation technology is the important means to realize this requirement. By mining, analysis of user behavior data, as well as the comment text data in the community, recommendation system can quickly locate the information which user needs, and provide users with reliable results. As a result, not only the time to retrieve information can be greatly reduced, but also the user experience was of the product was improved. Therefore, the recommendations have played an important role in improving user satisfaction with product experience, also provided users more convenient and faster life.

Collaborative filtering recommendation technology is one of the most common recommendation methods. By reference to the history of user behavior and articles' different attributes to predict users' favorite articles, we can select highest scores articles to recommend to the user. The research work of this thesis is to put forward to improve and optimize the traditional collaborative filtering technology. Although traditional collaborative filtering technology is widely used, it has following disadvantages: (1) sparsity of the rating matrix. In the large-scale recommendation system, both the number of user and number of item are very large. But the number of items in fact users expressed preferences is quite few. That is to say,

the known data is very sparse. Sparsity leads to the deviation in the process of calculation, such as calculating the similarity between users or items when making recommendations, but sparsity makes the calculating of similarity between users or items inaccuracy. (2) Cold Start. New user with few rating record, so it is difficult to analyze his preference and it is impossible to make effectively recommendation for him. (3) Homogenization. Users' interests are diverse, the explicit or implicit expressed interest is very limited, the recommendation method based on content filter recommends items which matches with current interest, therefore, this recommendation results homogenization, it is hard for items which users didn't express preference but actually interested to get recommendations.

In view of the above problems of the traditional collaborative filtering, this thesis puts forward the corresponding improved algorithm:

Latent Factor Model (LFM) connects user interests and articles by implicit features. LFM splits the two-dimensional users - articles matrix into user - hidden theme matrix and hidden theme - article matrix, which avoids the occurrence of zero that makes similarity inaccuracy.

Use provided data such as age, gender in user registration to do coarse granularity personalized recommendation. Asking the user to feedback of some items when logging in, collecting user's interest in these items, then recommends the similar items to the user.

In feature extraction and similarity calculation, the use of TFIDF, Word2Vec and some other methods will be applied in the research. Through the experimental analysis, it is found that introducing category features can improve the accuracy of the article recommendation.

To verify the recommend effect of combined model, we collected some real data from enterprise as datasets, by comparing various algorithms, and experiment on a variety of performance metrics, recommendation algorithm based on combined model achieved great score in the above datasets. The accuracy, recall and diversity have been improved. As a result, the algorithm was superior to other traditional recommendation algorithm was proposed, which proved the effectiveness of the recommendation algorithm based on combined model.

Keywords: Collaborative Filtering, Latent Factor Model, TFIDF, Word2Vec

目录

第一章 绪 论

1.1 研究背景

随着互联网技术的快速发展，互联网数据的增长速度远远超出我们的想象。数据爆炸的现象近在眼前。因此，为了适应新的数据时代，人们不得不改变原有获取信息的方式，并在生活中逐步适应新的环境。例如使用视频网站观看视频代替传统的看电视、听广播的方式；使用电子邮件的通信方式代替传统的寄信的方式等等。随着互联网上信息量日益增加，人们急需找到一种方法来解决信息过载所带来的压力[1]。在这种背景下，社区论坛的个性化推荐诞生了。

由于推荐结果是由用户的历史行为记录计算而来，因此，推荐结果因人而异，这也反映出其个性化的一面。协同过滤算法是最常用的推荐算法之一，经过多年理论研究和技术创新，目前已经被广泛应用。。

1.2 研究目的与意义

1.2.1 研究目的

简单地说，研究信息推荐技术的目标就是建立用户兴趣和物品之间的有效匹配算法、模型和系统，最终实现用户感兴趣物品的推荐，从而缓解用户在面对海量物品时信息过载的问题，提高物品信息的利用率[2]。另外，没有用户的明确需求时，个性化推荐可以快速找到满足用户需求的推荐结果并发送给用户

1.2.2 研究意义

如今，信息推荐技术已广泛应用于各个领域。不论是京东、淘宝、亚马逊这样的电商还是豆瓣电影、优酷视频、QQ 音乐、今日头条等互联网信息行业等等。这些网站根据用户的购物记录、页面浏览历史、搜索记录、加购物车情况和收藏/喜欢的物品，向用户推荐商品，此举既满足了消费者的购买需求又提升了商家的销售量。本文主要研究对象是为某企业网站的社区用户推荐文章，这样可以解决用户在巨量信息面前无从选择的难题。同时也可以减少用户的搜索时间，以提

高检索的效率。

1.3 国内外研究现状

1.3.1 国外现状

信息推荐具有重要的应用价值， 因此受到各界人士的普遍关注。自 2007 年以来，美国计算机组织 ACM 举行了推荐系统学习会议（简称 RecSys）。同时，许多知名企业也纷纷推出了自家的信息推荐竞赛，例如 Netflix 之类。这些活动大大激励了信息推荐技术的研究与发展。

在商业应用方面，电商领域率先将推荐系统落实到实践应用中。著名的电子商务网站亚马逊率先推出了个性化的推荐系统，享有“推荐系统之王”[3]的美誉。亚马逊的个性化产品推荐列表和相关产品的推荐列表功能带来了巨大的利润。其销售额中的 35%来自于推荐功能。个性化推荐的商业价值可见一斑。

目前，人们主要在探索推荐算法和其功能应用上。常见的推荐算法又包括以下几种：基于上下文推荐、基于协同过滤推荐和基于模型推荐。这几种算法各有所长，不仅算法不同，而且可以适应于不同的使用场景。

基于上下文推荐

传统的基于上下文的推荐算法通过手动提取或自动生成获得文章的描述或特征信息，通过对比文章特征之间的相似性，找到与目标文章的内容相似的其他文章。这种推荐方法不需要获取用户的行为记录，其特征由文章的固有特征中提取。所以这种方法可以克服个性化推荐系统中的冷启动问题和用户兴趣建模困难的情景。Davidson[4]等人通过研究视频推荐系统，并在 YouTube 网站上实现了个性化的推荐服务。Sarwar[5]等人研究了电子商务网站上的推荐系统。主要分析在大规模购买中，挖掘用户的购买倾向与偏好。在他们的调研中，使用了两部分的研究数据。一部分来源于网购购物记录，另一部分来源于 MovieLens。

随着机器学习和深度学习的研究与应用技术的日趋成熟，我们还可以通过挖掘用户的行为日志，用以构建用户肖像。用户肖像包含了如下几方面的数据：用户个性定义、社交行为、年龄、性别等个人化特征。因此，用户肖像也是基于上下文推荐算法的一种新的表达方法。通过提取用户肖像和产品肖像文件中的特

征，并且通过比较两个特征之间的相似性来测量用户的潜在兴趣水平，从而获取个人化的推荐。

基于协同过滤推荐

与其他推荐算法相比，协同过滤算法（Collaborative Filtering，简称 CF 算法）的特色在于不仅考察目标用户的历史行为，还考察相近用户的历史行为。因此，协同过滤算法有其自身的优越性：第一，它可以分辨出常见算法无法识别的项目（例如音乐、视频等）；第二，由于协同过滤基于相近用户的历史评价信息进行推荐，故其推荐质量得到了保障。

协同过滤算法又分为基于用户的协同过滤算法（User-based Collaborative Filtering，User CF，简称 User-CF）和基于物品的协同过滤算法（Item-based Collaborative Filtering，Item-CF,简称 Item-CF）。其原理和实现方法可以归纳为：“物以类聚，人以群分”。

基于协同过滤的推荐算法已广泛应用于各个行业,其算法和应用也日趋成熟。在$Qiu\ T$[6]等人的研究中，改进了传统的基于物品的协同过滤算法，并且为了增加数据源的多样性，使用了 RYM, Netflix 和 MovieLens 的数据源。此外，实验表明，使用改进的算法可以提高冷启动物品的推荐精确度。$Schein\ A\ I$[7]等人，在其设计的框架中，集成了基于上下文推荐的推荐技术和协同过滤推荐算法。并且还使用了朴素贝叶斯分类器。

基于隐语义模型推荐

自 Netflix 竞赛以来，隐语义模型(latent factor model,简称 LFM）在推荐领域越来越为人所知。在早期文本挖掘的研究中，隐语义模型已经初露锋芒。类似的算法有 pLSA、 LDA 和 Topic Model。

隐语义模型是近几年来热门讨论的推荐算法之一隐语义模型分类的颗粒度取决于隐因子的大小。隐因子数越大，分类的粒度越细，反之分类粒度越粗。此外，隐语义模型计算出文章/物品属于每个类的权重，因此每个文章/物品都不是硬性地被分到某一个类中。隐语义给出的每个类别都不属于同一维度。如果喜欢某种类型文章/物品的用户都喜欢某个文章/物品，那么该类别中文章/物品的权重将增加。

隐语义模型也在许多场景得到广泛应用。$Shen\ Y$[8]等人通过分析用户在互联网社交平台上的行为记录，挖掘用户间的关系，实现了相近用户间的兴趣分享功能。在他们设计的推荐系统中，基于传统的隐语义模型提出了改进，设计了PSLF(personal and social latent factor)模型。他们的模型结合了最新的协同过滤算法和社交网络模型。

1.3.2 国内现状

国内对推荐系统的研究主要集中在推荐算法和技术应用方面,这几年各大公司，比如百度、阿里、腾讯纷纷取得了不错的成果。推荐系统经历了从刚刚问世、高速发展、直到被广泛应用三个不同的阶段。如今已经成熟并适用于各行各业。

为满足不同场景的需求，新型算法不断涌现。随着研究的深入推荐算法通过不断改进和优化。在有些场景下，甚至和专业领域的业务知识相结合从而产生了适合特定环境的推荐算法。这极大地改善了推荐算法的多样性和适应不同场景地能力。例如数据挖掘、人工智能、信息检索、网络安全等理论领域在实际应用中与推荐算法相结合，为推荐系统提供了新的分析和方法。刘青文[9]通过用户历史行为和数据建立协同过滤模型，挖掘用户需求和偏好。并且，基于该算法，从海量信息中过滤出用户感兴趣的信息。曹毅[10]研究了基于上下文和协同过滤算法的推荐算法，并将其应用于电子商务领域。他的算法结合了基于内容推荐和基于协同过滤推荐的优点。可以对所有项目进行相似度计算，即使是没有被用户评价过的项目，仍然可以被过滤出来，推荐给用户。杨兴耀[11]等人开发并实现了基于物品特性的协同过滤模型。该模型通过计算客户评分的相近度（客户在不同项目上的评分数量）来求取项目的相近度，实验表明：该模型提高了推荐结果的准确性。

另外，由于获得了海量数据，使推荐算法在适应大数据环境方面又取得了新的进步。通过结合评论分析和隐语义模型，尹路通等人[12]通过分析客户在视频网站上评论内容来辨别客户的情绪倾向。此外，通过构造虚拟评估方阵以解决显示评估方阵数据疏落性的难题，从而成功减小了数据缺失对网络视频造成的影响。

此外，推荐系统也在各大行业、领域广泛使用。国内各大知名企业，比如腾讯，百度，阿里都有自己的推荐系统。在淘宝、新闻头条、QQ 音乐、喜马拉雅 FM 的应用程序界面上都赫然出现了诸如“猜你喜欢”、“推荐”之类的功能。这些功能的出现，快速提高了网站的点击量和商品的购买率，体现了极大的商业价值。

1.4 本文的贡献

本文详细阐述了推荐系统的发展现状及面临的一些问题。提出了可行并有效的技术和方法并将 CF 算法、LFM 推荐技术、基于神经网络(CNN, RNN)的文本过滤技术、基于 LDA 模型的客户喜好挖掘方法合理地融入到推荐系统中。针对目前行业内遇到的问题实现相应的改进措施，有效解决了推荐系统中的若干问题。社区文章平台数据集的实验结果表明：该方法在推荐准确率、召回率和多样性方面有较大提高。

1.4.1 创新点

随着互联网上数据量的不断膨胀，我们不知不觉中已经步入了大数据时代。传统的推荐算法可扩展性非常有限，单一的推荐算法往往无法挖掘出全面的数据特征和用户兴趣。常用的推荐算法的问题比较明显，比如协同过滤算法存在数据稀疏性问题和推荐结果同质化问题等。多维度的数据没有得到有效使用，非常可惜。因此，本文在对社区文章推荐系统的研究过程中采用了多种机器学习算法和模型，有效地融合了协同过滤算法、隐语义模型推荐、基于神经网络的文本过滤技术、基于 LDA 主题模型来挖掘用户兴趣实现推荐，实现了混合多元化的推荐系统。

1.4.2 难度

本文将自然语言处理技术与推荐技术相结合，高效地利用了多维度的数据，并提出了基于 LDA 主题模型的用户兴趣挖掘方法和协同过滤算法相融合的推荐系统。此外，为了进一步提高推荐效果，对原始用户评论的文本数据进行了文本过滤，确保输入给 LDA 主题模型的文本数据的可靠性。此外，还剔除了冗余无效的垃圾评论。此方法解决了常见的数据的稀疏性问题和同质化问题，在确保推荐准确率和召回率平稳的前提下，还大大提高了推荐结果的多样性。

1.4.3 工作量

首先，本文深入研究了 CF 算法和 LFM 模型，并提出了改进方法。通过对神经网络的研究并结合自然语言处理技术，将有关成果运用到推荐系统的文本过滤中。然后，通过深入分析目前推荐系统所遇到的问题，思考了各种解决方案，使用了融合推荐策略，设计一种融合 LDA 与 CF 算法的推荐系统，并通过多维度的数据加以实现。结合不同模型各自的优点和长处，实现了一种混合多种模型的推荐系统。实验表明：该模型在推荐准确性、查全率、广泛性上取得了良好的效果。

1.5 本文主要研究内容及各章安排

1.5.1 研究内容

一般而言，信息推荐包含了用户兴趣建模、 物品建模和用户兴趣-物品的匹配三个基本步骤。本文的研究工作也围绕着这三部分展开。另外，使用协同过滤推荐算法和隐语义模型推荐算法进行推荐。并且，在原来算法基础上，做了改进。除此之外，结合了自然语言文本处理技术和推荐技术，设计了融合神经网络文本过滤和 LDA 主题模型推荐相结合的推荐算法。

在用户兴趣建模过程中，可以基于用户的浏览、点击和收藏行为数据来预测用户的兴趣。通过将用户的兴趣表示成向量，系统可以通过向量空间中的计算方法来进行推荐。

在物品建模过程中，最常见的一种做法就是将物品表示为其重要特征或属性表示的向量。当进行特征选择时，可以借用传统文本分类中 TFIDF 的特征选择方法。在进行特征表示时，则可以从属性在物品内的出现次数和出现属性的物品数目两个方面来综合考虑属性的权重，前者描绘了属性的表示，也就是说，属性在项目中出现得越多，属性的权重就越大；后者刻画了属性的区别性，即该属性在所有物品中出现越多，则意味着其区别性不大，此时反而要降低权重。

在用户兴趣-物品的匹配度计算时，将会基于几个不同的推荐算法对上述的三部分进行研究与实验比对，并提出改进方法。在第一种推荐算法中，基于协同过滤的方法进行推荐计算，在此基础上，通过计算皮尔森相似度的归一化方法对推荐算法进行改进，剔除零得分文章和用户使用习惯差异引入的噪声。在第二种推荐算法中，在（LFM）隐语义模型的基础上提出基于增量更新的隐语义模型进

行推荐。在第三种推荐算法中，首先，使用 CNN（卷积神经网络）、RNN(循环神经网络)过滤社区论坛中用户评论数据，剔除无效评论数据。然后，使用 LDA 主题模型挖掘用户兴趣点和偏好，同时求得相似用户，根据相似用户的阅读文章对目标用户推荐文章。在这个混合模型中，扩展源数据维度，结合了自然语言处理技术、LDA 主题模型和协同过滤等推荐技术各自的长处改善推荐效果。最后，评估推荐结果的准确率、召回率、多样性和其他性能指标，以验证推荐模型的有效性。

1.5.2 各章安排

第一章是绪论，主要介绍了研究背景，研究目的和意义。重点介绍了国内外的现状，本论文的研究内容和各章的内容安排。

第二章是实现推荐系统的理论基础。主要介绍了 CF 算法原理、基于上下文推荐算法、基于矩阵分解的算法。另外，介绍了特征词提取技术，包括 TF-IDF 和词向量模型。突出了不同的网络模型，即 CNN 网络和 RNN 网络的不同点。并且，还介绍了各自的理论基础和特点。

第三章介绍了 Item-CF 所面临的问题，并详细说明了文章推荐系统的设计原则和实现方法。此外，还详细介绍了基于矩阵分解的文章推荐系统的设计原理和实现方法。

第四章介绍了基于神经网络和 LDA 模型的客户偏好挖掘算法和推荐系统实现。包括基于神经网络的文本过滤的设计原理和实现过程和基于 LDA 模型的推荐系统的实现过程。

第五章重点论述了 CF 算法的实验结果、基于 LFM 模型的实验结果、基于神经网络的文本过滤的实验结果和结合 LDA 主题模型的推荐系统的实验结果。详细分析了各自的优缺点，并且展现了各自的准确性、查全率和广泛性。

第六章为总结与展望。全面总结了本文中主要的研究工作、研究内容、研究方法和克服的难题。同时，也总结了自身的不足及改进方法。此外，思考了在现有推荐系统的基础上做进一步的改进。

第二章 推荐系统理论基础

2.1 协同过滤算法原理

2.1.1 基于物品的协同过滤

协同过滤（Collaborative Filter, 简称 CF）算法是目前研究并使用最为广泛的推荐算法之一。基于物品的协同过滤算法的主要思想是：根据用户项目/文章历史得分或者其他历史行为反馈数据计算项目/文章之间的相似度，然后对目标项目/文章的所有的相似项进行排序，取 Top-N 个物品，最后推荐给用户[13]。

其原理如图 2-1 所示，用户 A 在阅读了文章 3 和文章 4 的同时也阅读了文章 1，用户 B 在阅读了文章 3 的同时也阅读了文章 1。根据基于项目/文章的协同过滤推荐算法，我们可以认为文章 1 和文章 3 非常相似。故可以在用户 C 阅读了文章 3 时，向其推荐文章 1。

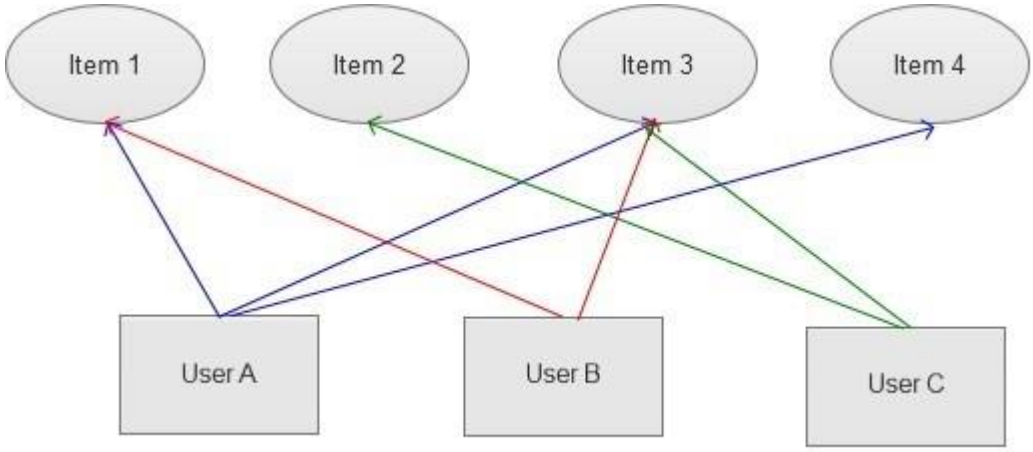

图 2-1 基于物品协同过滤原理图

Fig.2-1 Item-based Collaborative Filter diagram

2.1.2 基于用户的协同过滤

基于用户的协同过滤的算法思想也是同理，只不过计算的是用户间的相似度。将相似用户阅读过的文章推荐给目标用户。如图 2-2 所示，用户 A 阅读了文章 1、文章 3、文章 4，用户 B 阅读了文章 1、文章 3, 用户 C 阅读了文章 2、文章

5。这里的“阅读”是广义的对文章的正面行为。例如点赞或者正面的评价，也可以是隐式的方式，比如多次游览，停留时间较长等。故用户 A、用户 B 是相似用户，将文章 4 推荐给用户 B。

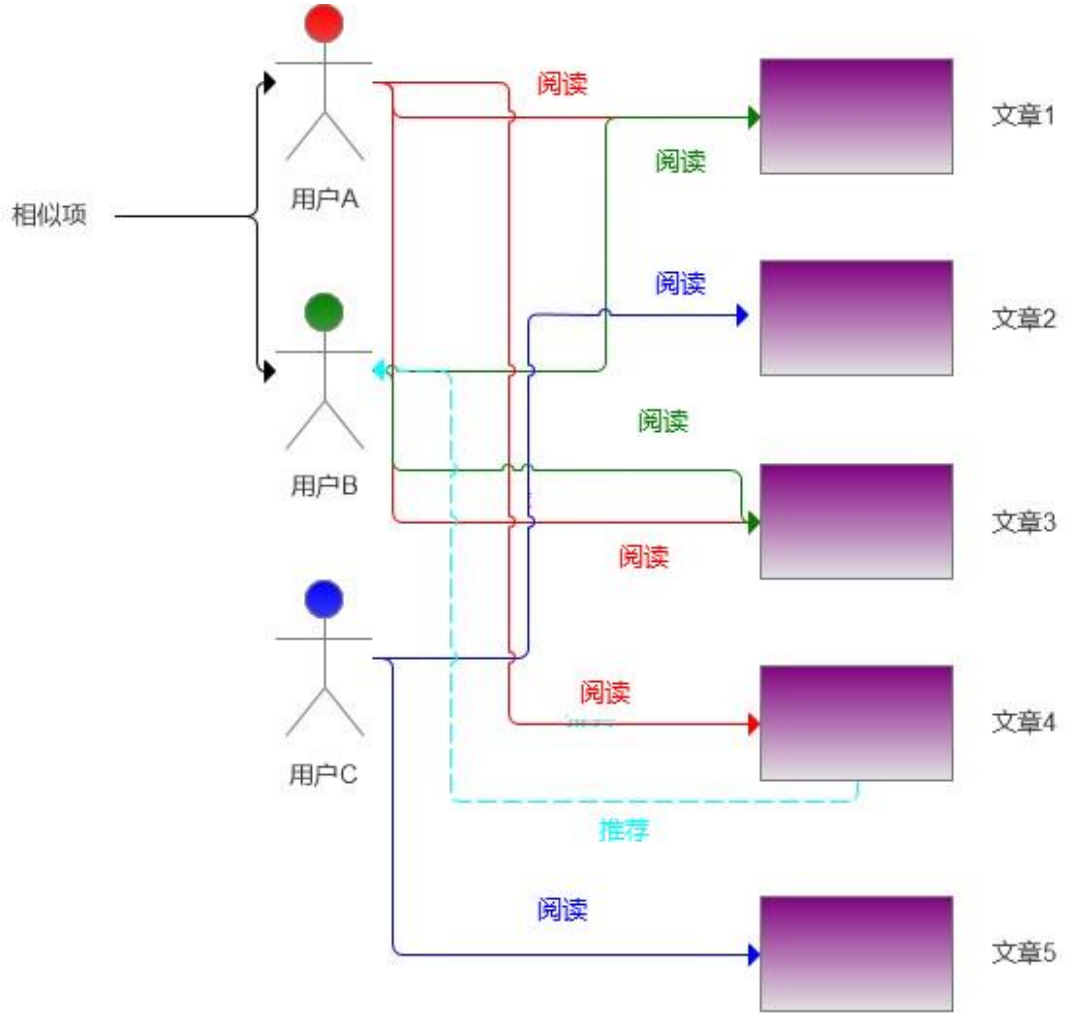

图 2-2 基于用户协同过滤原理图

Fig.2-2 User-based Collaborative Filter diagram

Item-CF 算法和 User-CF 算法都有自己的优点和缺点。这两者在性能、使用领域、实时性、冷启动问题[14][15]、推荐理由方面的对比情况如图 2-3 所示。

	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 2-3 协同过滤对比

Fig.2-3 Collaborate Filtering Comparison

总结基于物品的协同过滤与基于用户的协同过滤各自优缺点如下：

- 协同过滤优点：

1. 基于用户行为，因此对推荐内容不需要事先知道；
2. 仅凭用户和文章的评分方阵数据就可计算；
3. 用户行为越多样化，效果越显著。

- 协同过滤缺点：

1. 需要大量的显性/隐性用户行为；
2. 需要计算完全相同属性下的文章属性，不同属性之间无法计算；
3. 用户的兴趣完全取决于于用户历史行为，不受上下文环境影响；

2.2 基于上下文推荐

基于上下文的推荐算法是找到用户偏好的项目/文章，并向用户推荐具有其contexts/属性的项目/文章。基于上下文的推荐算法不需要考虑用户与用户之间的关联，而是需要分析目标内容与推荐内容之间的相似性。通常，使用在推荐相似文本内容上。比如，物品通过内容（关键词）关联：

- 电影题材：爱情/侦探/动作/喜剧/悬疑
- 标志特征：陈思诚/王宝强…

- 年代：2017，2018…
- 关键词：唐人街，搞笑

基于上下文推荐需要为每个目标内容创建结构文档，例如，词k_j在文件d_j中的权重w_{ij}。常用的方法比如 TF-IDF。另外，需要对用户也建立一份结构文档，比如，定义一个权重向量$(w_{u1},\dots,w_{uk})$。其中，w_{ui}表示第i个词对用户 u 的重要度。$i \in [1,k]$。最后，计算匹配情况。

- 基于上下文的推荐系统的优点有：

可以处理冷启动问题。只需要抽取新项目/文章的特征来匹配已经有过的项目/文章，从而得到推荐，并不需要考虑用户是否阅读过此种项目/文章。

解释性更强。客户知道推荐原因，因此对推荐结果更信服。客户的使用体验得到改善。此外，不受疏落数据的影响。只要用户与项目/文章关联，就可以基于所分析的项目/文章特征来推荐。

- 基于上下文的推荐系统的缺点有：

无法获取全量的物品/文章特征。

推荐结果的质量难以保障。比如两篇含有相同关键词的文章。一篇写得很流畅，另外一篇却写得很混乱。由于两篇文章非常近似，机器便难以分辨好坏。

推荐内容缺乏新颖性。由于算法根据客户历史评价记录进行推荐，故推荐出的文章都是用户曾经游览过或者非常熟悉的项目/文章。

存在用户冷启动问题。对于新用户由于缺乏历史记录，故推荐准确性不高。

2.3 基于矩阵分解

基于矩阵分解模型推荐算法核心思想是：通过发现隐藏因子，尝试将用户-项目/文章评分分解为用户-隐因子矩阵与项目/文章-隐因子矩阵的乘积。隐含语义分析技术采取基于用户行为统计的自动聚类，发展至今日，也已经衍生出许多著名的方法和模型，比如，pLSA, LDA 和 Topic Model 等等。如图 2-4 所示，假设有 m 个用户，n 个项目/文章，R 为打分矩阵。假定有 r 个隐藏因子，并 r< m, r< n。将 m x n 维的评分矩阵分解为用户-隐因子矩阵,隐因子方阵和隐因子-项目/文章矩阵的乘积。

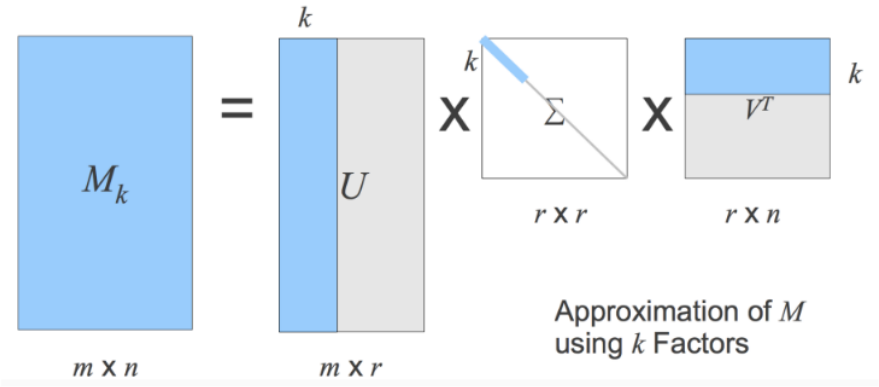

图 2-4 隐因子模型原理图

Fig.2-4 Latent Factor Model diagram

2.4 特征词提取技术

2.4.1 TF-IDF 模型

TF-IDF 模型是自然语言中最常用的关键词抽取技术之一。这个指标通常用来表示文章中词语的重要程度。TF-IDF 的含义是词频逆文档频率[16]，其核心思想是：单词或短语在文档中出现的频率越高，那么这个单词或短语在这篇文档中重要性就越高。如果单词或短语在所有文档中出现的越频繁，则单词越不重要。TFIDF 模型可以表示为公式(2-1)：

$$TFIDF_{ij}(t,d) = tf(t,d)_{ij} * idf(t)_{ij}$$ [16]

（2-1）

其中，tf(t,d)表示词 t 在文档 d 中出现的频次，idf(t)表示词 t 在文档 d 的逆文档词频系数[16]。

词频系数[16]（Term Frequency, TF）的重要性可表示为公式（2-2）：

$$TF_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$ [16]

（2-2）

在公式 2-2 中，分子是该词在文档中的出现次数，而分母是在文档中所有词的出现次数之和[16]。

逆文档词频系数[16]（Inverse Document Frequency, IDF）的重要性可表

示为公式（2-3）：

$$IDF_{ij} = log\frac{|D|}{|\{j: t_i \in d_j\}|} \quad [16]$$

（2-3）

其中，

|D|：语料库中文档总数

j: 包含词语的文档数目

TFIDF 算法虽然在许多场合被广泛应用[17]，但是也存在着一些缺点，比如，

(1) 文档中的每个词都是独立的，词与词之间的关系无法获得。

(2) 如果用向量来表示文档中的每个词，当词汇量增加时，词向量间的矩阵乘法、搜索词向量的索引值等计算会带来极大的时间开销，并且对内存的压力也非常大。因此，这种方法并不适合在神经网络中使用。

因此，Hinton提出[18]了使用低维空间的词汇表示方法。

2.4.2 词向量模型

Word2Vec 也称 Word Embeddings, 中文叫做“词向量”或者“词嵌入”。Word2Vec可以将文档中的字词转为向量形式的表达(Vector Representations）的模型模型[19]。Mikolov[20]等人提出了基于大规模数据的词向量模型。在他的研究中，这种方法可以用来计算词的相似性。并且在计算准确率上由大幅提高。在他的另外一篇论文中[21]，提到用这种词向量的方法引用于机器翻译。自然语言处理在 Word2Vec 出现之前，通常使用 One-Hot Encoder ，使用 One-Hot Encoder 存在一个问题，由于对特征的编码是随机的，所以这种方式没有考虑到字词间可能存在的关系。另外，稀疏矩阵的训练效率也非常低。而使用向量表达[22]（Vector Representations）可以非常有效地解决这个问题，向量空间模型（Vector Space Models）可以将字词转为连续值，具有相似含义的单词将被映射到向量空间中相近的位置。如图 2-5 所示为英语与西班牙语中，近义词的向量分布。由图中所示，相近词的向量位置非常相似。根据近义词的向量分布类似，这样的特征，应用词向量原理，便能解决识别同义词、近义词的问

题。

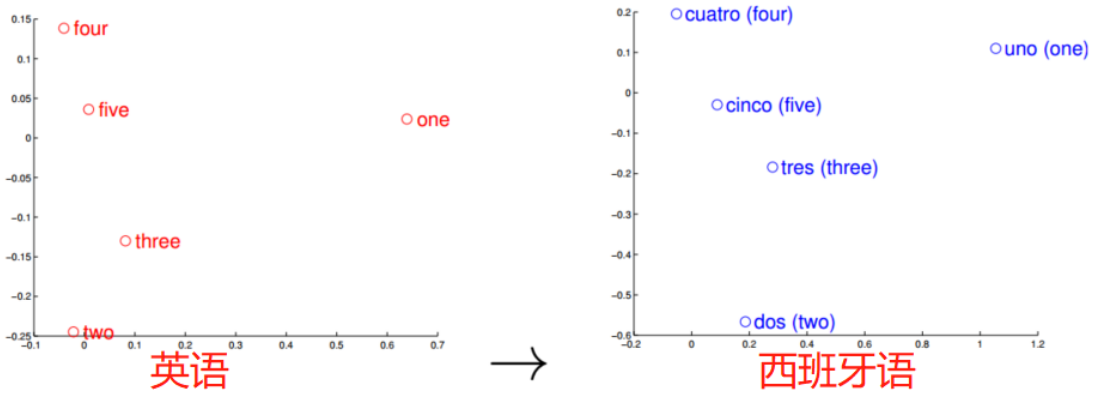

图 2-5 英语与西班牙语近义词的词向量对照图[21]

Fig.2-5 Synonym Of Word Embedding In English And Spanish Comparison Diagram[21]

n-gram 模型的核心思想是：句子中某个词汇出现概率和这个词前后相邻的几个词相关。相邻词汇的数量称为滑窗大小。句子 S 由若干个词$W_1, W_2, \ldots W_p$组成，词W_t出现在上下文中的概率可以近似等于W_t在滑动窗口中出现的概率。如公式(2-4)：

$$\mathrm{p}(W_t|\,Context) \approx p(W_t \mid W_{t-n+1}^{t-1}) \tag{2-4}$$

使用 n-gram 模型可以有效地估计词汇出现的概率。但也存在一些缺点：

(1) 滑窗大小局限性：只能考虑目标词汇前后相邻的词汇。对于不相邻但是起作用的词汇没有考虑到。

(2) 忽略了低频词汇的影响力。对于没有出现在训练样本中的单词，则设置出现概率就为 0，导致计算完整句子的概率为 0。

(3) 无法区分相似词和同义词。

针对于上述问题，Bengio 提出了一个使用三层神经网络来构建 n-gram 模型的方法。结构如图 $2-6$ 所示[18]。即通过前 n − 1 个词$W_{t-n+1}, W_{t-n+2}, \ldots W_{t-1}$预测下一个词$W_t$的概率。

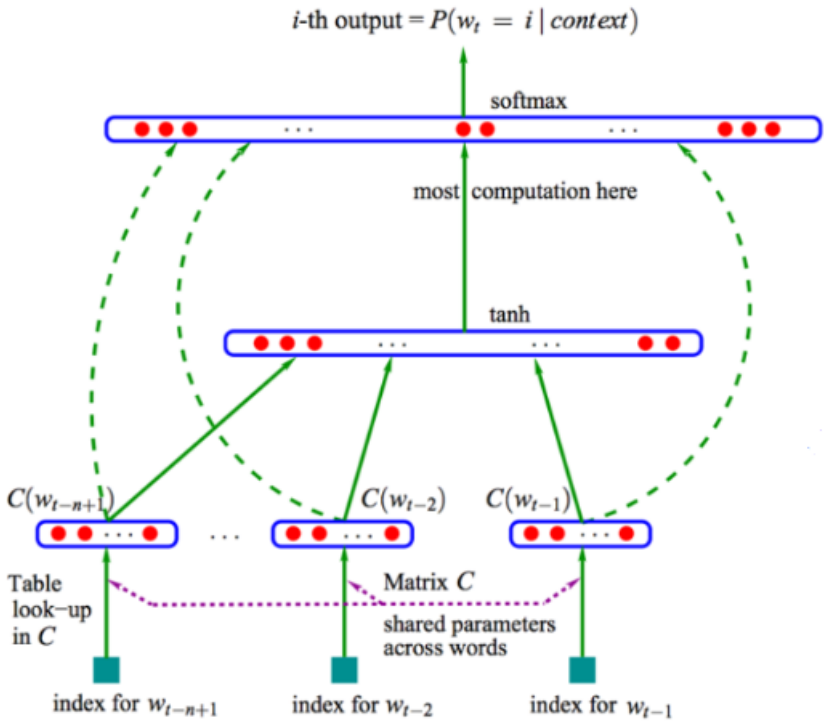

图 2-6 Bengio 神经网络模型[18]

Fig.2-6 Bengio Neural Network Model[18]

在 他提出 的模型中,隐藏层到输出层的参数和隐藏层的矩阵乘法限制了整个模型能达到的最好性能，换句话说，也是这个模型的“天花板”。后人又针对这一缺陷做了优化:(1) Ronan Collobert 等人[23]在 Begio 的基础上做出了改进。通过训练词向量去完成词性标注、命名实体识别等问题。将输出层简化为一个得分输出。(2) Geoffrey Hinton 等人[24]使用一个树形结果替代了原来模型中的输出层。这样一来，输出层的维数瞬间降低不少。同时，也起到了一个词汇在不同上下文中解释为多种含义的作用。(3) Mikolov 在 2010 年发表的论文中采用循环神经网络模型代替了 Bengio 的前馈神经网络模型[25]，这种改进使用了完整的上下文信息来预测词汇的出现。从而克服了 n-gram 理论中滑窗局限性的问题。(4) Huang 的语义强化：Eric H.Huang 在 C&W 的基础上进行了改进[26] 。

Word2Vec 是 Google2013 年的开源词向量工具。它的使用原理分为 CBOW(Continuous Bag of Words)和 Skip-Gram 两种模式。其中，CBOW 是从原

始上下文推测目标字词，而 Skip-Gram 是从目标字词猜测上下文情景。CBOW 与 Skip-Gram 的模型如图 2-7。在第五章中所搭建的神经网络模型中会用到词向量模型。

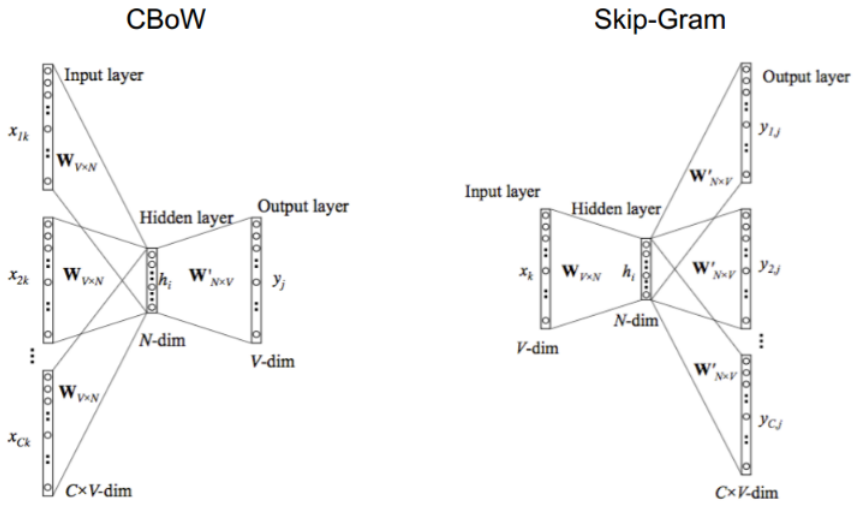

图 2-7 词向量模型图[20]

Fig.2-7 Word Embedding Model Diagram[20]

2.5 卷积神经网络

人工神经网络有很多种，不同类型的神经网络有各自的特点和使用场景。本文主要研究和使用了卷积神经网络（简称 CNN）和循环神经网络(简称 RNN)。

卷积神经网络（Convolutional Neural Networks, CNN）是 MLPs 的一个变种[27][28]。受到感受野（Receptive Filed）机制的启发而提出。

2.5.1 权值共享

在 CNN 中，每个过滤器h_i是整个视场的一个复制。这些复制的单元共享相同的参数（权重向量和偏置），并形成一张特征图，如图 2-8 所示。

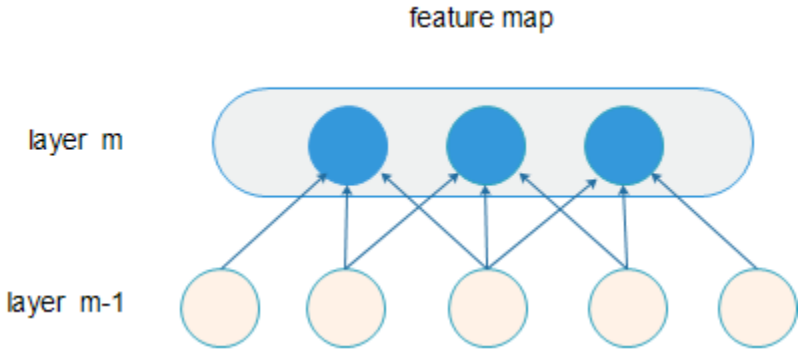

图 2-8 BP 特征映射图

Fig.2-8 Feature Mapping Diagram

在图 2-8 中，第一行的三个蓝色隐藏单元同属于一个特征图。相同颜色权重被共享，即它们的约束是相同的。梯度下降仍然可以用来学习这种共享参数。以这种方式复制的单元可以不管其在视场的位置即可检测出其特征。此外，权重共享通过极大地降低自由参数的数目而增加学习效率。在模型上的约束使 CNN 实现了更好的视觉泛化。

2.5.2 池化层

尽管卷积层降低了连接数量，但是感知器数量并没有明显降低。当感知器的数量过多时，可能导致的过拟合问题。因此，通常在卷积层之后再添加池化（Pooling）操作。从而大大降低特征维度避免过拟合。

假设卷积层输入为$x^{(l)}$，那么通过卷积操作和赤化操作的输入如公式（2-5）和公式（2-6）所示。

其中，$w^{(l+1)}$和$b^{(l+1)}$分别是卷积操作的权重和偏置。

$$X^{(l+1)} = f(Z^{L+1}) \tag{2-5}$$

$$= \mathrm{f}\left(w^{(l+1)} \cdot down(x^{l}) + b^{(l+1)}\right) \tag{2-6}$$

$\mathrm{down}(X^{l})$是指下采样后的输出特征。

下采样函数$\mathrm{down}(\cdot)$一般是取区域内所有感知器的最大元素（Maximum Pooling）或元素的平均值[30]（Average Pooling）。最大池化的计算方法如公式 2-7，a_i表示划片区域内所有的神经元，R_k表示划片神经元的总数。计算划片区

域内的最大值作为池化层里此区域的输出。如图 2-9 所示，在右上角$\begin{bmatrix}1 & 1\\5 & 6\end{bmatrix}$四个数区域内，6 为最大值，故此区域最大池化的输出为 6。同理，均值池化的计算方法如公式 2-8，计算划片区域内元素的均值，作为池化层此区域的输出。

$$pool_{max}(R_k) = \max_{i \in R_k} a_i \tag{2-7}$$

$$pool_{avg}(R_k) = \frac{1}{|R_k|}\sum_{i \in R_k} a_i \tag{2-8}$$

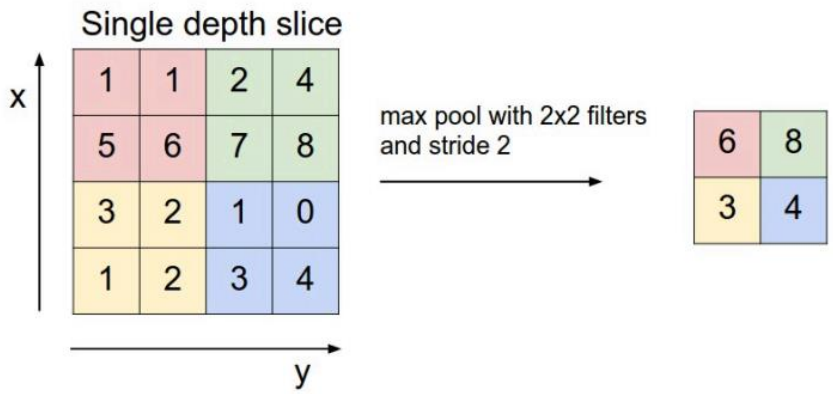

图 2-9 CNN 最大池化示例图

Fig.2-9 CNN Max Pooling Diagram

2.6 循环神经网络

循环神经网络（recurrent neural network，简称 RNN）是一种用于处理序列数据的神经网络[29]。循环神经网络是专门用于处理序列$x^{(1)},\cdots x^{(\tau)}$的神经网络。循环神经网络可以扩展到更长的序列（比非基于序列的特化网络长得多）。

2.6.1 网络结构

首先，我们来看一个简单的循环神经网络例子。

给定一个输入序列$x^{(1:n)} = \left(x^{(1)}, x^{(2)}, \cdots x^{(t)}, \cdots x^{(n)}\right)$，循环神经网络通过下面公式更新带反馈的隐藏层的活性值$s^{(t)}$：

$$s_t = \begin{cases} 0 & t = 0 \\ f(ws_{t-1}, ux_t) & otherwise \end{cases} \tag{2-9}$$

如图 2-10 是 RNN 的展开结构图。其中，

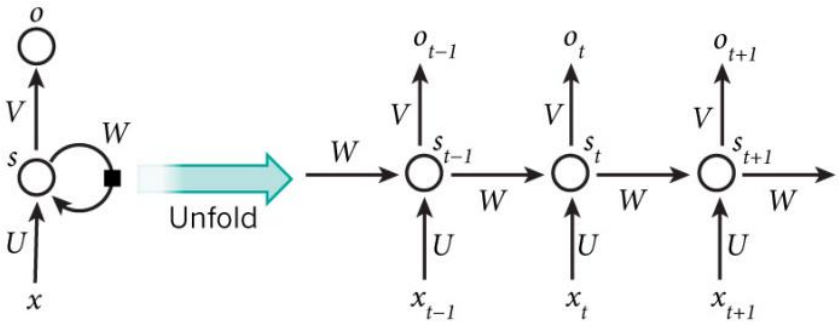

图 2-10 RNN 展开结构

Fig.2-10 RNN Unfold Structure

- x_t是时间 t 处的输入；
- s_t是时间 t 处的记忆；
- f 是 tanh 等函数；
- O_t是时间 t 的输出；

可以把隐状态s_t视作“内存”，捕捉前一时间点的信息。输出O_t由当前时间及之前所有的“内存”计算得出的。

与 CNN 不同， RNN 实际上在整个神经网络都共享一组参数（U，V，W），这可以大大减小需要训练和预估的参数量。

2.6.2 长短时记忆神经网络

循环神经网络存在长期依赖问题。为了解决这个问题，Hochreiter 和 Schmidhuber[31]于 1997 年提出了一个非常好的解决方案，就是引入门机制（Gating Mechanism)来控制信息的累计速度，并可以选择遗忘之前累计的信息。

长短时记忆神经网络（Long Short-Term Memory Neural Network, LSTM）是循环网络的一个特殊种类。LSTM 模型的特点是模拟细胞状态的输入输出过程以及状态的变化。选择性地更新旧数据和使用新数据更新“内存”的过程。假如在时刻 t，内存单元c_t， 更新流程如下：

1 忘记门f_t控制哪些数据要丢掉；

2 确定更新哪些数据到记忆体（内存）c_t中；

3 输出门o_t更新记忆体（内存）状态：

输入门i_t，遗忘门f_t和输出门o_t的范围$\in [0,1]$。

LSTM 的更新公式如下：

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1}) \tag{2-10}$$

$$f_t = \sigma\left(W_i x_t + U_f h_{t-1} + V_f c_{t-1}\right) \tag{2-11}$$

$$O_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t) \tag{2-12}$$

$$\tilde{c_t} = tanh(W_c x_t + U_c h_{t-1}) \tag{2-13}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c_t} \tag{2-14}$$

$$h_t = o_t \otimes tanh(c_t) \tag{2-15}$$

这里的参数设置如下：

x_t：当前时刻的输入，

σ：logistic 函数

对角矩阵 V_i, V_f, V_o。

LSTM 模型计算结构如图 2-11 所示。

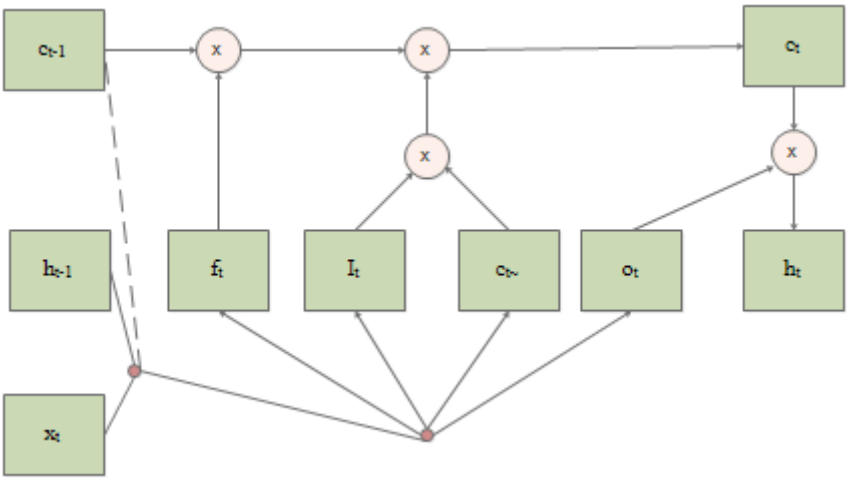

图 2-11 LSTM 结构

Fig.2-11 LSTM Structure

门限循环单元（Gated Recurrent Unit, GRU）是一种比 LSTM 更加简化的版本。GRU 将输入门与遗忘门合并成一个门：更新门（Update Gate），同时还合并了内存单元和神经元活性。

GRU 模型中有两个门（操作）：更新门 z 和重置门 r。更新门 z 用来控制当前

的状态需要遗忘历史信息的数量和接受新信息的数量。重置门 r 用来控制历史信息量占候选状态中的信息量的比例。

GRU 模型的更新方式如下：

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{2-16}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{2-17}$$

$$\widetilde{h_t} = tanh\big(W_c x_t + U(r_t \otimes h_{t-1})\big) \tag{2-18}$$

$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes \widetilde{h_t} \tag{2-19}$$

2.7 LDA 主题模型

2.7.1 LDA 主题模型定义

主题模型是一种聚类模型，经常用于处理自然语言问题。经过诸多学者和技术人员的深入研究和不断发展，主题模型已经有了大量的使用案例[32][33]。除了在自然语言领域使用，更广泛运用于其他领域。比如图像识别、数据挖掘研究等等。LDA 算法思想可以理解为：首先在文档中以概率p_{dt}找出某个主题，这里的p_{dt}是在文档中，找到主题的概率。然后再在这个主题下以概率p_{tw}选出某一个词，这里的p_{tw}是在主题下，找到词语的概率。这样就生成了这篇文档的第一个词。不断重复这个过程，就找到了文档中所有出现的词在不同主题下出现的概率。

2.7.2 算法流程

假设我们有 m 个文档，对应第 d 个文档中有N_d个词。即输入如图 2-12。我们的目标是找到每一个文档的主题分布和每一个主题中单词的分布。在 LDA 模型中，我们需要先假定一个主题数目 K，这样所有的分布就都基于 K 个主题展开。具体如图 2-12 所示：

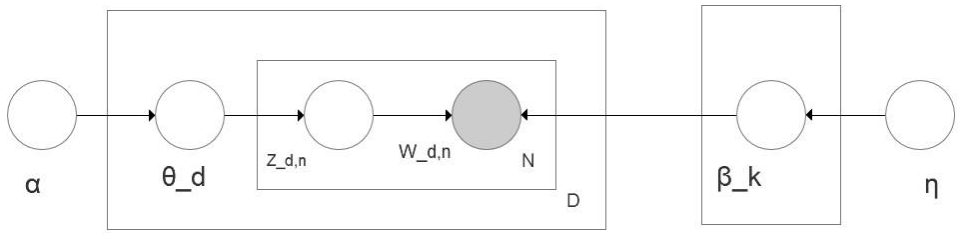

图 2-12 LDA 概率图模型

Fig.2-12 LDA Model

LDA 假设文档主题的先验分布是 Dirichlet 分布，即对于任何文档 d，其主题分布θ_d为：

$$\theta_d = Dirichlet(\vec{\alpha})$$

（2-20）

其中，α为分布的超参数，是一个 K 维向量。LDA 假设主题中词的先验分布是 Dirichlet 分布，即对于任一主题 k，其词分布β_k为：

$$\beta_k = Dirichlet(\vec{\eta})$$

（2-21）

其中，η为分布的超参数，是一个 V 维向量。V 代表词汇表里所有词的个数。对于数据中任一篇文档 d 中的第 n 个词，我们可以从主题分布θ_d中得到它的主题编号Z_{dn}的分布为：

$$Z_{dn} = multi(\theta_d)$$

（2-22）

而对于该主题编号，得到我们看到的词W_{dn}的概率分布为：

$$W_{dn} = multi(\beta_{zdn})$$

（2-23）

通过理解上面的这个模型就能理解 LDA 主题模型。这个模型里，我们有 M 个文档主题的 Dirichlet 分布，而对应的数据有 M 个主题编号的多项分布，这样$(\alpha \rightarrow \theta_d \rightarrow \overrightarrow{Z_d})$就组成了 Dirichlet-multi 共轭，可以使用贝叶斯推断的方法得到基于 Dirichlet 分布的文档主题后验分布。

如果在第 d 个文档中，第 k 个主题的词的个数为：$n_d^{(k)}$，则对应的多项分布

的计数可以表示为

$$\overrightarrow{n_d} = \left(n_d^{(1)}, n_d^{(2)}, \dots n_d^{(k)}\right)$$

（2-24）

利用 Dirichlet-multi 共轭，得到θ_d的后验分布为：

$$\text{Dirichlet}(\theta_d|\vec{\alpha} + \overrightarrow{n_d})$$

（2-25）

同样地，对于主题与词的分布，我们具有 K 个主题与词的 Dirichlet 分布，并且相应地数据具有 K 个主题数的多项分布，这样$\left(\eta \rightarrow \beta_k \rightarrow \overrightarrow{W_{(k)}}\right)$就组成了 Dirichlet-multi 共轭，可以使用贝叶斯推断的方法得到基于 Dirichlet 分布的主题词的后验分布。

如果在第 k 个主题中，第 v 个词的个数为：$n_k^{(v)}$，则对应的多项分布的计数可以表示为

$$\overrightarrow{n_k} = \left(n_k^{(1)}, n_k^{(2)}, \dots n_k^{(v)}\right)$$

（2-26）

利用 Dirichlet-multi 共轭，得到β_k的后验分布为：

$$\text{Dirichlet}(\beta_k|\vec{\eta} + \overrightarrow{n_k})$$

（2-27）

由于不同主题下不同词出现的概率不依赖于文档信息，因此文档主题分布和主题词分布是独立的。这就是 LDA 的基本原理了。

2.7.3 LDA 的应用

虽然 LDA 是一个文本模型，但也可以用来做图像处理，或者用于大规模离散数据分析。Rieping 等学者利用 LDA 模型的原理，分析老年人在家中的行为，并判断其健康程度的变化。从而可以达到检测老人身体健康状况的效果[34]。刘江华等人研究出一种基于 kmeans 算法来聚类和语义相关的文本主题。这种方法避免了传统 LDA 模型的一些缺陷[35]。徐盛等人使用 LDA 主题模型，对高空间分辨

率的遥感影像进行特征提取和分类研究[36]。

在本文中，将使用 LDA 主题模型判断文本的相似性。传统判断两个文档相似性的方法（例如 TF-IDF）是通过查看两个文档一起出现的单词的数量。两个文档中没有出现相同的词汇，但两个文档却很相似。

举个例子，有两个句子分别如下：

“支付宝接入网联。”

“支付宝无法直连银行卡？”

可以看到上面这两个句子只有“支付宝”这个单词相同，但这两个句子是相近的，如果按照 TFIDF 来判断这两个句子可能不相近，因此在判断文档相近度的时候需要考虑到文档的语义，而 LDA 就是挖掘语义的一种比较有效的模型。

2.8 本章小结

本章介绍了 Item-CF 和 User-CF 算法。另外还介绍了基于上下文推荐和基于矩阵分解推荐算法。重点介绍了相关理论知识，包括推荐算法的实现过程、比较各自的优缺点和使用场景。另外，还介绍了常见的关键词提取技术，包括 TF-IDF 和词向量，包括各自的使用场景和各自的优缺点。此外，还重点阐述了卷积神经网络和循环神经网络的工作原理和所实现的组成结构。详细分析了这两种神经网络各自的特点和意义。对比分析了其优缺点和使用场景。另外，本章还详细介绍了 LDA 主题模型的定义和算法流程。本章所介绍的关键技术为后续第三、四章节提出的改进算法提出了坚实的理论基础。

第三章 改进的协同过滤和隐语义模型的推荐设计与实现

本章主要介绍两个推荐系统的设计框架、改进原理和实现过程。这两个推荐系统分别是基于协同过滤的推荐系统和基于隐语义模型的推荐系统。这两个推荐系统用到的数据来源于某企业的社区数据。包括用户的行为数据和文章数据。

3.1 基于协同过滤的推荐系统设计

3.1.1 推荐系统架构

如第二章里 CF 算法介绍的那样，Item-CF 的实现方法：根据打分矩阵里文章的得分，寻找在相同用户中具有相似得分的文章，然后推荐给用户。基于用户的协同过滤的实现方法：根据打分矩阵里用户的得分，寻找在相同物品中具有相似打分情况的用户，然后推荐文章给目标用户。

如图 3-1 所示，推荐系统分为三个部分。部分 A 负责从用户行为数据库或缓存获取用户的行为数据。通过分析不同的行为提取用户行为特征。每条记录包括了用户 id、行为 id(visit_id)、行为内容、时间戳等字段。用户行为数据用到的数据库字段如表 3-1 所示。根据打分规则整理用户行为记录。打分规则如表 3-2 所示。行为记录以字典的结构保存,结构如图 3-2 所示，字典的 Key 是用户 id，Value 是用户行为字典。在用户行为字典中，Key 是 visit_id, Value 是行为的分值。

部分 B 负责从文章数据库获取文章数据。采集的文章相关字段包括文章 id、游览 id(visit_id)和时间戳。通过 join 这里获得的 visit_id 和部分 A 中获得的 visit_id 来构建用户-文章行为矩阵。表 3-3 展示的是整理好的用户-文章行为矩阵，其中每一列的索引值分别是用户 id、文章 id、分值和时间戳。根据整理好的用户-文章打分矩阵，计算用户-用户相似度和文章-文章相似度。

如图 3-3 所示为文章相近度的计算结果。然后，进行 User-CF 推荐和 Item-CF(此处的 item 为文章。故也可称为 Article-CF)推荐。

部分 C 根据得出的初步推荐结果，按照 score 降序排列。取 top10 作为推荐结果。

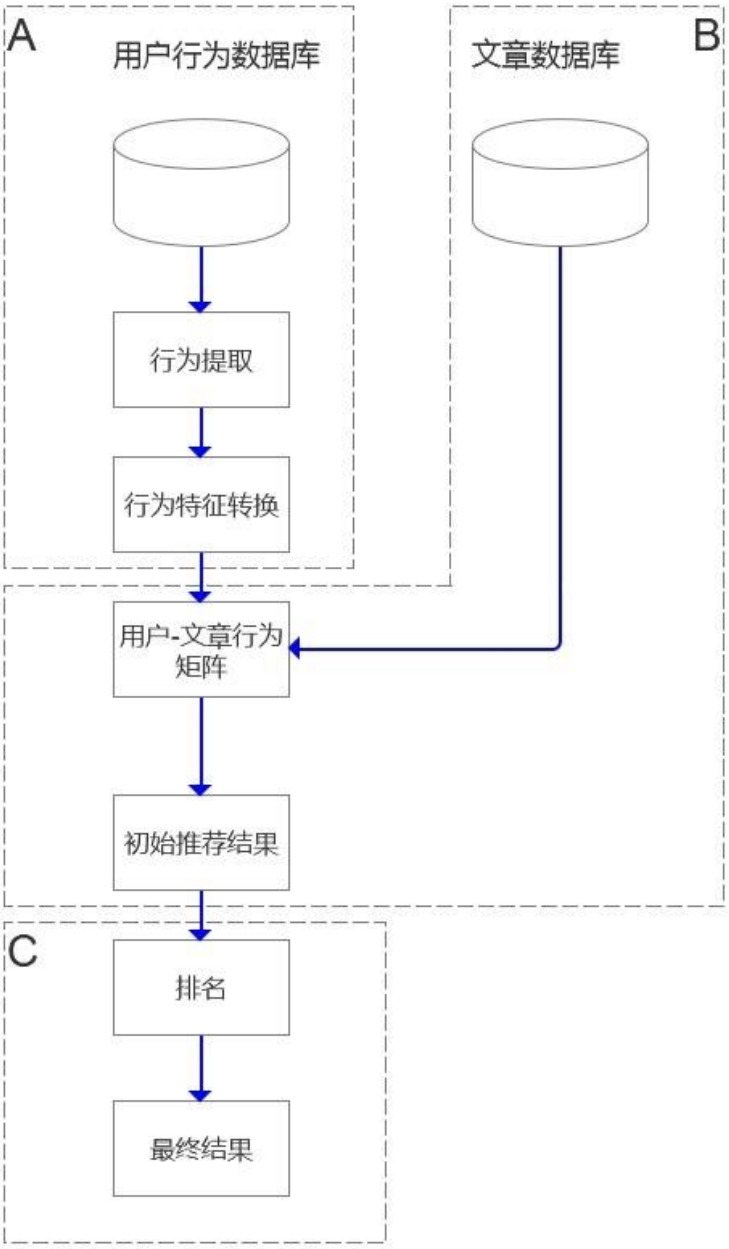

图 3-1 协同过滤推荐系统架构

Fig.3-1 Collaborative Filtering Recommendation System Architecture

表3-1 数据库原始数据字段

Table3-1 Raw Data from Database

字段名称	值
用户 id	uid
用户行为	点击/游览
发生页面名称	详情页/列表页
停留页面时长	（s）

是否分享	1/0
是否点赞	1/0

表3-2 用户行为打分

Table3-2 User Behavior Scores

分值	页面	行为	描述
1	列表页	游览	查看行为
2	详情页	游览	查看行为
3	详情页	游览、停留	停留>15 秒
4	详情页	分享	微信/QQ 分享
5	详情页	点赞	点击行为

字典的Key	字典的Value
用户id	用户行为记录
uid1	{visit_id11:score11, visit_id12:score12....}
uid2	{visit_id21:score21, visit_id22:score22....}
...	.....
uid_n	{visit_id_n1:score_n1, visit_id_n2:score_n2....}

图 3-2 用户行为数据结构

Fig.3-2 User Behavior Data Structure

表3-3 用户-文章行文矩阵

Table3-3 User-Article Behavior Matrix

Index	User_id	Article_id	ratings	timestamp
0	1	1	5	1392799358
1	1	2	3	1392809857
2	1	3	4	1393048286
3	1	4	3	1392879880
4	1	6	5	1393113570

```
计算相似度矩阵...
计算完成.
相似度矩阵样例: (item-item)
[[ 1.      0.296   0.279   0.388   0.252   0.114   0.518   0.41    0.416   0.199]
 [ 0.296   1.      0.177   0.405   0.211   0.099   0.331   0.31    0.207   0.152]
 [ 0.279   0.177   1.      0.275   0.118   0.104   0.311   0.125   0.207   0.121]
 [ 0.388   0.405   0.275   1.      0.265   0.091   0.411   0.391   0.357   0.219]
 [ 0.252   0.211   0.118   0.265   1.      0.016   0.28    0.214   0.202   0.031]
 [ 0.114   0.099   0.104   0.091   0.016   1.      0.128   0.065   0.164   0.139]
 [ 0.518   0.331   0.311   0.411   0.28    0.128   1.      0.342   0.43    0.279]
 [ 0.41    0.31    0.125   0.391   0.214   0.065   0.342   1.      0.364   0.166]
 [ 0.416   0.207   0.207   0.357   0.202   0.164   0.43    0.364   1.      0.25 ]
 [ 0.199   0.152   0.121   0.219   0.031   0.139   0.279   0.166   0.25    1.    ]]
```

图 3-3 文章-文章相似度计算结果

Fig.3-3 Article-Article Similarity

以余弦相似度计算文章间的相似度，使用基于文章的协同过滤为例的协同过滤计算过程如下：

```
def cal_similarity(ratings, kind, epsilon=1e-9):
    '''calculating Similarity'''
    '''epsilon: avoid Eception'''
    if kind == 'user':
        sim = ratings.dot(ratings.T) + epsilon
    elif kind == 'item':
        sim = ratings.T.dot(ratings) + epsilon
    norms = np.array([np.sqrt(np.diagonal(sim))])
return (sim / norms / norms.T)

print('calculating Similarity Matrix...')
user_similarity = cal_similarity(ratings, kind='user')
item_similarity = cal_similarity(ratings, kind='item')
print('calculating completed.')
print('Similarity Matrix: (item-item)')
print(np.round_(item_similarity[:10,:10], 3))

def predict_item-CF(user, article, k=100):
    '''item-based 协同过滤算法,预测 rating'''
    nzero = ratings[user].nonzero()[0]
    prediction = ratings[user, nzero].dot(article_similarity[article, nzero])\
                    / sum(article_similarity[item, nzero])
    return prediction

print('载入测试集...')
```

```
test_df = pd.read_csv(testset_file, sep='\t', names=names)
test_df.head()
predictions = []
targets = []
print('测试集大小为 %d' % len(test_df))
print('采用 item-based 协同过滤算法进行预测...')
for row in test_df.itertuples():
    user, article, actual = row[1]-1, row[2]-1, row[3]
    predictions.append(predict_item-CF(user, item))
    targets.append(actual)

print('测试结果的 rmse 为 %.4f' % rmse(np.array(predictions), np.array(targets)))
```

3.1.2 相似度计算方法

欧式距离相似度

欧几里德距离是最常用的距离计算公式之一，可用于测量多维空间中点之间的绝对距离。

余弦相似度

在一个 mxn 的评价矩阵中，如果有 m 个用户，n 个项目/文章，两个用户 u,v 之间的相似度被定义为矩阵的第 u 行和第 v 行的两个 m 维向量之间的角度的余弦，如公式 3-1 所示。

$$w(u,v) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \times \|\vec{v}\|} = \frac{\sum_{k=1}^{m} u_k \cdot v_k}{\sqrt{\sum_{k=1}^{m} u_k^2} \sqrt{\sum_{k=1}^{m} v_k^2}} \tag{3-1}$$

因为在实际应用中，不同用户的行为习惯不同造成不同用户的得分不同，使得不同用户 A，用户 B 对同一文章 i 具有不同的行为。所以需要采用去中心化的方法来统一评分标准。在这种情况下常常需要使用皮尔森相似度。

欧几里德距离相似度和余弦相似度的区别在于欧几里德距离相似度更关注空间中直线到直线的距离，而余弦相似度涉及是空间中两条直线间的夹角。

余弦距离更多地用于使用用户对内容评分来区分兴趣的相似度和差异，同时用于校正用户间度量标准不统一的问题（因为余弦距离对绝对数值不敏感）。

皮尔森相似度

相比之前两种计算相似度的方法，皮尔森相似度的应用范围更加广泛。变量的取值范围不统一时，其他方法往往无法得出准确得相似度。但是，皮尔森相似度便能克服这个难题。即使变量得量纲不同，通过标准化计算，一样可以将变量缩放到统一的变化幅度内。用户之间的相似度的定义如公式 3-2 所示，其中，

w(a,b)：用户 a，b 之间的皮尔森相似度；

r_{ai}：用户 a 对文章 i 的访问历史；

r_{bi}：用户 b 对文章 i 的访问历史；

I(a)：用户 a 所有访问过的文章的列表；

I(b)：用户 b 所有有访问过的文章列表；

$\overline{r_a}$：用户 a 在所有文章上访问的评分均值；

$\overline{r_b}$：用户 b 在所有文章上访问的评分均值；

$$w(a,b)=\frac{\sum_{i\epsilon I(a)\cap I(b)}(r_{ai}-\overline{r_a})\cdot(r_{bi}-\overline{r_b})}{\sqrt{\sum_{i\epsilon I(a)\cap I(b)}(r_{ai}-\overline{r_a})^2}\sqrt{\sum_{i\epsilon I(a)\cap I(b)}(r_{bi}-\overline{r_b})^2}} \quad (3\text{-}2)$$

相似地，在基于物品的方法中，项目之间的相似度定义如公式 3-3 所示，其中:

w(p,q):文章 p，q 之间皮尔森相相似度；

U(p)：所有用户在文章 p 上评分列表；

U(q)：所有用户在文章 q 上评分列表；

r_{up}：用户 u 对文章 p 的评分；

r_{uq}：用户 u 对文章 q 的评分；

$\overline{r_p}$：所有用户在文章 p 上的评分均值；

$\overline{r_q}$：所有用户在文章 q 上的评分均值；

$$w(p,q)=\frac{\sum_{p\in I(p)\cap I(q)}(r_{up}-\overline{r_p})\cdot(r_{uq}-\overline{r_q})}{\sqrt{\sum_{i\epsilon I(p)\cap I(q)}(r_{up}-\overline{r_p})^2}\sqrt{\sum_{i\epsilon I(p)\cap I(q)}(r_{uq}-\overline{r_q})^2}}$$

（3-3）

评分矩阵中的行代表用户向量，列代表物品（文章）向量，计算用户之间的相似度实质上就是计算打分矩阵中行向量之间的相似度，而计算物品（文章）之间的相似度实质上就是计算打分矩阵中列向量之间的相似度。

Jaccard 相似度

Jaccard 相似度主要用于计算两个集合 A,B 之间，交集的大小与两个集合并集的大小的比值。如果比较 X 与 Y 的 Jaccard 相似度，只比较X_n和Y_n中相同的个数，公式如下：

$$\text{Jaccard}(X_n, Y_n) = \frac{X_n \cap Y_n}{X_n \cup Y_n}$$

（3-4）

3.1.3 改进的协同过滤设计与实现

使用传统的协同过滤方法实现后，我发现推荐结果存在这样几个问题：

1. 用户具有自己的游览习惯，比如，有些用户习惯对于游览过的文章给予得分都很高，并且遇到文章都习惯性地给予点赞/好评。而另一些用户对所有浏览过的文章给予得分都很低。如此看来，不同用户的使用习惯会造成得分矩阵平均水平的参差不齐，这样便带来了噪声，使得预测得分的结果既不合理也不准确。
2. 另外，从文章的角度出发，有些冷门的文章，基本上要么没有游览行为记录，即使有也多是低分；而另外一些热门文章，由于长期占据网页首页，容易被用户点击/游览，故得分较高。
3. 全量的用户数量和文章数量都非常庞大，如果对其全部进行计算和训练的话，耗费的资源非常巨大。但是相似度越是往后排的文章或者用户，对预测准确率却没有很大的提升。

对基础的协同过滤算法进行了改进。方法如下：

[1] 计算全量用户-文章的得分情况（剔除未打分的值），并计算均值；

[2] 计算归约到用户的得分情况（剔除未打分的值），并计算均值；

[3] 计算归约到文章的得分情况（剔除未打分的值），并计算均值；

[4] 结合基于文章的协同过滤和基于用户的协同过滤算法，用皮尔森相似度将得分做归一化处理；

[5] 在计算文章-文章，用户-用户相似度时，选取 top20。

改进后的协同过滤算法的实现流程如图 3-5 所示，以改进后的基于物品的系统过滤算法为例。算法分以下几个步骤：

步骤一：剔除零得分的文章。在这一步骤中，制定每篇文章得分的 baseline(基准)。这个 baseline 取决于三方面的因素：所有用户对这篇文章打分均值、目标用户打分均值和所有文章得分的均值。在 baseline 的基础上，剔除目标用户零得分的文章。

步骤二：寻找和目标用户历史上打过分的文章相似的文章。使用皮尔森相似度计算归一化后的评分矩阵。

步骤三：由于已经做过归一化操作，所以需要将得分幅度进行缩放。当得分大于 5 时，将得分修改为 5。当得分小于 1 时，将得分修改为 1。

步骤四：根据相似度矩阵计算的结果，获取 top K 相似的文章作为推荐结果。剩下的推荐项丢弃。

改进的基于用户的协同过滤算法采用与上述相同的算法。只不过在计算相似度时，寻找相似用户打过分的文章。然后取 top K 推荐给目标用户。故这里就不重复叙述了。

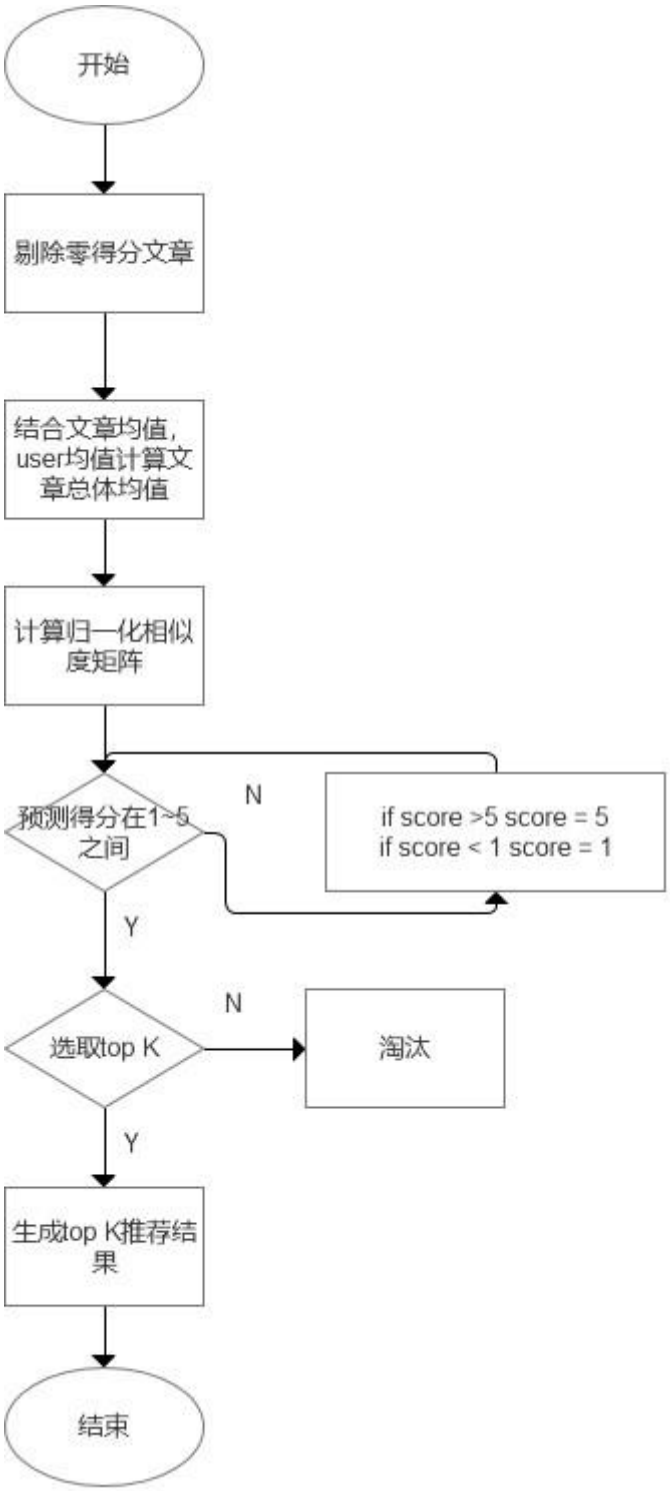

图 3-5 改进后的协同过滤算法流程

Fig.3-5 Improved Collaborative Filtering Flow

改进后的协同过滤算法编程实现如下：其中，item 代指文章（article）。

```
def predict_itemCF_base(user, item, k=100):
    '''结合 base 的 item-basedCF 算法,预测 rating'''
    nzero = ratings[user].nonzero()[0]
    base = article_mean + user_mean[user] - all_mean
    prediction = (ratings[user, nzero] - base[nzero]).dot(article_similarity[item, nzero])\
                / sum(article_similarity[item, nzero]) + base[item]
return prediction

print('载入测试集...')
test_df = pd.read_csv(testset_file, sep='\t', names=names)
test_df.head()
predictions = []
targets = []
print('测试集大小为 %d' % len(test_df))
print('采用结合 baseline 的 item-item 协同过滤算法进行预测...')
for row in test_df.itertuples():
    user, article, actual = row[1]-1, row[2]-1, row[3]
    predictions.append(predict_itemCF_base(user, item))
    targets.append(actual)

print('测试结果的 rmse 为 %.4f' % rmse(np.array(predictions), np.array(targets)))

print('------ Top-k 协同过滤(item-based + baseline)------')
def predict_topkCF(user, item, k=10):
    '''top-k CF 算法,以 item-based 协同过滤为基础，结合 baseline,预测 rating'''
    nzero = ratings[user].nonzero()[0]
    base = item_mean + user_mean[user] - all_mean
    choice = nzero[article_similarity[item, nzero].argsort()[::-1][:k]]
    prediction = (ratings[user, choice] - base[choice]).dot(article_similarity[item, choice])\
                / sum(article_similarity[item, choice]) + base[item]
    if prediction > 5: prediction = 5
    if prediction < 1: prediction = 1
    return prediction

print('载入测试集...')
test_df = pd.read_csv(testset_file, sep='\t', names=names)
test_df.head()
predictions = []
targets = []
print('测试集大小为 %d' % len(test_df))
print('采用 top K 协同过滤算法进行预测...')
k = 20
print('选取的 K 值为%d.' % k)
```

```
for row in test_df.itertuples():
    user, item, actual = row[1]-1, row[2]-1, row[3]
    predictions.append(predict_topkCF(user, item, k))
    targets.append(actual)

print('测试结果的 rmse 为 %.4f' % rmse(np.array(predictions), np.array(targets)))
print

def predict_biasCF(user, item, k=100):
    '''结合基线算法的 item-based CF 算法,预测 rating'''
    nzero = ratings[user].nonzero()[0]
    base = article_mean + user_mean[user] - all_mean
    prediction = (ratings[user, nzero] - base[nzero]).dot(article_similarity[item, nzero])\
                / sum(article_similarity[item, nzero]) + base[item]
    if prediction > 5:
        prediction = 5
    if prediction < 1:
        prediciton = 1
    return prediction

print('载入测试集...')
test_df = pd.read_csv(testset_file, sep='\t', names=names)
test_df.head()
predictions = []
targets = []
print('测试集大小为 %d' % len(test_df))
print('采用结合 baseline 的 item-based 协同过滤算法进行预测...')
for row in test_df.itertuples():
    user, item, actual = row[1]-1, row[2]-1, row[3]
    predictions.append(predict_biasCF(user, item))
    targets.append(actual)

print('测试结果的 rmse 为 %.4f' % rmse(np.array(predictions), np.array(targets)))
print

print('------ Top-k 协同过滤(item-based + baseline)------')
def predict_topkCF(user, item, k=10):
    '''top-k CF 算法,以 item-based 协同过滤为基础，结合 baseline,预测 rating'''
    nzero = ratings[user].nonzero()[0]
    base = item_mean + user_mean[user] - all_mean
    choice = nzero[article_similarity[item, nzero].argsort()[::-1][:k]]
    prediction = (ratings[user, choice] - base[choice]).dot(article_similarity[item, choice])\
                / sum(article_similarity[item, choice]) + base[item]
```

```
    if prediction > 5: prediction = 5
    if prediction < 1: prediction = 1
    return prediction

print('载入测试集...')
test_df = pd.read_csv(testset_file, sep='\t', names=names)
test_df.head()
predictions = []
targets = []
print('测试集大小为 %d' % len(test_df))
print('采用 top K 协同过滤算法进行预测...')
k = 20
print('选取的 K 值为%d.' % k)
for row in test_df.itertuples():
    user, item, actual = row[1]-1, row[2]-1, row[3]
    predictions.append(predict_topkCF(user, item, k))
    targets.append(actual)

print('测试结果的 rmse 为 %.4f' % rmse(np.array(predictions), np.array(targets)))
print

def cal_similarity_peason_norm(ratings, kind, epsilon=1e-9):
    '''采用归一化的指标:Pearson correlation coefficient'''
    if kind == 'user':
        # 对同一个 user 的打分归一化
        rating_user_diff = ratings.copy()
        for i in range(ratings.shape[0]):
            nzero = ratings[i].nonzero()
            rating_user_diff[i][nzero] = ratings[i][nzero] - user_mean[i]
        sim = rating_user_diff.dot(rating_user_diff.T) + epsilon
    elif kind == 'item':
        # 对同一个 item 的打分归一化
        rating_item_diff = ratings.copy()
        for j in range(ratings.shape[1]):
            nzero = ratings[:,j].nonzero()
            rating_item_diff[:,j][nzero] = ratings[:,j][nzero] - item_mean[j]
        sim = rating_item_diff.T.dot(rating_item_diff) + epsilon
    norms = np.array([np.sqrt(np.diagonal(sim))])
    return (sim / norms / norms.T)

print('计算归一化的相似度矩阵...')
user_similarity_norm = cal_similarity_peason_norm(ratings, kind='user')
article_similarity_norm = cal_similarity_peason_norm(ratings, kind='item')
```

```
print('计算完成.')
print('相似度矩阵样例: (item-item)')
print(np.round_(item_similarity_norm[:10,:10], 3))

def predict_norm_CF(user, item, k=20):
    '''baseline + item-based + 皮尔森归一化'''
    nzero = ratings[user].nonzero()[0]
    base = article_mean + user_mean[user] - all_mean
    choice = nzero[article_similarity_norm[item, nzero].argsort()[::-1][:k]]
    prediction = (ratings[user, choice] - base[choice]).dot(article_similarity_norm[item, choice])\
                / sum(article_similarity_norm[item, choice]) + baseline[item]
    if prediction > 5: prediction = 5
    if prediction < 1: prediction = 1
    return prediction

print('载入测试集...')
test_df = pd.read_csv(testset_file, sep='\t', names=names)
test_df.head()
predictions = []
targets = []
print('测试集大小为 %d' % len(test_df))
print('采用归一化矩阵方法，结合其它 trick 进行预测...')
k = 13
print('选取的 K 值为%d.' % k)
for row in test_df.itertuples():
    user, article, actual = row[1]-1, row[2]-1, row[3]
    predictions.append(predict_norm_CF(user, item, k))
    targets.append(actual)

print('测试结果的 rmse 为 %.4f' % rmse(np.array(predictions), np.array(targets)))
print
```

改进意义：零得分的文章只能表明用户没有访问过，但是并不意味着用户不感兴趣。所以，剔除 0 得分的文章，可以去除零得分对打分矩阵的噪声。另外，由于不同用户有不同的习惯，有的习惯打地很高，有的习惯打得很低。所以基于用户打分做归一化，可以去除因用户差异造成的打分偏差。同理，对文章做归一化也能够去除文章间差异造成的影响。

3.2 基于增量隐语义模型的推荐系统设计与实现

3.2.1 问题提出

如第二章中介绍的那样，隐语义模型的核心思想是：通过发现隐藏因子，试图将用户-物品评分矩阵分解为用户-隐因子矩阵与物品-隐因子矩阵的乘积。在本文采集的数据中，用户 id 和文章 id 构成一个巨大的矩阵。每一行就是一个用户的记录，每一列就是一个文章 id。由用户 id 和文章 id 构成下标的相应位置就是得分值。但是在计算这个庞大的矩阵时发现如下问题：

1. 打分矩阵非常稀疏，填充率大约在 5%左右。大量文章没有得分，这对文章而言就存在冷启动的问题。无法计算文章-文章的相似性；
2. 数据量非常庞大。海量的数据无法一次载入内存用于训练。对服务器造成巨大压力；
3. 另外，由于需要不断从用户行为数据里更新历史纪录。数据每天不断增加，是否能找到一些增量训练的方式去不断持续迭代更新模型？

3.2.2 基于隐语义模型的增量训练模型

根据上一节提出的几个问题，结合隐语义模型的算法思想。建立了基于隐语义模型的增量训练模型，模型结构如图 3-7 所示。最后预测得出的用户 u 对文章 i 的得分是$bias_{global}$,$bias_{user[u]}$,$bias_{item[i]}$,和

$\langle embedding_{user[u]}, embedding_{item[i]}\rangle$之和。

其中，

$bias_{global}$：整个矩阵的增量更新打分

$bias_{user[u]}$：用户 u 的增量更新打分

$bias_{item[i]}$：文章 i 的增量更新打分

$\langle embedding_{user[u]}, embedding_{item[i]}\rangle$ 增量提交的用户 u 对文章 i 的打分值。这部分可以看作是用户 u-factor 矩阵和 factor-文章 i 矩阵的乘积。这里的 factor 便是 k 维隐因子。

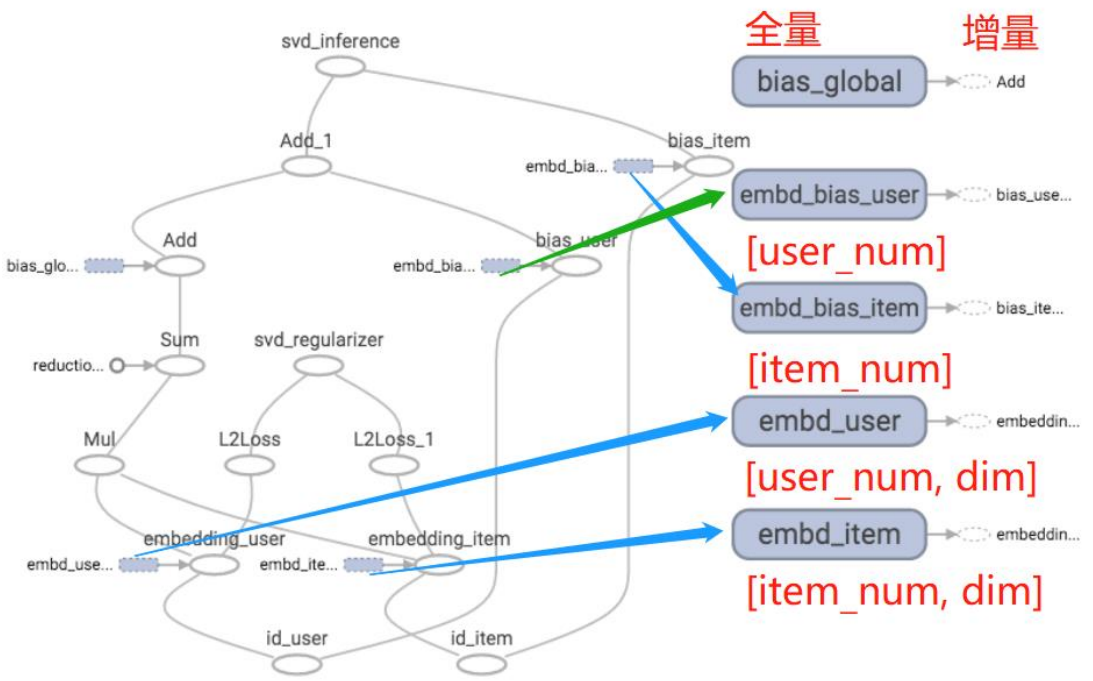

图 3-7 隐语义增量训练模型

Fig.3-7 LFM Increment Training Model

在图 3-7 的右半边已经标注了全量更新 tensor(张量)及其 shape(形状，可以可以看出其数据维度)、增量更新 tensor(张量)。embd_bias_user表示全量的用户打分数据，其 shape 为[*user_num*](用户的个数)。其增量子集为同一行的bias_user。而bias_user是在增量更新时，通过在embd_bias_user搜索user_id获得的。同理，embd_bias_item表示全量的文章被打分数据，其 shape 为[*item_num*](文章的个数)。其增量子集为同一行的bias_item。而bias_item是在增量更新时，通过在embd_bias_item搜索item_id获得的。

embd_user的 shape 为[*user_num*， dim]。其中，dim 表示隐因子的维数。正如隐语义矩阵分解模型中那样，embd_user表示全量 user(用户)-隐因子（k）的关系矩阵。embedding_user表示增量 user(用户)-隐因子（k）的关系矩阵。同理，embd_item的 shape 为[*item_num*， dim]。其中，embd_item表示全量 item(文章)-隐因子（k）的关系矩阵。embedding_item表示增量 item(文章)-隐因子（k）的关系矩阵。通过embedding_user和embedding_item的矩阵乘法，增量更新用户-文章打分矩阵。这样一来，避免计算结果受到用户-文章打分矩阵中稀疏性带来噪声的影响。这样便解决了上述问题 1。另外，在这个模型中，数据

以一个 batch 一个 batch 的方式提交，而非一下子将全量的数据倒入内存计算。这样对服务器内存的压力也会减少很多。而且，每天产生的增量数据将以增量迭代优化的方式进行训练，如此便解决上述问题 2 和问题 3。

3.2.3 增量更新算法流程

我们需要进行最小化损失函数计算如公式 3-5 所示。此处使用的是 L2 正则化，也称为权重衰减。学习率为 0.001。通过对目标函数添加一个参数范数惩罚来限制模型的学习能力。这是一种预防过拟合的有效措施。

$$\sum_{u,i}|y_{pred[u,i]} - y_{true[u,i]}|^2 + \lambda\left(|embedding_{user[u]}|^2 + |embedding_{item[i]}|^2\right) \quad (3\text{-}5)$$

隐语义模型算法流程如下所示：

1. 计算全量的用户数据张量embd_bias_user和全量的文章得分张量$embd_bias_item$。
2. 使用更新的user_id作为索引值，从全量的用户打分张量中计算增量的用户打分张量bias_user。
3. 。使用更新的item_id作为索引值，从全量的文章得分张量中计算增量的文章得分张量bias_item。
4. 将用户-文章打分矩阵分解为用户-隐因子矩阵embd_user和隐因子-文章矩阵embd_item。这里得到的分解后的矩阵都是全量的矩阵。
5. 使用更新的user_id作为索引值，从全量的用户-隐因子矩阵中计算增量的用户-隐因子矩阵embedding_user。
6. 使用更新的item_id作为索引值，从全量的隐因子-文章矩阵中计算增量的隐因子-文章矩阵embedding_item。
7. 将global_bias、bias_user、bias_item 相加，并使用 L2 损失计算损失函数。

隐语义模型实现过程的编程实现如下：

```
import tensorflow as tf

# 使用矩阵分解搭建的网络结构
```

```
def inference_lfm(user_batch, item_batch, user_num, item_num, dim=5, device="/cpu:0"):
    #使用 CPU
    with tf.device("/cpu:0"):
        # 初始化几个 bias 项
        global_bias = tf.get_variable("global_bias", shape=[])
        w_bias_user = tf.get_variable("embd_bias_user", shape=[user_num])
        w_bias_item = tf.get_variable("embd_bias_item", shape=[item_num])
        # bias 向量
        bias_user = tf.nn.embedding_lookup(w_bias_user, user_batch, name="bias_user")
        bias_item = tf.nn.embedding_lookup(w_bias_item, item_batch, name="bias_item")
        w_user = tf.get_variable("embd_user", shape=[user_num, dim],
                                 initializer=tf.truncated_normal_initializer(stddev=0.02))
        w_item = tf.get_variable("embd_item", shape=[item_num, dim],
                                 initializer=tf.truncated_normal_initializer(stddev=0.02))
        # user 向量与 item 向量
        embd_user = tf.nn.embedding_lookup(w_user, user_batch, name="embedding_user")
        embd_item = tf.nn.embedding_lookup(w_item, item_batch, name="embedding_item")
    with tf.device(device):
        # 按照实际公式进行计算
        # 先对 user 向量和 item 向量求内积
        infer = tf.reduce_sum(tf.multiply(embd_user, embd_item), 1)
        # 加上几个偏置项
        infer = tf.add(infer, global_bias)
        infer = tf.add(infer, bias_user)
        infer = tf.add(infer, bias_item, name="lfm_inference")
        # 加上正则化项
        regularizer = tf.add(tf.nn.l2_loss(embd_user), tf.nn.l2_loss(embd_item),
name="lfm_regularizer")
    return infer, regularizer

# 迭代优化部分
def optimization(infer, regularizer, rate_batch, learning_rate=0.001, reg=0.1, device="/cpu:0"):
    global_step = tf.train.get_global_step()
    assert global_step is not None
    # 选择合适的 optimizer 做优化
    with tf.device(device):
        cost_l2 = tf.nn.l2_loss(tf.subtract(infer, rate_batch))
        penalty = tf.constant(reg, dtype=tf.float32, shape=[], name="l2")
        cost = tf.add(cost_l2, tf.multiply(regularizer, penalty))
        train_op = tf.train.AdamOptimizer(learning_rate).minimize(cost,
global_step=global_step)
    return cost, train_op
```

3.3 本章小结

在本章中，我们重点介绍了基于协同过滤的推荐系统的设计与实现，并提出了改进的方法。通过使用皮尔森归一化的方法，剔除了零得分文章引入的噪声，避免了用户使用习惯差异导致的影响。此外，还详细介绍了基于增量更新的隐语义模型的推荐系统的设计和实现过程。具体阐述了改进的原因和解决的问题。详细介绍了改进算法的步骤。这两种推荐算法的实现结果和性能将在第五章中介绍。

第四章 基于神经网络与 LDA 的推荐系统实现

4.1 问题提出

推荐系统常常会遇到如下几个难题:

1. 冷启动问题

推荐系统主要根据用户的历史行为数据预测用户的兴趣爱好和未来行为。因此，大量的用户历史行为数据对推荐系统而言是非常重要的组成部分。当推荐系统中出现新用户时，系统里没有用户历史数据。于是便无法根据历史行为预测用户偏好。当出现新物品时，系统里没有新物品被关注/订阅/购买等历史记录。因此，不可能通过历史行为衡量物品间的相似性。

2. 同质化问题

用户的偏好多种多样，但其表达的方式却十分有限的，而基于上下文过滤的推荐方法只推荐与当前偏好匹配的文章/物品。因此，推荐结果同质化现象严重。对于用户没有表现出偏好但实际上感兴趣的物品却难以得到推荐。

3. 数据疏落性

随着推荐系统中用户数量和文章/物品数量的增加，评分方阵变得越来越稀疏。数据疏落性会带来计算过程中的偏差，因此取得得推荐结果可能是不准确的。

总之，为了进一步提高推荐准确率，解决上述问题。所采用的方法总结如下：

由于打分矩阵的密度占比比较低，无法覆盖所有的文章和用户。因此，除了打分矩阵的数据以外，还增加了用户在社区里发帖/回复的文本数据。根据用户的文本数据，计算用户间相似度，并挖掘用户兴趣点、喜好等。这样可以从不同维度进行推荐，以弥补稀疏矩阵所造成的影响。

增加推荐数据的维度。从用户查看、点击、停留页面时间等游览页面的历史行为，以及用户在社区论坛里的发帖/回复的文本数据进行分析。相比之前只使用一种推荐算法，本章将引入组合推荐策略。综合上述两方面的推荐结果，减轻推荐同质化问题。

对于存在冷启动问题的文章，可以采用最新最热的文章作为推荐的补充策略。

4.2 推荐系统结构

推荐系统的整体结构如图 5-1 所示。推荐功能主要有两部分的数据来源。一部分为用户评论数据。评论数据包括了用户在论坛的发帖内容、回复内容等文本数据。它们被用来挖掘用户的兴趣点和偏好和计算用户间的相似度。最后输出相似用户列表；第二部分为用户行为数据。用户行为数据包含了用户点击页面、游览页面、点赞等用户行为。通过用户行为定义规则转化为打分矩阵。使用第一部分输出的相似用户列表中的用户 id 作为索引值，在打分矩阵中寻找相应用户 id 下的文章 id,形成文章 id 候选集。通过计算目标用户打分过的文章和候选集里文章的相似度，取相似度 top K 的文章作为推荐结果发送给用户。上述方法是针对有历史行为的用户。对于新用户而言，使用补充策略，如图 4-1 中的右边部分，使用平台上热门 top K 的文章作为推荐结果发送给用户。

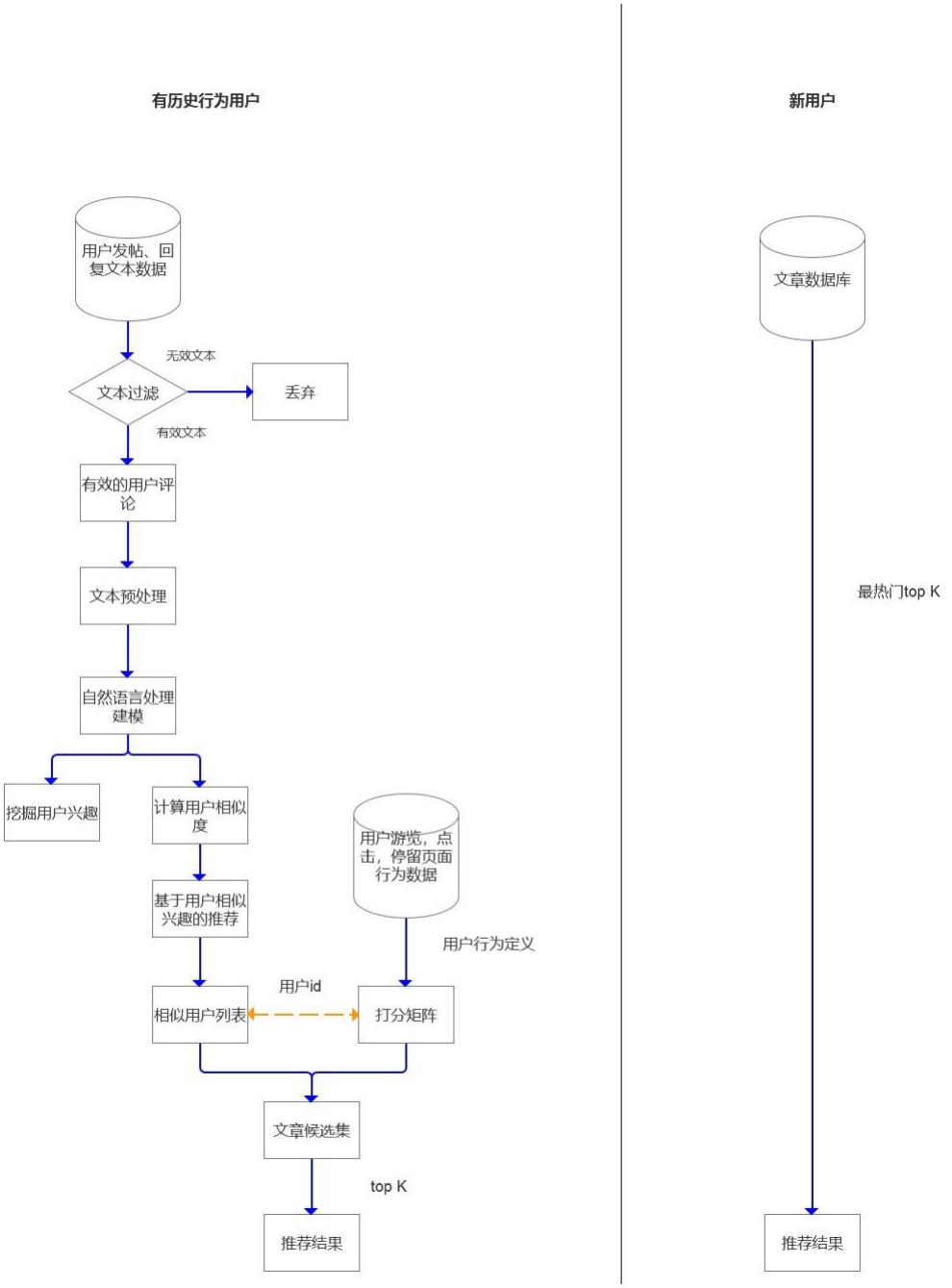

图 4-1 推荐系统结构

Fig.4-1 Recommendation System Diagram

4.3 文本过滤方法

数据库中存放了所采集的用户历史行为数据包括用户 id、发帖内容、回复内容、发帖时间和回复时间等字段。在这一节中，将会用到用户发帖内容和回复

内容。通过收集用户的言论和评语建立数据集，用来分析用户的兴趣偏好。但是原始的数据往往存在大量冗余或者无用的信息。类似用户回复：“我来占个坑”，“沙发”之类的评论。这些数据如果不及时剔除，一方面将会增加模型建立时的噪声，影响模型的准确率；另一方面，也会增加模型训练时的压力，消耗 GPU 资源。因此，文本过滤是建模前不可忽视的环节。

在本章中采用了文本分类的方法对原始用户评论数据进行分类。此处作为二分类来处理。比如 0 表示不是垃圾评论。1 表示垃圾评论需要过滤。文本过滤流程如图 4-2 所示。

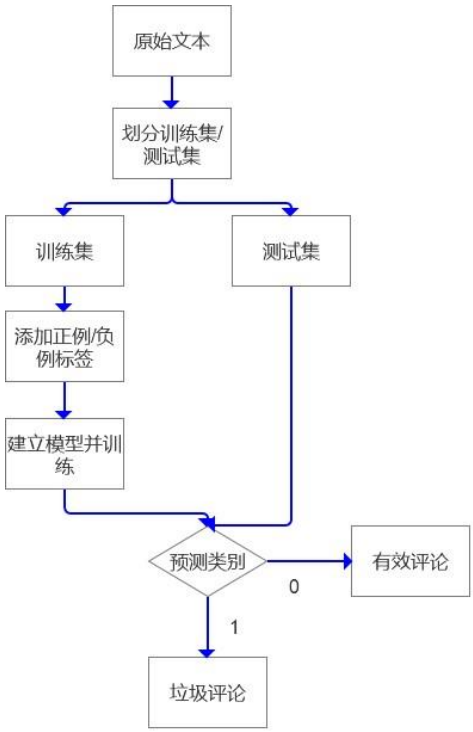

图 4-2 文本过滤流程

Fig.4-2 Text Filtering Flow

一个文本分类问题主要由特征工程和分类器两部分组成[37][38][39]。

在机器学习领域，特征工程就是将数据转化为信息的过程。因此特征工程完成的好坏程度直接影响分类结果的最大限度。而分类器则是在训练迭代得过程中不断逼近这个最大限度。通常，特征工程与特征任务结合紧密，因此不具备很强的通用性，需要理解特定的需求，并对特定的场景做出判断。处理文本的特征工程可以分为文本预处理、特征提取、文本表示三个步骤。

1. 文本预处理

文本预处理主要包括词汇切分和去除停用词两个部分。本文中使用中文分词工具 jieba 来实现，如图 4-3 所示。Jieba 可以用三种不同的模式对句子进行分词。。

停用词通常指文本中一些高频词如“的”、“地”、“得”、“非常”等。这些词在文本中没有实际意义，常见做法是采集这戏无实际意义的词，并贮存在一个文件中。在提取特征时，过滤掉这些词。

2. 提取特征

提取特征包括了选择特征和计算特征权重。选择特征的基本流程是根据评分标准对特征进行排序，从中选择得分 top K 的特征。本文中使用基于 TF-IDF 权重的方法提取特征权重。在待提取的文本中，根据 TF-IDF 权重最大的关键词，默认为 20 来提取。TF-IDF 的原理已经在 2.4.1 节中论述。此处不再重复论述。

```
import jieba.analyse as analyse
import pandas as pd
df = pd.read_csv("./data/technology_news.csv", encoding='utf-8')
df = df.dropna()
lines=df.content.values.tolist()
content = "".join(lines)
print "  ".join(analyse.extract_tags(content, topK=30, withWeight=False, allowPOS=()))
```

```
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.251 seconds.
Prefix dict has been built succesfully.
```

```
用户  2016  互联网  手机  平台  人工智能  百度  2017  智能  技术  数据  360  服务  直播  产品  企业  安全  视频  移动  应用  网络  行业  游戏  机器人  电商  内容  中国  领域  通过  发展
```

图 4-3 jieba 分词工具示例

Fig.4-3 Jieba Text Segmentation Tool Example

3. 文本表示

本文中用 gesim 实现词袋模型构建字典 ，字典的 key 是每个字的索引，value 是出现的次数。

4.4 过滤模型设计与实现

在完成文本预处理后，需要对文本过滤模型进行建模。本文分别使用卷积神经网络（CNN）和循环神经网络（RNN）来实现建模。

4.4.1 卷积神经网络过滤模型设计与实现

卷积神经网络分类原型

使用卷积神经网络实现文本过滤流程如下：首先使用 jieba 工具对原始文本做预处理。其中包括分词和去除停用词，然后构建数据集、划分训练集和测试集。训练集用于训练模型，所以需要添加正例、负例标签。接着，使用 CNN 建立模型，最后在测试集上进行分类。

CNN 是一种深度学习模型卷积过程包含了其局部敏感得特性和共享权重得特性。其网络结构原型如图 4-4 所示。首先将文本转化为词向量，这样，我们就可以得到一个词表映射矩阵。接着，将文本映射成矩阵形式。CNN 一般后面接上若干个卷积和池化层，然后进入全连接层。经过全连接层后进入分类器层，该层根据任务的不同而不同，即损失函数的不同所导致，可以是 Softmax 回归、逻辑回归等组件构成。由于 CNN 局部敏感的特性，在处理短文本分类时，效果比较明显。

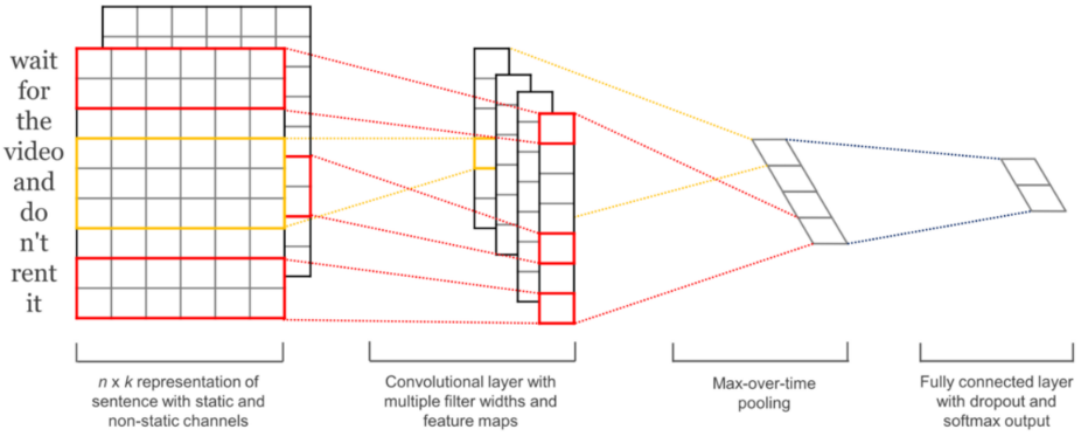

图 4-4 CNN 分类模型

Fig.4-4 CNN Classify Model

卷积神经网络文本过滤设计与实现

CNN（卷积神经网络）文本分类流程如图 4-5 所示。使用了交叉熵损失计算损失函数，学习率为 0.01。总结 CNN 文本过滤的步骤如下：

1 对评论源数据（这里时针对已经划分好的训练集）分词。本文中使用 jieba 对源数据做分词。

2 建立停止词集合。停止词一般包括语气词、助词、副词等经常在句子中

出现但并没有实际意义的词。这两步为数据预处理工作。为后续建立的分类模型提供可靠的数据源。

3 建立词嵌入层。将预处理后的文本转化为词嵌入的形式。这样我们可以得到一个形状为[n_words, EMBEDDING_SIZE]的词表映射矩阵。其中，n_words为字典大小，即输入数据的总词汇量。EMBEDDING_SIZE为嵌入矩阵的维度。

4 建立卷积层 1。输入为上一层输出的词向量。输出$N_FILTERS = 10$, 使用形状为[$WINDOW\_SIZE, EMBEDDING\_SIZE$]的 filter(滤波器/滤子)。其中$WINDOW_SIZE = 20$, $EMBEDDING\_SIZE = 20$。Padding = VALID

5 建立激活层 1。这里使用的时 relu 函数。即当输入数据小于零时，输出为零，当输入数据大于零时，输出等于输入。设置这个激活函数可以增加数据的非线性，帮助数据快速收敛，尤其当数据维度很高的时候。另外，也提供了神经网络的稀疏表达，避免梯度消失问题。

6 建立最大池化层 1。增加池化层也是下采样的一种手段。此处使用形状为 $[1, POOLING\_WINDOW, 1, 1]$ 的 filter(滤波器/滤子)，$POOLING\_WINDOW = 4$, stride(步长)为$[1, POOLING_STRIDE, 1, 1]$, $POOLING_{STRIDE} = 2$。Padding = SAME。从建立卷积层 1 到建立最大池化层 1，为 CNN_Layer1。

7 建立卷积层 2。输入为上一层最大池化的输出（max_pool1）。输出$N_FILTERS = 10$，使用形状为 [$WINDOW\_SIZE$, N_FILTERS] 的 filter(滤波器/滤子)。其中$WINDOW_SIZE = 20$, $N\_FILTERS = 10$。Padding = VALID

8 使用 squeeze 函数抽取特征。从卷积层 2 到本层为 CNN_Layer2。

建立全连接层。输入为上一层的输出。输出数为 2（二分类。1：垃圾评论，0：有效评论）

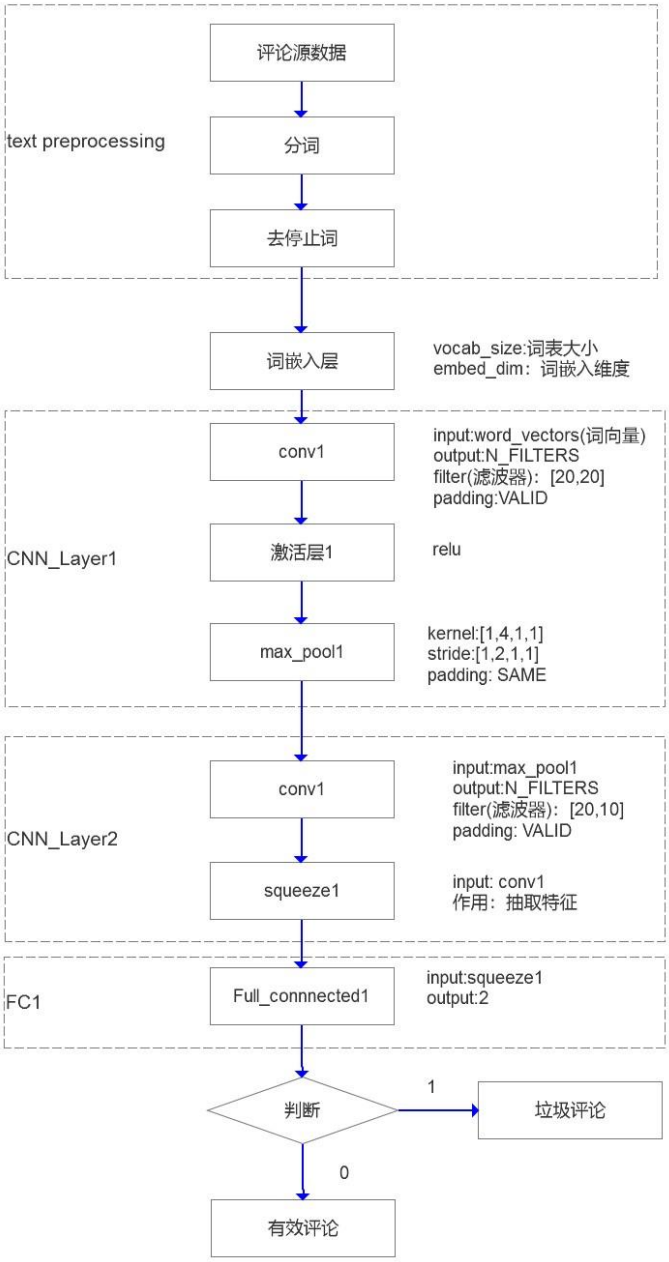

图 4-5 CNN 分类流程

Fig.4-5 CNN Classify Flow

分类实现过程的编程实现如下：

```
def cnn_text_classifier(features, target):
    """
    2 层的卷积神经网络，用于短文本分类
    """
    # 先把词转成词嵌入
    # 我们得到一个形状为[n_words, EMBEDDING_SIZE]的词表映射矩阵
    # 接着我们可以把一批文本映射成[batch_size, sequence_length, EMBEDDING_SIZE]的矩
```

阵形式

```
    target = tf.one_hot(target, 15, 1, 0)
    word_vectors = tf.contrib.layers.embed_sequence(
            features, vocab_size=n_words, embed_dim=EMBEDDING_SIZE, scope='words')
    word_vectors = tf.expand_dims(word_vectors, 3)
    with tf.variable_scope('CNN_Layer1'):
        # 添加卷积层做滤波
        conv1 = tf.contrib.layers.convolution2d(
                word_vectors, N_FILTERS, FILTER_SHAPE1, padding='VALID')
        # 添加 RELU 非线性
        conv1 = tf.nn.relu(conv1)
        # 最大池化
        pool1 = tf.nn.max_pool(
                conv1,
                ksize=[1, POOLING_WINDOW, 1, 1],
                strides=[1, POOLING_STRIDE, 1, 1],
                padding='SAME')
        # 对矩阵进行转置，以满足形状
        pool1 = tf.transpose(pool1, [0, 1, 3, 2])
    with tf.variable_scope('CNN_Layer2'):
        # 第 2 个卷积层
        conv2 = tf.contrib.layers.convolution2d(
                pool1, N_FILTERS, FILTER_SHAPE2, padding='VALID')
        # 抽取特征
        pool2 = tf.squeeze(tf.reduce_max(conv2, 1), squeeze_dims=[1])

    # 全连接层
    logits = tf.contrib.layers.fully_connected(pool2, 15, activation_fn=None)
    loss = tf.losses.softmax_cross_entropy(target, logits)

    train_op = tf.contrib.layers.optimize_loss(
            loss,
            tf.contrib.framework.get_global_step(),
            optimizer='Adam',
            learning_rate=0.01)

    return ({
            'class': tf.argmax(logits, 1),
            'prob': tf.nn.softmax(logits)
    }, loss, train_op)
```

4.4.2 循环神经网络过滤模型设计与实现

循环神经网络是一种可以捕捉时序信息的长短时记忆神经网络。对于长文的信息捕捉，凭借自带的“记忆”属性，具有较强的文本分类能力。在本文基于 RNN 的文本分类的实现中，先把词转成词嵌入，于是我们得到一个形状为[n_words, EMBEDDING_SIZE]的词表映射矩阵。n_words 是建立字典时，字典中词的个数，EMBEDDING_SIZE 是词嵌入的维度。接着把一批文本映射成[batch_size, sequence_length, EMBEDDING_SIZE]的矩阵形式。batch_size 是每批次(batch)送入模型的词的数量，sequence_length 是批次文本的长度。接着将文本分割成以词为单位的词嵌入，这样一来，词列表便转化成了形如[batch_size, EMBEDDING_SIZE]的矩阵组成的列表。然后，创建隐藏层，每当进入一个新词，就和上一个隐藏层联合计算出下一层，隐藏层反复利用，一直保留着最新的状态。最后添加一层全连接层。循环神经网络的最大优势在于可以真正充分地利用所有上下文信息来预测下一个词，而不像前面的训练方式那样，只能开一个 n 个词的窗口，只用前 n 个词来预测下一个词。

RNN（循环神经网络）文本分类流程如图 4-7 所示。使用了交叉熵损失计算损失函数，学习率为 0.01。总结 RNN 文本过滤的步骤如下：

1 对评论源数据（这里时针对已经划分好的训练集）分词。本文中使用 jieba 对源数据做分词。

2 建立停止词集合。停止词一般包括语气词、助词、副词等经常在句子中出现但并没有实际意义的词。这两步为数据预处理工作。为后续建立的分类模型提供可靠的数据源。

3 建立词嵌入层。将预处理后的文本转化为词嵌入的形式。这样我们可以得到一个形状为[n_words, EMBEDDING_SIZE]的词表映射矩阵。其中，n_words为字典大小，即输入数据的总词汇量。EMBEDDING_SIZE为嵌入矩阵的维度。

4 按列分解词向量，形成 word_list。word_list 里每一个元素为形为[batch_size, EMBEDDING_SIZE]的张量。其中，batch_size批次处理词的个数。

5 设置 GRU 神经元。

6 建立 RNN 层。此层的输入时上一层设置的 GRU 神经元和word_list。

7 建立全连接层。输入为上一层的输出。输出数为 2（二分类。1：垃圾评论，

0：有效评论）

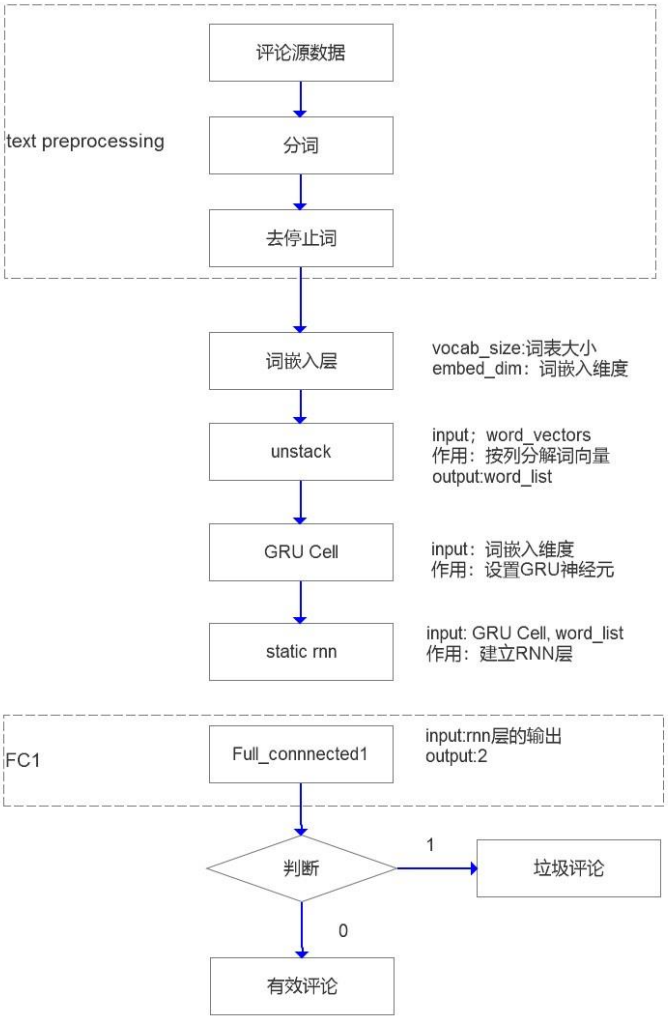

图 4-7 RNN 分类流程

Fig.4-7 RNN Classify Flow

RNN 文本分类实现过程如下：

```
def rnn_text_classifier(features, target):
    """用 RNN 模型(这里用的是 GRU)完成文本分类"""
    # Convert indexes of words into embeddings.
    # This creates embeddings matrix of [n_words, EMBEDDING_SIZE] and then
    # maps word indexes of the sequence into [batch_size, sequence_length,
    # EMBEDDING_SIZE].
    word_vectors = tf.contrib.layers.embed_sequence(
            features, vocab_size=n_words, embed_dim=EMBEDDING_SIZE, scope='words')
```

```
    # Split into list of embedding per word, while removing doc length dim.
    # word_list results to be a list of tensors [batch_size, EMBEDDING_SIZE].
    word_list = tf.unstack(word_vectors, axis=1)

    # Create a Gated Recurrent Unit cell with hidden size of EMBEDDING_SIZE.
    cell = tf.contrib.rnn.GRUCell(EMBEDDING_SIZE)

    # Create an unrolled Recurrent Neural Networks to length of
    # MAX_DOCUMENT_LENGTH and passes word_list as inputs for each unit.
    _, encoding = tf.contrib.rnn.static_rnn(cell, word_list, dtype=tf.float32)

    # Given encoding of RNN, take encoding of last step (e.g hidden size of the
    # neural network of last step) and pass it as features for logistic
    # regression over output classes.
    target = tf.one_hot(target, 15, 1, 0)
    logits = tf.contrib.layers.fully_connected(encoding, 15, activation_fn=None)
    loss = tf.contrib.losses.softmax_cross_entropy(logits, target)

    # Create a training op.
    train_op = tf.contrib.layers.optimize_loss(
            loss,
            tf.contrib.framework.get_global_step(),
            optimizer='Adam',
            learning_rate=0.01)

    return ({
            'class': tf.argmax(logits, 1),
            'prob': tf.nn.softmax(logits)
    }, loss, train_op)

model_fn = rnn_text_classifier
classifier = learn.SKCompat(learn.Estimator(model_fn=model_fn))

# Train and predict
classifier.fit(x_train, y_train, steps=1000)
y_predicted = classifier.predict(x_test)['class']
score = metrics.accuracy_score(y_test, y_predicted)
print('Accuracy: {0:f}'.format(score))
```

4.5 基于 LDA 模型的用户兴趣挖掘与推荐系统实现

4.5.1 基于 LDA 模型的用户兴趣挖掘

本文提出的用户兴趣挖掘方法是基于 LDA 主题模型构建的，其主要思想是通过建模用户在论坛里的评论文本，通过无监督学习的方法，得到相关的主题。在不同主题上获取不同关键词的分布情况，从而发现用户的兴趣点与喜好。在挖掘用户兴趣的过程中，使用 LDA 主题模型作为文本挖掘模型，搜集用户的评论文本，为每个用户建立评论文本记录，将用户的评论记录添加到用户文本库的列表中，文本库列表中的每个用户的评论记录被看作一个文档。对所有用户的评论文档库做无监督分类学习，从而得到用户评论文档在各个主题上的概率分布和各个主题上不同词语的概率分布。

如图 4-9 所示为用户兴趣模型结构。图中评论文档层和评论词层表示数据集中所有用户的评论文本组成的文档和这些文档中出现的词语。Topic 层是一个由文档层和词语层共享的 K 维空间，由于 K 值远小于文档数量和词语数量，本模型相当于对文档和词语做地位空间的嵌入。我们将原始数据集中文档-词语的关系，转化为文档-主题，主题-词语之间的关系。模型中的连线表示文档-主题，主题-词语上的权重。在文档-主题关系中，该权重表示某一文档在各个主题上的分布情况，在主题-词语关系中，该权重表示某一主题在各词语上的分布情况。模型的主要目标是学习各个连线上权重的大小。通过计算在不同主题上评论词的分布情况和概率可以统计出不同主题上分布出现哪些评论词。从而发现用户关注的热点和偏好。

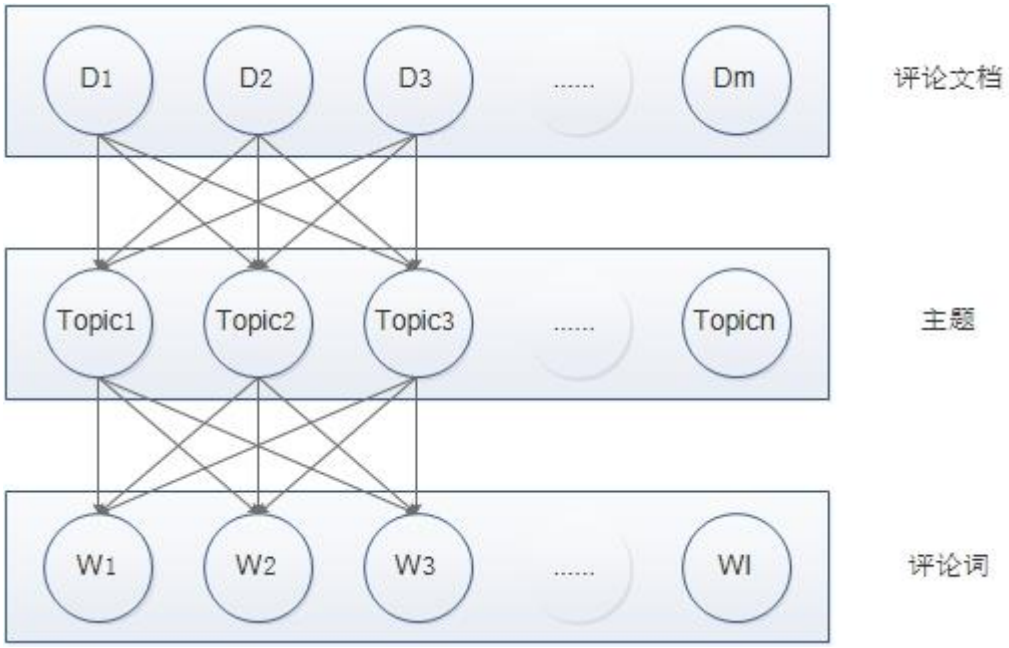

图 4-9 用户兴趣模型结构

Fig.4-9 User Interest Model Structure

如图 4-10 所示，为不同主题上，评论词的分布情况。这里取权重 top 10 的词语进行展示。

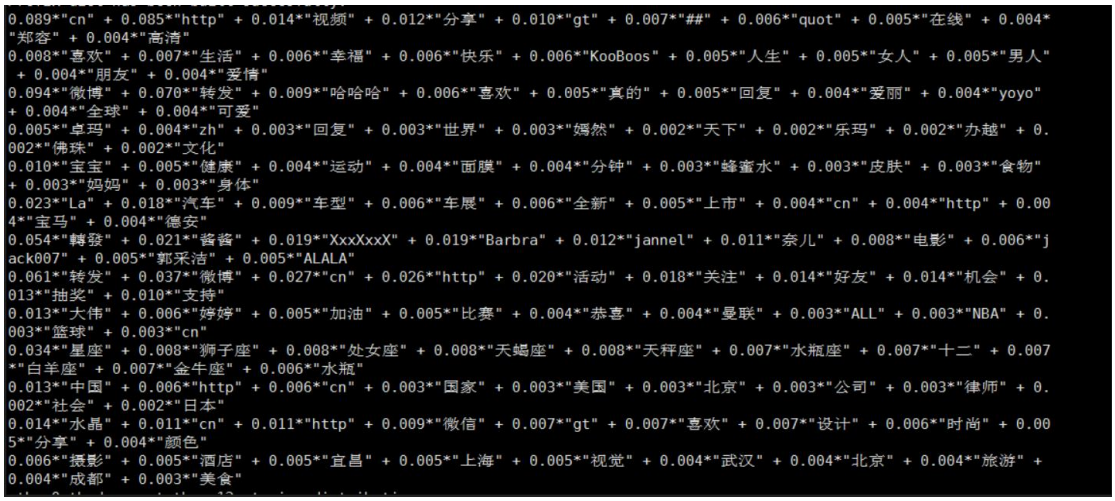

图 4-10 用户在不同主题上兴趣的分布

Fig.4-10 Users' Interest Distribution on Topics

如图 4-11 所示是随机选取 10 篇不同的文档，查看不同文档在 13 个 topic 上的分布情况。

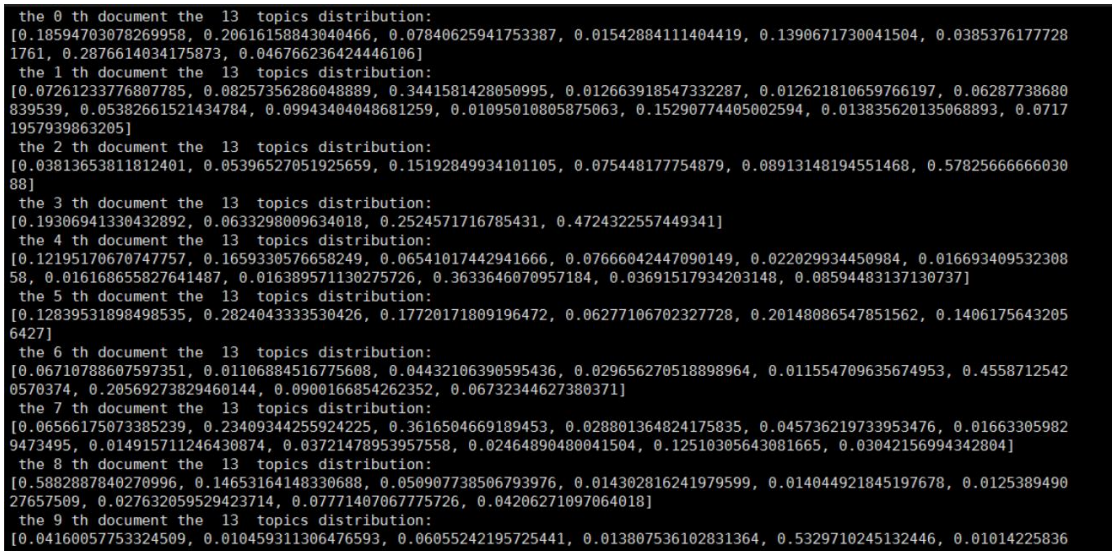

图 4-11 用户评论文档在主题上的分布

Fig.4-11 Users' Comments Document Distribution on Topics

在图 4-11 中，选取主题个数 K = 13。随着选取 K 值的不同，文档在主题上的分布情况也不同。此处根据论坛子类目的个数设置 K = 13。

4.5.2 基于 LDA 模型的推荐系统实现

在上一节中,我们通过分析评论词在不同主题上的分布情况挖掘用户的兴趣点。这个过程便是图 4-9 中主题层和评论词层的连接情况。而评论文档-主题的关系，则反映了不同文档在各个主题上的分布情况。而每个文档分别是由每个用户的发帖、回复、评论等文本组成。于是，每篇文档在各个主题上都形成一个分布向量。通过计算不同文档在各个主题上分布向量的距离便可以得出不同用户评论的相似度。从而得出用户-用户的相似度。这个相似度便体现了用户在不同主题上，发表评论的相似性。从中可以挖掘用户喜好的相近度。通过这个相近度计算的方法，找出和目标用户最相近的用户，然后将相近用户阅读过的文章推荐给目标用户。

如图 4-12 所示,首先将每个用户的发帖、评论、回复等数据整理成文档。如图中的D_1，D_2，D_3 …， 并且每个用户 id 下只有一个这样的文档。通过计算不同文档D_1，D_2，D_3 …D_n …在$Topic_1, Topic_2, Topic_3, \ldots Topic_m$,上的分布情况，计算不同用户之间的相似度。

将 top K 相似的用户 id 保存下来,这样便生成了目标用户的相似用户列表。通过查询相似用户列表中的用户阅读了哪些文章，形成文章候选集。通过计算文

章候选集中的文章和目标用户打分过的文章的相似度，就可以为目标用户推荐他们所需的文章了。推荐过程的计算方法如图 4-14 所示。

所以，基于 LDA 的推荐算法流程总结如下：

1 整理用户评论数据，并以字典的格式保存。字典的 key 是用户 id，value 是用户评论数据组成的列表。

2 通过使用 LDA 主题算法，计算用户评论文本在主题上的向量。计算各向量间相似度，从而获得目标用户的相似用户，保存相似用户的 user_id，添加入相似用户字典。字典的 key 是用户的 user_id， value 是候选用户与目标用户的相似度weight_{user}

3 从相似用户字典获取相似用户的 user_id 作为索引值，从用户打分矩阵中进行查询，获取相似用户访问过的文章。将这些文章添加入相似文章候选集。

4 计算文章候选集中的文章与目标用户访问过的文章的相似度得到文章相似度$\text{weight}_{article}$。最后，候选文章的相似度得分为weight_{user}乘以$\text{weight}_{article}$。

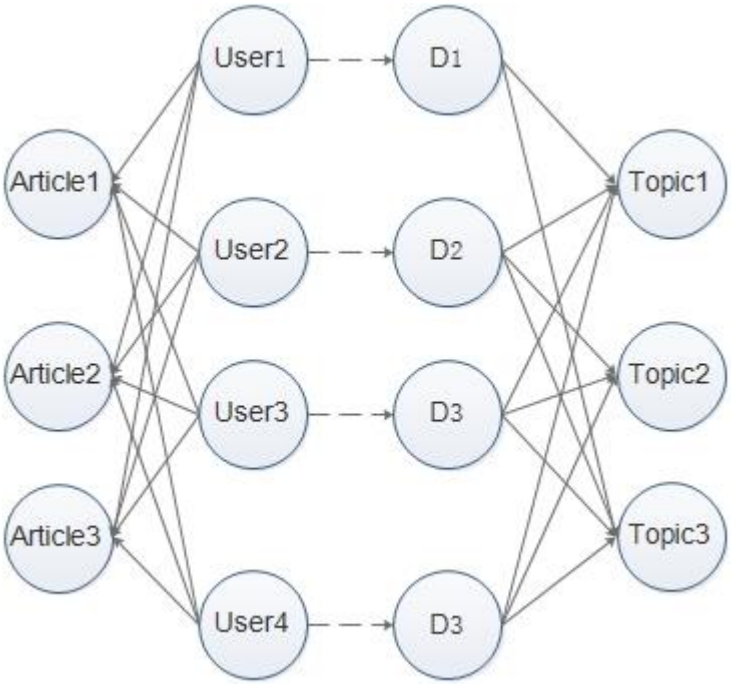

图 4-12 基于 LDA 模型的推荐方法

Fig.4-12 LDA Model based Recommendation Method

基于 LDA 主题模型计算用户相似度编程实现如下：

```
import json
import sys

visit_dict= {}
visit_list=[]
#contents = []
def parse_file(in_file, out_file):
    out = open(out_file, 'wb')
    #out = json.loads(in_file)
    #input = open(in_file).encode('utf-8')
    #for line in input:
    for line in open(in_file,'r',encoding='utf-8'):
        my_parse_line(line)
        #if(record):
        #    out.write(record.encode('utf-8').strip()+"\n")
    for userid in visit_list:
        visit_content = visit_dict[userid]
        outline = userid + ","+ '['+','.join(visit_content)+']'+"\n"
        #outline = userid + ","+','.join(visit_content)+"\n"
        out.write(outline.encode('utf-8'))
    out.close()


def my_parse_line(in_line):
    input = json.loads(in_line)
    #print (input[0])
    one_line = input[0]

    if one_line.get("userId"):
        #print("has userid")
        value_userid = one_line["userId"]
        #print(value_userid)
    else:
        return
    if one_line.get("content"):
        #print("has content")
        value_content = one_line["content"]
        #print(value_content)
    else:
        return

    if  visit_dict.get(value_userid):
```

```
            tmp = visit_dict[value_userid]
            tmp.append(value_content)
        else:
            visit_dict[value_userid] = [value_content]
            visit_list.append(value_userid)

def load_meterial(meterial, stopwords):
    contents = open(meterial, 'r')
    sentences=[]
    for line in contents:
        try:
            segs=jieba.lcut(line)
            segs = filter(lambda x:len(x)>1, segs)
            segs = filter(lambda x:x not in stopwords, segs)
            sentences.append(segs)

        except Exception,e:
            print line
            continue
    return sentences
if __name__ == '__main__':

stopwords=pd.read_csv("./data/stopwords.txt",index_col=False,quoting=3,sep="\t",names=['stop
word'], encoding='utf-8')
    stopwords=stopwords['stopword'].values
    #sentences_tech = load_meterial("./data/technology_news.csv",stopwords)
    sentences_wb = load_meterial("./document1_041902.txt",stopwords)
    #build the dictionary
    dictionary = corpora.Dictionary(sentences_wb)
    #print(dictionary.token2id)
    #print("token2id:\n",dictionary.token2id)

    corpus = [dictionary.doc2bow(sentence) for sentence in sentences_wb]
    #print corpus[5]

    corpus_lda = models.ldamodel.LdaModel(corpus=corpus, id2word=dictionary,
num_topics=13)
    #corpus_lda.save('lda_tech.model')
    #print lda.print_topic(3, topn=5)
    for topic in corpus_lda.print_topics(num_topics=13, num_words=10):
        print topic[1]

    doc_topics    = corpus_lda.get_document_topics(corpus)
```

```
for i in range(20):
    topic = np.array(doc_topics[i])
    topic_distribute = np.array(topic[:, 1])
    topic_idx = list(topic_distribute)
    print(' the %d th document the    %d    topics distribution:' % (i, num_show_topic))
    print(topic_idx)

test_doc1    = load_meterial("./testdoc/testdoc1.txt",stopwords)
test_cor1 = [dictionary.doc2bow(sentence) for sentence in test_doc1]
test_vec = corpus_lda[test_cor1] # convert the query to LSI space
print("test_vec:\n",test_vec)

index = similarities.MatrixSimilarity(corpus_lda[corpus])
index.save('./index/0419/myindex.index')
index = similarities.MatrixSimilarity.load('./index/0419/myindex.index')

sims = index[test_vec] # perform a similarity query against the corpus
print(list(enumerate(sims))) # print (document_number, document_similarity) 2-tuples
max_value = sims.max()
max_ind = np.where(sims == max_value)
print("max_value:\n", max_value)
print("max_ind:\n", max_ind)
```

$$Similarity\left(article_{u_{target}}, article_{u_{cadidate}}\right) = w_u \cdot Similarity_u\left(article_{u_{target}}, article_{u_{cadidate}}\right) \quad (4\text{-}1)$$

$$w_u = Similarity(User_{target}, User_{cadidate}) \quad (4\text{-}2)$$

如公式 4-1 所示，候选文章与目标文章的相似度等于用户权重与文章相似度的乘积。这里的用户权重就是先前获得的相似用户的 score。

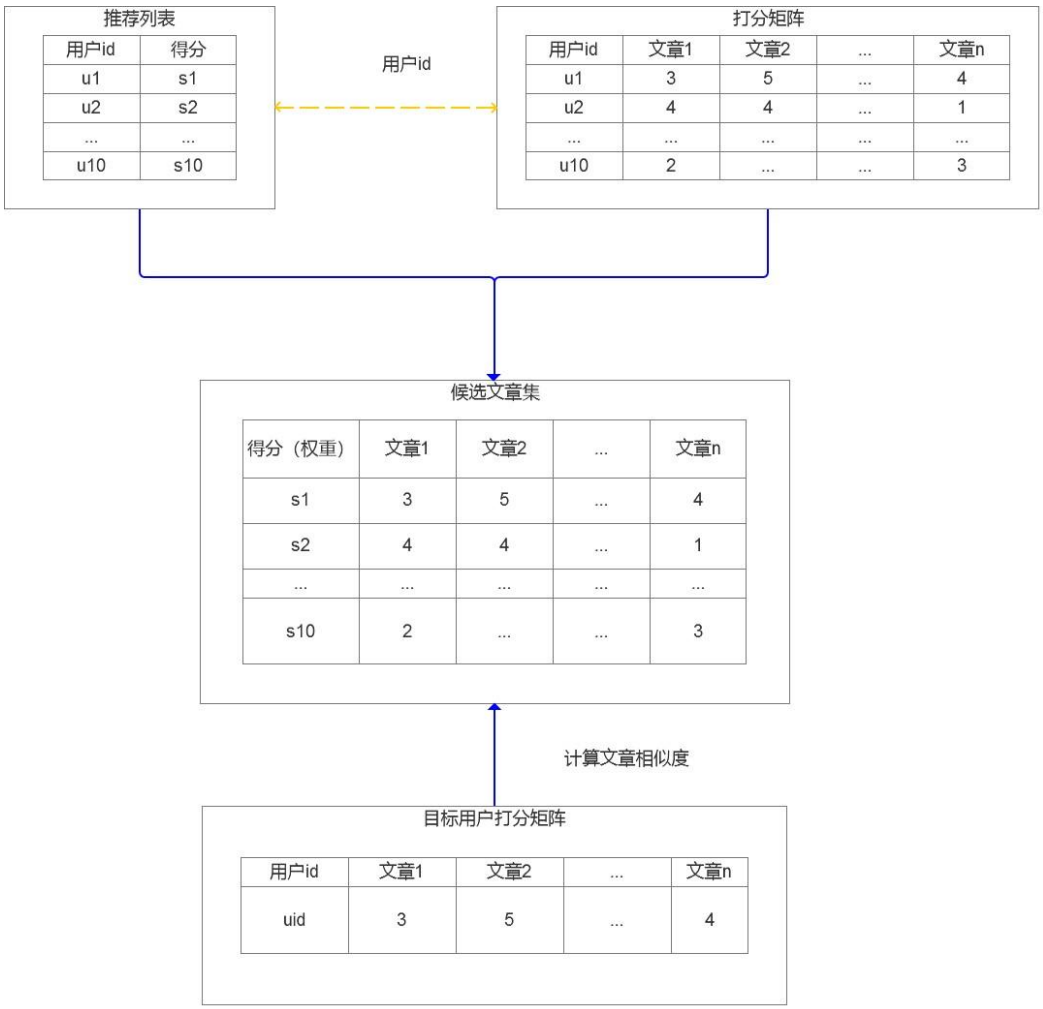

图 4-14 推荐结果计算方法

Fig.4-14 Computation Method for Recommendation Result

4.6 本章小结

在本章中，分析了目前的推荐算法所面临的问题并提出了新的推荐策略和解决方法。首先详细介绍了基于神经网络的文本过滤方法。分析了卷积神经网络（CNN）和循环神经网络(RNN)各自从文本预处理、神经网络建模到最后输入过滤结果的一系列流程。在建模过程中，详细阐述了不同神经网络中每一层的结构和设计方法。。接着，引入了 LDA 模型，介绍了基于 LDA 模型的推荐方法和实现流程。这种方法通过挖掘用户兴趣和分析用户偏好的分布情况，可以找出相近的用户，从而可以作为基于用户的推荐方法得以应用。

第五章 推荐系统实验与分析

5.1 平台环境与数据集简介

5.1.1 开发平台环境简介

在推荐系统实现过程中,所使用的机器学习工具库和自然语言处理工具库如表 5-1 所示。

表5-1 开发平台简介

Table 5-1 Developing Platform Introduction

序号	平台使用工具	版本信息
1	开发语言	Python2.7
2	科学计算工具库	Numpy, scikit-learn
3	计算框架	Tensorflow1.6
4	分词工具	Jieba
5	自然语言处理库	Gensim

5.1.2 数据集简介

1. 协同过滤与隐语义模型的数据结构

本文实验数据来源为某企业社区的用户数据。根据 3.1 节中介绍的打分规则，整理用户数据组织成打分矩阵。如图 5-1 所示，每一行代表一个记录，每一列分别为记录的序号、用户 id、文章 id、分值和时间戳。

no.	user_id	article_id	rating	timestamp
0	1	1	5	1392799358
1	1	2	3	1392809857
2	1	3	4	1393048286
3	1	4	3	1392879880
4	1	6	5	1393113570

图 5-1 打分矩阵

Fig.5-1 Ranking Matrix

2. 神经网络模型的数据结构

在神经网络模型中，训练和测试的数据字段分别是用户发帖、回复、评论的

文本数据。首先从数据库中提取用户 id、发帖内容、回复内容和评论内容这些字段，然后将数据按照用户 id 进行归并。这样便形成了以用户 id 为索引值的用户评论列表。数据集总共收集了 105420 个用户的评论数据。训练数据集与测试数据集的切分比例为 7：3。

5.2 评估标准

在通常情况下，推荐系统设计之初便会考虑评价指标。通过对指标的评估，逐步优化算法。比较常见的推荐系统评价指标有：精确度、准确性、查全率和广泛性等。

5.2.1 精确度

在预测用户对物品的评分时，通过计算预测评分和实际之间的差异来评估预测的精确度。常用的方法有均方根误差（Root Mean Square Error）和平均绝对误差（Mean Absolute Error）。

1. 均方根误差

对于测试集中的一个用户 u 和物品 I，令r_{ui}是用户 u 对物品 i 的实际打分，而$\widehat{r_{ui}}$是推荐算法给出的预测评分，故 RMSE（Root Mean Square Error）的定义为：

$$\mathrm{RMSE} = \frac{\sqrt{\sum_{u,i\in T}(r_{ui} - \widehat{r_{ui}})^2}}{|T|} \tag{5-1}$$

2. 平均绝对误差

MAE 采用绝对值计算预测误差，它的定义为：

$$\mathrm{MAE} = \frac{\sum_{u,i\in T}|r_{ui} - \widehat{r_{ui}}|}{|T|} \tag{5-2}$$

5.2.2 准确性

而对于取 Top-K score 的推荐系统，通常分析用户的历史行为数据，通过找到用户未购买/打分过的物品，并且预测用户兴趣最高的 Top K 个物品，推荐给用户。所以，这种推荐方法经常会使用准确性（Precision）和查全率（Recall）

来衡量推荐准确度。

令 R(u)表示系统预测用户感兴趣的物品，T(u)表示用户真实喜欢的物品。所以准确性就是用来衡量 R(u)中用户真正喜欢的物品的比例。一般情况下，准确率越高，预测结果越接近实际情况。计算公式如公式（5-3）所示。

$$\text{Precision} = \frac{\sum_{u \in U} |R(u) \cap T(u)|}{\sum_{u \in U} |R(u)|} \tag{5-3}$$

5.2.3 查全率

召回率是用来衡量用户真正感兴趣的物品 T(u)中有多少被推荐给用户的比重。计算方法如公式（5-4）所示。通常情况下，在保证准确性不下降的前提下，尽量提高推荐系统的查全率。

$$\text{Recall} = \frac{\sum_{u \in U} |R(u) \cap T(u)|}{\sum_{u \in U} |T(u)|} \tag{5-4}$$

5.2.4 广泛性

用户的兴趣是广泛的，对于一个用户而言，他可能既对 A 物品感兴趣，又对 B 物品感兴趣。在用户游览网站文章的时间跨度内可能偏爱的文章不尽相同，为了使推荐效果让用户满意，推荐结果应该覆盖不同时刻的大多数兴趣点。广泛性就是衡量不同时间段内不同文章多样化的指标。只有推荐的文章广泛性高了，才能满足用户广泛的爱好。

令 sim(i,j)表示文章 i 和 j 之间的相近度，系统广泛性如公式(5-5)所示。

$$\text{Diversity} = \frac{1}{|U|} \sum_{u \in U} \left(1 - \frac{\sum_{i,j \in R(u)} sim(i,j)}{\frac{1}{2} |R(u)|(|R(u)| - 1)} \right) \tag{5-5}$$

5.2.5 准确度

准确度（Accuracy）用于对分类器整体性能的评估。其中，TP 表示真实正例的个数，FP 表示伪正例的个数，TN 表示真实负例的个数，FN 表示伪负例的个数。

准确度计算方法如公式（5-8）所示。

$$\text{acc} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5-6}$$

5.3 基于协同过滤和隐语义模型的推荐系统

5.3.1 协同过滤实验与分析

为了验证相似度计算和 top K 取值等优化方法对推荐结果的优化效果，所以进行了几组对照实验。

实验数据：用户数：6040；文章数：3952；矩阵填充度：5%。

实验结果总结如表 5-2 所示。在第 1 组中，使用余弦相似度计算 item-item(article-article)之间相似度。未做归一化等操作，基于 item(article)-CF 进行推荐计算。RMSE 达到 1.0042。在第 2 组中，在组 1 的基础上，进行了归一化操作。由于不同用户打分偏好不同，故需要对用户的打分做归一化操作。避免不同用户的打分习惯引入的噪声。通过这项操作，RMSE 得到相应减小。达到 0.9345。第 3 组中，使用余弦相似度计算 user-user 之间相似度，未做归一化操作，基于 user-CF 进行推荐计算。RMSE 达到 1.0133。第 4 组在第 3 组的基础上，做了归一化操作。方法如第 2 组。获得 RMSE 也相应减小。第 5 组是在第一组的基础上，使用皮尔森相似度计算 item-item(article-article)之间相似度。并且采用 top-k 的推荐方法。RMSE 减小到 0.92。

表5-2 实验结果对照

Table 5-2 Experimental Results Comparison

分组	算法描述	测试集大小	RMSE
1	Item(article)-CF	20000	1.0042
2	Item(article)-CF、归一化到用户	20000	0.9345
3	User-CF	20000	0.9519
4	Item(article)-CF、归一化到用户	20000	0.9200
5	Item(article)-CF、皮尔森归一化、取 topK(K=13)	20000	0.9200

5.3.2 隐语义模型实验与分析

实验数据：用户数：6040；文章数：3952；矩阵填充度：5%

由图 5-2 可见，随着迭代次数增加，训练集误差逐渐下降。测试集上的误差在开始阶段逐渐下降，当迭代到 epoch=66 时，误差达到最小值，val_error = 0.847936。在此之后，误差又逐渐上升。总迭代次数 epoch=200。相比之前使用协同过滤的训练效果，使用隐语义模型的效果在测试集误差上降低了 0.05%。

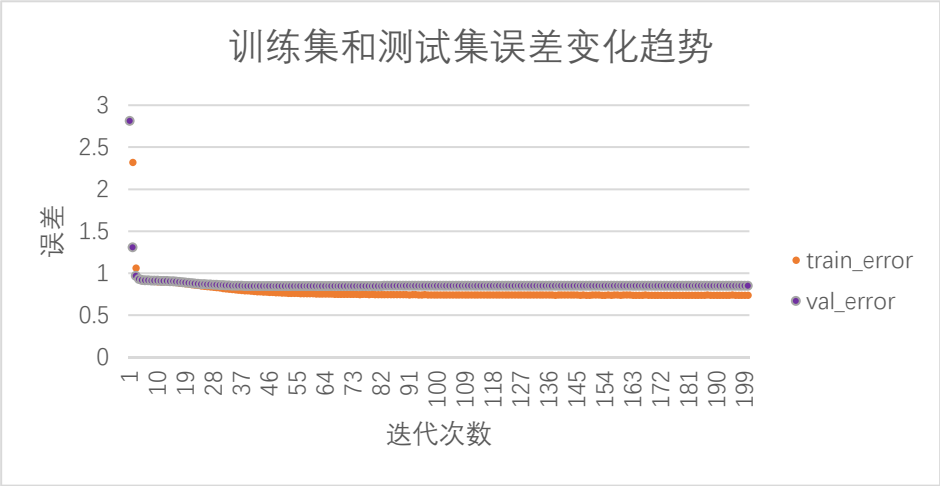

图 5-2 隐语义模型误差变化

Fig.5-2 LFM Model Loss Change

5.4 基于神经网络的推荐系统

5.4.1 文本过滤实验与分析

1. 模型对比

在 CNN 模型中，设置文档最长长度为 100， 最小词频为 2， 词嵌入维度为 20， 学习率为 0.01。

在 RNN 模型中，设置文档最大长度为 15， 最小词频为 1， 词嵌入维度为 15， 学习率为 0.01。

2. 文本过滤实验分析

通过文本分类的方法对原始用户评论文本进行过滤，剔除了无效冗余的评论文本。将文本分类后类别为有效文本的评论数据作为后续挖掘用户兴趣点和推荐

系统建模的输入数据。在整个推荐系统的结构中，文本过滤作为特征工程部分，为后续建模工作剔除了无效文本，降低了噪声带来的干扰，它是推荐系统中非常重要的环节。

文本共分为垃圾评论和不是垃圾评论两个类别。如图 5-3 所示为 CNN、词袋模型和 RNN 模型的分类效果。三种模型的损失都随着迭代次数增加逐渐下降。相比其他两种模型的收敛速度，RNN 模型相对较快，词袋模型次之，CNN 最慢。当完成 1000 次迭代时，CNN 模型的准确度达到 0.888625，词袋模型的准确度达到 0.890771， GRU(RNN)模型的准确度达到 0.890086。三种模型的准确度降序排名依次是词袋模型、RNN 模型和 CNN 模型。在图 5-4 中可以发现，同样经过 1000 次迭代，词袋模型耗时非常厉害，明显高于另外两个模型。CNN 模型和 RNN 模型耗时差距不大，RNN 模型率高一些。综合训练耗时和准确度两个评价指标，RNN 模型相对耗时比较短，准确度较高。

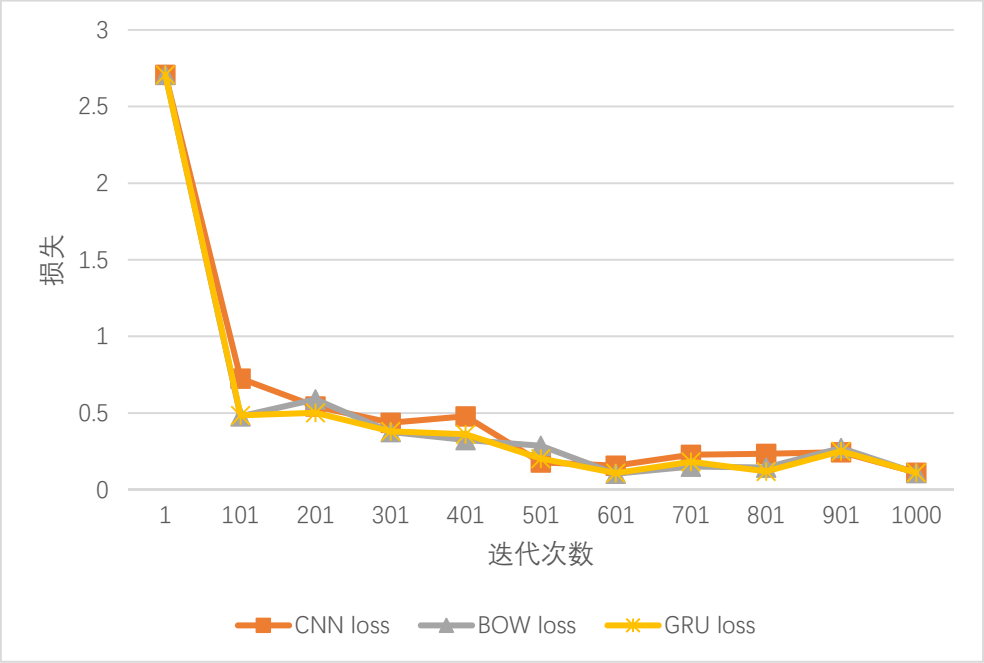

图 5-3 三组模型损失对比

Fig.5-3 Loss Comparison of Three Different Models

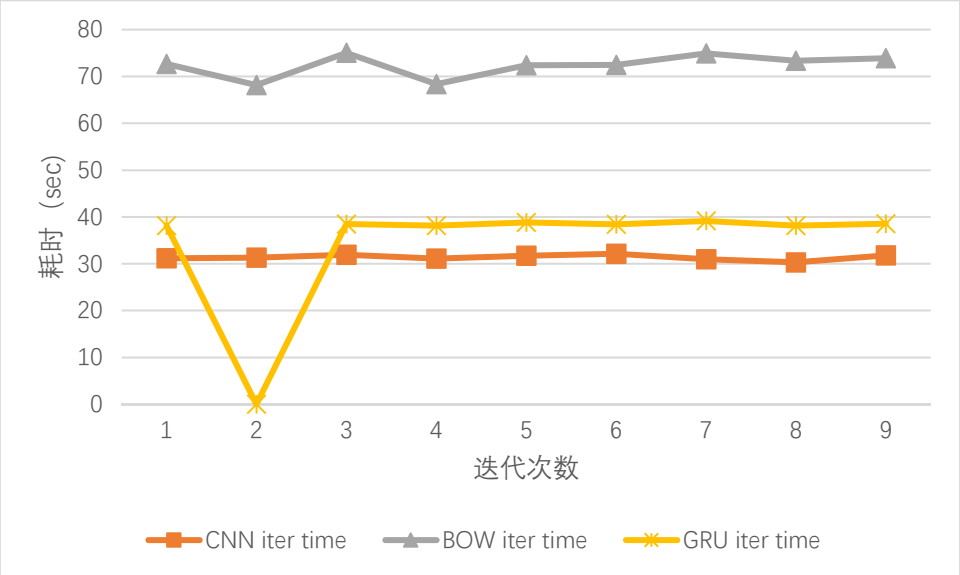

图 5-4 三组模型耗时对比

Fig.5-4 Spent Time Comparison of Three Different Models

5.4.2 LDA 实验与分析

三种推荐系统对比实验与分析

在实验结果的分析中，我们定义以改进过的基于用户的协同过滤为 User-CF、改进过的基于文章的协同过滤为 Item-CF、以结合 LDA 分析用户评论偏好与基于用户的协同过滤相结合的算法为 LDA+User-CF。

图 5-5 所示的是三种不同算法在 MAE（平均绝对误差）上的评测结果。X 轴为最近邻数 K，取值[5,50]。间隔为 5。Y 轴为 MAE 值。由图可知，基于 LDA 分析的算法的 MAE 最小（即准确率最高），其次是 Item-CF 算法，User-CF 算法准确率最差。在增长趋势上,User-CF 的 MAE 值在 K = 35 时，随着最近邻数增大下降较快，Item-CF 算法的 MAE 值在 K = 30 之前随着最近邻数增大下降较快，但在 K= 30 之后变化幅度很小，走势平稳。这说明寻找最近邻数的能力较弱，只有当最近邻数范围设置较大值时才能得到合适的匹配。而本文提出的算法变化平稳，能在较小的最近邻数下找到最相似的用户集合。

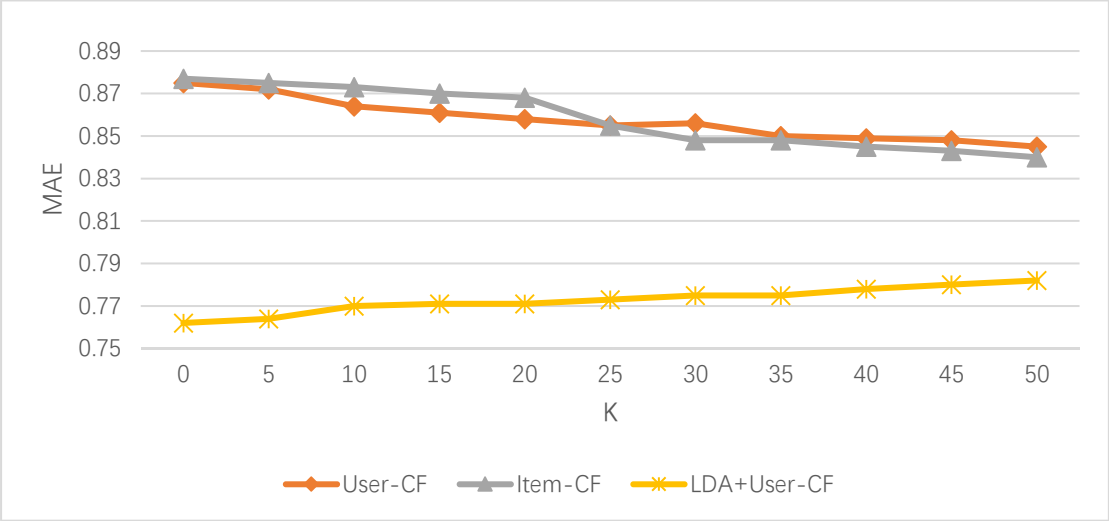

图 5-5 三种算法下 MAE 评估指标

Fig.5-5 Three Different Algorithm MAE

图 5-6 所示的是三种不同算法在 RMSE（均方根误差）上的评测结果。X 轴为最近邻数 K，取值[5,50]。间隔为 5。Y 轴为 RMSE 值。由图可知，基于 LDA 分析的算法均方根误差最小（即准确率最高），其次是 Item-CF 算法，User-CF 算法的准确率最差。在增长趋势上，RMSE 与 MAE 的增长趋势一致，但是 RMSE 比 MAE 对偏差的惩罚力度更大。

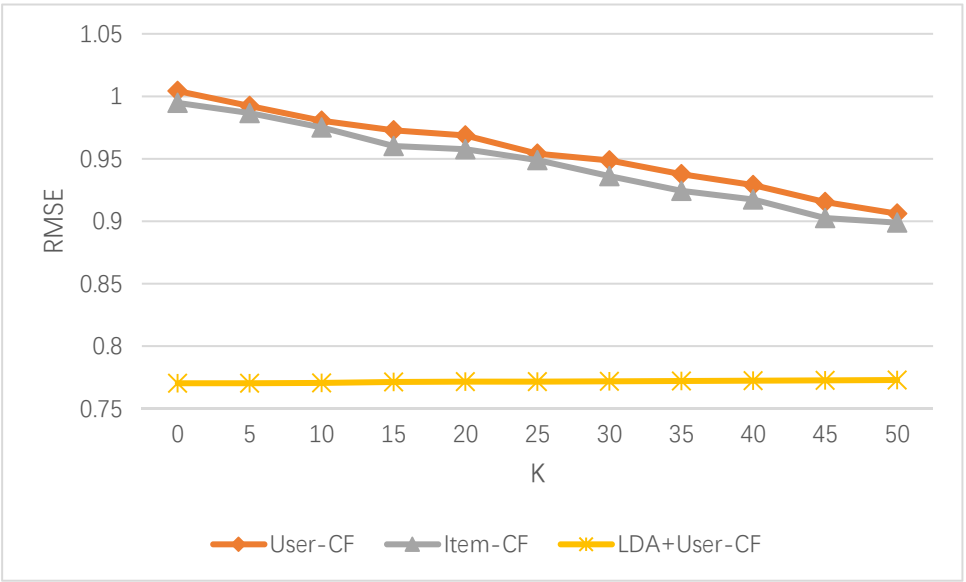

图 5-6 三种算法下 RMSE 评估指标

Fig.5-6 Three Different Algorithm RMSE

图 5-7 所示的是三种不同算法在 Recall（召回率）上评测结果。X 轴为最近邻数 K，取值[5,50]。间隔为 5。Y 轴为 Recall 的百分比值。由图可知，基于 LDA 分析的算法召回率明显优于其他两种算法。在增长趋势上,三种算法的变化幅度不大，在 K=20 时，基于 LDA 的算法变化逐渐平缓，而另外两种算法需要更大的 K 值才能趋于平稳。

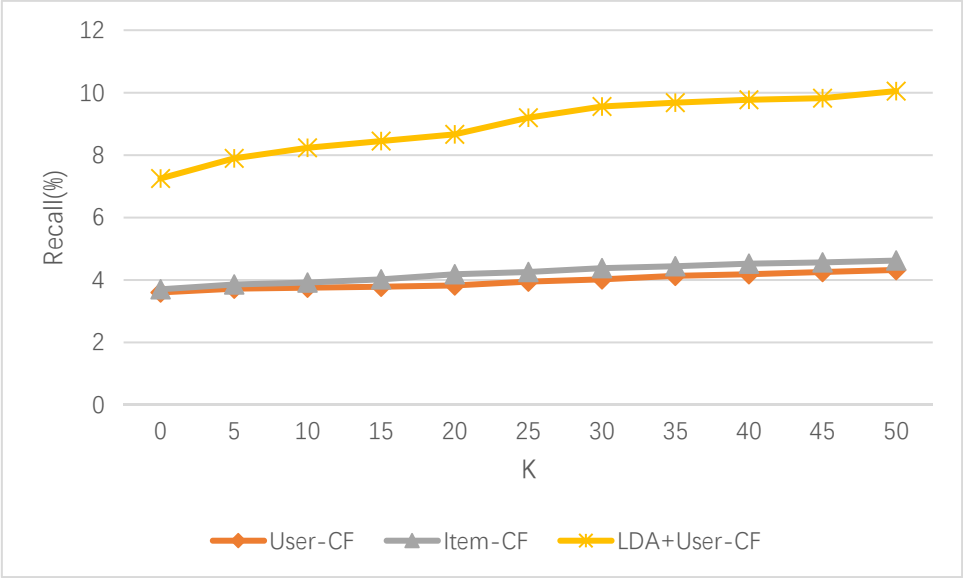

图 5-7 三种算法下 Recall 评估指标

Fig.5-7 Three Different Algorithm Recall

图 5-8 所示的是三种不同算法在 Precision（准确率）上评测结果。X 轴为最近邻数 K，取值[5,50]。间隔为 5。Y 轴为 Precision 百分比值。由图可知，基于 LDA 分析的算法准确率明显优于其他两种算法。Precision 的变化趋势与 Recall 的变化趋势基本相似。

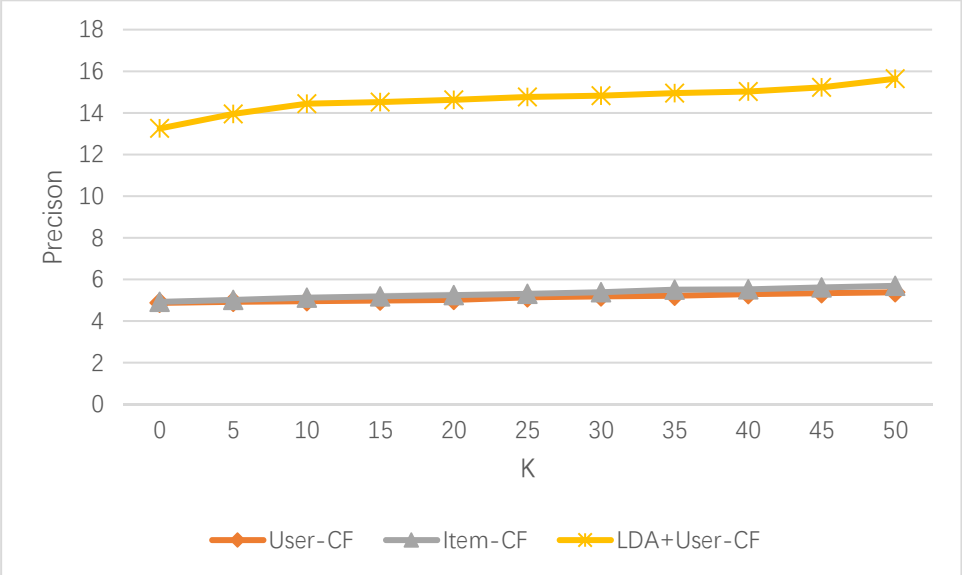

图 5-8 三种算法下 Precision 评估指标

Fig.5-8 Three Different Algorithm Precision

图 5-9 所示的是三种不同算法在 Diversity（多样性）上评测结果。X 轴为最近邻数 K，取值[5,50]。间隔为 5。Y 轴为 Diversity 值。由图可知，基于 LDA 分析算法的多样性明显优于其他两种算法，而另外两种算法的多样性比较相似。这说明基于 LDA 分析算法的推荐效果在保证准确率和用户兴趣的同时，提高了推荐结果的多样性。

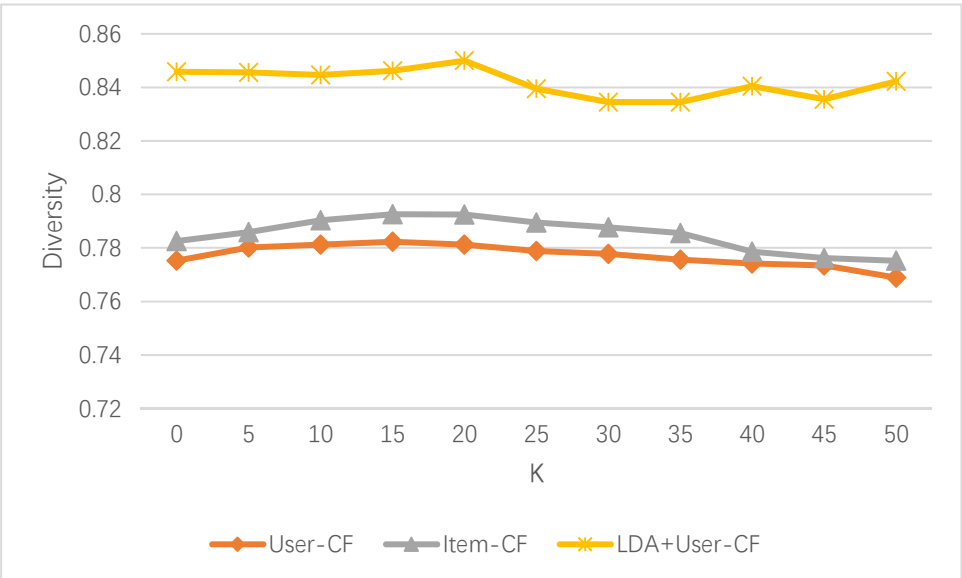

图 5-9 三种算法下 Diversity 评估指标

Fig.5-9 Three Different Algorithm Diversity

5.5 本章小结

在本章中，详细介绍了开发平台环境和使用工具库的版本信息。介绍了实验的评估标准。包括精确度、准确率、召回率和多样性的定义和计算方法。实验分析了基于协同过滤的推荐系统的推荐结果和基于增量的隐语义模型的推荐结果。详细分析了基于卷积神经网络的过滤方法、基于词袋模型的过滤方法和基于循环神经网络的过滤方法，这三种过滤方法的准确率。对比了这三种模型在损失和耗时上的性能优劣。此外，还重点分析并对比了改进过的基于用户的协同过滤算法、 改进过的基于文章的 CF 算法、结合 LDA 与基于用户协同过滤相结合的算法，这三种算法。分别对比分析了这三种算法在 MAE（平均绝对误差）、RMSE（均方根误差）、Recall(查全率)、Precison（准确性）和 Diversity（广泛性）上的差异和变化趋势。

第六章 总结与展望

6.1 总结

随着互联网技术的高速发展，数据量急速膨胀。如今已经进入了大数据时代，无论电子商务、社交、多媒体等领域，都面临信息量过载的严峻问题。帮助用户快速、高效地筛选出符合用户需求的数据迫在眉睫。本文研究的社区文章推荐系统就是希望要解决这样的问题。传统的推荐算法存在可扩展性差、数据稀疏性以及推荐结果同质化等问题，并且准确率、召回率在达到瓶颈后便无法继续提高。针对上述问题，本文做了深入的研究，并取得以下研究成果：

第一，详细研究了基于协同过滤的推荐算法、隐语义推荐算法、特征词提取技术、卷积神经网络相关理论和循环神经网络相关理论和 LDA 主题模型的理论基础。并且，将这些理论相互结合应用到实际的推荐系统中。

第二，将论文内容与世界工作相结合，采用社区论坛的真实数据，包括用户行为数据和评论数据。完成从数据采集、预处理、建立模型、得出推荐结果和实验验证分析一系列的工作。这既是一次大胆的创新尝试，也是一次对自我的挑战与锻炼。

第三，对于传统的协同过滤算法、隐语义模型进行了深入的研究，并提出自己的改进意见。此外，结合协同过滤、神经网络、LDA 主题模型各种算法和理论的长处和优点设计了融合模型的推荐系统。并且，通过实验验证在推荐准确率、召回率、多样性等多个性能指标上都取得明显提升。并且，也解决了传统推荐算法中常见的数据稀疏性问题和推荐同质化等问题。

由于个人能力有限，对推荐系统的研究和探索工作还不够全面，我将在今后的工作中逐步提高其各方面的性能，帮助用户方便快捷地找到需要地文章，从而提高用户体验。

6.2 展望

尽管目前的推荐系统已经在推荐准确率、召回率、多样性等指标上获得了不

错的成绩，但是对于推荐内容和功能仍然可以做进一步的扩展与提升。主要包括以下内容：

基于主题的用户兴趣挖掘模型能够准确地获取用户兴趣，并可以根据所获取的兴趣点为文章增加标签。通过匹配文章标签和用户兴趣点可以更快地找到符合用户偏好的文章。

此外，通过用户兴趣挖掘，可以建立用户画像，结合用户点击、游览、发帖、回复等历史行为记录和用户固有的属性数据，比如年龄、职业等等构建用户画像，最终应用到挖掘用户社交关系等方方面面。

最后，随着推荐文章数量和用户数量的攀升，用户行为将变得更加多样化。提取最合适的行为特征将成为影响推荐效果的关键因素。可以研究推荐效果与用户行为之间的关系，采用机器学习的方法提取含有最大信息熵的特征，并且随着用户行为的变化，同步更新这些特征。这样一来，推荐结果将实时效率更高。

参考文献

[1] 中国互联网信息中心（CNNIC). 第 32 次中国互联网发展状况统计报告.中国，2013-07

[2] 中文信息处理发展报告（2016），2016.

[3] 项亮，推荐系统实战[M]，北京，人民邮电出版社，2012，4-5.

[4] Davidson J, Liebald B, Liu J, et al. The YouTube video recommendation system [C] ACM Conference on Recommender Systems. ACM, 2010:293-296

[5] Sarwar, Badrul, Karypis , et al. Analysis of recommendation algorithms for e-commerce [J]. 2000

[6] Qiu T, Chen G, Zhang Z K, et al. An item-oriented recommendation algorithm on cold-start problem[J]. EPL (Europhysics Letters), 2011, 95(5): 58003.

[7] Schein A I, Popescul A, Ungar L H, et al. Methods and metrics for cold-start recommendations. Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2002. 253-360.

[8] Shen Y, Jin R. Learning personal+ social latent factor model for social recommendation[C] Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2012: 1303-1311.

[9] 刘青文．基于协同过滤的推荐算法研究［D］．中国科学技术大学，2013．

[10] 曹毅. 基于内容和协同过滤的海合模式推荐技术研究［D］. 中南大学，2007．

[11] 杨兴耀, 于炯, 吐尔根·依布拉音. 考虑项目属性的协同过滤推荐模型[J]. 计算机应用, 2013, 33(11): 3062-3066.

[12] 尹路通, 于炯, 鲁亮, 等. 融合评论分析和隐语义模型的视频推荐算法[J]. 计算机应用, 2015, 35(11): 3247-3251.

[13] 翁涛，基于协同过滤的个性化推荐算法研究[D]. 重庆：重庆大学计算机软件与理论，2011.

[14] A. Yamashita, H. Kawamura, K. Suzuki. Adaptive fusion method for user-based

and item-based collaborative filtering [J]. Advances in Complex Systems, 2011, 14(2):133-149

[15] Pazzani M J, Billsus D. Content-based recommendation systems[M] The adaptive web. Springer Berlin Heidelberg, 2007: 325-341.

[16] 郑捷，机器学习算法原理与编程实践 [M]，北京，电子工业出版社，2015，71-72.

[17] Ramos J. Using tf-idf to determine word relevance in document queries[C] Proceedings of the first instructional conference on machine learning. 2003.

[18] Yoshua Bengio, Rejean Ducharme,Pascal Vincent, Christian Jauvin. A Neural Probabilistic Language Model. Journal of Machine Learning Research 3(2003) 1137-1155.

[19] 黄文坚等，Tensorflow 实践 [M]，北京，电子工业出版社，2017，159.

[20] Tomas Mikolov , Kai Chen, Greg Corrado, et al. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781,2013

[21] Tomas Mikolov , Quoc V. Le, Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. arXiv:1309.4168v1,2013

[22] Quoc V. Le, Tomas Mikolov . Distributed Representations of Sentences and Documents. arXiv:1405.4053,2014

[23] Ronan Collobert, Jason Weston A Unified Architecture for Natural Language Processing : Deep Neural Networks with Multitask Learning

[24] David E Rumelhart , Geoffrey E Hinton, and Ronald J Williams. Learning representations by backpropagating error. Nature, 323(6088):533-536,1986

[25] Tomas Mikolov, M Karafiat , L Burget et al. Recurrent neural network based language model. Interspeech, Conference of the International Speech Communication Association, 2010:1045-1048.

[26] EH Huang, R Socher , CD Manning, et al. Improving word representations via global context and multiple word prototypes. Meeting of the Association for Computational Linguistics,2012, 1:873-882

[27] 卷积神经网络研究综述[J]. 李彦冬,郝宗波,雷航. 计算机应用. 2016(09)

[28] 黄文坚，唐源，TensorFlow 实战[M]，北京，电子工业出版社，2017，159-160.

[29] Ian Goodfellow , 深度学习[M]，北京，人民邮电出版社，2017，

[30] Deeplearning.ai，http://mooc.study.163.com/university/deeplearning_ai#/c

[31] Hochreiter, Schmidhuber, et al. Long Short-Term Memory,Neural Computation 9(8):1735-80, 1997

[32] 基于主题模型的专利文本挖掘方法及应用研究[D]. 陈虹枢.北京理工大学 2015

[33] 基于主题模型的多标签文本分类和流文本数据建模若干问题研究[D]. 李熙铭.吉林大学 2015

[34] 廖晓锋, 王永吉, 范修斌, 等. 基于 LDA 主题模型的安全漏洞分类[J]. 清华大学学报: 自然科学版, 2013, 52(10): 1351-1355.

[35] 刘江华，等 一种基于 kmeans 聚类算法和 LDA 主题模型的文本检索方法及有效性验证[J].情报科学，2017, 02.

[36] 徐盛，李德仁，方涛，等. 基于主题模型的高空间分辨率遥感影像分类研究. 上海交通大学，2012.

[37] 杨强，杨有,余春君.协同过滤推荐系统研究综述[J].现代计算机,2015,(9):3-6.

[38] Xiaoqing Zheng, Hanyang Chen, Tianyu Xu. Deep Learning for Chinese Word Segmentation and POS tagging. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pages 647-657

[39] 周昆，基于改进向量空间模型的中文文本分类研究．北京理工大学，2015．

附 录

致 谢

光阴似箭，一转眼三年的研究生学习即将结束了。回首这些岁月中，有太多需要铭记。无论是在完成硕士学位课程学习的过程中，还是在本硕士学位论文工作的研究过程中，多亏了老师们的悉心教导和同学们的关心帮助。在此，衷心感谢他们的鼓励和支持。

本文在我的导师姚天昉副教授悉心指导下完成的。姚老师深厚的学术功底和严谨的研究作风令我受益匪浅。在姚老师的指导下，我完成了论文的选题、开题报告的撰写、论文的撰写、实验研究等工作，最终完成了论文。感谢姚老师在我研究论文过程中无私的指点和耐心的讲解。为我提出了许多宝贵的建议意见，开拓了我的视野，扩展了思路。再次由衷地感谢姚老师给予指导和帮助！

感谢我的母校，为广大莘莘学子提供这样一个充满学术氛围的环境，让我们徜徉在知识的海洋中，让我的生活丰富多彩！怀着感恩的心，我会继续努力，祝福大家都能越来越好！

攻读学位期间发表的学术论文

[1] 刘炯，《一种基于 SIMD 指令集的高速有序抖动算法实现》，上海交通大学计算机科学与工程系工程硕士网站公示，http://www.csmoe.sjtu.edu.cn/.

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其它个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：刘炯

日期：2018年7月10日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□ 在 __年解密后适用本授权书。

本学位论文属于

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：刘炳 指导教师签名：姚天昉

日期：2018年7月10日 日期：2018年7月12日