**面向重复记录检测的数据清洗算法的研究**

**作者姓名 张 攀**

**学校导师姓名、职称 刘志镜 教授**

**企业导师姓名、职称 吴春苗 高工**

**申请学位类别 工程硕士**

**面向重复记录检测的数据清洗算法的研究**

**西安电子科技大学**

**硕士学位论文**

**作者姓名：**张 攀

**领　　域：**计算机技术

**学位类别：**工程硕士

**学校导师姓名、职称：** 刘志镜 教授

**企业导师姓名、职称：** 吴春苗 高工

**学　　院：**计算机学院

**提交日期：**2017年6月

**学　号　 1503121785**

**密　级　 公开**

**学校代码 10701**

**分类号 TP39**

Supervisor: Liu Zhijing

Supervisor: Wu ChunMiao

Title: Professor

Title: Senior Engineer

A thesis submitted to

XIDIAN UNIVERSITY

in partial fulfillment of the requirements

for the degree of Master

in Computer Technology

**Research of Data Cleansing Algorithm for Duplicate** **Elimination**

By

Zhang Pan

June 2017

**西安电子科技大学**

**学位论文独创性（或创新性）声明**

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同事对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文若有不实之处，本人承担一切法律责任。

本人签名： 日 期：

**西安电子科技大学**

**关于论文使用授权的说明**

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权属于西安电子科技大学。学校有权保留送交论文的复印件，允许查阅、借阅论文；学校可以公布论文的全部或部分内容，允许采用影印、缩印或其它复制手段保存论文。同时本人保证，结合学位论文研究成果完成的论文、发明专利等成果，署名单位为西安电子科技大学。

保密的学位论文在 年解密后适用本授权书。

本人签名： 导师签名：

日 期： 日 期：

摘要

随着信息技术的发展与信息化建设的深入研究，激烈的市场竞争对于企业信息化程度的要求也越来越高。在形形色色的企业信息化系统进行不断地迭代和重构时，数据库中积累了大量的脏数据[1]，主要包括错误数据、相似重复数据和缺失数据三种类型。这些脏数据产生的原因多种多样，包括数据来源不同，存储于不同的操作系统以及硬件平台等[2]。其中多源数据的归并造成的数据重复则是最关键的热点问题[3]。

本文在对相似重复记录技术发展和研究现状进行简要介绍的基础上，首先详细地阐述了基于不同实现方式的重复检测技术，然后在借鉴前人工作成果的前提下，提出和设计了一种改进的MPN-P算法。该算法主要有三个改进点：通过统计字段区分度一定程度上改善了传统的MPN[4,5]算法在选择排序关键字时过于依赖专家经验的缺点；通过动态调整滑动窗口的步长与大小以节约时间消耗；通过标记排序关键字为空的记录提高算法应对缺失字段的能力，增强了鲁棒性。

其次当数据量增大到一定量级的时候，算法的效率问题随之凸显。本文将Java中的多核并行框架Fork/Join应用到MPN-P算法中，经实验对比可以验证其高效性。多核并行并没有改变算法是基于排序-归并[6,7]思想的事实，所以该解决方案依旧存在算法的检测精度对于排序关键字过于依赖的问题。针对大数据量的相似重复记录检测，本文提出一种改进的人工神经网络算法。首先通过已知小数据集作为训练数据对BP神经网络进行训练，然后利用遗传算法弥补其收敛速度慢和易陷入局部极小值的缺点，改进后的神经网络对于相似度向量输入可以迅速得到记录是否相似的判断。该算法的主要优势在于并不需要直接估算各属性的权重，而是通过神经网络学习属性之间的内在关系来完成记录是否匹配的判断，随着环境的变化具有较强的自适应能力。经对比试验证明，该算法拥有较好的检测精度。

**关键词**：数据清洗，重复记录检测，MPN-P，相似度向量，多核并行，BP神经网络

ABSTRACT

With the developme of information technology and information construction, fierce market competiton drives the enterprises to improve their information management. During the process of iteration and reconsitution of EIS, databases have accumulated a lot of “dirty data”, which consists of error data, duplicate data and missing data. There are varies of reasons of these data, such as the different data source, the different OS, the different hardware and so on. The key issue of these data cleansing problem is deduplication caused by the merge of muti-source data, which becomes the research focus nowadays.

In this paper we introduce the development and research status of the duplicatoin detection technology firstly, including their theory and implemention. Then a promoted algorithm called MPN-P is present on basis of previos work. There are three improvement, decreasing the dependence of expertise and experience by the statistic of fields’ discrimination when choosing sort key, decreasing time consumption by dynamically adjusting the size and step length of sliding window, and promoting rubust by marking those records whose sort key is null..

The efficiency becomes a fierce problem as the data grows too large. Fork/Join, the multi-core parallel computing framework, is used to improve its performance. It is the fact that this solution is base on the Sort-Merge method, which depends much on the choice of sort key and perform not so good at time consuming, although with the Fork/Join framework. This paper provides a promoted artificial neural network algorithm for the deduplication of large dataset. Firstly, a small dataset is used to be the input of a BP network. After training process, the network will evolute as the chromosome, then this network could rapidly decide weather two records are the same entity with the similarity vector being the input. This solution’s advantage is that it gets the result through the inner relation of records rather than estimates the fields’ weight directly, which enhances its self-adaption ability as the environment changes. This algorithm reaches a higher accuracy as experiments prove.

**Keywords:** Recommender systems, Real-time, Collaborative filtering,Storm

插图索引

[图2-1 Flume组成结构示意图 5](#_Toc482816758)

[图2-2 Flume数据流向示意图 5](#_Toc482816759)

[图2-3 Flume总体架构示意图 6](#_Toc482816760)

[图2-4 Flume抽象数据流向示意图 7](#_Toc482816761)

[图2-5 Kafka架构示意图 8](#_Toc482816762)

[图2-6 Kafka消费组示意图 9](#_Toc482816763)

[图2-7 Storm总体架构示意图 11](#_Toc482816764)

[图2-8 Storm工作时序图 12](#_Toc482816765)

[图2-9 HDFS整体架构示意图 13](#_Toc482816766)

[图2-10 MapReuce整体架构示意图 14](#_Toc482816767)

[图2-11 HDFS读取文件时序图 15](#_Toc482816768)

[图2-12 HDFS写入文件时序图 16](#_Toc482816769)

[图3-1 User-CF算法基本原理示意图 19](#_Toc482887542)

[图3-2 Item-CF算法基本原理示意图 21](#_Toc482887543)

[图3-3 Memory-base CF基本过程示意图 21](#_Toc482887544)

[图3-4 Model-Based CF基本过程示意图 22](#_Toc482887545)

[图3-5 SVDFeature模型原理示意图 25](#_Toc482887546)

[图4-1 系统总体需求示意图 29](#_Toc482887674)

[图4-2 日志采集系统需求示意图 30](#_Toc482887675)

[图4-3 推荐系统需求示意图 32](#_Toc482887676)

[图4-4 推荐方式组成示意图 33](#_Toc482887677)

[图4-5 个性化推荐算法流程图 35](#_Toc482887678)

[图4-6 UI需求示例图 36](#_Toc482887679)

[图5-1 实时推荐系统的总体结构示意图 39](#_Toc483579301)

[图5-2 流式计算流程图 40](#_Toc483579302)

[图5-3 实时推荐系统的数据流向示意图 41](#_Toc483579303)

[图5-4 日志采集系统框架示意图 43](#_Toc483579304)

[图5-5 日志采集数据序列图 46](#_Toc483579305)

[图5-6 数据接入模式架构示意图 47](#_Toc483579306)

[图5-7 Kafka生产者示意图 48](#_Toc483579307)

[图5-8 Flume的配置示例图 48](#_Toc483579308)

[图5-9 推荐算法计算过程流程图 51](#_Toc483579310)

[图5-10 Storm流式计算框架示意图 54](#_Toc483579311)

[图5-11 实时推荐实现流程图 57](#_Toc483579312)

[图5-12 小型电商APP界面展示图 58](#_Toc483579313)

[图5-13 推荐准确率柱状图 60](#_Toc479433389)

[图5-14 推荐召回率柱状图](#_Toc479433389) 60

表格索引

[表4-1 采集框架比较表 31](#_Toc477465047)

[表5-1 用户浏览信息表 42](#_Toc482800994)

[表5-2 用户检索信息表 42](#_Toc482800995)

[表5-3 用户收藏信息表 42](#_Toc482800996)

[表5-4 用户购买记录信息表 43](#_Toc482800997)

[表5-5 Flume配置方案表 44](#_Toc482800998)

[表5-6 数据源表 52](#_Toc482800999)

[表5-7 物品评分表 52](#_Toc482801000)

[表5-8 物品出现次数表 52](#_Toc482801001)

[表5-9 物品同现矩阵表 53](#_Toc482801002)

[表5-10 物品-用户评分矩阵表 53](#_Toc482801003)

[表5-11 推荐结果表 53](#_Toc482801004)

[表5-12 集群环境配置表 59](#_Toc482801004)

[表5-13 推荐计算时间表 61](#_Toc482801004)

[表5-13 不同算法计算时间表 61](#_Toc482801004)

符号对照表

exp 以e为底的指数函数

min 取最小值函数

∑ 求和

缩略语对照表

EIS Enterprise Information System 企业信息化系统

OS Operating System 操作系统

BP Back Propagation 反向传播

MPN Multi-Pass Sorted Neighborhood 多趟近邻排序算法

SNM Sorted Neighborhood Method 紧邻排序算法

目录

[摘要 I](#_Toc483640073)

[ABSTRACT III](#_Toc483640074)

[插图索引 V](#_Toc483640075)

[表格索引 VII](#_Toc483640076)

[符号对照表 IX](#_Toc483640077)

[缩略语对照表 XI](#_Toc483640078)

[第一章 绪论 1](#_Toc483640079)

[1.1 项目的背景和意义 1](#_Toc483640080)

[1.2 国内外研究现状 1](#_Toc483640081)

[1.3 论文研究的主要内容 2](#_Toc483640082)

[1.4 论文结构 3](#_Toc483640083)

[第二章 相关技术概述 5](#_Toc483640084)

[2.1 Flume 5](#_Toc483640085)

[2.1.1 Flume的特性 5](#_Toc483640086)

[2.1.2 Flume的逻辑架构 6](#_Toc483640087)

[2.2 Kafka 7](#_Toc483640088)

[2.2.1 Kafka的特性 7](#_Toc483640089)

[2.2.2 Kafka的架构 8](#_Toc483640090)

[2.2.3 Kafka的消费者模型 9](#_Toc483640091)

[2.3 Storm 10](#_Toc483640092)

[2.3.1 Storm的特性 10](#_Toc483640093)

[2.3.2 Storm的基础框架 10](#_Toc483640094)

[2.3.3 Storm的工作过程 11](#_Toc483640095)

[2.4 Hadoop 12](#_Toc483640096)

[2.4.1 Hadoop的特性 12](#_Toc483640097)

[2.4.2 Hadoop架构 13](#_Toc483640098)

[2.4.3 Hadoop的读写流程 14](#_Toc483640099)

[2.5 本章小结 16](#_Toc483640100)

[第三章 推荐系统算法研究和优化 19](#_Toc483640101)

[3.1 Memory-CF模型 19](#_Toc483640102)

[3.1.1 User-CF模型 19](#_Toc483640103)

[3.1.2 Item-CF模型 20](#_Toc483640104)

[3.2 Model-CF模型 22](#_Toc483640105)

[3.2.1 Logistic模型 22](#_Toc483640106)

[3.2.2 LFM(SVD)模型 23](#_Toc483640107)

[3.2.3 FM模型 25](#_Toc483640108)

[3.3 系统推荐算法的优化 25](#_Toc483640109)

[3.4 本章小结 27](#_Toc483640110)

[第四章 系统的需求分析 29](#_Toc483640111)

[4.1系统功能性需求 29](#_Toc483640112)

[4.1.1 日志收集系统 30](#_Toc483640113)

[4.1.2 推荐系统 32](#_Toc483640114)

[4.1.3 UI系统 36](#_Toc483640115)

[4.2本系统的重点需求 36](#_Toc483640116)

[4.2.1 实时性问题 36](#_Toc483640117)

[4.2.2 冷启动问题 37](#_Toc483640118)

[4.2.3 时效性和多样性问题 37](#_Toc483640119)

[4.3 本章总结 38](#_Toc483640120)

[第五章 系统的设计与实现 39](#_Toc483640121)

[5.1 整体框架 39](#_Toc483640122)

[5.2 日志采集模块 42](#_Toc483640123)

[5.2.1 日志数据类型 42](#_Toc483640124)

[5.2.2 日志采集系统框架 43](#_Toc483640125)

[5.2.3 日志采集系统连接器 44](#_Toc483640126)

[5.3 数据接入模块 47](#_Toc483640127)

[5.3.1 数据接入框架 47](#_Toc483640128)

[5.3.2 数据接入连接器 48](#_Toc483640129)

[5.4 离线计算平台模块 50](#_Toc483640130)

[5.4.1 基于物品的协同过滤算法原理 50](#_Toc483640131)

[5.4.2 算法实现过程 50](#_Toc483640132)

[5.5 流式计算平台模块 54](#_Toc483640133)

[5.5.1 Kafka与Storm的连接器 54](#_Toc483640134)

[5.5.2 实时推荐计算 55](#_Toc483640135)

[5.6 数据存储和展示模块 57](#_Toc483640136)

[5.6.1 数据存储和输出 57](#_Toc483640137)

[5.6.2 UI展示 58](#_Toc483640138)

[5.7 实验与测试 58](#_Toc483640139)

[5.7.1 评测标准 58](#_Toc483640140)

[5.7.2 实验设计 59](#_Toc483640141)

[5.7 本章小结 62](#_Toc483640142)

[第六章 总结与展望 63](#_Toc483640143)

[参考文献 65](#_Toc483640144)

[致谢 67](#_Toc483640145)

[作者简介 69](#_Toc483640146)

# 第一章 绪论

## 1.1 研究的背景和意义

信息化时代人类正在由工业化时代进入信息化时代，经济学家们普遍认为，进入21世纪后，信息将成为第一生产要素，同时将构成信息化社会的重要技术物质基础。为了在激烈的市场竞争中占据先机，各行业如保险、金融等纷纷加快了信息化的步伐。随着数据库技术的快速发展和广泛应用，形形色色的企业信息化系统应运而生。数据库的大小也与日逐增，人们已经得出如下结论，每过20个月世界上信息的数据量就会翻一番。

从规模庞大数据库中提取重要信息，从而对企业单位的发展提供参考，为决策者提供技术支持，是近年来数据挖掘的研究重点。然而由于不可避免的人为录入错误，或者是不同的数据表示方法，抑或是从不同的数据源合并数据甚至数据存储于不同的操作系统和物理设备，都不可避免地降低了系统的数据质量，从而产生各种“脏数据”。脏数据的类型主要包括重复数据、不完整数据、错误数据等。对这些数据若不进行清洗，则会影响信息化系统的正确运行，使得数据中提取的信息不再可靠，为企业决策支持和商务应用带来负面影响。因此，为了确保数据的准确性、一致性，数据清洗显得尤为重要。

最早的数据清洗过程需要大量的人为操作，所以当遇到较大规模的数据集，就会凸显出人为操作的低准确性和低效性。所以在当前数据规模急剧加大的情况下，只有借助信息技术，数据清洗才能实现其高效性。目前的信息化清洗过程中，仍不能完全离开专家的经验、人工的操作等行为，所以研究的一个重要方向就是尽可能减少人为的参与和影响。

相似重复的记录是数据库中降低数据质量最重要的一个原因，所以如何高效地检测和去除重复数据是数据清洗研究范畴的一个热点问题[8,9]。

同一个实体在数据库中不同的展现形式是相似重复记录的本质，它主要会引发以下的问题：

(1)资源浪费：重复记录会造成数据冗余，导致存储空间的极大浪费。

(2)破坏数据一致性：相似重复记录之间的关系可能是互为补充，也可能存在部分的冗余，甚至互相矛盾。当它们共同对应的现实中的实体发生变化，极有可能导致这些记录中只有某个或者某些记录发生改变，而其余无法同步更新。

以上，相似重复记录的检测与消除，保证了数据的一致性、减少资源的浪费，是数据清洗的重要环节。

## 1.2 国内外研究现状

早在上个世纪50年代，数据清洗已经开始了相关研究。将出自不同数据源的数据集进行整合自研究开始以来就被认为是一个困难而且极为重要的问题，开始主要是从数据连接[10]、数据实体识别[11]、对象识别等问题来研究，是商业保险、医疗、等领域中的研究重心之一。美国当时清除全美社会保险号数据集中的错误数据被视为数据清洗技术研究的开端[12]。

数据清洗的研究重心在以下几点：重复记录检测、异常数据检测、缺失数据的处理。数据仓库的出现以及数据挖掘相关技术的发展和应用，造成了多源数据进行合并容易出现大量重复数据的问题。因而相似重复记录的检测与清除成了数据清洗领域的研究重点。

在重复记录清洗方面，国外展开了大量的研究，主要的工作在于两个方面——属性匹配和重复检测。属性匹配问题的解决方法主要有Smith-Waterman算法、递归属性匹配算法、和R-S-W算法[13]。

相似重复记录检测大部分的算法采用排序然后归并的思想，即先将数据连接成一整个数据集，之后按照某种规则进行排序，将相似重复的记录排列在附近，最后通过某种相似判断方法检测出重复的记录，主要的差异在于排序方法和相似检测方法。最基本的算法是Jaro[14]提出的“排序&合并（Merge/Purge）”算法。这种算法存在明显的缺点，许多研究人员在此基础上提出了各种各样的改进思路和算法实现。

Monge等[15]将数据库中的一条记录视为一个字符串，在排序和比较的时候采用优先级队列的方法，检测相似重复时则使用了基于字符串的编辑距离。Hernandez等[5,4]提出了一种名为“多趟近邻排序”的算法，即MPN（Multi-Pass Sorted Neighborhood），该算法多次执行SNM（Sorted Neighborhood Method）同的排序关键字段以及较小的滑动窗口，最后判定记录相似时候使用的是一种规则定义的判等理论，它是使用C语言重写的OPS5[16]规则编程。Newcombe等[17]则采用应用相关的键值以将重复记录聚类到相近的位置。Qiu等[18]则是首先计算每条记录的N-gram统计值，然后根据这个N-gram值对数据集进行排序，最后再用用优先级队列的方式聚类检测重复记录。Gianni Costa等[19]采用文本聚类中的增量技术将新数据划分到最近的已知重复的聚类中，目的是为了解决大文本库中的相似检测问题。Alfredo Ferro等[20]使用了基于q-grams的相似度衡量函数，可以避免许多不必要的比较和判断，提高了时间效率。

国内的相关研究主要是对已知算法的改进和创新以实现更高的精度和效率。复旦大学周傲英等[21]等比较早开始数据清理的研究工作。邱越峰等[22]提出了基于N-Gram的相似记录检测算法。算法以一条数据的N-Gram值作为排序键，在对因为拼写造成的重复记录进行聚类时表现良好。!!!1韩京宇等[23]提出了一种基于q-gram空间层次聚类的方法。不同于传统的“排序&合并”的方法，该算法将数据映射成q-gram空间中的点，形成具有相似性层次子空间序列，采用层次聚类来实现相似重复记录的检测和清洗。方法不仅能避免应外排序引起的大量 I/O，而且克服排序不能保证将相似记录排在邻近位置的缺陷，还能通过层次逐步细化聚类来检测相似重复记录的方法提高检测精度。陈伟[24]提出一种按字段等级划分并将等级转变成权重的方法检测相似重复记录，给各个属性赋予合适的权重以提高检测精度，同时提出了长度过滤方法减少不必要的编辑距离计算进行相似重复记录检测优化。李星毅、庞雄文和周丽娟等[25,26,27]对大数据集相似记录检测问题。采用某种方法确定关键属性，依据关键属性进行数据分组，然后再划分后的组内进行相似重复记录检测的研究。这两种方法不同之处是确定关键属性的方法和组内聚类比较方法。

近几年，国内外的数据清洗市场的发展很快，其中包括商业上的数据清洗软件,也有各大学和研究机构开发的数据清洗软件[28,29]。

综上所述，国内外研究人员对数据清洗尤其对相似重复记录清洗的检测的研究已经得到取得了诸多进展，但仍旧或多或少存在适用面窄或者检测效率和精度不足等问题。

## 1.3 论文研究的主要内容

从国内外的研究现状可以看出，数据清洗领域己取得发展，促进了各行各业的发展。但数据清洗领域还存在一定问题，主要体现在：

(1)尽管检测重复记录受到很大关注，采取了许多措施，但检测效率与检测精度问题并不令人满意。大数据量的相似重复检测消耗大量时间，有待于更好的检测算法。

(2)大多数数据清理工具都是针对特定领域，其应用受到一定限制，未来特定领域的数据清洗仍是应用重点，但通用的清理方案应受到越来越多的关注。!!!1

(3)传统的相似重复记录检测大多基于“排序&归并”的思想，排序的效果以及最终归并的结果受排序关键字影响较大，尤其是当数据库排序关键字对应的字段为空或者是错误数据时，部分重复记录无法被正确的检测到，从而影响数据清洗的质量。

如何高效地检测相似重复记录，进而剔除数据库中的冗余数据，一直是数据清洗研究的重点问题。本文在分析了常用相似重复记录检测算法的基础上，针对传统的基本紧邻排序算法MPN在时间消耗和检测精度的不足，提出了改进的MPN-P算法。MPN-P算法的改进点主要在于：

(1)通过统计字段区分度一定程度上改善了传统的MPN算法在选择排序关键字时过于依赖专家经验的缺点。

(2)通过动态调整滑动窗口的步长与大小以节约时间消耗。

(3)通过标记排序关键字为空的记录提高算法应对缺失字段的能力，增强了鲁棒性。

为了节约MPN-P算法的时间消耗，本文将多核并行框架Fork/Join应用到改进后的算法中，经过充分的对比实验可以验证其高效性，从而充分地利用了多核多线程计算机的优势。

随着人工神经网络研究的兴起，越来越多的跨学科研究正在如火如荼地展开。本文将训练后的反向传播神经网络应用于相似重复记录检测中，利用两条记录对应字段间的相似度，构建基于神经网络的检测模型，利用遗传算法对网络模型的权值进行优化，使用遗传神经网络组合多个字段上的相似度来检测相似重复记录。在多个测试数据集上的测试结果表明，该方法能够提高相似重复记录检测的准确率和检测精度。

## 1.4 论文结构

论文的结构如下所示：

第一章，介绍了的数据清洗研究的背景和意义，相似重复记录的国内外的研究与发展现状，简单描述了论文的主要研究目的以及研究内容，展示了论文的组织架构。

第二章，主要介绍了相似重复记录检测的相关算法。首先简单介绍了最原始的算法直观方法，然后对最基本的紧邻排序算法SNM，介绍了算法的基本原理、设计思路以及大致实现和算法的优缺点。其次是在SNM基础上改进的MPN算法的设计原理、实现与优缺点分析。除此之外还有一些常用的算法思路比如优先队列算法等。

第三章，首先介绍了MPN-P算法的提出背景，然后详细介绍了算法的设计思路和其改进点，并采用SNM算法和MPN算法作为对照，进行了对比实验以验证算法的效率和检测精度等优势。

第四章，介绍了多核并行的基础理论，将Fork/Join并行框架应用到MPN-P的设计与实现思路，并与MPN-P做对比实验，验证多核并行实现对于算法在时间效率上的提高。

第五章，首先介绍了神经网络的基础知识以及神经网络在相似重复记录检测中的应用，接下来介绍了多种对于BP神经网络的改进思路，重点介绍了遗传算法对其的改进，并说明了如何使用改进的BP神经网络进行相似重复记录检测。通过使用不同数据量的测试数据集进行改进证明该算法的优势，并分析了它的缺点。

第六章，总结了本文的内容包括MPN-P算法与遗传神经网络进行重复检测研究中遇到的问题，以及算法本身的不足之处，并对未来的研究进行展望。

# 第二章 相关技术概述

## 2.1 Flume

Flume作为分布式的海量数据采集系统[5]，是一个可以对数据进行储存和传输，

且具有可靠性和可扩展性的组件框架，其组成结构如图2-1所示。



图2-1 Flume组成结构示意图

Flume的核心概念是利用数据源（Source）从Web端或移动端收集数据，然后将这些数据送达到目的地（Sink）。同时，数据会进行缓存（Changnel），待数据成功送达到终点，这些缓存才会被删除，这种方式保证了数据不会轻易的丢失。如图2-2，Flume都是将数据封装为Event进行传输，而传输的方向就是上述所描述的，从数据源经过缓冲器最后到达目的地。图2-2中事件（Event）这个数据，本质上就是一个Byte的数组，里面包含：Event headers、Event body、Event本身描述信息。

图2-2 Flume数据流向示意图

### 2.1.1 Flume的特性

Flume作为一个优秀的采集框架，其具有良好的性能特性，具体如下：

（1）可靠性：Flume利用事务来保证数据事件Event传输的可靠性。只有当数据事件Event被保存在缓存区或者数据已经传输给外部目的地的时候，Event数据才会被清楚掉，这种方式保证了数据的安全性。在分布式的条件下，当某个节点发生故障的时候，为了保证数据不会因此发生丢失，Flume采用了将发送的数据保存在Agent中，只有数据被传输成功之后，才会被删除，否则重新再发送，直至成功。如果数据接收方发生故障，数据会被写在本地，一旦数据接收方恢复正常，才会继续完成传输。

（2）可管理性：Flume中的Agent和Collector都由Matser进行管理，便于维护。分布式的情况下，可以利用Zookeeper和Gossip来保证数据的一致性。用户可以随时查看数据源和数据流的情况，同时可以采用Web操作和Linux下的shell操作，对数据进行配置和动态加载。

（3）功能可扩展性：开发者可以根据自己的需求定制或者添加每一层的组件。当然Flume本身也提供这些组件，如Agent(File、Systemlog)，Collector，Storage（File、HDFS）等。

### 2.1.2 Flume的逻辑架构



图2-3 Flume总体架构示意图

Flume采用分层的架构，每层的结构如图2-3所示。其中代理客户端Agent和收集端都由Source（数据源）、Channel（缓存池）和Sink（数据目的地）组成，Flume使用Master对代理客户端和存储端的资源进行调配控制。接下来我们简单介绍这些组件结构：

（1）Agent：该组件本身就可以作为一个完整的数据收集器。因为它具有完整的收集的组成组件（Source、Channel和Sink），而且Agent的Sink都是和Collector结合。

（2）Collector：如图2-3所示，本层结构主要任务是将传输过来的数据进行整合，再加载到存储层中。

（3）Storage：顾名思义是一个存储系统，该层可以存储一些普通的文件，也可以是复杂的分布式存储（如：HDFS）等。

（4）Master：Master可以理解为大脑中枢，管理每层的结构，负责控制和协调集群。

图2-4中描述了Flume中抽象的数据流向，其中数据流的概念就是描述数据从产生、传输和最后被写入保存的过程。



图2-4 Flume抽象数据流向示意图

## 2.2 Kafka

Kafka是一个分布式的消息队列系统[6]。当面对大量的数据汇入，导致计算处理模块无法一次性处理这么多的数据，那么就可以采用Kafka的发布订阅模式来解决。Kafka采用这种解决方案可以很好的解决数据吞吐量的问题，因此Kafka通常被用到处理分析系统的海量数据。

### 2.2.1 Kafka的特性

Kafka具有缓存海量数据和控制数据传输速度的作用，满足海量数据吞吐量的要求。该框架具有以下特性：

（1）高性能：采用最普通的硬件也可以处理几十万的消息，并且解决海量数据吞吐量的需求。因此，实现部署该框架无需花费大量的代价来满足硬件的需求。

（2）高可扩展性：当Kafka的容量不够时，可以通过简单的服务器横向扩展来增加Kafka集群的容量。

（3）分布式：面对数以万计的消息，当出现数据量过大，那么单机很难处理这样量级的数据。为了解决容量不足、性能不好等状况，Kafka支持分布式部署。

（4）持久化：Kafka为了防止数据的丢失，会将数据存储到硬盘上。同时Kafka对消息采取线性的存储方式，这种存储方式可以保证数据的查询速度。

### 2.2.2 Kafka的架构

Kafka属于分布式的架构，组件包含：Producer（生产者）、Broker（存储和转发组件）和Consumer（消费者）。外部数据的传输过程是通过Producer传达给Broker，Broker负责持久化和备份消息，实现了中间的缓存和分发的作用，同时通知注册过的消费者，消费者会根据自己的需求来主动获取消息，这些消息都被封装成Kafka消息的基本对象Topic。消息生产者使用推入（Push）的方式发布消息，而消费者消费该订阅消息采用拉取（Pull）的方式。生产者客户端和消费者客户端可以从多个Topic同时的读取数据。从图2-5可以看出Kafka的集群包含多个Broker，这些Broker类似一个存储池可以存储最近的消息记录（默认为7天，也可以自定义）。



图2-5 Kafka架构示意图

如图2-5所示，Kafka的基本组成结构包含生产者、存储层和消费者。下面简略介绍这些组件：

（1）Broker：Kafka的存储节点，每一个Kafka都会有这样的一个节点结构。该节点存储着各种各样的消息，按照队列形式保存，而Kafka集群由多个节点组成。

（2）Topic：Kafka消息的基本对象Topic，即Kafka需要处理的消息队列，一个Topic对应一组消息类型。

（3）Partition：如图2-5所示，Topic中存在P1和P2，可以得知P1和P2属于Topic，那么可以理解为Partition就是Topic的组成部分。

（4）Producer：顾名思义是生产者，是消息和数据的生产者，生产出的消息传输给Broker。

（5）Consumer：有生产者必定有消费者，和Producer配对成套的，主要负责处理消费这些消息。

### 2.2.3 Kafka的消费者模型

传统意义上的消息模型主要是两种，分别是队列和订阅发布。

在队列模型中，可以简单理解为消费者从队列中逐渐获取消息进行消费。在订阅发布的模型中，则是一组消费者订阅了消息，当出现新的消息的时候，就会将消息广播通知所有的消费者。举一个简单的例子：用户关注微信里面的公众号，当关注的公众号有新的消息内容，会推送提示用户，用户可以点击查看。

在Kafka中，不同的服务器、不同的进程、不同的线程上都对Consumer进行了唯一的标识（即唯一的ID），确保消费者的唯一性，保证了在一个消费组中，每一个消息只会被消费一次，当然其他的消费组的消费者也可以对这条消息进行消费。当消息需要被消费多次，则定义为不同的消费者即可。

如何实现消息的强顺序性，来保证消息的产生顺序和消费顺序保持一致，传统的消息队列是一种策略。但是当存在多个消费者时，即使服务端能够保证传递消息的顺序性，但是不同的消费者之间并不能保证同步，这样就使先产生的消息不一定被先处理，这种情况就无法保证了消息的顺序性。解决的方案是通过“独占消费”来实现，独占消费的概念是在每一个时刻，只能一个消费者来消费一个队列，不过这样的策略，会使并发性下降。



图2-6 Kafka消费组示意图

Kafka通过在分区上实现“独占消费”，确保了强顺序性和Topic上的并发性。如图2-6所示，一个Topic由多个分区组成，每一个分区在任意时刻只能被一个消费者消息，而多个分区之间互不影响。对于传统的队列，Kafka有两个明显的特性：

（1）Kafka上只能实现分区上的强顺序性，原因即是上述所说的，分区之间消费是互不干扰的。如果需要实现Topic级别的强顺序性，只需要建立一个分区的Topic来实现。

（2）同一个消费者中的消费者数目最好不要超过分区数目，因为某个时刻一个消费者只能消费到一个消息。

## 2.3 Storm

Storm是一个分布式的、高容错的实时计算系统[7]。如果说Hadoop的工作机制类似是一桶桶的搬运水，那么Storm就像是在已经安装好水管的前提下，只要打开水龙头，就可以即刻得到水，而且基本是源源不断。这就说明消息在Storm可以得到实时的处理，相比于其他的流式系统如：Samza[8]、MillWheel[9]、S4[10]。Storm具有更好的容错性和扩展性。

### 2.3.1 **Storm的特性**

Storm作为一个具有实时处理性能的计算框架，本身具有多种特性来满足数据的实时计算，主要的特性如下：

（1）容错性：主节点通过心跳机制来监控各个工作节点的状态，这些状态信息记录在ZooKeeper里面，当节点出现故障问题，可以重新启动。

（2）易用性：开发相对简易，只要按照开发的规范就可以很容易开发出适应性强的应用。Storm采用了简单的开发模型，可以降低实时处理复杂度；同时Storm还支持多种开发语言，开发者可以采用Java、Python或者Ruby等语言进行开发，方便开发者实现应用的需求。

（3）扩展性：运行基础上，依靠并行机制，可以通过增加硬件机器来扩展运行速度，解决处理之前没有考虑好计算量的情况。

（4）安全性：采用Acker 的机制，保证每一个数据包不轻易发生丢失，如果出现任务失败时，则会进行重新处理。

（5）处理快：采用ZMQ来进行通信处理，该方式具有并发性，数据处理的速度可以得到保证。

### 2.3.2 **Storm的基础框架**

Storm集群采用的是和Hadoop类似的主从结构框架，当然对应的语意上略有不同。在主从结构中，针对主节点的指定有两种模式，一种是通过配置文件中写入被称为静态指定。另一种则是在运行中会根据规则动态选择出主节点，称为动态指定。而Storm采用的是静态指定。

如2-7图所示，Storm集群具有一个Nimbus和多个Supervisor。Nimbus即Master主节点，Supervisor是对应Slave的工作节点。在这两个结构中间主要是依靠多个ZooKeeper的来负责协调工作运行。Nimbus负责管理，相当一个大领导，对集群上运行的Topology进行协调和监控。而Supervisor作为一个主任就是等待上层大领导发布的任务并管理自己的属下Worker进程，如果当前进程出现故障，则会分配给其他正常的Worker继续进行[11]。



图2-7 Storm总体架构示意图

### 2.3.3 **Storm的工作过程**

Storm的工作过程如图2-8所示，根据图示，下面列出具体的工作步骤：

（1）客户端将Topology的代码提交给Nimbus。

（2） Nimbus建立本地的Topology目录，并根据Topology提供的配置来计算任务Task，并分配Task资源。同时在Zookeeper上创建节点来保存任务Task和Supervisor中Worker的映射关系。

（3）Zookeeper采用心跳机制进行检测Task的运行情况，并启动Topology。

（4）Supervisor获取分配到的Task之后启动Worker，因为Worker是真正工作的进程，它会分配给每一个Task对应的线程资源，让Task可以正常的工作。

（5）根据Topology里面的配置信息创建Task之间的联系，而这些Task的连接

通过ZeroMQ进行管理。

（6）做好前面的工作之后，最后运行拓扑代码。



图2-8 Storm工作时序图

## 2.4 Hadoop

Hadoop[12]由分布式文件系统HDFS[13]和分布式计算框架MapReduce[14]构成。其中，HDFS主要负责大数据的存储，MapReduce负责在存储在HDFS中的大数据进行计算处理。

### 2.4.1 **Hadoop的特性**

Hadoop作为一个分布式的存储系统，本身特性可以满足大数据的存储需求，同时适用于海量数据批量的并行计算[15]。其具有如下特性：

（1）容错性：为了保证数据不会丢失，会备份多个数据块副本，当数据丢失或者某些节点出现故障导致，则会从其他节点的副本中复制过来，从而实现自动恢复，一般默认副本数目的3份。

（2）高吞吐：该系统适用于处理大文件，HDFS会将每个文件以64M这样的单位进行分割。当然如果小文件太多，而且文件大小远低64M，会导致内存吃紧，加重内存负担。

（3）低成本：HDFS系统可以运行在性能不高，廉价的机器上，这样可以利用一些性能不高的电脑来部署环境。

### 2.4.2 **Hadoop架构**

上面已经简单介绍了Hadoop的组成结构，接下来具体介绍组成结构HDFS和MapReduce的架构。HDFS为了存储海量的数据同时保证扩展性采用了主从的结构，其中NameNode作为Master，而Slave则是DataNode。整体的架构图如图2-9所示，下面简要介绍一下关于HDFS的各个组成部分：



图2-9 HDFS整体架构示意图

（1）NameNode：作为一个领导节点，记录着HDFS的名称空间。管理着数据块的分配位置，文件读取或者读写都需要通过该节点，同时也管理着各种配置情况，如：文件配置的副本数量。

（2）SecondaryNameNode：相当于NameNode的秘书，作为NameNode的冷备份，主要是负责合并Fsimage和Fsedits（定时默认为每个小时），然后再发给NameNode。

（3）DataNode：作为一个基层工人，负责HDFS主要的存储任务。

（4）Fsimage：文件系统中的元数据的镜像文件。

（5）Fsedits：文件系统中的元数据的操作日志，其中Namenode内存中存储结构由Fsimage和Fsedits组成。

Hadoop的MapReduce是一个专门针对海量数据计算的处理框架，该框架主要有以下结构组成：



图2-10 MapReuce整体架构示意图

（1）Client：可以理解为接入到Hadoop的客户端。

（2）JobTracker：作为一个后台的进程，如图2-10所示，利用心跳机制来监听下面任务的运行情况，包括资源使用和运行状态。一旦下面的某个TaskTacker出现故障，那么该节点上面原本的工作就会交接给其他的节点继续处理，保证任务执行的正确性。同时当任务调度器检测到空闲的资源的时候，则会通过它来对资源的利用。该结构很好的保证了hadoop的容错性并且提供了任务调度的依据。

（3）TaskTracker：通过心跳机制将该节点上的Task运行状况、Task资源的使用状况、Task健康状况汇总给JobTacker。同时也负责接收来自上层领导JobTacker发送来的命令，包括：开启任务，结束任务等。如果JobTracker是一个大领导，那么TaskTacker就是一个主任，负责和上层沟通，并将上层的指令下达给员工，员工的工作情况，他也需要汇报给上级领导。

（4） Task：顾名思义就是任务的意思，图2-10里面对应的两种Task是MapReduce实现的最基本的概念依据。

### 2.4.3 **Hadoop的读写流程**

本节介绍了HDFS中文件读取和文件的写入的过程，分布式文件系统在读写方面都有独特的机制来实现分布式和容错性。图2-11展示了HDFS文件读取过程，具体的步骤如下：

（1）如图所示，客户端打开文件系统，然后分布式文件系统通过RPC机制和NameNode建立连接，从而获取NameNode里面关于数据块地址的记录信息，包含文件数据块的列表信息和数据块对应到数据节点上的地址。

（2）HDFS客户端通过FSDataInputStrem的Read函数来读取数据。

（3）FSDataInputStrem会先获得和客户端最近的数据节点（DataNode）读取里

面的Block数据。

（4）当客户端读取数据节点上面的Block数据结束后，FSDataInputStrem则会关闭当前的连接，然后再连接下一个最适合的节点（即：最接近的节点）。

（5）读取信息顺序是按照NameNode里面的Block列表，当出现上次读取的列表无法保证文件被完全读取，那么可以从主节点中继续拉取接下来数据块的地址信息列表，保证文件可以被完全的读取完整。

（6）在读取过程中，当遇到DataNode读取失败，或者DataNode出现故障，则

需要从下一个备份的DataNode中读取，该备份机制很好的保证了数据的完整性和安全性。



图2-11 HDFS读取文件时序图

分析完HDFS读取文件的过程，接下来介绍HDFS写入文件的过程，如图2-12

所示，具体步骤如下：

（1）HDFS客户端发起创建文件的请求，然后分布式文件系统就会通过RPC，向主节点发出一个创建文件的请求，主节点会先判断是否已经存在该文件，如果没有，则还会检查是否具有创建文件的权限，如果具有才会真正的创建文件。

（2）创建文件成功之后，FSDataOuputStream会将数据切分成块，并写入到数据队列中。那么数据队列通过数据流读取，并通知主节点分配对应的数据节点来存储数据块，这些存储数据的数据节点会保存在同一个管道（Pipeline）里面。

（3）写入的过程是将数据块写到管道的第一个节点，然后第一个节点再复制到管道里面的第二个节点，以此传递复制到最后一个数据节点，呈现一种流水线的工作模式。

（4）数据存储完成之后会发送一个Ack包来通知客户端。当然客户端会维护这样一个Ack队列来确认每个文件是否写入成功，成功之后就会将该Ack从队列中去除，从而保证文件能够被正确的写入到HDFS。

（5）如果出现DataNode的故障，导致无法写入数据，那么在管道上就会将该节点去除，然后继续传输数据，如果这个时候默认是三块数据块，那么主节点在得知一个节点不能使用，会再分配一个数据块来填补坏掉的节点的空缺，保证写入的数据不丢失。



图2-12 HDFS写入文件时序图

## 2.5 本章小结

本章主要介绍了Flume、Kafka、Storm、Hadoop这几大组件框架，具体介绍了这几个框架组件。

其中，关于Flume组件方面，主要解释了Flume的特性和逻辑架构。Kafka方面，从它的特性、架构和消费模型进行了简要的阐述。Storm方面，解析了Storm的特征、基础架构和工作流程。Hadoop方面，主要是从它的特征、架构和HDFS读写流程进行介绍。本章技术的介绍，主要是为了后面论文的设计章结做铺垫。

# 第三章 推荐系统算法研究和优化

实现推荐系统，采用什么推荐算法是至关重要的一环。本章主要介绍协同过滤算法（CF Collaborative Filtering)，该算法从用户的操作行为中挖掘出有效的信息，通过算法计算出适当的内容和物品再推荐给用户[16]。目前，CF的算法主要有两种类型：Memory-CF和Model-based。本文采用Memory-CF中的基于物品的协同过滤推荐算法，利用该算法的进行离线计算，并且得到离线的物品相似度信息和离线推荐结果。最后简要分析了如何通过公式来对物品的相似度进行实时的更新。同时采用时间窗口，获取在时间窗口内的评分记录来更新计算从而得到最后的实时推荐结果。

## 3.1 Memory-CF模型

基于内存的协同过滤算法主要包含：基于用户的协同过滤算法和基于物品的协同过滤，基于内存的协同过滤算法通过之前的行为信息得到和用户相似的爱好（如：购买过同样的产品），结合用户本身购买或者收藏过的物品进行推荐[17]。

### 3.1.1 User-CF模型



图3-1 User-CF算法基本原理示意图

User-CF算法产生于1992年，主要在邮件过滤中被采用，其推荐机理如图3-1所示，从图中可以看出，该算法就是通过用户A和用户C同时喜欢物品A和物品C，然后判断用户A和用户C是具有相似性，所以图中用虚线标出将物品D指向给用户A来完成推荐。

如图3-1所示，可以发现User-CF很重要的一点是通过用户爱好相同物品来关联用户的关系。如何判断这种关系（即用户的相似度）是该算法核心。那么选择评判相似度方法是很重要的。而当前计算相似度的方法有很多种如：余弦相似度[18]、皮尔逊相似度[19]、欧几里得距离[20]和谷本系数[21]等。公式(3-1)和公式(3-2)表示了前两种方法的公式，两个公式里面的项表示用户x对物品i的评价， 项表示用户x对所有物品评分得到的平均值。

|  |  |  |
| --- | --- | --- |
|  |  | (3-1) |

|  |  |  |
| --- | --- | --- |
|  |  | (3-2) |

User-CF适用于用户的爱好比较稳定，而物品的更新比较快的场景，比如：音乐推荐，新闻内容推荐等。但是该算法也存在比较大的缺点：

（1）如果用户对物品的评分较少，就会出现稀疏矩阵的问题[22]，导致很难通过评分矩阵计算用户之间的相似度。

（2）该模型如果出现用户快速增长，就会造成计算的时间和空间复杂度过度增长，使得计算的难度增大。

### 3.1.2 Item-CF模型

Item-CF（基于物品相似度）算法是目前在商业中被广泛采用的一种算法。该算法是计算物品之间的相似度，通过用户的操作行为得到用户的兴趣物品，找到和该兴趣物品相似物品，推荐给该用户，当然需要过滤掉用户之前已经消费过的物品。Item-CF算法基本原理示意图如图3-2所示，应用A和应用C都是属于新闻娱乐的类型，那么之间相似度较大，就可以将应用C推荐给喜欢应用A的用户A。

如图3-2所示，Item-CF算法主要分为两个步骤：

（1）需要计算物品之间相似度，公式可以参考上面的公式(3-1)和公式(3-2)。

（2）计算得到推荐列表，通过用户的购买行为结合物品相似度进行推荐。



图3-2 Item-CF算法基本原理示意图

相对于User-CF注重通过用户群体来进行关联的特点，Item-CF更加注重用户的独特的兴趣爱好，可以更加彰显个性化。同时当遇到物品数目远远低于用户数目，如：某电商网站，用户数目达到上亿，而物品只有几百万。那么如果采用User-CF的话，就会得到一个庞大的稀疏矩阵，很难通过计算得到用户的关联，或者得到的关联结果也是可有可无。而Item-CF可以很好的解决上述的问题，即使用户的行为快速的变换，之前的计算得到的矩阵也不需要过大的变化，推荐结果可以快速产生。当然Item-CF也具有一个缺点，如果使用公式(3-1)或者公式(3-2)会导致热门或者热销的产品会和所有存在的产品关联相似度过高，特别是他们本身的关联会特别的显著，那么推荐的结果会存在大量的热门产品。当然这种情况可以采用基于二部图的Heat Spread算法在某种程度上平衡推荐的效果。



图3-3 Memory-base CF基本过程示意图

Momory-based CF算法的基本流程如图3-3所示，算法的输入端是一个评分矩阵，将矩阵数据传输给CF算法进行计算，在推荐计算过程中包含两个步骤，分别是预测和推荐。其中预测的环节就是计算用户可能对物品可能的评分值，推荐则是利用用户近期的行为和所在的场合对上述预测得到的物品进行一个合适的排序，从而将排序之后得到的响应用户供其选择。

## 3.2 Model-CF模型

Model-based CF算法是需要学习训练的一类推荐算法的统称[23]，其中包括LFM模型[24]、Logistic回归、FM模型、贝叶斯网络算法[25]等。图3-4展示了model-based CF实现的简单流程，其中包含了和业务关联的数据源、各种特征的构造、离线的模型训练、离线和在线的推荐物品打分。



图3-4 Model-Based CF基本过程示意图

### 3.2.1 Logistic模型

Logistic模型将用户的消费行为进行一个分类，分为正样本和负样本（即二分类思想）。正样本可以是用户消费该物品，包括对物品的收藏、点击或者购买，这些对于推测用户的兴趣起到正面的作用，而针对负样本可以从用户从来没有关注或者浏览过的物品中选择，负样本的构造方式则是多种多样。构建完成之后需要提取前面得到的样本对应的特征向量，再将得到的样本放入Logistic分类器中进行学习，通过学习可以得到一个消费模型，该消费模型可以得到一个评分列表，如果评分越高就说明用户对应物品的偏好程度越高。

针对Logistic模型，假设正负样本满足如公式(3-3)的概率分布，其中y代表样本x的一个标签，可以取值正1或负1，而w、b表示对应的分布参数。

|  |  |  |
| --- | --- | --- |
|  |  | (3-3) |

通过最大似然估计方法可以得到如下的优化目标表达式：

|  |  |  |
| --- | --- | --- |
|  |  | (3-4) |

公式(3-4)中的项表示正则项， C是一个可调的常数，L表示样本数。实现上述目标可以可以采用牛顿法、拟牛顿法、共轭梯度法等优化算法，获取权重向量w。

Logistic模型主要针对特征维度数目不过高的场景，在这种场景下，模型训练得到的结果还是比较得当。但是该模型无法刻画非线性的关系，如：假设取得x维度的用户的行为特征，y维度的物品基础特征，该模型只能对x+y维度的线性特征进行描述关联，而无法得出特征之间的非线性关系。为了使Logistic模型能够在推荐系统中起到显著的作用，需要获取用户和物品之间cross的特征，对他们的基础特征进行互相关联。

### 3.2.2 LFM(SVD)模型

LFM(Latent Factor Model)隐含因素模型利用之前已经存在的评分数据，通过分析用户对应的隐含因素（如物品包含各种因素的程度），最后得到分析结果进行预测，来填补用户的评分矩阵。补全矩阵要尽量减少对原本矩阵影响，即原本的矩阵和处理后的矩阵之间的特征值相差不能太大。为了获取用户和物品之间的隐含量，通常采用SVD的算法来实现，主要是将用户和物品的评分矩阵进行分解得到。我们可以假设有m个用户和n个物品，那么 对应的是用户的评价矩阵，则SVD算法将补全后的矩阵进行分解得到如下形式：

|  |  |  |
| --- | --- | --- |
|  |  | (3-5) |

公式(3-5)中，项和为项表示对应的正交矩阵，表示一个对角矩阵，为了得到用户和物品之间存在k个因素的关联，需要获得k个特征值（特征值越大越好）表示左右奇异向量，可以得到近似得公式如下所示：

|  |  |  |
| --- | --- | --- |
|  |  | (3-6) |

公式(3-6)中的 项表示对应的m个user的隐向量矩阵，表示对应的n个item的隐向量矩阵，通过计算向量内积可以得到用户对物品的评分。

最基本的SVD的算法具有两个缺点，一个是由于在真实的平台上一般用户只会对自己感兴趣的东西进行操作，而对于大多数的产品是不会产生关联的。所以这样就会导致在获得的评分矩阵的具有很大的稀疏性，那么针对矩阵补全就需要巨大的存储空间。另外一个缺点是该算法的复杂度太高，达到O()这样的级别，这是一个可怕的复杂度，一旦数据量太大，该算法就无法进行计算。针对这种情况，Simon Funk提出一种可以快速实现矩阵分解算法，将评分矩阵分解为两个低维度矩阵的乘积如公式(3-7)所示：

|  |  |  |
| --- | --- | --- |
|  |  | (3-7) |

Simon Funk的算法主要是通过学习这两个矩阵，获得最小化损失函数，如公式(3-8)所示：

|  |  |  |
| --- | --- | --- |
|  |  | (3-8) |

公式(3-8)中的代表用户u对于物品i的一个评价，其中 分别表示用户u对应的隐向量和物品i对应的隐向量，表示可调参数，控制正则项影响，隐向量特征取值表示为用户u对第j类物品的喜爱值，隐向量特取值表示物品i隶属于第j类物品的置信度。

后来出现的BiasSVD、SVD++和SVDFeature算法都是基于Simon Funk的算法的基础上进行改造的。而这里面SVDFeature算法可以作为LFM系列中相对具有代表性算法，该算法模型了涵盖了基本的SVD、BiasSVD和SVD++模型，公式如下所示：

(3-9)

公式(3-9)中的表示求得所有的评分的一个平均数，表示的其他的所有特征取值，项代表用户特征取值，项表示为物品特征取值，为全局特征，是用户特征、表示物品特征的偏置权重，、为用户、物品特征对应隐向量。该算法的模型如图3-5所示。

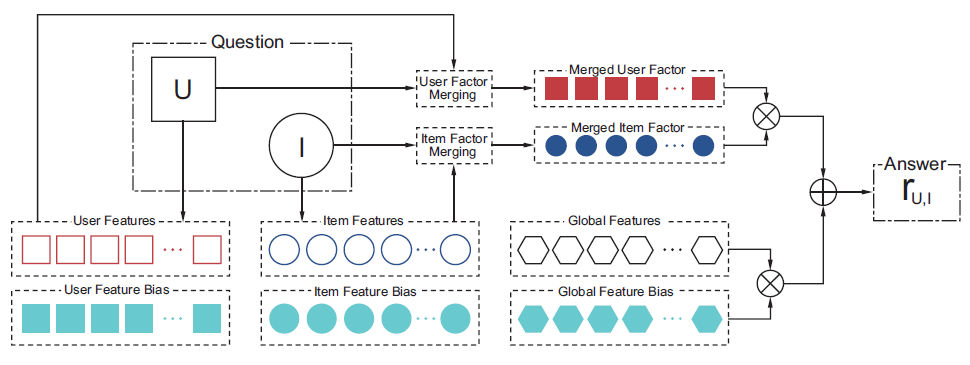


图3-5 SVDFeature模型原理示意图

SVDFeature的目标函数可以通过SGD(Stochastic Gradient Descent)随机梯度下降算法或者ALS(Alternating Least-Squares)交替最小均方误差来实现。但是SVDFeature算法具有一个缺陷，即通过学习得到的全局特征偏置项、用户特征偏置项和物品特征偏置项，他们的权重都会比较高，而他们之间隐向量的内积值很低，这很不利于进行个性化的推荐。

### 3.2.3 FM模型

FM(Factorization Machine)框架模型作为一种比SVDFeature模型更加完整的分解机的模型，可以实现SVD对用户特征的隐向量和物品对应的特征进行交叉的功能及特性，它还可以满足用户本身的特征隐向量的交叉和物品自己本身的特征隐向量的交叉，它对应的模型如公式(3-10)所示。

|  |  |  |
| --- | --- | --- |
|  |  | (3-10) |

公式中项对应的是用户和物品特征，p为特征总维数，则是特征j对应隐向量。FM公式中项和SVDFeature公式中 项表达的含义是相同的，都是对用户和物品基础特征在模型中权重影响的分布的描述。FM模型中的则是表达所有特征两两之间的交叉特征的在模型中的影响。

## 3.3 系统推荐算法的优化

由于本系统是基于物品的推荐，通过上述的介绍和分析，因此本系统采用的是基于物品的协同过滤算法。系统中采用的相似度计算公式如(3-11)所示。

|  |  |  |
| --- | --- | --- |
|  |  | (3-11) |

公式中表示喜欢物品x的用户数目，分子表示为同时喜欢物品x和物品y的用户数目。因此上面的公式可以理解为喜欢物品x的用户中有多少的比例用户喜欢物品y。

根据上述得到的物品相似度计算用户对物品的预测评分值，系统中采用的如下公式进行计算：

|  |  |  |
| --- | --- | --- |
|  |  | (3-12) |

公式里面R(u)表示用户已经评分的集合，S(j,k)表示和物品j最相似的K个物品，而表示了物品j和物品i的相似度，表示用户u对物品i的评分。通过该公式计算得到评分列表再进行排序，即最终得到的推荐结果列表。本文中采用了增量更新物品相似度和利用时间窗口来优化本系统采用的算法。

（1）增量更新物品相似度

本系统计算物品相似度是放在离线平台上进行的，当面对庞大的数据量想要通过实时计算并快速得出推荐结果是不现实的。所以将重量级的计算模块放在离线平台进行周期的批量处理，这个周期可根据数据量情况而定。同时流式计算平台实时的获得近期用户的行为，对用户行为兴趣程度进行分析。系统主要是对物品相似度进行更新计算，因为该计算公式比较简单，所以该计算规模比较小，计算更新速度比较快，采用的更新公式为（3-11）。

针对上述的公式进行分析，假设用户M 对物品x有过评分，那么相对而言物品相似度如何更新：

a. 如果用户M对物品x有过评分，即对该物品有产生行为信息，那么我们不需要更新任何物品相似度。

b. 如果用户M首次对物品X产生事件行为，对其进行了评分。那么相对的就会增加一。而通过公式中的，如果出现用户M对物品y也有过事件行为，则需要增加一；如果用户对物品y没有过事件行为，则保持不变。

（2）实时推荐的算法

实时推荐算法和离线处理的推荐算法不同之处在于，实时推荐要求其计算量要小，需要避免过多的计算导致用户体验下降。离线计算推荐物品按照公式(3-12)进行计算，主要是根据其过往的评分记录进行计算。而在线实时推荐需要保证响应速度，并根据用户当前的一段时间内最能反映用户的兴趣的原理。在实时推荐计算阶段，不会使用用户全部的评分记录，而是使用最近一段时间的行为记录通过公式进行计算，同样是根据公式(3-12)进行计算。

实时推荐算法里面是根据用户最近的行为进行推荐，那么就需要进行一个时间上的更新定义，对用户早期不能对当前行为产生影响的行为记录进行清洗。

1. 根据时间的窗口，只有在一定的时间范围内的用户行为才会被保存起来，用

来推荐计算，而在该范围外的数据将被清洗掉。

1. 根据操作行为的个数，保存一个用户的操作行为队列，如果该队列满了，则

将清洗队列最前面的事件。

本系统的实时推荐算法，主要是采用的是根据时间窗口的方法。该算法采用就是基于物品的协同过滤算法，而算法时间复杂度取决于相似度量的方法。算法采用了Jaccard的余弦公式（即：公式3-11），其时间复杂度为O（|X|\*|X|），X表示物品的数目，这适用于物品数目相对稳定的电商的平台上。而如果采用基于用户的推荐算法，那么X则对应就是用户数目，因为在电商平台上，用户的数目可能会急速得增加导致时间复杂度过大。而如果采用基于模型的协同过滤的算法，那么在实时更新计算上，几乎无法通过相应的公式计算得到推荐结果。

系统中采用相对简单的更新相似度方法，目的是为了让系统能在短时间内，对离线的物品相似度矩阵进行快速更新。对推荐列表的计算本文根据时间窗口来减少对全部用户行为计算，这样可以很好得保证了计算速率和推荐效果。所以和上述复杂的推荐算法相比，采用这种算法和公式，可以更加容易满足实时推荐的效果。

## 3.4 本章小结

本章主要是对推荐算法中的协同过滤算法及算法流程、适应场景进行介绍。其中基于内存的协同过滤算法主要包括基于基于物品的协同过滤和基于物品的协同过滤，

这些算法主要以相似性为核心，依赖用户的行为数据。基于模型的协同过滤算法主要包括Logistic模型、LFM模型和FM模型。这些模型需要对数据进行特征训练，基本过程包括：数据源的准备、特征构造、模型训练、最后得到推荐的物品，依据用户需求将物品推荐给对应的用户。最后简述了本系统采用的基于物品的协同过滤推荐算法，并利用公式对推荐结果进行实时的更新。

# 第四章 系统的需求分析

许多的互联网用户在网上都会体验到推荐系统带来的福利。当你的一个朋友推荐给你一件商品之后，你在淘宝或者京东上面搜索这件商品，它会出现一系列返回的结果。同时，这些结果会出现推荐的类似商品，或者是列出“买了这件商品的用户可能还买了哪些产品”。如果你经常在京东或者淘宝上，那么这些平台会根据你的购买记录和操作行为为你推荐一组商品。而我们所探讨的推荐系统，就是决定把哪些商品推荐给特定的用户。

在电商平台上由于每个用户对物品的兴趣需求不同，对商品的要求不同，那么推荐给不同用户，所对应的推荐列表必然不一样，为了实现这种效果就需要运用到个性化推荐的技术。相对而言，部分现有的产品只是将最近最热门的或者最畅销的商品推荐给每一个用户，导致对不同的用户推荐并没有任何区别。这当然也可以理解为一种推荐，毕竟是一种客观的大众思维。但实际上，热门和畅销的并不一定就适合每个用户，所以推荐热销产品的作用度有限。因此，个性化推荐更加具有价值性。

而在个性化推荐的基础上，如何实时的将这些信息推荐给用户，将具有更大的价值。在市场上的如“今日头条”就是一款号称会根据用户的每次操作，在短时间内展示给用户最新的推荐结果。

## 4.1系统功能性需求

本系统为了实现电商的实时推荐系统，采用了目前比较流行的流计算框架Storm进行开发，那么要构建实时的推荐系统需要以下几个模块系统。如图4-1的功能图所示：



图4-1系统总体需求示意图

（1）日志采集系统：构建该模块的目的是为了收集大量的实时数据，这些数据主要是用户在平台上的操作信息，例如：浏览商品信息，购买商品信息，收藏商品信息等。这些数据需要被收集并且进行存储，存采集的数据需要随时提供给后面的推荐系统使用。

（2）推荐系统：这部分是本系统的核心内容，解决如何能够实时推荐的问题。该部分包含了离线平台和在线平台两个部分，离线平台主要是将存储在HDFS的大量数据进行一个离线的计算，通过基于物品的协同过滤算法周期性的（一天或者一周，根据数据量情况而定）处理过去一段时间的数据，并获得离线的物品相似度信息和物品画像信息。在流式计算平台中，主要计算实时的数据流，针对用户实时的操作，通过在线引擎进行增量计算得出最后的推荐结果。采用双平台架构实现实时推荐系统，主要是由于互联网信息的数据量的过于庞大，那么用户之前积累了大量的行为信息需要通过离线平台进行预处理，处理得出基本的物品的模型特征。而在线平台则只负责对这些特征进行增量更新计算即可得到最后的推荐结果。

（3）UI系统：本部分是将推荐结果展现出来，但是该部分具有很大的讲究，UI系统需要解决如何将推荐结果展示让用户觉得不突兀，甚至让用户觉得惊喜的问题。

### 4.1.1 日志收集系统

日志采集系统，从Web端和移动端采集数据，采集之后将数据进行存储。



图4-2日志采集系统需求示意图

现今，许多的公司的平台每天都会产生大量的信息日志，这些日志基本都是流式数据如：搜索信息、查询信息、点击信息等。而针对自己的业务，每个公司都会设计一个日志采集系统实时的收集平台上实时产生的数据，日志采集系统一般需要具有如下特点：

（1）需要建立应用系统（Web、APP等）和分析处理系统（如：推荐系统，日志分析系统）之间的通道。让实时产生的数据可以及时的被采集并可控制的传输给分析系统进行处理，这也很好的解耦了应用系统与分析处理系统的直接关联。

（2）需要满足实时在线分析系统的要求，通过实时采集数据并将这些数据实时的发送给在线分析系统进行解析。

（3）需要扩展性强，当数据量突然变大的时候（典型如：淘宝的双11），可以

通过增加节数来满足数据的涌入。

（4）需要容错性好，当出现故障的时候，需要很好的容错，避免出现问题。

（5）需要采用合适的框架来实现，这样可以方便开发并缩短开发时间，针对不同的框架需要考虑各种条件包括：开发简易性、容错性、负载均衡、可扩展性等。

根据采集系统的需求，本文对目前的一些开源的采集框架进行了探讨和研究，得到采集框架比较表如下：

表4-1采集框架比较表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 框架 | Scirbe[26] | ChuKwa[27] | Kafka | Flume |
| 公司 | Facebook | Apache/yahoo | LinkedIn | Cloudera |
| 开源时间 | 2008.10 | 2009.11 | 2010.12 | 2009. 7 |
| 实现语言 | C/C++ | JAVA | SCALA | JAVA |
| 框架模式 | Push/Push模式 | Push/Push模式 | Push/Pull模式 | Push/Push模式 |
| 容错性 | 具有一定的容错机制，不过容错机制部分需要用户自己实现。 | Agent会记录数据偏移量。一旦出现问题，可根据记录进行恢复。 | 该框架的容错机制和ChuWa类似，也是记录了数据偏移量。 | 该框架组件提供了三种级别的机制来保证容错性。 |
| 负载均衡 | 无 | 无 | Zookeeper | Zookeeper |
| 可扩展性 | 好 | 好 | 好 | 好 |
| Agent | 需要开发者自己实现 | 本身已经自带一些agent。 | 用户需根据实现Kafka提供的组件，进行对应的开发。 | 提供了各种非常丰富的agent，可以直接使用。 |
| Collector | 可以理解为是一个简单的存储服务。 | -- | 使用了各种技术提高性能能。 | 系统本身提供很多的存储层，且可直接使用。 |
| Store | 支持HDFS | 支持HDFS | 支持HDFS | 支持HDFS |
| 评价 | 该框架设计相对比较简单，且也容易使用，但是没有实现负载均衡，自身的容错性不够好。 | 该框架主要是针对比较大型的采集系统。 | 该框架的设计非常的巧妙，采用了push和push模型。 | 在采集和输出方面都有组件可以直接使用，开发相对方便简单。而且本身容错性好。 |

日志采集系统的开发可以利用已经存在的框架进行开发，相对而言自己去造轮子，这种基于成熟的框架来实现开发是比较简单的。而经过比较之后，本论文更倾向使用Flume架构来进行开发。

### 4.1.2 推荐系统

随着推荐系统的日益发展，人们对它的要求也日益增长，那么实现推荐系统需要考虑哪些方面才能满足用户的体验。如图4-3，我们来逐一讨论推荐系统的需求。



图4-3推荐系统需求示意图

（1）满意度：这是推荐系统中最重要的一个评判标准，因为推荐系统本身就是为了推荐给用户最满意的物品，所以推荐系统里面的推荐准确率是衡量推荐系统最重要的一个因素。

（2）实时性：本文需要实现实时的推荐系统，所以实时性需要重点考虑。随着技术的发展，实时性的推荐已经越来越成为主流。其中实时性体现在短时间内，系统可以根据上下文信息，或者用户的操作行为来快速得出用户模型，然后针对不同的用户模型对推荐列表进行更新。毕竟有些用户的爱好和兴趣点总是容易变化，那么推荐内容就需要实时的更新来满足用户的需要。

（3）新颖性：推荐内容如何让用户觉得特别新鲜，一种推荐方法是推荐给用户从来没有见过的物品，当然这种物品又需要让用户有点兴趣。如：用户A在电商平台上面见到 “购买过该物品的用户也购买这些产品”，那么如果用户B也购买过同一个产品（或者多个相同的产品），可以得到用户A和用户B的具有共同的爱好，用户B所购买的物品很有可能也符合用户A的需求，那么再去掉用户A之前购买或者浏览过的物品，这样推荐给用户A的效果就是就会非常具有新颖性。

（4）惊喜度：和上述的新颖性的基本上是类似。相对新颖性，惊喜度的概念更加简单，只要推荐给用户想要的产品，让用户觉得系统可以洞悉自己的内心的需求，惊喜度和新颖度的区别是推荐集合不需要排除掉自己之前浏览过的物品。

（5）多样性：推荐系统中，推荐的多样性作为一种兼顾的需求。在满足用户满意度的前提下，推荐的物品越丰富多彩，那么给予用户选择也就更大。推荐多样性的物品，需要根据推荐物品权重比值进行推荐。其中推荐排序，都和该权重有关系，权重越大，推荐顺序越靠前，反之越靠后。

（6）透明度：这个需求是一种友好的用户体验，让用户知道为什么推荐这些物品给他们。如：我们经常在电商平台上看到“买过这件衣服的人曾经也买过如下产品”、“你现在购买的鞋子和如下的产品类似…”。这样给予用户一种透明推荐的感觉，让用户明白推荐的物品的依据，让用户体验得到提升。

（7）覆盖率：推荐系统推荐的内容需要覆盖大量的内容信息，而这些信息就需要进行挖掘物品的长尾内容。一般评判标准是推荐概率分布的物品占总物品的比例和所有物品被推荐的概率进行计算。

关于推荐系统的方式，推荐方式多种多样，为了达到上诉的推荐效果，在理论上可以归结为以下几种推荐方式，如图4-4所示。



图4-4 推荐方式组成示意图

（1）热门推荐：顾名思义是热门排行，最典型的莫过于微博热搜，根据用户点击量来判断微博的热度，进行一个热门排行展示。如：音乐热门排行、书籍热门排行，这种方式在生活中无处不在。作为一种大众的推荐方式，按曝光率排序物品展现给用户，大多数从众心理的用户会被吸引去浏览这些物品。这种推荐也具有良好的效果。

（2）人工推荐：这是一种最原始的方式，但也很有效。在没办法建立特定模型，就需要依靠人工的方式进行推荐，这种方式很依靠推荐人对该领域积累的经验。还有关于热门实时产生的新闻，内容信息一开始需要人工加入推荐列表，这样才会引起更多的关注度。

（3）个性化推荐：这种推荐方式具有独特性，主要是根据用户之前做过的行为来建立用户兴趣模型和其他的关系模型，计算得到各种不同的推荐列表。因此每个用户的推荐就会都不一样，这和热门推荐从众心理完全不一样。

（4）相关推荐：通过用户目前的行为状态和兴趣爱好关联相关的内容的进行推荐的方式。如：用户浏览一本书的某章节内容的时候，那么系统会提示目前用户浏览与此书籍相关的内容。这种推荐和个性化推荐的关联规则基本上是一致的。

本文讨论的系统主要是针对个性化推荐的基础上的应用，因为个性化的推荐具有独特性，同时可以解决信息过载问题，让用户感受到推荐系统的强大性。目前个性化推荐的算法主要有：

（1）基于内容的推荐[28][29]：通过推荐内容本身的信息来进行推荐，如内容关于历史信息的，那么就会推荐关于历史的一些内容。

（2）基于关联规则的推荐：该推荐方法需要进行大量的数据挖掘得到这种推荐物品的关联，最出名的就是“尿布和啤酒”的案例，看似没有什么关联的两个物品，通过关联挖掘得到联系。

（3）协同过滤推荐：上述的章节对此算法已经做了较详细的研究，主要是Item-CF、User-CF、Mode-CF等几种推荐算法，通过该算法会向不同的用户推荐可能感兴趣的商品[30]。该算法主要是通过获取用户的行为数据并进行长期的积累，然后进行各种模型分析得到用户的个性化的特点，最后得出推荐结果的一种特色推荐。当然也存在各种复合的算法，如协调过滤和基于内容的结合算法[31]。

那么，本文重点讨论关于个性化的推荐算法的基本需求，那么下面具体的阐述个性化推荐算法的流程的各个步骤。

（1）获取日志信息：关于用户的行为信息，如：浏览某个物品，点击某个操作等。这些大量的行为信息都会被存储起来，主要是依靠日志采集系统来进行收集，并进行存储，一般存储在HDFS中。

（2）前期的ETL（Extract-Transform-Load）数据：采集用户最原始的行为数据之后，需要对原始的数据进行清洗，主要是将后期不需要的信息进行过滤掉，这样可以减少大量的数据，同时也避免脏数据或者无用数据对后面计算存在干扰。可以将数据按照后面要求的格式进行加工如：字段要求、属性要求、格式化等，这样可以满足算法对数据的直接调用。

（3）推荐算法：选用什么推荐算法，要根据不同的业务要求来进行选择和设计，目前已经存在各种各样的算法可以运用。第三章节已经对协同过滤算法进行了仔细的探讨和研究，这边不再重复阐述。

（4）后期的ETL数据：该数据是经过推荐算法处理之后的信息，不过需要将这些数据进行分离处理，因为在算法计算过程并不只是产生我们要的推荐内容，还会产生很多关于用户或者物品的偏好矩阵，如：用户相似度、物品相似度、内容信息的关联等。

（5）画像信息：该画像信息包含算法计算过程中得到的用户兴趣和操作的行为信息。这些信息可以重复利用，不需要每次都重新计算得到。当然这些爱好行为会随着时间的向前，画像信息也需要进行局部的更新或者完全更新。

（6）推荐列表：按照要求存储得到的推荐结果，如：用户对应推荐列表，物品对应的推荐列表。系统中对于这些数据的存储的容量和读写性能要求都比较高，可以选择使用Redis集群作为此部分数据的存储方案。

（7）API服务接口：这个是暴露给外部的调用接口，满足开发者的调用，开发者可以调用得到的推荐结果列表，也可以调用用户或者物品的的画像信息。

通过上述的流程阐述，具体的算法流程如图4-5所示；



图4-5 个性化推荐算法流程图

个性化推荐算法的几个重要流程，包括对数据的多次清洗处理、算法的择取，画像信息和结果的存储，向外部提供接口。这些步骤在不同的业务中，采用的策略流程稍微会有所不同，不过基本都是大同小异。

### 4.1.3 UI系统

关于UI的展现，呈现的方式不同，给用户的体验也会不同。针对电商平台，在购买物品的时候，为了给用户良好的用户体验，会在用户购买的时候提供一些针对该用户的特色推荐，而这些推荐，会列出原因，让用户体验到系统推荐的透明度，如：你可能喜欢这些产品、猜你喜欢等。

虽然UI系统并不是本论文阐述的重点，但是作为一个项目产品，UI如何展现有时候会决定一个产品成败，因此在这边特别的列出来做一个简要的说明。

图4-6是京东平台推荐的UI展现。通过“猜你喜欢”的模块来进行个性化的推荐，这里面的推荐确实存在我最近关注的手机和VR眼镜，整体上的效果还是不错的。



图4-6 UI需求示例图

## 4.2本系统的重点需求

上面已经很好的阐述了关于本系统的几个大模块需求，下面再重点的讨论一下本系统需要注意到的重点需求问题。因为本论文实现的是实时的推荐系统，所以下面会讨论本系统中重点关注和需要解决的点。

### 4.2.1 实时性问题

对于本系统，实时性是至关重要的一个点。也是本系统需要解决的难点。为了实现实时性，本系统引入了Storm框架进行实时的计算，但是在数据量巨大的情况下，Storm的实时计算也很难快速得到我们需要的结果。因为即使采用常用的协同过滤算法，通过公式得到用户之间的关系矩阵、物品之间的关系矩阵。如果维度太大，则需要进行矩阵分解，这些计算都是比较复杂耗时的。一旦数据量太过庞大，要在短时间内得出推荐结果，基本是天方夜谭。

那么如何解决这种情况。本系统采用将复杂的计算放到离线系统进行计算，通过采用协同过滤算法离线批处理的计算用户相似度、物品相似度等，并这部分的内容当做画像信息进行存储，在Storm上利用画像数据进行一个增量计算得出最后的推荐结果。

### 4.2.2 冷启动问题

推荐系统中都需要面对关于冷启动的问题，冷启动的概念是一个新的用户，在系统中没有任何关于他的信息，那么这时候该用户找不到任何和他有关联的数据，这样就会导致系统无法给他推荐物品。同时由于被推荐内容也会不断的产生和更新，那么又如何将这些新的内容信息推荐给用户，也是一大问题。所以冷启动问题可以归纳为两点：新用户的推荐和新内容的推荐。

关于新用户的推荐，我们在使用任何平台都需要进行注册，才能拥有属于自己的账户。例如：我们在网易音乐上面注册一个用户，那么网易APP端会展现一个页面，上面会列出好几种音乐风格（古典、流行、轻音乐等），让你选择音乐爱好，这本身就是一种规避冷启动的方法，可以很直接的得到用户的喜爱的歌曲类型。当然有些用户会不喜欢去做这些调研，会直接忽视，点击跳过。那么最简单的方式就是采用热门推荐，这是一种最简单的处理方式，把最热门的信息推荐给用户也会取得意想不到的效果。

关于新内容的推荐，即这个内容才刚产生而且不属于那种特别爆炸的新闻（如果是比较爆炸的信息，会直接被人工推荐直接置顶），是否要将该新闻推荐给用户。这部分就需要特殊处理，可以通过时间的阈值来进行控制传播。

### 4.2.3 时效性和多样性问题

时效性主要是针对和时间有强关联的内容。例如：新闻内容、时事内容等。可想而知，如果在用户的推荐首页出现一条关于去年的新闻消息，如：2016年NBA冠军是骑士的内容。这样的推荐结果会让用户觉得莫名其妙，让用户对该推荐产生失望。所以，关于时效性的内容推荐问题需要单独处理，主要是根据“时间”来做影响因子。

多样性的概念本质就是得知用户的多种兴趣爱好，推荐结果可以包含着多种兴趣爱好，而不只是推荐一种类型，这样就不会显的单一。其实这个点很好理解也很好解决，就是在分析用户的行为的时候，可以得到多种的兴趣特征，那么推荐的时候可以加入多个用户的兴趣特征（如：该用户爱好体育，也喜欢电玩。），那么可以根据两个特征来计算来得到用户会被推荐的信息，这样的策略就可以丰富推荐列表。一般推荐的数目可以根据用户和相关度来度量，例如：用户的标签有t1、t2、t3、t4， 对应的相关度为w1、w2、w3、w4，总的推荐数目为TotalNum，则：List(u)=T1[TotalNum \* w1] + T2[TotalNum \* w2] + T3[TotalNum \* w3] +T4[TotalNum \* w4]。

## 4.3 本章总结

本章主要是描述推荐系统的需求，包括实时性、满意性、惊喜度、多样性、推荐透明和新颖性。具体的阐述了系统的整体上的需求，对需求的每个部分进行解析。同时突出了本系统中要实现的实时推荐系统里面实时行的重要性，当然本文主要解决了系统里面的实时性和冷启动问题，另外两种问题只做讨论。

# 第五章 系统的设计与实现

本章主要是阐述推荐系统的设计和实现，主要包括系统的整体框架，及在框架的基础上系统模块的的设计和模块和模块之间的衔接，最终通过简单的例子阐述系统采用的基于物品的协同过滤算法流程及原理。

## 5.1 整体框架

 图5-1实时推荐系统的总体结构示意图

本系统的整体架构分为如下几个大部分：

（1）日志采集系统：如图5-1所示，该模块系统需要实时的收集用户大量的操作信息，如：收藏物品信息、购买物品信息等，系统将这些数据收集之后进行存储。存储之后的数据流向两个方向，一个方向是HDFS，提供给离线计算平台进行算法处理。另一个方向则是流向缓冲模块，提供给流式计算平台进行实时的推荐。

（2）推荐系统：作为核心部分，推荐系统主要包括离线平台和在线平台，离线平台主要是将大量数据进行离线的计算，该模块被设计为日处理历史的数据，并提取离线特性，即物品的相似度信息。在流式计算平台，主要计算实时的数据流，获取用户时刻的操作，通过在线引擎对物品的相似度矩阵进行更新，再实时计算得到最后的推荐结果。这种设计是比较符合实时推荐的，由于互联网信息的数据量的庞大，用户历史数据信息需要一个离线平台进行处理，计算出基本的用户模型特征。之后短时间内出现的数据，则可以通过一个流式计算快速处理，更新模型，进行最后的推荐。

（3）输出展示系统：该系统模块，也算是比较重要的模块，该系统最后通过开发的一个APP来进行展示，给予用户更加直接的体验。

本论文主要的重点在于日志采集系统和流式计算平台的设计，如图5-2简述了流式计算平台的流程图，这个流程主要是对数据进行采集，然后将采集数据传输到数据接入模块，从而起到缓冲的作用。将传输的数据存储起来等待流式计算模块来拉取数据，流式计算模块实时处理信息，处理完之后再从上个模块拉取，实现即拉即用的模式，最后将数据结果存储并暴露接口供外部调用。



图5-2流式计算流程图

如图5-3具体的阐述了整个系统的详细的数据流向过程，我们可以从该示意图将系统分为为五个模块：

（1）数据采集模块：本模块采用了Flume框架进行开发，图中可以看到日志采集模块从各个不同的平台上通过多个Agent采集实时产生的数据，接着会将数据传输给Collector，最后Flume的Sink一个是HDFS，另一个则是下一个模块Kafka。

（2）数据接入模块：本模块采用Kafka框架，图中所示，Kafka的Produce是Flume的Sink，通过编写Flume的配置文件，可以很容易实现两者之间的结合。实现该模块的目的主要起到一个数据缓冲的作用，当Flume采集的数据量太大导致Storm实时计算模块无法及时处理，则会产生数据堵塞。因为Kafka采用Push/Pull的模式，即采集的消息是推入进去的，而在下一个的流式计算模块采用拉取数据的模式，流式计算模块可以根据自己的计算情况，把目前可以承载的数据拉取过来进行计算。这种方式可以很好得避免出现由于处理不及时导致的堵塞的情况，而且采集信息会存储在Kafka的Broken中，数据会一直存储7天（默认7天，也可以自定义），保证数据不会丢失。

（3）离线计算模块：本模块是基于Hadoop平台上进行计算，可以得到物品的相似度和一个推荐列表。为了实现实时的推荐，同时也要保证推荐的准确性，系统设计就将存储在HDFS的数据采用线下的基于物品的协同过滤算法进行处理，根据数据量该过程设计为日处理。

（4） 流式计算模块：本模块采用Storm来实现，是实现实时推荐的关键，这边处理实时得到的数据，同时结合离线模块提供的画像信息，对物品的相似度进行增量更新，最后加入时间因素，对推荐列表结果进行增量更新，从而实现实时的推荐效果。

（5）数据存储和输出模块：对计算过程得到的数据采用Redis存储，为方便外部接口的调用将最后得到的推荐列表结果存储到Mysql持久化数据库中。

图5-3实时推荐系统的数据流向示意图

## 5.2 日志采集模块

在该系统上，用户通过点击和浏览会产生一定的行为。如：浏览商品、点击收藏、购买产品等。这些行为都会产生对应的数据保存起来。

### 5.2.1 日志数据类型

日志数据主要包含这几类的日志内容：用户浏览的信息记录，用户检索信息记录、用户收藏信息记录，用户购买信息记录。下述列出应用后台的表设计：

（1）用户浏览的信息日志是在用户点击产品时候产生，日志数据字段包含：用户ID信息、点击的商品ID、商品类别、浏览时间。

表5-1用户浏览信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| UserID | Int | 20 | 是 | 用户的ID |
| ItemID | Int | 20 | 否 | 商品的ID |
| ItemType | Int | 20 | 否 | 商品类别 |
| Time | String | 30 | 否 | 浏览时间 |

（2）用户检索信息日志是在用户输入检索关键字点击搜索的时候产生，日志数据字段包含：用户ID、检索的关键词，检索到商品类别ID子串、检索时间。

表5-2 用户检索信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| UserID | Int | 20 | 是 | 用户的ID |
| SearchKey | String | 100 | 否 | 检索的关键词 |
| SearchInfo | String | 1000 | 否 | 检索到商品类别ID子串，ID之间用空格隔开。 |
| SearchTime | String | 30 | 否 | 检索时间 |

（3）用户收藏信息日志是在对商品进行点击收藏操作的时候产生，日志的数据字段包含：用户ID、收藏的物品ID，收藏物品的时间。

表5- 3 用户收藏信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| UserID | Int | 20 | 是 | 用户的ID |
| CollectID | Int | 20 | 否 | 收藏物品的ID |
| CollectTime | String | 30 | 否 | 收藏时间 |

（4）用户购买记录信息日志是在用户点击购买操作的时候产生，日志数据字段包含：用户ID、购买的物品ID、购买物品的时间。

表5- 4用户购买记录信息表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| UserID | Int | 20 | 是 | 用户的ID |
| ConsumeID | Int | 20 | 否 | 购买物品的ID |
| ConsumeTime | String | 30 | 否 | 购买时间 |

### 5.2.2 日志采集系统框架



图5-4 日志采集系统框架示意图

系统平台上的数据通过日志采集系统来收集，如图5-4，日志采集系统采用Flume来构建，对多个Web Agent进行监听，将产生的日志数据源发送给Collector，Collector会把全部的数据都加载到存储端（Storage），存储端会将数据一份保存在HDFS，另外一份则会发送给Kafka。而HDFS部分主要的数据供离线平台使用，而发送给Kafka的则是供实时处理的流式平台的数据。

（1）Agent：收集平台上实时产生的数据并推送给Flume里面的Collector模块，这里可以支持自己定制来满足要求，当然本文只是简单得利用了该功能将数据收集。

（2）Collector：该模块主要进行存储和转发，保证数据不轻易丢失，负责将推送过来的数据发送给Kafka模块和HDFS进行存储。从图中可以看到，两部分数据流向不同平台。

（3）Master：为了保证在分布式的情况下，可以对组件进行管理，主要是针对出现故障的时候，不让数据丢失，原则上可以设置多个Master，为简化系统的开发本系统只设置了一个。

（4）ZooKeeper：维护上述几个组件，保持数据一致性，在数据发生变化时候，及时通知master。

### 5.2.3 日志采集系统连接器

这部分主要是配置Flume的环境变量同时采用Flume提供的组件进行收集数据。 （1）配置环境变量

Flume该框架已经实现了很多功能，开发过程只需要根据系统的需求来进行配置，配置主要包括了数据源（Source）、存储管道（Channels）和数据目的地（Sink）。配置文件flume-conf.properties是整个工作的核心，是满足个性化系统定制的支撑点。

数据源包括的配置方案有exec、avro、netcat、systcp、sysudp、seq等。

存储管道的目的是避免数据可能发生丢失的情况，让数据高可靠，一般设置的模式是基于内存和硬盘的。对应的配置类型memory（内存）、file（文件）、spillablememory （文件内存结合）等。当然也可以自己定义。一般为了保证足够的安全性和稳定性，最好都是基于硬盘存储，这样可以支持断点续传。

数据目的地（sink），配置方案有Logger（日志）、HDFS、HBase、AsyncHBase、MorphlineSolr、Custom ElasticSearch等。

下列表格列出不同的采集方式的配置方案：

表5-5 Flume配置方案表

|  |  |  |
| --- | --- | --- |
| 采集方式 | 配置文件 | 描述 |
| Exec | a.sources = r  a.sinks = k  a.channels = c  a.sources.r.type = exec  a.sources.r.channels = c  a.sources.r.command=tail-F /home/hadoop/flume-1.6.0-bin/log\_exec\_tail  a.sinks.k.type = logger  a.sinks.k.channel=ca.channels.c.type =memory  a.channels.c.capacity = 1000  a.channels.c.transactionCapacity = 100 | a.sources.r.type= exec  采集方式是通过命令执行（从Unix读取tail -F）  sinks.k.channel  =ca.channels.c.type = memory  上述表示将数据放在内存，可以加快速度，减少IO时间，但是也容易导致内存溢出。 |
| avro | a.channels = c  a.sources = r  a.sinks = k  a.channels.c.type = memory  a.sources.r.channels = c  a.sources.r.type = avro  a.sources.r.bind = 192.168.0.23  a.sources.r.port = 8080  a.sinks.k.channel = c  a.sinks.k.type = logger  a.channels.c.transactionCapacity = 100 | 使用Avro Netty  RPC协议，如果采用avro source可以提高线程数目，从而会提高处理的速度。 |
| Spool | a.sources = r  a.sinks = k  a.channels = c  a.sources.r.type = spool dir  a.sources.r.channels = c  a.sources.r.spoolDir= /home/hadoop/flume-1.6.0-bin/logs  a.sources.r.fileHeader = true  a.sinks.k.type = logger  a.sinks.k.channel = c  a.channels.c.type = memory  a.channels.c.capacity = 1000  a.channels.c.transactionCapacity = 100 | Source spool，适合静态文件，即文件本身不是动态变化的。 |
| Syslogtcp | a.sources = r  a.sinks = k  a.channels = c  a.sources.r.type = syslogtcp  a.sources.r.port = 9999  a.sources.r.host = localhost  a.sources.r.channels = c  a.sinks.k.type = logger  a.sinks.k.channel = c  a.channels.c.type = memory  a.channels.c.capacity = 1000  a.channels.c.transactionCapacity = 100 | Syslog 支持TCP和UDP等2种模式，而这边采用的tcp的模式。 |
| JSONHandler | a.sources = r  a.sinks = k  a.channels = c  a.sources.r.type= org.apache.flume.source.http.HTTPSource  a.sources.r.port = 8888  a.sources.r.channels = c  a.sinks.k.type = logger  a.sinks.k.channel = c  a.channels.c.type = memory  a.channels.c.capacity = 1000  a.channels.c.transactionCapacity = 100 | JSONHandler采集模式是通过Http。 |

从上述的表格中可以看到各种不同的采集方式的配置方案都大不相同，而我们这边的Source采用JSONHandler和Spool的方式，直接从Http进行响应。而Sink主要有两步，第一步数据流向Kafka，另一步数据流向HDFS进行存储。因此本系统需要设置两个数据的目的地（sink），一个是kafkaSink，另一个是hdfsSink。配置代码如下所示：

|  |
| --- |
| a.sources = r  a.sinks = kl k2  a.channels = c1 c2  a.sources.r.type=org.apache.flume.source.http.HTTPSource  a.sources.r.port = 8888  a.sources.r.channels = c  a.sinks.k1.channel = c1  a.sinks.k2.channel = c2  a.channels.c.type = memory  a.channels.c.capacity = 1000 |

（2）Flume采集的数据流向



图5-5 日志采集数据序列图

图5-5表示了Flume数据采集的序列图，从用户通过操作外部应用产生信息数据，这些信息会通过Https或者File的方式被Flume的Source进行采集，这里面采用JSONHandler的Http方式采集。为了保证数据不丢失，数据从source开始，采用了事务的机制。开启事务之后，传输的数据都会被封装为一个个事件(即：byte数组)流向channel，最后再将数据推送给Kafka进行后期处理以及HDFS进行存储。下面列出如何将数据message从Source放到Channel的代码过程：

|  |
| --- |
| ChannelProcessor chan = source.getChannelProcessor();  Event en = EventBuilder.withBody(message.getBytes());  channel.processEvent(en); |

发送到Sink（即Kafka组件）代码如下：

|  |
| --- |
| Channel chan = getChannel();  Transaction tran = chan.getTransaction();  tran.begin();//开始事务  List<KeyedMessage<byte[], byte[]>>infolist= Lists.newLinkedList();  for (int i = 0; i < infoSize; i++) {  Event en = chan.take();//获得一条日志信息  byte[] ens=event.getBody();  infolist.add(new KeyedMessage<byte[], byte[]>(topic, ens));  }  producer.send(infolist);//把信息发送给kafka  tran.commit();//提交事务 |

## 5.3 数据接入模块

### 5.3.1 数据接入框架

该系统上应用了Kafka框架来接收来至Flume采集的信息，信息数据主要是从Flume获取。而事实上Flume的Sink就是配置为Kafka，Kafka在系统中发挥着相当重要的作用，它主要包括以下的几个方面作用：



图5-6数据接入模式架构示意图

（1）将各个不同的日志进行汇总，让日志得到统一的处理。如图5-6所示从Flume Sink将采集的数据推送给Kafka，这些数据都会被集中存储到Broken节点中。所有后面模块需要处理数据只需要从Kafka的Broken中拉取出来，加入该模块方便后期对日志的处理。

（2）该模块可以减小后面流式平台中Storm的处理压力。在系统中Storm的Spout作为一个Kafka的消费模块，当数据量很大，利用传统的推送数据的方式一直发送数据过来，很容易遇到数据堵塞，让系统陷入瓶颈。而Kafka中里面的Consume（消费者）即：系统的Storm模块，采用从Broken拉取数据的模式。这样的话Storm就可以很恰当的利用自己的空闲时间去从Kafka的Broken中拉取数据。同时Broker数据默认可以缓存7天，这边直接采用默认方式，当然也可根据具体的需要进行配置。

（3）该模块设计可以很好的让数据不会丢失掉。由于Kafka本身就具有比较好的容错性的，因此当一个节点发生问题，另一个节点可以保证正常的工作，从而可避免出现各种问题。

### 5.3.2 数据接入连接器

从图5-7中可以看出Flume集群最后的数据目的地是Kafka，本论文的系统数据一部分是给HDFS，另一部分则是同时发给Kafka。那么Kafka需要接受来自Flume的数据，这个数据可以表示为Kafka的Produce，所以Flume的Sink和Kafka的Produce是连接重合的，那么系统在配置变量同时编写对应的Topic将数据合理的传输到Kafka并让Kafka消费掉，即再发送给下一个Storm框架进行实时处理。



图5-7 Kafka生产者示意图

（1）配置Flume环境变量

Flume框架里面的flume-conf.properties可以配置，将Flume里面的Sink输出设置为Kafka的Produce，一般配置就是配置flume-ng里面client的source、channels、sink。下面给出一个配置例子，里面的sink的type设置为提供的Kafkasink，如图5-8的配置文件示例。

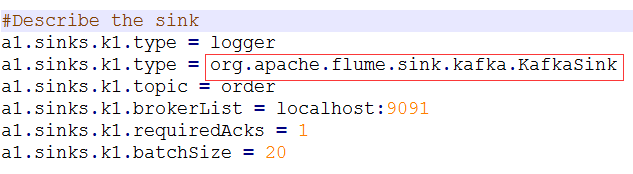


图5-8 Flume的配置示例图

（2）Kafka的连接器

系统中关于Kafka的连接器，需要编写Kafka对应的Producer，Producer本质上就是一个生产者，代表消息的生产者，也是和Flume里的Sink对应。Producer通过配置为Flume的Sink，将该消息内容推送给Broken里面进行缓存，该缓存时间使用默认为7天的配置值，并且等待着Kafka的Consumer来拉取数据，这边的消费者配置为Storm的Spout，关于生产者推送消息内容给Broken的编码如下：

|  |
| --- |
| public Producer extends Thread {  public void sendData() {  Properties props =  PropertiesParser.getProperties(PropertiesSettings.PRODUCER\_FILE\_NAME);  ProducerConfig con = newProducerConfig(props);  Producer<String, String> pd= new Producer<String, String>(con);  KeyedMessage<String, String> msginfo;  String ip = "192.168.1.11";  StringBuffer sb = newStringBuffer();  …..  pro.send(msginfo);  }  public void run() {  sendData();  }  } |

下面简要列出关于消费者从Broken结构中拉取数据的代码，主要是通过监听然后循环拉取数据。

|  |
| --- |
| public class ConsumerCluster extends Thread {  private ConsumerConnector consumer;  private String topic;  public Consumer(String topic) {  consumer =kafka.consumer.Consumer.createJavaConsumerConnector(  createConsumerConfig());  this.topic = topic;  }  private static ConsumerConfigcreateConsumerConfig() {  Properties props = new Properties();  props.put("zookeeper.connect", KafkaProperties.Connect);  return new ConsumerConfig(props);  }  public void run() {  Map<String, Integer> TopicMap = new HashMap<String, Integer>();  TopicMap.put(topic, OrderInfo);  Map<String,List<KafkaStream<byte[], byte[]>>>  CountMap =consumer.createMessageStreams(TopicMap);  KafkaStream<byte[], byte[]> st = CountMap.get(topic).get(0);  ConsumerIterator<byte[], byte[]>it = stream.iterator();  while (it.hasNext()) {  //获取数据…  }  } |

## 5.4 离线计算平台模块

离线平台主要是基于Hadoop对数据进行批处理，由于数据量的庞大，如果把协同过滤算法放在流式平台上，必然会加重流式计算的负担，无法实现实时的效果。所以，构建离线平台的目的是将之前的积累的数据进行线下处理。

### 5.4.1 基于物品的协同过滤算法原理

该小节具体阐述本系统的离线模块中采用的基于物品的协同过滤算法的实现过程。首先关于数据集的字段如下：

（1）UserId：用户的ID；

（2）ItemId：物品的ID；

（3）Score：用户对该物品的评分；

下列列出Item-CF的实现流程：

（1）先计算获得物品的同向矩阵，即计算两个物品同时出现的次数，目的为了得到物品之间的关系度。

数据格式：ItemId1:ItemId2 Num(出现次数)

（2）根据用户对物品进行一个评分（评分机制可以自己建立），得到一个评分矩阵B，这个评分矩阵的每一项就是每个用户对每个物品的评价。

数据格式：ItemId  user\_id:Score

（3）最后将第一步得到的物品同向矩阵乘以第二步的评分矩阵，就可以得到最后的推荐列表。

数据格式：userId  ItemId,Scores（推荐分值）

（4）得到推荐列表需要将用户之前已经评分过的物品（可以理解为购买过或者收藏过该物品）过滤掉，保证推荐的新颖性。

（5）最后根据推荐的分值按照倒排的取得Top（N）。

### 5.4.2 算法实现过程

根据上述的基于物品的协同过滤算法原理，通过搭建Hadoop平台之后，再编写MapReduce来实现该算法，该推荐算法实现过程如图5-9所示。



图5-9 推荐算法计算过程流程图

（1）从HDFS读取日志信息数据，然后对其进行处理。如：数据集里面没有评分的概念，那么系统进行构造评分机制，当用户点击浏览该物品评分设置1分，点击收藏则设置为3分，购买设置为5分（针对不同的情况进行评分机制的构建）。

（2）经过上述的机制处理之后，就可以获得 UserId,ItemId,Score 的数据集如表5-6所示。

表5-6 数据源表

|  |
| --- |
| 数据(UserId,ItemId,Score） |
| 1,11,5.0 |
| 1,12,3.0 |
| 1,13,2.5 |
| 2,11,2.0 |
| 2,12,2.5 |
| …… |

（3）如图5-9，编写MapReduce的计算程序，第一步读取处理得到的数据，按照用户ID分组，输出的格式如表5-7所示：

表5-7 物品评分表

|  |  |
| --- | --- |
| 用户ID | 数据（ItemID:Score） |
| 1 | 13:2.5,11:5.0,12:3.0 |
| 2 | 11:2.0,12:2.5,13:5.0,14:2.0 |
| 3 | 17:5.0,11:2.0,14:4.0,15:4.5 |
| 4 | 13:3.0,16:4.0,14:4.5,11:5.0 |
| 5 | 11:4.0,12:3.0,13:2.0,14:4.0,15:3.5,16:4.0 |
| …… | …… |

（4）通过上述的信息可以计算两个物品同时出现的次数，得到结果如表5-8所示：

表5-8 物品出现次数表

|  |
| --- |
| 数据(ItemId:ItemId Num） |
| 11:11 5 |
| 11:12 3 |
| 11:13 4 |
| 11:14 4 |
| 11:15 2 |
| …… |

（5）根据物之间出现的次数表，通过mapper1计算得到物品同现矩阵，结果如表5-9所示：

表5-9 物品同现矩阵表

|  |
| --- |
| 数据(ItemId: ItemId Num） |
| 11:11 5 |
| 11:12 3 |
| 11:13 4 |
| 11:14 4 |
| 11:15 2 |
| 12:11 3 |
| …… |

根据上述的信息可以计算得到用户评分矩阵和物品的同现矩阵，通过mapper2生成用户的评分矩阵，结果如下表5-10所示：

表 5-10 物品-用户评分矩阵表

|  |
| --- |
| 数据(ItemId UserId:Score） |
| 11 2:2.0 |
| 11 5:4.0 |
| 11 4:5.0 |
| 11 3:2.0 |
| 11 1:5.0 |
| 12 2:2.5 |
| …… |

（6）根据上面得到的两个矩阵，可以做矩阵乘法，那么就会得到推荐列表。公式是：物品同现矩阵 \* 用户的评分矩阵 = 推荐列表。推荐结果如表5-11所示：

表 5-11 推荐结果表

|  |  |
| --- | --- |
| ID | 推荐结果（ItemID,Scores） |
| 1 | 17,10.5 |
| 1 | 16,18.0 |
| 1 | 14,33.5 |
| 2 | 17,11.5 |
| 2 | 16,20.5 |
| …… | …… |

## 5.5 流式计算平台模块

该模块在实现的推荐系统中扮演一个非常重要的角色，前两节搭建日志采集系统和数据接入模块就是为这个模块服务。该模块需要处理实时的数据，主要是通过Storm框架来实现。利用Storm的性能并且结合推荐算法来实现实时推荐的效果。

### 5.5.1 Kafka与Storm的连接器

上述已经介绍了关于Kafka和Storm的原理，以及在该系统中的关系。Kafka将收集的数据日志汇集起来，等待Storm的拉取。在这个拉取的过程主要是将Kafka的Consumer和Storm的Spout两个模块连接在一起，这种方式很好的保证了两个模块之间高效的数据传输。

图5-10 Storm流式计算框架示意图

如图5-10所示，当数据存储在Kafka的Broken中，我们就可以采用Storm来主动拉取消费。由于Kafka和Storm具有良好的兼容性，所以可以采用Kafka Spout从Kafka中拉取数据，而数据源Spout需要处理特定的接口类。然后通过实现Bolt来消费从Spout发送出来的数据流，并根据具体的需求来实现对应的逻辑。如果是比较复杂的数据，可以定义多个Bolt来帮助处理，最后通过Kafka Spout和Kafka Bolt来生成Topology对象交给Storm来处理。

（1）关于Kafka Spout的实现，它们主要是处理用户的购物订单操作，当然还包括各种其他的操作信息，通过ordersAnalysisTopology来解析kafkaSpout，下面简单的列出订单事件分析的代码。

|  |
| --- |
| public class ordersAnalysisTopology {  private static String topicInfoName = "topicInfo";  private static String StormRoot = "/Storm/"+topicInfoName;  public static void KafkaSpolt(String topic) {  BrokerHosts hst = new ZkHosts("Storm1:3219,Storm2:3219,Storm3:3219");  SpoutConfig Confi = new SpoutConfig(hst,topicInfoName, StormRoot,  UUID.randomUUID().toString());  Config.scheme = new SchemeAsMultiScheme(new StringScheme());  KafkaSpout spout = new KafkaSpout(Config);  TopologyBuilder build = new TopologyBuilder();  build.setSpout("kafkaSpout",Spout);  build.setBolt("OrdersBolt", new OrdersAnalysisBolt(), 2)  .shuffleGrouping("kafkaSpout");  ….  } |

（2）关于Kafka Bolt 实现，主要是对业务进行转化，然后对消息进行处理更新。执行Storm里面的消息Tuble，将消息进行转化为可处理的信息再交给下一个Bolt对物品相似度进行更新，下面简单列出订单信息处理的代码。

|  |
| --- |
| public class OrderAnalysisBolt extends BaseRichBolt {  private OutputCollector collector;  JedisPool pool;  public void execute(Tuple tuple) {  String orderInfo = tuple.getString(0);  ordersBean order = handler.getOrdersBean(orderInfo);  Jedis dis = pool.getResource();  dis.zincrby("orderAna:Score", order.getID(), order.getScore());  }  public void prepare(Map arg0, TopologyContext arg1, OutputCollector collector) {  this.\_collector = collector;  this.pool = new JedisPool(new JedisPoolConfig(), "Storm",3219,2 \* 50000,"123");  }  } |

### 5.5.2 实时推荐计算

实时推荐模块是在Storm的框架基础上进行计算，主要是从Kafka获得实时的数据，然后将离线平台计算得到的数据（其中包括：物品的相似度信息、用户对物品评分信息以及离线的推荐列表信息），全部存储在Redis的缓存中，以供流式平台的调用。Storm流式平台主要通过时间窗口和热搜物品来进行调节。

（1）增量更新物品相似度

本系统计算物品相似度的过程是放在离线平台上进行，当面对庞大的数据量想要通过实时计算并快速得出推荐结果是不现实的。所以将重量的计算模块放在离线平台进行周期的批量处理，这个周期本系统规定为一天。同时流式计算平台实时的获得近期用户的行为，对用户行为兴趣程度进行分析。系统主要是对物品相似度进行更新计算，因为该计算公式比较简单，所有该计算规模比较小，计算更新速度比较快，采用的更新公式为(5-1)。

|  |  |  |
| --- | --- | --- |
|  |  | (5-1) |

针对上述的公式进行分析，假设用户M 对物品X有评分，那么相对而言物品相似度如何更新：

a. 如果用户M对物品x有过评分，即对该物品有产生行为信息，那么我们不需要更新任何物品相似度。

b. 如果用户M首次对物品X产生事件行为，对其进行了评分。那么相对的就会增加一。而通过公式中的，如果出现用户M对物品y也有过事件行为，则需要增加一；如果用户对物品y没有过事件行为，则不变。

（2）实时推荐方法

根据上述得到的物品相似度计算用户对物品的评分，系统中采用的评分计算公式如（5-2）所示：

|  |  |  |
| --- | --- | --- |
|  |  | (5-2) |

公式里面R(u)表示用户已经评分的集合，S(j,k)表示和物品最相似的K个物品，而表示了物品j和物品i的相似度，表示用户u对物品i的评分。

实时推荐的方法就是要根据用户最近期的产生的行为来进行推荐。人的兴趣随着时间推移会有一些变化，而近期的行为最能反映前用户的兴趣爱好。然后根据更新的物品相似度进行预测。方法主要是根据时间窗口，来得到用户近期对物品的评分。 当过了一个时间阈值过后的数据就会被过滤掉，这样的目的就是为了突出近期的用户对物品的兴趣度的影响。

对于新的用户，即冷启动问题。在离线中无法获得用户对物品的评分和推荐列表。则系统会在一开始将热门物品进行推荐，随着用户近期的浏览慢慢取代热门的物品，最终根据近期评分规则来进行推荐。

（3）实时推荐实现流程

由图5-11可以得知，在Storm上实时推荐计算的流程如下：

第一步，从Kafka得到数据信息，KafkaSpout作为一个数据源，订阅消息队列中的信息，并将消息发送给OrderAnalysisBolt进行处理。

第二步，OrderAnalysisBolt接收到数据源关于用户的事件消息之后，对消息进行一个分类评分，评分主要根据近期事件进行一个增值评分。

第三步，如果在离线平台推荐列表中没有关于该用户的推荐评分信息，那么该用户属于新用户，属于冷启动问题。则进入到ColdDealBolt处理，这个Bolt会将热门的物品列表进行推荐。

第四步，利用UpdateItemBolt对物品相似度进行一个增量更新。根据物品的ID从Redis缓存中取得离线计算的物品相似度矩阵，然后增量更新物品相似度。

第五步，得到最新的增量的相似度矩阵之后，通过RateBolt进行物品的评分计算。

第六步，对物品进行一个RateBolt降序排序，选取K个推荐物品，得到推荐列表。



图5-11 实时推荐实现流程图

## 5.6 数据存储和展示模块

### 5.6.1 数据存储和输出

数据存储部分主要存储在计算过程中产生的数据包括物品之间的相似度矩阵、物品评分矩阵以及最后推荐的列表。Redis是一个基于键值对且具有持久化的内存存储系统，可以满足本系统中对各种数据的存储和使用。为加快运算将数据存储在Redis中。

输出部分主要将推荐的结果数据保存在Mysql数据库中。通过持久化操作，为下次计算推荐表进行参考。在实时的基础上，短时间内进行更新，那么页面上可以实时的推荐给用户需要的产品。

### 5.6.2 UI展示

在完成推荐系统构建之后，本人主要开发了一个小型的APP，通过APP将推荐结果展现出来。关于该APP的开发，是采用开源代码ECMobil，进行二次的轻量级别的APP，由于本部分不是重点，所以就不多加阐述，APP的UI展示如图5-12所示。

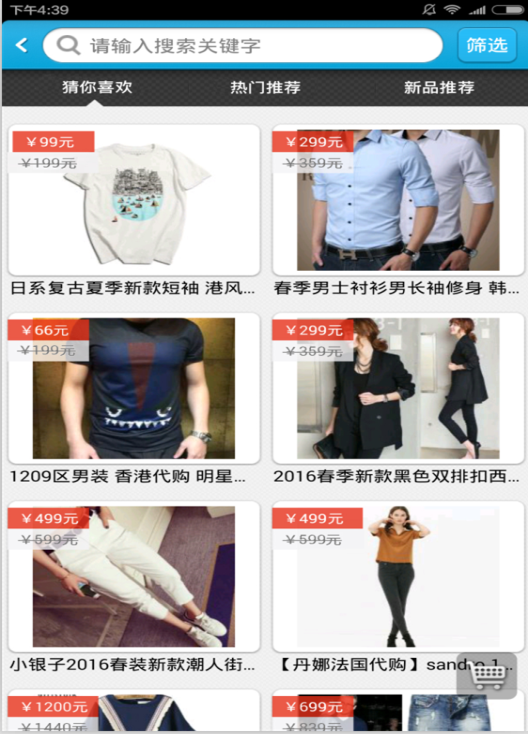
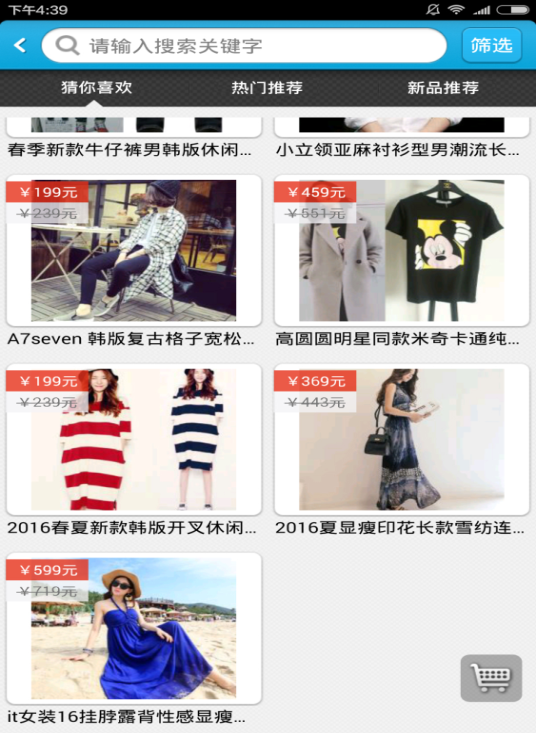
 

图5-12 小型电商APP界面展示图

图5-12中里面通过三种方式进行推荐，分别是猜你喜欢、热门推荐、新品推荐，物品的推荐主要是在“猜你喜欢”模块。当然这只是一种简单的表示。由此可知推荐系统的搭建是很复杂的，而最终只是简单的几个页面的展示。但是应用信息的展示需要考虑在哪里展示，如何展示，才能给用户一种新奇感，并且促动用户消费。

## 5.7 实验与测试

### 5.7.1 评测标准

针对推荐结果的评测，主要根据推荐结果的准确率和召回率进行评价。

推荐的准确率表示物品和数据集中用户喜爱的物品的取交集占物品的比例，准确率的公式如(5-3)所示：

|  |  |  |
| --- | --- | --- |
|  |  | (5-3) |

推荐的召回率表示物品和数据集中用户喜爱的物品的取交集占数据集中用户爱好的物品的总数的比例，公式如(5-4)所示：

|  |  |  |
| --- | --- | --- |
|  |  | (5-4) |

### 5.7.2 实验设计

（1）数据集

由于平台上面的数据量不够大，所以实验数据采用的是阿里巴巴大数据竞赛“天池”的淘宝穿衣搭配大赛的数据。这里面数据包含了十万条的商品信息，和一千多万条的用户购买记录信息。由于没有物品的评分，我采用自定义的评分机制，通过购买记录对商品进行一个评分，得到可以处理的信息数据。

（2）实验环境

实验中的算法使用Java语言实现，算法的开发工具是eclipse。使用[Storm 0.10.2 Released](http://storm.apache.org/2016/09/14/storm0102-released.html)版本构建分布式计算环境，Storm集群由四台虚拟机模拟Storm的集群环境，每台计算机都分配得到单核的CPU进行试验，具体的配置情况如表5-12所示：

表5-12集群环境配置表

|  |  |  |
| --- | --- | --- |
| 主机名 | 配置参数 | 系统 |
| Master | GPU：i7 2.3GHz 内存 8G | Ubuntu 15.04 |
| Slave1 | GPU：i7 2.3GHz 内存 8G | Ubuntu 15.04 |
| Slave2 | GPU：i7 2.3GHz 内存 8G | Ubuntu 15.04 |
| Slave3 | GPU：i7 2.3GHz 内存 8G | Ubuntu 15.04 |

（3）实验内容和结果

实验内容包括两个实验，第一个实验是关于实时推荐算法的准确率和召回率的验证，第二个实验是关于该系统的实时性能的验证，第三个实验是关于不同算法计算时间性能验证。

第一个实验采用的方法如下：

第一步，由于数据中没用用户的评分，因此需要模拟评分，那么按照公式(5-5)，将用户购买物品转化为用户对物品的评分，其中系数x代表用户的购买次数，系数a为评分的上限，通过调节系数b来调节评分变化速率。

|  |  |  |
| --- | --- | --- |
|  |  | (5-5) |

第二步，由于实时的推荐物品需要根据当前的用户的行为来更新物品相似度矩阵，并得到推荐列表，因此需要模拟用户的实时行为。同时从数据集合中按照时间读取用户记录R1，然后选取和这次购买记录时间相近并且晚于R1的数据集R2（该部分就是模拟的用户实时行为），并对R1的数据在离线平台上通过基于物品协同过滤算法进行计算得到离线的物品相似度矩阵和推荐列表，最终根据R2的模拟实时数据进行更新物品相似度矩阵，再计算得到推荐列表，最后计算推荐系统中的召回率和准确率。

通过实验可以得到实验结果，下列通过柱状图来阐明实验得到的结果：



图 5-13 推荐准确率柱状图



图 5-14 推荐召回率柱状图

图5-13和图5-14，分别是根据公式(5-3)和(5-4)得出离线推荐和在线推荐的准确率和召回率的数据。其中离线推荐利用了该系统的离线平台进行计算得出结果，而在线推荐则是在离线平台基础上，通过最近的行为信息（即：上述中模拟的R2的数据集）来对物品进行更新推荐，然后通过公式计算得出结果。从两个图中可以看出随着推荐个数的增加，这两种推荐方法的准确率和召回率就趋于接近。说明实时产生的行为对推荐的效果还是有限的，推荐个数越少则实时的行为对推荐的效果越好。

第二个实验采用的方法如下：

该实验主要测试关于实时推荐系统的实时性能，和第一个实验一样，该实验也需要对数据的评分进行处理。首先从数据集合中按照时间读取用户购买记录R1，这部分作为离线的数据进行处理，从而计算的物品相似度矩阵和推荐列表。选取和这次购买记录时间相近并且晚于R1的数据R（该部分就是模拟的用户实时行为），然后根据通过数据输入R的数据进行计算得到推荐结果。选取不同的R1数据集合的数量进行测试。得出离线计算时间和实时计算时间表结果如下所示：

表 5-13 推荐计算时间表

|  |  |  |
| --- | --- | --- |
| R1（条） | 离线计算时间（s） | 实时计算时间（s） |
| 1000 | 120 | 1.9 |
| 10000 | 445 | 2.3 |
| 100000 | 1239 | 2.5 |

通过表5-13可知，离线计算部分会因为数据的增加，而导致计算时间的增加。而实时推荐因为是在离线的基础上做增量的更新，所以可以在短时间内得出因为用户新的操作行为而得到新的推荐结果。这也很好的证明了，本系统采用离线平台来进行重操作。而在线推荐在此基础上，进行计算量小的更新。由上述的实验结果，可以得出实时推荐结果响应时间都在用户可接受的范围之内。

第三个实验采用的方法如下：

本实验只是简单的利用同样的方式进行离线计算和在线的更新计算，采用不同模型和方法进行一个简易的比较，同时都取得数据记录集10000条进行操作。

表 5-14 不同算法计算时间表

|  |  |  |
| --- | --- | --- |
| 采用的方法 | 离线计算时间（s） | 实时计算时间（s） |
| 系统采用的方法 | 445 | 2.3 |
| User-CF | 610 | 3.4 |
| Logistic模型 | 569 | -- |
| LFM模型 | 538 | -- |
| FM模型 | 579 | -- |

通过表5-14可知，通过简单的比较，不同的算法之间的离线计算时间和实时计算时间都不同，由于比较复杂的算法，本人没有具体的研究这些算法对应的在线更新算法，所以没办法得到实时的计算时间，最后得到的结果是本文采用的方法的计算时间和其它几种算法计算时间的对比。

通过上述的三个实验进行，可以发现在经过实时的计算处理之后，物品推荐的准确率和召回率和基于物品协同过滤算法的离线计算有了略微的提升，当然采用该算法里面最简易的相似度和评分方法，和其他比较复杂的公式方法对比，确实是牺牲掉了一些准确率，目的是为了保证在实时性能上的快速更新，提供更好的用户体验。而且第二个实验和第三个实验结果，说明该算法在实时性能方面能更好的满足用户的体验，从而表明本系统的设计思想有一定的价值意义。

## 5.7 本章小结

本章是整个论文的核心部分，具体的阐述了系统的整体框架，同时通过详细的流程图，来具体的说明整个系统的设计。该部分将系统通过五个模块进行表述以及最后关于算法的进行的实验测试。

第一个模块，日志采集模块，采用Flume框架进行搭建，同时介绍了系统应用端涉及到的一些日志信息的数据格式，然后概述了数据采集框架的流程，最后对Flume进行配置，并将各种配置方案以表格的形式进行展现，通过时序图来说明采集数据流的过程。

第二个模块，数据接入模块，该模块采用Kafka框架，简要说明了利用该框架对整个系统的优势。介绍了该模块和采集系统连接的配置和模块之间的连接器。

第三个模块，离线计算平台模块，该模块是基于Hadoop上进行批处理的计算平台。该小节也具体通过流程图和实验例子来阐述整个算法的计算流程。

第四个模块，流式计算平台模块，该模块是本文的重点，前面的框架搭建和设计都是为这块服务的。流式计算平台采用Storm框架，从接入数据模块中获得实时的流数据，并结合离线提供的物品相似度矩阵和推荐列表，同时对物品相似度和推荐列表进行更新，最后实现实时的推荐。系统中的创新点是如何利用时间因子，来优化评分。同时针对用户冷启动问题提出解决方案。

第五个模块，数据存储和输出模块。该模块只是很简单得介绍了在计算过程需要将这些计算数据缓存在Redis中，为了方便实时调用，增加响应速度，将最终的推荐结果保存在Mysql中，进行持久化存储。

最后介绍了评测标准：准确率和召回率，然后设计实验，实验包括: 数据集的选择（阿里大数据竞赛的数据集）、实验环境搭建、实验内容，最终通过柱状图的方式展现推荐的准确率和召回率，并通过表格阐述了实时推荐消耗的时间和离线消耗时间的比较，通过实验可以表明本系统的实时推荐确实起到一定的作用。

# 第六章 总结与展望

随着“大数据”时代的来临，互联网的信息资源越来越丰富，从而造成信息的过载，导致人们在网上很难容易得到需要的信息。于是，推荐系统的发展对于互联网越发的重要。

本文主要是实现关于电商的实时推荐系统，利用目前比较流行的实时计算框架Storm来实现。前期也尝试过通过流式计算直接实现推荐算法，但是在数据量比较大的情况下，实现的效果并不太理想，所以在后面加入离线平台，把复杂的算法计算放在线下处理，来解决由于计算过于复杂无法及时得到推荐列表的问题，这算是该系统的一个亮点。

该系统包括了系统采集模块、系统接入模块、系统的离线计算、系统流计算模块、系统存储和输出模块。其中，也用简单例子说明算法的计算的过程。由于为了体现实时性，所以该系统上使用比较基础的基于物品的协同过滤算法，目的是为了让线上更新推荐计算不会过于复杂，导致推荐结果输出太慢而影响用户的体验。在文中已经分析了如何通过增量算法更新物品相似度，并利用时间窗口来获取近期的用户行为来进行推荐结果的计算。当然本人也还在尝试利用复合的算法来实现的推荐系统，但是目前效果还是不理想。

由于各种因素的存在，包括时间、环境和本身的能力。本系统还有很大的改进空间。本系统还可以向这几个方向进行探讨：

（1）关于算法方面，可以研究和探讨更适合的算法，不管是复合算法还是新型的算法；本系统只是采用基于物品的协同过滤的算法来实现，结合了在线的更新算法满足推荐效果。但是不同的推荐算法有不同的优势，所以还需要继续试验和探讨。

（2）需要更多的真实商业环境下进行测试工作，目前，这边的数据集，只是从

网上获取的数据用来测试，还是具有片面性。

（3）在离线方面可以利用深度学习，来获取更多有用的信息，以便于在线的推荐。

参考文献

1. 蔡自兴. 人工智能及其应用:研究生用书[M]. 清华大学出版社, 2004.
2. 李勇, 徐振宁, 张维明. Internet个性化信息服务研究综述[J]. 计算机工程与应用, 2002, 38(19):183-188.
3. 项亮. 推荐系统实践[M]. 人民邮电出版社, 2012.
4. Liu F, Tang B, Yuan X, et al. Recommender System in E-commerce[C]// International Conference on E-Business and E-Government. IEEE Computer Society, 2012:700-703.
5. D’Souza S. Apache flume[J]. 2003.
6. Garg N. Apache kafka[M]. Packt Publishing, 2013.
7. Components S, Implementation S. Easy, Real-Time Big Data Analysis Using Storm[J]. Dr Dobbs Journal.
8. Somasundaram N. "Apache Samza - A Stream Processing Framework"[J]. 2014.
9. Akidau T, Balikov A, Bekiro, et al. MillWheel: fault-tolerant stream processing at internet scale[J]. Proceedings of the Vldb Endowment, 2013, 6(11):1033-1044.
10. Neumeyer L, Robbins B, Nair A, et al. S4: Distributed Stream Computing Platform[C]// IEEE International Conference on Data Mining Workshops. IEEE, 2010:170-177.
11. Vavilapalli V K, Murthy A C, Douglas C, et al. Apache Hadoop YARN: yet another resource negotiator[C]// Symposium on Cloud Computing. 2013:5.
12. Shvachko K V. Apache hadoop[J]. Usenix Org.
13. Ghemawat S, Gobioff H, Leung S T. The Google file system[C]// Nineteenth ACM Symposium on Operating Systems Principles. ACM, 2003:29-43.
14. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]// Conference on Symposium on Opearting Systems Design & Implementation. DBLP, 2004:137-150.
15. Miner D, Shook A. MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems[M]. O'Reilly Media, Inc. 2012.
16. Su X, Khoshgoftaar T M. A survey of collaborative filtering techniques[J]. Advances in Artificial Intelligence, 2009, 2009(12):4.
17. Schafer J B, Konstan J A, Riedl J. E-Commerce Recommendation Applications[J]. Data Mining and Knowledge Discovery, 2001, 5(1):115-153.
18. 张振亚, 王进, 程红梅,等. 基于余弦相似度的文本空间索引方法研究[J]. 计算机科学, 2005, 32(9):160-163.
19. Adler J, Parmryd I. Quantifying colocalization by correlation: the Pearson correlation coefficient is superior to the Mander's overlap coefficient.[J]. Cytometry Part A, 2010, 77(8):733.
20. 孟海东, 张玉英, 宋飞燕. 一种基于加权欧氏距离聚类方法的研究[J]. 计算机应用, 2006, 26(s2):179-180.
21. Fligner M A, Verducci J S, Blower P E. A Modification of the Jaccard–Tanimoto Similarity Index for Diverse Selection of Chemical Compounds Using Binary Strings[J]. Technometrics, 2002, 44(2):110-119.
22. 刘庆鹏, 陈明锐. 优化稀疏数据集提高协同过滤推荐系统质量的方法[J]. 计算机应用, 2012, 32(4):1082-1085.
23. Canny J. Collaborative filtering with privacy via factor analysis[C]// International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 2002:238-245.
24. O'Donovan J, Smyth B. Trust in recommender systems[C]// International Conference on Intelligent User Interfaces, January 10-13, 2005, San Diego, California, Usa. DBLP, 2005:167-174.
25. Breese J S, Heckerman D, Kadie C. Empirical analysis of predictive algorithms for collaborative filtering[J]. New Page, 2013, 7(7):43--52.
26. Cannon R K. Scribe: US, US3100346[P]. 1963.
27. 张川, 邓珍荣, 邓星,等. 基于Chukwa的大规模日志智能监测收集方法[J]. 计算机工程与设计, 2014, 35(9):3263-3269.
28. Pazzani M J, Billsus D. Content-based recommendation systems[M]// The adaptive web. Springer-Verlag, 2007:325-341.
29. Cantador I, Bellog, Alejandro N, et al. Content-based recommendation in social tagging systems[C]// ACM Conference on Recommender Systems. ACM, 2010:237-240.
30. Resnick P, Varian H R. Recommender systems. Commun ACM[J]. 1997, 40(3):56–58.
31. 李忠俊, 周启海, 帅青红. 一种基于内容和协同过滤同构化整合的推荐系统模型[J]. 计算机科学, 2009, 36(12):142-145.

致谢

时光荏苒，我的硕士生涯已接进尾声。这几年的时光既漫长又短暂，其中充满了酸甜苦辣，更有收获和成长。几年来，感谢陪我一起度过美好时光的每位尊敬的老师和亲爱的同学，正是你们的帮助，我才能克服困难，正是你们的指导，我才能解决疑惑，直到学业的顺利完成。

本人的学位论文是在我的恩师刘志镜教授和姚勇副教授的殷切关怀和耐心指导下进行并完成的，衷心感谢我的恩师对我的淳淳教诲和悉心关怀。从课题的选择、项目的实施，直至论文的最终完成，他们都始终给予我耐心的指导和支持，我取得的每一点成绩都凝聚着恩师的汗水和心血。他们开阔的视野、严谨的治学态度、精益求精的工作作风，深深地感染和激励着我，在此谨向他们致以衷心的感谢和崇高的敬意。

感谢西安电子科技大学大学 14 级硕士班的全体同学陪我一起走过这段人生难忘的历程!

作者简介

##### 基本情况

朱群，男，福建人，1991年11月出生，西安电子科技大学计算机学院计算机技术专业2014级硕士研究生。

##### 教育背景

2010.09～2014.07 福建师范大学，本科，专业：软件工程

2014.09～2017.07 西安电子科技大学，硕士研究生，专业：计算机技术

##### 攻读硕士学位期间的研究成果

###### 3.1发表学术论文

1. Hong ZHOU, Zhi-jing LIU, Qun Zhu, Bing-hua WANG, Jing-jie LI. The Problem-Solving Agricultural Knowledge Service System for Farm Households, ICAMM, Bangkok Thailand, Octobe 2016, 2016 International Conference on Applied Mathematics and Mechanics, DEStech Publications,Inc. 2016, 472-478.

###### 3.2参与的科研项目

1. “陕勤网”项目，2015-5~2016-07，负责该项目手机Web的开发和维护。
2. 陕西省慧农科技“农掌门”项目，2015-10~2016-04，负责该项目Android APP农民端和专家端的开发和维护。