# mahout in action 中文翻译 第1章 初识mahout

**mahout in action 中文翻译**

**1. 初识Mahout**

本章涵盖以下内容：

* Apache Mahout是什么？
* 现实中推荐系统引擎、聚类、分类概述
* 配置mahout

读者可能从本书的标题中依然知晓，本书是一本使用的工具书，讲解如何将mahout应用于业界。Mahout是Apache开源的机器学习库。它实现的算法都被归入机器学习或者集体智慧的范畴，但是在这里Mahout主要注重协同过滤/推荐引擎、聚类和分类。

Mahout是可伸缩的。Mahout致力于实现海量数据，单机无法处理情况下的机器学习工具。在目前阶段，这种可伸缩性由java实现，有些部分基于Apache Hadoop这个分布式计算框架实现。

Mahout是java库。它不支持用户接口，预装好的服务器。以及安装等功能。它是一个利于开发者使用的工具框架。

**1.1 Mahout适合你吗？**

你也许想知道-Mahout是一个什么工程，或者是一本什么书？

如果你在找一本机器学习的教材，那本书就不适合了。本书没有尝试去全面解释各种算法和展现技术的理论基础和来源。阅读本书可以，但不保证，能够对机器学习技术，类似矩阵、向量等相关概念的熟悉。

如果你正在开发现代智能应用，那本书很适合。这本书提供了一种实践非理论的处理方式，有完整的实例、解决方法指南。这本书在展示Mahout如何展示解决这些问题的时候，证明了一些被有经验的开发者收集的独特见解。

如果你是一个人工智能、机器学习以及相关领域的研究者，也适合用本书。你最大的挑战就是将一个新算法代码实现，Mahout提供了一个丰富的框架，模式集合以及测试、部署大规模新算法的现成模块。这本书是一张快车票，让你的机器学习算法运行在复杂的分布式计算框架上。

如果你正在领导一个产品团队或者初创公司想利用机器学习创造竞争优势，这本书也适合你。通过真实世界的例子，这本书可以启发你很多技术可能有多种实 现方式的想法。它可以帮助你充满斗志的技术团队成员直接跳到很划算的实现能处理大量数据的应用，而在放在以前必须组织大量的技术资源才可以实现。

最后，你可能想知道怎么读“Mahout” – 这是一个常用的英式，它与“trout”押韵。这是一个北印度语的单词，指的是驱使大象的人，为了解释这个词，下面介绍一段关于它的历史。Mahout项 目开始于2008年，作为Apache Lucene的子项目，Apache Lucene项目是大家熟知的开源搜索引擎。Lucene提供了搜索、文本挖掘和信息检索的高级实现。在计算机科学领域，这些概念和机器学习技术近似，像 聚类、分类。所以，Lucene贡献者的一部分机器学习相关工作被剥离进入子项目。不久后，Mahout吸收进“Taste”开源协同过滤的项目。

自2010.4月起，Mahout成为Apache的顶级项目。

Mahout的大量工作不只是传统的实现这些算法，也实现将这些算法，让它们工作在hadoop之上。Hadoop的吉祥物是一头大象，这也解释了Mahout的工程名字。

[](http://www.adintellig.com/wp-content/uploads/2013/05/574fa266-9926-3432-9b6e-e67208970e89.png)

图1. Mahout以及相关项目 （见附件）  
Mahout孵化了相当多的技术和算法，很多都是在开发和实验阶段。在工程的早期阶段，有3个核心主题：协同过滤/推荐引擎、聚类和分类。这并不是Mahout中所有内容，但却是最显著、成熟的主题（在本书写作时），因此，这是本书的范围。

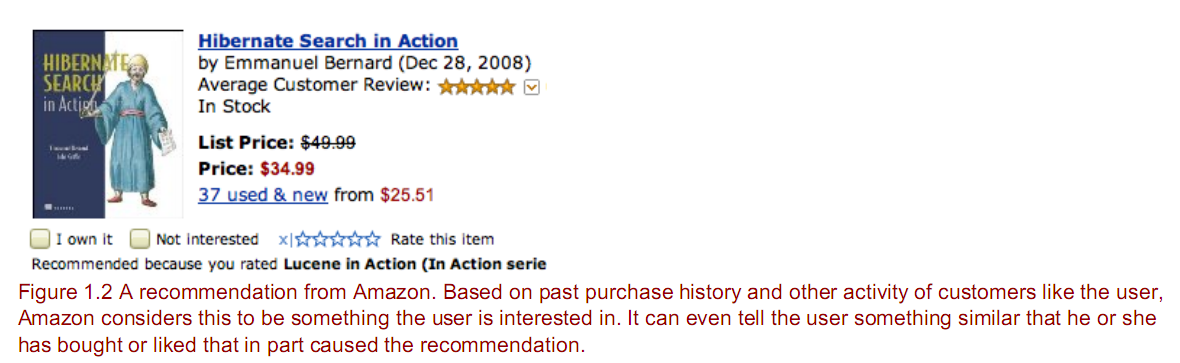
如果你读了以上内容，说明你已经对这3个系列的技术有潜在的兴趣。但是为以防万一，请继续阅读下面的内容。

# mahout实战 第1章 初识mahout 1.2 推荐系统引擎

**1.2 推荐系统引擎**

推荐引擎是目前我们使用的机器学习技术中最容易识别的。你可能已经见过相关的服务或网页，基于历史行为推荐书、电影、文档。他们尝试推论出用户偏好，并标记出用户不知晓的、感兴趣的item:

* Amazon.com可能是最出名的使用推荐系统商务网站。基于交易和网页活性，Amazon推荐给用户可能感兴趣的书籍和其他item。请参见图1.2（见附件）
* Netflix类似于推荐用户感兴趣的DVDs，并且为研究者提供百万大奖去提升推荐质量。
* 约会网站像Líbímseti将一部分用户推荐给其他用户。
* 社交网络网站像Facebook用推荐技术的变形来为用户识别最可能成为一个尚未建立联系的朋友。



对于Amazon和示例其他网站，通过这种聪明的交叉销售，推荐系统确实有具体的经济价值，同一家公司的报告指出推荐产品给用户能够带来8-12%的销售增长。  
<http://www.practicalecommerce.com/articles/1942-10-Questions-on-Product-Recommendations>

**1.3 聚类**

Clustering turns up in less apparent but equally well-known contexts.（首句该如何翻译，哪位仁兄帮个忙？）  
顾名思义，聚类技术尝试去将大量的拥有相同相似度的事物聚集到不同的类中。聚类是在海量或者难于理解的数据集里发现层次和顺序，展现兴趣模式，或使得数据集容易被理解。

* Google News据为了根据具备逻辑性的故事聚集展示新闻，而不是所有文章的行列表，使用新闻文章的Topic聚集新闻。图1.3做了说明（见附件）。
* 搜索引擎像Clusty基于相同的原因聚集搜索结果。
* 使用聚类技术，基于消费者属性，收入、位置、购买习惯，可将不用用户分到不用的类中。

**1.4 分类**

分类技术用于决定一个事物是不是属于一种类型、类目，或者该事物是不是含有某些属性。同样地，分类无处不在，尽管更多的时候隐于幕后。  
这些系统通过评估item的很多实例来学习，以推导出分类规则。这个平常的想法可以找到很多应用：

* Yahoo! Mail决定接收的信息是不是垃圾邮件，基于先前邮件和用户的垃圾邮件报告，以及邮件的特性。一些信息被分类为垃圾邮件，如图1.4所示（见附件）。
* Picasa (<http://picasa.google.com/>)和其他的照片管理应用可以判断一张照片中是否含有人脸。
* 光学字符识别软件通过将小区域作为独立字符来分类，将扫描文本的若干小区域归类到独立的字符上。
* 在iTunes中Apple’s Genius feature使用分类将歌曲划分到不同的潜在播放列表。



分类有助于判断一个新进入事物是否匹配先前发现的模式， 也常用于分类行为或者模式。分类也可用来检测可疑的网络活动或欺诈。也可用于根据用户发信息判定表示失望或者满意。

**1.5 扩展性**

当有海量、高质量的数据输入时，这些技术都能够达到最佳效果。在一些情况下，这些技术不仅要将海量数据作为输入，而且需要很快计算出结果。很快，这些因素使得可扩展性成为一个很大的问题。

依据一些粗略估计，Picasa在3年前可能已经拥有5亿张照片。这意味着每天需要分析数百万图片。分析一张图片并不是一个大问题，尽管需要重复数百万次。但是，学习阶段需要亿万图片都提供相关信息 — 上了规模的计算，使用单机是不行的。

<http://blogoscoped.com/archive/2007-03-12-n67.html>

依据一个类似的分析，Google News大约每天有350万新文章。尽管数量并不是很大，考虑到这些文章必须和目前其他文章同时聚类，为了及时响应计算时间需要在几分钟内。

Netflix为Netflix大奖发布的子集中包含1亿的打分。这只是适合竞赛的数据，据推测，Netflix实际上拥有的和必须用于创建推荐系统的整个数据远远大于这个这个量级。

<http://archive.ics.uci.edu/ml/machine-learning-databases/netflix/>

这些技术非常有必要应用于输入数据量很大的情形–因为很大，所有不适用于单机处理，甚至高配置的机器也不可以。所以，任何人实现这些技术都不能回避 可扩展性问题。这就是为什么Mahout将可扩展性作为最高优先级，也是为何本书关注可扩展性问题，别人所没有涉猎的一种方式，用于有效处理海量数据。

复杂的机器学习技术，上规模的应用，直到目前为止，只是一些大的、先进的技术公司在考虑。但是，今天计算能力已经比之前廉价，借助像Hadoop这 样的开源框架更方便。Mahout尝试通过使用hadoop提供优质、开源实现，能够在这个规模上解决问题，解决这类困惑，并且将其送到所有技术组织的手 中。

**1.5.1 MapReduce and Hadoop**

Mahout的一些部分使用了Apache hadoop工程，hadoop是一个开源的、基于java的MapReduce(<http://labs.google.com/papers/mapreduce.html> ) 实现。MapReduce是一种分布式计算框架，在Google内部使用。它是一种编程模式，开始听起来感觉很奇怪，或者太简单了以至于很强大。 MapReduce编程模式适用于输入是key-value键值对集合的问题。“map”函数将这些键值对转换为中间键值对。“Reduce”函数通过某 种方式将同一个中间键的值合并到一起并产出结果。实际上，很多问题可以设计成MapReduce问题，或者他们的一个系列。并且这种模式非常易于并行化实 现：所有的处理过程都是独立的，所以可以划分到不同机器上去。这里不再详细叙述MapReduce，hadoop提供的教程(<http://hadoop.apache.org/common/docs/current/mapred_tutorial.html> )。

Hadoop实现了MapReduce模式，这是一个不小的壮举，甚至让MapReduce像听起来那么简单。它管理输入数据、中间键值对、输 出数据的存储。这些数据可能是海量的，需要在多台计算机上运行，而不只是存储在某一台机器的本地。它管理不同机器之间的划分、数据传输。它负责检测、恢复 单个机器失败的情况。理解了有很多工作在幕后进行能够帮助你准备使用Hadoop的相对复杂性。不只是将它作为类库添加到你的工程中，它包含很多模块，每 一个模块都有若干类库和独立的服务器进程，可能运行在几台机器上。基于Hadoop的操作处理不简单，但投入在可扩展的、分布式实现可以让你在之后有很大 的收获：因为你的数据可能成倍的增长，这种可扩展的性质对你的应用来说是一种面向未来的方式。

稍后，本书将试着剪掉一些复杂性让你很快熟悉hadoop，基于这一点，你可以探索操作完整聚类的重点和细节，调整整个框架。因为这个需要大量 计算能力的复杂框架变的越来越流行，所以一点都不奇怪，云计算已经开始提供hadoop相关的支持。例如Amazon提供的Elastic MapReduce(<http://aws.amazon.com/elasticmapreduce/> )是一种管理Hadoop集群、提供计算能力、提供友好的接口的服务，可基于hadoop操作、监控大规模复杂任务。

**1.6 安装Mahout**

你需要安装很多工具，在你能“可以在家里玩”我们后面章节提供的一些示例代码之前。我们假设你已经熟悉Java开发。

Mahout以及和它相关的框架都是基于Java的，因此是平台无关的，你可以在任何一台安装了JVM的机器上使用他。有时候，我们需要给出许 多不同平台的示例和介绍，特别地，命令行模式在windows脚本和FreeBSD tcsh脚本有一些不同。我们使用命令行和与bash一起的句法，是一种类Unix平台的脚本。这是默认的在大部分Linux、 Mac OS X, 一些Unix变种和Cygwin（一种windows上的类Unix环境）。期望使用windows脚本的用户很可能不方便。尽管如此，本书下面的介绍列 表对你来说应该可以很容易的理解。

**1.6.1 Java和IDE**

你如果之前做过任何Java开发的工作，你的个人电脑上已经装上Java了。注意Mahout需要使用Java6。如果有所疑虑，打开一个终端并键入java -version。如果报告不是以“1.6”开头，你仍需要安装Java6。

windows和Linux用户能在 [http://java.sun.com](http://java.sun.com/) 找 到Java 6 JVM。苹果公司为Mac OS X 10.5 和 10.6提供Java 6 JVM。如果发现Java 6没有被应用，在“/Applications/Utilities”打开“java perferences”。在这里可以选择Java 6作为默认选项。

大部分人能发现在IDE的帮助下，非常好编辑、编译和运行这些示例；所以强烈推荐你使用IDE。Eclipse(<http://www.eclipse.org>)是非常常用、免费的一款Java IDE，安装配置Eclipse不在本书的讲解范围内，你可以花点时间去熟悉它。NetBeans (<http://netbeans.org/>) 也是一款常用开源IDE。 IntelliJ IDEA (<http://www.jetbrains.com/idea/index.html>)是一款强大的、流行的IDE，它有一个免费的社区版本。

例如，IDEA能从现有的Mavan模型中直接创建一个新工程；在创建新工程的时候，通过指定Mahout源代码的根目录，它将用一种友好的组 织方式自动配置、展示整个工程。例如，丢掉整本书的源代码 core/src/…，然后在IDE里面运行，只需要一次点击，依赖和编译的细节都会自动搞定。这证明了使用IDE比人工编译、运行容易很多。

**1.6.2 安装Maven**

和其他Apache工程一样，Mahout的构建、发布系统基于Maven(<http://maven.apache.org>)。Maven是一种命令行模式的管理代码，打包发布，生成文档，发布正式版本的工具。尽管它和Ant工具有一些相同之处，但是不同。Ant是一种灵活的、低级别的脚本语言，Maven是一种高级别的工具，其目标倾向于发布管理。

因为Mahout使用maven，你应该先安装maven。Mac OS X用户会发现maven是默认安装，如果没有，安装苹果开发工具包。在命令行键入mvn –version。如果你能看到版本号，这个版本号最低2.2，那么可以继续进行。否则，你需要安装mavn的本地拷贝。

linux用户可以使用系统带有的包管理系统可以非常快的获取最近的maven版本。另外，标准方式是下载二进制版本，将其解压到/usr /local/maven，然后编辑bash配置文件~/.bashrc，添加一行export PATH=/usr/local/maven/bin:$PATH。保证maven命令行一直有效。

如果你使用IDE，像Eclipse或者IntelliJ，他们已经集成了Maven。参考文档它的文档去了解如何将Maven整合。这将在IDE中使用Mahout变得简单一些，因为IDE可以根据那个Maven配置文件（pim.xml）配置、导入工程。

**1.6.3 安装Mahout**

Mahout仍在开发过程中，这本书写的时候Mahout是0.4版本。可以在这里下载发布版本和其他介绍http://lucene.apache.org/mahout/releases.html  源文件可以解压到你电脑的任何地方。

因为Mahout的更新很频繁，有规律的进行漏洞修复、功能改进，在实际使用过程中，使用最新发布版本的软件（或者使用svn上的最近的未发布 代码）可能对你的帮助更大，详见http://lucene.apache.org/mahout/developer- resources.html)。未来的发行版本应该向后兼容本书的示例。

一旦你从svn或者发布的归档文件中获取了源代码，在IDE中创建一个Mahout的新工程。这个对不同IDE是过程不同的，需根据如何完成创建的文档细节区别对待。使用IDE的Maven整合用工程根目录下的pom.xml导入Maven工程是很简单的。

一旦配置完成，你可以很容易的创建一个新的源代码文件夹来管理下面章节提到的样例代码。随着工程的配置完成，你应该可以透明的编译和运行代码，不需要再做其他工作。

**1.6.4 安装hadoop**

本书后面的一些行为，你需要在本机安装hadoop。你不需要一个集群去运行hadoop。配置hadoop不是很麻烦、繁琐。不重复这个过 程，我们直接指导你在hadoop官网行获取hadoop 0.20.2版本的副本，http://hadoop.apache.org/common/releases.html，然后使用“伪分布式”方式安装 hadoop，详见http://hadoop.apache.org/common/docs/current/quickstart.html。

**1.7 总结**

Mahout是一个年轻的、开源的、可扩展的，Apache的机器学习库，本书是一本使用Mahout中的机器学习算法解决现实问题的使用指 南。尤其是，你可以很快的探索推荐引擎、聚类、分类。如果你是一个机器学习领域的研究者，正在寻找一个实用指导、又或者你是这个领域的开发者，想从其他从 业者处快速学习实用方法，本书是非常适合你的。

这些技术不仅仅是路论：我们已经注意到很多知名的推荐引擎、聚类、分类的样例在现实世界中已经在应用：电子商务、邮件、视频、图片，更涉及大规模的机器学习算法。这些技术已经用来解决现实问题，甚至为企业产生了价值 — 这些现在在Mahout中都是可以触及的。

我们已经注意到很多时候这些技术时常伴随着海量的数据 — 可扩展性是这个领域特别需要持续关注的。我们首先来看一下MapReduce和hadoop，以及他们如何为Mahout提供可扩展支持。

因为这是一本动手实践的、实用书籍，我们马上开始使用Mahout。此刻，你应该已经安装好了Mahout需要的工具并且马上开始行动。因为这本书以实用为主，所以将这些开幕词收起来去见识一下Mahout的代码吧。继续读下去。

# mahout实战 第2章 推荐系统简介 2.1 推荐系统是什么

**第2章 推荐系统简介**

本章包含以下内容：

* 首先看一下实战中的推荐系统
* 推荐引擎的精度评价
* 评价一个引擎的准确率和召回率
* 在真实数据集：GroupLens 上评价推荐系统

我们每天都会对喜欢的、不喜欢的、甚至不关心的事情有很多观点。这些事情往往发生的不知不觉。你在收音机上听歌，因为它容易记住或者因为听起来可怕而关注它 — 又或者根本不去关注它。同样的事情有可能发生在T恤衫，色拉，发型，滑雪胜地，面孔，电视节目。

尽管人们的爱好差异很大，但他们仍然遵循某种模式。人们倾向于喜欢一些事物，这些事物类似于他们自己喜欢的其他事物。因为我喜欢培根-生菜-西 红柿三明治，你可能猜到我可能也喜欢总汇三明治，类似于前者的带有火鸡肉的三明治。另外，人们倾向于喜欢兴趣和他们相似的人所喜欢的事物。当一个朋友进入 设计学校，发现几乎周围所有同学都用一个苹果电脑 — 不用吃惊，他已经是一个终生的苹果用户。

这些模式可以用来预测喜好与否。如果将一个陌生人带到你面前并问你，她是否喜欢指环王三部曲，你只能胡乱猜测了。但是，如果她告诉我们她喜欢头 两部指环王，如果她不喜欢第3部，你会感到震惊。另一种情况，如果她说讨厌指环或者问“什么王？”，你可以准确的猜测她对指环王不感兴趣。

推荐系统是关于这些预测偏好模型的系统，并且使用他们为你发现新的、称心的事物。

**2.1 推荐系统是什么**

你因为某种原因从书架上拿起这本书。可能你看到它临近其他的你熟知的书，发现他们是有用的，想到书店是因为喜欢那些书的人也会喜欢这本书才把他们放到一起。也可能你看到这本书在你同事的书架上，你和这个同事经常分享机器学习方面的知识，或者他直接将这本书推荐给你。

后面的章节探索了人们实现推荐系统的很多方法，发现新事物，当然有这些过程在Mahout软件中是如何实现的。我们已经提到一些策略：为了发现 你喜欢的items，你可能去寻找有相同兴趣的人。另一方面，你可以由其他人的明显偏好计算出类似你喜欢的items的其他items。这描述了两大类推 荐系统算法：“user-based”和“item-based”推荐系统。

**2.1.1 协同过滤 vs 基于内容的推荐**

严格的讲，上面章节提到例子是“协同过滤” — 仅仅基于user和item关系的生产推荐系统。这些技术不需要item自己的属性信息。在某种程度上，这是一个优点，该推荐系统框架不需要关心 “item”是书、主题公园、花、甚至人，因为他们的任何属性都没有作为输入数据。

有很多其他的方法基于item的属性，一般被称之为“基于内容”的推荐技术。例如，如果一个朋友推荐一本书给你，因为这是一本Manning出 版的书，这个朋友喜欢其他Manning的书，那么你的这个朋友在做类似于基于内容推荐的事情。这种想法基于书籍的属性：出版社。Mahout推荐框架没 有直接实现这些技术，尽管它提供一些方法将item的属性信息添加到计算中。基于这些，技术上称前面的方法为协同过滤框架。

这些技术没有没什么问题；正相反，他们工作的很好。他们是特定领域的方法，可能很难有效地将他们整理到一个框架中。为了建立一个基于内容的图书 推荐系统，我们需要决定书籍的哪些属性对推荐系统是有意义的、作用多少程度 — 页数、作者、出版社、颜色、字体。但是这些知识不能简单的应用于别的领域；像图书推荐的方法对披萨配料推荐是没什么帮助的。

基于这个原因，Mahout没有过多涉及这种类型的推荐。可以将这些想法融入进来，但首要任务是Mahout提供些什么。其中的一个样例将在下 一个章节中讲解，在那里你将建立一个约会网站的推荐引擎。在介绍完Mahout基于协同过滤推荐的实现后，你有机会探索它和基于内容的推荐之间关系的细 节。

**2.1.2 推荐系统成为主流**

现在，很多人已经看到了推荐系统在现实网站上的实现，像Amazon或者Netflix：基于浏览和交易历史，页面将产生一个认为对你有吸引力 的商品列表。这类推荐引擎在上世纪九十年代已经出现，但直到现在它仍然是使用大型计算机的专门研究者的领域。因为这些技术已经变更加主流，对它们的需求在 增加，开源实现的提供也是一样。随着理解的深入和计算资源越来越廉价，推荐引擎变得越来越容易接近和广泛使用。

实际上，推荐技术不只是为用户推荐DVD类似的东西。这种方法非常通用，可以用来评估很多事物之间的关系的强弱。

在社区网络中，一个推荐系统可以为用户推荐用户。

**2.2 运行你的第一个推荐引擎**

Mahout包含了一个推荐引擎 — 有很多类型，实际上都是源于传统的user-based和item-based推荐系统。它还包含了基于“slope-one”技术的实现，一个新的、有 效的方法。你将会找到很多实验性的、初步的的SVD（singular value decomposition）实现。下面的章节将重新看Mahout的内容。在这些章节中，你将看到数据展示、浏览可用的推荐算法，评价推荐系统的有效 性，针对特殊问题协调和定制推荐系统，最后看一下分布式计算。

**2.2.1 创建输入数据**

为了探索Mahout中的推荐系统，最好从一个很小的示例入手。推荐系统的输入是必要的 — 这些数据是推荐的基础。因为非常熟悉的推荐系统引擎将item推荐给user，很容易的认为偏好是user和item之间的联系 — 尽管前面提到了user和item可以是任何事物。偏好包含了一个user ID和一个item ID，通常情况下，一个数值代表了user对item偏好的强度。Mahout中ID都是数值，实际上是整数。偏好值可以是任何值，值越大代表这正向偏好 越大。例如，这些值可能是1到5的打分，user将1给于他不喜欢的，5给他很喜欢的。

创建一个文本文件包含用户数据，命名为“1”到“5”，他们对四本书的偏好，简单的称之为“101”到“104”。在现实情况中，这些可能是公 司数据库中消费者ID和产品ID；Mahout并不需要user和item的ID一定为数值类型。使用下面的格式，生成文件intro.csv。

1,101,5.0

1,102,3.0

1,103,2.5

2,101,2.0

2,102,2.5

2,103,5.0

2,104,2.0

3,101,2.5

3,104,4.0

3,105,4.5

3,107,5.0

4,101,5.0

4,103,3.0

4,104,4.5

4,106,4.0

5,101,4.0

5,102,3.0

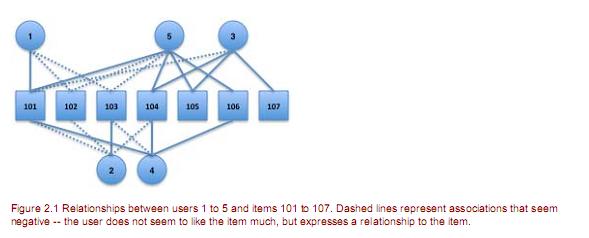
5,103,2.0

5,104,4.0

5,105,3.5

5,106,4.0

经过一些学习之后，趋势就显现出来了。用户1和用户5具有相同的兴趣。他们都喜欢101这本书，对102的喜欢弱一些，对103的喜欢更弱。同 理，用户1和4具有相同的兴趣，他们都喜欢101和103，没有信息显示用户4喜欢102。另一方面，用户1和用户2的兴趣好像正好相反，用户1喜欢 101，但用户2讨厌101，用户1喜欢103而用户2正好相反。用户1和3的交集很少，只有101这本书显示了他们的兴趣。看图2.1可能显现了 user和item之间的关系，可能是正的也可能是负的。



2.2.2 创建推荐系统

那么你应该给用户1推荐哪本书？不是101, 102或者103，因为用户已经知道自己对他们感兴趣，推荐系统需要发现新的事物。直觉告诉我们，用户4、5与用户1类似，所以推荐一些用户4和5喜欢的 书籍给用户1可能是不错的。这样使得104、105和106成为可能的推荐。整体上看，104是最有可能的一个推荐，这基于item 104的4.5和4.0的偏好打分。现在运行下面的代码：

Listing 2.2 a simple user-based recommender program with mahout 代码  收藏代码

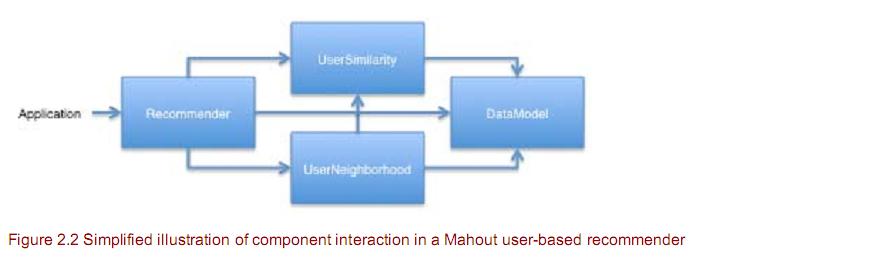
1. package mia.recommender.ch02;
2. import org.apache.mahout.cf.taste.impl.model.file.\*;
3. import org.apache.mahout.cf.taste.impl.neighborhood.\*;
4. import org.apache.mahout.cf.taste.impl.recommender.\*;
5. import org.apache.mahout.cf.taste.impl.similarity.\*;
6. import org.apache.mahout.cf.taste.model.\*;
7. import org.apache.mahout.cf.taste.neighborhood.\*;
8. import org.apache.mahout.cf.taste.recommender.\*;
9. import org.apache.mahout.cf.taste.similarity.\*;
10. import java.io.\*;
11. import java.util.\*;
12. class RecommenderIntro {
13. private RecommenderIntro() {
14. }
15. public static void main(String[] args) throws Exception {
16. DataModel model = new FileDataModel(new File(“intro.csv”);  A
17. UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
18. UserNeighborhood neighborhood = new NearestNUserNeighborhood(2,
19. similarity, model);
20. Recommender recommender = new GenericUserBasedRecommender(model,
21. neighborhood, similarity);   B
22. List<RecommendedItem> recommendations = recommender.recommend(1, 1);  C
23. for (RecommendedItem recommendation : recommendations) {
24. System.out.println(recommendation);
25. }
26. }
27. }

A 加载数据文件

B 创建推荐系统引擎Create the recommender engine

C 对user1, 推荐一个item

为了简洁，后面许多其他章节的示例中，代码列表将省略imports、类声明、方法声明，只是重复程序语句。为展示很好的展示各个模块之间的关系，请看图2.2。并不是Mahout中所有的推荐都是这样的，但这幅图可以给你一个样例的逻辑的初步印象。



接下来两章，在更细节的讨论这些模块之前，我们可以总结一下每个模块所扮演的角色。DataModel存储了所有的偏好信息，提供了对user 和item信息的访问。UserSimiliarity提供了两个用户如何相似的概念，这可能基于很多可能的矩阵和计算之一。 UserNeighborhood定义了一个给定用户的用户组的概念。最终，一个推荐系统将这些模块组合在一起将items推荐给users和相关功能。

**2.2.3 分析输出**

使用你细化的IDE编译运行，运行程序的输出应该是：RecommendedItem[item:104, value:4.257081]

请求一个推荐结果并得到一个。推荐系统引擎将书104推荐给用户1。甚至，这样做是因为推荐系统引擎将用户1对书104的偏好是4.3，这是所有推荐结果的最高打分。

这个结果并不算坏。107没有出现，本应该也是可以推荐的，但它只是和另一个具有不同爱好的user相关联。选104而不是106，因为104的打分高一些。还有，输出结果包含了一个用户1喜欢104的评估值 — 是介于用户4和5所表示的介于4.0和4.5的一个值。

直接看数据正确的结果并不明显，但是推荐系统引擎推荐了一个得体的结果。如果对从这个简单程序给出的有用并不明显的结果感到有一种愉快的刺痛，那么机器学习的世界是适合你的。

小数据集、产生推荐结果是一件微不足道的事情。现实中，数据集很大，并且噪声数据很多。例如，想象一个新闻网站推荐新闻给用户。偏好从文章的点 击中获取。但是，这里面的很多点击都是伪造的 — 可能很多读者点击一篇文章但他不一定喜欢它，或者点错了。可能很多点击都是在未登录的时候发生的，不能将其关联到一个用户。想象一下数据集有多大，可能是 每月几十亿的点击量。

要从数据集产生正确的推荐结果并快速计算出是一件不一般的事情。稍后我们将展示工具Mahout如何解决这些问题。他将展示标准方法如何产生差的推荐结果或者占用了大量的cpu和内存时间，如何配置Mahout以提升性能。

**2.3 评价推荐系统**

推荐系统引擎是一个工具，一种回答问题的手段，“对用户来讲什么是最好的推荐？”，在研究回答的前先研究一下这个问题。一个好的推荐的准确含义是什么？如何知道推荐系统是如何生成推荐的？下面的章节将探索推荐系统的评价，在寻找特定推荐系统时，这将是一个有用的工具。

最好的推荐系统是心理学的范畴，有人在你做事情之前知道确切的知道你还没有看过的、或者没有任何现象说明你喜欢的一些item，以及你对这些item的喜欢程度。

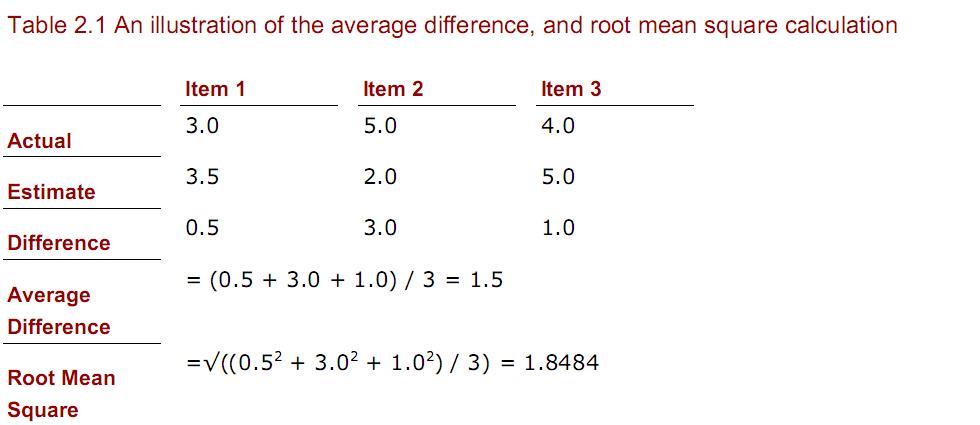
大部分的推荐引擎通过给item评价打分来实现。所以，评价推荐引擎的一种方式是评价它的评估偏好值的质量 — 评价评估偏好和实际偏好的匹配度。

2.3.1 训练集和打分

“真实偏好”并不充分，没有人会知道你将来是否会喜欢一些新的item。推荐引擎可以通过设置一部分真实数据作为测试数据。这些测试数据偏好在训练集中并不展示偏好值 — 要求推荐系统对这些缺少偏好值的数据作出评估，并比较和实际值的差距。

对于推荐系统产生一系列打分是很简单的。例如，计算评估值和实际值之间的平均距离，在这种方法下，分值越低越好。0.0表示非常好的评估 — 评估值和实际值根本没有差距。

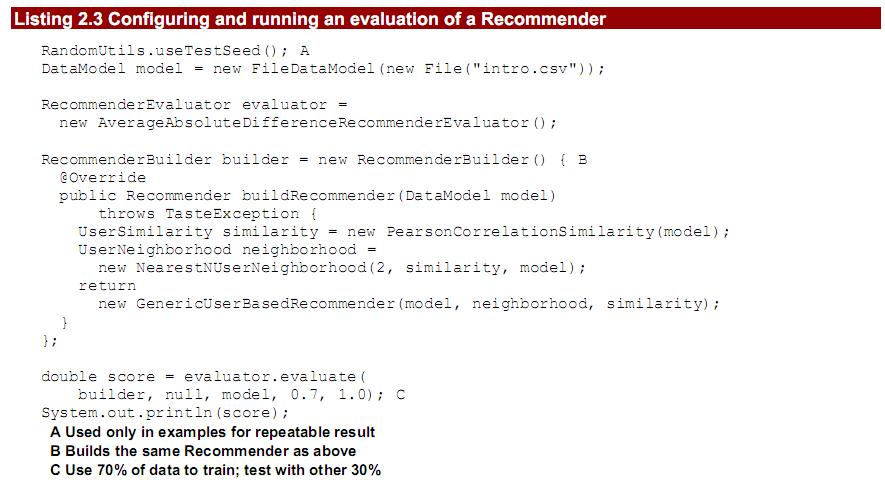
均方根（root-mean-square）也是一种方法，也是分值越低越好。



上面的表中展示了实际偏好度和评估偏好度集合的不同值，以及如何将它们转化为打分。均方根能比较重的处罚距离远的，例如item2，这是基于某种考虑在内的。因为平均距离容易理解，接下来的示例将使用它作为评估方法。

**2.3.1 运行RecommenderEvaluator**

下面是代码示例：



大部分的操作发生在evaluate()这个方法中。内部，RecommenderEvaluator将数据划分为训练集和测试集，创建一个新的训练DataModel和推荐引擎测试，比价评估结果和实际结果。

注意，没有将Recommender传给方法，这是因为在其内部，将基于创建的训练集的DataModel创建一个Recommender。所以调用者必须提供一个RecommenderBuilder对象用于从DataModel创建Recommender。

**2.3.3 评估结果**

程序打印出了评估结果：一个表明推荐系统表现如何的打分。在这种情况下你能看到很简单的1.0。尽管评价器内部有很多随机方法去选择测试数据， 结果可能是一致的因为RandomUtils.useTestSeed()的使用，每次选取的随机数都一样。这只用于示例、单元测试来保证重复的结果。不 要在真是数据上用它。

AverageAbsoluteDifferenceRecommenderEvaluator

基于AverageAbsoluteDifferenceRecommenderEvaluator实现，得到的这个值是什么含义？1.0意味着，平均意义上，推荐系统评估偏好和实际偏好的的距离是1.0.

1.0早1-5规模上并不大，但是我们的数据太少。如果数据集被随机划分结果可能不一样，因此训练、测试数据集可能每次跑都不一样。

这种技术可以应用于任何Recommender和DataModel。使用均方根打分的实现类RMSRecommenderEvaluator

替代AverageAbsoluteDifferenceRecommenderEvaluator。

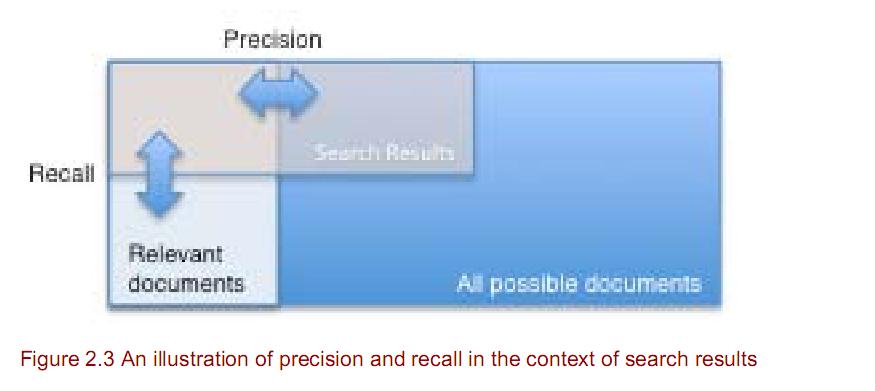
evaluate()的null参数是DataModelBuilder的实例，用于控制训练DataModel是如何从训练数据上建立的。正 常情况下默认就好，如果需要，可以使用特别实现的DataModel。DataModelBuilder用于将DataModel注入评价过程中。

参数1.0表示使用整个数据集的比例。这样用于产生一个很快的、可能精度低一些的评价方式。例如，0.1可能意味着用数据集的10%，忽略其他90%。这对于快速检测到Recommender的细微变化是非常有用的。

**2.4 评估准确率和召回率**

借用更普遍的看法，我们接收经典的信息检索矩阵去评价推荐系统：准确率和召回率。这些是用于搜索引擎的术语，通过query从众多可能结果中返回最好结果集。

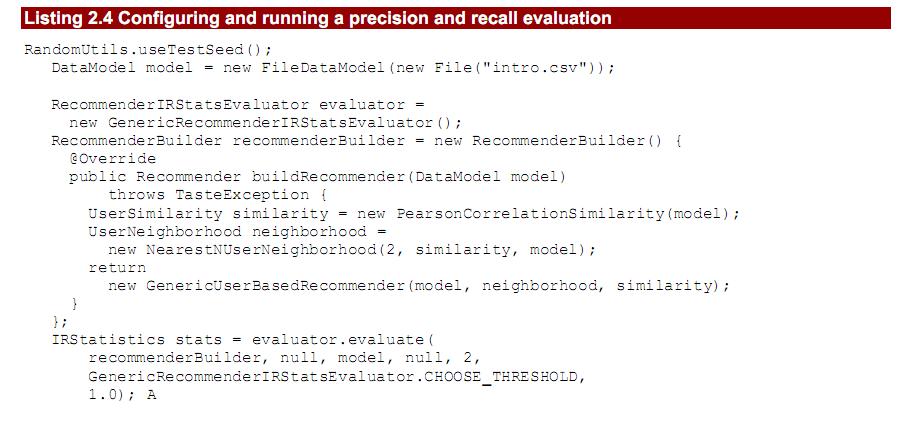
一个搜索引擎不应该在靠前的结果中返回不相关的搜索结果，即使他致力于得到尽可能多的相关结果。”准确率”是指在靠前的结果中相关结果所占的比 例，当然这种相关是某种程度上我们定义的相关。”precision at 10″应该是从前10个结果中判断得到的准确率。“召回率”靠前的结果中相关结果占的比例。看图2.3可以有一些直观的概念。

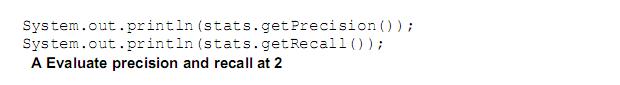


这些术语也可以应用到推荐系统中：准确率是靠前的推荐中好的推荐所占的比例，召回率是指出现在靠前推荐中好的推荐占整个好的推荐的比例。

**2.4.1 运行RecommenderIRStatsEvaluator**

Mahout提供了非常简单的方式为推荐系统计算结果。







**2.5 评价GroupLen数据集**

有了这些工具在手，我们不仅可以考虑速度，还要考虑推荐系统的效率。尽管大量数据集的例子还在几章以后，小数据集上评价性能已成为可能。

**2.5.1 抽取推荐系统输入**

GroupLens (<http://grouplens.org/>)是一个研究型的项目。提供了很多不同大小的数据集。每个都是用户对电影打分的真实数据。这是几个大的真实可用数据集之一，更多的将会在本书后续探寻。从GroupLens网站上下载“100K data set”,[http://www.grouplens.org/node/73.解压下载文件，在解压后后，有一个叫ua.base的文件。该件tab分割user](http://www.grouplens.org/node/73.%E8%A7%A3%E5%8E%8B%E4%B8%8B%E8%BD%BD%E6%96%87%E4%BB%B6%EF%BC%8C%E5%9C%A8%E8%A7%A3%E5%8E%8B%E5%90%8E%E5%90%8E%EF%BC%8C%E6%9C%89%E4%B8%80%E4%B8%AA%E5%8F%ABua.base%E7%9A%84%E6%96%87%E4%BB%B6%E3%80%82%E8%AF%A5%E4%BB%B6tab%E5%88%86%E5%89%B2user) IDs, item IDs, ratings（偏好值）和其他信息。

这个文件能使用吗？Tabs, 不是逗号，分隔字段，在每行结尾还有一个额外的字段。答案是肯定的。该文件可以类似FileDataModel的读取。回到前面列表2.3，试着用 ua.base的路径代替小文件路径。重新跑一遍数据。这时，评估可能需要几分钟，因为数据量是100k。

最后，你应该得到一个大约0.9的值。这不算坏，尽管值几乎分布在1-5区间内，听起来还不错。可能特殊的推荐系统对这种类型的数据来讲是不完全的？

**2.5.2 其他推荐系统实验**

在这个数据集上测试“slope-one” ，一个简单的算法，在后面的章节中会讲到。很容易替换RecommenderBuilder，使用 org.apache.mahout.cf.taste.impl.recommender.slopeone.SlopeOneRecommeder, 像这样：

[](http://www.adintellig.com/wp-content/uploads/2013/06/1.jpg)

运行评价方法。你应该发现它很快，得到一个大约是0.748的值。正朝好的方向发展。

这并不能说明slope-one算法总是很快、很好。在给定数据集上，每一个算法都有自己的特性和属性，这些属性很难去检测出来。例 如，slope-one算法运行时能够很快的计算推荐过程，但是它需要在计算前需要花费很大一部分时间构建内部数据结构。基于用户的推荐引擎，能够在其他 数据集上拥有很快的计算速度和更高的准确度，我们将在第四章探索各种算法的优点。

**2.6 总结**

在这一章我们介绍了推荐引擎的思想。我们创建了一些简单的Mahout Recommender的输入，运行一个简单计算，然后输出结果。

然后，我们花时间介绍了推荐引擎系统的数据结果的质量评价，因为接下来的章节会频繁用到。这一章包含了评价推荐引擎的准确性，像传统的准确性和召回率。最后，我们尝试去评价GroupLens的真实数据集，观察如何将评价结果用于推荐引擎的性能提升上。

在我们继续详细学推荐引擎之前，需要花一些时间去了解另一个Mahout中推荐系统的基本概念：数据表示。接下来一章会着重讲这一点。

# ****mahout in action 3 数据展现****

这章主要讲述：   
。Mahout如何展现recommender data   
。DataModel的实现及其用法   
。没有评分数据

Recommendations的质量主要是由数据的数量和质量决定的。“无用输出，无用输入” 在这里是最真实的。同样，推荐器算法都是集中数据，运行的性能主要受数据的数量和展现的影响。这一章介绍Mahout的一些关键class，和访问推荐器相关的数据。

**mahout in action 3.1 Preference对象**

一个推荐引擎的输入数据是评分数据：它喜欢什么以及多少。所以，Mahout recommenders的输入数据是一组简单的“userID”,“itemID”,和“评分数据”元组，当然，这是一个大的集合。评分数据有时候会被省略。

**3.1.1 Preference对象**

Preference 是一个最基础的概念，它表现一个单一的userID,itemID和一个评分数据。这个对象表现为一个用户对一个项目的打分。Preference是一个 接口，通常使用的实现类为GenericPreference。例如：创建一条记录，user（123），对item（456）的打分是3.0： new GenericPreference(123, 456, 3.0f)。

一组Preferences如何表现？如果你给出了一个合 理的答案，如Collection<Preference> 或者 Preference[]，大部分情况下，在Mahout APIs中不是这样实现的。Collections和arrays对处理海量Preference对象是无效的。如果你在Java中从未研究过上面的对 象，你可能会感到困惑！

单个的GenericPreference包含20个字节的有用数据：一个8字节的user ID(Java long),8字节的item ID(long),4字节的分值(float)。这个对象的存在使GenericPreference包含的字节有惊人的增长：28个字节！这个变化依赖 的是JVM的实现；这个数字是从苹果Mac OS X 10.6的64位Java 6 VM 得到的。由于上面的对象和其他线性问题，对这个对象来说，28个字节中包括8字节的参考值，另外20个字节的空格，在对象自身的表现内。由于上面的现象， 因此一个GenericPreference对象已经比它需要多消耗了140%的存储。

为什么这么做？在recommender算法中，都需要所有评分数据的集合，这些评分数据是与一个用户或一个项目联系在一起的。在这样一个集合里，user ID或者item ID与所有好像多余的Preference对象将会是配套的。

**3.1.2 PreferenceArray和实现**

进 入PreferenceArray，这个接口的实现表现为一个具有类似与数组的API的分值的集合。例 如，GenericUserPreferenceArray表现为一个用户的所有打分.它在内部包括一个单一的user ID,一系列的item IDs,一系列的评分值。在一个用户的所有打分中，需要占用12个字节的内存（一个8字节的item ID和一个4字节的评分值）。把它与需要一个完整的Preference项目的大约48个字节相比较。这个4字节内存，包括对齐这个特殊的实现，   
但它也提供了小的性能提升，更小的对象必须被垃圾回收器分配和检查。比较图3.1 and 3.2去理解这些保存是如何完成的。   
图3.1效率较低的评分值的表现，利用一系列的Preference对象。灰色的区域代表上面的对象。白色的区域是数据，它包括引用对象。   
图3.2利用GenericUserPreferenceArray更有效的表现

下面的代码表现一个PreferenceArray的典型的构造和使用   
列表3.1在一个PreferenceArray中设置评分值   
Java代码：   
PreferenceArray user1Prefs = new GenericUserPreferenceArray(2);      
user1Prefs.setUserID(0, 1L); //A      
user1Prefs.setItemID(0, 101L);      
user1Prefs.setValue(0, 2.0f); //B      
user1Prefs.setItemID(1, 102L);      
user1Prefs.setValue(1, 3.0f); //C      
Preference pref = user1Prefs.get(1); //D

PreferenceArray user1Prefs = new GenericUserPreferenceArray(2);   
user1Prefs.setUserID(0, 1L); //A   
user1Prefs.setItemID(0, 101L);   
user1Prefs.setValue(0, 2.0f); //B   
user1Prefs.setItemID(1, 102L);   
user1Prefs.setValue(1, 3.0f); //C   
Preference pref = user1Prefs.get(1); //D   
A 为所有打分设置user ID   
B User 1当前为item 101的打分2.0   
C User 1为item 102的打分3.0   
D Item 102的一个Preference实现

同样这里存在一个称为GenericItemPreferenceArray的实现，它内部的所有分值，与item关联而不是与user关联。它的目的和用法都是完全类似的。

**mahout in action 3.2 加速聚集**

非常高兴的是，Mahout已经重新创造了“java数组对象”。这只是万里长征的第一步。我们提及到规模是重要的吗？可能，你已经被说服，我们将会面对处理巨大数量的数据，和不寻常响应。

这 个reduced的内存需求，由PreferenceArray和它的实现，带来的复杂性是值得的。削减内存需求的百分之七十五不只是节约一对M字节。在 一个合理的规模上，它节约了10分之一G内存。这可能是在你现存的硬盘上是否装配之间的不同。这是是否必须投资大量的RAM和可能的一个新的64-bit 系统之间的不同。那是一个小的，但真正节能的技术，非常重要。

**3.2.1 FastByIDMap 和 FastIDSet**当 你听到Mahout recommenders大量的使用如map和set的典型的数据结构时将不会感到奇怪，但是不要使用如TreeSet和HashMap的普通的Java 实现。相反，遍历这个实现和API你将会找到FastByIDMap和FastIDSet。它们是像Map和set一样的程序，但是是被明确的详细说明， 并只提供Mahout recommenders需要的程序。它们减少内存占用而不是在性能上显著的增加。

这里没有一个像java中的Collections。但是，它们在一个大范围的环境内，为有效的目的而精心设计。它们不能对未来的使用做出更多的假设。Mahout的需要对可得到的用法有更加特殊，更强的假设。主要不同是：

。如同HashMap，FastByIDMap 是 hash-based。它使用线性探索而不是分离链接来处理hash collisions。这避免了一个额外的Map.Entry对象的每个入口的需要；如我们所讨论的，Objects占用了令人惊奇的内存数量。

。Keys和members在Mahout recommenders中总是长的基元，而在objects中则不是。使用长的keys节约内存并提高性能。

。Set的实现不是使用下面的一个Map来实现的。

。FastByIDMap可以像一个cache一样起作用，因为它有一个“maximum size”的概念；超过这个尺寸，当增加了新的entries时，infrequently-used entries将会被删除。

存 储的不同是有意义的：与HashSet 的84个字节相比，FastIDSet平均每个member需要大约14个字节。与HashMap 的每个入口的84个字节再次比较，FastByIDMap每个入口占用28个字节。这显示当一个人对用法做了更强的假设时，有意义的改善是可能的：主要在 内存需求上。考虑到为recommender系统提供的讨论中的数据量，这些习惯的实现不仅仅证明了它自己。所以，这些类用在哪里？

**mahout in action 3.3 内存中的DataModel**

这 是个抽象概念，在Mahout中，recommender的输入数据是DataModel。DataModel的实现为各种推荐器算法需要的数据提供了有 效的使用。例如，一个DataModel可以在输入数据中，提供一个包括所有user IDs的列表，或提供与一个item相关联的所有分值，或者提供一个为一系列item IDs打分的所有用户的列表。我们将会集中研究一些highlights；一个关于DataModel的API的更详细的描述，可以通过在线文档中找到。

**3.3.1 GenericDataModel**

这个我们先来看一下，最简单 的实现（在内存中实现），GenericDataModel。当你想用编程的方法，而不是基于一个现存的外部数据资源。例如一个文件或相关数据库在内存中 创建你的数据表现时，这是非常合适的。它只是以这种形式把分值当作输入数据，这个形式就是一个FastByIDMap映射user IDs到有这些用户的数据的PreferenceArrays上。

列表3.2 基于GenericDataModel，定义输入数据   
Java代码    
FastByIDMap<PreferenceArray> preferences =      
new FastByIDMap<PreferenceArray>();      
PreferenceArray prefsForUser1 = new GenericUserPreferenceArray(10); //A      
prefsForUser1.setUserID(0, 1L);      
prefsForUser1.setItemID(0, 101L); //B      
prefsForUser1.setValue(0, 3.0f); //B      
prefsForUser1.setItemID(1, 102L);      
prefsForUser1.setValue(1, 4.5f);      
… (8 more)      
preferences.put(1L, prefsForUser1);// C      
DataModel model = new GenericDataModel(preferences); //D

FastByIDMap<PreferenceArray> preferences =   
new FastByIDMap<PreferenceArray>();   
PreferenceArray prefsForUser1 = new GenericUserPreferenceArray(10); //A   
prefsForUser1.setUserID(0, 1L);   
prefsForUser1.setItemID(0, 101L); //B   
prefsForUser1.setValue(0, 3.0f); //B   
prefsForUser1.setItemID(1, 102L);   
prefsForUser1.setValue(1, 4.5f);   
… (8 more)   
preferences.put(1L, prefsForUser1);// C   
DataModel model = new GenericDataModel(preferences); //D  
A为user 1建立PreferenceArray   
B添加第一个preference，在刚刚创建的10中   
C把user 1的preference添加到输入数据上   
D创建DataModel

一 个GenericDataModel使用多少内存？储存的分值的数目占内存占用的绝对优势。通过一些经验揭示，每一preference占用28个字节的 Java heap space 。它包括所有的数据和其他次要数据结构--如指数。如果你喜欢你也可以尝试一下;下载一个GenericDataModel，调用 System.gc() ，几次后，比较Runtime.totalMemory()和Runtime.freeMemory()的结构。这是未加工过的，但应该可以给出一个合理 的估计，这个估计就是数据占有多少内存。

3.3.2 基于文件的数据

通常我们不会直接地使用GenericDataModel，而是可能使用FileDataModel：  FileDataModel从一个文件中读取数据，并可以在内存中储存作为结果的分值数据，从而转化为GenericDataModel。

几 乎任何一个合理的文件都将会这么做。我们在第一节里已经看到了一个这样文件的例子，在这节里，我们创造了一个简单的用逗号分割数据的文件，在这个文件里， 每一行都包含一个数据：user ID,item ID,分值。使用Tab分割的文件也同样这么做。如果它们的名字各自以“.zip” 或“.gz”为后缀，使用对应的zip和gzip解压。在压缩格式储存这一数据是一个好想法，因为它是巨大的，并且被压缩好的。

**3.3.3 Refreshable组件**

## 3.3 处理惩罚偏好值为空的数据（布尔偏好）

　　有时推荐引擎中呈现偏好值为空的记录。它代表了用户和项目是接洽关系的，然则并没有发挥解析出接洽关系程度。举了例子，一个消息网站按 照用户已浏览内容为用户推荐消息。“已浏览”使一个用户和一个项目产生了接洽关系，然而这是独一可以或许获取的信息。一般网站也不会让用户去给文章做个排 序，更不会让用户再做除了浏览之外的其他什么事了。所以我们仅仅知道用户和那些文章接洽关系了，而再也没有其他的内容了。

　　面对如许的景象，我们别无选择。这里不会有偏好值。后续几章将会依然供给处理惩罚如此景象的技巧和建议。然而有时我们忽视掉偏好值也未尝不是坏事，只要景象须要。

　　丢掉用户和项目之间的接洽很轻易，或者说我们可以直接忽视偏好值就可以办到。比如，你宁可推敲有没有看过一个电影，也不肯去给一个电影 打分。换句话说，我们宁肯把数据改为“用户1和项目3有关系”，也不肯意写成“用户1喜好项目103的程度为4.5”。下面是一个示意图，来申明二者之间 的差别：

**图3.4 带偏好值的数据（左）与布尔偏好数据（有）的差别**

　　用Mahout的说话，没有不带偏好值的记录就是布尔偏好，因为一个接洽关系只有两种值：要么有接洽关系，要么没接洽关系。这不代表偏好只有两个值：是和否，而是有三个：喜好、不喜好、不知道。

### 3.3.1 什么时辰应当忽视偏好值

　　为何有时会忽视偏好值？因为会有如许的景象，用户喜好或者不喜好一个项目标程度相对一致，至少与那些和用户没有接洽关系的项目比拟。还 记得那个例子吗？一个家伙不喜好拉赫玛尼诺夫（Rachmaninoff），当然世上有很多曲子他没有听过（比如灭亡金属音乐（Norwegian death metal））。如许我们只获得了他与这个作家的一个接洽关系，刚巧他又喜好另一个作家布拉姆斯，然后他把拉赫玛尼诺夫标为1，把布拉姆斯标为5，和其余 未知作家比拟，如许的偏好值是没有什么意义的，认为二者在某种程度上是对等的。所以我们索性不要这个偏好值，仅仅是喜好、不喜好、不知道或者说未知的关 系。

　　你可能会说这都是用户的错。莫非他就不克不及为拉赫玛尼诺夫打4分而为灭亡金属打1分。也容许以，但你还是认了吧！因为事实上推荐所须 要输入的数据是很难断定的。你或许还要提出否决，固然对于所有流派都可以如许去推理，然则当你要向他推荐古典作曲家时该根据哪些数据呢？没错，在某范畴中 的好的解决计齐截般不会公开出来。

### 3.3.2 布尔偏好在内存中的数据表达

　　抛去偏好值可以很大程度上化简数据表达，当然也会提拔机能、占用更少的内存。像看我之前懂得的，在Mahout中须要用4个字节的浮点 数去存储偏好值，至少若是没有偏好值也就意味着我们每一条会罕用4个字节。在内存方面的测试中也确切均匀每笔记录都削减了4个字节到24字节了。

　　这个测试用的是GenericDataModel的双胞胎兄弟GenericBooleanPrefDataModel。它看上去是一 个内存版的实现，然则它没有偏好值。事实上，它仅仅用FastIDSets去存储一个用户和项目标接洽关系。例如，只为每个用户存储和他接洽关系的所有项 目ID，这里不会再呈现偏好值。

　　因为GenericBooleanPrefDataModel也是DataModel对象，所以它完全可以直接庖代 GenericDataModel。一些数据模型的办法在这种实现下会变得快速高效，比如：getItemIDsForUser（），因为该办法在它里面 取项目ID是现成的。某些办法也会变慢，比如：getPreferencesFromUser（），因为它的实现没有应用 PreferenceArrays，并且必须实现以下该办法。

　　你可能会好奇getPreferenceValue（）在这里可以返回什么？因为这里不是没有偏好值吗？你调用该办法并不会抛出 UnsupportedOperationException的异常，它会恒定的返回一个子虚值1.0。这一点值得重视，每个组件都邑依附于一个偏好值， 它们必须从DataModel对象中获取一个。这些捏造并且固定的值也会产生一些奥妙的题目。

　　让我们在文章最后来调查一下GroupLens例子的成果。下面是应用了GenericBooleanPrefDataModel的代码片段：

**清单 3.7 用布尔偏好值创建和评估一个推荐器**

DataModel model = new GenericBooleanPrefDataModel（

GenericBooleanPrefDataModel.toDataMap（

new FileDataModel（new File（"ua.base"））））; A

RecommenderEvaluator evaluator =

new AverageAbsoluteDifferenceRecommenderEvaluator（）;

RecommenderBuilder recommenderBuilder = new RecommenderBuilder（） {

public Recommender buildRecommender（DataModel model）

throws TasteException {

UserSimilarity similarity = new PearsonCorrelationSimilarity（model）;

UserNeighborhood neighborhood =

new NearestNUserNeighborhood（10， similarity， model）;

return

new GenericUserBasedRecommender（model， neighborhood， similarity）;

}

};

DataModelBuilder modelBuilder = new DataModelBuilder（） {

public DataModel buildDataModel（

FastByIDMap<PreferenceArray> trainingData） {

return new GenericBooleanPrefDataModel（

GenericBooleanPrefDataModel.toDataMap（trainingData））; B

}

};

double score = evaluator.evaluate（

recommenderBuilder， modelBuilder， model， 0.9， 1.0）;

System.out.println（score）;

**A 基于雷同的数据，应用GenericBooleanPrefDataModel**

**B 建树一个GenericBooleanPrefDataModel**

　　这里的转折是DataModelBuilder。它可以使评估法度为练习数据机关DataModel对象，而不是去机关一个 GenericDataModel对象。GenericBooleanPrefDataModel用一种稍微不合的体式格式去获取数据——用一些 FastIDSets而不是PreferenceArrays。并且它还供给了一个很便利的办法toDataMap（）用来对二者进行转换。在进入下一末 节时，测验测验运行一下这个法度，他将不会很成功的履行完毕。

### 3.3.3 选择匹配的实现

　　你会发明在PearsonCorrelationSimilarity的机关办法中抛出了 IllegalArgumentException。刚开端你可能感觉很不成思议，GenericBooleanPrefDataModel莫非不也是 DataModel对象吗？它与GenericDataModel的独一差别不就是没有偏好值吗？

　　这个类似度怀抱对象EuclideanDistanceSimilarity拒绝在没有偏好值的景象下工作，因为如许产生的成果是无意 义的。皮尔森相干系数（Pearson correlation）在两个数据集数据完全反复、一样时就是没有意义的。在这里DataModel的偏好值全部为1.0，那么策画出来的欧几里得间隔 只能在一个点空间（1.0， 1.0， …， 1.0）中进行，如许是毫无意义的，因为所有的类似度都是0。

　　大体上讲，不是所有的实现都可以在一路很好的匹配在一路，甚至都实现自一个接口的对象也不成能完全匹配。为懂得决这个当前题目，我们须 要调换掉这个策画类似度的类。LogLikelihoodSimilarity就是一个好的选择，因为它不须要偏好值，我们将即速评论辩论它。在法度顶用 它调换PearsonCorrelationSimilarity，这个成果就变成了0.0。很不错吧！这意味着它的成果正确了，不过这真的很好吗？

　　不幸的是，所有的偏好值都是1当然会导致所有的偏好相差为0了。这个测试本身没有什么意义，因为它将永远都是0。

　　然而，查准-召回评估还是依然有效的，让我们来试一下吧：

**清单3.8 应用布尔数据评估查准率和召回率**

DataModel model = new GenericBooleanPrefDataModel（

new FileDataModel（new File（"ua.base"）））;

RecommenderIRStatsEvaluator evaluator =

new GenericRecommenderIRStatsEvaluator（）;

RecommenderBuilder recommenderBuilder = new RecommenderBuilder（） {

＠Override

public Recommender buildRecommender（DataModel model） {

UserSimilarity similarity = new LogLikelihoodSimilarity（model）;

UserNeighborhood neighborhood =

new NearestNUserNeighborhood（10， similarity， model）;

return new GenericBooleanPrefUserBasedRecommender（ model， neighborhood， similarity）;

}

};

DataModelBuilder modelBuilder = new DataModelBuilder（） {

＠Override

public DataModel buildDataModel（FastByIDMap<PreferenceArray> trainingData） {

return new GenericBooleanPrefDataModel（

GenericBooleanPrefDataModel.toDataMap（trainingData））;

}

};

IRStatistics stats = evaluator.evaluate（

recommenderBuilder， modelBuilder， model， null， 10，

GenericRecommenderIRStatsEvaluator.CHOOSE\_THRESHOLD， 1.0）;

System.out.println（stats.getPrecision（））;

System.out.println（stats.getRecall（））;

　　评估成果为查准率和召回率均为15.5％，这个成果并不睬想。这意味着推荐成果中6个只有1个是好的推荐，6个好的推荐中只推荐出了1个。

　　这引出了第三个题目，偏好值依然埋伏于GenericUserBasedRecommender对象之中。因为推荐成果遵守偏好值排 序，而偏好值满是1.0，所以如许的次序是完全随机的。所以我们引出GenericBooleanPrefUserBasedRecommender（是 的，正如它名字一样，它可以办到）。它可以在推荐成果中产生一个很有意义的排序。它为每个项目接洽关系于其他类似的用户加权重，用户越类似，那么权重也会 越高。它没有供给一个加权均匀数。调换之后在运行代码，成果可能在18％，好了一点，但没有好太多。这种成果预示着推荐体系对于这些数据不是很是有效，我 们的目标不是要去批改它，仅仅是去看如安在Mahout中有效的安排如许带有“布尔偏好”的数据。

　　其他DataModel对象的布尔变种也是有的。FileDataModel在记录中没有偏好值的景象下内部会主动的应用一个 GenericBooleanPrefDataModel的对象。类似的，MySQLBooleanPrefDataModel适实用在数据库中无偏好值 的景象。它与之前的对象完全类似，并且它的实现可以充沛的哄骗数据库供给的某些捷径来提拔自身的机能。

　　最后，你若是好奇布尔型和非布尔型是不是可以混在一路处理惩罚，答案是否定的。在这种景象下，我们更偏向于遵守存在偏好值的景象去向理 惩罚，因为毕竟成果有些偏好值是存在的。那些没有偏好值的记录可能或者应当可以经由过程某些手段揣摩出它的偏好值，甚至我们可以吧均匀偏好值填进去当做 一个占位符。

## 3.4 总结

　　这一章我们看到了数据在Mahout的推荐器中如何去表达、浮现。这里包含了Preference对象，酷似凑集实现的特别数组PreferenceArray以及FastByIDMap。这些特定定制的对象很大程度上削减了内存的消费。

　　再看看DataModel，它是推荐器输入数据的一种抽象封装。在FileDataModel读取完本地文件之后，GenericDataModel在内存中存储这些数据。JDBCDataModel用来接见数据库的数据，并且我们还在MySQL上特地进行了测试。

　　最后我们评论辩论了在无偏好值的景象下，这些模型该如何的变更。一般上述这些对象都邑用到，当然存储上会稍微少一些。我们列举了一些标 准对象与该类数据不匹配的例子，如PearsonCorrelationSimilarity。最后经由过程检查题目的原因地点而解决了题目，目标是为了 构建一个基于布尔偏好数据的推荐器。

# 6 分布式推荐计算

本章概述：

分析维基百科上的一个大数据集

利用Hadoop和分布式计算产生推荐结果

伪分布式上存在的非分布式推荐

本书着眼于持续增长的数据集，从10条到100,000再到1千万再到1.7千万。不过这依然是中等大小的推荐系统所处理的 数据。本章依然放手一搏，处理了来自维基百科语料库中的1.3亿条数据，这些数据主要是以文章对文章的连接形式存在的。在这些数据集中，文章既充当了用 户，也充当了项目。这也显示了Mahout在特定情形下十分灵活的应用。

为满足演示目的，1.3亿的数据还是可管理的。但是对于单机处理推荐过程还是有一些困难的。这就需要使用一种新的推荐算法，通过Mahout基于的Mapreduce范式与Hadoop来实现分布式计算。

我们首先要检测一下维基百科的数据来认识一下它对于分布式推荐计算意味着什么。你将会学习到如何在分布式的环境下设计一个推 荐器，因为这与非分布式的推荐计算有很大的不同。你也将看到Mahout如何把之前的算法设计翻译成基于MapReduce和Hadoop的实现。最后， 你将运行一个完整的基于Hadoop的推荐Job程序。

## 6.1 分析维基百科数据集

我们从分析维基的数据开始，但是很快我们会发现由于问题规模而引起的问题。我们将很快退后几步去深入了解下分布式计算是如何执行的。

维基是一个由用户编写和维护的在线百科全书。据报道，在2010年它已经包含320万篇英文文章了。维基文章提取的开源项目(<http://download.freebase.com/wex/>)估计所有维基文章大概有42GB。基于互联网，维基文章可以相互链接。这些链接代表了某种兴趣。可以做个比方，一篇文章像是用户一样“喜欢”另一篇文章作为其偏好的项目。

幸运的是我们无需下载上述项目所提取的文章也不需要去分析出它们之间的链接。研究人员Henry Haselgrove已经在[http://users.on.net/~henry/home/wikipedia.htm](http://users.on.net/%7Ehenry/home/wikipedia.htm)上发表了他所完成的提取信息。另外，还有一些深入的提取，例如文章的讨论页、图片等等。这些数据采用数字ID而不是文章的标题，这样非常有好处，因为Mahout本身就是使用数字ID的。

在往下继续之前，先下载并解压links-simple-sorted.zip，其中包含5,706,070篇文档对 3,773,865篇不同文档的130,160,392个链接。主要说明的是，这些数据中没有偏好值或者排名等级，这里只有文章与文章之间的关联关系。也 就是说，这属于一种“布尔偏好”。文章A链接了文章B，不能代表B对A也有链接。在这里，用户和项目是一样多的，所以不论是基于用户的算法还是基于项目的 算法无好坏之分。如果使用一个涉及相似度度量的算法并且不依赖于偏好值，那么LogLikelihoodSimilarity会比较合适。

直觉上这些数据具有什么意义？并且我们期望获得什么样的推荐结果？一条从A到B的链接表示了B向A提供了一些相关参照信息。 一个基于这些信息的推荐信息将会一些和A指向相同或相似的文章。这些推荐出来的文章被解释成它们没有被A指向却应该被A指向。推荐系统将揭示一些对A感兴 趣或者会偶然关联上的一些文章。

### 6.1.2 数据规模的考虑

在处理这样大的数据时，一个非分布式的推荐引擎可能就搞不定了。单单这些数据就要消耗JVM堆空间2GB大小，再加上 Mahout程序所需空间，可能需要2.5GB。这已经超过了某些32位的机器的堆空间大小。也就是说，你需要一台64位的机器，今儿不换明儿也得换。再 由于推荐算法固有的时间复杂度，推荐系统需要超过一秒的时间去完成推荐任务，这对于现代Web实时应用来说已经是非常慢了。

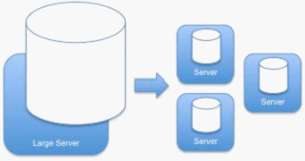
如果我们有足够多的硬盘，那么我们可以搭建这样一个平台。但是总有一天，随着数据的日益增多，到了十亿或者更多，所需的堆空 间也达到了上限32GB呢？对了，也许有人会开始琢磨如何对数据降噪去减少数据的量，提高数据的精度。但是这并不能从根本上回避数据增长所带来的问题。

如果你的系统在处理数据规模上有限制的话，那么只能说你Out了。计算资源可以无限的增加；那么我们要解决的问题是如果把这 些计算资源整合在一起去协同的完成任务。你的收益和制作一个超级计算机（计算能力的提升）所需的成本是不成比例的，所以我们应该想办法利用多个普通机器的 计算资源。对于那些单点无任何联系的机器，堆在一起只能是浪费电源，最有效利用他们的方式就是联合它们去为推荐系统效力。

这些维基数据的大小代表了一个基于Mahout的实时推荐系统所处理数据的上限。——这其实已经超过现代的标准了。对于这样的数据规模，我们需要一个新的方法。

### 6.1.2 分布式计算的优缺点评估

根据上面说明的原因，我们不得以才使用多个普通机器与代替一个高性能机器。一个公司或学术组织可能拥有很多未充分利用的普通 机器，这样这些机器额外的空闲资源就可以用来做推荐任务。另外有一些计算资源也可以通过云计算的提供商哪里得到，比如亚马逊EC2服务 （http://aws.amazon.com）。



**图 6.1 分布式可以将一个数据规模很大的问题化简成很多小规模数据的问题，从而分配给较小的服务器上去计算。**

分布式处理一个推荐问题从根本上改变了推荐引擎。到目前为止我们所见到的所有推荐算法都是一个具有一个偏好返回值的函数。为 了给一篇文章推荐一个新的连接，我们需要去遍历所有的文章-文章的连接；因为推荐过程需要所有的这些数据。然而，如此大量的数据，我们要遍历全部或者大部 分不可能一次性完成。我们之前所介绍的方法已经变得不再适用了，至少对于目前的问题形式。接下来分布式推荐引擎将为你我点亮一展明灯。

需要指出的是，分布式处理一个问题不能提搞计算的效率，相反，它相对于单机处理要消耗更多的资源。比如，在很多机器之前传输 数据需要消耗网络带宽。这种计算往往要被结构化，包括中间结果的排序，它会占用一些时间去进行序列化、存储以及反序列化等过程。而且这些是非常占用内存和 消耗电能的。

需要注意的是，这样的计算需要在线下执行，因为即使计算很少的数据也是秒级别的，而不是毫秒级别的。这样它就不能对用户的请求做出实时响应。一般地，这样的计算和存储会定期的进行，然后在运行时返回给用户。

然而，这种做法可以为推荐系统在处理大规模数据时提供一种途径，这是单机处理方案无法具备的。因为分布式计算可以从很多机器 上利用计算资源，把那些未使用、零散的计算资源整合在一起，从而在计算能力上超越专用计算机。最后，分布式计算可以使用更少的时间完成任务，虽然它需要更 多对原料处理的过程和时间。一般而言，对于同一个问题，分布式要比单机的计算多消耗一倍的CPU资源。如果我们用10CPU，那么速度将会是单机处理的5 倍，而且对于机器的利用率也大大提高了。

## 6.2 分布式处理一个基于项目的推荐算法

对于如此规模的数据，我们希望使用分布式处理的方法。首先，我们会对基于项目的算法的分布式变异版本进行描述。它在某种程度上和以前非分布式的版本很相似。当然它看起来是完全不同的，因为以前的算法未曾被翻译到分布式的世界。接下来我们要用Hadoop来跑一跑。

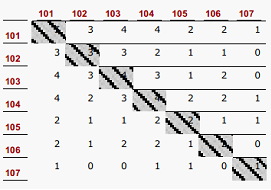
### 6.2.1 构造一个协同矩阵

这个算法由一些简单的矩阵操作都成，它易于解释并且容易实现。如果你上次接触矩阵是好几年前了，不要担心，最棘手的操作只有矩阵乘法（如果您连矩阵乘法都不知道，拜托，那还搞啥数据挖掘...）。这里保证没有什么行列式、行分解、特征值等等。

回忆一下，基于项目的推荐，我们依赖于ItemSimilarity的实现。它可以提供两个项目间的相似程度。试想一下，我 们会把计算完的相似度存放在一个很大的方阵里面，行和列和项目一一对应。每一行（或者每一列）代表了一个特定的项目与其他项目的相似度的一个向量。事实上 它是个对称方阵。因为X与Y的相似度就是Y与X的相似度。

对与这个算法，我们需要一个叫做“协同矩阵（co-occurrence matrix）”的东西。我们用两个不同项目同时 被喜欢的次数（协同因子）来代替之前相似度，这样可以构造出一个协同矩阵。比如，9个用户同时对X，Y感兴趣，那么他们的协同因子就是9，如果X，Y没有 同时被任何用户喜欢那么协同因子就是0，对于项目对自己的协同因子我们不考虑，或者说我们不去计算它。

协同因子很像相似度；两个项目同时出现可以描述这两个项目的相似程度。所以，正如我们之前所看到的，协同矩阵扮演了ItemSimilarity所需要的数据的角色。



**表 6.1 样本集中的协同矩阵。第一行和第一列分别是项目序号。**

生成矩阵的过程就是简单的计数，这个矩阵和偏好值无关。一会我们会对这些值进行计算。表6.1 表示了我们通篇用到的一个小 样例数据集所产生的协同矩阵。像上面所说，它是个对称方阵。有7个项目，矩阵有7行7列，对角线上的值将不会被使用，为了矩阵显示完整，我们暂且把数据写 在上面。

### *****6.2.2 计算用户向量*****

下一步就是我们把先前的推荐方法转化为一个基于矩阵的分布式模型，我们需要把用户对项目的偏好看作是一个向量。这一步你已经做到了。当我们使用欧几里德距离来度量用户相似度时，只需把用户看成是空间里的点，而相似度就是点之间的距离。

同样的，在模型中，用户的偏好就是一个n维的向量，每个维度代表一个项目。偏好值就是向量的每个分量。0代表用户对该项目无偏好。这个向量是稀疏的，很多分量都是0，因为实际中的用户仅对很少的项目有偏好。

例如，在上面的小样本数据集中，用户3的向量就是[2.0, 0.0, 0.0, 4.0, 4.5, 0.0, 5.0]。为了完成推荐，每个用户都需要这样一个向量。

### 6.2.3 产生推荐结果

为了给用户3产生推荐结果，只需要用矩阵去乘以他的用户向量，如6.2表中所示。

花一些时间回忆一下矩阵乘法把，如果需要来这里看看：（<http://en.wikipedia.org/wiki/Matrix_multiplication> ）。协同矩阵和用户向量的乘积是一个向量，维度个数等于项目的个数。这个向量就是我们要的推荐结果。分量的值越大说明该项目的推荐排名越靠前。

***表6.2 协同矩阵与用户3向量（U3）的乘积，最后用R来表示结果。***

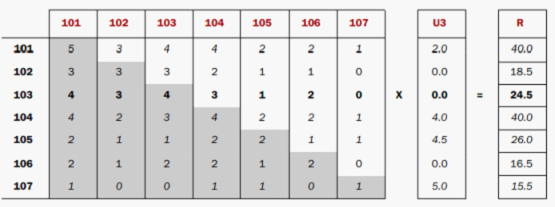


表6.2 展示了协同矩阵与U3的乘法运算过程，以及得到最终的推荐结果R。对于用斜体表示分数的项目101、104、 105还有107，我们完全忽略他们的结果，因为我们事先知道用户3对这些项目产生了偏好。剩下的结果中，103的分数是最高的，用黑体表示，那么103 就应该排在推荐结果中的首位。其他按照分数往下排即可。

### 6.2.4 分析结果

让我们停下来理解一下上面产生的结果，为何R中值较高的项目是较好的推荐呢？R中的没个分量是估计出来的偏好值，但是为何这个值就可以来表征用户对该项目的偏好呢？

回顾一下上面的计算过程，比如，R的第三个分量来自于矩阵的第三行与用户向量的点积。拿上面的数据就是：  
4(2.0) + 3(0.0) + 4(0.0) + 3(4.0) + 1(4.5) + 2(0.0) + 0(5.0) = 24.5

矩阵的第三行包含了项目103与其他项目的协同因子。直觉告诉我们，如果103和该用户所偏好的其他项目比较相似，那么用户 很可能会对103也有偏好。这个公式就是项目的协同因子与用户偏好乘积之和。如果103和该用户所偏好的很多项目同时出现时，那么103的分数就会越高。 相应的项目偏好越大也会对分数有所贡献。这就是为何R可以作为预测结果。

需要说明的是，R不能代表用户对该物品的偏好值，因为他们对于偏好值来说太大了。我们可以通过归一化来将这些值变位符合要求 的偏好值。如果你愿意的话，可以自己对结果处理一下。但是对于我们的目标来说，归一化是没必要的，因为我们关心的只是这些项目的得分排序，而不是具体的 值。

### 6.2.5 向分布式计算迈进

这是个很有意思的事情，但是用户大规模处理的算法究竟是什么样子的呢？

算法的每个单元在任何时候都仅仅包含了数据的一个子集。例如，在创建用户向量时，每个计算单元都只为一个特定的用户去计算。 计算协同矩阵时，计算单元仅为一个项目去计算它的协同因子向量。产生推荐结果时，计算单元只是用矩阵的一个行或者一列去乘以用户向量。另外，一些计算单元 可以立足于收集这些相关数据。比如，用户向量的创建，它所需的每个偏好值可以各自单独去计算，然后收集。

MapReduce范式专门为此而生。

# mahout实战 第7章 聚类介绍

本章包括

**1 实战操作了解聚类**

**2.了解相似性概念**

**3 使用mahout运行一个简单的聚类实例**

**4.用于聚类的各种不同的距离测算方法**

      作为人类，我们倾向于与志同道合的人合作—“鸟的羽毛聚集在一起。我们能够发现重复的模式通过联系在我们的记忆中的我们看到的、听到的、问道的、尝到的东 西。 例如，相比较盐 ，糖能够是我们更多地想起蜜。所以我们把糖和蜜的味道结合起来叫他们甜蜜。甚至我们不知道甜蜜的味道，但是知道他跟世界上所有的含糖的东西是相似的，是同 一类的。我们还知道它与盐是不同类的东西。无意中，我们不同的味道使用了聚类。把糖和盐做了聚类，每个组有数百个项目。

        在自然界中，我们观察不同类型的群体。认为猿与猴是同一种灵长类动物。所有的猴子 都有一些性状如短高度，长尾巴，和一个扁平的鼻子。相反，猿类有较大的尺寸，长长的手臂，和更大的头。猿看起来不同于猴子，但都喜欢香焦。所以我们可以认 为猿和猴子作为两个不同的组，或作为一个单一的爱香蕉的灵长类动物群。我们考虑一个集群完全取决于我们选择的项目之间的相似性测量的特点（在这种情况下， 灵长类动物）。

    在这一章中，从mahout学习的聚类的例子中，我们将会知道聚类是什么，如何有数据概念联系起来。让我们从基础开始吧！

**7.1 聚类的基础**

聚类的过程是怎么样的呢?假如你可以去有成千上万的书籍的图书馆。在图书馆内 ，图书是杂乱无章的。要找到想读的书必须横扫所有的书籍，一本一本的才能特定的书。这不仅是繁琐和缓慢，而且非常枯燥。

     按照标题的字母顺序排序会对读者通过标题寻找有很大的帮助，如果大多数人只是简单地浏览一下，只是找一个大概的主题的书籍呢？这样通过按照书籍的主题进 行分类要比按照标题的字母顺序更有用。但是如何进行分组呢？刚刚接手这份工作，你也不会知道所有的书籍是什么的？是冲浪的、浪漫的，或你没有遇到过的课 题。

     通过按主题把书籍分类，你不得不把所有书放在同一线上，一本一本的开始阅读。当你遇到一本书的内容是类似以前的一本书，你就返回去把它们放在在一起。当你完成，你从数千上万的书中获取到你要的主题的一堆书。

    干得好！这是你的第一个聚类的经验。如果一百个主题组太多了，你就得从头开始和重复刚才的过程获得书堆，直到你的主题，从另一个堆是完全不同的。

    聚类是所有关于组织项目从一个给定集合成一组类似的项目。在某些方面，这些集群可以被认为是作为项目相似集，但是与其他集群项目不同的。

**聚类集合包含三项**

**●**算法 ———–这是用来组书籍的方法

**●**相似性和差异性的概念——-一个在前面讨论的，依赖于这本书是否属于已经存在的堆，还是应          该另组新一堆的判断。

**●**终止条件———图书馆的例子中，当达到这些书不能堆积起来了，或这些堆已经相当不同的，这就是终止

          在这个简单的例子中，圈出来的显然是基于距离三个集群，代表了聚类的结果。圆是一个在聚类方面是一个很好的方法。由于群组通过中心点和半径来定义的，圆的中心被叫为群重心，或者群平均（平均值），群重心的坐标是类簇中的所有点的x,y轴的平均值

           项目的相似性测量

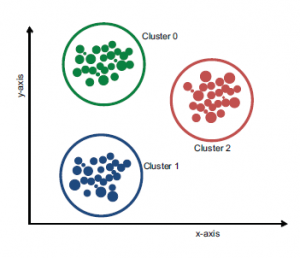
[](http://www.adintellig.com/wp-content/uploads/2013/06/clipboard.png)

                                        图7.1x-y平面图的点，圆圈代表了聚类，在平面团中的点归类了单个逻辑群组，聚类算法有益于识别群组

*在本章中，我们将聚类可视化为一个几何问题。聚类的核心是使用几何的技术表达不同距离的测算。我们找到一些重要的距离测算法和聚类关系。平面聚类点与文本聚类之间的具体相似性就可以抽象出来。*

*在后面的章节中，我们探讨了广泛用于聚类的方法数据，以及mahout 中使用*

*方法。图书馆的例子是将书分堆直到达到一定阈值的一种策 略。在这个例子中形成的簇的数目取决于数据；基于许多书和临界值，你可能发现了100后者20，甚至是1个类簇。一个更好的策略是建立目标簇，而不是一个 临界值，然后找到最好的群组与约束。接下来我们将详细的介绍目标簇和不同的变量*

***7.2项目的相似性测量***

*聚类的重要问题是找到一方法，通过任何两个数中的一个数来量化相似性。注意一下我们整片文章中使用的专业术语 ：项目和点，这两个是聚类数据的单位。*

*在X-Y平面的例子，相似性的度量（相似性度量）的分为两个点之间的欧几里德距离。图书馆的例子没有这种清晰的数学手段，而不是完全依赖的智慧馆员之间的相似度来判断书。这工作不在我们的案例，因为我们需要一个度量，可在计算机上实现.*

*一个可能的度量是基于两本书的标题共同含有的词的数量。基于哈利·波特：哲学家的石头和哈利·波特这两本书：阿兹卡班的囚徒中常见的三个词：哈利、波特、the。即时魔戒也：两塔是类似于哈利·波特系列，这一措施相似不会捕获这一切。你需要改变的相似性度量来对书籍本身的内容帐户。你可以将单词计数。*

*只可惜，说的容易做起来难。这些书不仅有几百个网页的文本，而且英语的特点也使的分类方法更加困难。在英语文本中有一些很频繁的次例如 a,an ,the 等等 ，它总是经常发生但又不能说明这是这两本书的相似点。*

*为了对抗这种影响，你可以在计算中使用的加权值，并且使用低权重表示这些词来减少对相似度值的影响。在出现很多次的词使用低权重值，出现少的用高权重。你可以衡量一个特定的书经常出现的，比如那些强烈建议内容的书籍–类似于魔术类的哈利波特。*

***你给书中的每一个单词一个权重，就能算出这本中的相似性–就是所有词的词频乘以每一个词的权重的和。如果这两本书的长度相同，那么这个就是一个比较适当的方法。***

*如果一本书有300页，另一本有1000页呢？当然书页大的书的词也多。*

# mahout实战 第9章 聚类算法

## 9.1 K-means聚类

K-means需要用户设定一个聚类个数（k）作为输入数据，有时k值可能非常大（10,000），这是Mahout闪光的（shines）地方，它确保聚类的可测量性。

为了用k-means达到高质量的聚类，需要估计一个k值。估计k值一种近似的方法是根据你需要的聚类个数。比如100万篇文章，如果平均500篇 分为一类，k值可以取2000（1000000/500）。这种估计聚类个数非常模糊，但k-means算法就是生成这种近似的聚类。

### 9.1.1 All you need to know about k-means

下面看一下k-means算法的细节，K-means算法是硬聚类算法，是典型的局域原型的目标函数聚类方法的代表，它是数据点到原型的某种距离作 为优化的目标函数，利用函数求极值的方法得到迭代运算的调整规则。算法采用误差平方和准则函数作为聚类准则函数。K-means算法是很典型的基于距离的[聚类](http://baike.baidu.com/view/31801.htm)算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。k个初始类聚类中心点的选取对聚类结果具有较大的。

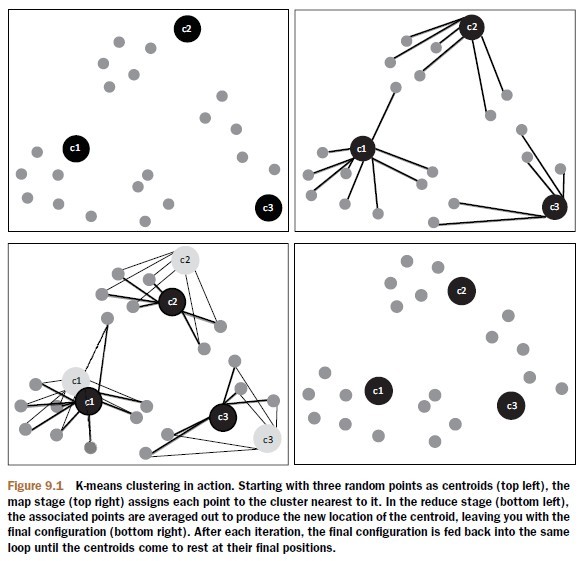
算法步骤：

（1）随机选取任意k个对象作为初始聚类中心，初始代表一个簇；

（2）计算点到质心的距离，并把它归到最近的质心的类；

（3）重新计算已经得到的各个类的质心；

（4）迭代2~3步直至新的质心与原质心相等或小于指定阈值，算法结束。



这种两步算法是最大期望算法（EM）的典型例子，

第一步是计算期望（E），利用对隐藏变量的现有估计值，计算其最大似然估计值；

第二步是最大化（M），最大化在 E 步上求得的最大似然值来计算参数的值。

M 步上找到的参数估计值被用于下一个 E 步计算中，这个过程不断交替进行。

### 9.1.2 Running k-means clustering

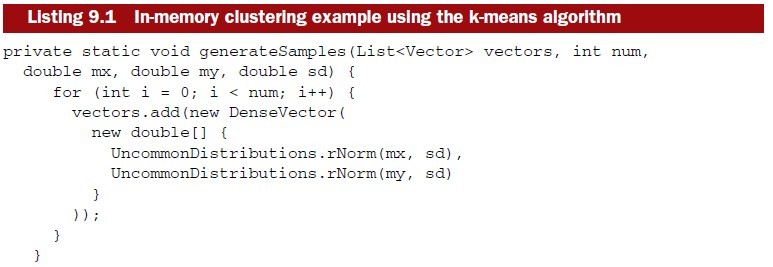
K-mean聚类用到KMeansClusterer或KMeansDriver类，前一个是在内存（in-memory）里对节点聚类，后者采用 MapReduce任务执行。这两种方法都可以就像一个普通的Java程序运行,并且可以从硬盘读取和写入数据。它们也可以在hadoop上执行聚类，通 过分布式文件系统读写数据。

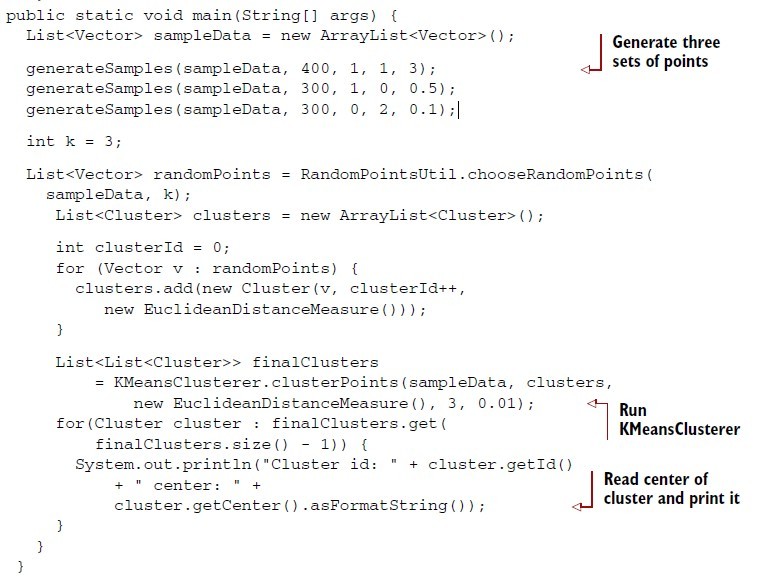
下面举例,使用一个随机点生成器函数来创建一些点。这些点生成矢量格式的点作为一个正态分布围绕着一个中心。使用Mahout的in-memory K-means 聚类方法对这些点聚类。

创建节点：generateSamples方法，取（1,1）为中心点，标准差为2,400个围绕着（1,1）的随机点，近似于正态分布。另外又取了2个数据集，中心分别为（1,0）和（0,2），标准差分别为0.5和0.1。

KMeansClusterer.clusterPoints()方法用到一下参数：

* List<Vector>作为输入；
* 距离算法DistanceMeasure采用EuclideanDistanceMeasure；
* 聚类的阈值Threshold为0.01；
* 聚类的个数为3；
* 聚类的中心点采用RandomPointsUtil，随机选取的节点。





Mahout-example里的DisplayKMeans类可以直观的看到该算法在二维平面的结果，9.2节将介绍运行一个Java Swing application的DisplayKMeans。

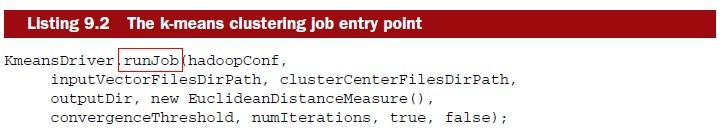
如果数据量很大时，应采取MapReduce运行方式，将聚类算法运行在多个机器上，每个mapper得到一个子集的点，每个子集运行一个mapper。这些mapper任务计算出最近的集群作为输入流。

**K-means聚类的MapReduce Job**

采用KMeansDriver类的run()方法，需要输入的参数有：

* Hadoop的配置conf；
* 输入Vectors的路径，SequenceFile格式；
* 初始化聚类中心的路径，SequenceFile格式；
* 输出结果的路径，SequenceFile格式；
* 求相似距离时采用的方法，这里采用EuclideanDistanceMeasure；
* 收敛的阈值，没有超过该阈值的点可移动时，停止迭代；
* 最大迭代次数，这是硬性限制，到达该最大迭代次数时，聚类停止；
* true-迭代结束后聚类；
* true-串行执行该算法，false-并行执行该算法；

public static void run(Configuration conf,Path input,Path clustersIn,Path output,  
DistanceMeasure measure,  
double convergenceDelta,  
int maxIterations,  
boolean runClustering,  
boolean runSequential)



采用SparseVectorsFromSequenceFile工具，将sequenceFile转换成Vector，因为K-means算法需要用户初始化k个质心。