中国科学技术大学

学士学位论文

基于 Spark 平台的推荐系统设计与实现

作者姓名：　　沈国栋

学科专业：　信息安全专业

导师姓名：　沈耀 薛开平

完成时间：二〇一六年五月

University of Science and Technology of China

A dissertation for bachelor's degree

Design and Implementation of Recommender Systems Based on Spark Platform

Author's Name: Guodong Shen

Speciality: Information Security

Supervisor: Yao Shen, Kaiping Xue

Finished Time: May, 2016

致　谢

在毕业设计的完成过程中，我有幸得到了两位老师的指导，他们是：上海交通大学计算机科学与工程系沈耀副教授，中国科学技术大学信息安全专业薛开平副教授。两位老师深厚的学术功底，严谨的工作态度和敏锐的科学洞察力使我受益良多。衷心感谢他们给予我的悉心教导和热情帮助。

感谢科大，大学四年中我收获颇丰，感谢四年里教授过我课程的所有老师。

感谢夏国卿师兄和陈静师姐对于选题与方向上的指导，感谢陈健同学对于Spark平台的测试工作，感谢洪佳楠师兄、李威师兄，张祥师兄、薛颖杰师姐、李少华同学、陈炜铿同学的帮助。

感谢陪伴我四年的室友们，是你们在我疲倦怠惰的时候给予我鼓励和勇气，我们共同进步，度过了人生中重要的四年。

还有，感谢我的父母和亲人，感谢他们在我生命中的每一刻，对于我的爱与支持。

最后，感谢本文参考文献涉及到的所有学者们。

目 录

摘　要

现代互联网的高速发展促进了信息的产生与交流，目前人们面临着数据的总量越来越大，数据的产生速度越来越快的问题。与此同时，人们认识到数据作为无形的资产，在当今世界及其重要，做好数据的收集与处理，对于企业的发展而言意义重大。在这样的背景下，大数据平台的研究成为了当代的热点研究项目。随着 Google 提出 MapReduce 计算模式，Hadoop 项目对于 MapReduce 模式进行开源实现之后，MapReduce 并行计算模式受到了广泛的采用，而 Spark 平台的诞生，通过引入 RDD 和 DataFrame 数据模型以及基于内存的计算方式，在计算速度上，要明显在数量级上优于 Hadoop 平台的 MapReduce，因而更加适应大数据背景下的多迭代计算场景。因此，Spark 平台自提出之后，一直受到工业界和学术界的关注，成为了研究重点。

随着信息的加快流通，经济全球化的发展，当前的人们在消费商品时所面临的商品种类达到了前所未有的数量。随着电商的兴起，网购行为的不断普及，电商平台对于商品的推荐极大地影响了商家的收益和买家的最终购买行为；随着音乐电影等多媒体资源数字化的发展，人们在进行文化消费时也会面临纷杂的选项。当面临用户数的不断增多和商品种类的不断增加过程中，如何通过给用户推荐喜欢程度较高，有更大消费可能的商品，成为了许多企业发展所面临的一大难题，在这一背景下，推动了人们对于推荐系统的研究。

本文主要研究在推荐系统中协同过滤算法的可扩展性问题。协同过滤算法不利用商品的自身属性，仅仅需要用户对于商品的评分就可以对用户推荐其可能喜欢的商品。本文主要对协同过滤算法大类下的两种不同算法在 Spark 平台上进行了研究分析，实现了基于交替最小二乘法的隐式因子模型推荐算法，和基于邻域关系中的用户-用户相似关系与物品-物品之间的相似关系的推荐算法，同时对交替最小二乘法和邻域关系的推荐算法的并行性进行了分析，对基于邻域关系的推荐算法中加入了采样的操作，极大地降低了推荐系统中的计算开销。

关键词：协同过滤　交替最小二乘法　推荐系统　分布式系统　并行计算

Abstract

The rapid development of modern Internet promotes the generation and exchange of information. People are currently facing problem which is the data amount is increasing fast and the speed of data generation is becoming higher. At the same time, it is recognized that data is also another form of asset which is very important to today's world. It makes sense for enterprise to do the collection and processing of data for development. Under this background, big data platform has become a hotspot of contemporary research projects. With Google proposed MapReduce computing pattern and Hadoop made the open source implementation of MapReduce for later, this pattern has been widely adopted. The birth of Spark platform, through the introduction of RDD and DataFrame data structure and calculation based on memory, which win Hadoop in computing speed, becomes a research focus of the industry and academia.

Now people are facing an unprecedented number of consumer goods and it is very hard for them to choose. With the growing popularity of electronic business of online shopping, the recommendation from electronic business platform greatly affect the final income of shop and the purchasing behavior of buyers. When faced with the growing number of users and types of goods, how to recommend goods to users has become a major problem in many companies. In this context the systematic research on recommendation systems has been made.

In this paper, we mainly focus on the scale up problem of collaborative filtering in recommendation systems. Collaborative filtering is algorithm which only leverage user-item ratings without any information from the item and user itself. We have done some research on the two main kinds of algorithm, that is latent factor model and neighborhood-like algorithm, and their implementation on Spark platform. We also have done the analysis of the parallel implementation of ALS algorithm, and added the sample-down method to the neighborhood-like algorithm to speed up our algorithm.

关键词：Collaborative Filtering, ALS, Recommender Systems, Distributed Systems, Parallel Computing

第 1 章　绪论

1.1　课题背景

随着信息技术的不断发展，如今的数据产生速度越来越快，信息的交流越来越便捷。在如今的大数据时代，人们在进行消费面临的选择十分之多，很多时候用户真正感兴趣的商品被淹没在了无尽的其他商品之中。现代的电商行业发展迅速，网络购物对于人们来说十分普遍，在网络购物的过程中，电商平台如何通过给用户推荐感兴趣的商品，成为了重要的研究问题。这促进了对于推荐系统的研究。国外的 Amazon 电商平台[1]，Pandora 音乐推荐平台，Netflix 影音平台，YouTube 平台[2]，国内的淘宝，京东等企业，都面临着在用户数和商品数不断增多下如何准确地推荐的问题。

在目前对于推荐系统的研究中，主要将推荐系统划分为个性化推荐系统与非个性化推荐系统，主流的研究集中于个性化推荐系统。个性化推荐算法可以划分成基于内容过滤与基于协同过滤两大类的个性化推荐算法。协同过滤算法主要分为隐式因子模型和基于邻域的推荐算法两大类，或者根据用户与系统的交互方式分为显式评分系统与隐式评分系统。对于推荐系统的研究主要基于以下问题：

1. 数据总量大，用户多，产品多。
2. 有些应用对推荐的实时性比较强。
3. 新用户、新产品的冷启动问题。
4. 老用户的评分过多，拖慢计算速度。
5. 用户的兴趣是一直在变的，通常来说用户与系统新的交互具有的价值更高。
6. 恶意用户利用推荐算法的漏洞攻击系统。

在这些问题中，对于推荐系统算法的并行化和计算速度的提升的研究尤为重要。

1.2 相关研究现状

1994 年，来自明尼苏达大学的 GroupLens 研究小组开始对一个名叫 MovieLens 的电影网站进行研究，研究的主要内容是对该网站的用户打分情况进行分析，从而得到用户可能感兴趣的电影。随后，著名电商亚马逊将推荐系统应用到了购物中，通过分析用户历史购买行为向他们推荐可能会购买的物品，从而一举将销售额提高了近三成。自此，作为一个计算机科学、信息学。社会学等多学科交叉的新兴领域，推荐系统及其技术吸引着越来越多学者的关注。

推荐系统目前被人们普遍接受的定义是由 1997 年 Resnick 和 Varian 提出的[3]。最近 20 年来，对于推荐系统的主要研究主要集中在两个方面：在推荐算法层面上，发明和改进推荐算法，从而能够面对新的应用场景，或者在相对应的场景下提高某些性能指标，如准确度等；另外在工程实践方面，如何使得推荐系统能够面对大数据环境下的挑战，提升推荐系统的实时性和稳定性，以及推荐系统中计算的并行化。这两个方面理论与技术的不断发展，推动者推荐系统的不断发展与进步。

对于推荐系统而言，当前的推荐算法主要可以分为，基于协同过滤的推荐，基本思想是利用用户的行为数据，基于群体智慧进行推荐；基于知识的推荐，在某些特殊领域，用户的行为数量不足，我们需要基于余弦了解的知识对于用户的喜好与需求进行预测；基于内容的推荐，该算法核心是利用分析物品的描述信息及其相关特征，通过聚类来分析出用户的感兴趣商品。

本文主要针对推荐系统中的协同过滤算法进行相关的研究，在协同过滤算法方面存在一些相关的研究如下。文献[4] 提出了一种在利用 KMeans 聚类的时候一种新的中心点选择的方案。文献[5] 在常规的协同过滤算法中结合 KMeans 聚类方法并在 Android 平台上实现了推荐系统。文献[6] 同样适用 KMeans 聚类结合邻居投票的方法实现了一种新的协同过滤算法。文献[7][8] 对于当代基于用户商品评分矩阵的协同过滤算法总结了当前的研究现状与未来的挑战。文献[9] 分析了当前协同过滤算法在可扩展与高性能的需求下的一些限制. 文献[10] 主要提出了一个分布式可扩展的协同过滤算法，文献[11] 对于基于相似性关系的推荐算法在 MapReduce 计算模式下提出了可扩展的方案。文献[12] 对于推荐系统中冷启动问题进行了研究，主要可以分为系统、用户和商品三者的冷启动问题。文献[13] 对于 Hadoop 平台下基于用户的协同过滤算法进行了分析与实现，文献[14] 对于 Hadoop 平台下基于物品的协同过滤算法进行了扩展性方案实现。文献[15] 对于

推荐系统中的隐式反馈评分方案进行了研究。文献[16] 对于推荐系统中的矩阵分解技术进行了研究。文献[17] 对于协同过滤算法的隐式语义模型进行了研究。文献[18] 对于推荐系统中混合推荐算法进行了研究。

1.3 本文的研究内容

本文主要研究推荐系统中的协同过滤算法的可扩展性问题以及进行相应的算法实现。传统的推荐算法可能面临着难以分布式处理，当数据量较大时扩展性不好的问题。本文完成了两方面的工作，其一是基于交替最小二乘法完成了隐式因子模型，主要是利用 ALS 算法对大矩阵进行分布式分解，通过分解结果重建矩阵，利用重建的矩阵进行评分；其二的工作可以细分为两个小方面，分别对基于用户-用户和物品-物品的相似关系的邻域推荐算法进行了分布式实现，并且根据在实现过程中加入了下采样的操作，降低了邻域推荐算法的时间复杂度。

1.4 论文结构

论文第一章节对于推荐系统和大数据环境下的相关背景进行了介绍；论文第二章节对采用的相关技术进行了介绍与分析，包括 Spark 平台的搭建与使用、推荐系统中协同过滤算法下的隐式因子模型和邻域推荐算法详细介绍与分析；论文第三章节介绍了我们的系统设计，包含数据源的选择，并行算法的实际设计；论文第四章节对于我们设计的系统进行了实现，并记录了相应的实验结果，对实验结果进行了综合分析；论文第五章节对我所做的工作进行了总结并展望了可以下一步扩展的地方。

第 2 章　相关技术介绍

2.1　Spark 平台介绍

Apache Spark[19] 是为了快速计算而设计出来的集群计算技术。它基于 Hadoop MapReduce 技术，并且扩展了 Hadoop MapReduce 模型来有效地进行更多类型的计算，包括交互式查询以及流处理。Spark 最主要的特性是内存集群计算技术，将数据保持在内存而不序列化到硬盘上大大提高了计算速度。

Spark 最早是由 UC Berkeley 的 Matei Zaharia 在 2009 年作为 Hadoop 的一个子项目进行开发的。它在 2010 年基于 BSD 证书开源并在 2013 年捐献给了 Apache Software Foundation(ASF)。从 2014 年的二月起，Spark 已经成为了 ASF 中的顶级项目之一。Apache Spark 官网将 Spark 定义为大规模数据处理的一个快速、通用的引擎。

Spark 有如下主要特性：

- **速度快**。如果数据全部在内存中，Spark 平台上的计算能比 Hadoop MapReduce 快 100 倍以上，如果将数据序列化到硬盘上，则能快 10 倍以上。
- **便于使用**。Spark 支持 Java/Scala/Python/R 四种语言的 API，提供了超过 80 多种高度抽象的操作来使得编写并行程序更加容易，并且提供了 Scala，Python 和 R 的交互式 Shell 用以快速熟悉并验证原型程序。
- **通用性**。Spark 联合了 SQL、streaming 和 complex analytics。Spark 由 SQL and Dataframes, MLLib for machine learning[20]，GraphX 和 Spark Streaming 驱动。我们可以在同一个程序中无缝地联合使用这么多库。
- **随处运行**。Spark 可以运行在 Hadoop、Mesos、独立模式，还可以方便地部署到现有的商业云上。Spark 可以访问包括 HDFS、Cassandra、HBase 和 S3 等在内的多种数据源。我们可以在独立集群模式下或者 EC2 或者 Hadoop YARN 或者 Apache Mesos 下运行 Spark，访问位于 HDFS、Cassandra、HBase、Hive、Tachyon 和任何 Hadoop 数据源。

Spark 有如下四部分主要组成：

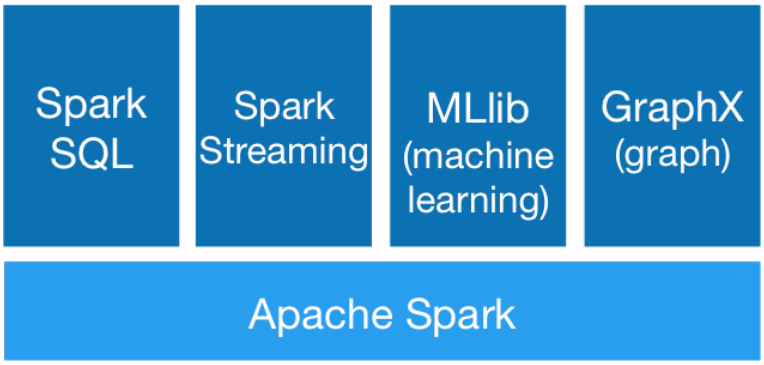


图 2.1　Spark Stack

- **Spark SQL**。Spark SQL 使得你可以通过 Java、Python、R、Scala 语言在 Spark 程序中使用 SQL 或者熟悉的 DataFrame API 来查询结构化数据。并且 DataFrames 和 SQL 提供了一种通用的方式来访问多种不同的数据源，包括 Hive, Avro, Parquet, ORC, JSON 和 JDBC。我们甚至可以使用对数据使用连接（join) 操作。Spark SQL 完全兼容 Hive 的操作，我们可以对于已有的数据运行未经修改的 Hive 查询。通过 JDBC 和 ODBC，我们可以使用标准的数据库连接方式。

- **Spark Streaming**。Spark Streaming 提供了 Apache Spark 语言级整合的 API 来进行流处理，我们可以像编写批处理任务的代码一样来编写流任务。支持 Java、Scala、Python。容错性较高，开箱支持从 lost work 和 operator state 中恢复。Spark 中不同部分整合性较好，我们可以把流处理和批处理以及交互式请求方便地结合起来。

- **Spark MLlib**。Spark 中方便使用的机器学习库，MLlib 可以和 Python 中的 Numpy 库进行交互，使用任何 Hadoop 数据源 (HDFS, HBase 或者本地文件)，这使得 MLlib 很容易插入 Hadoop workflow 中。在性能上，Spark 中的高质量算法，比 MapReduce 快 100 倍。Spark 非常方便部署，我们可以在 Hadoop 2 集群上不预装任何东西的情况下运行 Spark 和 MLlib。

- **GraphX**。灵活性非常好，我们可以在 collections 和 graphs 之间无缝整合，GraphX 在单个系统中统一了 ETL，解释分析，迭代图计算。我们可以把同一份数据视为 graphs 或者 collections，采用 Spark 中 transform 或者 join graphs 操作。在性能上，与其他专门的图处理系统相比 Spark 是最快的。在算法数量上，Spark 拥有相当多的图算法实现，包括 PageRand, Connected components,

label propagations, SVD++, Strongly connected components, Triangle count 等。

2.1.1 Spark 内部结构

Spark 中内部情况如图2.2。一个 Spark 集群，由初始化了 SparkContext 的驱

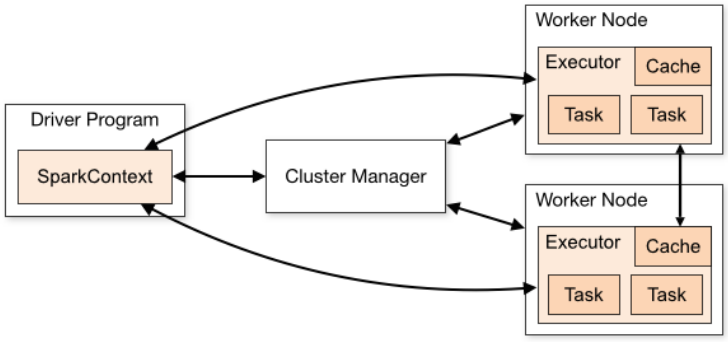


图 2.2 cluster overview

动程序，和许许多多个工作节点组成。我们把程序提交到 Spark Driver 上，由 cluster manager 来进行任务调度，实际的计算任务几乎都落在 Spark 的工作节点上。由于任务调度的关系，驱动程序和工作节点最好能够在同一个局域网下，不然因为网络延迟以及带宽的缘故，花在网络等待的时间可能较长，会影响到计算的时长。

2.1.2 Spark 基本编程模型

Spark 中最基本的数据类型是弹性分布式数据集 RDD(Resilient Distributed Dataset)[21]。它可以通过读取 Hadoop InputFormat 或者通过其他 RDD Transform 得到。在 Spark 中由两种最基本的操作 transform 和 action，可以认为分别是 Map 和 Reduce 的扩展。

下面介绍一下我们的算法中用到的一些 Spark 提供的变换和操作。如表2.1和表2.2。

通常来说，传递给 Spark 操作（例如 Map 或 Reduce）的函数是在工作节点上运行的。函数内的所有变量在所有节点上都是一份独立的拷贝。在远程工作节点上对于这些变量的修改不会反馈到驱动程序中。在工作节点上，支持通用的可读写的共享变量是低效的。虽然如此，Spark 提供了两种有限的共享变量: broadcast variables 和 accumulators。Broadcast variables 允许我们在每台机器上保

表 2.1　Spark 中的 transformation 操作

map(func)	对每个元素使用 func 操作之后返回新的 RDD。
filter(func)	过滤 func 操作结果为假的元素，返回新的 RDD。
flatMap(func)	与 map 相似，但是每个输入元素映射到 0 或多个输出元素。
sample(withReplacement, fraction, seed)	使用给定的随机种子，对输入的数据进行给定比例的采样。
union(otherDataset)	返回输入数据与参数联合的新 RDD。
intersection(otherDataset)	返回数据数据与参数交集的新 RDD。
distinct([numTasks]))	返回输入数据中不同元素的新 RDD。
groupByKey([numTasks])	对数据集 (K ,V) 对进行操作，返回 (K, Iterable<V>) 对。
reduceByKey(func, [numTasks])	当在 (K, V) 对上调用的时候，返回 (K, V) 对数据集。值 V 必须满足类型 (V, V)=>V，然后用 func 函数进行聚合得到。
sortByKey([ascending], [numTasks])	当 K 是可排序的，在 (K, V) 对数据集上调用该函数的时候，返回 ascending 指定的升序或者逆序排列的 (K, V) 数据集对。
join(otherDataset, [numTasks])	当在类型为 (K, V) 和 (K, W) 的数据集上调用的时候，返回 (K, (V, M)) 的数据集对。此外还有 leftOuterJoin/rightOuterJoin/LeftOuterJoin。
cogroup(otherDataset, [numTasks])	在数据类型 (K, V) 和 (K, W) 上进行调用的时候, 返回 (K, (Iterable<V>, Iterable<W>)) 数据集对。
cartesian(otherDataset)	当在数据类型 T 和 U 上进行调用时，返回所有的 (T, U) 对。

表 2.2　spark 中的 action 操作

reduce(func)	对数据集中的元素使用 func 函数 (输入两个元素，输出一个元素) 进行聚合。func 函数需要符合结合律和交换律以防在并行计算中出错。
collect()	把数据集中的所有元素作为数组返回到驱动程序中。通常在其他操作返回一个较小的数据集后比较有用。
count()	返回数据集中元素的个数。
first()	返回数据集中的第一个元素。
take(n)	返回包含数据集中前 n 个元素的数组。
takeSample(withReplacement, num, [seed])	返回数据集中随机采样后的数组。
takeOrdered(n, [ordering])	使用自然顺序或者自定义的比较器来返回数据集中的前 n 个元素。
saveAsTextFile(path)	将数据集中的元素写到本地文件系统、HDFS 或者其他支持 Hadoop 的文件系统的指定目录中，每个元素使用 toString 方法后作为文本的一行。
countByKey()	只对 (K, V) 类型的 RDD 数据有效. 返回对每个 Key 进行计数后的 (K, Int) 的 hashmap。
foreach(func)	对数据集中的每一个元素应用 func 函数。

持一个只读的变量，而不用随着任务下发变量。Accumulators 是符合交换律的只能“添加”的变量，在 Spark 可以用来实现计数器或者求和等操作。

2.2 推荐系统介绍

推荐系统在现在的场景中越来越重要，由于信息的爆炸式增加，互联网上的商品数量也在不断的增加，但是用户能够浏览的信息和商品都是有限的，如何让用户浏览他们需要的东西成为商品/服务提供者所面临的一个问题，这引起了人们对于推荐系统的研究。推荐系统最早是由美国明尼苏达大学的小组进行研究的，目前针对推荐系统的研究较为广泛，推荐系统的应用也较为普遍。在电商行业，推荐系统的重要性更是尤为突出，以 Amazon 为例，如图2.3中，Amazon 给购买某产品的用户推荐其他购买该产品的用户也购买了的产品，Amazon 的 Web 界面中处处有着推荐系统的应用。

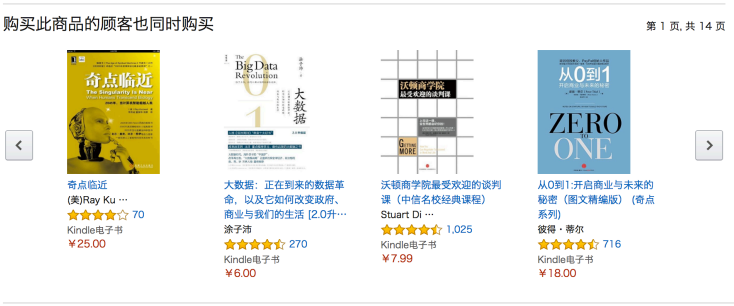


图 2.3 Amazon 推荐

目前的推荐系统主要可以分为个性化推荐系统与非个性化推荐系统。个性化会根据针对不同用户推荐不同的数据，而非个性化推荐系统考虑的因素没有用户相关的要素。除了基于用户评分的推荐算法，推荐系统还有很多变种，比如基于用户显式评分的推荐系统，基于用户隐式评分的推荐系统，基于信任关系的推荐系统。在推荐系统中也很很多值得研究的问题，比如冷启动问题，如何防止恶意用户攻击推荐系统等。

2.2.1 算法总览

在目前的非个性化推荐系统中，主要考虑是物品的热度。实现方法是对每个物品，用户可以进行投票赞成 (up vote)，也可以投票反对 (down vote)，点赞数较多的就可以排在比较靠前的位置。对于需要考虑时间因素的新闻推荐，会综合考虑点赞数和已经发布的时长两个因素，比如在知名的 reddit 网站，推荐算法有如

下的数学表示，其中 A 代表某篇新闻的发布时间，B 代表过去的某个固定时间点，t_s 代表他们时间的时间差。

$$t_s = A - B$$

U 代表点赞数，D 代表反对数。

$$x = U - D$$

$y \in \{-1, 0, 1\}$ 代表了 x 的符号。

$$y = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x = 0 \\ -1 & if\ x < 0 \end{cases}$$

z 代表 x 和 1 绝对值的最大值。

$$z = \begin{cases} |x| & if\ |x| \geq 1 \\ 1 & if\ |x| < 1 \end{cases}$$

最终我们得到评分函数如下公式2.1，反对时间越晚，用户点赞数 − 反对数越大，分数越高，越靠前：

$$f(t_s, y, z) = \log_{10} z + \frac{y t_s}{45000} \tag{2.1}$$

reddit 的非个性化推荐系统是较为成功的推荐系统之一，并且系统开销很小。

对于个性化推荐算法推荐算法，主要有我们如下介绍的基于内容过滤和协同过滤两大类，我们将在接下来的章节详细介绍这一部分。

2.2.2 基于内容过滤的推荐算法

基于内容过滤的推荐算法，是根据一个用户消费的商品与其他的商品的相似性来进行推荐。而商品之间的相似性是通过商品的自身属性来计算，不依赖于用户对于商品的评分，适合于在能够获取商品较多的信息下使用。使用基于内容过滤算法的推荐系统，能够避免对于商品的冷启动问题。而基于内容的过滤算法的关键在于对描述商品的信息的选取，通常情况下我们会使用商品的标签 (tags) 来表示一个商品，标签的添加可以通过人工录入或者机器学习的方式来进行录入。每个标签可以视为一个维度，这样我们可以将一个商品表示成依赖标签的一个向量，通过计算两个向量之间的相似性，我们可以得出商品之间的相关性。我

们需要注意的几个问题有，商品之间的标签可能是近义标签，所以需要注意标签的选取。

举例如表2.3，表中相应的位为 0 代表没有这个标签，为 1 则代表有这个属性。对于《阿凡达》和《星球大战》来说，他们在使用三个标签标示的维度下，

表 2.3　以电影为例对基于内容的过滤进行举例

电影	爱情	动作	科幻
星球大战	0	1	1
阿凡达	0	1	1
泰坦尼克号	1	0	0

完全一样，而相对应的《泰坦尼克号》则拥有完全互补的标签，当我们制定了相似性计算规则之后，我们可以认为星球大战和阿凡达更为相似，有理由给喜欢星球大战的用户推荐阿凡达而不是泰坦尼克号。

2.2.3　基于协同过滤的推荐算法

基于协同过滤算法实现的推荐系统不依赖于商品本身的属性信息，仅仅需要收集用户对于商品的评分就可以计算出应该给用户推荐哪些商品，这对于无法收集或者收集商品信息不全的应用场景较为有用。算法基本输入如下，我们有一个用户-商品的评分矩阵如表2.4，我们利用用户已经评分过的商品，来计算用户对于未评分的矩阵的期望评分，对用户推荐用户预期评分较高的产品。协同过滤具体可分为两大类算法，基于邻域的推荐算法和基于矩阵分解的隐式因子模型，在下面章节中进行详细介绍。

表 2.4　用户评分矩阵

	物品 1	物品 2	物品 3	物品 4
用户 1	2	?	3	4
用户 2	?	4	4	5
用户 3	5	?	2	?

2.2.4 基于邻域的推荐算法

2.2.4.1 相似关系

基于邻域的推荐算法最重要的方法就是描述相似关系，对于物品的相似性的定义关系到计算时的权重分配，使用不同的相似性计量标准会带来有一定差异的相似性结果。常用的相似性计量有很多算法，我们整理如下，当我们计算两个向量 u 和 v 的相似性时，我们可以遵循如下统一的流程对多种相似性计算公式进行计算，首先进行预处理 preprocess，预处理后进行标准化 norm，最后在套用 similarity 公式进行计算。如下表2.3。基于领域的推荐算法可以通过使用用

表 2.5 相似性关系一览表

measure	preprocess	norm	similarity
Cosine	$\frac{v}{\|v\|}$	-	dot_{ij}
Pearson correlation	$\frac{v-\bar{v}}{\|v-\bar{v}\|}$	-	dot_{ij}
Euclidean distance	-	$\hat{v}^2$	$\sqrt{n_i - 2 \cdot dot_{ij} + n_j}$
Common neighbors	$bin(v)$	-	dot_{ij}
Jaccard coefficient	$bin(v)$	$\|\hat{v}\|$	$\frac{dot_{ij}}{n_i+n_j-dot_{ij}}$
Manhattan distance	$bin(v)$	$\|\hat{v}\|$	$n_i + n_j - 2 \cdot dot_{ij}$
Pointwise Mutual Information	$bin(v)$	$\|\hat{v}\|$	$\frac{dot_{ij}}{\|U\|} \log \frac{dot_{ij}}{n_i n_j}$
Salton IDF	$bin(v)$	$\|\hat{v}\|$	$\frac{\|U\| \cdot dot_{ij}}{n_j n_j^2}(-\log \frac{n_i}{U})$
Log Odds	$bin(v)$	$\|\hat{v}\|$	$\log \frac{\frac{\|U\| \cdot dot_{ij}}{n_j n_j^2}}{1-\frac{\|U\| \cdot dot_{ij}}{n_i n_j^2}}$
Loglikehood radio	$bin(v)$	$\|\hat{v}\|$	$2 \cdot (H(dot_{ij}, n_j - dot_{ij}, n_j - dot_{ij}, \|U\| - n_i - n_j + dot_{ij}) - H(n_j, \|U\| - n_j) - H(n_i, \|U\| - n_i))$

注：表的相似性关系及其数学表示

户-用户之间的相似关系，物品与物品之间的相似关系细分为两类。

2.2.4.2 用户与用户之间的相似关系

每个用户和其他用户之间的相似关系可以通过该用户给消费过的物品评分来计算。我们可以认为每个物品都是一个维度，每个用户都是在高维空间的一个向量，用户对商品的评分是对于用户的度量，通常我们使用余弦角度公式2.2来计算相似性。

$$similarity(\vec{A}, \vec{B}) = cosine(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| * \|\vec{B}\|} \tag{2.2}$$

我们可以通过计算两个用户共同评分过的的物品的评分组成的向量之间的夹角，通过夹角的大小我们来评估用户之间的相似性。当夹角超过 90 度的时候，我们认为用户之间存在负相关性。这种负相关性不需要出现在我们的用户的邻居中，需要剔除掉。如果两个用户共同评价的商品只有一个，向量蜕化到标量，相似性计算必然为 1，这种情况需要去掉。传统的串行算法如算法1。

2.2.4.3 物品与物品之间的相似关系

基于物品-物品之间的相似性关系的计算，在算法上与基于用户-用户的的推荐算法十分相似，如果将用户评分矩阵转置之后再输入到算法中，那么后续步骤就是一样的，这里就不详细展开说明了。需要注意的是，就是如果存在两个物品只有一个用户评分的话，向量蜕化到标量，那么他们之间的相似性必然为 1，我们在计算相似性的时候，要去掉这种情况。

2.2.5 隐式因子模型

隐式因子分解模型通常使用矩阵分解来来完成，我们使用 ALS(Alternating Least Square) 最小交替二乘法[22] 算法来进行矩阵分解的计算。我们把行数为用户数 m，列数为物品数 n 的 $m \times n$ 矩阵分解成为两个 $m \times f$ 和 $f \times n$ 的矩阵。我们把分解出的因子矩阵认为是描述了物品和用户特征的矩阵，与在基于内容的推荐中的可以显式解释的矩阵做出区分，称之为隐式因子。我们有用户 u 和 i 如下公式

$$Q_{ui} = \begin{cases} r & \text{if user u rate item i} \\ 0 & \text{if user u did not rate item i} \end{cases}$$

r 是评分的真实数据。如果我们有 m 个用户 n 和物品，我们希望从评分矩阵中学习出代表物品和用户特征的的矩阵。学习出可以代表用户的因子向量，可以让

算法 1 基于用户相似性关系的串行 Top-N 推荐算法

输入:

- M, 用户-物品评分矩阵。每个 $M_{u,i}$ 元素代表代表一个评分或者为空。
- u，我们希望为其进行计算的用户。
- n，希望推荐的商品数目。
- $Sim(u, v)$，计算两个向量之间的的相似性的函数
- $WeightedSums(u, S)$ 计算对一个给定的用户计算某个商品推荐得分的函数
- $TopNRecommendations(R_u, n)$，用来给排序出 Top-N 高评分的商品的函数。

输出: $R_u(n)$，给用户 u 推荐的 Top-N 商品。

for $user\ v \in M$ **do**
  if $v \neq u$ **then**
    $S_{u,v} \leftarrow Sim(u, v)$
  end if
  for $item\ i \in M$ **do**
    if $u\ did\ not\ interacwith\ i$ **then**
      $R_u \leftarrow WeightedSums(u, i, S_{u,v})$
    end if
  end for
end for
$R_u(n) = TopNRecommendatiions(R_u, n)$

我们在特征空间中表示这个用户。学习出来能够表示物品的因子列向量，同样可以在特征空间中表示这个物品。对于物品分解出的矩阵 $Y \in \mathbb{R}^{f\times n}$（每部电影都是一列）和用户的因子矩阵 $X \in \mathbb{R}^{m\times f}$（每个用户是一个行向量）。由于我们有两个未知的变量需要计算。我们采用一种使用正则化的交替最小二乘法来进行求解。通过这样，我们先使用 X 来求解 Y，再使用 Y 来 X。通过足够次数的迭代，我们可以到达一个收敛点，此时矩阵 X 和 Y 都在迭代中不再改变，或者改变极小。虽然如此，在数据中有一个小问题。我们没有完整的用户数据，也没有完整的物品数据，这也是我们我们希望设计实现推荐系统的本意。因此我们在更新过程中没有获取评分的的电影进行惩罚。这样我们仅仅依赖于用户已经评分过的电影的评分数据，而不用依赖于在推荐系统中还没有评分的数据。我们定义了一个权重 w_{ui} 如下

$$w_{ui} = \begin{cases} 0 & \text{if } q_{ui} = 0 \\ 1 & \text{else} \end{cases}$$

现在我们需要定义我们的优化目标，定义损失函数（loss function）如下。

$$J(x_u) = (q_u - x_u Y)W_u(q_u - x_u Y)^T + \lambda x_u x_u^T$$

$$J(y_i) = (q_i - X y_i)W_i(q_i - X y_i)^T + \lambda y_i y_i^T$$

我们致力于优化这个函数。注意到我们加入了正则化系数来避免对于数据的过拟合（overfiting）。这个损失函数是有解的，我们可以不需要利用梯度下降的方法，而是直接求解来完成。解如下

$$x_u = (YW_uY^T + \lambda I)^{-1}YW_uq_u$$

$$y_i = (X^TWiX + \lambda I)^{-1}X^TW_iq_i$$

注意到这里 x_{u_1} 和 x_{u_2} 之间并无关系，这就使得我们可以在不同的机器上并行计算这些值。$W_u \in \mathbb{R}^{n\times n}$ 和 $W_i \in \mathbb{R}^{m\times m}$ 都是对角矩阵。对于正则化系数的选择我们可以通过交叉验证来完成。

2.2.6 内容过滤与协同过滤之间的比较

协同过滤相比内容过滤的优势如下

1. 不需要了解商品的内容
2. 可以适应用户一直在变化的兴趣

3. 可解释性更好 (基于邻域的推荐算法)

4. 推荐的多样性更好 (ALS)

协同过滤相比内容过滤的劣势如下

1. 用户评分的矩阵十分稀疏带来了计算上的困难。

2. 隐私泄露问题。

3. 可能存在针对性的恶意评分，会影响到给其他用户的推荐。

第 3 章　系统设计

3.1　数据源选择

我们所采用的数据[23] 是由明尼苏达大学 GroupLens 研究小组提供的 MovieLens 数据。数据是真实的用户对于电影的评分，MovieLens 数据从上世纪 90 年代开始收集，目前评分总数已经超过两千万条。我们采用的数据集数据主要划分五个部分。

- 100k 数据集。数据来源于 1000 用户对于 1700 部电影的评价，评分总数达到 10 万。使用 5 分制。
- 1M 数据集。数据来源于 6000 用户对于 4000 部电影的评价，评分总数达到 100 万。使用 5 分制。
- 10M 数据集。数据来源于 72000 用户对于 10000 部电影的评分。评分总数达到 1000 万。使用 5 分制，可以使用半分。
- 20M 数据集。数据来源于 138000 用户对于 27000 部电影的评分。评分总数达到 2000 万。使用 5 分制，可以使用半分。
- small 数据集。数据来源于 668 个用户对于 10325 部电影的评分，评分总数达到 10 万。使用 5 分制，可以使用半分。

3.1.1　数据预处理部分

标准提供的数据包含时间戳，并且几类数据采用不同的分割方式，由于我们不需要时间戳，并需要统一数据的表示。在预处理过程将时间戳去除，并统一每行数据的分割方式使用逗号进行分割。预处理过程我们可以在单机上完成，并不需要分布式。

3.2　Spark 架构搭建

前面提及到，Spark 框架由 master 节点和 slave 节点组成，在我们搭建的 Spark 集群中，使用了一台 master 节点和两台 slave 节点进行搭建。如下图3.1。Driver

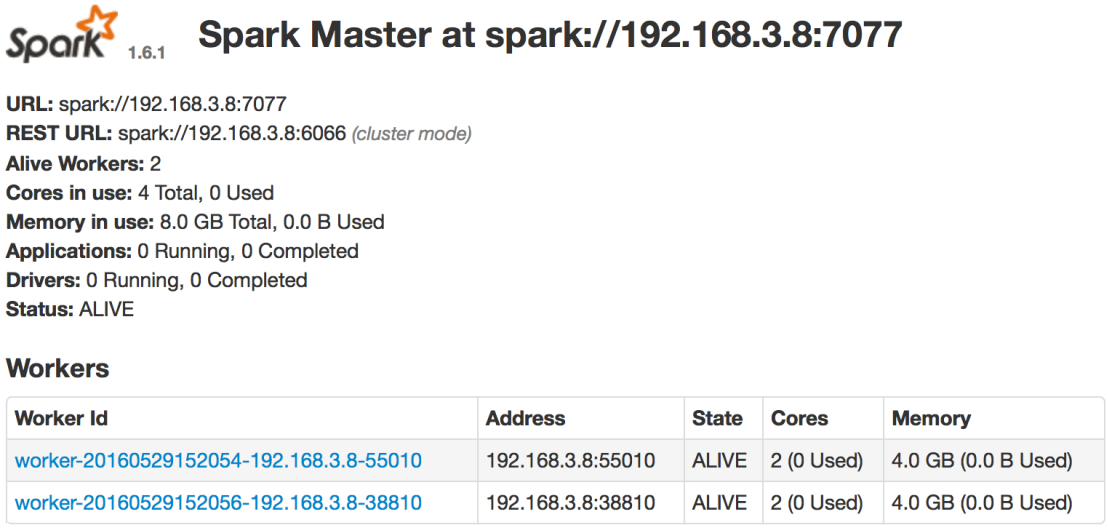


图 3.1　Spark cluster

使用了 2g 内存，使用了两个 work 节点，每个 work 节点使用 2 个核，4G 内存。

3.3　并行算法设计

前述描述基于邻域的推荐的算法时使用的是串行算法，无法进行并行化。下面我们设计并行化方法来实现基于邻域的推荐算法，而 ALS 算法由于本身迭代过程可以并行化，不需要做额外的修改。

使用基于用户-用户的相似性计算和物品-物品的相似性计算所采用的算法主体框架基本相同，只需要在输入出稍微变化一下即可通用。在朴素的并行算法之外，我们在还对用户的评分总数进行了采样操作，通过采样，我们可以在保持运算精度的情况下降低运算时间，下一节在实验结果中我们会进行分析。

3.3.1　算法复杂度分析

我们先分析 ALS 算法中 $x_u = (YW_uY^T + \lambda I)^{-1}YW_uq_u$ 这一步计算的复杂度。在求逆 $YW_uY^T + \lambda I$ 中，我们忽略单位矩阵的影响，假设我们希望分解的特征数是 f。所以 $X = X_{m\times f}$ 和 $Y = Y_{f\times n}$。对于 x_u，求逆里面的运算复杂度是 $O(f*n*f)$。$f \times f$ 矩阵的求逆运算复杂度来说 $O(f^3)$，之后乘法复杂度是 $O(f*f*n)$ 和 $O(f*n*1)$。所以对于求 x_u 的复杂度是 $O(f^2n)+O(f^3)+O(f^2n)+O(fn) = O(n)$，求 y_i 的复杂度是 $O(m)$。因为每次迭代，需要求 m 次 x_u 和 n 次 y_i。得出单次迭代 ALS 算法的复杂度为 $O(mn)$。

未加采样的的基于用户-用户的相似关系的推荐算法，最坏情况每个物品会

算法 2 相似性的并行计算

输入:

- $M_{u,*}$, 一个代表用户-物品评分矩阵的的 RDD，用户的索引作为键值。每个 $M_{u,*}$ 都代表一个评分或者为空。
- $Sim(((u,v),Seq(V)))$, 一个用户自定义的用来计算用户 u 和 v 余弦相似性的函数。V 是用户 u 和 v 都交互过的物品-物品评分评分对的列表。
- $SampleInteractions(K, Seq(V), n)$ 一个用户自定义的用来对 $Seq(V)$ 进行采样的函数，如果 $Seq(V)$ 的长度超过 n，返回其本身，否则，就返回 $Seq(V)$ 中的 n 个随机采样。
- $KeyOnFirstItem(((u,v),V)$, 是一个用户自定义的用来转换用来把 (u,v) 转换成 u 并添加 v 到 V 值行程 (u,V) 对的函数。通常用来在单个物品上产生对。
- $NearestNeighbors(K, Seq(V), k)$ 是一个用户自定义的函数用来从邻居物品中选择最相似的 K 个物品。
- $Map(f,(K,V))$, $Filter(f,(K,V))$, and $GroupByKey((K,V))$ 像之前定义的一样，(K,V) 代表一个 RDD 对象，f 是一个用户自定义函数。

输出: $S_{u,v}$ 一个稀疏的用户相似性矩阵，每个 $S_{u,v} \in [0,1]$ 或者为空。

function PAIRWISEITEMS(M, N)
    itemRatingPairs ← Map(FindItemPairs(), Map(SampleInteractions(n), $M_{u,*}$))
    pairwiseItems ← GroupByKey(itemRatingPairs)
    emit pairwiseItems
end function

function ITEMSIMILARITY(pairwiseItems, k)
    itemSims ← Map(KeyOnFirstItem(), Map(Sim(), pairwiseItems)
    nearestItems ← Map(NearestNeighbors(k), GroupByKey(itemSims))
    emit nearestItems
end function

算法 3 并行的 Top-N 推荐算法

输入:

- $M_{u,*}$, 一个代表用户-物品评分矩阵的的 RDD，用户的索引作为键值。每个 $M_{u,*}$ 都代表一个评分或者为空。
- $nearestItems$, 在 M, 中最相似的 K 个物品。这个变量需要用 Spark broadcast 方式初始化。
- n, 需要返回的 n 个推荐商品。
- $WeightedSums(K, V, NearestItems, n)$, 用户自定义的根据相似关系计算加权后的评分的函数。
- $Map(f, (K, V))$, $Filter(f, (K, V))$, and $GroupByKey((K, V))$ 像之前定义的一样，(K, V) 代表一个 RDD 对象，f 是一个用户自定义函数。

输出: itemRecs, 对一个用户评分最高的 n 个推荐商品

```
function GROUPITEMRATINGS(M)
    UserItemRatings ← GroupByKey(M_{u,*})
    emit UserItemRatings
end function

function TOPNRECOMMENDATIONS(UserItemRatings, NearestItems, n)
    ItemRecs ← Map(WeightedSums(NearestItems, n), UserItemRatings)
    emit ItemRecs
end function
```

产生 m^2 对用户评分组合，每个用户组合最多有 n 个共同交互的产品，复杂度是 $O(m^2n)$，基于物品-物品的相似关系的推荐算法复杂度是 $O(mn^2)$，通过加入 sample 采样数 K，理论上将复杂度降低到了 $O(K^2m)$ 或者 $O(K^2n)$ 的复杂度，K^2 是一个较大的常数。

第 4 章　实验结果分析

实验完成了 Top-N 推荐算法，并对提出的每种推荐算法进行了评估分析，并对实验结果给出了解释。

4.1　实验结果度量标准

推荐系统的度量标准有很多，在我们设计的推荐系统中，我们主要关心推荐出来的商品评分与用户真实的评分之间的误差大小。因此，我们采用较为通用的均方根误差 (rmse, root mean square error) 方法，其中 r_{pre} 是对一个商品的预测评分，r_{test} 是对一个商品的真实评分。

$$rmse = \sqrt{\frac{\sum_{i=1}^{n}(r_{pre} - r_{test})^2}{n}} \tag{4.1}$$

在我们的每次实验中，都将输入数据集分割为 80% 的训练集和 20% 的测试集，使用训练集训练出的模型来对测试集进行预测，并使用 rmse 公式计算均方根误差。除了我们的实验算法，我们还使用一个用户的平均评分来预测新评分，一个商品的平均评分来预测新评分作为对照组。通过多次实验，我们在后续章节分析实验结果。

4.2　ALS 算法推荐结果分析

使用 ALS 算法，我们需要对算法进行参数训练，寻找到比较好的参数后用于预测。我们的 ALS 算法接受三个参数，矩阵分解时需要指定的特征数 rank 值，regParam 正则化参数，以及算法的迭代次数 iterNum。我们通过对于 100k 的数据进行参数训练，得到一组较优的参数值，rank=4，regParam=0.1，iterNum=10。我们使用这组参数对后续更大的数据集进行训练。

对于 ALS 算法的运行结果如图4.1所示，最下面的一条线是使用 ALS 算法，中间那条是使用 itemMean(物品的平均分作为物品的预测得分) 的，最上面的线使用的是 userMean(用户的平均得分作为用户对一物品的预测得分)。通过图中所示我们可以观察到当数据量增大的时候，rmse 基本上与数据量的对数呈现出线性降低关系。随着数据量的增加，使用用户的评分平均分与商品的评分平均分所

产生的均方根误差都都在减少，ALS 算法的结果明显比其他两类要小。我们还使用 0 分作为对未知评分的预测，rmse 约为 3.7。由于与图中的朴素算法相差较大，故没有在图中表现出来。

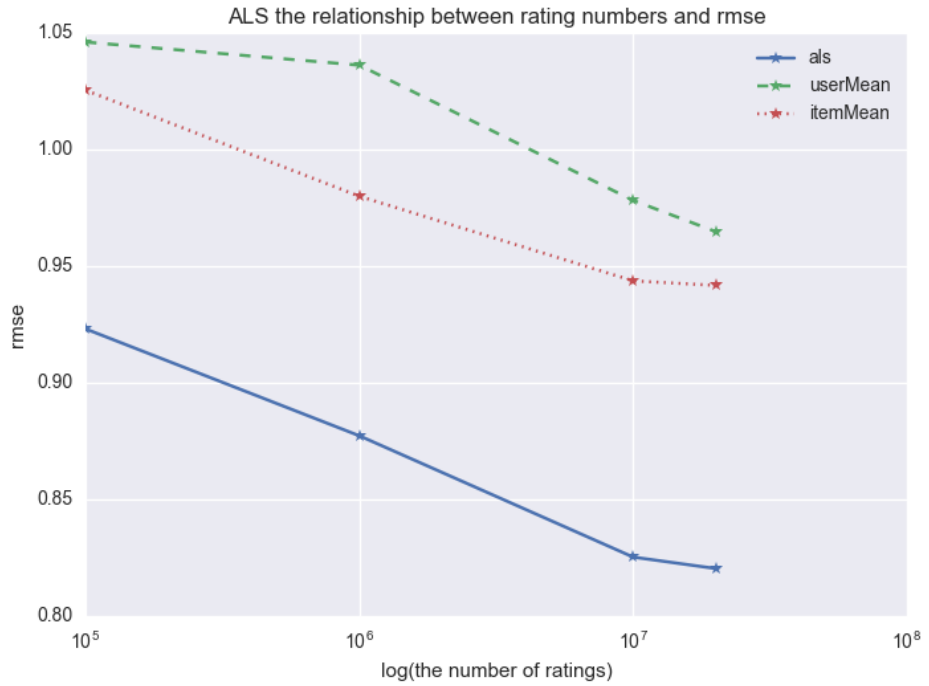
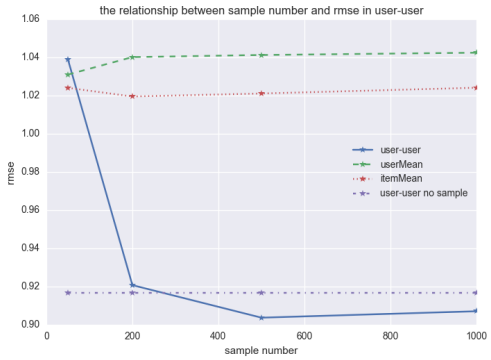


图 4.1　alg

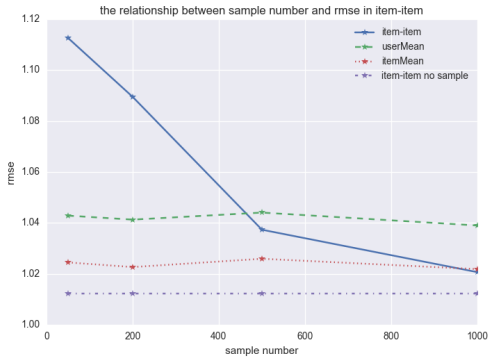
4.3　基于领域关系的算法推荐结果分析

由于基于邻域的推荐算法对于存储开销要求较大，而我们的集群相对较小，所以在基于邻域的方案中，我们只测试了两组 10 万数据量的数据集，值得注意的是其中一组为 5 分制，评价不包含半分，而另一组虽然同样为 5 分制，但是可以评价半分。我们的计算过程中使用了采样的操作，对于用户-用户的相似关系，限制了用户评价商品的总数，对于商品-商品的相似关系，限制了每个商品所被评价的用户总数。

在图4.2a中，从上到下条线分别采用的是 userMean，itemMean 和加采样 user-user 与未加采样的 user-user 算法。采样总数达到 500 时基本上基本用户和用户之间的相似关系就已经收敛了，说明这时候已经不需要更多的样本了。在图4.2b中，从上到下四条线分别为加了采样的 item-item，userMean，itemMean，未加采样的 item-item 算法。此时使用物品-物品的相似性进行推荐，算法最后的收敛效果与

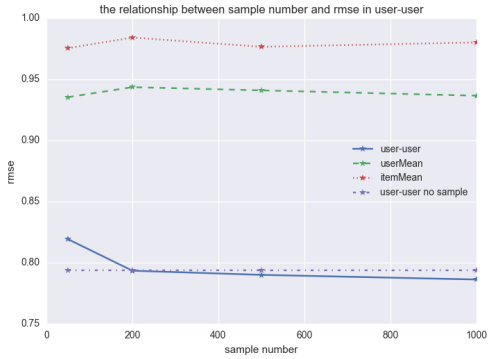


(a) user user 100k (no half rating)

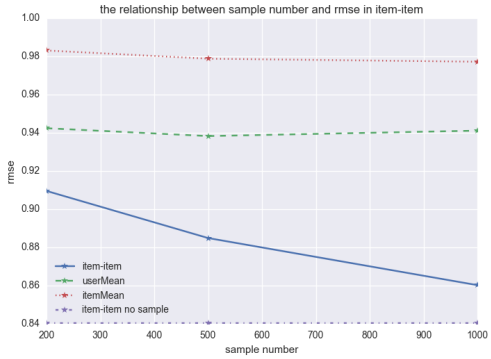


(b) item item 100k (no half rating)

图 4.2　不带半分的 100K 数据分析结果



(a) user user 100k (with half rating)



(b) item item 100k (with half rating)

图 4.3　带有半分的 100K 数据数据分析结果

朴素的平均值推荐效果类似。由图4.2a和4.2b中还可以得出另外一个结论，当物品数多于用户数时，我们使用物品的平均分来预测相比使用用户的平均分来预测效果更好一些。

我们在对另外一组使用了包含半分制的评分效果进行分析。在图4.3a中，从上到下四条线分别为 itemMean、userMean 和加了采样的 user-user 与未加采样的 user-user 算法。从图中可以看出，使用 user-user 算法的推荐结果十分明显优于另外两种对比算法，并且也比在不包含半分的测试中效果要好，在评分包含半分之后，对照组的推荐效果也比上一组实验的对照组要强。这是因为这一组 100k 的数据中，用户数只有 700 左右，而商品数达到了 10000，所以每个用户的评分数量较多，对于用户的刻画较为准确，而相比之下，每个商品所被评分的用户数就较少，所以基于 item-item 的评分效果就无基于 user-user 的评分效果好。

我们还对比了一组采样带来的运行时间上的差异，对于带有半分评价的数据集来说，拥有 10000 的物品，我们对其使用 item-item 算法得到的运行时间绘图如4.4。因为我们在算法中产生 item-item 对的时候，某些少数活跃的用户可能评分相当之多，影响了整个的运行时间，我们通过对这些用户进行下采样，可以显著提高系统的运行效率。我们需要在对于采样设定的阈值进行权衡，从而可以得到一个较为合理的兼顾计算精度和运行时间的值。

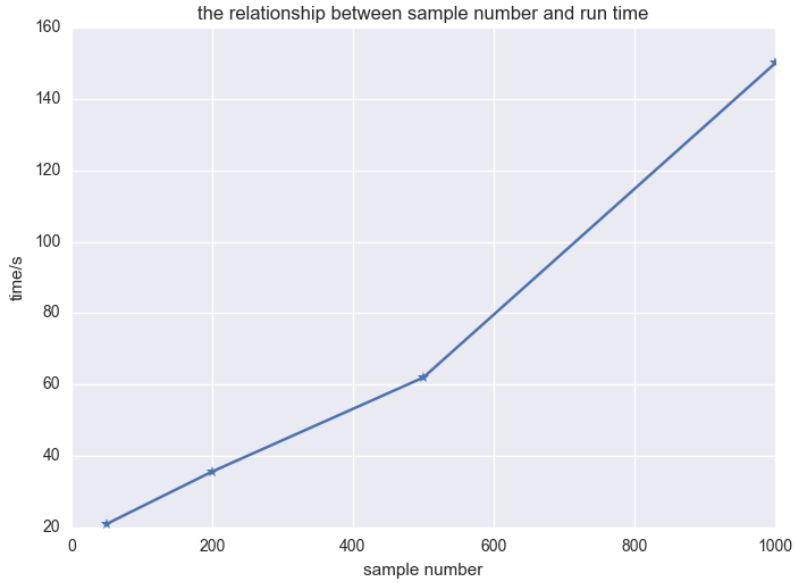


图 4.4　runtime

4.4　两种算法推荐结果对比总结

ALS 算法随着输入数据的增加，推荐效果越来越好，在数据量较小的时候，推荐效果有时会弱于朴素的推荐算法。基于邻域的推荐算法在数据量较小时就有不错的推荐效果，但是扩展性不如 ALS 算法，即使我们通过采样的方法使得数据量进行下降，在运算速度上也不如 ALS 算法，但是基于邻域的推荐算法在推荐算法中是可解释的，我们可以向用户说明，因为和你相似的人也喜欢这个产品，所以我们给你推荐这个产品，或者因为这个产品和你喜欢的某个产品很相似，所以我们给你推荐这个产品，ALS 算法计算的结果，我们无法向用户说明推荐的理由。但是因为采用了协同过滤算法中，不包含商品本身的信息，使得我们可能需要计算在真实场景中完全不相关的物品之间的相似性，这可能会导致一定的问题，我们需要根据应用场景来选择所需要的应用算法。

第 5 章　总结与展望

大数据的发展是由于当今的互联网时代面临的信息爆炸越来越严重，如何从纷杂的信息中筛选出对用户有用的信息，是服务/商品提供商面临的重要问题。推荐系统的出现，是服务平台为了最大程度满足用户的需求与喜好，增加自身收益的系统。传统的单机系统，已经不能适应大数据环境下的应用场景，我们需要通过分布式计算来满足对于大规模计算的需求。Spark 平台的出现，较好地解决了大数据时代对于迭代计算的要求。

在我的毕设设计中，较好地完成了基于 Spark 平台采用协同过滤算法的推荐系统，并对系统的可扩展性进行了一定的研究与优化。具体来说，基于 Spark 平台我们使用交替最小二乘法实现了基于矩阵分解的隐式因子模型，并且针对多组数据分析了结果；基于 Spark 提供的原语操作，我们实现了基于邻域的用户-用户相似关系、物品-物品相似关系的推荐算法，并且针对算法中会遇到的计算瓶颈问题采用了下采样的降低了计算复杂度。

未来我们可以考虑对基于邻域的协同过滤算法中使用不同的相似性关系进行研究，分析出不同的相似关系对于推荐系统的影响。同时可以将协同过滤与内容过滤进行结合起来，进行混合推荐。

参考文献

[1] Linden G, Smith B, York J. Amazon.com recommendations: item-to-item collaborative filtering[J]. *IEEE Internet Computing*. 2003, 7(1):76–80.

[2] Davidson J, Liebald B, Liu J, et al. The YouTube video recommendation system[M]. New York, New York, USA: ACM, September 2010.

[3] Resnick P, Varian H R. Recommender systems[J]. *Communications of the ACM*. 1997, 40(3): 56–58.

[4] Zahra S, Ghazanfar M A, Khalid A, et al. Novel centroid selection approaches for KMeans-clustering based recommender systems[J]. *Information Sciences*. November 2015, 320:156–189.

[5] Kularbphettong K, Somngam S, Tongsiri C, et al. A recommender system using collaborative filtering and k-mean based on android application[C]. Proceedings of International Conference Applied Mathematics, Computational Science and Engineering. [S.l.], 2014: 161–166.

[6] Dakhel G M, Mahdavi M. A new collaborative filtering algorithm using K-means clustering and neighbors' voting[M]. [S.l.]: IEEE, 2011.

[7] Shi Y, Larson M, Hanjalic A. Collaborative Filtering beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges[J]. *ACM Computing Surveys (CSUR)*. July 2014, 47(1):3–45.

[8] Su X, Khoshgoftaar T M. A survey of collaborative filtering techniques[J]. *Advances in artificial intelligence*. January 2009, 2009(12):4–19.

[9] Cacheda F, Carneiro V, Fernández D, et al. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems[J]. *ACM Transactions on the Web (TWEB)*. February 2011, 5(1):2–33.

[10] Narang A, Srivastava A, Katta N P K. Distributed Scalable Collaborative Filtering Algorithm[M]. Berlin, Heidelberg: Springer Berlin Heidelberg, August 2011: 353–365.

[11] Schelter S, Boden C, Markl V. Scalable similarity-based neighborhood methods with MapReduce[M]. New York, New York, USA: ACM, September 2012.

[12] Liu J H, Zhou T, Zhang Z K, et al. Promoting cold-start items in recommender systems[J]. *arXiv.org*. April 2014, (12):e113457.

[13] Zhao Z D, Shang M s. User-Based Collaborative-Filtering Recommendation Algorithms on Hadoop[M]. [S.l.]: IEEE, 2010.

[14] Jiang J, Lu J, Zhang G, et al. Scaling-Up Item-Based Collaborative Filtering Recommendation

Algorithm Based on Hadoop[J]. *2011 IEEE World Congress on Services (SERVICES)*. 2011, 490–497.

[15] Hu Y, Koren Y, Volinsky C. Collaborative Filtering for Implicit Feedback Datasets[C]. 2008 Eighth IEEE International Conference on Data Mining (ICDM). [S.l.]: IEEE, 2008: 263–272.

[16] Koren Y, Bell R, Volinsky C. Matrix Factorization Techniques for Recommender Systems[J]. *Computer*. July 2009, 42(8):30–37.

[17] Hofmann T. Latent semantic models for collaborative filtering[J]. *ACM Transactions on Information Systems (TOIS)*. 2004, 22(1):89–115.

[18] Burke R. Hybrid Recommender Systems: Survey and Experiments[J]. *User modeling and user-adapted interaction*. 2002, 12(4):331–370.

[19] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets.[J]. *HotCloud*. 2010, 10:10–10.

[20] Meng X, Bradley J, Yavuz B, et al. Mllib: Machine learning in apache spark[J]. *arXiv preprint arXiv:1505.06807*. 2015.

[21] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]// USENIX Association. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. [S.l.]: USENIX Association, 2012: 2–2.

[22] Zachariah D, Sundin M, Jansson M, et al. Alternating Least-Squares for Low-Rank Matrix Reconstruction[J]. *IEEE Signal Processing Letters*. April 2012, 19(4):231–234.

[23] Harper F M, Konstan J A. The MovieLens Datasets[J]. *ACM Transactions on Interactive Intelligent Systems*. January 2016, 5(4):1–19.