上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：Spark中推荐系统实现

学生姓名：颜荣圻

学生学号：5120309211

专　　业：信息工程

指导教师：孟桂娥

学院(系)：电子信息与电气工程学院

Spark 中推荐系统实现

摘要

互联网技术和大数据时代提供的海量信息资源不断改变着人们的工作和生活方式。然而随着信息的爆炸式增长，如何突破越来越困难的检索局面，从海量数据中挖掘出我们真正需要的信息成为一个重要的问题。为了解决这个问题诞生了相关的推荐系统应用，推荐系统可以协助用户从大量数据中选择有用的信息，在不同场景下的推荐算法也不同。在实际情况中，一台计算机通常不足以支撑和完成大量数据的存储与运算，或者需要消耗大量时间，所以我们采用分布式计算平台进行并行计算来完成我们对大量数据的处理。Apache Spark 是近几年来出现的一个关注度颇高的分布式计算平台，非常适合进行推荐算法所需要的多次迭代计算。本文将设计多种推荐算法并基于 Spark 平台进行推荐算法的实现和结果评估，研究如何充分利用用户评分数据、行为数据、关系数据优化推荐结果。本文还介绍了基于 Spark 的流式推荐算法的设计和实现，真实模拟在线系统的环境。

关键词：海量数据，推荐系统，Spark，并行计算，推荐算法

IMPLEMENTAYTION OFRECOMMENDER SYSTEM BASEDONAPACHESPARK

ABSTRACT

The Internet technology and massive informationresources provided by the Big Data era keep changing people's work andlifestyle. However, the exploration of information causes the problem of how to find the useful information from massive datasets and get out of the searching difficulty. To solve this problem, we have some recommender systems to help us choose useful information, and there are different recommendation algorithms in different cases. In fact, the storage and calculation of large amounts of data usually can't be accomplished a single computer. So we use distributed computing platform to do parallel computing on very large-scale data. Apache Spark is a popular distributed computing platform generated in recent years which is very suitable for iterative computations in recommendation algorithms. This paper will design some recommendation algorithms and implement these algorithms, evaluate outcomes based on Spark. The paper shows how to make full use rating data, behavioral data and relational data to optimize the recommendation results. It also introduces the design and implementation of a streaming algorithm on Spark for online system.

Key words: Big Data, Recommender System, Spark, Parallel Computing, Recommendation Algorithm

目　录

第一章 绪论

1.1 课题背景

1.1.1 大数据研究概况

互联网技术的高速发展和信息的快速传播让大数据成为当前热门的研究方向之一，大数据指的是数据集的规模已经大到超过了现有的数据库软件和其他运算工具的处理能力[1]。和传统的数据相比，大数据呈现出数据量大、速度快、数据多样化、数据价值增加等特征。数据量大指的是大数据的量级已经突破 TB 级别达到 PB 级别以上。速度快既指大数据的增长速度快，也指我们对数据的处理速度加快，由于数据持续实时产生，IDC (互联网数据中心)统计估测出约每两年全球的数据量就增长一倍，到 2020 年全球将有 35ZB 的数据量；此外，现有的物联网、移动互联网、社交网络、个人电脑和各式各样的移动通讯设备及信息采集装置也颠覆了传统处理数据的方式，实时分析大大加快了数据处理速度。数据多样化指数据的类型越来越多，信息以文本日志、图像、视频、位置坐标等方式呈现，如何在不同类型的数据中进行交叉分析成为数据分析的一个重点。隐含在大数据的信息价值也是大数据技术发展的一大重要原因，数据量的增加和数据的多样性让数据挖掘和数据分析、预测有了更多意义和实现的可能，但由于大数据的价值密度往往比较低，如何设计更高效的数据挖掘算法成为大数据计算的研究热点之一。

大数据技术的发展潜力和市场前景有目共睹：在 2012 年美国和日本分别启动了“大数据研究和发展计划”与“新 ICT 计划”，我国国务院于 2015 年发布了《促进大数据发展行动纲要》[2]，大数据已经成为重要的战略性资源被重点关注和开发应用。除了各国政府加大对大数据研究领域的投入，许多影响力巨大的科技巨头也相继进入大数据领域，Oracle、IBM、微软、Intel 等都推出了大数据相关的产品和解决方案，同时类似 MapReduce、Hadoop、Storm、Spark、HBase 的一些大数据处理技术和开源项目也逐步进入主流，发展迅速。我国的一些知名互联网企业如百度、腾讯、阿里巴巴也纷纷成立了大数据部门，开发相关的大数据技术和应用。

对大数据源源不断的投入诞生了许多不同场景下的大数据应用实例。淘宝运用分布式计算框架 Spark 的 Streaming 模块进行交易数据流、用户浏览数据流的数据流实时处理，从而进行更快速和准确的问题发现和分析预测，淘宝还运用 GraphX 模块对交易信息中的不同对象构建超大规模图，用来实现社区发现、关系衡量、属性广播等操作，从而方便了对大规模电子商务信息的高效处理。优酷土豆等视频网站运用 Hadoop、Spark 等大数据计算平台进行视频推荐和广告业务的管理。IBM 开发大数据实时分析软件帮助澳大利亚网球协会进行实时数据分析，完成对澳网的赛程状态、运动员人气、比赛历史记录和观众需求的信息获取，完善决策和赛事服务。总之，大数据已经逐渐渗透到各个行业和场景，改变着许多企业的运作模式，成为推动信息产业发展的新动力。

1.1.2 推荐系统研究背景

随着大数据时代到来，对于信息的消费者或者说互联网的用户们，如何从海量信息中获取自己真正需要的信息成为一个难题，虽然目前已经有许多功能日益强大的搜索引擎如 Google、百度等，但是搜索引擎产生的搜索结果通常并不具备不同用户的针对性；对于信息的提供者如各类互联网公司等，如何让自己提供的信息更有针对性和高效性，让自己的服务得到广大用户的认可，也是尤为关键的一个问题。

为了解决信息过量和检索困难的问题，个性化推荐系统开始出现，个性化推荐系统是根据用户的信息需求、兴趣等，将用户感兴趣的信息、物品等推荐给用户的个性化信息推荐系统。和搜索引擎相比，推荐系统着力于研究用户的兴趣爱好，利用用户的行为数据等信息进行计算，由系统发现用户的兴趣点，进而引导用户发现自己的信息需求。目前大多数推荐系统作为一个应用存在于网站或应用程序中，在电子商务、影音娱乐、社交网络、学术百科和商业广告领域中发挥作用。广为人知的有电商网站亚马逊 (Amazon) 的个性化推荐列表、淘宝的商品和广告推荐系统、Netflix 的电影推荐系统、豆瓣网的“豆瓣猜”个性化推荐、著名社交网站 Facebook 和 Twitter 的内容及好友推荐、个性化阅读工具 Google Reader 等[3]。

常规的推荐系统的工作原理是输入推荐所需要的各种类型的数据，经过推荐系统的核心部分推荐算法运算之后给出推荐结果[4]。一般来说，推荐系统所需要输入的数据集包括：推荐物品的信息，如标题、属性标签、表述等；用户的基本信息，如性别、年龄、地理位置等；用户与物品的关系信息，如用户对物品的评分、用户查看物品的历史记录、用户的购买记录等；用户或物品的关系信息，如用户之间存在联系与否，物品之间存在联系与否。推荐系统的输入信息通常中包含显式信息和隐式信息，显式的信息可以直接准确地反映出用户与物品的关系，例如音乐网站上的某个听众给某一首歌打了很高的评分或点赞，证明该用户喜欢这个物品，由此我们就可以得到一些用户的爱好信息；隐式的信息通常需要我们加以分析，选择合适的应用场景才能更好地发挥作用，例如电商网站平台上用户的购买记录信息，购买记录无法直接反映出用户对购买物品的评价，我们只能给出对用户属性的大致判断，要结合用户的行为信息等才能做出更好的判断。

推荐系统最重要的部分是推荐原理或者说推荐算法，根据推荐原理的不同推荐系统可以分为基于人口统计学的推荐机制、基于内容的推荐、基于协同过滤的推荐、社会化过滤推荐和基于位置信息的推荐。

基于人口统计学的推荐：基于人口统计学的推荐是一种较基本的推荐方式，通过计算用户基本信息的相关性，发现和目标用户相似的用户群体，将相似用户喜爱的物品推荐给目标用户。常用的信息有用户的性别、年龄、爱好等，然后根据这些特征属性计算用户相似度，得到和目标用户相似性大的用户作为目标的邻域，在邻域的物品中选择推荐物品。这种推荐方式的优点是不需要依赖物品的属性信息和用户的行为信息(用户对物品的浏览、收藏、评分等信息)，适合解决“冷启动”问题，即如何给一个没有行为信息的新用户推荐物品的问题。这种方式的缺点是推荐的精度不高，即便用户的属性相同，但对物品的看法喜好未必一样，甚至截然不同，而且这种方法需要采集用户的许多身份信息，可能因为保护个人隐私的原因无法完全实现。

基于内容的推荐：基于内容的推荐从物品的角度入手，计算物品或内容的相似度，然后根据用户的喜好物品推荐相似物品。这种推荐方式的关键在于物品特征的提取和相似度计算，优点是能够提供更准确，更有针对性的推荐，缺点是需要比较完整的物品信息(标签、特征值等)，此外该推荐方式的着力点是物品的相似度分析，忽略了人对物品的关系，而且该方式需要用户的喜好信息记录等历史信息，对于新用户不适用。

基于协同过滤的推荐：协同过滤的实质也是相似度的定义和计算，根据用户对物品的喜好信息(一般以评分的形式表示)，找到用户的相关性或物品的相关性，利用大量已知的用户喜好信息来预测用户对一些尚未接触过的物品的喜好关系。协同过滤可以分为基于用户的协同过滤、基于物品和基于模型的协同过滤。基于用户的协同过滤主要是根据用户对物品的偏好信息，计算出用户对各个物品的评分信息，根据预测的物品得分来做出推荐。与人口统计学推荐方式不同的是，基于用户的协同过滤通过用户的喜好信息和评分记录等信息计算相似度[5]。基于物品的协同过滤是根据用户对物品的喜好或评分信息来计算物品间的相似度，相似用户评分大致相似的物品有较大的相似性。基于模型的协同过滤是通过用户对物品的评分信息等训练出推荐模型，再根据实时用户信息进行物品的评分的预测和推荐。协同过滤是目前推荐系统广泛使用的推荐方式，它兼顾了用户之间、物品之间和用户对于物品的相关性，优势是不需要对用户或物品的多余特征描述，可以充分发现用户潜在的感兴趣对象，缺点是需要比较多的用户历史数据才能产生较理想的结果，且可能无法对一些喜好比较独特的用户做出有效的推荐。

社会化过滤推荐：通过用户、物品的社会化关系构建关系网络，基于邻域算法或图模

型进行用户、物品的推荐。例如通过社交网站上用户好友的信息可以得到用户间存在联系与否的一个关系网络，在关系网络中进行聚类、社区发现，从而得到和目标用户或目标物品同一类别或关联性较强的对象，进而向用户推荐相关用户(或相关用户偏好的物品)或是向用户推荐其喜爱物品的相关物品。这种社会化过滤的好处是利用明显的关系信息缩小推荐范围，提高了准确度，同时避免了对一些特征信息的采集和计算，加快了推荐的速度。缺点是社会化关系信息可能较难获取，聚类、社区发现等方式得到的推荐面比较狭窄，不利于发现用户潜在的兴趣点。

基于位置信息的推荐：位置信息也是重要的信息，目前各种社交网站和移动应用给位置信息的采集提供了可能，利用位置信息可以给带空间属性的用户推荐具备空间属性的物品，如实体商店、餐厅、电影院、景点和交通线路等。此外，用户对于物品的喜好可能也与地理位置相关，不同地方的用户兴趣一般有比较大的差异，一些线下活动的进行也受到位置信息的影响，位置信息有时候也是推荐物品的一个决定性因素。当然，基于位置信息的推荐的运用也有其局限性，对于互联网上的很多线上产品位置信息的作用并不大。

1.2 大规模分布式计算的研究现状

1.2.1 分布式计算架构

大数据的数据规模使得我们无法在单机上进行数据存储与计算，所以大规模分布式计算开始成为当前解决大数据处理的主要手段。当前比较热门的分布式计算架构有 Hadoop MapReduce、Storm、Spark 等。

（1）Hadoop MapReduce

Hadoop 是 Apache 下开发的分布式系统的基础架构，Hadoop 实现了一个分布式的文件系统 HDFS，具有很高的容错性，提供海量数据存储方案和高吞吐量供用户进行数据访问，适合在需要运用大规模的数据集的应用[6]。MapReduce 是 Hadoop 计算平台的核心。MapReduce 是 Google 提出的一个分布式计算框架，MapReduce 大致可以分为 Map 和 Reduce 两个计算步骤。对于一个计算作业，首先把计算任务分割成多个 Map 任务，分别在不同的计算节点上执行，每一个任务处理一部分数据，Map 任务完成后会生成一些中间结果，这些中间结果经过 Reduce 步骤汇总输出，实际上 Reduce 之前还有 Shuffle 等让计算性能更均衡的操作。MapReduce 的计算过程如图 1-1 所示。

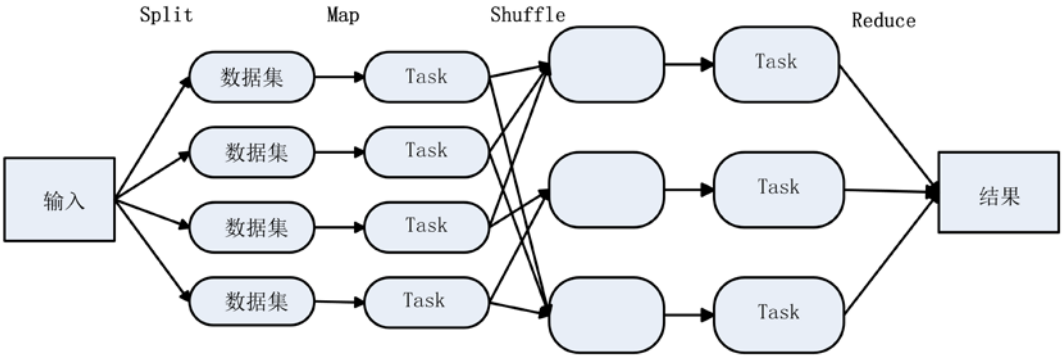

图 1-1 MapReduce 计算过程

在 Hadoop 中，执行 MapReduce 计算由 JobTracker 和 TaskTracker 负责，JobTracker 进行任务调度，包括具体计算任务的分割和任务执行状态的监控，TaskTracker 进行任务执行，包括任务计算的数据调用、计算状态汇报和计算结果提交等。

（2）Storm

Strom 是 Twitter 开发的实时计算平台，在保证能处理大规模数据的同时，Strom 还可以让数据处理更加实时，在低延迟条件下充分处理每条信息。从集群结构上看 Storm 和 Hadoop 很相似，但 Storm 上运行的是拓扑结构。Strom 集群的主要组成部分有 Nimbus, Supervisor,

Worker, Zookeeper。Nimbus 相当于主节点，负责任务分配和状态检测。Supervisor 负责接受主节点的任务分配信息并启动或停止所管理的任务进程。Worker 负责任务的具体执行。Zookeeper 的作用是协调 Nimbus 和 Supervisor 之间的消息指令发配。应用的具体实时实现逻辑是根据 Storm 中的拓扑结构(Topology)，Storm 的拓扑图结构由 Spout 数据源和 Bolt 数据操作组成，Spout 读取外部数据并转化为结构内部数据源给 Bolts 执行处理，Spout 和 Bolt 由 Stream Grouping 连接，Streaming 定义了数据流在 Bolt 间的划分问题。Strom 在信息流处理、连续计算、分布式远程程序协调(RPC)被广泛运用[7]。

（3）Apache Spark

Spark 是 UCB 实验室的开发的项目，之后迅速成为了 Apache 的顶级项目，在开源社区拥有很高的关注度[8]。Spark 的特点是运行速度快、高效易用、通用性好。Spark 集成 MapReduce 分布式计算框架的优点，对原有计算架构的缺点做了进一步改进，与 Hadoop 相比，Spark 将主要的计算步骤和计算中间结果都在内存中进行和存储，引入 RDD 弹性分布式数据集的数据抽象，从而提高了运算效率，避免了多次磁盘读写造成的计算速度缓慢，Spark 支持的有向无环图的运行流程也减少了迭代计算的数据重复读取次数。而且，Spark 的容错性更高，RDD 的计算可以通过 CheckPoint 操作来实现容错，也可以基于 RDD 弹性只读特征的进行计算数据的重建。Spark 提供多种数据操作类型，和 Hadoop 相比 Spark 更加通用，Spark 的数据集操作大致可分为 Transformation 和 Action 两类，此外各处理节点的通信模型种类较多。

Spark 除了提供高效的分布式计算框架来支持复杂的批量数据处理、数据交互式查询和实时数据流处理外，还拥有一个可以涉及机器学习、数据库、自然语言处理和信息检索等多领域的生态圈，以方便我们更好地运用 Spark 进行大数据应用的实现。Spark 生态圈以 SparkCore 为核心，包括 Spark Streaming 实时流处理组件、SparkSQL 数据查询、Spark ML/MLlib 机器学习算法组件、Spark GraphX 图处理组件和 Spark R 等。由于具有多种功能强大的组件，Spark 的适用范围和处理能力不断扩大，现有 Spark 应用的成功案例如淘宝运用 Spark 进行广告和推荐业务，腾讯用 Spark 进行数据实时处理解决系统用户请求量问题，一些电信运营商用 Spark 组件改造了用户信息分析查询平台和内容识别平台，提高了数据处理速度和能力。总之，由于 Spark 的诸多优势和快速便捷的应用实现，在 Spark 上开发实现的应用越来越多。

1.2.2 并行算法研究

为了在大数据计算平台上进行数据的灵活处理，相关的支持并行的计算方案或算法也是一个关键。基于 MapReduce 框架在 Hadoop 上就可以实现对数据的简单操作处理，但是如果我们需要进行更复杂的计算，做出较详细的数据分析和结果预测等，就需要在计算平台上引入一些模型与算法。

Mahout 是 Hadoop 下的一个机器学习与数据挖掘的分布式计算项目，通过调用 Hadoop 的库，Mahout 可以实现一些可扩展的机器学习领域的经典算法，包括分类、聚类、过滤推荐、距离计算等。

和 Mahout 相比，Spark 实现的算法更全面，由于 Spark 基于内存的计算模型而有更高的计算效率。Spark MLlib 主要支持四种类型的机器学习算法：分类、回归、聚类和协同过滤，SparkML Optimizer 会分析并帮助用户选择最合适的算法和参数，Spark MLI 可以进行特征抽取和高级机器学习编程抽象实现。出了机器学习算法的部分，Spark 还有支持图模型计算的 Spark GraphX，流式算法实现的 Spark Streaming 等，提供一套完善的大数据处理方案。

并行化算法的实现主要从三个方面入手：计算过程的批量数据统计处理的并行化；优化计算结果、迭代计算过程的并行化；算法设计方案本身的并行需求。

1.3 论文内容及研究意义

1.3.1 论文研究主要内容

本文研究了在 Spark 平台上推荐系统的实现，研究重点研究 Spark 平台推荐系统的优势和不同推荐推荐方案的推荐效果和具体实现，同时也进行完整的流式推荐算法的设计和实现，并对结果做出评估。具体研究内容如下：

（1）Spark 技术和推荐系统内容的研究。包括 Spark 系统架构和计算模型的分析，推荐系统类型、评测指标的分析整理。

（2）基于 Spark 的推荐方案的设计和实现。本文研究的推荐方案包括基于用户行为和物品属性的协同推荐，对用户的行为信息和物品属性进行特征提取和相似度的计算；基于矩阵分解的用户和物品的协同推荐，根据隐含语义模型进行评分矩阵的分解降维和评分预测；基于 K 均值聚类算法的协同过滤，根据用户或物品特征进行聚类和邻域分析[9]；基于关系网络的图模型建立和社会化推荐，利用论文引用数据搭建关系网络，进行图计算和节点分析。

（3）实现基于 Spark 的流式协同推荐算法。采用 Spark Streaming 组件进行实时数据流的处理和流式算法的实现，突破批量数据处理的局限进行数据的实时接收和协同推荐算法的动态运行，模拟真实的推荐环境。

（4）分布式推荐系统的实验设计结果。本文进行了推荐算法在 Spark 平台不同条件下的测试对比及 Spark 平台上推荐算法的单机和分布式测试等实验，并进行结果的分析。

1.3.2 研究意义

推荐系统在互联网的海量信息检索中具有重要作用，大数据时代的数据处理和推荐系统的运作需要用到专用的大数据处理工具如 Hadoop 和 Spark 等。与 Hadoop 相比。Spark 基于内存的计算模型和 RDD 数据模型减少了计算过程中数据在硬盘的读取和存入次数，从而减少了计算所需的时间，大大提高了计算效率。

在 Spark 平台上实现推荐算法，搭建推荐系统可以运用 Spark RDD 的多种操作便捷地对输入数据进行预处理，对推荐结果也可以直接进行评估，因此 Spark 是十分适合推荐系统实现的一个平台。本文设计和实现的推荐算法针对数据集和推荐场景分别从不同角度入手，有基于用户行为数据的相似度计算，基于评分数据的协同过滤，基于特征数据的聚类，也研究了如何将关系数据用于推荐和实时推荐系统的实现，对于大数据的处理和推荐系统的实现有重要意义。

1.4 论文结构安排

本论文的结构安排如下：

第一章介绍了课题研究背景，总结分析了当前大数据技术的发展情况和研究概况，推荐系统的研究背景，分布式计算的架构和算法研究及本论文的主要内容和研究意义。

第二章介绍了实现课题内容的相关技术，重点是 Spark 计算平台的计算原理和推荐系统的重要概念。

第三章介绍了基于 Spark 平台的几种推荐方案，包括基于用户行为和物品属性的协同推荐、基于矩阵分解的用户和物品的协同推荐、基于 K 均值聚类算法的协同过滤和基于关系网络的图模型建立和社会化推荐。

第四章介绍了基于 Spark 平台的流式推荐算法的实现，包括 Spark Streaming 流数据处理架构和流式协同过滤算法的设计和实现。

第五章介绍了算法实现的方法和结果分析及优化。

第六章总结了 Spark 中推荐系统实现的特点和难点，分析了未来 Spark 大数据应用开发和推荐系统的发展前景。

第二章 相关技术

本文介绍的推荐系统主要是基于 Apache Spark 实现对数据集的分布式处理，所以 Spark 的架构和运行原理对系统的设计非常重要

2.1 Apache Spark 的工作原理

Apache Spark 是通过优化 MapReduce 的计算框架实现的分布式并行计算平台，它的特点是将计算得到的中间结果存放在内存中，减少对磁盘中数据的多次读取和网络通信，提高了迭代计算的效率。Spark 的数据抽象 RDD 保证了程序的高容错性，并且增加了多种对数据的操作方式，应用场景更广。Spark 可以在不同环境下进行分布式并行计算，但模式基本相同，一般的分布式系统的集群模式如图 2-1 所示。

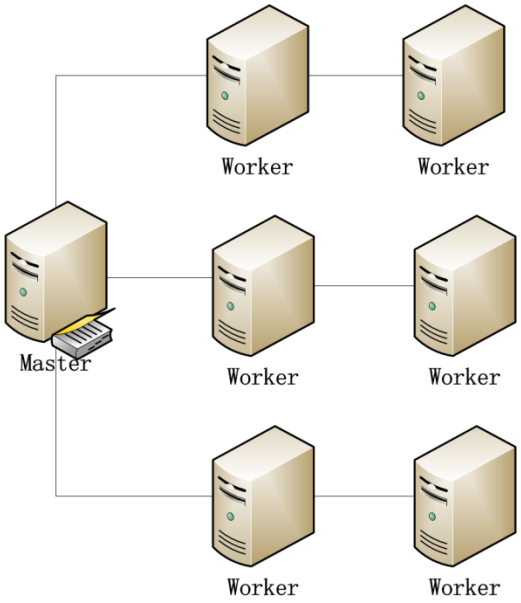

图 2-1 分布式系统的集群模式

一个集群由主机 Master 和从机 Worker 组成，Master 的作用是进行任务的协调分配，将计算任务分给 Worker，多个 Worker 同时进行计算任务，然后将计算结果发给 Master。

2.1.1 Spark 的系统架构和运行流程

我们从 Spark 的应用入手进行 Spark 架构和运行流程的解读。Spark 的应用程序中包含了完成 Driver 功能的部分和在集群节点上执行的 Executor 部分，Driver 相当于是 Master 的执行指令，Executor 相当于是 Worker 的执行指令。

Driver 部分主要是应用程序的主函数 main()，主函数中包含 SparkContext()接口，SparkContext()的主要作用是定义 Spark 应用的执行环境，请求外部资源并和其管理器进行通信，分配任务和任务。SparkContext()在 Executor()执行完毕后被停止。

Executor 是 Worker 上运行的一个进程，一般来说一个 Worker 有多个 Executor 进程，每个 Executor 有多个 Task 线程，Executor 能够运行的 Task 线程的数目取决于系统分配给它的 CPU 数目。

对于每个程序的计算任务的分配如下：Task 是发送到 Executor 的基本任务，Job 是多个

Task构成的并行计算作业，每个Job一般有多个RDD及其操作，每个Job可以分为多个Stage，每个 Stage 含有一些 Task，如图 2-2 所示。

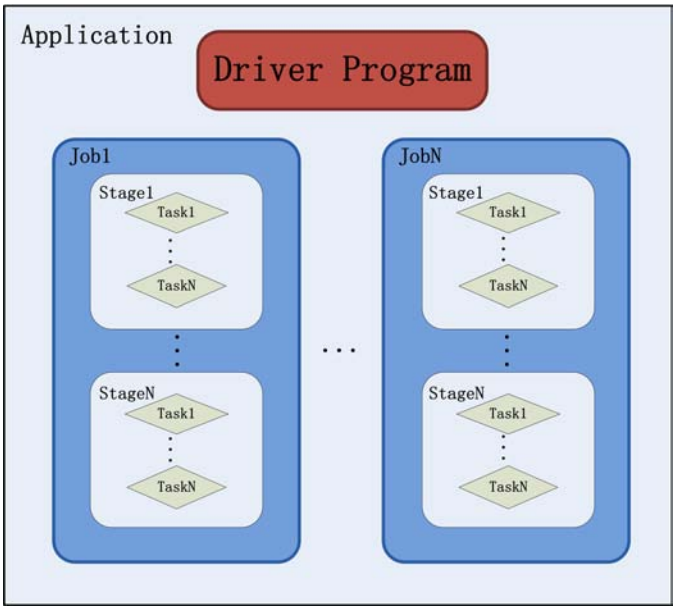

图 2-2 Spark 任务分配的结构

那么 Task 的划分是怎么进行的呢？SparkContext()申请到资源后用 DAGScheduler 一个计算流程转换为有向无环图(DAG)，将有向无环图分成 Stage 后发送给 TaskScheduler，再由 TaskScheduler 将 Task 发送给 Executor 执行。

DAGScheduler 将 RDD 的一系列计算过程转化为 Stage 的有向无环图，每个 Stage 可能包含 RDD 的几个操作，然后根据 Stage 与 RDD 的关系得到最优化的方法即 Task 运行的最佳位置，将 Stage 用 TaskSet 的方式传给 TaskScheduler。如果 RDD 在 Shuffle 阶段出现错误导致部分数据丢失，DAGScheduler 会重新给 TaskScheduler 发送 Stage。TaskScheduler 负责所有 TaskSet 的处理，当接收到一个 Executor 发送请求给 Driver 的消息时，TaskScheduler 会根据 Executor 的资源情况给其分配一个 Task，TaskScheduler 同时还进行所有 Task 运行的监控，如果有一个 Task 运行失败了，TaskScheduler 会让 Executor 重新执行一次。

Spark 的运行架构可以在不同模式的集群或本地中运行，运行模式有本地模式 Local，集群模式 Standalone，Yarn，Mesos 和 Cloud。Standalone 是 Spark 自带的集群运行模式，Yarn 和 Mesos 分别是运行在 Yarn 资源管理器和 Mesos 资源管理器上的模式，Cloud 是借助于一些云端服务的支持实现分布式。下面介绍 Spark 在自带的 Standalone 模式下如何实现典型的主机和从机的分布式计算流程。

Standalone 模式中主要有三个操作对象：客户端 Client、主机节点 Master 和从机节点 Worker，围绕着三个对象集群的运行流程如图 2-3 所示。

在定义每个 Spark 应用程序的 SparkContext 接口后，程序和 Master 节点连接，同时向 Master 节点请求 CPU 和内存资源。Master 根据 Worker 节点的资源分布情况调用 Worker 的计算资源，启动 StandaloneExecutorbackend 进程执行管理器，进程执行管理器和 SparkContext 取得联系。SparkContex 将应用程序的指令发给进程执行管理器，同时 SparkContext 解析执行，生成有向无环图发送给 DAG Scheduler，DAG Scheduler 将有向无环图分成多个 Stage 发送给 TaskScheduler，TaskScheduler 将任务给配给 Worker，进程执行管理器负责人物进程的执行和状态监控。在进程的各个线程执行的同时，进程执行管理器会不断和 SparkContext 进行通信，反馈程序的执行状态，在任务完成后，SparkContext 注销在 Master 上的全部信息，Master 对调用的计算资源进行释放。

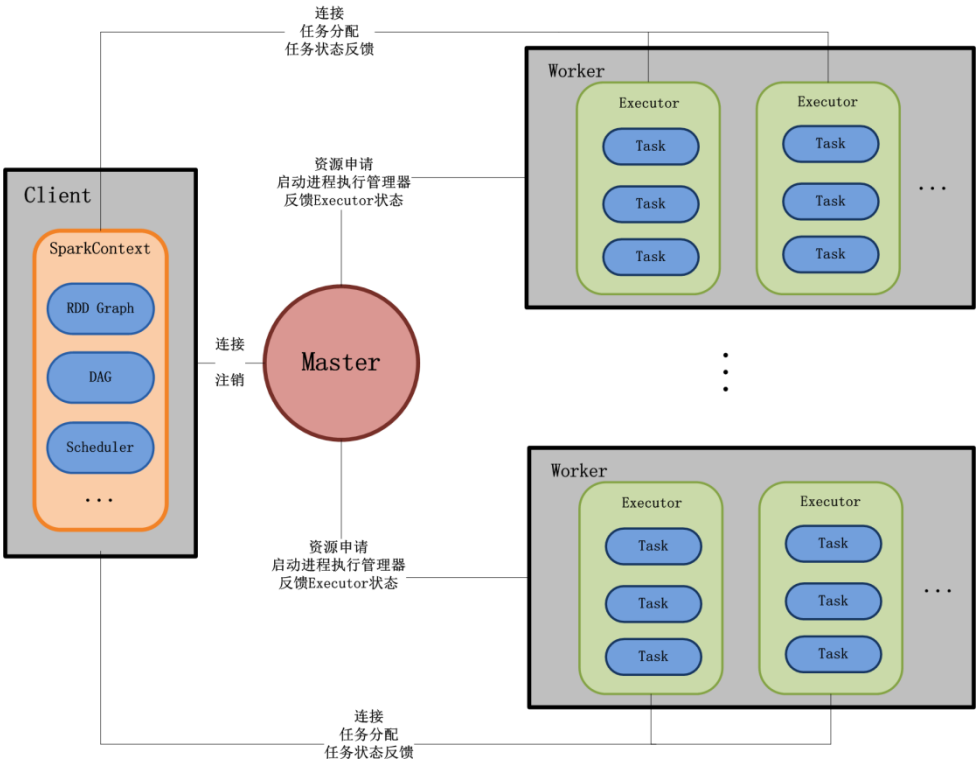

图 2-3 集群的运行流程

2.1.2 Spark 应用程序的编程模型

和集群的结构相对应，Spark 应用程序一般可以分为 Driver 和 Executor 两部分，Driver 相当于程序的驱动部分，Executor 包含多个任务进程，Driver 和 Executor 通过集群管理器 ClusterManager 连接。Driver 主要是进行 SparkContext 的定义和程序运行环境的配置，ClusterManager 负责在集群上取得外部的资源服务如 StandaloneYarn 等，Executor 是对数据的处理和计算。

Spark 的计算数据 RDD 可以来自于 Scala 函数的集合数据，通过 makeRDD 或 parallelize 操作将 Scala 数据生成 RDD，也可以来自于本地或 Hadoop 文件系统的数据集，用 textFile 操作将文件生成 RDD。

一般在运行时 Spark 可以通过 partition 操作在每个计算节点上都进行 RDD 的备份，这样在计算时各个节点相互之间不必进行数据的传输。对于一些需要共享的变量可以用 broadcast 广播变量类型进行数据存储。

Spark 计算模型的核心是 RDD，RDD 是弹性可分布数据集，也是 Spark 运算的基本单元。RDD 作为一种不可修改的只读数据类型，只能从存储系统生成或是通过原有的 RDD 生成。RDD 可以进行分区，可以被缓存到内存中，所以能够支持并行操作。

RDD 主要有两种操作：转换(Transformation)和动作(Action)。RDD 的转换操作会返回一个新的 RDD，转换操作有 map()，filter()，flatMap()，reduceByKey()，join()，sort()，Cartesian()等，但是转化操作并不是立即执行的，而是等到新的 RDD 有动作操作时才真正生成。RDD 的动作操作有 take()，reduce()，collect()，count()，save()，foreach()等，动作操作时 RDD 计算的启动操作，而且会将计算结果写入文件存储系统。

RDD 可以用 cache()和 presist()操作进行缓存，当 RDD 被缓存到内存中时，程序使用 RDD 的读取速度大大加快，缓存的 RDD 还保证了容错性，因为计算的中间结果出错时可以用 RDD 重新生成。

2.2 Spark 生态系统和主要模块

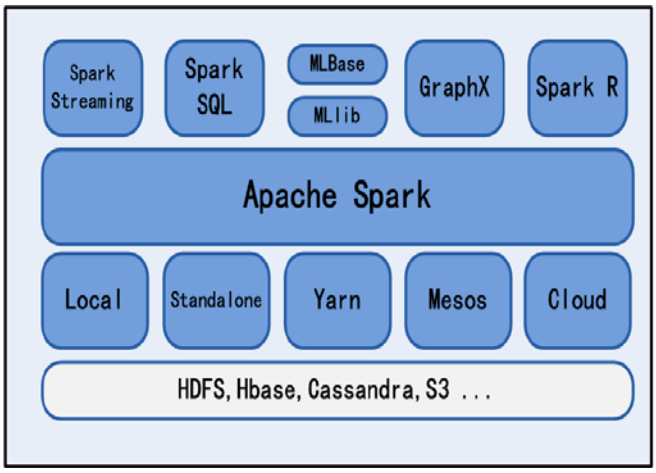

图 2-4 Spark 生态系统的组成

如图 2-4 所示是 Spark 生态系统的大致组成，从上到下是顶层的应用程序模块到系统核心架构到文件系统。Spark 的核心是 Spark Core，也就是基本的分布式计算框架，Spark 的运行模式有本地运行、独立运行和借助 Yarn 和 Mesos 等资源调度器运行，从 HDFS、Hbase 等分布式文件系统进行数据的获取和存储[10]。对于应用程序，Spark 有 Spark Streaming、Spark SQL、Spark MLlib、GraphX、Spark R 等模块支持不同功能的应用程序的实现。

Spark SQL 模块提供直接统一处理 RDD，用 SQL 的指令实现外部查询和各种数据统计分析操作。Spark SQL 设计了新的 RDD 类型 SchemaRDD，使 RDD 的定义可以同一般数据库中表的定义一样，此外 Spark SQL 解析 SQL 命令后通过 Catalyst 优化架构的一些类将 SQL 的执行转化为 RDD 的计算。Spark 采用内存的列存储、特殊的字节编码生成技术和 Scala 函数式编程的代码优化保证了其高效的查询性能。

Spark MLlib 是 Spark 的机器学习算法库，基于内存的计算特点使得 Spark 在进行机器学习算法的迭代步骤时具有很大的速度优势。Spark MLlib 的底层部分是一些运行库和向量、矩阵等计算类型的库，算法部分有回归算法、支持向量机、朴素贝叶斯、决策树、聚类算法等算法，每个算法还包括对应的测试文件和接口定义等。

GraphX 是 Spark 进行图计算的模块，Spark 对图的分布式计算是将大图分割成多个子图，对子图进行迭代和分段计算。GraphX 在 RDD 的基础上定义了适用于图元素表示的 RDD，每个图用表示节点的 RDD、表示边的 RDD 和记录节点分区的 RDD 来存储信息，对图的分布式存储有边分割和节点分割的方式，图的计算模型是全体同步并行模型。GraphX 还对图的缓存、相邻边的聚合操作和图的消息传递模式 Pregel 进行了改进，提高了图计算效率。GraphX 还提供一些现成的图算法。

Spark Streaming 是处理流数据的模块，它支持多种流式数据源的数据读取和数据流的实时批处理。Spark Streaming 在 RDD 的基础上定义了 DStream 的数据抽象，Dstream 很好地继承了 RDD 的一系列操作和容错性。Spark Streaming 还可以通过指定时间间隔来控制数据流的接收和处理速度。

第三章 基于 Spark 平台的推荐方案

根据数据种类和数据格式的不同可以实现不同的推荐方案，而推荐方案或推荐算法是推荐系统的核心，推荐算法的适用于否将直接关系到推荐效果的好坏。目前大多数推荐算法都和数据挖掘领域的机器学习算法有很大相关。运用大数据计算平台实现分布式算法是大规模数据集处理的首选方式，目前最普遍的机器学习算法基本是基于 MapReduce 框架或改进 MapReduce 框架实现的，最通用的计算平台是 Hadoop 和 Spark，Spark 的 RDD 数据格式和多种操作非常适合推荐算法对于数据集多次操作的需要。本章推荐算法实现采用的数据集主要有 Movie Lens 的数据集和 IEEE 的论文引用数据。

3.1 基于用户行为和物品属性的协同推荐

3.1.1 算法原理

在一个推荐系统中，用户的行为信息和物品的属性(标签信息)往往是最容易采集到的数据信息，相比于可能涉及到用户隐私的用户个人信息，用户的行为信息在用户使用推荐系统浏览物品、对物品做出收藏、点赞、评分的行为时就可以获取，而且通过分析用户的行为对于推荐也更有针对性。基于用户行为和物品属性的协同推荐方案的实施步骤如下：

（1）通过推荐系统 web 前端收集用户对物品的“收藏”、“点赞”或点击次数超过一定阈值的信息，然后进行人工定义用户行为，用户行为的定义如表 3-1 所示：

表3-1 用户行为定义

行为名称	行为描述
浏览	取值为（0，1），浏览点击次数超过规定值为 1
收藏	取值为（0，1），收藏为 1
搜索	取值为（0，1），搜索为 1

（2）对收集得到的数据进行处理的到结构化数据，如表 3-2 所示：

表 3-2 数据结构化

序号	用户	物品	浏览次数	浏览（阈值 3）	收藏	搜索	得分
1	User 1	Item 1	5	1	1	1	1
2	User 1	Item 2	4	1	0	1	1
3	User 1	Item 3	4	0	0	0	0
……	……	……	……	……	……	……	
N	User n	Item n	2	0	1	0	1

用户行为作用在物品上的得分是用户浏览、收藏、点赞、搜索值的做并操作得到的值，用户如果存在对一个物品的搜索、收藏、点赞和多次浏览等行为，说明用户对该物品感兴趣的可能性较大，相应地我们将该用户与该物品的关系标记为 1。

（3）统计物品的属性信息，如电影、音乐等物品对象一般有属性标签表明物品的题材、特征等，通过统计物品的属性我们可以进行物品的区分。物品的特征统计方式是对于每个物

品 j 建立一个长度为 m 的特征向量V_j。

$$V_j = [v_1, v_2, v_3, \ldots, v_m] \tag{3-1}$$

其中 m 是统计所有物品得到的特征类别数目，向量中的每个值表示物品 j 对应特征值的大小。例如我们统计了一个电影数据集得到电影的类别有剧情类、冒险类、动画类、科幻类、文艺类、喜剧类、爱情类、动作类、惊悚类，则我们对于每一部电影可以初始化一个和类别一一对应的特征向量[0,0,0,0,0,0,0,0,0]，对于一部电影 a，若它的属性标签有剧情类、喜剧类和动作类，并且这三个特征没有大小的区分，那么在向量V_a对应剧情类、喜剧类和动作类的值可标为 1，向量的其他值为 0，$V_a = [1,0,0,0,0,1,0,1,0]$。

对于每一个用户 i 同样建立和物品特征向量格式相同的特征向量U_i

$$U_i = [u_1, u_2, u_3, \ldots, u_m] \tag{3-2}$$

其中u_k和v_k代表的是同样的特征属性信息，$1 \le k \le m, k \in N$，用户特征向量的计算是通过用户行为数据和物品属性计算得到，对于每个用户，首先筛选出与该用户间标记为 1 的物品集合 A，统计集合 A 包含的对象数目得 S，然后将集合 A 中每个物品对象的长度为 m 的特征属性向量相加，相加得到的向量再除去 S，得到的向量就是用户特征向量。用户特征向量的含义是通过统计用户感兴趣物品的特征属性来得到用户感兴趣的物品类型。这样当每个物品的特征属性较少，难以进行准确的物品区分时我们也可以通过用户行为判断用户的喜好。累加对象特征向量再除去对象总数 S 的意义是为了避免用户行为信息的多少对用户特征统计产生影响，如果没有除 S 的操作，则收藏、搜索、浏览次数多的用户得到的特征向量对应位置上的值更大，这样得到的特征向量就无法进行后续计算。

（4）得到用户的特征向量后，开始进行用户相似度计算，我们实质上是 计算向量之间的相似度，对每个用户 i 的特征向量U_i，计算U_i与$U_t (1 \le t \le m, t \in N, t \ne i)$的相似度。相似度计算的方法一般有欧氏距离计算、皮尔逊相关系数计算、余弦相似度计算、曼哈顿距离计算等[11]。

欧氏距离计算一般用于欧氏空间中两个点的距离计算，若 x，y 是 n 维空间中的两个点，则他们之间的欧氏距离的计算公式是：

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3-3}$$

在用户特征向量相似度的计算中两个特征向量可以视为两个点进行计算，计算量较少。皮尔逊相关系数可以衡量两个数据集的相关性，对于两个大小为 n 的向量或数据集 x 与 y，皮尔逊相关系数计算公式如下：

$$P_{xy} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{3-4}$$

或

$$P_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{3-5}$$

皮尔逊相关系数的取值在[-1,1]之间，计算过程中涉及到 x 和 y 的均值、标准差等指标的计算，计算量较大。

余弦相似度计算被广泛用在计算文本数据相似度上，对于两个 n 维向量 x 和 y，余弦相似度的计算公式为：

$$\cos\theta = \frac{x \bullet y}{\|x\|^2 \|y\|^2} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2} \quad (3\text{-}6)$$

对于多维向量，曼哈顿距离是计算量最少但计算结果较为粗劣的一种相似度计算方法，对于两个 n 维向量 x 和 y，曼哈顿距离的计算公式为：

$$d(x,y) = \|x - y\| = \sum_{i=1}^{n} |x_i - y_i| \quad (3\text{-}7)$$

计算得到每个用户向量和其他用户向量的相似度后再可以进行相似度的排序，选取和目标用户相似度最高的几个用户，将这些用户曾经收藏、点赞、搜索或多次浏览的物品推荐给目标用户。

3.1.2 在 Spark 上的具体实现方法

在实现过程中由于缺少用户行为数据的来源，实验采用的是 MovieLens 的电影评分的数据集，该数据集提供了一百万条用户对电影的评分数据，每条数据的格式为用户 ID::电影 ID::评分，我们可以把每条评分数据看做用户对于物品的行为记录，该数据集还提供了每部电影的属性信息，格式为电影 ID::电影名称::电影类别，根据这个数据集的数据信息基于用户行为和物品属性的协同过滤算法在 Spark 具体实现的主要步骤如下：

（1）原始用户行为数据和的读取，每个数据集生成一个 RDD；

```
val rawData = sc.textFile("/…/movies.dat")
val rawRatings = sc.textFile("/…/ratings.dat")
```

（2）数据的预处理，对原始数据进行分割和数据类型转化，得到格式规范的数据，包括将所有物品的属性生成一个新的 RDD 以便于进行物品属性的统计；

```
val rawCategories = records.map(r => r(2)).flatMap(line=>line.split("\\|"))
```

（3）通过 RDD 的 distinct，collect 和 toMap 等操作对物品属性 RDD 进行去重和映射的生成，得到一个物品属性的集合 RDD categories，categories 的维度是 numCategories，表示物品特征的数目，然后对每个物品建立一个特征向量，生成一个物品特征向量的 RDD dataCategories，数据格式是 LabeledPoint(label, Vectors.dense(categoryFeatures))，一个 LabeledPoint 代表一个物品的记录，其中参数 label 是物品的 ID，categoryFeatures 表示物品的属性；

```
val categories = rawCategories.distinct.collect.zipWithIndex.toMap
val numCategories = categories.size
val dataCategories = records.map{ r =>
  val label =r(0).toInt
  val categoryIdx = r(2).split("\\|").map(d => categories(d))
  val categoryFeatures = Array.ofDim[Double](numCategories)
  for(i <- 0 until categoryIdx.length -1) categoryFeatures(categoryIdx(i)) = 1.0
  LabeledPoint(label, Vectors.dense(categoryFeatures))}
```

（4）以用户的 ID 为 key 值用 RDD 的 combineByKey 操作进行每个用户评分不为 0 的物品的统计，得到 itemsOfUsers 的 RDD；

```
val itemsOfUser = { ratings.combineByKey(
  (v: Int) => List(v), (c: List[Int], v: Int) =>v :: c, (c1: List[Int],c2: List[Int]) => c1 ::: c2)}
```

（5）对 itemsOfUsers 进行 map 操作，统计每个用户对应物品的特征属性，生成一个特征向量，最终得到一个新的 RDD userCategories，数据格式为(label, Array(categoryFeatures))，label 是用户 ID，categoryFeatures 是特征向量；

（6）得到用户的特征向量后，接到了的操作是通过一次 for 循环对每一个用户做一次和所有用户的相似度计算，每一次循环操作中先将用户的特征向量用适合进行代数操作的 DoubleMatrix()格式存储，然后用 userCategories 的 map 操作进行相似度计算，这样避免了两个 for 循环嵌套的情况，提高了计算效率，然后用相似度 RDD sims 的 top()操作选择出和用户相似度最高的 30 个用户数组 sortedSims，其中 Similarity() 表示相似度计算函数

```
val sims = userCategories.map { case (id, factor) =>
    val factorVector = new DoubleMatrix(factor)
    val sim = Similarity(factorVector,itemVector)
    (id , sim)}
val sortedSims =sims.top(30)(Ordering.by[(Int, Double),Double]{
    case (id, similarity)=>similarity}
```

（7）对于相似度的计算一共进行了三种实现，包括皮尔逊相关系数的两种公式和余弦相似度计算。

皮尔逊相关系数的计算方式 1：

```
def pearsonCorrelation(u: DoubleMatrix, v: DoubleMatrix): Double = {
    val ux=u.subi(u.sum / u.length)
    val vy=v.subi(v.sum / v.length)
    ux.dot(vy) / (ux.norm2() * vy.norm2())}
```

皮尔逊相关系数的计算方式 2：

```
def pearsonCorrelation2(u: DoubleMatrix, v: DoubleMatrix): Double = {
    val n = u.length
    val sumU=u.sum()
    val sumV=v.sum()
    val sum1 = n*u.dot(v)-sumU*sumV
    val sum2 = math.sqrt(n*u.norm2()*u.norm2()-sumU*sumU)
    val sum3 = math.sqrt(n*v.norm2()*v.norm2()-sumV*sumV)
    sum1 / (sum2*sum3)}
```

余弦相似度计算：

```
def cosineSimilarity(vec1:DoubleMatrix,vec2:DoubleMatrix):Double ={
    vec1.dot(vec2)/(vec1.norm2() * vec2.norm2())}
```

（8）得到与用户相似度最高的其他用户后，可以通过返回 itemsOfUser 给用户推荐相似用户喜欢的物品，完成推荐。

3.2 基于矩阵分解的用户和物品的协同推荐

3.2.1 算法原理

当我们有条件获取用户对物品的评分数据时，这些评分数据可以用二维矩阵的形式存储，矩阵的行代表用户，列代表物品，通常情况下用户评过分的物品只占所有物品的很小一部分，所以矩阵只有少量数据。例如我们得到如下的一些用户对物品的评分数据：

（User1，Item1，5）、（User1，Item2，3）、（User2，Item2，2）、（User2，Item3，4）、（User3，Item2，3）、（User3，Item2，4）、（User3，Item4，1）

可以把这些评分数据转化为如表 3-3 所示的评分矩阵 R：

表 3-3 评分矩阵 R

User/Item	Item1	Item2	Item3	Item4
User1	5	3	Null	Null
User2	Null	2	4	Null
User3	Null	3	Null	1

一般来说推荐系统的用户数量巨大，所以实际评分矩阵的维度也很大。我们要实现的推荐目标其实就是预测出评分矩阵中空缺位置的评分数值大小。操作一个维度很大的矩阵的计算量很大，所以我们采用矩阵分解的方式进行降维处理，矩阵分解就是将评分矩阵分解为两个低维度的矩阵 U 和矩阵 V，U 和 V 的乘积为 R，如图 3-1 所示。

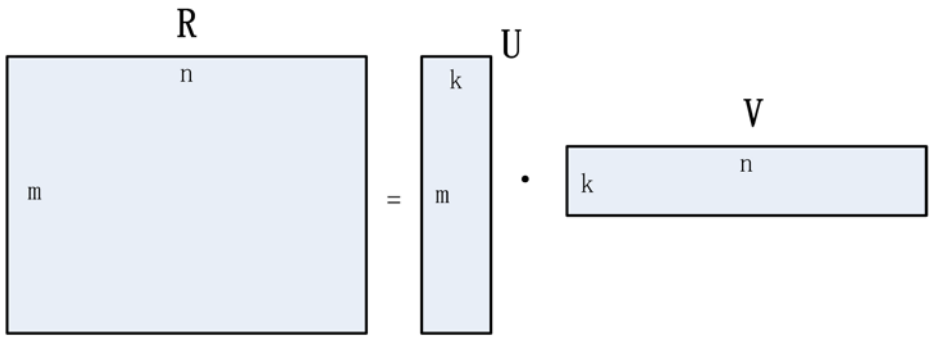

图 3-1 矩阵分解

其中，评分矩阵 R 的行数目为 m，列数目为 n，m 表示 m 用户数目，n 表示物品数目，U 矩阵是用户矩阵，行数目为 m，列数目为 k，V 矩阵是物品矩阵，行数目为 k，列数目为 n，其中 k 是一个远小于 m 和 n 的正整数，这样整个系统的复杂度就从 O(mn)降到了 O((m+n)*k)。U 矩阵和 V 矩阵可以视为因子矩阵，U 和 V 的稀疏度比 R 矩阵小得多。这个矩阵分解模型可以理解为：评分矩阵 R 的元素值大小由矩阵的内在用户-物品行为的隐含特征决定，隐含特征为用户对物品的种类、风格、属性的偏好，隐含特征的大小为 k。例如在用户对音乐的评分中，隐含特征可能是音乐的不同流派：摇滚、蓝调、古典、爵士、乡村、流行和电子音乐等，所以 R 矩阵分解得到的 V 矩阵的每一列表示一首歌在各个流派下的权重大小，相应的 U 矩阵的每一行表示一个用户对每种流派的喜好程度，所以矩阵 U 和矩阵 V 相乘就是计算每个用户对每首歌喜好程度的总和。实际上我们并没有定义矩阵分解的隐含特征类型，我们的计算目标就是通过已知的评分值来计算出最准确的矩阵 U 和矩阵 V，预测评分矩阵 R 中未知的评分值，我们希望 $U_{m\times k}V_{k\times n}$ 的值最接近 $R_{m\times n}$，也就是希望 $\sum_{ij}(r_{ij}-u_i v_j)^2$ 的值最小，其中 $1\le i\le m, 1\le j\le n$，r_{ij} 为 R 中的评分值，u_i 和 v_j 分别为矩阵 U 的行和矩阵 V 的列[12]。

为了便于计算我们将 R 和 UV 乘积的差值优化后的表示为损失函数，优化后的损失函数为：

$$L=\sum_{i,j}(r_{ij}-u_i v_j)^2+\lambda\|u_i\|^2+\lambda\|v_j\|^2 \qquad (3\text{-}8)$$

其中 λ 是正则化系数，加入正则化项是为了避免产生过拟合等问题，这样矩阵 U 与矩阵 V 的求解问题就转化为一个基本的优化问题。为了求解这个优化问题，我们引入最小二乘法，最小二乘法是一种合适求解矩阵分解问题的最优化方法，它的计算效果理想并且容易进行并行化。最小二乘法的原理是通过迭代计算求解最小二乘回归问题，在每一次迭代时规

定两个分解的低维矩阵的其中一个，用被固定的低维矩阵和评分矩阵的数值来更另一个分解的低维矩阵，然后固定更新过的低维矩阵进行另一个矩阵的更新，以此方式反复直到损失函数的结果收敛到一定值或完成一定量的迭代次数。迭代的流程如下：

（1）设定 k 的值，随机初始化矩阵 U 和矩阵 V。

（2）固定矩阵 V，损失函数对 u_i 求偏导，令导数为 0，得

$$u_i = (V^T V + \lambda I)^{-1} V^T r_i \tag{3-9}$$

进行矩阵 U 中所有值的更新。

（3）固定矩阵 U，损失函数对 v_j 求偏导，令导数为 0，得

$$v_j = (UU^T + \lambda I)^{-1} U^T r_j \tag{3-10}$$

进行矩阵 V 中所有值的更新。

（4）当迭代次数达到一定值或均方根误差收敛到某个值时停止迭代，均方根误差公式：

$$RMSE = \sqrt{\frac{\sum (R - UV)^2}{N}} \tag{3-11}$$

3.2.2 在 Spark 上的具体实现方法

在 Spark 上我们采用 MovieLens 的一百万条电影评分数据实现矩阵分解，算法的具体实现如下：

（1）读入用户对电影的评分数据，生成评分矩阵 R

```
val rawData = sc.textFile(" /…/ratings.dat")
val R = generateR(rawData)
```

（2）初始化低维矩阵 U 和 V

```
var U = Array.fill(m)(randomVector(k))
var V = Array.fill(n)(randomVector(k))
```

（3）用最小二乘法进行矩阵 U 和矩阵 V 的更新，跟新过程中使用广播变量类型存储 R、U 和 V，进行更新结果的实时同步

```
val Rb = sc.broadcast(R)
var Ub = sc.broadcast(U)
var Vb = sc.broadcast(V)
for (iter <- 1 to ITERATIONS) {
  U = sc.parallelize(0 until m).map(i => update(i, Ub.value(i), Vb.value, Rb.value)).collect()
  Ub = sc.broadcast(U)
  V = sc.parallelize(0 until n).map(i =>
      update(i, Vb.value(i), Ub.value, Rb.value.transpose())).collect()
  Vb = sc.broadcast(V)}
```

（4）得到矩阵 U 和矩阵 V 后，可以计算每个用户对于每部电影的评分，通过选取预测评分最高的电影作为推荐。

3.2.3 推荐结果的评估方法和实现

计算得到分解后的低维矩阵 U 和矩阵 V 后，我们要如何衡量矩阵分解的预测效果呢？这就需要我们运用一些评估模型预测效果或准确度的方法，有些评估方法直接衡量模型的预测目标变量值的好与坏，有些评估方法侧重于关注预测模型对于那些未进行针对性优化但较接近真实情况的数据的预测效果。我们可以用评估模型对同一个模型在不同参数下进行比较，或是对不同模型进行比较，从而找到预测效果最好的模型。在矩阵分解模型中本文选择

均方误差和 K 均值平均准确率最为模型的评估指标。

均方误差（MSE）可以直接评估我们对评分矩阵重建的效果，作为矩阵分解迭代的最优化的目标函数，均方误差的定义是计算所有真实值和预测值的误差的平方和与总数的商。

均方误差的计算公式为：

$$MSE = \frac{\sum_{i,j}(r_{ij} - \hat{r}_{ij})^2}{N} \quad (3\text{-}12)$$

r_{ij} 是真实值，$\hat{r}_{ij}$ 是预测值，在 Spark 中均方误差的具体计算步骤如下：

（1）生成评分数据的 RDD ratings，用 map 操作从 ratings 中生成用户 ID 和物品 ID 的 RDD，使用模型的预测函数对每个用户和物品的评分做预测，得到的 RDD preditions 以用户和物品的 ID 为 key 值，预测评分为键值对的值。

```
val usersProducts = ratings.map{ case Rating(user, product, rating)  => (user, product)}
val predictions = model.predict(usersProducts).map{
case Rating(user, product, rating) => ((user, product), rating)}
```

（2）再对 ratings 进行 map 操作，生成以用户和物品 ID 为 key 值的真实评分的 RDD，用 join 操作将真实评分的 RDD 和 predictions RDD 连接起来，生成一个以物品用户和物品 ID 为 key 值，真实评分和预测评分为键值对的值的新 RDD ratingsAndPredictions。

```
val ratingsAndPredictions = ratings.map{
    case Rating(user, product, rating) => ((user, product), rating)}.join(predictions)
```

（3）计算均方误差，用 RDD 的 reduce 操作进行误差平方的求和，用 count 函数计算总的评分数目。

```
val MSE = ratingsAndPredictions.map{
    case ((user, product), (actual, predicted)) =>  math.pow((actual - predicted), 2)
}.reduce(_ + _) / ratingsAndPredictions.count
```

计算得均方误差可以再求平方根即得均方根误差（RMSE）：

```
val RMSE = math.sqrt(MSE)
```

均方根误差也是一个普遍使用的方法。

K 值平均准确率（APK）通常用于衡量对于某个查询得到的前 K 个结果的平均相关性，是信息检索中的一个常用指标。在矩阵分解得到的推荐结果中，我们选取 K 个评分最高的物品作为推荐结果，那么这些评分相对更高的物品与用户的相关性是不是也更高呢？我们可以用 APK 对每个用户的推荐结果做评估，然后计算所有用户的 APK 的均值 MAPK[13]。

APK 的实现如下：

（1）提取出用户给过评分的物品 ID，生成一个物品序列 actual，通过矩阵分解模型预测出用户对物品的评分并选取分数最高的 K 个物品，生成物品序列 predicted。

（2）对 predicted 序列中的每个物品 ID 按排序序列检测其是否出现在物品序列 actual 中，如果出现则进行带权重分数的叠加。

```
def averagePrecisionK(actual: Seq[Int], predicted: Seq[Int], k: Int): Double = {
  var total = 0.0
  var numCovers = 0.0
  for ((p, i) <- predicted.zipWithIndex) {
    if (actual.contains(p)) {
      numCovers += 1.0
      total += numCovers / (i.toDouble + 1.0)}}
  if (actual.isEmpty) {1.0}
```

```
    else {total / k.toDouble}}
```

（3）对于所有用户评分的 MAPK 的计算，先生成矩阵 U 的 RDD，对矩阵 U 的每一行进行 map 操作，与矩阵 V 的每一列相乘得一个用户对所有物品的评分，这样就可以得到一个所有用户和各自对应的物品列表的 RDD recsForAll。用 groupBy 操作对实际评分数据 ratings RDD 根据用户 ID 进行合并，得到用户对应的所有物品序列 RDD userItems，再用 join 操作将 recsForAll 与 userItems 通过用户 ID 连接，结果可作为 APK 计算函数的输入值，这样便可以计算所有用户评分数据的 K 值平均准确率。

```
val MAPK = recsForAll.join(userItems).map{ case (userId, (predicted, actualWithIds)) =>
  val actual = actualWithIds.map(_._2).toSeq
  avgPrecisionK(actual, predicted, k)
}.reduce(_ + _) / recsForAll.count
```

其实 Spark MLlib 内置的 RegressMetrics 和 RankingMetrics 类也有相应的函数可以直接进行模型的 MSE、MAPK 的评估，其中 MSE 的计算和前述的基本一致，而 RankingMetrics 类的 K 值平均准确率和前述的方法略有不同，但 RankingMetrics 的平均准确率（MAP）计算结果和前述 K 值较大的 MAPK 的评估结果相同

3.3 基于 K 均值聚类算法的协同过滤

3.3.1 算法原理

前述的两种推荐方案都是直接对一个数据集进行处理从而进行推荐，这样的推荐方案在数据规模不是特别大的情况下比较适用，而当我们要处理的是一个超大规模的数据集时，即使我们运用 Spark 平台进行分布式计算，对数据的批量计算也需要占用比较多的资源。考虑到很多物品的推荐结果本身会因为用户类别存在较大差异。例如对于书籍的推荐，一般情况下不同职业和年龄段的用户对书籍的需求差别会很大，成年人一般不会喜欢少儿读物，人文社科的学者一般不会对工程类书籍感兴趣，男性也不太会对母婴读物产生兴趣。所以我们如果能对用户或物品进行类别的划分，那么就可以针对每一个不同类别群体做出最合适的推荐，在提高推荐准确度的情况下也能够减少计算量。

大多数物品会被人为地打上属性标签或分类，所以物品也比较容易区分，而用户群体存在很多复杂性和不确定性，对用户的划分和聚类比较困难，但用户聚类的意义也更大。在推荐系统中，我们可以利用用户行为特征或一些原始信息数据提取特征值，通过特征值将用户分成不同群体。

聚类的模型很多，其中 K 均值聚类是简单但效果很好的聚类算法之一，K 均值聚类也很适合做用户的聚类。K 均值聚类算法的目标是计算出几个类簇的中心点，将所有样本分成不同的类，其计算原理是使所有类簇中点的方差之和 WCSS 最小，K 均值聚类的算法流程如下[14]：

（1）初始化生成 K 个中心点，K 是预先设定的聚类的类别数目，中心点可以随机选取或设定为每个类簇中所有样本向量的均值；

（2）计算每个样本到各个中心点的距离，将其分类至距离最近的中心点所属的类簇，距离的计算公式为欧式距离公式；

（3）在每个类簇中计算出新的中心点，使得类簇内的平方差之和最小；

（4）重复（2）与（3）的操作，直到完成一定迭代次数或 WCSS 收敛。

$$WCSS = \sum_{i=1}^{m}\sum_{j=1}^{n}(p(j)-c(i))^2 \qquad (3\text{-}13)$$

其中，m 为类簇的数目，n 为某个类簇中的点数，$p(j)$ 为样本点，$c(i)$ 为中心点。

如图 3-2 所示表示了 K 均值聚类的一个简单流程：

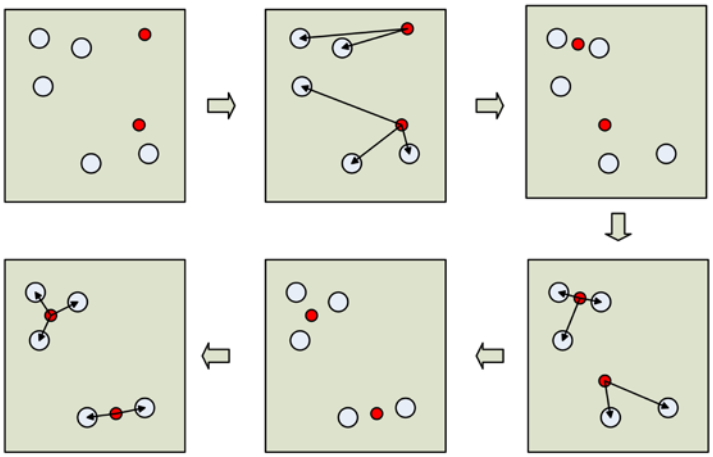

图 3-2 K 均值聚类过程

3.3.2 在 Spark 上的具体实现方法

要实现用户的聚类，我们首先需要提取用户的特征值，特征值可以用 3.1 节的方法通过用户的行为数据和物品的特征提取，也可以用 3.2 节矩阵分解的方式计算出矩阵 U，U 的每行代表一个用户的特征。为了减少计算量，本文选择通过用户行为数据和物品特征提取用户特征，具体步骤见 3.1 节。

由于 K 均值聚类算法是通过距离的计算完成中心点的定位，在得到用户特征后我们需要进行标准化和归一化，使得每个特征满足 0 均值和单位标准差，这样在距离计算时才不会出现大的偏差。否则一些每个点的距离计算容易受到一些数值大的特征值的影响，而一些数值较小的特征难以发挥作用。

数据的归一化可以用 Spark 的 StandardScaler 类的方法实现，StandardScaler 有两个参数，一个决定是否对数据减去均值，一个决定是否用标准差进行数据的缩放，数据归一化的实现如下，userVectors 是用户特征向量的 RDD:

```
val scaler = new StandardScaler(withMean = true, withStd = true).fit(userVectors)
val scaledData = userFactors.map(lp => (lp.label, scaler.transform(lp.features)))
```

然后就是 K 均值聚类的迭代步骤:

（1）随机初始化 K 个中心点

```
val centers = new Array[Vector](K)
  for (i <- 0 until K) {centers(i) = points(new Random().nextDouble(points.length))}
```

（2）计算每个点与中心点的距离并取距离最近的点作为样本的中心点

```
def closestPoint(p: Vector[Double], centers: Array[Vector[Double]]): Int = {
  var best Center = 0
  var closest = Double.PositiveInfinity
  for (i <- 0 until centers.length) {
    val tempDist = squaredDistance(p, centers(i))
    if (tempDist < closest) {
      closest = tempDist
      bestCenter = i}}
```

（3）进行类簇中心点的更新，

```
while(tempDist > convergeDist) {
  val closest = data.map (p => (closestPoint(p, kPoints), (p, 1)))
```

```
    val newPoints = closest.map {pair =>
  (pair._1, pair._2._1 * (1.0 / pair._2._2))}.collectAsMap()
    tempDist = 0.0
    for (i <- 0 until K) {tempDist += squaredDistance(kPoints(i), newPoints(i))}
    for (newP <- newPoints) {kPoints(newP._1) = newP._2}}
```

（4）经过一定的迭代次数后输出每个类别的一定数目的数据点，通过 RDD 的 map 和 collectAsMap 操作一并输出样本点对应的用户信息。

K 均值聚类模型的性能评估可以使用内部评价指标 WCSS，Spark MLlib 提供了函数 computerCost 进行数据集 RDD 的 WCSS 的计算，对每个点求其与中心点误差的平方和[15]。

3.4 基于关系网络的图模型建立和社会化推荐

除了利用用户的行为数据、物品特征信息和用户对物品的评分信息进行推荐的实现外，用户或物品的关系数据也是适合实现推荐的一个数据源。本文所说的关系指的是用户之间或物品之间的关系，如社交网站上用户 U1 添加了一个新的好友 U2，则 U1 与 U2 间多了一种好友的关系，再比如某学者新发表的某篇论文 P1 引用了其他几篇论文 P2、P3、P4 的内容，则论文 P1 和论文 P2、P3、P4 间多了引用关系。

3.4.1 算法原理

用户或物品间的关系连接实际上是确定对象相关性和重要性的重要依据，对用户或物品间关系的处理可以通过把关系网络转化为图来实现。例如当我们得到一系列论文间的引用关系数据集，我们可以将每篇论文视为一个点，论文与其引用对象之间的关系视为节点的边，论文 A 引用了论文 B 则存在一条由 A 指向 B 的边，对论文的相关性重要性的排序主要通过节点度的计算来实现，重要性可以简单地用节点的入度衡量，但是实际上节点间边的权重不会都相同，所以采用类似 PageRank 算法进行计算，步骤如下：

（1）对于一个论文引用的数据集，建立论文-论文的引用关系矩阵，若引用关系为 A->B/C/E/G，B->D/E/F，C->F/G，D->E/G，E->C，F->G，则建立矩阵 A' 如下：

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

同时因为一篇论文若有 k 篇引用，则它与引用的关系为 1/k，矩阵 A' 可优化为：

$$\begin{bmatrix} 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

（2）将论文-论文的引用关系矩阵和论文的重要性的相关性向量（可以根据论文的阅读量、下载量等）做矩阵运算，得到重要性矩阵V；如所得论文 A、B、C、D、E、F、G 的词频差别不大或是不好从内容区分相关性，则重要性矩阵$V=\left[\frac{1}{7}\ \ \frac{1}{7}\ \ \frac{1}{7}\ \ \frac{1}{7}\ \ \frac{1}{7}\ \ \frac{1}{7}\ \ \frac{1}{7}\right]$，$V_1=V\bullet A^{'}=\left[0\ \ \frac{1}{28}\ \ \frac{5}{28}\ \ \frac{1}{21}\ \ \frac{13}{84}\ \ \frac{5}{42}\ \ \frac{9}{28}\right]$，$V_2=V_1\bullet A^{'}$，$V_3=$……直至$V_n$收敛得重要性矩阵，通过重要性矩阵可以对论文进行排序推荐；

（3）论文 A 中的引用论文 B 一般是与 A 相同领域的论文或是 A 作者感兴趣的论文，同理得论文 B 的引用论文 C 可能也会引起 A 的作者的兴趣，通过追溯 A->B->C->……这层引用关系链，选择关系链路上入度较大或重要性较大的节点对应的论文作为推荐，同时可要考虑节点与 A 的距离，因为随着路径的变长，节点与 A 的相关性也会相应减弱。于是我们可以从推荐目标用户的某篇论文出发，通过引用关系建立论文和引用论文及引用论文的下级引用论文的关系网（类似树结构），如 A->B/C/E/G，B->D/E/F，C->F/G，D->E/G，E->C，F->G 关系建立树结构 S；

（4）考虑到论文的引用数量一般情况下这棵树可能会有较大规模，我们可以设置树的层数的最高值 n（n>2，n 为正整数），例如从论文 A 开始的引用关系 A->B->C->D，则层数为 3，B、C、D 分别为第一层、第二层、第三层的节点；

（5）从 i=2 到 i=n 遍历第 i 层的节点，然后比较此层中每个节点的入度值（可以是节点在所有论文的关系网内的入度或是在某一列别关系网中的入度）或出现次数，可以有调用 Spark 中 GraphX 模块统计出所有节点的的入度信息，入度最大或出现次数较多的点对应的是这颗树中重要性最大的一篇论文，我们可认为它是和父节点对应论文的作者有关系的论文中较有用的，可作为推荐。

如若仅考虑 A、B、C、D、E、F、G 在 S 的入度，分别为 0、1、2、1、3、2、4，重要等级可排列为 G、E、C/F、B/D、A，从 A 开始的第一层有 B、C、E、G，第二层有 D、E、F、G，第三层有 C、E、G，第四层有 C、G，第五层有 G，发现 G 的入度最大且出现次数最多，G 可作为推荐首选。而若考虑在整个论文的关系网中的节点度这个等级排序可能会有所不同，一般可以更好地区分论文的重要性等级；

（6）当这棵树较稀疏（如无回路）时我们可以单独考虑每条链路上的节点，选择拥有较长路径的链路上的点对应的论文 S 中的一些链路为 A->B->D->E->C->G 上入度较大的点对应的论文。

3.4.2 在 Spark 上的具体实现方法

Spark 的 GraphX 模块是一个高效的分布式图处理框架，它扩展了 RDD 抽象，有 Graph 和 Table 两种图模型的视图，分别有各自的操作符，提供了了灵活多样的操作和很高的运行效率。GraphX 是基于 Pregel 模式实现并行计算的，之所以不用 MapReduce 是因为在图计算方面 MapReduce 框架存在很多缺点，主要是在计算过程中需要进行多次的硬盘读取和网络通信，效率很低。Pregel 的运行原理是将图的运算通过在图的拓扑节点上的一系列迭代步骤实现，Pregel 中的主要数据对象有节点、边和消息，在每一次迭代过程中，每一个节点接收上一次迭代的消息并发送消息给其他节点，同时根据接收的消息和定义的规则改变自身和边的状态，进行图拓扑结构的一次调整。在一次迭代中节点对数据的接收和处理称为一个超步，也就是说每个节点在一个超步中接收上一个超步的消息并进行本次超步的消息发送。这种消息传递模式避免了远程读取数据的延迟和数据同步的复杂工作。每个节点在一次超步的数据处理操作后根据定义判断是否继续进行消息的发送处理，如果节点进入停止状态，只有等待下一次触发才会重新进行数据处理和消息发送，当图中的所有节点都是停止状态时图计算的迭代过程就结束了[16]。

Spark 中 Pregel 模型的实现主要有三个函数，分别是 vertexProgram，sendMessage 和 messageCombining。vertexProgram 函数的作用是定义每个节点对数据处理步骤，sendMessage 函数用于发送消息，进行邻接节点之间的数据传递，messageCombining 函数定义消息的合并的逻辑，对于同一个节点接收到的多个其他节点发送的消息进行合并处理[17]。

实际上 Spark GraphX 模块已有对 PageRank 的实现，具体实现步骤如下[18]：

（1）初始化每个节点的重要性 Rank 值，取值为$1/N$，N 为节点总数，考虑到并不是所有论文都存在被引用关系就是并不是所有节点的入度都大于 0，对所有节点 Rank 向量的更新处理可以用如下改进公式：

$$V_{i+1}=(\frac{\alpha}{N}\bullet I+(1-\alpha)A^{'})\bullet V_i \qquad (3\text{-}14)$$

相应地 vertexProgram 的实现为：

```
def vertexProgram(id, attr, msgSum): (Double, Double) = {
val (oldPR, lastDelta) = attr
val newPR = oldPR + (1.0 - resetProb) * msgSum
    (newPR, newPR - oldPR)}
```

（2）消息的发送函数 sendMessage 的实现为：

```
def sendMessage(edge: EdgeTriplet[(Double, Double), Double]) = {
    if (edge.srcAttr._2 > tol) {Iterator((edge.dstId, edge.srcAttr._2 * edge.attr))}
    else {Iterator.empty}}
```

（3）消息合并的函数 messageCombiner 的实现为：

```
def messageCombiner(a: Double, b: Double): Double = a + b
```

（4）调用 Pregel 模型实现以上函数：

```
Pregel(pagerankGraph, initialMessage, activeDirection = EdgeDirection.Out)(
        vp, sendMessage, messageCombiner).mapVertices((vid, attr) => attr._1)
```

对节点树的构建也可以通过 Pregel 框架实现，实现方案是先定义从一个节点 P 出发要建立的树的层数 T，然后节点对它的邻接节点发送数值 T，邻接节点接收到数值 T 后对邻接节点发送数值（T-1），这样与节点 P 距离为 n 条边的节点接收到的数值为（T-n），节点接收到数值 0 后变不再发送消息。最后我们提取所有接收到消息的节点，根据节点接收到数值的大小可以区分每个节点属于树的哪一层，具体实现函数如下：

（1）vertexProgram 函数是节点数据的更新处理：

```
def vetrexProgram(vid, vdata, message):Map = messageCombiner(vdata,message)
```

（2）sendMessage 函数对每个节点对进行操作，节点发送接收到数值减一后的数值消息给邻接节点：

```
def sendMessage(e:EdgeTriplet[VMap, _])={
   val srcMap=(e.dstAttr.keySet -- e.srcAttr.keySet).map { k => k->(e.dstAttr(k)-1) }.toMap
   val dstMap=(e.srcAttr.keySet -- e.dstAttr.keySet).map { k => k->(e.srcAttr(k)-1) }.toMap
   if(srcMap.size==0 && dstMap.size==0) {Iterator.empty}
        else {Iterator((e.dstId,dstMap),(e.srcId,srcMap))}
```

（3）消息合并函数 messageCombiner，若一个节点接收到两个不同的数据时，取较小的一个作为节点的数值：

```
def messageCombiner(spmap1, spmap2) = (spmap1.keySet ++ spmap2.keySet).map {
      k=>k=math.min(spmap1.getOrElse(k), spmap2.getOrElse(k))}.toMap
```

（4）最后是调用 Pregel 框架进行函数的实现，进行起始节点数值 T 的定义和 Pregel 迭代次数的规定:

```
val Tree=initialGraph.mapVertices((vid,_)=>Map[VertexId,Int](vid-> T))
    .pregel(Map[VertexId,Int](), T, EdgeDirection.Out)
    (vertexProgram, sendMessage, messageCombiner)
```

（5）完成 PageRank 节点重要性的计算和节点树的建立后，可以对一棵树某一层的节点进行入度或 Rank 值的排序作为推荐对象，先根据节点数值用 filter()操作过滤出某棵树某层的所有节点，再根据节点 ID 和 Rank 的键值对查找该层节点的 Rank 值:

```
val twoJumpFirends=Tree.vertices.mapValues(_.filter(_._2==n).keys)
val tjf =twoJumpFirends.map{ case (a,b) =>b}.take(k)
val tjfRank=tjf.map { layer =>
  val layerLength =layer.size
  val layerRank =Array.ofDim[(graphx.VertexId,Double)](layerLength)
  var i=0
  val idArray = layer.toArray
  for (k <- 0 until layerLength -1) {
    val r = rankCategory(idArray(k))
    layerRank(i)=(id,r)
      i+=1} layerRank.toIterable}
```

事实上，每篇论文的作者也可以通过论文的引用关系联系起来，建立了一棵树后可以对树节点对应论文作者及合作者进行一个聚类，向这个类簇中的作者推荐其他作者的关注度高的论文。

第四章 基于 Spark 平台的流式推荐算法

在现实推荐系统的应用中，我们不仅要实现对大规模数据的处理，还要考虑数据集实时变动的问题，因为在线推荐系统每天都会收集到新的用户数据和反馈信息，以及新上线物品信息，通常来说新的用户和新的物品是实现推荐的重点关注对象。如我们直接将新的数据添加到原来的数据集，重新进行批量数据的处理，那么需要的计算量太大，长时间的计算也无法适应推荐的时效性。那么我们能否实现仅仅对新增数据的处理而更新推荐结果呢，这就是流数据的处理问题。Spark 的 Streaming 模块就是 Spark 核心部分的一个扩展，可以实现大批量实时流数据的处理，具备完善的容错机制，运用 Spark Streaming 模块我们可以完成一个流式推荐系统的搭建。

4.1 Spark Streaming 流数据处理架构

4.1.1 Streaming 运行原理

Spark Streaming 支持 Kafka、Flume、HDFS/S3、TCP socket 等数据源，用和 RDD 操作相同的一系列操作进行数据处理和算法实现。Streaming 对流数据的处理机制是按照一定的时间间隔将接收到的实时数据流 DStream 分成小批的 RDD，每一 个 RDD 包含了从 Streaming 应用程序接收到的一个时间段中的消息记录[19]，DStream 的处理如图 4-1 所示。

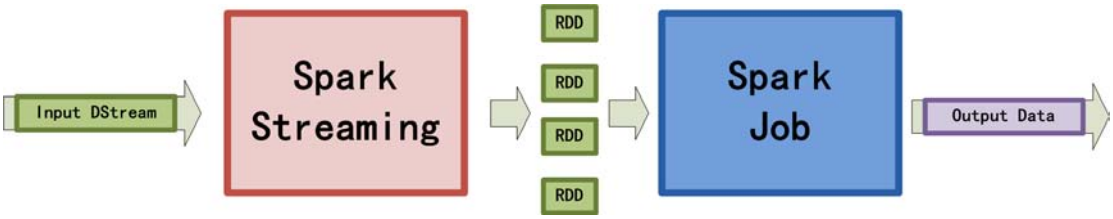

图 4-1 DStream 处理流程

Spark Streaming 基于时间的顺序进行数据流的批量处理，处理数据的基本单位由时间窗决定，时间窗函数对数据流上一个滑动窗口范围内的数据转换进行计算，时间窗有两个属性：窗口长度和滑动间隔滑动间隔小于窗口长度，通常时间窗以一定倍数的批处理时间间隔滑动，滑动时保证能覆盖所有的 RDD，对所有 RDD 都进行操作[20]，如图 4-2 所示。

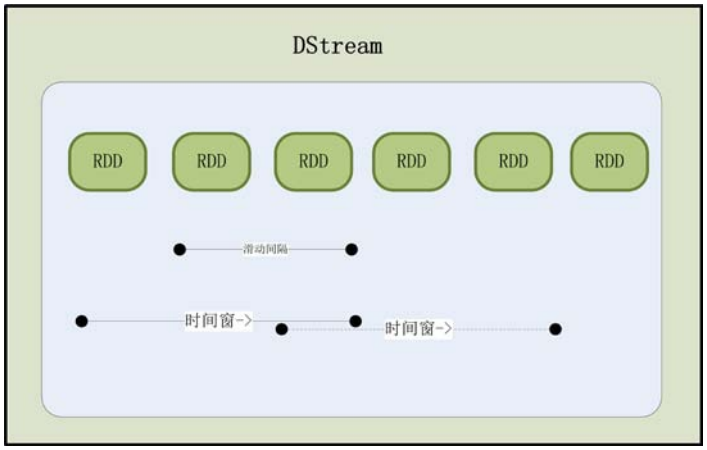

图 4-2 时间窗和滑动时间

因为 Spark Streaming 把 DStream 分解为一系列的批处理的 RDD 数据块，所以对 DStream 的数据操作就等同于对 RDD 进行操作，由于每个 RDD 是一个不可变的数据集，只要我们输入的数据具有容错性，则任一个 RDD 的出错都可以通过原始数据的转换操作重新运算，相应的 DStream 和 RDD 一样具有很好的容错性。Dstream 既可以通过外部数据源生成，也可以对现有的 DStream 进行 tramsformation 的相关操作得到。由于对数据的处理都是通过 Spark 核心部分实现的，所以 Spark Streaming 模块集成了 Spark Core 的一系列优势。

4.1.2 Streaming 编程模型

Spark Streaming 的编程主要分为以下几个部分：StreamingContext 接口的创建，InputStream 的创建，对 DStream 的操作和 Spark Streaming 的启动。

StreamingContext 接口的创建和 SparkContex 的创建基本一致，包括指定 Master 主机地址，设定程序名称，不同的是需要指定时间间隔参数 Second()，Second(1)表示设定时间窗窗长为 1 秒，时间间隔需要根据实际需求和集群具备的处理能力设置。

InputStream 是数据源的定义，若以 socket 套接字连接作为流数据的数据源，则为 socketTextStream()，括号内为端口号，当然 InputStream 接收的也可以是 Kafka、Flume、HDFS/S3 等数据源。

从数据源得到 DStream 后，可以对 DStream 进行一系列数据的预处理操作，操作方式类似 RDD，包括数据的分割、映射、合并、输出显示等。

以上的几个步骤是运行流程的基本定义，Streaming 程序并没有开始执行，需要调用 ssc.start()才真正启动程序进行数据的接收和处理。

4.2 流式协同过滤算法的设计和实现

由于协同过滤算法是最常见的推荐算法，运用场景也最广，因此我们希望实现流式的协同过滤算法，算法的模型是矩阵分解。

4.2.1 流式协同过滤的算法原理

流式算法与批量数据的静态算法最大的区别是处理的数据集在不断更新中，而我们希望算法的迭代过程优先对新的数据进行处理，更新计算结果，然后再而对原有的老数据进行计算，因为新数据对推荐结果产生的影响一般会比老数据迭代产生的作用来得大[21]。

静态的基于矩阵分解模型的协同过滤算法采用最小二乘法进行矩阵分解的最优化求解，矩阵分解优化的目标函数是：

$$L=\sum_{i,j}(r_{ij}-u_i v_j)^2+\lambda\|u_i\|^2+\lambda\|v_j\|^2 \tag{4-1}$$

最小二乘法每次迭代时固定一个低维矩阵，更新另一个低维矩阵，这样的方式适用于一个大规模的矩阵的分解，而流式算法每次对多批小量数据的处理若可以同时进行两个低维矩阵的更新，则可以减少一些重复操作。因此本文采用随机体第下降的方法进行矩阵分解目标函数的最优化求解。

随机梯度下降是通过多次迭代，每次迭代通过真实评分 (i,j,r_{ij}) 来更新低维矩阵的部分 u_i,v_j，其中 $1\le i\le m,1\le j\le n$，更新的方式是对 u_i,v_j 做损失函数梯度的反向调整：

$$u_i \leftarrow u_i-\eta\nabla_{u_i}L(r_{ij},u_i,v_j) \tag{4-2}$$

$$v_j \leftarrow v_j-\eta\nabla_{v_j}L(r_{ij},u_i,v_j) \tag{4-3}$$

其中 η 是每次梯度下降调整的步长，由于每次调整都只用到评分矩阵的部分数据，并且更新部分结果，所以随机梯度下降适合并行算法，在进行流式并行计算时我们可以优先用新

的评分数据 r_{ij} 进行计算，所以流式算法中随机梯度下降部分的实现流程如下：

（1）随机初始化矩阵 U 和矩阵 V；

（2）接收数据流并更新评分矩阵 R，对 R 中的新数据进行优先计算的标记；

（3）按一定条件选择要计算的数据集 (i, j, r_{ij})，通过每个 (i, j, r_{ij}) 更新 u_i 和 v_j

$$u_i = u_i - 2\eta[(u_i v_j - r_{ij})v_j + (\lambda / m)u_i] \quad (4\text{-}4)$$

$$v_j = v_j - 2\eta[(u_i v_j - r_{ij})u_i + (\lambda / n)v_j] \quad (4\text{-}5)$$

（4）当矩阵 U 和矩阵 V 收敛或不接受数据流时停止迭代。

随机梯度下降作为矩阵分解的主要计算步骤如何实现并行化呢？如果我们希望把随机梯度下降的计算量分配到集群的 p 个 Worker 上，这就需要我们对评分矩阵 R 进行分块处理，对每个 Worker 分配一定数量的数据块[22]，如图 4-3 所示，评分矩阵 R 被分为 p^2 块大小相同的数据块，矩阵 U 和矩阵 V 各被分成 p 块大小相同的数据块 B_{xy}，$1 \le x, y \le p$。

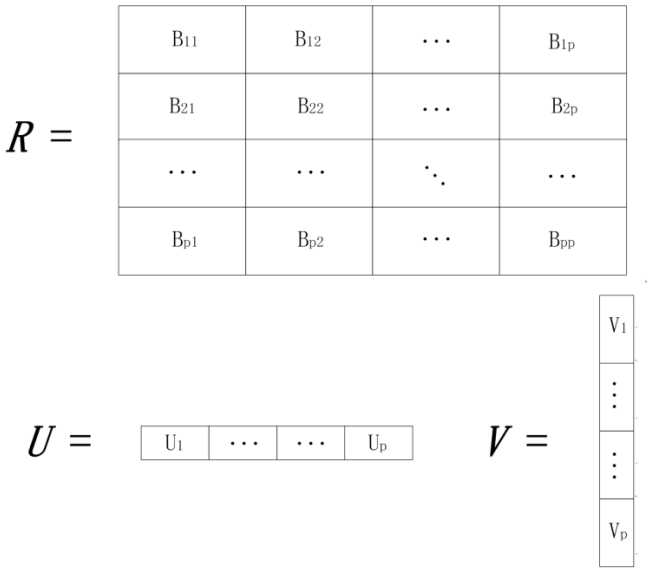

图 4-3 矩阵分块和并行化的实现

实现并行化要注意的主要问题是要避免两个 Worker 对矩阵 U 或矩阵 V 的相同部分进行更新产生的冲突。从图 4-3 可以看出评分矩阵的每个数据块 B_{xy} 由 U_x 和 V_y 计算得到，$1 \le x, y \le p$，每个 B_{xy} 中的评分数据对 U_x 的其中一行和 V_y 的其中一列产生影响，所以只要任意两个评分矩阵的数据块 B_{xy} 和 $B_{x'y'}$ 所包含的行和列不重合，所更新的 U_x 和 $U_{x'}$ 或 V_y 和 $V_{y'}$ 就不会产生冲突，即 $x \ne x', y \ne y'$ 是避免冲突的必要条件。根据这个条件我们每次从评分矩阵的数据块中选出 p 块数据块需要满足任意两个数据块不在同一行和同一列上，这 p 块数据块由 p 个 Worker 进行处理可以一次更新矩阵 U 和矩阵 V 所有部分，因此流式矩阵分解的并行化方案实现步骤如下[23]:

对于 Master:

（1）建立矩阵 R，随机初始化矩阵 U 和矩阵 V；

（2）根据 Worker 的数目 p 对矩阵 R 进行分块，建立矩阵块的锁函数，锁函数决定选块操作能否选择某行或某列的块；

（3）接收数据流，对于矩阵 R 中有新数据的数据块进行标记；

（4）进行数据块的选取，优先选择有新数据的数据块，当选择了 B_{xy} 后，用锁函数对 B_{x*} 和 B_{*y} 的所有块上锁，上锁后的数据块不再选取，直到选取了 p 块数据块再对每个数据块解锁，然后将 p 个 (B_{xy},u_x,v_y) 发送给 p 个 Worker，等待 Work 完成计算和 (u_x,v_y) 的更新后，收集每个 Worker 的 (u_x,v_y) 数据更新矩阵 U 和矩阵 V；

（5）重复执行（3）和（4）的步骤直到矩阵 U 和矩阵 V 收敛。

对于每个 Worker：

（1）从 Master 接收 (B_{xy},u_x,v_y) 数据；

（2）对于 B_{xy} 中的每个评分数据 r_{ij}，用随机梯度下降法进行 u_i 和 v_j 的更新：

$$u_i = u_i - 2\eta[(u_i v_j - r_{ij})v_j + (\lambda / m)u_i] \quad (4\text{-}4)$$

$$v_j = v_j - 2\eta[(u_i v_j - r_{ij})u_i + (\lambda / n)v_j] \quad (4\text{-}5)$$

（3）更新 u_i 和 v_j 后将结果返回给 Master，等待下一次计算。

4.2.2 流式协同过滤的在 Spark 上的实现方法

用 Spark Streaming 实现流式算法分为两个部分，流数据的生成和流应用程序的创建。流数据的生成步骤为[24]：

（1）先定义每秒钟要处理的最多数据数据 MaxEvents；

（2）流数据源的定义，本算法采用的数据源是读取本地的评分数据再生成数据流：

```
val ratings = scala.io.Source.fromInputStream(ratingResource)
```

（3）定义一个生成一定数目的评分数据的函数 generateRatingEvents，返回格式为(user, product, rating)的评分数据：

```
def generateRatingEvents(n: Int) = {(1 to n).map { i =>
      val inputData = random.shuffle(ratings). split()
           (inputData(0), inputData(0), inputData(0))}    //(user, product, rating)
```

（4）创建网络套接字并定义消息生成器产生数据流：

```
val supervisor = new ServerSocket(9999)
while (true) {
   val socket = supervisor.accept()
   new Thread() {override def run = {
        val out = new PrintWriter(socket.getOutputStream(), true)
        while (true) {
           val num = random.nextInt(MaxEvents)
           val productEvents = generateProductEvents(num)
           productEvents.foreach{ event =>
```

```
                out.write(event.productIterator.mkString(","))
                out.write("\n")}
              out.flush()}
            socket.close()}}.start()}
```

流应用程序的创建：

（1）数据接口的创建，StreamingContext()的第一个参数是 Master 的地址，时间间隔定义为 10 秒：

```
val ssc = new StreamingContext("*", "Streaming App", Seconds(10))
```

（2）从套接字读取文本信息进行流数据的接收和数据格式的规范化处理，创建一个 Dstream 的序列 rddQueue，rddQueue 中加入接收的数据流生成待输入数据 inputStream：

```
val stream = ssc.socketTextStream("localhost", 9999)
val ratings = stream.map { record =>
  val event = record.split(",")
  MatrixEntry(event(0).toInt -1, event(1).toInt -1, event(2).toDouble)}
val rddQueue = new SynchronizedQueue[DStream[MatrixEntry]]()
rddQueue += ratings
val inputStream = rddQueue.dequeue()
```

（3）流式协同过滤计算模型 CollabFilter()的调用和流数据处理程序的启动，启动程序后可以调用我们编写的函数显示预测结果和模型评估结果：

```
val model = new CollabFilter()
model+train(inputStream)
ssc.start()
model.judge() / model.predict() / model.RMSE() / …
ssc.awaitTermination()
```

流式矩阵分解模型 CollabFilterModel()的实现：

（1）首先定义了一些方便计算的数据类评分数据的表示 Rating，锁函 数 coordinate，数据块类型 PerBlock，数据块的标记 block；

```
class Rating(user: Int, product: Int, rating: Double)
class coordinate(var x: Int, var y: Int)
class PerBlock(rows: Int = m, cols: Int =n)
class block(var rIndex: Int, var cIndex: Int, bloM: Array[Array[Double]])
```

（2）对矩阵 U 和矩阵 V 进行初始化和广播操作，使 U 和 V 的更新操作可以实时同步，对锁函数进行初始化：

```
U = Array.fill(numUsers)(Array.fill(kValues)(rand.nextDouble()))
V = Array.fill(numProducts)(Array.fill(kValues)(rand.nextDouble()))
val U_br = sc.broadcast(U).value
val V_br = sc.broadcast(V).value
availableBlock = Array.fill(numAvailableBlock)(new coordinate(20, 20))
```

（3）数据块的选取函数 selectBlock()，通过 flag 函数和 availableBlock()判断该数据块的数据是否更新和是否可以选取；

```
def selectBlock(): Unit = {
var tmpU = flagU.clone()     var tmpV = flagV.clone()          var num = 0
for (t <- 0 until numPartForRow ) {
      for (j <- 0 until numPartForCol )
```

```
            if (blockMatrixFlagUpdate(t)(j) == 1 && tmpU(t) && tmpV(j) ) {
            availableBlock(num) = coordinate(t, j)
            tmpU(t) = false
            tmpV(j) = false
            blockMatrixFlagUpdate(t)(j) = 0
            num += 1}}}
```

（4）遍历选取的每个数据块中的评分数据并进行矩阵 U 和矩阵 V 的部分更新：

```
val newRDD = blockMat.filter(entry =>
    CollabFilterModel.Filter(entry, availableBlock_br)).flatMap {
    case ((blockRowIndex, blockColIndex), matrix) =>
    val elements = for (i <- 0 until rowsPerBlock_br) yield {
      val single = for (j <- 0 until colsPerBlock_br) if matrix(i)(j).toInt != 0) yield {
        val uIndex = blockRowIndex * rowsPerBlock_br + i
        val vIndex = blockColIndex * colsPerBlock_br + j
        val res = CollabFilterModel.SGD(matrix(i)(j))
        U_br(res._1) = res._3
        V_br(res._2) = res._4
        (res._1, res._2, res._3, res._4)}
    }elements.flatMap(i => i)}
```

（5）随机梯度下降的函数如下所示，函数传入的参数被省略，梯度下降的步长可以由我们设定：

```
def SGD(): (Int, Int, Array[Double], Array[Double])    = {
var dotProduct = 0.0
for (k <- 0 until kValues_br){
dotProduct += uFeatures(effrow)(k) * pFeatures(effcol)(k)}
val ratingDiff = dotProduct - rating
val uFeatures_new    = uFeatures(effrow).clone()
val pFeatures_new    = pFeatures(effcol).clone()
for (k <- 0 until kValues_br){
      val oldUserWeight = uFeatures(effrow)(k)
      val oldProWeight = pFeatures(effcol)(k)
      uFeatures_new(k) -= 2 * stepSize_br*(ratingDiff*oldProWeight +
        (lambda_br / numProducts_br) * oldUserWeight)
       pFeatures_new(k) -= 2 * stepSize_br*(ratingDiff*oldUserWeight +
       (lambda_br / numUsers_br) * oldProWeight)
}(effrow, effcol, uFeatures_new, pFeatures_new)}}
```

流式算法的推荐效果可以通过计算 RMSE 衡量，在流式应用程序中读取测试数据，每隔一段时间返回测试结果的 RMSE，RMSE 大致反映了推荐结果的准确性。一般情况下，随着时间推移系统接收到的数据总量越来越多时，推荐的效果会越来越好，这也是设计流式算法的重要意义[25]。

第五章 算法测试和结果分析

5.1 测试环境

算法的本地测试平台为：

计算机型号：MacBook Pro

CPU 型号：Intel Core i5 2.7GHz

内存：8 GB

系统：OSX EI Captitan

开发和测试环境：Intellj IDEA 15.0.5

测试用分布式集群配置：

集群服务器数量：6

操作系统： Linux Server release 6.2

CPU 型号：Intel Xeon E5-2660 2.2GHz

服务器 CPU 核数：4/台

Worker 服务器内存：30.4 GB/台

Master 服务器内存：30.0 GB

Spark 版本：Spark-1.6.0

Hadoop 版本：Hadoop-2.6.0

JDK 版本：1.8.0

Scala 版本：2.11.7

5.2 测试方法和结果分析

5.2.1 基于用户行为和物品属性的协同推荐算法

本地测试时采用 MovieLens 的一百万条电影评分数据作为用户行为数据，该数据包含 3952 部电影和 6040 个用户的信息，电影的信息数据格式是“电影 ID::电影名称和出品年份::电影类型”，电影类型共有 18 种，分别是 Action、Adventure、Animation、Children's、Comedy、Crime、Documentary、Drama、Fantasy、Film-Noir、Horror、Musical、Mystery、Romance、Sci-Fi、Thriller、War 和 Western，相应地我们统计用户行为后生成的用户特征向量也有 18 个维度[26]。

例如 ID 为 1894 的用户特征向量为：

(1894,[0.0,0.0,0.0,0.25,0.2,3.4500000000000006,1.0833333333333333,0.3333333333333333,6.666666666666666,7.616666666666665,1.3333333333333333,11.2,0.0,0.8333333333333333,0.25,0.6666666666666666,3.833333333333334,15.250000000000004])

ID 为 450 的用户特征向量为：

(450.0,[0.0,0.0,0.0,0.25,0.0,2.0,1.0833333333333333,0.0,2.5,2.1666666666666665,0.0,4.0,0.0,0.0,0.0,0.0,1.25,5.166666666666666])

应用程序计算每个用户间的相似度并向每位用户推荐 30 位用户(再这些相似用户喜欢的电影)，算法的核心步骤是相似度的计算，关于相似度的计算一共比较了余弦相似度、改进的余弦相似度和皮尔逊相关系数的两种表示这四种方法。为比较四种相似度计算方法的

效率，我们进行的测试方案为：在本地模式运行程序，执行程序的核数为 4，比较每种方法进行对 20 位用户的推荐所需的计算时间。

表 5-1 不同相似度计算方法的计算时间

相似度计算方法	余弦相似度	改进的余弦相似度	皮尔逊相关系数 1	皮尔逊相关系数 2
完成 20 位用户推荐的时间/秒	33.742	34.468	33.367	35.928

如表 5-1 所示，可以看出几种相似度计算方法的计算时间大致相同，其中皮尔逊相关系数进行相似度计算的效率最高。

我们采用皮尔逊相似度计算不同核数对计算时间的影响测试，比较采用不同核数完成 30 位用户的推荐所需时间：

表 5-2 不同核数完成相似度计算的计算时间

核数	1	2	3	4	5	6	7	8
推荐时间/秒	59.285	50.194	50.097	48.804	49.565	50.516	52.263	51.360

从表 5-2 的结果可以看出计算核数为 4 时相似度的计算时间最短，核数越多计算时间并不会越短，因为核数越多任务的划分、网络通信和 Stage 结果的读取操作也越多，所以计算总时间并不一定减少。

5.2.2 基于矩阵分解的协同推荐算法测

我们采用 MovieLens 的一百万条电影的评分数据进行测试，测试的方式是将 80%的评分数据分为训练集，20%的评分数据作为测试集，训练完矩阵分解模型后对测试集的数据进行评分预测和均方根误差、K 均值准确率的计算，结果的样例如下：

（1）预测 ID 为 1000 的用户对 ID 为 1000 的电影的评分：

PredictedRating of User 1000 to Item 1000 = 1.0680724315518517

（2）对 ID 为 1000 用户预测值最高的 10 部电影(数据格式为用户 ID，电影 ID，预测评分)：

Rating(1000,1809,6.163432015543837)
Rating(1000,1147,5.9956598079618715)
Rating(1000,911,5.976773564074044)
Rating(1000,3028,5.793992738290936)
Rating(1000,3035,5.727262643842942)
Rating(1000,213,5.713677404844961)
Rating(1000,2064,5.706082810096112)
Rating(1000,497,5.664848142473643)
Rating(1000,1197,5.587355958394658)
Rating(1000,2499,5.490345368891513)

（3）列出对应推荐列表中的电影名称(数据格式为电影名称，类型，预测评分)：

((Hana-bi (1997),Comedy|Crime|Drama),6.163432015543837)
((When We Were Kings (1996),Documentary),5.9956598079618715)
((Charade (1963),Comedy|Mystery|Romance|Thriller),5.976773564074044)
((Taming of the Shrew, The (1967),Comedy),5.793992738290936)
((Mister Roberts (1955),Comedy|Drama|War),5.727262643842942)
((Burnt By the Sun (Utomlyonnye solntsem) (1994),Drama),5.713677404844961)
((Roger & Me (1989),Comedy|Documentary),5.706082810096112)

((Much Ado About Nothing (1993),Comedy|Romance),5.664848142473643)
((Princess Bride(1987), Action|Adventure|Comedy|Romance),5.58735595839465)
((God Said 'Ha!' (1998),Comedy),5.490345368891513)

（4）ID 为 1000 的用户评分最高的电影(数据格式为电影名称，类型，用户评分):

((Terminator 2: Judgment Day (1991),Action|Sci-Fi|Thriller),5.0)
((Toy Story (1995),Animation|Children's|Comedy),5.0)
((Beauty and the Beast (1991),Animation|Children's|Musical),5.0)
((Indiana Jones and the Last Crusade (1989),Action|Adventure),5.0)
((Verdict, The (1982),Drama),5.0)
((Hunt for Red October, The (1990),Action|Thriller),5.0)
((Night to Remember, A (1958),Action|Drama),5.0)
((Thelma & Louise (1991),Action|Drama),5.0)
((Star Wars:Episode New Hope(1977),Action|Adventure|Fantasy|Sci-Fi),5.0)
((Streetcar Named Desire, A (1951),Drama),5.0)

（5）对 ID 为 1000 的电影计算和其相似对最高的 10 部电影(数据格式为电影 ID，相似度):

(1000,1.0000000000000002)
(2935,0.7276557890970585)
(3132,0.7184516447300241)
(920,0.7121610121956142)
(1251,0.7109236645656659)
(3747,0.708729402257763)
(3634,0.7074973825305609)
(922,0.7070078545183339)
(1237,0.7066920378406563)
(2732,0.7059564754876478)

（6）和 ID 为 1000 的电影相似度最高的电影(数据格式为电影名称，相似度):

(Curdled (1996),Crime, 1.0000000000000002)
((Lady Eve, The (1941),Comedy|Romance),0.7276557890970585)
((Daddy Long Legs (1919),Comedy),0.7184516447300241)
((Gone with the Wind (1939),Drama|Romance|War),0.7121610121956142)
((8 1/2 (1963),Drama),0.7109236645656659)
((Jesus' Son (1999),Drama),0.708729402257763)
((Seven Days in May (1964),Thriller),0.7074973825305609)
((Sunset Blvd.(a.k.a. Sunset Boulevard)(1950),Film-Noir),0.70700785451839)
((Seventh Seal, The (Sjunde inseglet, Det) (1957),Drama),0.7066920378406563)
((Jules and Jim (Jules et Jim) (1961),Drama),0.7059564754876478)
((To Kill a Mockingbird (1962),Drama),0.7024426175558592)

（7）测试集的均方误差、均方根误差和 K 均值准确率:

Mean Squared Error = 0.2931497239577046
Root Mean Squared Error = 0.5414330281370953
Mean Average Precision at K = 0.0168432079785557

矩阵分解模型的性能和分解低维矩阵的低维维度设置及最小二乘法进行目标函数值优化的迭代次数有关，我们分别对这两个参数进行测试.

对低维矩阵的维度参数的测试方法是固定迭代次数为 10，设置维度参数分别为 10、20、30、40、50、60、70、80，计算模型预测测试集数据的均方误差和 K 均值准确率的值，测试结果如图 5-1 所示。

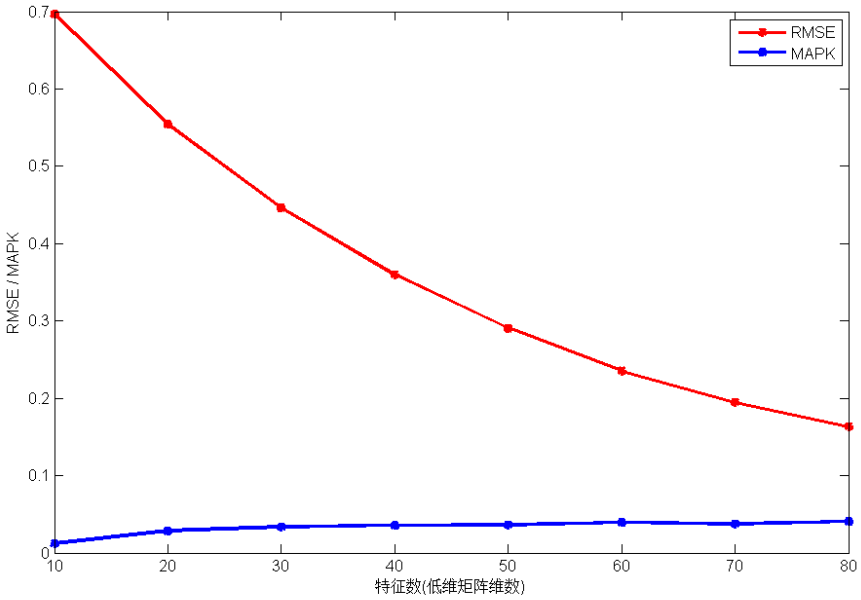

图 5-1 低维矩阵维度和均方误差及 K 均值准确率的关系图

从图 5-1 可以看出随着低维矩阵维度的增加，测试集的 RMSE 下降，MAPK 增大，说明设置的隐含特征数越多，预测结果越准确，但特征数目越多，相应地模型计算量也越大，一般来说设置矩阵维度在 50-60 之间最合适。

对迭代次数参数的测试方法是低维矩阵维度为 10，设置最小二乘法迭代次数分别为 4、6、8、10、12、14、16、18、20，计算模型预测测试集数据的均方误差和 K 均值准确率的值，测试结果如图 5-2 所示。

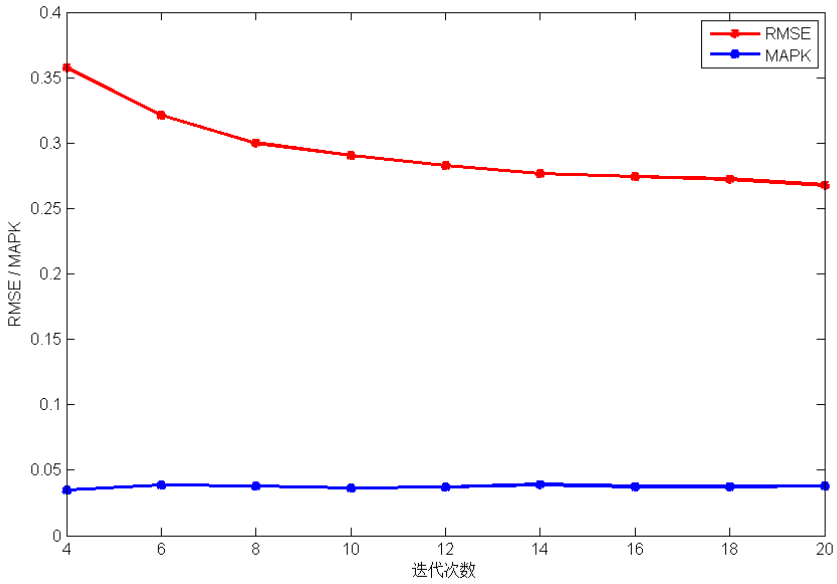

图 5-2 最小二乘法迭代次数和均方误差及 K 均值准确率的关系图

从图 5-2 可以看出随着最小二乘法迭代次数的增加，测试集的 RMSE 下降，MAPK 增

大，说明迭代计算次数越多，预测结果越准确，但迭代次数越多，相应地模型计算时间越长，迭代次数到一定值后增加迭代次数对结果的改善程度就渐渐变弱，从图中可以看出设置最小二乘法迭代次数在 10-15 次之间最合适。

5.2.3 基于 K 均值聚类的推荐算法的测试

聚类算法的测试方法是用 MovieLens 的一百万条电影的评分数据和电影信息数据作为用户的行为数据统计的数据源，提取用户特征并进行正则化，然后进行用户聚类，设置 10 个聚类类别，结果显示了每个类别的 10 位用户。聚类结果附加了用户的职业信息，职业种类共有 21 种，分别是 other or not specified、academic/educator、artist、clerical/admin、college/grad student、customer service、doctor/health care、executive/managerial、farmer、homemaker、K-12 student、lawyer、programmer、retired、sales/marketing、scientist、self-employed、technician/engineer、tradesman/craftsman、unemployed、writer。测试结果如下：

类别 1： (M,50,Scientist,52.40582638287887)
(M,25,Executive/Managerial,10.250128741502754)
(M,35,Programmer,79.16871238268708)
(F,35,Unemployed,39.33245326551578)
(M,56,Writer,63.12421228796569)
(M,25,Programmer,52.104433941844476)
(M,25,Sales/Marketing,79.16871238268708)
(M,25,Technician/Engineer,59.46683803391619)
(M,50,Other or not specified,4.987795692340949)
(M,25,Programmer,4.8223781968398685)

类别 2： (F,1,Other or not specified,3.7003681141757765)
(F,25,Other or not specified,5.589724471105057)
(F,25,College/Grad student,6.984200716096401)
(M,18,College/Grad student,3.7003681141757765)
(F,18,Technician/Engineer,4.695248226113707)
(M,18,Technician/Engineer,2.015199889282433)
(M,18,College/Grad student,2.015199889282433)
(F,25,Academic/Educator,0.9556883456838785)
(M,25,Writer,0.9556883456838785)
(M,25,Other or not specified,6.984200716096401)

类别 3： (F,45,Clerical/Admin,0.9064170152047551)
(F,25,Clerical/Admin,0.9064170152047551)
(F,35,Academic/Educator,5.690082847556626)
(M,45,Academic/Educator,1.8041938114356477)
(F,25,Artist,8.535014384658764)
(M,18,College/Grad student,1.564581031694773)
(M,25,College/Grad student,3.7993461965946835)
(F,56,Doctor/health care,8.535014384658764)
(F,56,Doctor/health care,1.2672213929038887)
(M,18,College/Grad student,1.295563560159483)

类别 4：(M,25,Programmer,0.5088254094205316)
(F,1,K-12 student,3.165116489232812)
(M,18,Sales/Marketing,2.603426348387728)

(M,25,Other or not specified,7.330186911767836)
(F,25,Homemaker,0.3750225925191404)
(M,18,Scientist,0.6010789305472714)
(M,50,Lawyer,2.162816019749213)
(M,25,Lawyer,3.165116489232812)
(M,25,Sales/Marketing,4.731313672331421)
(F,1,K-12 student,2.603426348387728)

类别 5： (F,56,Executive/Managerial,38.62855322553736)
(M,25,Self-employed,29.220238918079204)
(F,25,Sales/Marketing,6.610866771960582)
(M,25,Other or not specified,11.989576817622691)
(M,18,Artist,25.936905584745872)
(M,18,Sales/Marketing,26.104713803923957)
(M,18,Writer,18.740862966785443)
(M,56,Unemployed,2.02549005963178)
(M,35,Retired,25.936905584745872)
(M,56,Self-employed,58.55298244928166)

从聚类的结果可以看出，类别 1 用户的特点是以男性为主，年龄偏大；类别 2 用户的特点是年龄偏小，学生、技术人员和不确定的职业居多；类别 3 用户的特点是以女性为主，学生、教育工作者、医护人员居多；类别 4 用户群体主要有是律师、商业人员、高中生；类别 5 的用户特点是商业人员和自由职业者居多。通过聚类结果的用户信息的角度看来，聚类算法对用户群体做了大致的划分，因为聚类的依据主要是用户对电影的一系列为，所以职业信息只是分类结果的粗略参考。

因为聚类的类别由我们指定，类别的数目对聚类效果会产生影响，我们通过设置不同个数的聚类中心点来比较所有类簇中点的方差之和 WCSS 的变化情况，测试结果如图 5-3 所示。

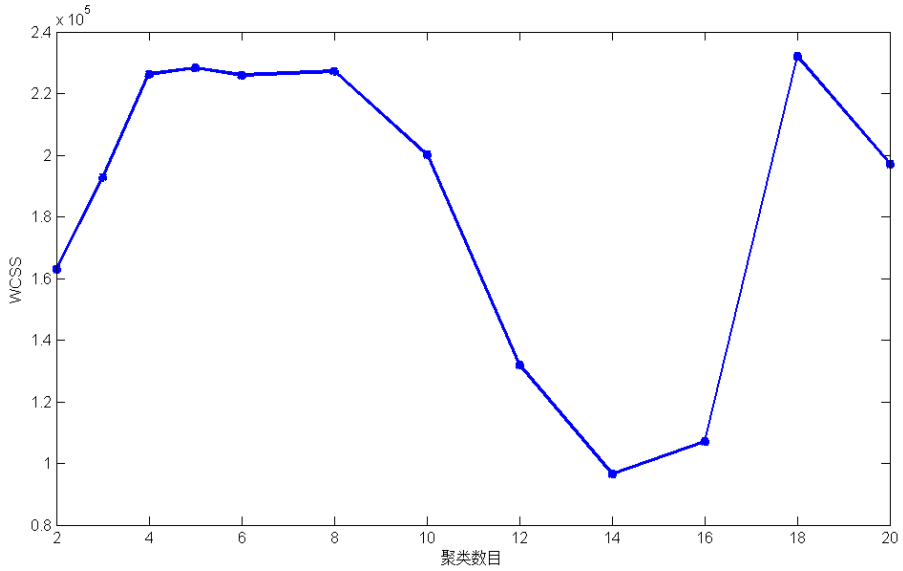

图 5-3 聚类数目和类簇 WCSS 的关系图

从图 5-3 可以看出 WCSS 值随聚类数目的增加先减少后增大，当类别数目在 12-16 间 WCSS 的值最小，说明聚类类别数目取在 12-16 间最合适。

5.2.4 基于关系网络和图模型的推荐算法测试

对于关系网络的图模型的测试测试主要我们用IEEE的论文引用关系的数据进行算法测试，当我们用 10000 篇论文的引用关系数据时，测试结果如下：

（1）论文网络中的节点重要性排序(重要性最大的 10 个节点 ID 和重要性值)：

1775749144: 14.729606980745308；　　1639032689: 8.014975654181024；
2128635872: 7.005844346020679；　　2024556021: 6.186568452380956；
2137358449: 6.106844191340468；　　2100837269: 5.984574101828234；
2140956556: 5.273203986603688；　　1594031697: 5.244791794825772；
1847168837: 4.673831892807713；　　2024509488: 4.639815427910413；

（2）入度最大的 10 个节点 ID 和入度值：

1775749144: 109；　　2061977616: 99；
2128635872: 95；　　2100837269: 84；
1639032689: 73；　　20179835: 66；
2097706568: 65；　　1987258130: 64；
2116199508: 64；　　2108795964: 63；

当我们用 100000 篇论文的引用关系数据时，测试结果如下：

（1）论文网络中的节点重要性排序(重要性最大的 10 个节点 ID 和重要性值)：

1775749144: 278.64086398476525；　　2100837269: 130.18671532783523；
2128635872: 122.7838678997339；　　1506043065: 66.43817109712019；
2060333964: 63.034304954631565；　　1625660792: 49.55180356211482；
2111301002: 49.16557460434511；　　2144634347: 46.65390252272484；
2123977424: 46.180410439528664；　　2106882534: 45.79297581853894；

（2）入度最大的 10 个节点 ID 和入度值：

1775749144: 2381；　　2100837269: 1924；
2128635872: 1856；　　2106882534: 840；
2143981217: 828；　　2097706568: 746；
2028556299: 716；　　2107277218: 708；
2144634347: 701；　　144423133: 647；

从中可以看出用 10000 篇论文的数据集测试时，ID 为 1775749144、1639032689、2128635872、2100837269 的节点在两个结果中都出现；用 10000 篇论文的数据集测试时，ID 为 1775749144、2100837269、2128635872、2144634347、2106882534 的节点在两个结果中都出现。说明入度越大的节点重要性很可能也越大，对应一般越有价值论文被引用的次数会越多，但是也不完全如此，还要考虑引用网络中节点和相邻节点的关系，证明重要性的计算算法是有效的。

对于节点引用关系的树的建立,我们采用 10000 篇论文的引用关系数据的测试，结果的样例如下：

（1）对 ID 为 985934603 的节点建立的树：

(985934603, Map(32122371 -> 0, 113873023 -> 0, 2092713296 -> 0, 55431777 -> 0, 985934603 -> 2, 2059287674 -> 1))

这是节点 985934603 的层数为 2 的一课引用树，Map()中每个映射 2 的节点和节点 985934603 的距离为 1，每个映射 1 的节点和节点 985934603 的距离为 2，每个映射 1 的节点和节点 985934603 的距离为 3。

（2）找出和节点 985934603 距离为 3 的节点：

(985934603,Set(32122371, 113873023, 2092713296, 55431777))

（3）通过节点重要性进行节点推荐的排序：

((2092713296,0.6643258928571429),(32122371,0.15),(113873023,0.15),null)

因为 ID 为 2092713296 的节点的重要性较大，所以作为优先推荐给节点 985934603 的候选集，55431777 节点显示为 null 因为其重要性太低可暂时忽略。

此外，我们对引用关系的树建立的时间进行了测试，比较用 10000 篇论文的引用关系数据集对所有进论文节点建立 2 层、3 层、4 层的树所需的时间，测试环境是 5 个内存都为 4G 的 Worker，CPU 总数 20 的集群，测试结果如下：

表 5-3 不同层数的树建立和程序运行时间

层数	2	3	4
树建立时间/秒	16.150	46.650	97.764
程序运行时间/分	1.3	1.7	4.8

5.2.5 流式推荐算法测试

对于流式算法来说，算法的计算结果可能由于接受到的数据量太少而不理想。但随着时间的推移接收到越来越多的数据后，算法的效果会不断改善，这也是我们希望达到的目标。

本文第四章设计的流式协同过滤算法进行目标函数结果优化的方法是随机梯度下降，随机梯度下降根据真实的数据值进行预测结果的调整，计算结果很大程度上依赖于真实数据量的多少，随机梯度下降调整步长的选取也会对计算结果产生较大影响。

为了测试数据量对流式推荐算法推荐结果的影响，我们的测试方案采用 MovieLens 的电影评分数据集，将 MovieLens 的电影评分数据分为训练集和测试集，训练集占 80%，其余 20%为测试集，流应用程序的时间间隔设定为 10 秒，训练集作为数据流的数据源分别以每秒 3、5、8、10 条的速度发送，推荐的结果的准确率可通过计算测试集的均方根误差得到，测试集的均方根误差 RMSE 如图 5-4 所示。

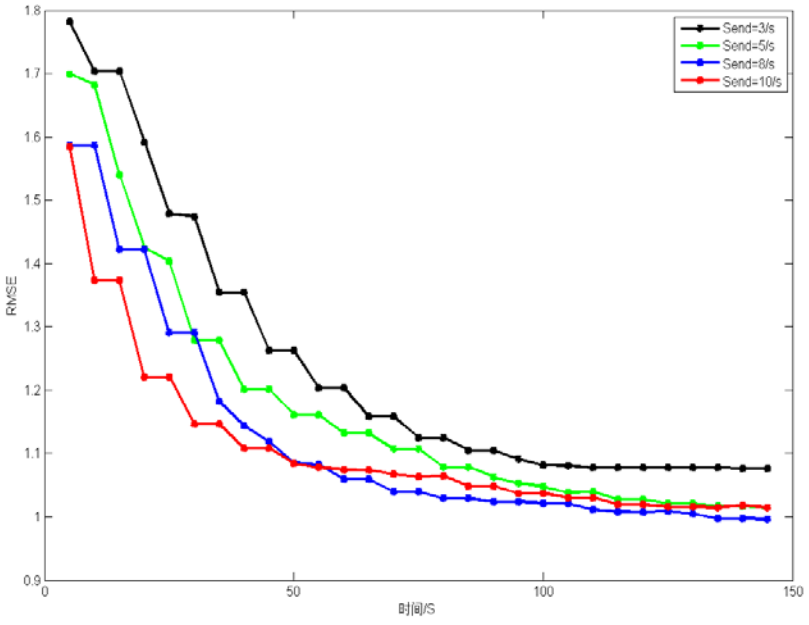

图 5-4 不同数据流量对应的推荐准确率和时间关系图

从图 5-4 可以看出，当应用程序的时间窗长度相同时，协同推荐的推荐准确率与单位时间的数据流大小有关。四条折线表示评分信息数据流为 3 条/秒、5 条/秒、8 条/秒、10 条/秒时测试集的 RMSE 随时间的变化。

因为 RMSE 越小对应推荐结果的准确度越高，所以四条折线的下降趋势证明流式算法的有效性。对于每条 RMSE 折线，在起始阶段，当数据流的数据量越大 RMSE 的下降速度

越快，说明接收到越多信息矩阵分解模型的低维矩阵更新越快，通过分解矩阵计算得到的评分值也更接近真实值。

经过一定时间后，每条折线的变化趋于平缓，对应数据流量越大的 RMSE 折线的 RMSE 值总体是越小的，但数据流为 10 条/秒对应的折线的下降速度和 RMSE 都没有数据流为 8 条/秒对应的折线的理想，这种情况可能是由于数据流的数据量太大导致流应用程序没有充分利用所有数据的信息，这种情况可以通过调整时间窗长度得到更好的结果。

对随机梯度下降的调整步长的选取，我们进行的测试时步长为 0.005、0.01、0.05 的矩阵分解模型对测试集的预测结果的均方根误差 RMSE 随时间的变化，测试结果如图 5-5 所示，测试中还加入了训练集的 RMSE 值作为对比。

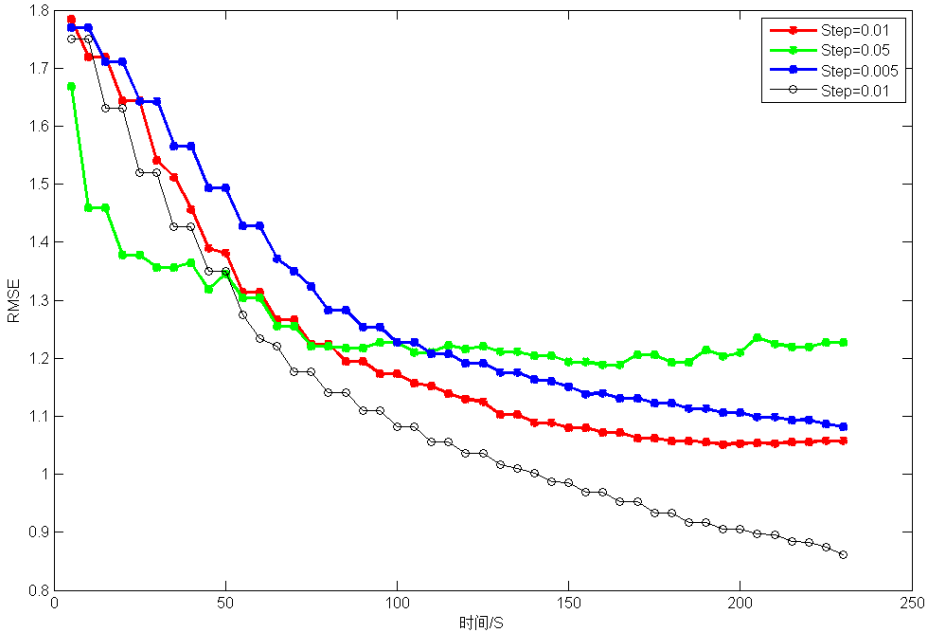

图 5-5 不同步长的随机梯度下降对应的推荐准确率和时间关系图

从测试结果中可以看出当梯度下降的调整步长为 0.05 时，在起始阶段 RMSE 下降速度很快，但一定时间后下降停滞甚至有变化不定的趋势，说明步长太大可能会导致计算结果偏离最优值；当步长为 0.005 时，梯度下降的速度较慢，和步长 0.01 的折线相比达到相同结果所需的时间更长；步长为 0.01 的梯度下降是较为合适的最优化方法，在起始阶段的下降速度接近了训练集的 RMSE 的梯度下降速度，随着时间的推移，推荐准确率也基本接近最优值。

5.3 结果优化的构想

本文设计的推荐算法的推荐结果还未能达到理想状态，推荐效果随着数据量的增加会有波动，本文针对算法实现过程中遇到的问题提出了几种优化推荐结果的构想。

收集更多数据集，进行用户和物品特征的扩充。本文实现的基于用户行为的特征的算法是基于 MovieLens 的电影评分数据实现的，这个数据集只提供用户的观影信息，无法反映收藏、点赞、搜索等行为，这样提取用户的特征有限，对于用户相似度的计算和聚类不能达到最好的效果。

进行关系网络建立时增大数据量常常会遇到内存不足无法正常计算的问题。内存不足的原因是每个分区中的数据量太大，解决的办法可以是增加分区的数量来减少每个分区的数据

量。但由于测试用集群的核数限制，增加分区导致通信和任务列表的信息增加，内存问题无法解决。对于这个问题的优化构想是用核数更多的集群来扩大分区的容量以解决内存不足的问题。

流式算法的梯度下降步长对算法效率结果准确率的影响较大，可以设置步长的自适应调整算法，让步长在每一阶段以最理想的值更高效地更新计算结果。流式算法根据预先的估测值设计评分矩阵大小，这样评分矩阵最终会饱和，无法提供长时间的运算，对于这个问题的解决办法可以是建立一个超大的可分块矩阵，这个矩阵可以随时提供新的空间容纳数据。

本文设计的推荐系统的不同算法还是以独立运行为主，未能很好地组合，一方面由于各个算法使用的数据集不统一，一方面由于算法的接口还没有实现。优化构想是用一个信息比较完整的数据集进行推荐算法接口格式的统一，并完成推荐系统的用户界面设计。

第六章 结论

本课题设计了基于 Spark 平台的推荐系统的实现方案并实现了多种推荐算法。Spark 中的推荐系统充分利用了 Spark 处理大数据的一系列优势进行大数据场景下数据的并行处理和并行推荐算法的分布式实现。本文设计和实现的内容主要有：

（1）对 Spark 架构和编程语言 Scala 进行深入理解和实践；

（2）掌握 Spark 生态系统中各个模块的库的运用和 Spark 内核源码的阅读；

（3）学习多种机器学习算法和数据分析和挖掘的相关技术；

（4）基于用户行为和物品属性的协同推荐方案的设计和实现；

（5）基于矩阵分解的用户和物品的协同推荐算法的实现和参数选取的研究；

（6）基于 K 均值聚类算法的协同过滤算法的实现和结果分析；

（7）基于关系网络的图模型建立和社会化推荐的设计和实现；

（8）流式的矩阵分解推荐算法的设计、实现和优化的研究。

本课题的特点和难点主要有：

（1）用高效的大数据分布式处理架构 Spark 实现并行算法，与传统的大数据处理工具相比具有较大的性能优势；

（2）系统的实现涉及 Hadoop、HDFS、Linux、Spark 等多种系统和平台的相关知识和操作，实现需要大量的知识基础和实践；

（3）运用函数式编程语言 Scala 实现算法，使应用程序更简洁，但函数式编程语言不同于传统的编程语言，在运用和程序优化上也更有难度；

（4）充分发现有限的数据集的信息，利用并不完整信息设计和改进推荐方案；

（5）对实现的算法进行单机和分布式测试，算法性能的调优需要大量的对照实验和数据统计。

本课题最终实现了本文介绍的各种不同场景下的推荐算法，是对大数据时代的数据处理和信息发现的一个探索。在数据过量的情况下信息推荐的意义和实现的技巧必然会越来越重要。借助于专业的大数据处理工具将成为今后大数据技术的发展方向。未来系统还可以进行算法改进、应用程序接口统一、添加系统用户界面等完善，争取在当前热门的智能推荐领域发挥更大作用。

参考文献

[1] 查礼,程学旗. 天玑大数据引擎及其应用[J]. 集成技术,2014(4):18-30.
[2] 陈伟.《促进大数据发展行动纲要》解读[J]. 中国信息化,2015(10):11-14.
[3] 项亮. 推荐系统实践[M]. 北京:人民邮电出版社,2012:12-17.
[4] 吴晓黎. 基于数据挖掘的个性化营销算法的设计与实现[D]. 北京:北京邮电大学,2010.
[5] 曾怡,吕慧,朱江楠. 基于协同过滤推荐技术的作业资源个性化推荐系统的设计与研究[J]. 电子世界,2013(11):107-108.
[6] Hadoop home page. http://hadoop.apache.org/.
[7] Apache Storm home page. http://storm.apache.org/.
[8] Apache Spark home page. http://spark.apache.org/.
[9] 董喜梅,白振轩,陈勤,李余. 基于聚类模式的推荐算法研究[J].系统仿真技术,2011(1):43-47.
[10] 付惠惠.一种分布式存储管理原型系统客户端软件的设计与实现[D].北京:北京交通大学,2011.
[11] 彭石.基于用户兴趣和项目特性的协同过滤推荐算法研究[D].中南大学,2012.
[12] 杨志伟.基于Spark平台推荐系统研究[D].中国科学技术大学,2015.
[13] Nick Pentreath. Machine Learning with Spark[M]. Packt Publishing Ltd, 2015:109-112
[14] Fuzhi ZHANG. A Two-stage Recommendation Algorithm Based on K-means Clustering In Mobile E-commerce[J]. Computational Information Systems 2010(6:10):3327-3334.
[15] R.Thiyagarajan. Recommendation of Web Pages using Weighted KMeans Clustering[J]. International Journal of Computer Applications, 2014(1):45-48.
[16] Grzegorz Malewicz. Pregel: A System for Large-Scale Graph Processing[C].SIGMOD'10, 2010.
[17] Apache Spark repository on Github. https://github.com/apache/spark
[18] Holden Karau. Learning Spark: Lightning-Fast Big Data Analysis[M].O'Reilly Media,2015:65-68
[19] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, Ion Stoica. Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. University of California, Berkeley, 2012.
[20] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica. Discretized Streams: Fault-Tolerant Streaming Computation at Scale. University of California, Berkeley, 2013.
[21] Muqeet Ali, Christopher C and Johnson Alex K. Tang. Parallel Collaborative Filtering for Streaming Data. Dec. 2011.
[22] N. Marz. Trident: a high-level abstraction for realtime computation. http://engineering.twitter.com/2012/08/tridenthigh-level-abstraction-for.html.
[23] Marz N.Twitter'S Storm: Distributed Real-Time Computation System. The Apache Software Foundation[J]. 2014.
[24] Sumit Gupta. Learning Real-time Processing with Spark Streaming[M]. Packt Publishing Ltd, 2015.

[25] Pawel Matuszyk, Myra Spiliopoulou. Selective Forgetting for Incremental Matrix Factorization in Recommender Systems[C]. Discovery Science, 2014:204-215.

[26] MovieLens Datasets. http://grouplens.org/datasets/movielens/

谢辞

完成了毕业设计后，我的大学生涯也即将结束。在此我要向所有给予我指导、帮助和关照的老师和同学们表示最衷心的感谢。

首先要感谢我在实验室的导师王新兵老师，在王老师的指导下我在智能物联网研究中心接触学习了我感兴趣的大数据领域的前沿知识和技术。作为一名电子工程系信息工程的学生，我对物联网和大数据技术很感兴趣，阅读了很多相关的文献，但一直没有找到一个比较好的实践机会，直到加入智能物联网研究中心。在实验室里，不仅有老师的不断鼓励和大力支持，有良好的实验设备和实验环境，还有很多优秀的学长学姐传授学习经验，提供宝贵的指导和帮助。在过去的半年里，王老师对毕业设计从选题到课题进展的各个阶段都十分关心，也让学长们给了我很多重要的指导性意见。从毕业设计的开题到系统的方案的设计实现到论文的撰写，我曾经遇到过许多困难，最终在大家的帮助和自己的努力下顺利完成。我要感谢孟桂娥老师对我的毕业设计各个阶段的指导意见，孟老师是一位认真负责的老师，对我的问题总是能够及时回复。我要感谢 Spark 组的姚春楠学长、丁舟奕海学长和陈戈学长，在我实现 Spark 应用程序的遇到技术难题时给我提供了许多帮助，为我讲解了很多算法实现的技巧和需要注意的问题，让我在入门 Spark 时少走了一些弯路。感谢组里的各位同学，不论是在搭建、使用集群，还是编写代码和文论撰写上我都得到了各位的许多宝贵的建议，大家在相互讨论、思维碰撞的同时也加深了对知识的掌握和理解，学习氛围由此变得更加浓厚。我非常清楚，如果是孤军奋战的话，毕业设计这一系列过程都会变得更艰难。

我要感谢大学里的我的每一位老师，你们平易近人的态度和朴实的作风营造了大学良好的教学氛围。感谢电子系的宫新保老师，您严谨的治学态度和循循善诱的教学方法让我打下了扎实的专业学习的基础；电子系的蒋乐天老师，您耐心地为我解答电子技术的许多问题，并且热心地担任我的出国申请的推荐人；电子系的徐奕老师，您生动地授课方式让我对图像处理技术有了很好的掌握，对于课程设计的指导也让我受益匪浅；电子系的袁焱老师、杨宇红老师，在你们的指导下我完成了很多理论的实践，对通信和电子电路的知识有了更深的认识。还有众多未提及的恩师，你们的教导是我大学中最值得的体验，我将铭记于心。

感谢我的班级同学和室友们，没有你们对我生活上的支持和帮助，大学的生活也无法丰富多彩。曾经，我们在同一个课堂、同一个自习室并肩作战，认真学习；我们在假期时运动、旅行，谈人生的理想，做对未来的展望。你们是我大学生活的重要的一部分，也是我人生旅途上的挚友。

感谢我的父母，一直以来你们的默默支持都是我的勇气和动力的重要来源。虽然往往一学期也难得回几次家，但是你们对我的爱纵然相隔万里我也能感受到。岁月在你们的额头留下皱纹，也让我不断成长，懂得要更加努力，报答你们对我无私的爱。

一路走来，有太多美好的回忆和说不完的感谢。感谢上海交通大学这个承载着无数人梦想和希望的学校，让我成长得更快，看得更远。也许那些在教室认真听课的时光、那些在田径场上挥洒汗水的时光、那些参加集体嘹亮歌唱的日子、那些在思源湖畔朗读英语的日子已经远去、那些在实验室废寝忘食，但是我所收获的一点一滴都将伴随着我去迎接未来更大的挑战。希望在不远的将来，我的一切付出也能值得自己感谢。

IMPLEMENTATION OF RECOMMENDER SYSTEM BASED ON APACHE SPARK

The fast development of Internet technology has created and spread more and more data every day, making Big Data technology one of the most popular research areas nowadays. To process massive datasets, we can rely on some regular tools because of the large volume and various types of Big Data. According to IDC, the total amount of data of the world doubles each year, we will have up to 35ZB data by 2020. What's more, countless mobile devices and small processors help us raise the speed of data processing and realize real-time data processing. There is usually much valuable information in the massive datasets, which explains why we spend lots of time and money doing data analysis, although the density of value among our datasets is often very low. And to make real use of massive datasets we need to design efficient intelligent algorithms to do data processing.

The potential market of Big Data Technology is obvious. In 2012, America government and Japan government both launched some plans to develop data technology, Chinese government also make policy to make use of data resources last year. Apart from the investment of some governments, many big companies also head for Big Data areas. We see Oracle, IBM, Microsoft, Intel all produce some products and solutions for big data. At the same time, some big data processing technologies and open source programs have caught our attentions and gradually became the main stream, such as MapReduce, Hadoop, Strom, Spark, HBase and so on. Many famous Internet company like Baidu, Tencent, Alibaba all set up Big Data department for the research of Big Data technology.

There are many examples of Big Data applications for different cases. Taobao use the Streaming module of the distributed computing architecture Spark to process real time transaction data streams and user data streams in order for faster and more precise data analysis and prediction. Taobao also use GraphX module of Spark for construction of large graph of trading information, community detection and relationship measuring. Some video websites like Youku use Hadoop, Spark for video recommendation and advertisements management. IBM develop software for real time data analysis to help Tennis Australia collecting some information such as game schedule, the situation of athletes, the history records of games and the requirements of audiences for the improvement of services. In a word, Big Data technology is used in all walks of life, even changing the operation mode of many enterprises, becoming an important driver of economic growth and development of technology.

One of the problems caused by Big Data is that we find it hard to search for the right information we need when we come across massive amount of data online. Even we got some powerful search engine like Google, we still can't solve the problem. That's why these days many website will recommend items for its users to assist them get what they want quickly. Examples like Netflix Movie Recommender System, Amazon commodity recommender system, Facebook friend recommender system make recommendation from different perspectives. Generally the

operating principle of a recommender system is that the system trains a recommender model with input data and then make prediction. The input data contains items information, user information, rating information and relation information and so on. The core of a recommender system is its recommendation algorithms or recommendation solutions.

There are some common kinds of recommendation algorithms or solutions. We can make recommendation based on demography theory, which is a basic recommendation solution. By computing the similarity among different users, we can find users of similar taste for a user, then recommend the items of those similar users for our target user. Usually we use the user features such as gender, age, hobbies and so on, then we make a feature vector to represent each user, by computing the similarity of two vectors we can measure the similarity of two corresponding users. This solution is suitable for the cold start of a recommender system, however, we often find it hard to collect personal information of users. We can make recommendation based on content, the content here means item contents. By computing the similarity among items, we group similarity and make recommendation for users according to their record. The accurate of this method is higher, but it is often useless to new users. We can make recommendation using collaborative filtering models. Collaborative filtering is actually another way of similarity computing. By analyzing the rating data of users, we can know if there if some similarity between two users, then predict the unknown ratings of users to items. We can also make recommendation using relation data or location data in some cases.

If we design a recommendation solution, how do we implement it? There are some Big Data processing tools for us to choose, among which Hadoop, Storm, Spark are mostly used. Hadoop is based on MapReduce framework, Storm is a streaming data processing tools which has a similar architecture as Hadoop, Spark is a faster data processing engine compare to Hadoop and Strom for its memory based computing developed by UCB AMP Lab. We choose Spark because its great performance in cases of large datasets. Besides, Spark offers many modules and API for different applications. There are Spark Streaming module for data streams processing, Spark SQL for data query, Spark MLlib for calling of machine learning algorithms, GraphX for large graph processing, Spark R for mathematical calculation.

This paper implement a recommender system based on Spark, this system contains four different recommendation solutions and a streaming recommendation algorithm. The recommendation solutions are solution based on user behavioral data and item features, solution based on matrix factorization, solution based on K-means clustering, solution based on relation data and social recommendation. The streaming recommendation algorithm is based on matrix factorization.

The solution based on user behavioral data and item features is that we create a vector for each user by collecting the user behavior to items such as search, collect, follow and so on, then we map item features to users, after that we compute the similarity of between each user and recommend the most similar users to the target user.

The solution based on matrix factorization is that we use rating dataset and build a matrix for rating data, then we make matrix factorization by factor the original rating matrix to two low rank matrix , in this way we decrease computing volume and can make prediction of unknown rating. We use the alternating least squares method to optimize low rank matrix. The result of matrix factorization can be evaluated by computing RMSE and MAPK of test dataset.

The solution based on K-means clustering is that we use user feature vector or item feature

vector to cluster users or items. K-means model is a classic classification model. First we initializing some cluster centers, then we computing the distance between each vector and center, then we assign a center for each vector, after that we reassign the cluster centers. We make iteration until the result converge to a final value.

The solution based on relation data and social recommendation is that we build a graph of paper citation network and then make recommendation for each paper node according to relationship. First we compute the importance of each paper node in the citation network in a method similar to PageRank, then we build a citation tree for each paper node, the height of the citation tree can be defined by us. The nodes in the citation tree are somewhat related to the source nodes, so by ranking the nodes in the citation tree, we can make recommendation for source paper node.

The streaming recommendation algorithm is based on Spark Streaming, we first creating a data stream using TCP socket. The core of streaming algorithm is also collaborative filtering, we make matrix factorization for a divided matrix, the optimization method is distributed stochastic gradient descent. By changing the step size and iteration times of SGD we can get the most suitable parameter for this model.

This paper mainly makes contribution to the following tasks:

The analysis of Spark framework and its programming model RDD, how spark works on server cluster and the basic use of Spark modules.

Compare different recommendation solutions and big data processing method.

Design and implementation of the recommendation solution based on user behavioral data and item features, including the research of feature collecting.

Design and implementation of a recommendation solution based on matrix factorization.

Design and implementation of a recommendation solution based on K-means clustering, including parameter choosing.

Design and implementation of a recommendation solution based on relation data and social recommendation.

Design and implementation of the streaming recommendation algorithm is based on matrix factorization.

The recommender system implemented in this paper contains many algorithms for different datasets and cases. Although we haven't connected each algorithm together and make a whole system work together, each part of the system can work well as we describe in the paper. The future work of this paper is to normalize the interface of each algorithm and make better GUI for better use.