中国科学技术大学

专业硕士学位论文

（专业学位类型）

**基于Spark协同过滤推荐系统研究与实现**

作者姓名： 张宁

专业领域： 软件工程

校内导师： 戴礼荣 教授

企业导师： 程宁 博士

完成时间： 2017年9月13日

University of Science and Technology of China

A dissertation for master’s degree

（Professional degree type）



**Research and implementation of collaborative filtering recommendation system based on Spark**

Author： Ning Zhang

Speciality： Software Engineering

Supervisor：Lirong Dai

Advisor： Ning Cheng

Finished time:

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：\_\_\_\_\_\_\_\_\_\_\_ 签字日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定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，可以将学位论文编入《中国学位论文全文数据库》等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、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开 □保密（\_\_\_\_年）

作者签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 签字日期：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 摘 要

大数据时代的背景下，信息的挖掘过滤对于用户来说是十分重要的。大多数应用场景，用户往往对需求不明确或难以表达，以及由于资源信息的过于丰富，很难从中找到自己需要的内容，这就是如今的信息过载的现象。推荐系统作为缓解信息过载问题的有效方式之一，在社交媒体中的作用也日趋重要。系统通过计算提取用户或商品的特征信息。根据一定的算法规则，从而实现更加主动、智能的为用户推送可能感兴趣的产品信息，从而帮助用户从海量数据中找到自己感兴趣的内容。

然而随着信息量的不断增加，以及推荐系统的应用场景的多样性，推荐系统的实现遇到了各种各样的挑战。其一、是推荐信息的时效性，由于用户某一时刻产生的行为日志信息随着时间推移所产生的推荐结果可能是无效的，因此如何根据用户当前行为，为用户提供实时的推荐成为了关键问题之一。其二，对于新用户或商品，缺少必要的特征信息，如何进行推荐(冷启动问题)。

本文针对上述问题，为了解决推荐系统的实时性和冷启动问题，设计实现了基于Spark实时计算的推荐系统。其的主要内容如下：

1）基于Spark分布式计算框架，根据需求，设计推荐系统基础架构，整个系统结构分为三层：一、实时流处理层，基于Spark Streaming对消息队列的数据进行实时流计算。二、离线计算层，用户处理耗时任务，为实时推荐提供数据基础。三、业务逻辑层，整合实时计算和离线计算资源。为上层结构提供推荐服务。

2）为了弥补传统的协同过滤算法的不足，提出了基于矩阵分解的协同过滤的异步分布随机梯度算法。

3）通过用户时间权重的大小，利用三分图的形式描述各个属性间的关系，进一步获取个性化的预测评分公式，从而解决推荐系统的冷启动问题。

**关键词**：信息过载 spark 协同过滤 冷启动

# ABSTRACT

In the background of large data age, the mining of information filtering is very important for users. In most scenarios, users are not clear or difficult to express their demand. Because of the huge number of resource information, users can not find useful information they need easily, which is the phenomenon of information overload. As an effective way to alleviate the problem of information overload, the recommendation system is becoming more and more important in social media. The system extracts the feature information of the user or commodity by calculation. According to certain rules of the algorithm to achieve a more active, intelligent recommend for users which product information they might be interested in. So that we can help users to find their own interesting information from massive data.

However, with the increasing amount of information and the diversity of the scenarios of the recommended system, the implementation of the recommended system has encountered various challenges. First, is the timeliness of the recommended information. As the result of the recommended action log information generated by the user at some point in time generated may be invalid. So how to provide users with real-time recommendations based on the user's current behavior has become one of the key issues. Second, for new users or goods, how to recommend them is another key issue because of lacking necessary feature information(cold start problem).

In order to solve the above problems, timeliness and cold start problems of recommender systems, a recommendation system based on Spark real-time calculation is designed and implemented. Its main contents are as follows:

1) Based on the Spark distributed computing framework, according to the demand, design the recommended system infrastructure, the whole system structure is divided into three layers: First, the real-time stream processing layer calculates the data stream of the message queue in real-time based on Spark Streaming. Second, offline computing layer, users deal with time-consuming tasks and provide the data base for the real-time recommendation. Third, the business logic layer integrates real-time computing and offline computing resources. Provide recommendation services for the superstructure.

2) In order to compensate for the shortcomings of the traditional cooperative filtering algorithm, a random gradient algorithm of asynchronous distribution based on matrix decomposition is proposed.

3) Through user time weight, the size of the figure in the form of three points is used to describe the relationship between attributes, further access to personalized prediction score formula, thus to solve the problem of the cold start the recommendation system.

**Key Words**: Printing Spark, Production,iOS,XML

**目 录**

[摘 要 I](#_Toc488932004)

[ABSTRACT 2](#_Toc488932005)

[第一章 绪论 5](#_Toc488932006)

[1.1 课题背景 5](#_Toc488932007)

[1.2 研究现状 6](#_Toc488932008)

[1.3 本文的研究内容 7](#_Toc488932009)

[1.4 论文结构 8](#_Toc488932010)

[第二章 相关工作 8](#_Toc488932011)

[2.1 Spark 分布式框架 8](#_Toc488932012)

[2.1.1 Spark简介 8](#_Toc488932013)

[2.1.2 Spark Core技术 9](#_Toc488932014)

[2.1.2 Spark SQL 10](#_Toc488932015)

[2.1.2 Spark Streaming 11](#_Toc488932016)

[2.1.3 Spark MLlib 技术 11](#_Toc488932017)

[2.1.4 Spark与Hadoop区别 11](#_Toc488932018)

[2.2 HDFS 分布式存储 12](#_Toc488932019)

[2.3 推荐算法 16](#_Toc488932020)

[2.3.1 推荐系统介绍 16](#_Toc488932021)

[2.3.2 协同过滤概述 17](#_Toc488932022)

[2.3.3 基于SVD的推荐 17](#_Toc488932023)

[2.3.4 基于人口统计学的推荐 18](#_Toc488932024)

[2.3.5 基于内容的推荐 19](#_Toc488932025)

[第三章 基于Spark 的推荐系统设计 20](#_Toc488932026)

[3.1 推荐系统实时流计算 20](#_Toc488932027)

[3.1.1 业务需求分析 20](#_Toc488932028)

[3.1.2 功能需求和系统整体结构 20](#_Toc488932029)

[3.2 推荐系统离线计算 22](#_Toc488932030)

[3.1.1 业务需求分析 22](#_Toc488932031)

[3.1.2 功能需求和系统整体结构 22](#_Toc488932032)

[3.3 相关详细介绍 23](#_Toc488932033)

[3.3.1 分布式日志框架 23](#_Toc488932034)

[3.3.2 推荐引擎组 24](#_Toc488932035)

[3.3.3 推荐系统整体架构 25](#_Toc488932036)

[第四章 关于推荐算法的相关问题研究 26](#_Toc488932037)

[4.1 基于item相似度计算 26](#_Toc488932038)

[4.1.1 余弦相似度计算 26](#_Toc488932039)

[4.1.2 基于相关的相似性计算 26](#_Toc488932040)

[4.1.3 调整余弦相似度 27](#_Toc488932041)

[4.2 预测计算 27](#_Toc488932042)

[4.2.1 加权求和 28](#_Toc488932043)

[4.2.2 回归 28](#_Toc488932044)

[4.2.3 性能影响及意义 28](#_Toc488932045)

[4.3 基于矩阵分解的协同过滤的异步分布随机梯度算法 31](#_Toc488932046)

[4.3.1 算法概述及引入原因 31](#_Toc488932047)

[4.3.2 基于Item的矩阵计算 31](#_Toc488932048)

[4.3.3 随机梯度优化 32](#_Toc488932049)

[4.4 关于算法冷启动问题的研究 34](#_Toc488932050)

[4.4.1 冷启动问题概述 34](#_Toc488932051)

[4.4.2 解决方案 34](#_Toc488932052)

[第五章 实验与测试 37](#_Toc488932053)

[5.1 实验环境 37](#_Toc488932054)

[5.2 数据集 37](#_Toc488932055)

[5.3 评价指标 38](#_Toc488932056)

[5.4 实验结果 38](#_Toc488932057)

[参考文献 82](#_Toc488932058)

[致 谢 84](#_Toc488932059)

# 第一章 绪论

## 课题背景

随着大数据时代的到来，信息资源日益丰富。然后海量数据的出现也伴随着很多问题，例如“信息过载”[1][3]现象，大量的信息也给人们的生活带来了一定的困扰。因此无论是用户要想从海量数据中找到自己需要的信息，还是商家想要引领消费者进行消费，都离不开个性化推荐系统。为了更好的解决“信息过载”问题，推荐系统的概念应运而生，并且已经得到了越来越多的关注，相关的专家学者们对其研究也更加深入。因此，如何解决大数据时代的实时准确推荐是一个非常重要的课题。

推荐系统的出现则在一定程度上弥补捜索引擎的缺陷。推荐引擎用于处理信息过载的情况，通过分析日志数据、用户历史行为、用户属性等信息创建用户兴趣偏好模型[2]，依靠推荐算法预测用户对商品权重偏好并进行排序，从而帮助用户没法准确定位自身需求描述的同时，向用户推送满足需求与兴趣的推荐列表。因此推荐系统和搜索引擎相互补充，当用户有明确需求时则采用捜索引擎搜索相关信息，而在用户没有明确自身需求或者以新成员身份进入到一个新的环境中时推荐系统可以帮助他们获取到感兴趣的内容。因此推荐系统可以通过该方式在电子商务平台上获取广泛应用。然而如何推送高质量推荐列表、快速地响应推荐请求、实时生成推荐列表、海量数据分析都成为必须面对和解决的难题。

随着购物网站规模的不断发展，用户和项目的数量不断增多，这也就导致了用户项目评分阵的稀疏性问题越来越无法避免，协同过滤算法中的数据稀疏性过于严重，推荐的结果更加不准确，从而影响用户对购物网站的信任，造成用户的流失。并且在协同过滤的推荐算法中，往往不能很好的展现用户的爱好迁移的问题，因此也会对推荐系统的性能造成一定的影响[5]。所以，缓解数据稀疏性问题和适应用户喜好化问题已经成为了网络购物系统发展的严峻考验之一。

在新一代推荐系统中为了解决如上难题以及保证系统的可扩展性，需要存储不断增长的数据集以及快速并行化处理。常用的技术主要包括：大数据存储技术以及大数据处理技术。推荐算法主要特点在于多个并行操作之间重用数据工作集，属于非循环应用模型的应用，因此引入SPARK计算框架[17]。SPARK是由UC Berkeley推出的大数据处理框架，它提出分布式内存抽象，称为弹性分布式数据集RDD，它允许在计算过程中将中间工作集缓存在内存中，而在后续操作中重用该工作集，运样可以极大加速处理过程。由于大部分数据挖掘算法设及到数据集迭代计算，因此SPARK框架更符合推荐算法的需求，本课题利用改进后的协同过滤算法[4]和用SPARK大数据处理技术实更快、更准确的推荐平台。

## 1.2 研究现状

推荐系统[6]的研究至今已有约20多年的历史，其最早期的研究源自于美国明尼苏大学GroupLens研究小组对一个叫MovieLens的电影推荐系统的研究。研究内容主要是对网站用户打分情况进行分析，从而得到用户可能感兴趣的电影，从而进行推荐。早期的算法主要是基于内容的推荐，通过对推荐物品本身进行特征提取。该方式会导致如果用户没有对某类产品评价，该类产品就永远不会被推荐。随着时间推移，旧的推荐系统问题逐渐暴露，新的推荐算法应运而生。

协同过滤推荐（Collaborative Filtering，简称CF）起源于1992年。从算法理论方面来说，最早期的个性化推荐算法[7]是GroupLens小组提出的协同过滤算法(即基于用户的协同过滤推荐算法)，至于为止该算法在推荐领域仍占有一席之地。基于该算法，人们陆续提出了基于物品的协同过滤推荐算法，基于模型的协同过滤推荐算法等等。经过近20多年的发展，当前推荐系统衍生出多种多样的推荐方式。一、基于上下文的推荐，即推荐系统会根据用户的访问时间、所在城市、天气、季节等因素研究给用户生成TopN推荐，提高推荐效果。二、基于用户画像的推荐，所谓用户画像，就是想用户的所有特征信息标签化，是对用户相关联数据可视化的一种展现形式。多角度抽象用户信息，从而实现个性化推荐系统等。三、基于协同过滤[8]的推荐系统，其中有分为两大类，其一是基于内存的协同过滤推荐算法，包括基于用户的协同过滤推荐算法和基于项目的协同过滤推荐算法等等，其二是基于模型的协同过滤推荐算法，即利用数学建模的手段，评估你的意图，包括基于贝叶斯的协同过滤推荐算法、基于隐语义的协同过滤推荐算法和基于线性回归的协同过滤推荐算法等等。四、基于内容的推荐、即对数据内容进行分析并得到物品的结构化描述信息，分析用户在过去时间中评分活评论的商品，利用用户画像构建预测模型，从而生成推荐策略。五、基于机器学习的推荐系统、根据数据的多样性选取合适的算法策略，自动的归纳逻辑和规则，并根据归纳总结的结果(模型)来对未来数据进行推荐。除此之外，还有基于关联规则的推荐、基于效用的推荐、基于知识的推荐等等。但是，上述每种单一方式的推荐系统都有各自的缺点，比如基于内容的推荐有严重的矩阵稀疏问题，基于关联规则的推荐较为耗时只能做离线处理等等。

由于各种推荐方法都有优缺点，因此目前主流推荐系统都是以上各个策略的混合，研究和应用最为广泛的是基于Item的推荐和协同过滤推荐的组合[8]。尽管从理论上有多种组合推荐的方式，但并非对于某一具体问题一定有效，组合推荐中最重要的原则是通过各种策略的组合，使得各自的推荐技术能够相互弥补，从而实现精准高效的推荐。

## 1.3 本文的研究内容

    本文主要研究基于Spark平台推荐系统技术应用与优化，着重研究推荐系统架构设计以及系统的推荐效率以及相关算法优化。文中详细讨论基于Spark的实时推荐的设计与实现，以及该系统中其它基础设施的原理与框架架构，以及基于Spark的RDD算子优化算法执行效率。同时对协同过滤推荐算法进行研究，着重介绍基于Item的协同过滤推荐算法的优化和改进，根据其不足引入了基于矩阵分解的协同过滤推荐算法，并采用随机梯度优化方法对该算法进行进一步优化。最后对于推荐系统的冷启动问题作出相关介绍，并对此提出解决方案。具体研究内容如下：

    (1) Spark生态系统的相关研究。分析Spark系统架构以及生态系统，了解Spark RDD数据模型，通过函数式编程的模式把分布式数据处理转换成分布式数据集。使用以命名列方式组织的分布式数据集DataFrame代替直接对Spark RDD做直接操作，使得计算更加简单。用SparkSQL读取Hbase中的数据转换成DataFrame,将数据加载到内存，通过并行化计算完成相关数据分析。针对系统实时性问题，设计能够实现实时计算的流处理架构，将离线计算与实时计算相结合，根据用户在线实时行为和其历史行为数据实时计算，并给出推荐结果。

    (2) 协同过滤算法的相关研究。重点研究了基于Item的协同过滤算法，分析其性能瓶颈，测评算法的推荐准确性。并引入了基于矩阵分解的协同过滤异步分布随机梯度算法，该方法的主要思想是将用户评级矩阵分布到不同机器上并且通过网络异步的传播随机梯度优化的更新。一旦机器收到一个更新的权重，它会使用自己的更新对参数权重进行平均，这样可以保证每台机器能够全面了解整个网络数据情况，实验证明，该方案具有更好的收敛性。

    (3) 推荐系统冷启动问题。根据当今推荐系统冷启动问题的现状，提出了解决该类冷启动问题的推荐算法，在此我们引入了用户时间权重的概念，通过对用户评价的时间间隔判断用户活跃度并根据用户对新项目偏爱程度等因素创建构建个性化的预测评分公式。综合考虑项目本身的诸多属性信息、标签对项目的标注信息等较为全面的考虑影响推荐系统的诸多因素。分别根据用户-项目-标签信息和用户-项目属性信息来挖掘他们之间的潜在关系。从而通过预测评分值来解决新项目的冷启动问题。

## 1.4 论文结构安排

本文余下部分组织如下：

第二章介绍了论文的相关工作：主要是介绍了论文中使用到的关键技术，如Spark 的相关技术栈，以及HDFS分布式文件系统以及Hbase数据库的原理。并对推荐算法以及协同过滤相关算法做了大致的介绍。

第三章主要介绍了基于Spark推荐系统的系统设计，包括离线计算和实时计算部分，以及它们各自使用到的技术以及在整个推荐系统的角色。整个系统以数据收集、数据存储、数据计算、数据合并、数据存储为主线进行描述。

第四章主要介绍了推荐算法的相关问题以及推荐系统的冷启动问题，包括基于Item的推荐算法的优化，以及对性能影响的因素进行分析。引入了矩阵分解的协同过滤的异步分布随机梯度算法，更好的优化了集群场景下算法的收敛时间。此外对推荐系统冷启动问题进行分析研究、通过用户时间权重，以及项目本身的属性和标注信息，挖掘其潜在的关系，以此改善协同过滤算法的冷启动问题。

第五章对改进的算法使用标准数据集进行测试，并对相关的实验结果进行分析。

第六章总结与展望，对本文的相关工作做出总结并对下一步工作进行展望。

# 第二章 相关工作

## 2.1 Spark 分布式框架

### 2.1.1 Spark简介

Spark发展成立于2012年，以利用当时的五年WordPress经验。 近四年半之后，Spark与广泛的客户合作，从个人和中小企业到跨国公司和政府机构。在2015年底之前，Apache Spark已经成为Hadoop生态系统中最为活跃的项目。作为2009年加州大学伯克利分校AMPLAB研究项目的一部分，Spark与Apache Hadoop并仍在积极发展。 2015年有三个版本，从1.3到1.5，分别包含DataFrames API，SparkR和Project Tungsten等功能。版本1.6在2016年初发布，包括新的Dataset API和数据科学功能的扩展。 Spark 2.0于2016年7月发布，这是一个主要版本，有很多新功能和增强功能，值得拥有自己的一部分。

目前，在Spark核心框架基础上，其主要提供了四个范畴的计算框架其中包括Spark Streaming, Graphx, MLbase, SparkSQL等，其中Spark Streaming是Spark核心API的一个扩展，它对实时流式数据的处理具有可扩展性、高吞吐量、可容错性等特点。Spark Streaming 可以从各种数据源中获取数据并进行复杂计算后推送到文件系统、数据库等存储介质保存。Spark Streaming接收实时的输入数据流，然后将这些数据切分为批数据供Spark引擎处理，Spark引擎将数据生成最终的结果数据。Graphx主要并行图计算，Spark Graphx性能良好，又具有非常丰富的API,并可以基于海量数据进行复杂的图计算从而得到了较为广泛的应用。MLbase 作为Spark生态系统中的一部分，由AMPLab实验室的Mlbase团队开发，是Apache Spark中的核心机器学习库，专门提供机器学习相关的丰富的API。从而让机器学习的门槛更低，从而使那些不了解机器学习的开发人员也能通过MLbase来处理自己的数据。Spark SQL的引入专门用于处理结构化数据比如（DB,JSON），Spark SQL 不仅是SQL,它对各种数据源做了抽象，能够对各种数据源进行ETL操作，并提供丰富的对复杂数据读写的API,使用非常方便，并对多种数据源都提供了很好的支持。其中DataFrame API作为一种具备高伸缩性的数据操作方式，相比RDD API 效率更高。此外Spark SQL还提供了Catalyst Optimizer查询优化器，可以自动优化程序，使得程序更加高效。

Spark引入的弹性分布式数据集RDD，是一种高度受限的共享内存模型，是基于内存的。 这使得其计算效率和速度相比Hadoop 的 MapReduce （基于磁盘，每次执行时都要从磁盘读取数据，计算完毕后都要把数据存放到磁盘上）提升很多。

Apache Spark是围绕速度、易用性和复杂分析构建等方面的大数据处理框架，适用于多种不同的情形的分布式应用场景，包括批处理，流处理，交互式查询等等，并大大减轻了原先需要对各种平台分别管理的负担。统一的RDD抽象和操作，基于统一的技术堆栈，使得Spark称为大数据通用计算平台。在大数据时代逐渐普及的背景下得到了越来越广泛的应用。

### 2.1.2 Spark Core技术

Spark Core是spark整个生态系统的核心，它主要是提供了一套基于RDD的API以及分布式的计算引擎。Spark采用了Master-Slave的分布式计算模型，Master作为主节点，负责任务调度，管理Worker节点，并命令Worker 启动Executor。Worker则是计算节点，负责管理本节点资源已经接收Master的命令并向Master汇报心跳。此外Spark还引入了弹性分布式数据集RDD(Resilient Distributed Datasets)，它是一个容错的、并行的数据结构，可以让用户显式地将数据存储到磁盘和内存中，并能控制数据的分区。同时，RDD还提供了一组丰富的操作来操作这些数据。下面具体介绍spark 执行流程：

如图2-1所示，当RDD算子遇到Action方法时，DAG图即确定了边界，DAG图形成，随后将DAG提交给DAGScheduler。DAGScheduler 会将DAG切分成多个stage,切分依据(宽依赖—shuffledRDD—即数据需要网络传递) 。多个stage封装到TaskSet后提交给TaskScheduler ，随后TaskScheduler把任务提交给worker执行。其中DAGScheduler 和TaskScheduler都在Driver端，main函数创建SparkContext时会使得driver和Master建立连接，Master会根据任务所需资源在集群中找到符合条件的worker。随后Master对worker进行RPC通信，通知worker启动Executor ，Executor会和Driver 建立连接，随后的工作worker和Master不再有关系。 随后Driver会向Executor提交Task。启动Executor。



图2-1 Spark任务执行流程

### 2.1.2 Spark SQL

基于Spark Core，SparkSQL构建了一套基于SQL的引擎。该引擎有两层构成，最底层是查询优化器Catalyst Optimizer, 可以优化程序，使得SQL的运行更加高效。比如两个表的join操作，有多种方式，其一、轮询操作，具有O(n^2)的的时间复杂度。其二、先排序再遍历，该时间复杂度为O(nlogn)。其三则是基于Hash的一种实现，其时间复杂度是O(m+n)。Catalyst Optimizer则会根据其数据分布，数据的特点等应用场景生成合适的执行计划，从而达到SQL高效运行的目的。



图2-2 Spark SQL整体架构

在Catalyst Optimizer之上，Spark SQL提供了两种编写应用程序的方法，一种是经典的SQL方式，另一种是DataFrame/DataSet API 。由于Spark SQL相比于Spark Core显著的优势，使大部分程序的开发不再基于RDD API （如MLlib）而转向基于DataFrame/DataSet API进行开发。

### 2.1.2 Spark Streaming

Spark Streaming类似于Apache Storm，主要应用于流式数据计算，巨局有高吞吐量和容错能力强等特点。Spark Streaming支持的数据输入源很多，例如：Kafka、Flume、Twitter、ZeroMQ和简单的TCP套接字等等。而结果也能保存在很多地方，如HDFS，数据库等。另外Spark Streaming也能和MLlib（机器学习库）以及Graphx完美融合。

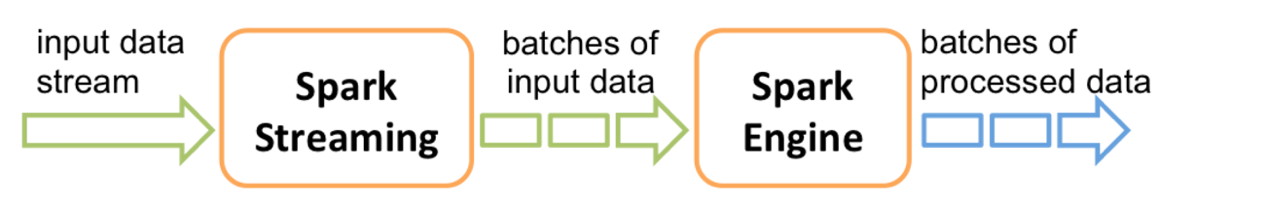


图2-3 Spark Streaming 工作流程图

Spark Streaming 主要功能是根据时间片的长短，将流数据累积为RDD, 然后再将每个RDD进行批量处理，从而达到大规模处理流数据的作用。由于这样的一种模式，使它的吞吐量比现在主流框架Storm还大，同时，它还提供了丰富的API 用于流数据计算中。具体的相关应用会在实时计算章节进行展开。

### 2.1.3 Spark MLlib 技术

机器学习和统计学技术是从复杂的海量数据中挖掘有用的信息，对于一般开发者来说，即使了解其中的相关算法，但是要想实现，也是有一定困难的。因此MLlib的愿景就是解决开发者的这些问题，MLlib 构建在apache spark之上，是一个专门针对大量数据处理的通用的、快速的引擎，并实现了一些常见的机器学习算法和实用应用程序，包括分类、回归、聚类、协同过滤、降维以及相关的底层优化。这样只要具备一定的理论知识，便可以进行机器学习及统计学方面的开发了。

### 2.1.4 Spark与Hadoop区别

Hadoop和Apache Spark都是大数据框架，但它们并不真正具有相同的用途。 Hadoop本质上是一个分布式数据基础架构：它在大量服务器群集中的多个节点之间分发海量数据集，这意味着您不需要购买和维护昂贵的定制硬件。 它还可以索引并跟踪数据，从而大大提高了数据处理和分析效率。 另一方面，Spark是一种数据处理工具，可以对这些分布式数据集合进行操作; 它不做分布式存储。

Hadoop不仅包括称为Hadoop分布式文件系统的存储组件，还包含一个名为MapReduce的处理组件，因此您不需要Spark来完成处理。 相反，您也可以使用Spark而不使用Hadoop。Spark是一个多阶段的具有RAM能力的计算框架，具有机器学习，交互式查询和图形分析的库。 Spark并没有自己的文件管理系统，所以它需要与一个HDFS（而不是另一个基于云的数据平台）集成。 Apache Spark非常适合批处理和流处理，这意味着它是一个混合处理框架。 Spark通过内存计算和处理优化加快了批处理速度，因此其速度通常比MapReduce快得多。 这是流式传输工作负载，交互式查询和基于机器学习的不错选择。 Spark也可以与Hadoop及其模块一起使用。 实时数据处理能力使Spark成为大数据分析的首选。

Hadoop和Spark都是开源项目，因此免费。 但是，Spark使用大量的RAM来运行内存中的所有内容，RAM比硬盘更昂贵。 Hadoop是磁盘绑定的，因此节省了购买昂贵的RAM的成本，但需要更多的系统来分配多个系统上的磁盘I / O。

就成本而言，如果要处理大量的大数据，Hadoop将会更便宜，因为硬盘空间比内存空间要低得多。

## 2.2 HDFS 分布式存储

对于大数据处理应用来说，依靠集中式的物理服务器来保存数据是不现实的，容量和数据传输速度都会成为瓶颈。要解决这样的问题，现在最好的解决方案就是数据分开存储的几十台，几百台甚至更多的机器上。为了统一管理这些机器上存储的数据，必须要使用一个特殊的文件系统——分布式文件系统。因为是基于网络的，众多网络程序具有的病症也威胁这它的工作，这使得分布式文件系统要比常规的硬盘文件系统更加复杂。例如，如何在不同的机器之间保证数据的可靠性、可恢复性等。下面介绍一下现阶段较为流行的分布式文件系统HDFS.

### 1) 数据块Block

HDFS文件系统为了提高磁盘的读写效率，引入了数据块(Block)概念，作为文件系统的存储数据的最小单元，其默认大小为64MB，也可以根据项目具体需求进行设置一般为128MB，设置较大的目的是为了减少寻址开销，数据的备份也是以块为单位。

### 2) NameNode和DataNode

HDFS采用Master/Slave的体系结构，其中NameNode节点作为集群的主服务器，用于管理整个文件系统的元数据(Metadata)，进行文件系统的相关命名空间操作，如打开、关闭和重命名文件夹和目录等，并确定数据块到DataNode的映射。若干个DataNode节点作为HDFS集群的一般节点，负责管理用户的文件数据块，数据存储、并定期向NameNode汇报存储的文件信息、DataNode还可以根据NameNode的指示执行块创建，删除和复制。并维持NameNode之间的心跳。在系统内部，文件被分割成一个或多个Black块(文件系统可读写数据的最小单元)中，并且这些数据块被存储在一组DataNode服务器中。



图2-4 HDFS架构图

### 3) NameNode管理元数据机制

NameNode中仅仅存储文件目录树的相关信息，具体则存储在fsimage(内存镜像文件)中，当NameNode启动时，会先读取该文件，并将目录树加载到内存中。而edits文件存储的是数据操作的日志信息，如图2-5。



图2-5 节点目录树

当有Metadata进行更新操作时，首先在NameNode中的内存中进行，并将此次操作记录在edits中。该记录并不代表Metadata本身，但可以根据该记录恢复Metadata信息，如果NameNode宕机，仍然可以根据操作日志恢复历史数据。如果随着NameNode运行时间的增加，edits文件会越来越大，下次服务器启动会消费较长的时间来合并edits生成新的fsimage镜像文件。因此引入了secondNameNode, secondNameNode会定期通过rpc请求NameNode询问是否触发checkpoint，NameNode会根据自定义规则(运行时间/edits数据大小)返回给secondNameNode是否满足checkpoint条件。如果满足条件，secondNameNode会通知NameNode滚动刷新正在写入的edits\_inprogress文件，并下载该文件和之前历史的edits文件以及历史的内存镜像文件fsimage，并加载到内存，通过操作日志更新fsimage,生成最新的fsimage.chkpoint,并上传到NameNode,覆盖历史的fsimage文件。具体流程图如图2-6 所示：

图2-6 元数据管理机制

其中checkpoint条件触发在配置文件中进行设置：

dfs.namenode.checkpoint.check.period=60 #检查触发条件是否满足的频率，60秒

dfs.namenode.checkpoint.dir=file://${hadoop.tmp.dir}/dfs/namesecondary

#以上两个参数做checkpoint操作时，secondary namenode的本地工作目录

dfs.namenode.checkpoint.edits.dir=${dfs.namenode.checkpoint.dir}

dfs.namenode.checkpoint.max-retries=3 #最大重试次数

dfs.namenode.checkpoint.period=3600 #两次checkpoint之间的时间间隔3600秒

dfs.namenode.checkpoint.txns=1000000 #两次checkpoint之间最大的操作记录

## 2.3 Hbase 数据库

### 2.3.1 Hbase概述

 Hbase是一个分布式的、面向列的开源的key-value数据库，其数据是按照key的字典序顺序存储，Hbase区别与传统的数据库，是一个面向列的数据库，因此对于schema限制很少，可以自由的添加字段，对半结构化数据的存储提供条件。Hbase基于Hadoop的HDFS文件系统，数据存储在hdfs上，备份机制健全。支持线性扩展，用Hbase技术可在廉价的PC server上搭建大规模结构化集群，其容错能力和处理能力随集群数量增长而变强，可以处理PB级以上的数据，并且是一个强一致性的K-V系统。

Hbase也是主从结构，主节点HMaster,从节点    Hregionserver,  HMaster只管理数据偏移和表的信息，不存储实际数据。Hbase作为缓存层，当数据到达一定数量后再由内寸写入HDFS中，从而减少HDFS元数据个数。

### 2.3.2 Hbase表的数据结构

1) 行：同一个Key对应的所有数据，其数量没有上限。

2) 列簇：相似的列通常会被划分到一个列簇，列族的数量有限，通常是在 建表是定义好，后期无法动态增加。

3) 列：一个列簇中可以有多个列，列名在数据写入中确定，列的数量没有 明确的限制。

4) Cell及时间戳：Cell用户存放每个列的内容，其内容不能修改只能追加， 因此每个Cell都可以有任意多个版本。



图2-7 Hbase数据库表示例图

Hbase的表类似于传统的DBMS表，数据按row-key的值从小到大排列，数据被组织到若干个Region中，一张表会被切分成若干个Region，Region是Hbase作为负载均衡调度的最小单位，Hbase之所以可以提供分布式的服务，是因为它可以把一个从宏观上看是一个整体的表，切分成若干个Region，分别由不同的机器管理。

Hbase作为Hadoop的子项目，对Hadoop的Map-reduce有着良好的支持，可以通过在Hadoop上的Map-reduce任务来并发的对Hbase表进行并发Scan(扫库)操作。

### 2.3.3 Hbase系统组成



图2-8 Hbase系统结构图

如上图（2-8）为hbasex系统结构图，其中 Master节点，可以由一台或多台服务组成，用于实现Region Server集群的负载均衡，当某一台的Region Server数据量过多时，Master节点会发起负载均衡请求，使该Region Server上的数据迁移到较为空闲的Region Server上。此外Master还用于管理MetaData的相关的事务控制等操作。

Hbase中每个表被分成多个Region，每个region会被分配到多个RegionServer上，RegionServer可以看做是该架构中的slave节点，主要用户读写数据的操作，完成Master的相关指令。RegionServer由Zookeeper集群统一管理，Zookeeper集群可以提供分布式环境的多机协调功能，当某一个RegionServer宕机时，Zookeeper的通知机制会使得Master节点感知到RegionServer的宕机事件，并将该RegionServer上的数据分配的其他服务器中。Zookeeper还提供分布式锁的服务，在为多用户提供服务的系统中，多个用户可能在同一时刻对关键资源进行访问，通常我们会在程序中添加锁来控制，而对于分布式的系统则是通过Zookeeper来完成。

## 2.4 推荐算法

### 2.4.1 推荐系统介绍

推荐系统将知识发现技术应用于在实时互动中的信息，产品或服务制定个性化建议的问题。在这些系统，特别是基于k-最邻近法（k-nearest）协同过滤的系统已经在网络上取得了广泛的成功。近年来，可用信息量和网站访问量的巨大增长形成了对推荐系统的一些关键挑战。这些挑战是：产生高质量的建议，为数百万用户和项目在每秒钟内提供大量的建议，并在面对较少的数据的情况下实现高覆盖率。在传统的协同过滤系统中，工作量随着系统参与者的数量而增加。因此需要一个即使在非常大规模的问题的情况下，也可以快速产生高质量建议的新的推荐系统技术。为了解决这些问题，我们探讨了基于项目的协同过滤技术。基于项目的技术，首先分析用户项目矩阵以识别不同项目之间的关系，然后使用这些关系间接计算用户的建议。在章节中，我们分析了不同的推荐算法，已经其各自的优缺点。

### 2.4.2 协同过滤概述

协同过滤算法的目标是根据用户以前的喜好和其他志同道合的用户的意见来建议新项目或预测特定用户的某个项目的效用。 在典型的CF场景中，有一个m个用户U={u1,u2,…..um}列表和n个项目I={i1,i2,….in}的列表。 每个用户ui都有一个项目列表Iui，来表达了他/她的意见。 用户可以通过分析定时日志，通过挖掘Web超链接等来将用户的意见明确地进行评分操作，通常这些分数在一定数值范围内，或者可以隐含地从采购记录中导出。 请注意，和Iui可能是一个空集。 存在一个被称为活动用户的专用用户，协同过滤算法的任务是为了找到可以是两种形式的项目可能性。

预测是一个数值，Pa, j，表示对于活跃用户Ua的项目ij ∉Iua的预测可能性。 该预测值与Ua提供的意见值在相同的比例内。

推荐是N个项目的列表，Iτ∈I,是活跃的用户最喜欢的项目。 请注意，推荐列表必须是活动用户尚未购买的项目，即 。 CF算法的这个接口也被称为N类（Top-N）建议。

图2-1显示了协同过滤过程的示意图。 CF算法将整个m x n个用户项目数据表示为评级矩阵A。A中的每个条目ai, j表示第i个用户在第j个项目上的偏好分数（评级）。 每个单独的评分都在数字范围内，并且也可以为0，表示用户尚未评估该项目。 研究人员设计了一些协同过滤算法，可以分为两大类：基于内存（基于用户）和基于模型（基于项目的）两种算法。 在本节中，我们提供了基于CF的推荐系统算法的详细分析。

图2-9协同过滤过程

### 2.4.3 基于SVD的推荐

在评分系统中，如果建立一个二维评分矩阵，则这将是一个巨大的矩阵，假设用户有10000个，而物品也有10000个，那么这个矩阵的规模就是10000\*10000=1亿。对于大型电商动辄数以亿计的用户及物品而言，这个矩阵的规模更是无法衡量，更重要的是，该矩阵很可能非常稀疏。例如Netflix数据集中非零元素个数只占所有元素个数的1%左右，这种稀疏性给存储和计算带来了巨大的挑战。

奇异值分解(SVD)是一种分解技术。假设A是一个m行n列()的矩阵，则对A进行奇异分解如下：

(公式2.1)

其中∑是一个m\*n的矩阵，除了对角线的元素外，其他的值都是0，而对角线上的非零元素成为奇异值。令s=(), s中的每一个元素为一个奇异值，且按由大到小排列，则一般s中元素会减小的很快，当我们去前r个奇异值进行降维时，即得到：

(公式2.2)

其中r远远小于m和n，这样 , , 三个矩阵的计算会比基于A的运算快很多。

基于SVD的推荐就是先将评分矩阵进行SVD分解，选取适当的奇异值个数r进行低阶近似，得到，,三个矩阵，然后基于这三个矩阵进行推荐。

基于SVD的推荐本质上还是利用用户行为数据，只是在推荐前后先离线对评分矩阵进行分解，然后进行降维，从而大大减小存储的数据量和在线计算量。但它也有以下两个主要缺点：

1) 降维时r维不好确定。r太大，则降维效果不好，r太小又会损失太多信息。在实际使用过程中往往根据奇异值的衰减程度确定r值。

2）SVD本身计算量就非常大，当数据量比较大时，消耗的时间会难以忍受。不过现在Spark已经实现了SVD的并行化计算，当r值选取的较小时，Spark的计算速度还是比较快的。

### 2.4.4 基于人口统计学的推荐

基于人口统计学的推荐是利用人的静态描述信息进行推荐的方法。它简单的根据人口统计信息发现用户的相似度，然后给当前用户推荐与他相近的人喜欢的物品。

具体来说，该方法首先会利用用户的统计信息，如年龄，性别，住址等，然后采用特定的相异度计算策略，计算用户之间的距离。实际计算时用户属性类别往往是不同的，例如性别是二元属性，年龄是序列属性，职业是名称属性等。假设数据集含有p个混合属性，对于对象i和j，我们采用下面的公式计算它们的距离：

(公式2.3)

公式中是属性f的指示符，是含有属性f的记录i和j的相异度。

如图2-10所示，左边是用户甲、乙、丙的信息，根据公式（2.3），用户的属性均是名称类或者二值类属性，计算后得到的用户丙和用户甲的距离为0，而和乙的距离为1，因此用户丙和用户甲更加相似，故我们就能将甲喜欢的东西向用户丙推荐。这里的邻居数只有一个，在实际应用中，可以通过调整邻居数目进行覆盖率更高的推荐。



图2-10基于人口统计学的推荐

### 2.4.5 基于内容的推荐

基于内容的推荐是利用物品的内容信息，先算出物品的相似度，然后根据用户过去喜欢的物品，来向用户推荐与这些记录相似的物品。物品间相似度计算方法很多，这里采用Jaccard公式：

(公式2.4)

或者采用简化的余弦相似度公式：

(公式2.5)

公式（2.4）和公式（2.5）中的表示电影u与电影v的相似度，表示电影u的内容，表示电影v的内容。

如图2-11所示，电影A类型为“爱情，浪漫”，电影B类性为“恐怖，惊悚”，电影C类性为“爱情，浪漫”使用公式2.5计算的到电影A与B的相似度为0，电影A与C的相似度为1，用户甲喜欢电影，故系统会将C推荐给甲。



图 2-11基于内容的推荐

# 第三章 基于Spark 的推荐系统设计

## 3.1 推荐系统离线计算

### 3.1.1 业务需求分析

    推荐系统在数据基础构建阶段有大量的耗时操作比如模型训练，中间数据计算如：线性分类器参数，协同过滤中item的相似矩阵的计算，用户画像构建，关联规则的发现等，这些计算通常比较复杂并且有大量的数据库查询操作，计算周期长，有且是算法的瓶颈，需要通过离线计算获得。然后为实时推荐提供数据基础，通过实时数据校正计算结果，从而达到近乎即时的推荐。

### 3.1.2 功能需求和系统整体结构

新数据产生时，我们通过flume/sqoop（用于关系型数据库）将其收集到hdfs中，每半小时存放到hdfs的一个目录中，然后进行批处理(可以使用mapReduce或者spark)处理完后倒入impala, impala是一个sql引擎，可以通过sql语句进行查询。

批处理层特点，因为写到了hdfs系统中，因此数据不可修改，只可以追加，可通过spark或map-reduce进行任意计算，支持水平扩展（通过扩展机器），缺点是高延迟，根据计算量或者数据量的不同，运行时间可能是几分钟到几个小时。 具体设计架构见图3-2。



图3-2 离线计算系统架构

该模块主要承担两个主要职责，一、存储Master DataSet, 即一个持续增长的数据集。数据来源于系统的实时数据采集模块。二、对Master DataSet 进行预计算。离线计算的模式与批处理为主，我们使用Hadoop的Map-Reduce方式进行：

|  |
| --- |
| 批处理计算伪代码： |
| function runOffLineCompute():  while(true)  recomputeBatchViews() |

利用离线计算模块进行数据预处理的主要作用是将实时计算的数据量尽可能的减小，从而有效地利用资源，改善实时计算的性能。但这里有一个前提，就是我们需要预先知道实时计算需要的数据，如此才能在离线计算中安排执行计划，定期对数据进行批量处理。此外，还要求这些预运算的统计数据是支持合并 。

 一、功能需求

(1) 数据收集

数据采集我们使用的是flume框架，该框架主要用于采集非关系型数据-如log，flume为收集、移动大量的日志数据提供了一种可靠的可用的分布式框架，它具有基于流式数据简单灵活的架构，并具有故障转移和恢复机制，为系统提供一定的可靠性和容错能力。flume需要在产生日志的agent结点上部署,用于监听节点产生的数据，agent节点可以将收集的数据传输给collector, 然后collector对数据流进行汇总(该设计的主要目的是为了减少hdfs中小文件的个数，减轻hdfs的并发压力)，并将其写入hdfs中。



图3-3 flume数据采集流程图

（2） 用户评分模块设计

该模块用户存储用户历史评分行为，为实时计算提供数据查询。本小节主要涉及到用户ID,商品ID,商品类别ID,用户-商品评分,时间戳五个字段，其中用户ID和商品ID分别代表用户和商品的唯一标识，用户-商品评分代表用户对商品的喜爱程度，范围是1-5的整数，喜爱程度与数值大小成正比。详细数据表如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用户ID | 商品ID | 商品类别 | 用户-商品评分 | 时间戳 |
| 1 | 2312 | 13 | 3 | 1502204779 |
| 2 | 4212 | 22 | 5 | 1502207499 |
| 3 | 3211 | 9 | 1 | 1502208499 |
| 4 | 4212 | 6 | 3 | 1503207900 |

当用户对商品进行打分时，我们会在相应的log文件中记录这条评分数据，包括表1的5个字段，其中两个字段间用双冒号连接，得到的数据格式为：

3::3211::9::1:: 1502208499。随后我们会将数据批量写入到HDFS中。供离线的批量计算使用。

（3）Spark离线计算数据准备

推荐系统离线计算由后台守护线程定期读取HDFS行为日志文件大小，若该文件大小超过指定阈值，便会触发离线计算部分。我们会读取HDFS的日志数据并将其转化成便于计算的DataFrame的数据结构。

|  |
| --- |
| spark分布式离线批处理部分源码 |
| case class RatingDate(userId:Int, itemId:Int, catId:Int, rating:Float,timestamp:Long)  def parseData(str:String): RatingDate = {  val fields = str.split(“::”)  RatingDate(fields(0).toInt, fields(1).toInt, fields(2).toInt, fields(3).toFloat,  fields(0).toLong)  }  //不定期根据URL读取hdfs数据  val conf = new SparkConf.setAppName(“test”)  val sc = new SparkContext(conf)  //将hdfs数据转换成DataFrame  val ratings = sc.testFile(“hdfs:”+url).map(parseData).toDF()  //进行相关算法计算并返回评级矩阵  //… |

## 3.2 推荐系统实时流计算

### 3.2.1 业务需求分析

  对于推荐系统而言，随着应用场景的多样性，以及许多业务场景的特殊性，对推荐系统实时性的要求日益提高，如果推荐系统不能根据用户的行为变化和商品信息的变化做出及时的结果更新，推荐结果的有效性可能会大打折扣，用户可能会因此无法获取有效的符合自身需求的推荐结果。

传统的推荐方式采用定时离线计算来更新推荐结果，该方式计算周期往往以小时、天为单位。推荐给用户的商品或信息往往是滞后的、过时的，是基于用户前几个小时或者前一天的行为得到的结果，往往无法满足用户当前的需求，因此这就迫使实际的推荐系统实时性有本质的提高。

### 3.2.2 功能需求和系统整体结构

 一、功能需求

（1）数据收集

      要完成用户实时推荐的计算任务，第一步，要对数据进行采集和清洗操作。对于实时推荐系统的计算需要两部分数据集，一部分是用户之前的离线的数据，第二部分是用户最新的实时数据。其中实时数据主要包括用户ID,用户最新的行为日志(如：用户对商品和评价操作，即用户-商品信息，包括用户ID，商品ID，商品分类，评分等级，时间戳字段等。

（2）数据计算

实时流计算是指对源源不断的数据流进行系统的连续计算，并实时的返回结果。为了确保计算的精确性和有效性，就要求系统提供实时的用户行为信息，然后通过相关计算，将结果写入hbase中，为后续推荐操作提供推送和查询。

二、性能需求

(1) 低要求，高效率，可扩展

对于该系统使用的大数据模型采用分布式框架，支持良好的横向扩展性，对于并发的提高只需要横向扩展机器即可。

(2) 高容错性，高可靠性

对于分布式数据集的容错方式一般来说有两种，数据检查点和记录数据的更新。Spark采用记录更新的方式。但是如果更新粒度太细太多，则记录更新成本也会相应的增加。因此spark提供的弹性分布式数据集RDD具有数据流模型的特点：自动容错，位置感知调度和良好的可伸缩性。与分布式共享内存系统需要付出高昂代价的检查点以及回滚机制不同，RDD通过Lineage来创建丢失的分区：RDD记录了单个块上执行的单个操作，每个RDD都记录了他变换的所有操作，以便恢复丢失的分区，这种容错机制有着较低的资源开销和一定的性能提升。

三、系统设计

        实时流计算作为加速层主要用于实时处理，缺点是不能进行复杂计算。

        实时计算层的特点主要是对数据进行流式处理，持续性计算，存储和分析某个窗口内的数据，该方式通常是基于实时性和准确性的一种折中，因此准确性有时并不高(临时解)，后期最终的结果，会通过批处理获取，批处理会覆盖掉流式处理的结果。具体架构图详见图3-1：



图3-1 实时计算系统架构图

其中对于新产生的数据会收集用于离线计算部分，因为实时计算不允许进行耗时的复杂计算。因此其中一小部分数据会选择性的提交给kafka消息总线，用于实时计算。对于数据收集部分会通过节点上部署的flume上进行，而且flume与kafka也有很好的兼容性，而对于flume的数据收集部分在之前离线计算章节已经有详细的介绍。

## 3.3 其他模块详细介绍

### 3.3.1 分布式日志框架

基于Spark平台的分布式日志框架如图3-4所示，该框架用于采集访问各个业务系统的日志信息，通过可靠的消息传送中间件作为模型训练与数据源采集之间的纽带，系统构建了基于Kafka集群的消息分发中间件，实现了日志数据的统一下发。由于日志数据中包含着各个业务系统的日志以及用户点击流的日志，在进入离线或实时推荐节点前，需要进行统一的数据清洗。本系统采用Spark Streaming技术按时间分片，对固定时间间隔内收到的数据进行统一的批处理，能达到实时处理的效果，并具有很高的吞吐量。



图3-4 基于Spark分布式日志框架

数据收集主要会经历如下阶段：日志数据采集，日志数据聚合和日志数据过滤。

1)分布式日志实时采集

实时推荐系统需要大量的隐式的用户行为作为基础数据,而且没种用户的行为的源日志信息分布在不同的业务系统中，需要构建分布式日志汇总系统将日志进行收集。以备后续流程使用。分布式日志采集如图3-5所示



图3-5分布式日志采集模块

2)基于Kafka集群的数据传输

通过构建分布式日志实时采集模块，完成用户行为日志的采集。但是在进入日志过滤阶段之前，由于日志流并发产生且数据量大，为了保证数据的实时性以及尽量减少数据丢失，使用分布式消息订阅和发布系统-Kafka,图3-6为基于Kafka集群的数据分发架构。



图3-6基于Kafka集群的数据传输

### 3.3.2 推荐引擎组

本系统在线计算部分与具体的推荐目标相关，需要在离线计算的基础上，以最快的速度给出推荐结果。在线计算部分计算量不是很大，数据量也不多，引入spark后，先将输入载入内存，来提高推荐速度。通过部分离线数据和实时的用户行为数据进行在线计算，得到的的结果就是推荐引擎的输出。在经过一系列处理后，就可以将结果存入Hbase或者返回给前台页面展示。

****

图3-7推荐系统整体架构

### 3.3.3 推荐系统整体架构

推荐系统整体架构主要包含批处理层、实时处理层，数据存储层、应用层。其中批处理层用户进行复杂的耗时的数据处理，实时处理层主要是从消息队列中选择部分数据通过流计算快速获取一个临时解，从而实现准实时的推荐效果。数据存储层用于读取批处理层和实时处理层通过一定的合并策略(择优选择，累加等) 将数据合并，支持随机读，可以在较短的时间内返回查询结果。应用层则是用于推荐结果的展示等一系列的业务处理。

该架构设计能够满足实时大数据系统，并具有高容错、低延时和可扩展性等特性。该架构整合了Hadoop、kafka、Spark、Hbase等大数据组件。详细架构图详见图3-8。



图3-8 推荐系统整体架构图

# 第四章 关于推荐算法的相关问题研究

电子商务在过去十年中的日益普及，以及互联网用户的迅速增长已经导致了推荐系统的日益普及，能够帮助客户找出最符合个人品味的产品变的越来越重要。推荐系统系统将数据分析技术应用于帮助用户在电子商务网站上找到他们想要购买的项目的问题，产生预期的可能性得分或着给定用户的N类推荐项目列表。近年来提出了许多推荐系统算法的实现，其中基于协同过滤的算法被发现是非常有效的，协同过滤算法的基本思想是根据其相似用户或产品通过相似度计算提出对项目的建议和预测。下面章节我们将重点介绍基于早期基于用户的协同过滤推荐算法和基于Item的协同过滤推荐算法的研究，以后对后者在算法上的改进以及相关测试，并且根据其不足提出了基于矩阵分解的协同过滤算法，以及该算法的优化。

## 4.1 基于用户的相似度计算

基于内存的算法是利用整个用户项目数据库来生成预测的。 这些系统使用统计技术来找到具有与目标用户相似的历史的一组用户（称为邻居）（即，它们可以类似地评估不同的项目，或者倾向于购买相似的项目集合）。 一旦形成了用户邻域，这些系统使用不同的算法来组合邻居的偏好以产生用于活跃用户的预测或顶层推荐。 这些技术，也称为最近邻或基于用户的协同过滤，在实践中更受欢迎并被广泛使用。

基于用户的协同过滤系统在过去的研究已经取得了成功，但它们的广泛应用逐渐暴露出很多缺陷和不足，如：

* 稀疏性。实际上，许多商业推荐系统被用于评估大型项目集（例如，淘宝（taobao.com）推荐的商品和网易云音乐推荐的歌曲或专辑。 在这些系统中，活跃的用户可能甚至只购买了不到0.01％的项目。 因此，基于最近邻算法的推荐系统可能不能为特定用户做出任何项目建议。 因此，推荐的准确性可能很差。
* 可扩展性。 最近邻算法需要随着用户数量和项目数量的增长而增长。 拥有数百万用户和项目，运行现有算法的典型的基于Web的推荐系统将遭受严重的可扩展性问题。

对于大型稀疏性数据库，最近邻算法的弱点使得我们探索其他推荐系统算法。我们的第一种方法试图通过将半智能过滤代理纳入系统来弥合稀疏性问题。这些代理使用语义特征评估每个项目。通过提供一个密集的评级集合，它们有助于减轻覆盖面并提高质量。然而，过滤代理解决方案没有解决稀疏评级矩阵与用户之间关系差的根本问题。在本文中，我们研究了另一种技术，其主要思想是分析用户-项目矩阵以识别不同项目之间的关系，然后使用这些关系来计算给定的某对用户的预测分数。主要思想是，用户有兴趣购买与用户之前喜欢的项目相似的项目，并且倾向于避免/不予以推荐类似于用户之前不喜欢的项目的项目。当请求建议时，这些技术不需要识别类似用户的邻域;因此，他们倾向于推出更快的建议。已经提出了许多不同的方案来计算从概率方法到更传统的项目-项目相关性之间的关联。我们将在下一节中详细分析我们的方法。

## 4.2 基于Item的协同过滤算法

在本节中，我们研究了一类基于项目的推荐算法，用于向用户生成推荐。 与4.1节中讨论的基于用户的协同过滤算法不同，基于项目的方法基于目标用户已评价的项目集合，并计算它们与目标项目i的相似度，然后选择k个最相似的项目{i1，i2，…..，ik}。 同时他们相应的相似度为{si1，si2，…. sik}。 一旦发现最相似的项目，则通过对这些相似项目采用目标用户的评级的加权平均来计算预测。 我们在这里详细描述这两个方面，即相似度计算和预测生成。

### 4.2.1 项目相似度计算

基于项目的协同过滤算法的一个关键步骤是计算项目之间的相似度，然后选择最相似的项目。 两个项目i和j之间的相似度计算的基本思想是首先隔离对这两个项目进行评级的用户，然后应用相似度计算技术来确定相似度si,j。 图4-1说明了这个过程; 这里矩阵行表示用户，列表示项目。



图 4-1 共评项目的隔离与相似度计算

计算项目之间的相似性有许多不同的方法。 在这里我们提出三种这样的方法。 这些是基于余弦的相似性计算，基于相关的相似性和调整余弦相似度。

1. cosine相似度计算

在这种情况下，两个项目被认为是m维用户空间中的两个向量。 它们之间的相似性通过计算这两个向量之间的角度的余弦来测量。形式上，在图4-1中的个等级矩阵中，项目i和j之间的相似度表示为sim(i,j)，sim(i,j)由以下公式给出

(公式4.1)

1. 皮尔逊相似度

在这种情况下，通过计算Pearson-r相关系数corrij来测量两个项目i和j之间的相似度。 为了使相关性计算准确，我们必须首先隔离被同一用户评分的item（即用户同时对i和j进行评分）。将评价i和j的用户集合用U表示相关相似性由下式给出：

(公式4.2)

这里Ru,i表示用户u对项目i的评分，R̅i是第i项的评分的平均。

1. 调整余弦相似度

基于用户的协同过滤算饭和基于项目的协同过滤算法之间的相似度计算之间的一个根本区别在于，在基于用户的CF的情况下，沿着矩阵的行计算相似度，而基于项目的CF的情况下，是沿着矩阵列计算相似度，即共同设定中的每对对应于不同的用户（图4-1）。 在基于项目的案例中使用基本余弦测量的计算相似度有一个重要缺点，不考虑不同用户之间的评级量表的差异。 调整的余弦相似性通过从每个共同对中减去相应的用户平均值来尽可能抵消这个不足。 一般来说，使用该item i和j之间的相似性由下式给出:

(公式4.3)

这里R̅u是第u项用户评级的平均值。

### 4.2.2 预测计算

协同过滤算法中最重要的一步是根据预测生成输出接口。 一旦我们基于相似性度量来找到最相似项目的集合，下一步是研究用户评级，并使用一种技术来获得预测。这里我们考虑两种这样的技术。

1. 加权求和

顾名思义，该方法通过计算用户对类似于i的项目给出的评分的总和来计算用户u的项目i的预测。 每个评级由项目i和j之间的对应相似度si,j加权求和。 使用图4-2所示的概念，我们可以表示预测Pu,i 的公式为

(公式4.4)

基本上，这种方法尝试了解活跃用户如何评价相似项目。通过相似项的加权和，以确保预测在预计范围内。



图4-2 Item-based 协同过滤算法预测产生过程

1. 回归

这种方法类似于加权求和的方式，使用基于回归模型的评级得到相似度，而不是直接通过相似项目的评分的方式获得。 实际上，使用余弦或相关性度量计算出的相似度可能会产生误差，因为两个评估矢量可能是相差甚远的（欧几里得），但可能具有非常高的相似度。 在这种情况下，使用所谓的“相似项目”的原始评级可能导致较差的预测。因此基本思想是使用与加权求和技术相同的公式，而不是使用相似项目N的“原始”评分Ru, N的值，该模型基于线性回归模型使用其近似值。 如果我们用Ri和RN分别表示目标项目i和相似项目N的各自 的特征向量，则线性回归模型可以表示为

(公式4.5)

回归模型的参数α和β的值是通过两个评估向量来确定。 是回归模型的误差。

1. 性能影响及意义

在基于邻域的协同过滤系统中，邻域形成过程尤其是用户相似度计算步骤成为性能瓶颈，这可能会使得整个推荐系统无法完成实时推荐。 确保高可扩展性的一种方法是使用基于模型的方法。 基于模型的系统有可能有助于推荐系统高效运行。

本文中，我们提出了一种基于模型的预测计算项目相似度分数的方法。 相似度计算的约束仍然是基于相关的，但是在项目空间上执行计算。 在典型的电子商务方案中，我们通常会有有一组item，它与用户数量相比而言是变化不频繁的。 项目的静态性质使我们产生预先计算项目相似性的想法。 预先计算项目相似度的一种可能的方式是计算所有相似性，然后执行快速表查找以检索所需的相似度值。 这种方法虽然节省时间，却需要一个O(n2)空间用于n个项目。

事实上，我们只需要一小部分相似的项目来计算预测，这就导致了一个替代的基于模型的方案。 在这个方案中，我们只保留少量的类似物品。 对于每个项目j，我们计算k个最相似的项目，其中k<<n并记录这些项目编号及其与j的相似度。 我们将k作为模型大小。 基于此建模步骤，我们的预测生成算法工作如下。 为了为项目i上的用户u生成预测，我们的算法首先检索与目标项目i相对应的预先计算的k个最相似的项目。 然后统计k个项目中多少是由用户u购买的，然后使用item-based的协同过滤算法来计算预测。

我们在这里观察质量表现的折衷点：为了确保质量，我们必须具有较大的模型尺寸，从而导致上述性能问题。 在一个极端，我们可以有一个模型大小为n，这将确保与原始方案完全相同的质量，但会具有高的空间复杂度。 然而，我们的模型构建步骤确保我们保留最相似的项目。 在产生预处理之前，这些项目对预测分数贡献最大。 因此，我们假设这种基于模型的方法将提供相当好的预测质量，甚至具有小的模型尺寸，因此提供了良好的性能。 我们在本文后面实验验证了我们的假设。 特别地，我们通过改变要存储的类似项目的数量来实验模型大小。 然后，我们进行实验来计算预测和响应时间，以确定模型大小对整个系统的质量和性能的影响。

### 4.2.3 实验与测试

## 4.3 基于矩阵分解的协同过滤异步分布随机梯度算法

### 4.3.1 算法概述及引入原因

该方法的主要思想是将用户评分矩阵分发在不同的机器上，每个机器只能访问一部分信息，并且通过网络异步的传播随机梯度优化的方式进行更新。每次机器接收到一个参数向量时，它会将当前参数的特征向量与所接收特征向量进行平均，并从该点继续进行迭代。此外，我们还引入了基于正则化的相似度计算，约束用户和项目因子接近其相似用户的平均因素，以及在分布式用户评价矩阵的子部分中找到的项。我们分析正则化对MovieLens（100K，1M，10M）和NetFlix数据集的影响，并且显示它使得计算结果的均方根误差（RMSE）和平均绝对误差（MAE）有明显的改善， 并且与等效同步分布式方法相比，异步分布式方法在收敛时间方面显着提高。

上一章节我们提到基于item的协同过滤推荐算法，其推荐结果是较为精准和有效的，其基本思想是，如果两个用户对一组项目具有相似的喜好程度，则它们可能在新项目上也具有相似的喜好程度。基于这个想法开发的的协同过滤技术有: (1) 邻近方法，使用相似用户或项目的过去的评分生成对用户未知物品的预测; (2) 矩阵分解技术，其目标就是把用户-项目评分矩阵分解成用户因子矩阵和项目因子矩阵乘的形式。

在本章节中，我们将分解后的矩阵分发到集群的各个服务器中，并提出了一种新颖的矩阵分解异构框架，使用SGO(随机梯度算法)的方式进行推荐。在现实场景中，由于网络中连接的机器通常加载不同，同步更新可能变得非常慢，因为每台机器的每个阶段必须等待最慢的机器计算完毕后才能进行下一步的计算。此外，为了增强上述方案，我们将经典RMSE目标的两个相似性的正则化项加入到约束用户和项目因素使其在接近于他们最相似的用户和子部分中找到存储在机器中的评级矩阵。在我们的异步框架中，每个机器并行执行SGO来计算在评分矩阵的不同部分，并将其计算的权重发送到网络中的其他机器。一旦机器收到一个更新的权重，它会使用自己的更新对参数权重进行平均，并继续进行下一阶段的SGO运算。 这样可以保证每台机器能够全面了解整个数据的计算情况。

我们仍然使用Movielens和Netflix数据集进行测试，我们表明，提出的用户和项目正则化项的添加导致在RMSE和MAE方面更有效的矩阵分解。 与同等的同步分布式方法相比，实验结果表明异步分布式更新方案具有更好的的收敛性。

### 4.3.2 用户矩阵分解和基于Item的矩阵正则化

在本章节中我们将介绍矩阵分解中的基本定义和目标。用户和项目的数据集分别由集合U和I表示，其中ui和ij分别是第i个用户和第j个项目。此外，我们用rij表示用户ui的项目ij的评分，相应的评分矩阵R为rij的矩阵。用户（相应的项目矩阵为），pi（和qj）是对应于第i个用户（第j项项目）的特征向量。

随机梯度优化（SGO）的矩阵分解，其前提是用低维度因子矩阵PQT的乘积来代替高维矩阵R.这些因子矩阵试图对k个隐含因子的项目进行用户偏好建模。 对于存在评级rij的一对用户和项目（ui，ij），该方法是基于最小化的二次正则化二次误差：

(公式4.7)

其中λ≥0是正则化参数。 因此，整个矩阵分解问题写入

(公式4.8)

注意，误差仅依赖于P和Q通过pi和qj; 然而，项目ij也可以由用户ui来评估，使得最佳因子qj取决于pi和pi'。 因此，随机梯度优化（SGO）方法如下进行：在每次迭代t中，i选择存在等级的随机用户/项目对（uit，ijt）; ii）对执行梯度步长。

为了加强类似用户具有相似度的假设，我们强调每个用户（相应项目）的因子向量应该接近于其类似用户（相应项目）的平均因子向量。 为了计算最相似的用户（或项目），我们考虑了两个用户ui和uj写的Pearson相关系数的修改版本：

(公式4.9)

这里，lc是由两个用户共同评估的项目，ri和rj分别表示ui和uj的平均等级。因此，我们可以找到N个最相似的用户ui，用Ni表示（Nj对应类似于ij的项目）。我们现在提出稍微修改方程式的单个评级目标函数l。 （1）加上另一个正规化术语。 对于存在评级rij的一对用户和项目（ui，ij），相似性规则化的单个目标写成：

(公式4.10)

其中≥0和≥0分别是与相似用户和相似item相关联的正则化参数。执行与传统SGO相同的更新，但将l替换为l1，我们得到算法l。

以最小化整个矩阵分解问题（方程4.10），其中l被l1替代，即相似性规则化问题：

### 4.3.3 异步分布式随机梯度优化

即使随机梯度优化在推荐系统数据集上能够达到较高的推荐精度，但它也存在与之相关的一些计算挑战。 在单个机器上顺序执行SGO需要较长的收敛时间。 因此，需要以分布式方式对大型数据集进行随机梯度优化。 对于并行化SGO而言，直接实现的一个缺点是因子矩阵的更新可能不是独立的。 例如，对于位于相同行（或列）的训练点，它们同时更新P和Q矩阵对应行（或列）。 因此，对于同步更新因子矩阵而言计算节点之间的有效通信是非常必要的。

一种常见的方法是将评分矩阵划分为几个块，并将分割后的矩阵分发到多个服务器上进行SGO计算。在每个机器会根据对应的评级来更新因子矩阵。虽然每个机器上的评级矩阵都各不相同，但是因子矩阵之间的更新是同步的。因此，在每个阶段，每个机器中计算得到的因子矩阵是同步的。我们将该方法称为“同步SGO”，由于所有机器会每个计算阶段完成之后同步它们的因子矩阵，因此是较为低效的，主要是因为同步SGO的每个服务器的各个阶段的计算时间是随其机器的配置不同而不同。因此，在每个计算阶段之后，这种机器之间的同步更新操作会变得滞后，因为更快的机器必须等待较慢的机器计算完成才能进行同步操作。

根据上述的不足之处在此提出了一种基于参数矩阵的异步传播的分布式框架。每个机器被分配了相似用户评分矩阵的一部分，我们可以通过大规模图形聚类技术离线找到对应的分割矩阵。 每台机器仅通过其本身的评分矩阵来更新因子矩阵，与同步SGO方法相反，只要其中一台机器完成一个时期的计算，就会广播给集群中所有的机器更新其相应的因子矩阵。因此，每当一个新的时期开始，它都会收集其他机器中最新的因子矩阵，并且在完成一个时期计算之后，将其更新的因子矩阵广播到所有其他机器，并立即开始下一个时期，而不需要考虑其他机器的计算状态。 这样，所有的机器都是相互独立运行的。 因此，更快的机器将更快地执行，而较慢的机器将在时间上落后，但在完成每个时期之后，它们将从更快的机器接收最新的因子矩阵。



图4-4异步分布式SGO框图

基于MovieLens和NetFlix数据集，我们对不同的方法计算其MAE和RMSE。详见下表：

表4-1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Dataset | | SGO | | 基于正则化的SGO | | | |
| MAE | RMSE | Similar Users and Similar Item | | Similar Users | |
| MAE | RMSE | MAE | RMSE |
| ML-100LK | ua | 0.7380 | 0.9469 | 0.7388 | 0.9328 | 0.7396 | 0.9355 |
| ub | 0.7592 | 0.9580 | 0.7555 | 0.9557 | 0.7539 | 0.9580 |
| ML-1M | ra | 0.7314 | 0.9699 | 0.7207 | 0.9514 | 0.7166 | 0.9476 |
| rb | 0.6882 | 0.8859 | 0.6894 | 0.8780 | 0.6510 | 0.8398 |
| ML-10M | rb | 0.6491 | 0.8409 | 0.6479 | 0.8382 | 0.6510 | 0.8386 |
| NF-Subset | NA | 0.6500 | 0.8277 | 0.6459 | 0.8257 | 0.6485 | 0.8265 |

在我们的设置中，我们将等级矩阵R划分到m行到新的矩阵中，其中m是可用的机器（或进程）的数量。 P矩阵也以相同的方式被阻塞。每一个计算节点都有一个R矩阵，对应的P矩阵和基于R矩阵的本地Q矩阵。在每个计算时期，我们基于R矩阵和各个误差函数来更新P和本地Q矩阵的块。整个过程如图4-4所示，整个网络由三个计算节点组成，R1，R2和R3表示每台机器上的评分矩阵。 ep代表了每个时期的结束和不同机器上新时期的开始。如图所示，在完成一个时期之后，每个机器将其更新的Q矩阵广播到给集群中的其他机器，而在新时期开始时，每个机器将通过接受到的其他机器更新的Q矩阵来更新自身的Q矩阵。通过这种方式，计算较快的机器产生的更新矩阵可以帮助较慢的机器更快地收敛，从而改善整个分布式网络的收敛时间。

## 4.4 关于算法冷启动问题的研究

### 4.4.1 冷启动问题概述

推荐系统的冷启动问题主要包括新项目冷启动和新用户冷启动两个方面。而对于新用户的冷启动问题一般可以通过利用用户在其他地方已经沉淀的数据和用户兴趣偏好进行冷启动或者制造选项生成粗粒度的推荐等方式进行。这里着重解决新项目的冷启动问题，在此我们引入了用户时间权重的概念，通过对用户评价的时间间隔判断用户活跃度并根据用户对新项目偏爱程度等因素创建构建个性化的预测评分公式。由于新项目不存在任何一个用户对它的评价信息，故无法使用协同过滤算法来实现推荐，因此，本章节提出解决该类冷启动问题推荐算法，该算法不仅仅使用用户对物品的评分信息对来进行推荐，而是要综合考虑项目本身的诸多属性信息、标签对项目的标注信息等较为全面的考虑影响推荐系统的诸多因素。实验证明该方式，在解决推荐系统在新项目的冷启动的问题上是较为有效的。

### 4.4.2 解决方案

现实生活中，有些用户喜欢去关注并评价新事物，我们定义这类用户为活跃用户；而有的用户可能比较喜欢去关注已经被很多用户评价过的事物，我们定义这类用户为消极用户。对于积极用户来说，他们评价项目的时间与项目发布的时间间隔一般都很短；而消极用户则恰恰相反。在推荐系统应用的过程中，这两类用户是普遍存在的，因此在推荐的过程中我们往往会优先将新项目推荐给积极用户。根据这种思想将时间信息加入到推荐算法中，以此来解决冷启动问题。

为了描述时间信息对推荐算法的影响，首先定义用户时间权重如下：用户项目总数与用户所有评价项目的时间与项目发布时间的差值的总和。

其中sum表示用户评价过的项目总数；timeui 表示用户评价的第i个项目的时间，datai 表示项目i发布的时间。则用户u的时间权重Wu 计算公式如下：

(公式4-11)

由此可知，timeui - datai 越小，说明项目评价时间与项目发布时间的距离越近，用户越积极。反之亦然。

与实际项目为例，用户评价时间表如下：

表4-2 : 用户评价时间表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Item | U1 | | U2 | | U3 | |
| Time | Date | Time | Date | Time | Date |
| i1 | 2016 | 2015 | 2015 | 2014 | 2017 | 2017 |
| i2 | 2016 | 2014 | 2016 | 2013 | 2017 | 2014 |
| i3 | 2015 | 2012 | 2017 | 2014 | 2017 | 2016 |

以表一为例，采用公式1计算得到每个用户的时间权重wui 即：

根据如上计算可知，用户积极程度由大到小依次是Wu1>Wu3>Wu2 ,即用户Wu1更偏爱于新发布的产品。

此外我们还根据项目标签信息预测评分值。

1. 基于项目评分信息预测评分值

这里，我们假定以用户u为研究对象，分析用户-项目评分矩阵的信息，可以得到用户u评价项目的个数和项目j被用户评价的次数。令ruj 表示用户u是否评价项目j,1表示评价过，0表示没有评价过；ICk 表示用户k评价的项目个数，UCj 表示共同评价项目j的用户个数。那么根据用户对项目的评分信息以及用户评价项目个数以及项目被用户评价的次数信息，可以定义基于项目评分信息的预测评分值，从而挖掘他们之间的关系。具体公式如下:

(公式4-12)

2）基于项目属性信息预测评分值

如公式所示表示基于用户u对项目j的预测评分值。如果用户u 对项目 j 做了评价ruj =1, 反之为0。如果项目j具有属性l，则hlj=1，反之为0。

(公式4-13)

3）基于项目标签信息预测评分值

基于项目标签信息的预测评分计算公式如下：

(公式4-14)

公式(3-4) 中， 表示基于标签的用户u对项目j的预测评分值。如果标签g标注的项目j，则bgj=1分之为0。

下面我们基于具体的数据矩阵和公式来围绕研究对象u1计算预测评分值。

表4-3 用户项目评分矩阵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| rij | i1 | i2 | i3 | i4 |
| u1 | 2 | 0 | 4 | 0 |
| u2 | 0 | 3 | 1 | 0 |
| u3 | 5 | 0 | 2 | 0 |

表4-4 属性-项目矩阵

表4-5 标签-项目标注矩阵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bij | i1 | i2 | i3 | i4 |
| t1 | 2 | 3 | 0 | 0 |
| t2 | 0 | 3 | 1 | 0 |
| t3 | 1 | 0 | 0 | 0 |
| t4 | 0 | 2 | 1 | 0 |
| t5 | 0 | 0 | 2 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| hij | i1 | i2 | i3 | i4 |
| a1 | 1 | 0 | 1 | 0 |
| a2 | 1 | 1 | 1 | 0 |
| a3 | 1 | 0 | 1 | 1 |
| a4 | 0 | 1 | 0 | 1 |
| a5 | 0 | 1 | 0 | 1 |

其中{i1,i2,…in}为n个项目的集合，{u1,u2…um}表示m个用户，rij表示用户ui对项目j的评分。{a1,a2…,an}表示p个属性的集合，hij表示项目j是否具有an属性。{t1,t2…,tq}表示q个标签的集合,bij表示项目j被标注的次数。为了更好的理解公式与表之间的关系，使用两个三分图的形式来表示。



图一 用户-项目-标签信息 图二 用户-项目-属性信息

因此我们可以根据上述理论基础，分别根据用户-项目-标签信息和用户-项目属性信息来挖掘他们之间的潜在关系。从而通过预测评分值来解决新项目的冷启动问题。

# 第五章 实验与测试

## 5.1 基于Item的协同过滤算法测试

### 5.1.1 实验环境

实验采用Linux（Centos6.4）操作系统，2.9 GHz Intel Core i5，需要安装的软件有：JDK1.7 、Hadoop2.2.0、Spark1.2.0、Scala2.10.4等。编译工具为IntelliJ IDEA, 采用Scala语言编写算法和相关代码。

### 5.1.2 数据集

我们使用我们的MovieLens推荐系统的数据。 MovieLens是一个基于网络的研究推荐系统，在1997年秋季首次亮相。每周hundreds的用户访问MovieLens评分和提供电影建议。该网站如今已有超过83000名用户对4800多种不同的电影发表了意见。我们随机选择了足够的用户以及他们对电影的100000个评分（我们只考虑了20个已对电影评分用户和相当多的电影）。我们把数据库分成一个训练集和一个测试集。为此，我们引入了一个变量来确定用作训练和测试集的数据百分比;我们称这个变量x。 x = 0.8将表示80％的数据用作训练集，20％的数据用作测试集。将数据集转换为具有943行（即943个用户）和1682列（即由至少一个用户评级的1682个电影）的用户项目矩阵A。对于我们的实验，我们还考虑了另一个因素，数据集的稀疏程度。对于数据矩阵R这被定义为 。因此，电影数据集的稀疏性程度是 = 0.9369。在整个论文中，我们称这个数据集为ML。

### 5.1.3 评价指标

推荐系统研究使用了几种类型的评估推荐系统质量的措施。 它们可以主要分为两类：

* 统计精度指标：通过将预测推荐分数与测试数据集中实际用户对项目评级进行比较来评估系统的准确性。 实际评分和预测之间的平均绝对误差（Mean Absolute Error，MAE）是广泛使用的指标。 MAE是衡量推荐结果偏离其真实值的度量。 对于每个评级 - 预测对 ˂pi，qi˃这个度量对待它们之间的绝对误差，即|pi –qi| 平等。 通过首先对N个相应的等级 - 预测对的绝对误差求和然后计算平均值来计算MAE。从形式上看，

(公式4.6)

MAE越低，推荐引擎预测用户评级越准确。 此外均方根误差（Root Mean Squared Error ，RMSE）也可用作统计精度度量。

* 决策支持精度指标：评估推荐引擎在帮助用户从所有项目集合中选择高质量项目的有效性。无论项目是被推荐（好）还是不被推荐（坏）, 这些指标假设预测过程都是二进制的。通过观察，如果用户仅选择考虑4或更高的预测，那么项目所有预测评分低于4的项目均变得无关紧要的。 最常用的决策支持精度指标是反转率，加权误差和ROC敏感度。

因为MAE指标是最常用和最简单的直接解释，因此我们使用它作为评估指标的选择来作为评价试验好坏的指标。

### 5.1.4实验过程

首先我们将实验数据集划分为训练和测试部分。 在开始对不同算法进行全面实验评估之前，我们确定了不同参数对不同算法的灵敏度，并从灵敏度图中确定了这些参数的最优值，并将其用于其它实验。 为了确定参数灵敏度，我们仅与训练数据一起工作，并进一步细分为训练和测试部分，并对其进行实验。 通过每次随机选择不同的训练和测试集合并获取MAE值的平均值，对我们的实验进行了10次交叉验证。

### 5.1.5 实验结果

在本节中，通过使用基于项目的协同过滤算法生成推荐的实验结果数据，主要分为两部分质量结果和性能结果。 在评估推荐质量时，我们首先在运行主要实验之前确定了一些参数的敏感度。 这些参数包括邻域大小，不同相似度的影响。 为了确定各种参数的灵敏度，我们仅关注训练数据集，并进一步将其分为训练和测试部分，并用它们来学习参数。

* 相似度算法的影响

我们实现了三个不同的相似度算法，如4.2.1节所述的余弦相关性，调整余弦和相关性，并用指定的数据集进行测试。 对于每种相似度的计算方式，我们都通过算法计算其邻域和加权和来生成预测。 我们对我们的训练数据进行了推荐预测实验，并使用测试集来验证并计算平均绝对误差（MAE）来评判算法质量。 图4-3为实验结果。 从结果可以看出，设置调整余弦相似度计算的用户平均绝对误差值具有明显的优势。 因此，我们认为使用调整余弦的方式计算相似度最为准确。

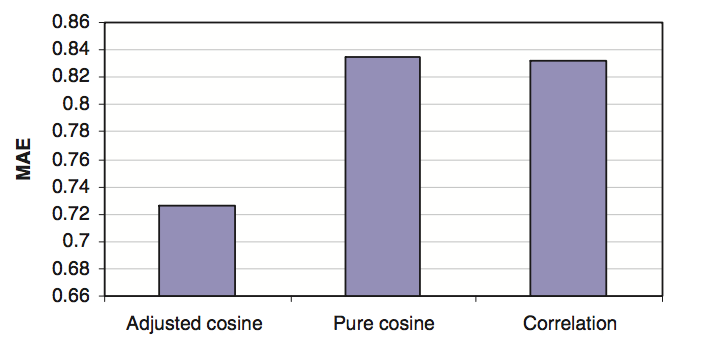


图4-3相似度计算对项目协同过滤算法的影响

* 邻域大小对实验结果的影响

邻域的大小对预测的质量有很大影响。 为了确定这个参数的对推荐预测结果的影响，我们做了进一步的实验，我们改变了邻居数并计算MAE的值。 实验结果如图4-4所示。我们可以观察到邻域的大小确实影响预测的质量。 但是这两种方法显示出不同的曲线变化。 基于Item-Item的算法的MAE随着邻域数的增加而增加，之后增加率逐渐减小，曲线趋于变大。 另一方面，基于回归的算法表明，邻域数较少时(邻居数量少于30) 随着邻居数量的增加，预测质量有非常明显的改善，之后随着邻居数量增加，推荐质量逐渐下降。 考虑到这两个趋势，我们选择30作为邻域大小的最佳选择。

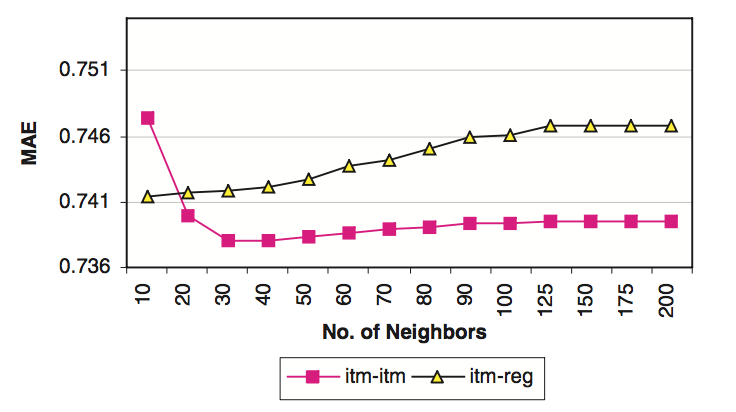
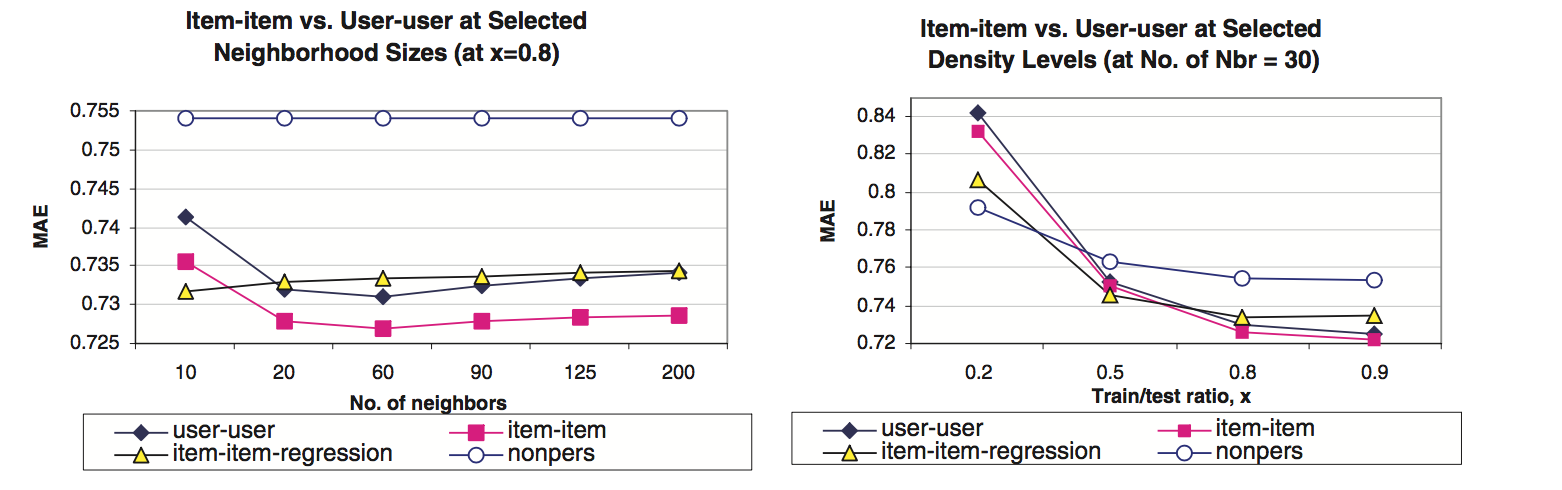
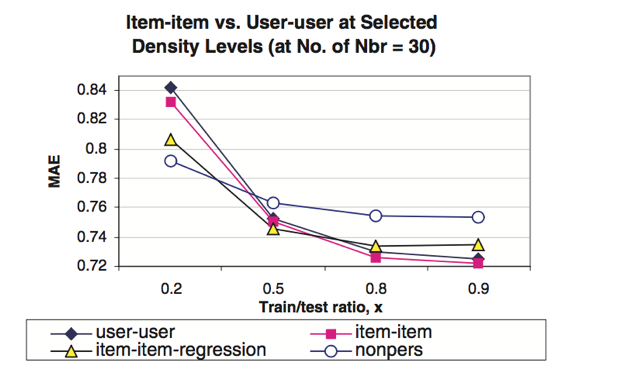


图4-4 邻域对MAE的影响

* 算法对比

一旦我们获得了参数的最优值，我们将基于项目的方法与基于用户算法进行比较。从图4-5中可以看出，基于item-item算法在x（邻域大小= 30）的所有值和邻域大小的所有值（x = 0.8）。例如，在x = 0.5时，基于用户的推荐算法的MAE为0.755，而基于Item的协同过滤算法的MAE为0.749。类似地，邻域数为60时，基于Item和基于User的MAE为别为0.726和0.732。然而，基于回归的算法显示出有意思的结果。当x值和邻域数较小的情况下，执行其他两种协同过滤算法，随着数据集的密度x增加，或者随着我们添加更多的邻居，它们所给出的推荐质量越来越差，甚至与user-based的算法的推荐效果相类似。

因此我们可以从这些结果中得出两个结论。 首先，基于Item的算法在所有稀疏级别上提供比基于用户的算法更好的质量。 第二，基于回归的算法使用非常稀疏的数据集执行得更好，但随着我们添加更多数据，推荐质量下降。 我们认为这是因为回归模型当受到高密度的数据时会出现过拟合的现象。

1. 实验结论

推荐系统是一种强大的新技术，用于从其用户数据库中提取潜在的有价值的数据。 这些数据可帮助用户在海量数据中找到他们需要的内容。 推荐系统通过让用户找到他们喜欢的项目来满足用户的需求，同时帮助企业增加相应的销售额。 推荐系统正在迅速成为网络电子商务的重要工具。推荐系统受到现有公司数据库中庞大的用户数据量的压力，尤其是受到网络上用户数据量的增加的压力。本章节中提出的协同过滤推荐算法虽然在不考虑负载消耗的前提下能够提供较高质量的推荐，但对于目前对与实时性要求更加苛刻的推荐场景下，需要新的可以显著提高推荐系统的可扩展性的技术。

## 5.2 基于异步SGO算法测试

  我们进行了一系列的实验，以评估相似用户和商品的正则化项对矩阵分解的影响，以及所提出的异步框架如何帮助加速SGO算法的收敛性。

    数据集：我们对MovieLens数据集（ML-100K，ML-1M和ML-10M）和NetFlix集合的一个子集（参见表1）进行了实验。 对于ML-100K和ML-1M，我们使用了两组（ra和rb或ua和ub），而对于ML-10M和Netflix，我们使用了单个数据集。

    我们使用PySpark 1.5.1实现了分布式框架。 为了演示不同机器之间的通信，我们部署了五台具有不同计算负载的服务器。其中一台机器为其他机器分配评级矩阵。当验证集上两次验证之间的RMSE差值低于我们在试验中设置的预定义阈值ε，将停止所以机器上的运算。本文提出的异步方法收敛时是当最慢的机器最慢的机器运算结束后，而对于同步的SGO来说则是在所有机器同时停止计算后完成。

表5-1：实验数据集特征

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DataSet | |U| | | I | | K | training size | test size | sparsity |
| ML-100K | 939 | 1679 | 20 | 90570 | 9430 | 93.7% |
| ML-1M | 6039 | 3952 | 40 | 939809 | 60439 | 95.8% |
| ML-10M | 71567 | 10681 | 100 | 9301274 | 688380 | 98.7% |
| NF-Subset | 28978 | 1821 | 40 | 3235492 | 101078 | 93.7% |

其中, | U | 和| I | 分别表示用户和项目的数量。

以下，数据集ML-10M的潜在因素数量K固定为100，而对于其他较小的数据集，实验选择最佳结果/速度为40,40和20。学习率和正则化参数λ 固定为0.005和0.05。

     对于之前提出的基于相似性的正则化参数λu，λi，和相似的用户/物品的数量N，其中 λu, λi 从集合{10−1, 5.10−2 , 10−2 , 5.10−3 , 10−3 , 5.10−4 , 10−4} 中选择，N从集合{10, 20, 30, 40, 50}中选择，恰当的选择λu, λi以及N的值会使我们t通过验证集得到较好的RMSE的值。

    评估措施：在我们的实验中，我们使用平均绝对误差（MAE）和均方根误差（RMSE）作为性能测量。 此外，我们比较了同步和异步计算中每个算法的收敛时间。

1. 基于相似的正则化效果

    首先，我们比较了基于传统SGO方法和我们提出的基于相似性正则化修正的SGO方法的结果。其区别仅仅是最小化的目标函数。

我们进行了基于分布式的同步SGO和异步SGO方式对不同的数据集(MovieLens/Netflix)进行基于相似用户和基于项目的正则化矩阵分解，得到的实验结果见图5-1。

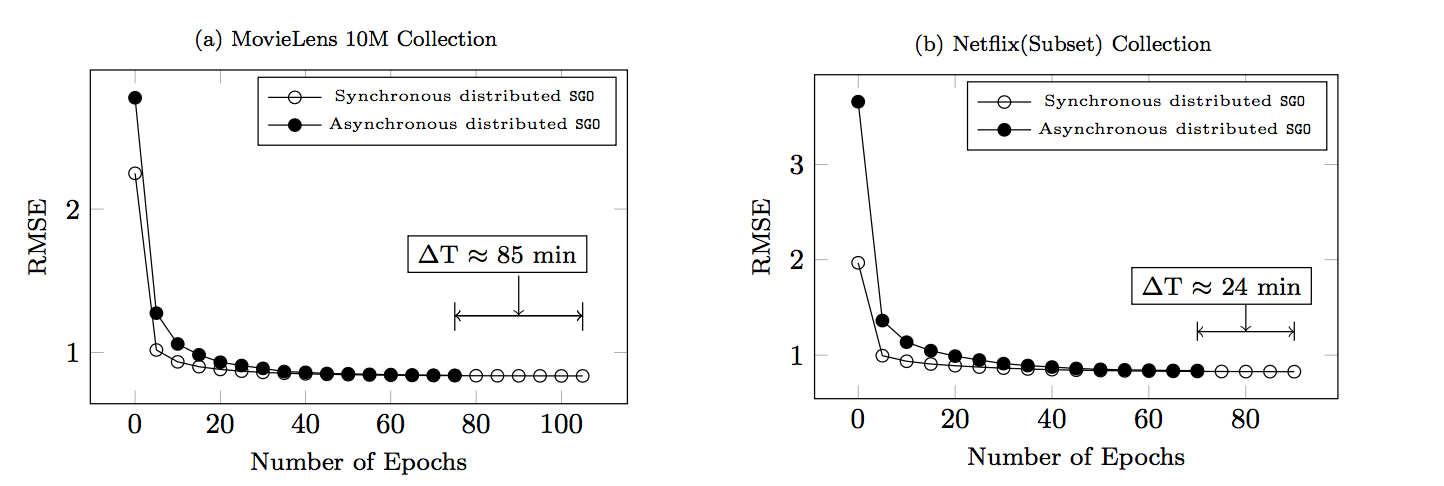


图5-1

我们测试了两种情况,（1）只有用户具有相似性规则(即项目的正则化参数λi = 0); （2）当用户和项目都使用相同的正规化参数(λu =λi)。图5-1显示了我们实验的完整结果。

     实验结果表明用户和物品的特征向量位于他们最相似的用户的中心和Pearson相似性度量中找到的物品是有效的，因为最终的RMSE和MAE与算法1总是比传统的SGO更好。因此，在机器学习方面使用这种正则化方法是有显著效果的。 我们还通过实验验证了相似用户(不是item)的正则化带来的影响（λu> 0，λi= 0）。如(表4-1)所示，对于某一数据集而言，仅用户的正则化(λu> 0，λi= 0)比传统SGO的效果更好更稳定，甚至比用户和项目的正规化效果更好。

1. 收敛时间的评估

现在，我们将通过与同步分布式SGO方法进行比较来评估我们提出的异步分布式SGO的效果。为了展示其可扩展性，我们对用户-项目相似性正则化问题的收敛速度进行评估。如图5-1所示，对于一定的epochs, 不同的数据集Movielens和Netflix,得出了对于我们提出的算法和同步的SGO方法的RMSE曲线对比图。

对于异步分布式方法，

就异步方式而言，我们通过实验展示了最慢机器的收敛时间。正如预期的那样，由于最慢的机器会从较快的机器上获取更新数据从而使得整体以较少的阶段趋于收敛。而对于传统的同步计算方式，运行速度较快的机器则不得不等待每个阶段运行较慢的机器的计算结果。

1. 结论

    在这个工作中，我们提出了一个异步分布式框架SGO。该框架是通用框架并且已应用于推荐系统的矩阵分解。此外，我们还为这项任务提出了两个额外的正则化项。 实验结果展示了对于不同的数据集MovieLens和Netflix使用不同方法的收敛速度。

# 第六章 总结与展望

## 6.1 总结

    目前，大数据背景下，互联网中海量信息中蕴含了很多有价值的信息，如何挖掘这些信息为人所用变得越来越重要，同时也是大数据平台首要需要解决的问题，其中推荐系统的出现，其本质正式通过挖掘用户的行为信息，更好的了解用户的喜好，从而帮助用户了解自己，并帮助用户从海量数据中找到自己需要的内容。如今几乎所有的大型电子商务系统都不同程度的使用了推荐系统。推荐系统能够有效的提高电子商务系统的服务能力，具有良好的发展和应用前景。本文以设计并实现一个大数据环境下的推荐系统为主线，介绍了大数据的相关技术以及推荐系统的相关概念和面临的问题。并基于所研究的算法在数据集MovieLens和Netflix下进行了相关指标的测评。

    首先我们介绍了推荐系统所解决的问题，即大数据背景下的信息过载问题。然后我们详细介绍了Spark生态系统的相关框架以及所使用到的大数据处理的相关模块。使用Spark来代替MapReduce，使用Scala编程语言，从而实现更高的编程以及算法的执行效率。

    在设计和实现部分，介绍了系统的整体架构，包括系统各层次的作用以及相关的功能，详细介绍了分布式日志框架的实现等。接着我们介绍了论文的主要研究内容，即推荐算法的相关原理及性能。并对推荐系统的冷启动问题做了相关介绍并提出解决方案。具体工作如下：

    (1) 基于Spark分布式框架设计推荐系统架构，通过spark实现算法的并行计算，针对系统实时性问题，设计能够实现实时计算的流处理架构，将离线计算与实时计算相结合，根据用户在线实时行为和其历史行为数据实时计算，得出推荐结果。

    (2) 协同过滤算法的相关研究。重点研究了基于Item的协同过滤算法，测评算法的推荐准确性，以及影响该算法准确率的因素。并引入了基于矩阵分解的协同过滤的异步分布随机梯度算法缩短算法收敛时间。

    (3) 分析研究推荐系统冷启动问题。引入了用户时间权重的概念，根据用户-项目-标签信息和用户-项目属性信息来挖掘他们之间的关系。从而通过预测评分值来解决新项目的冷启动问题。

## 6.2 展望

    本文主要关于Spark分布式框架的推荐系统的相关技术研究，着重的介绍了推荐系统的架构设计，以及研究了相关推荐算法，并测试分析其性能。不过该系统仍存在很多不足和需要改进的方面。

    1、整个系统如何在数据快速增长的情况下，有效的调配计算资源，进行有效的推荐。

    2、系统设计本身没有考虑到安全因素，例如商家恶意操作影响推荐结果以谋取利益的情况。

    3、针对算法提出相应的优化方案，考虑扩展更多的应用场景，并考虑在应用环境下设计相应的特征向量。

    4、本文的所有实验都是在离线的环境下完成的，并且实验的数据集均为MovieLens和Netflix提供的数据，希望在有条件的前提下，可以将其部署在线上，并测试其真实的结果。

# 参考文献

1. 刘建国，周涛.个性化推荐系统的研究进展[J].自然科学进展，2009,19(1):21-15.
2. X. Amatriain and J. Basilico. Netix recommendations: http://techblog.netix.com/2012/04/netix-recommendations-beyond-5-stars.html/.
3. 蔺丰奇,刘益.信息过载问题研究述评[J].情报理论与实践，2007,30(5):710-714.
4. R. Burke, M. O'Mahony, and N. Hurley. Robust collaborative recommendation. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, Recommender Systems Handbook, chapter 25, pages 805-835.
5. Kantor P B,Rokach L,Ricci F,et al.Recommender systems handbook[M].Berlin:Springer,2011.
6. Resnick P, Varian H R, Recommender system[J]. Communications of the ACM,1997,40(3):56-58.
7. 刘建国，周涛.个性化推荐系统的研究进展[J].自然科学进展,2009,19(1);1-15.
8. Cao J, Wu Z Wang Y, et al. Hybrid collaborative filtering algorithm for bidirectional Web service recommendation[J]. Knowledge and Information Systems,2013,36(3):607-627。
9. W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In Advances in Knowledge Discovery and Data Mining, pages 442–455. Springer, 2015.
10. J. J. Whang, X. Sui, and I. S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In International Conference on Data Mining, pages 705–714, 2012.
11. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In CSCW, 1994.
12. K. Hofmann, A. Schuth, A. Bellogin, and M. De Rijke. Effects of position bias on click-based recommender evaluation. In ECIR’14, pages 624–630. Springer,2014.
13. [Youcong Ni](http://scholar.cnki.net/result.aspx?q=%e4%bd%9c%e8%80%85:(Youcong+Ni)),[Bei Chen](http://scholar.cnki.net/result.aspx?q=%e4%bd%9c%e8%80%85:(Bei+Chen)),[Peng Ye](http://scholar.cnki.net/result.aspx?q=%e4%bd%9c%e8%80%85:(Peng+Ye)),[Chunyan Wang](http://scholar.cnki.net/result.aspx?q=%e4%bd%9c%e8%80%85:(Chunyan+Wang)) A Framework for iOS Application Development[J].Journal of Software,2014 (02):398-403.
14. 郭艳红. 推荐系统的协同过滤其法与应用研究[D]. 大连:大连理工大学,
15. Deshpande M, karypis G. Item-Based top-N recommendation algorithms. ACM Trans. On Information Systems,2004,22(1):143-177.
16. Li G, Li. A new algorithm of cold-start in a collaboration filtering system. Journal of Shandong University (engineering science),2012,2(24):12-17 (in Chinese with English abstract).
17. 梁彦.基于分布式平台Spark和YARN 的数据挖掘算法的并行化研究[D].中山大学,2014.
18. 冯琳.集群计算引擎Spark中内存优化研究与实现[D].清华大学，2013.
19. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In International conference on World Wide Web, pages 285-295. ACM, 2001.
20. F. M. Harper and J. A. Konstan. The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst., 5:19:1-19:19, Dec. 2015
21. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems.Computer, (8):30-37, 2009.
22. 项亮.动态推荐系统关键技术研究[D].博士学位论文.中国科学技术大学,2011.
23. Zheng N, Li QD. A recommender system based on tag and time information for social tagging systems. Expert Systems with Applications, 2011,38(4):457-485.
24. 李崇欣. 分布式数据库Hbase: the definitive guide[M]. Sebastopol: O’Reilly Media, Inc. 2015.
25. Huang CG, Yin J, Wang J, Liu YB, Wang JH. Uncertain neighbors collaborative filtering recommendation algorithm. Chinese Journal of Computers,2010,8(33):1370-1377(in Chinese with English abstract).
26. 李改. 一种解决协同过滤系统冷启动问题的新算法[J]. 山东大学学报(工学版),2012
27. T. Tang and P. Winoto. I should not recommend it to you even if you will like it: the ethics of recommender systems. New Review of Hypermedia and Multimedia, 19:111-138, 2016.
28. 徐翔. 协同过滤算法中的相似度优化方法[J]. 计算机工程,2010,36(06):52-56.
29. M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. neural information processing systems, pages 2595-2603,2010.
30. Lops P, De Gemmis M, Semeraro G. Content-based recommender systems: State of the art and trends[M] Springer US,2011:73-102.
31. Park S T, Chu W.Pairwise preference regression for cold-start recommendation[M].
32. List C, Pettit P.Aggregating sets of judgments: An impossibility result[J].Economics and Philosophy,2002,18(1) .

# 致 谢

首先，感谢我的母校中国科学技术大学给了我读研深造的机会，并且给了我良好的教育。在读研期间，学校的学习氛围以及严谨的治学态度给我留下深刻的印象，同时学校尽可能的为学生提供学习资料与良好的环境，增长了学生的见识，我以我的母校而自豪。

其次，要感谢我的班主任朱洪军老师，感谢他在读研期间对我的生活和学习上的关心及帮助，朱洪军老师在教学上的认真和对学生负责的态度给我留下深刻印象，同时也是我学习的榜样。最重要的是要感谢戴礼荣老师作为我的校内导师，感谢戴礼荣老师细心指导我的开题报告及论文的撰写，戴礼荣老师无论多忙多晚都会耐心为我解决疑惑，同时给予技术上的支持以及论文修改的建议。若没有戴礼荣老师的细心指导，论文的完成非常的困难。在这里非常感谢戴礼荣老师的指导，戴礼荣老师对学术要求严谨，同时又平易近人，给我留下深刻印象，衷心祝愿她永远健康快乐，工作顺利。

再次，要感谢平安科技(深圳)有限公司给了我实习的机会，使我了解了企业工作和学校学习的区别，让我提前进入工作氛围。同时感谢企业导师程宁，感谢他在工作中给予我的指导，为我解决了很多工作中的疑惑，同时为我的论文选题以及工作学习提供很多宝贵的建议，感谢她对我的工作和生活的关心。也要感谢同事们在工作上给予的技术指导，使我的技术得到很大提升。

最后，感谢我的家人和朋友，是他们一直在背后默默的支持和鼓励我，在遇到挫折时他们总会为我加油。感谢父母对我的栽培和付出，希望他们永远健康快乐。

2017年8月

**学位论文修改说明**

**意见1：**英文摘要有错误。

修改说明1：针对英文摘要重新做了翻译，纠正错误。详见英文摘要。

**意见2：**存在错字。

修改说明2：通读文章修改错字，如第3页“在加上公司的...”改为“再加上公司的...”。  
**意见3：**总结内容不准确。

修改说明3：重新撰写总结内容，概括本文所做工作。详见第68页7.1节。

**意见4：**英参考文献格式不对。

修改说明4：针对参考文献格式问题做相应调整，如行间距，字体，期刊引用起止页码。详见第82至83页参考文献。

**意见5：**一些流程图不对。

修改说明5：由于流程图中方框只允许有一个出口，第4章概要设计流程图中并行执行的操作流程不对，故将存在问题的流程图改为活动图，能够并行执行某些操作。具体见第4章图4.3，图4.4，图4.5，图4.6，图4.8，图4.11，图4.12。  
**意见6：**在系统实现中，没有系统模块的关联与调用关系。

修改说明6：针对该问题，在第5章添加5.1节系统总体设计与实现，给出系统各模块之间的类图关系，并描述各类作用及调用关系。详见第37至38页5.1节。