Apache Mahout 简介

通过可伸缩、商业友好的机器学习来构建智能应用程序

Grant Ingersoll, 技术人员, Lucid Imagination

简介: 当研究院和企业能获取足够的专项研究预算之后，能从数据和用户输入中学习的智能应用程序将变得更加常见。人们对机器学习技巧（比如说集群、协作筛选和分类）的需求前所未有地增长，无论是查找一大群人的共性还是自动标记海量 Web 内容。Apache Mahout 项目旨在帮助开发人员更加方便快捷地创建智能应用程序。Mahout 的创始者 Grant Ingersoll 介绍了机器学习的基本概念，并演示了如何使用 Mahout 来实现文档集群、提出建议和组织内容。

发布日期: 2009 年 10 月 12 日
级别: 中级
其他语言版本: 英文
访问情况 10156 次浏览
建议:

内容

在信息时代，公司和个人的成功越来越依赖于迅速有效地将大量数据转化为可操作的信息。无论是每天处理数以千计的个人电子邮件消息，还是从海量博客文章中推测用户的意图，都需要使用一些工具来组织和增强数据。 这其中就蕴含着机器学习 领域以及本文章所介绍项目的前景：Apache Mahout（见 参考资料）。

机器学习是人工智能的一个分支，它涉及通过一些技术来允许计算机根据之前的经验改善其输出。此领域与数据挖掘密切相关，并且经常需要使用各种技巧，包括统计学、概率论和模式识别等。虽然机器学习并不是一个新兴领域，但它的发展速度是毋庸置疑的。许多大型公司，包括 IBM®、Google、Amazon、Yahoo! 和 Facebook，都在自己的应用程序中实现了机器学习算法。此外，还有许多公司在自己的应用程序中应用了机器学习，以便学习用户以及过去的经验，从而获得收益。

在简要概述机器学习的概念之后，我将介绍 Apache Mahout 项目的特性、历史和目标。然后，我将演示如何使用 Mahout 完成一些有趣的机器学习任务，这需要使用免费的 Wikipedia 数据集。

机器学习 101

机器学习可以应用于各种目的，从游戏、欺诈检测到股票市场分析。它用于构建类似于 Netflix 和 Amazon 所提供的系统，可根据用户的购买历史向他们推荐产品，或者用于构建可查找特定时间内的所有相似文章的系统。它还可以用于根据类别（体育、经济和战争等）对网页自动进行分类，或者用于标记垃圾电子邮件。本文无法完全列出机器学习的所有应用。如果您希望更加深入地探究该领域，我建议您参阅 参考资料。

可以采用一些机器学习方法来解决问题。我将重点讨论其中最常用的两个 — 监管 和无监管 学习 — 因为它们是 Mahout 支持的主要功能。

监管学习的任务是学习带标签的训练数据的功能，以便预测任何有效输入的值。监管学习的常见例子包括将电子邮件消息分类为垃圾邮件，根据类别标记网页，以及识别手写输入。创建监管学习程序需要使用许多算法，最常见的包括神经网络、Support Vector Machines (SVMs) 和 Naive Bayes 分类程序。

无监管学习的任务是发挥数据的意义，而不管数据的正确与否。它最常应用于将类似的输入集成到逻辑分组中。它还可以用于减少数据集中的维度数据，以便只专注于最有用的属性，或者用于探明趋势。无监管学习的常见方法包括 k-Means、分层集群和自组织地图。

在本文中，我将重点讨论 Mahout 当前已实现的三个具体的机器学习任务。它们正好也是实际应用程序中相当常见的三个领域：

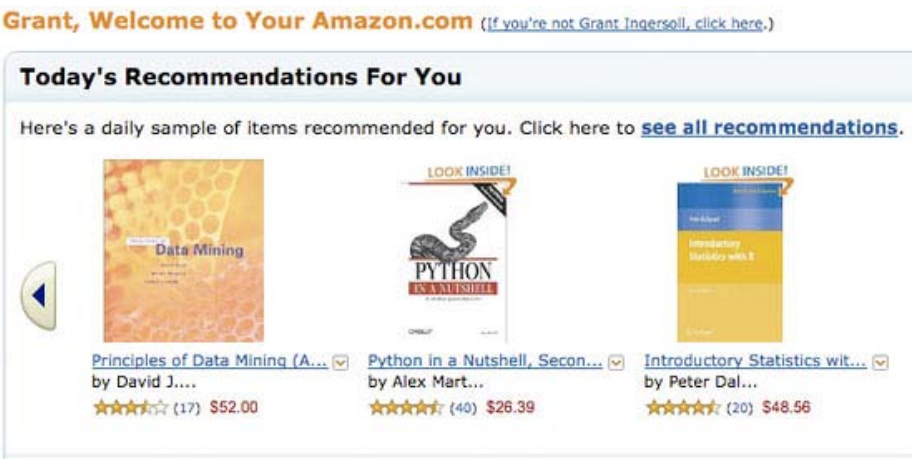
- 协作筛选
- 集群
- 分类

在研究它们在 Mahout 中的实现之前，我将从概念的层面上更加深入地讨论这些任务。

协作筛选

协作筛选 (CF) 是 Amazon 等公司极为推崇的一项技巧，它使用评分、单击和购买等用户信息为其他站点用户提供推荐产品。CF 通常用于推荐各种消费品，比如说书籍、音乐和电影。但是，它还在其他应用程序中得到了应用，主要用于帮助多个操作人员通过协作来缩小数据范围。您可能已经在 Amazon 体验了 CF 的应用，如 图 1 所示：

图 1. Amazon 上的协作筛选示例



CF 应用程序根据用户和项目历史向系统的当前用户提供推荐。生成推荐的 4 种典型方法如下：

- 基于用户：通过查找相似的用户来推荐项目。由于用户的动态特性，这通常难以定量。
- 基于项目：计算项目之间的相似度并做出推荐。项目通常不会过多更改，因此这通常可以离线完成。
- Slope-One：非常快速简单的基于项目的推荐方法，需要使用用户的评分信息（而不仅仅是布尔型的首选项）。
- 基于模型：通过开发一个用户及评分模型来提供推荐。

所有 CF 方法最终都需要计算用户及其评分项目之间的相似度。可以通过许多方法来计算相似度，并且大多数 CF 系统都允许您插入不同的指标，以便确定最佳结果。

集群

对于大型数据集来说，无论它们是文本还是数值，一般都可以将类似的项目自动组织，或集群，到一起。举例来说，对于全美国某天内的所有的报纸新闻，您可能希望将所有主题相同的文章自动归类到一起；然后，可以选择专注于特定的集群和主题，而不需要阅读大量无关内容。另一个例子是：某台机器上的传感器会持续输出内容，您可能希望对输出进行分类，以便于分辨正常和有问题的操作，因为普通操作和异常操作会归类到不同的集群中。

与 CF 类似，集群计算集合中各项目之间的相似度，但它的任务只是对相似的项目进行分组。在许多集群实现中，集合中的项目都是作为矢量表示在 ***n*** 维度空间中的。通过矢量，开发人员可以使用各种指标（比如说曼哈顿距离、欧氏距离或余弦相似性）来计算两个项目之间的距离。然后，通过将距离相近的项目归类到一起，可以计算出实际集群。

可以通过许多方法来计算集群，每种方法都有自己的利弊。一些方法从较小的集群逐渐构建成较大的集群，还有一些方法将单个大集群分解为越来越小的集群。在发展成平凡集群表示之前（所有项目都在一个集群中，或者所有项目都在各自的集群中），这两种方法都会通过特定的标准退出处理。流行的方法包括 k-Means 和分层集群。如下所示，Mahout 也随带了一些不同的集群方法。

分类

分类（通常也称为归类）的目标是标记不可见的文档，从而将它们归类不同的分组中。机器学习中的许多分类方法都需要计算各种统计数据（通过指定标签与文档的特性相关），从而创建一个模型以便以后用于分类不可见的文档。举例来说，一种简单的分类方法可以跟踪与标签相关的词，以及这些词在某个标签中的出现次数。然后，在对新文档进行分类时，系统将在模型中查找文档中的词并计算概率，然后输出最佳结果并通过一个分类来证明结果的正确性。

分类功能的特性可以包括词汇、词汇权重（比如说根据频率）和语音部件等。当然，这些特性确实有助于将文档关联到某个标签并将它整合到算法中。

机器学习这个领域相当广泛和活跃。理论再多终究需要实践。接下来，我将继续讨论 Mahout 及其用法。

回页首

Mahout 简介

Apache Mahout 是 Apache Software Foundation (ASF) 开发的一个全新的开源项目，其主要目标是创建一些可伸缩的机器学习算法，供开发人员在 Apache 在许可下免费使用。该项目已经发展到了它的最二个年头，目前只有一个公共发行版。Mahout 包含许多实现，包括集群、分类、CP 和进化程序。此外，通过使用 Apache Hadoop 库，Mahout 可以有效地扩展到云中（见 参考资料）。

Mahout 的历史

Mahout 项目是由 Apache Lucene（开源搜索）社区中对机器学习感兴趣的一些成员发起的，他们希望建立一个可靠、文档翔实、可伸缩的项目，在其中实现一些常见的用于集群和分类的机器学习算法。该社区最初基于 Ng et al. 的文章 “Map-Reduce for Machine Learning on Multicore”（见 参考资料），但此后在发展中又并入了更多广泛的机器学习方法。Mahout 的目标还包括：

背景知识
mahout 的意思是大象的饲养者及驱赶者。Mahout 这个名称来源于该项目（有时）使用 Apache Hadoop — 其徽标上有一头黄色的大象 — 来实现可伸缩性和容错性。

- 建立一个用户和贡献者社区，使代码不必依赖于特定贡献者的参与或任何特定公司和大学的资金。
- 专注于实际用例，这与高新技术研究及未经验证的技巧相反。
- 提供高质量文章和示例。

特性

虽然在开源领域中相对较为年轻，但 Mahout 已经提供了大量功能，特别是在集群和 CF 方面。Mahout 的主要特性包括：

- Taste CF。Taste 是 Sean Owen 在 SourceForge 上发起的一个针对 CF 的开源项目，并在 2008 年被赠予 Mahout。
- 一些支持 Map-Reduce 的集群实现包括 k-Means、模糊 k-Means、Canopy、Dirichlet 和 Mean-Shift。
- Distributed Naive Bayes 和 Complementary Naive Bayes 分类实现。
- 针对进化编程的分布式适用性功能。
- Matrix 和矢量库。
- 上述算法的示例。

Map-Reduce 简介

Map-Reduce 是 Google 开发的一种分布式编程 API，并在 Apache Hadoop 项目中得到了实现。与分布式文件系统相结合，它可以为程序员提供一个定义良好的用于描述计算任务的 API，从而帮助他们简化并行化问题的任务。（有关更多信息，请参见 参考资料）。

Mahout 入门

Mahout 的入门相对比较简单。首先，您需要安装以下软件：

- JDK 1.6 或更高版本
- Ant 1.7 或更高版本
- 如果要编译 Mahout 源代码，还需要安装 Maven 2.0.9 或 2.0.10

您还需要本文的示例代码（见 下载 部分），其中包括一个 Mahout 副本及其依赖关系。依照以下步骤安装示例代码：

1. 解压缩 `sample.zip`
2. `cd apache-mahout-examples`
3. `ant install`

步骤 3 将下载必要的 Wikipedia 文件将编译代码。所使用的 Wikipedia 文件大约为 2.5 GB，因此下载时间将由您的宽带决定。

回页首

建立一个推荐引擎

Mahout 目前提供了一些工具，可用于通过 Taste 库建立一个推荐引擎 — 针对 CF 的快速且灵活的引擎。Taste 支持基于用户和基于项目的推荐，并且提供了许多推荐选项，以及用于自定义的界面。Taste 包含 5 个主要组件，用于操作 用户、项目 和 首选项：

- `DataModel`：用于存储 用户、项目 和 首选项
- `UserSimilarity`：用于定义两个用户之间的相似度的界面
- `ItemSimilarity`：用于定义两个项目之间的相似度的界面
- `Recommender`：用于提供推荐的界面
- `UserNeighborhood`：用于计算相似用户邻近度的界面，其结果随时可由 `Recommender` 使用

借助这些组件以及它们的实现，开发人员可以构建复杂的推荐系统，提供基于实时或者离线的推荐。基于实时的推荐经常只能处理数千用户，而离线推荐具有更好的适用性。Taste 甚至提供了一些可利用 Hadoop 离线计算推荐的工具。在许多情况中，这种合适的方法可以帮助您满足包含大量用户、项目和首选项的大型系统的需求。

为了演示如何构建一个简单的推荐系统，我需要一些用户、项目和评分。为此，我们会使用 `cf.wikipedia.GenerateRatings` 中的代码（包含在示例代码的源代码中）为 Wikipedia 文档（Taste 称之为 项目）随机生成大量 用户 和 首选项，然后再手动补充一些关于特定话题（Abraham Lincoln）的评分，从而创建示例中的最终 recommendations.txt 文件。此方法的内涵是展示 CF 如何将对某特定话题感兴趣的人导向相关话题的其他文档。此示例的数据来源于 990（标记为从 0 到 989）个随机用户，他们随机为集合中的所有文章随机分配了一些评分，以及 10 个用户（标记为从 990 到 999），他们对集合中包含 *Abraham Lincoln* 关键字的 17 篇文章中的部分文章进行了评分。

注意虚构数据！

本文中的示例完全使用的是虚构数据。我自己完成了所有评分，模拟了 10 个对 Abraham Lincoln 感兴趣的实际用

首先，我将演示如何为在 recommendations.txt 文件中指定了分数的用户创建推荐。这是 Taste 最为常见的应用，因此首先需要载入包含推荐的数据，并将它存储在一个 `DataModel` 中。Taste 提供了一些不同的 `DataModel` 实现，用于操作文件和数据库。在本例中，为简便起见，

我选择使用 FileDataModel 类，它对各行的格式要求为：用户 ID、项目 ID、首选项 — 其中，用户 ID 和项目 ID 都是字符串，而首选项可以是双精度型。建立了模型之后，我需要通知 Taste 应该如何通过声明一个 UserSimilarity 实现来比较用户。根据所使用的 UserSimilarity 实现，您可能还需要通知 Taste 如何在未指定明确用户设置的情况下推断首选项。清单 1 实现了以上代码。（示例代码中的 cf.wikipedia.WikipediaTasteUserDemo 包含了完整的代码清单）。

户。虽然我相信数据内部的概念很有趣，但数据本身以及所使用的值并非如此。如果您希望获得实际数据，我建议您参阅 University of Minnesota 的 GroupLens 项目，以及 Taste 文档（见 参考资料）。我选择虚构数据的原因是希望在所有示例中都使用单一数据集。

清单 1. 创建模型和定义用户相似度

```
//create the data model
FileDataModel dataModel = new FileDataModel(new File(recsFile));
UserSimilarity userSimilarity = new PearsonCorrelationSimilarity(dataModel);
// Optional:
userSimilarity.setPreferenceInferrer(new AveragingPreferenceInferrer(dataModel));
```

在 清单 1 中，我使用了 PearsonCorrelationSimilarity，它用于度量两个变量之间的关系，但是也可以使用其他 UserSimilarity 度量。应该根据数据和测试类型来选择相似度度量。对于此数据，我发现这种组合最为合适，但仍然存在一些问题。有关如何选择相似度度量的更多信息，请访问 Mahout 网站（见 参考资料）。

为了完成此示例，我需要构建一个 UserNeighborhood 和一个 Recommender。UserNeighborhood 可以识别与相关用户类似的用户，并传递给 Recommender，后者将负责创建推荐项目排名表。清单 2 实现了以下想法：

清单 2. 生成推荐

```
//Get a neighborhood of users
UserNeighborhood neighborhood =
        new NearestNUserNeighborhood(neighborhoodSize, userSimilarity, dataModel);
//Create the recommender
Recommender recommender =
        new GenericUserBasedRecommender(dataModel, neighborhood, userSimilarity);
User user = dataModel.getUser(userId);
System.out.println("-----");
System.out.println("User: " + user);
//Print out the users own preferences first
TasteUtils.printPreferences(user, handler.map);
//Get the top 5 recommendations
List<RecommendedItem> recommendations =
        recommender.recommend(userId, 5);
TasteUtils.printRecs(recommendations, handler.map);
```

您可以在命令行中运行整个示例，方法是在包含示例的目录中执行 ant user-demo。运行此命令将打印输出虚构用户 995 的首选项和推荐，该用户只是 Lincoln 的爱好者之一。清单 3 显示了运行 ant user-demo 的输出：

清单 3. 用户推荐的输出

```
[echo] Getting similar items for user: 995 with a neighborhood of 5
    [java] 09/08/20 08:13:51 INFO file.FileDataModel: Creating FileDataModel
           for file src/main/resources/recommendations.txt
    [java] 09/08/20 08:13:51 INFO file.FileDataModel: Reading file info...
    [java] 09/08/20 08:13:51 INFO file.FileDataModel: Processed 100000 lines
    [java] 09/08/20 08:13:51 INFO file.FileDataModel: Read lines: 111901
    [java] Data Model: Users: 1000 Items: 2284
    [java] -----
    [java] User: 995
    [java] Title: August 21 Rating: 3.930000066757202
    [java] Title: April Rating: 2.203000068664551
    [java] Title: April 11 Rating: 4.230000019073486
    [java] Title: Battle of Gettysburg Rating: 5.0
    [java] Title: Abraham Lincoln Rating: 4.739999771118164
    [java] Title: History of The Church of Jesus Christ of Latter-day Saints
             Rating: 3.430000066757202
    [java] Title: Boston Corbett Rating: 2.009999990463257
    [java] Title: Atlanta, Georgia Rating: 4.429999828338623
    [java] Recommendations:
    [java] Doc Id: 50575 Title: April 10 Score: 4.98
    [java] Doc Id: 134101348 Title: April 26 Score: 4.860541
    [java] Doc Id: 133445748 Title: Folklore of the United States Score: 4.4308662
    [java] Doc Id: 1193764 Title: Brigham Young Score: 4.404066
    [java] Doc Id: 2417937 Title: Andrew Johnson Score: 4.24178
```

从清单 3 中可以看到，系统推荐了一些信心级别不同的文章。事实上，这些项目的分数都是由其他 Lincoln 爱好者指定的，而不是用

户 995 一人所为。如果您希望查看其他用户的结构，只需要在命令行中传递 `-Duser.id=USER-ID` 参数，其中 *USER-ID* 是 0 和 999 之间的编号。您还可以通过传递 `-Dneighbor.size=X` 来更改邻近空间，其中，*X* 是一个大于 0 的整型值。事实上，将邻近空间更改为 10 可以生成极为不同的结果，这是因为邻近范围内存在一个随机用户。要查看邻近用户以及共有的项目，可以向命令行添加 `-Dcommon=true`。

现在，如果您所输入的编号恰好不在用户范围内，则会注意到示例生成了一个 `NoSuchUserException`。确实，应用程序需要处理新用户进入系统的情况。举例来说，您可以只显示 10 篇最热门的文章，一组随机文章，或者一组 “不相关” 的文章 — 或者，与其这样，还不如不执行任何操作。

如前所述，基于用户的方法经常不具有可伸缩性。在本例中，使用基于项目的方法是更好的选择。幸运的是，Taste 可以非常轻松地实现基于项目的方法。处理项目相似度的基本代码并没有很大差异，如清单 4 所示：

清单 4. 项目相似度示例（摘录自 **cf.wikipedia.WikipediaTasteItemItemDemo**）

```
//create the data model
FileDataModel dataModel = new FileDataModel(new File(recsFile));
//Create an ItemSimilarity
ItemSimilarity itemSimilarity = new LogLikelihoodSimilarity(dataModel);
//Create an Item Based Recommender
ItemBasedRecommender recommender =
        new GenericItemBasedRecommender(dataModel, itemSimilarity);
//Get the recommendations
List<RecommendedItem> recommendations =
        recommender.recommend(userId, 5);
TasteUtils.printRecs(recommendations, handler.map);
```

与 清单 1 相同，我根据推荐文件创建了一个 `DataModel`，但这次并未实例化 `UserSimilarity` 实例，而是使用 `LogLikelihoodSimilarity` 创建了一个 `ItemSimilarity`，它可以帮助处理不常见的事件。然后，我将 `ItemSimilarity` 提供给一个 `ItemBasedRecommender`，最后请求推荐。完成了！您可以通过 `ant item-demo` 命令在示例中代码运行它。当然，在此基础上，您可以让系统支持离线执行这些计算，您还可以探索其他的 `ItemSimilarity` 度量。注意，由于本示例中的数据是随机的，所推荐的内容可能并不符合用户的期望。事实上，您应该确保在测试过程中计算结果，并尝试不同的相似度指标，因为许多常用指标在一些边界情况中会由于数据不足而无法提供合适的推荐。

我们再来看新用户的例子，当用户导航到某个项目之后，缺少用户首选项时的操作就比较容易实现了。对于这种情况，您可以利用项目计算并向 `ItemBasedRecommender` 请求与相当项目最相似的项目。清单 5 展示了相关代码：

清单 5. 相似项目演示（摘录自 **cf.wikipedia.WikipediaTasteItemRecDemo**）

```
//create the data model
FileDataModel dataModel = new FileDataModel(new File(recsFile));
//Create an ItemSimilarity
ItemSimilarity itemSimilarity = new LogLikelihoodSimilarity(dataModel);
//Create an Item Based Recommender
ItemBasedRecommender recommender =
        new GenericItemBasedRecommender(dataModel, itemSimilarity);
//Get the recommendations for the Item
List<RecommendedItem> simItems
        = recommender.mostSimilarItems(itemId, numRecs);
TasteUtils.printRecs(simItems, handler.map);
```

您可以通过在命令中执行 `ant sim-item-demo` 来运行 清单 5。它与 清单 4 之间的唯一差异就是，清单 5 并没有请求推荐，而是请求输出最相似的项目。

现在，您可以继续深入探索 Taste。要了解更多信息，请阅读 Taste 文档和 mahout-user@lucene.apache.org 邮件列表（见 参考资料）。接下来，我将讨论如何通过利用 Mahout 的集群功能来查找相似文章。

回页首

使用 Mahout 实现集群

Mahout 支持一些集群算法实现（都是使用 Map-Reduce 编写的），它们都有一组各自的目标和标准：

- Canopy：一种快速集群算法，通常用于为其他集群算法创建初始种子。
- k-Means（以及 模糊 k-Means）：根据项目与之前迭代的质心（或中心）之间的距离将项目添加到 k 集群中。
- Mean-Shift：无需任何关于集群数量的推理 知识的算法，它可以生成任意形状的集群。
- Dirichlet：借助基于多种概率模型的集群，它不需要提前执行特定的集群视图。

从实际的角度来说，名称和实现并不如它们生成的结果重要。了解了这一点之后，我将展示 k-Means 的运行原理，而其余内容将由您自己去研究。请记住，要有效运行每个算法，您需要满足它们各自的的需求。

简单来说（详细信息见下文），使用 Mahout 创建数据集群的步骤包括：

1. 准备输入。如果创建文本集群，您需要将文本转换成数值表示。
2. 使用 Mahout 中可用的 Hadoop 就绪的驱动程序运行所选集群算法。
3. 计算结果。
4. 如果有必要，执行迭代。

首先，集群算法要求数据必需采用适合处理的格式。在机器学习中，数据通常被表示为矢量，有时也称作特征矢量。在集群中，矢量是表示数据的一组权重值。我将使用通过 Wikipedia 文档生成的矢量来演示集群，但是也可以从其他地方获取矢量，比如说传感器数据或用户资料。Mahout 随带了两个 `Vector` 表示：`DenseVector` 和 `SparseVector`。根据所使用的数据，您需要选择合适的实现，以便实现良好的性能。通常而言，基于文本的问题是很少的，因此应该使用 `SparseVector` 来处理文本。另一方面，如果大多数矢量的大多数值都是非零的，则比较适合使用 `DenseVector`。如果您对此不确定，可以尝试这两种实现来处理数据的一个子集，然后确定哪种实现的运行速度更快。

通过 Wikipedia 内容生成矢量的方法如下（我已经完成了此工作）：

1. 将内容索引编入 Lucene，确保存储相关字段（用于生成矢量的字段）的 term 矢量。我不会讨论这方面的详细信息 — 不在本文讨论范围之内 — 但我会提供一些简要提示以及 Lucene 上的一些参考资料。Lucene 提供了一个称为 `EnWikiDocMaker` 的类（包含在 Lucene 的 `contrib/benchmark` 包中），该类可以读取 Wikipedia 文件块中的内容并生成编入 Lucene 索引的文档。

2. 使用 `org.apache.mahout.utils.vectors.lucene.Driver` 类（位于 Mahout 的 `utils` 模块中）通过 Lucene 索引创建矢量。此驱动程序提供了大量用于创建矢量的选项。Mahout wiki 页面 “Creating Vectors from Text” 提供了更多信息（见 参考资料）。

运行这两个步骤的结果是生成一个文件，该文件类似于与您从 Getting started with Mahout 入门 部分下载的 n2.tar.gz 文件。需要说明一下，n2.tar.gz 文件中的矢量是通过由 `ant install` 方法之前下载的 Wikipedia “块” 文件中的所有文件的索引创建的。矢量将被格式化为 Euclidean 格式（或者 L^2 格式；请参见 参考资料）。在使用 Mahout 时，您可能希望尝试采用不同的方法来创建矢量，以确定哪种方法的效果最好。

创建了一组矢量之后，接下来需要运行 k-Means 集群算法。Mahout 为所有集群算法都提供了驱动程序，包括 k-Means 算法，更合适的名称应该是 `KMeansDriver`。可以直接将驱动程序作为单独的程序使用，而不需要 Hadoop 的支持，比如说您可以直接运行 `ant k-means`。有关 `KMeansDriver` 可接受的参数的更多信息，请查看 build.xml 中的 Ant k-means 目标。完成此操作之后，您可以使用 `ant dump` 命令打印输出结果。

评估结果
可以采用多种方法来评估集群结果。许多人最开始都是使用手动检查与随机测试相结合的方法。但是，要实现令人满足的结果，通常都需要使用一些更加高级的计算技巧，比如说使用一些准则开发一个黄金标准。有关评估结果的更多信息，请参见 参考资料。在本例中，我使用手动检查来判断结果集群是否有意义。如果要投入生产，则应该使用更加严格的流程。

成功在独立模式中运行驱动程序之后，您可以继续使用 Hadoop 的分布式模式。为此，您需要 Mahout Job JAR，它位于示例代码的 hadoop 目录中。Job JAR 包可以将所有代码和依赖关系打包到一个 JAR 文件中，以便于加载到 Hadoop 中。您还需要下载 Hadoop 0.20，并依照 Hadoop 教程的指令，首先在准分布式模式（也就是一个集群）中运行，然后再采用完全分布式模式。有关更多信息，请参见 Hadoop 网站及资源，以及 IBM 云计算资源（参见 参考资料）。

回页首

使用 Mahout 实现内容分类

Mahout 目前支持两种根据贝氏统计来实现内容分类的方法。第一种方法是使用简单的支持 Map-Reduce 的 Naive Bayes 分类器。Naive Bayes 分类器为速度快和准确性高而著称，但其关于数据的简单（通常也是不正确的）假设是完全独立的。当各类的训练示例的大小不平衡，或者数据的独立性不符合要求时，Naive Bayes 分类器会出现故障。第二种方法是 Complementary Naive Bayes，它会尝试纠正 Naive Bayes 方法中的一些问题，同时仍然能够维持简单性和速度。但在本文中，我只会演示 Naive Bayes 方法，因为这能让您看到总体问题和 Mahout 中的输入。

简单来讲，Naive Bayes 分类器包括两个流程：跟踪特定文档及类别相关的特征（词汇），然后使用此信息预测新的、未见过的内容的类别。第一个步骤称作训练（*training*），它将通过查看已分类内容的示例来创建一个模型，然后跟踪与特定内容相关的各个词汇的概率。第二个步骤称作分类，它将使用在训练阶段中创建的模型以及新文档的内容，并结合 Bayes Theorem 来预测传入文档的类别。因此，要运行 Mahout 的分类器，您首先需要训练模式，然后再使用该模式对新内容进行分类。下一节将演示如何使用 Wikipedia 数据集来实现此目的。

运行 Naive Bayes 分类器

在运行训练程序和分类器之前，您需要准备一些用于训练和测试的文档。您可以通过运行 `ant prepare-docs` 来准备一些 Wikipedia 文件（通过 `install` 目标下载的文件）。这将使用 Mahout 示例中的 `WikipediaDatasetCreatorDriver` 类来分开 Wikipedia 输入文件。分开文档的标准是它们的类似是否与某个感兴趣的类别相匹配。感兴趣的类别可以是任何有效的 Wikipedia 类别（或者甚至某个 Wikipedia 类别的任何子字符串）。举例来说，在本例中，我使用了两个类别：科学（science）和历史（history）。因此，包含单词 *science* 或 *history* 的所有 Wikipedia 类别都将被添加到该类别中（不需要准确匹配）。此外，系统为每个文档添加了标记并删除了标点、Wikipedia 标记以及此任务不需要的其他特征。最终结果将存储在一个特定的文件中（该文件名包含类别名），并采用每行一个文档的格式，这是 Mahout 所需的输入格式。同样，运行 `ant prepare-test-docs` 代码可以完成相同的文档测试工作。需要确保测试和训练文件没有重合，否则会造成结果不准确。从理论上说，使用训练文档进行测试应该能实现最

的结果，但实际情况可能并非如此。

设置好训练和测试集之后，接下来需要通过 `ant train` 目标来运行 `TrainClassifier` 类。这应该会通过 Mahout 和 Hadoop 生成大量日志。完成后，`ant test` 将尝试使用在训练时建立的模型对示例测试文档进行分类。这种测试在 Mahout 中输出的数据结构是混合矩阵。混合矩阵可以描述各类别有多少正确分类的结果和错误分类的结果。

总的来说，生成分类结果的步骤如下：

1. `ant prepare-docs`
2. `ant prepare-test-docs`
3. `ant train`
4. `ant test`

运行所有这些命令（Ant 目标 `classifier-example` 将在一次调用中捕获所有它们），这将生成如清单 6 所示的汇总和混合矩阵：

清单 6. 运行 `Bayes` 分类器对历史和科学主题进行分类的结果

```
[java] 09/07/22 18:10:45 INFO bayes.TestClassifier: history
                                  95.458984375    3910/4096.0
[java] 09/07/22 18:10:46 INFO bayes.TestClassifier: science
                                  15.554072096128172      233/1498.0
[java] 09/07/22 18:10:46 INFO bayes.TestClassifier: =================
[java] Summary
[java] -------------------------------------------------------
[java] Correctly Classified Instances          :       4143
                                                      74.0615%
[java] Incorrectly Classified Instances        :       1451
                                                      25.9385%
[java] Total Classified Instances              :       5594
[java]
[java] =======================================================
[java] Confusion Matrix
[java] -------------------------------------------------------
[java] a           b       <--Classified as
[java] 3910        186      |  4096        a     = history
[java] 1265        233      |  1498        b     = science
[java] Default Category: unknown: 2
```

中间过程的结果存储在 base 目录下的 wikipedia 目录中。

获取了结果之后，显然还有一个问题：“我应该如何做？”汇总结果表明，正确率和错误率大概分别为 75% 和 25%。这种结果看上去非常合理，特别是它比随机猜测要好很多。但在仔细分析之后，我发现对历史信息的预测（正确率大约为 95%）相当出色，而对科学信息的预测则相当糟糕（大约 15%）。为了查找其原因，我查看了训练的输入文件，并发现与历史相关的示例要比科学多很多（文件大小几乎差了一倍），这可能是一个潜在的问题。

对于测试，您可以向 `ant test` 添加 `-Dverbose=true` 选项，这会显示关于各测试输入的信息，以及它的标签是否正确。仔细研究此输出，您可以查找文档并分析它分类错误的原因。我还可以尝试不同的输入参数，或者使用更加科学数据来重新训练模型，以确定是否能够改善此结果。

在训练模型时考虑使用特征选择也是很重要的。对于这些示例，我使用 Apache Lucene 中的 `WikipediaTokenizer` 来标记初始文档，但是我没有尽力删除可能标记错误的常用术语或垃圾术语。如果要将此分类器投入生产，那么我会更加深入地研究输入和其他设置，以弥补性能的每个方面。

为了确定 Science 结果是否是个意外，我尝试了一组不同的类别：共和（Republican）与民主（Democrat）。在本例中，我希望预测新文档是否与 Republicans 或者 Democrats 相关。为了帮助您独立实现此功能，我在 src/test/resources 中创建了 repubs-dems.txt 文件。然后，通过以下操作完成分类步骤：

```
ant classifier-example -Dcategories.file=./src/test/resources/repubs-dems.txt -Dcat.dir=rd
```

两个 `-D` 值仅仅指向类别文件以及 wikipedia 目录中存储中间结果的目录。此结果概要和混合矩阵如清单 7 所示：

清单 7. 运行 `Bayes` 分别器查找 `Republicans` 和 `Democrats` 的结果

```
[java] 09/07/23 17:06:38 INFO bayes.TestClassifier: --------------
[java] 09/07/23 17:06:38 INFO bayes.TestClassifier: Testing:
                              wikipedia/rd/prepared-test/democrats.txt
[java] 09/07/23 17:06:38 INFO bayes.TestClassifier: democrats      70.0
                                                       21/30.0
[java] 09/07/23 17:06:38 INFO bayes.TestClassifier: --------------
[java] 09/07/23 17:06:38 INFO bayes.TestClassifier: Testing:
                             wikipedia/rd/prepared-test/republicans.txt
[java] 09/07/23 17:06:38 INFO bayes.TestClassifier: republicans    81.3953488372093
```

```
                                                                35/43.0
[java] 09/07/23 17:06:38 INFO bayes.TestClassifier:
[java] Summary
[java] -------------------------------------------------------
[java] Correctly Classified Instances          :         56           76.7123%
[java] Incorrectly Classified Instances        :         17           23.2877%
[java] Total Classified Instances              :         73
[java]
[java] =======================================================
[java] Confusion Matrix
[java] -------------------------------------------------------
[java] a           b       <--Classified as
[java] 21          9        |  30          a     = democrats
[java] 8           35       |  43          b     = republicans
[java] Default Category: unknown: 2
```

虽然最终结果在正确性方面差不多是相同的，但您可以看到我在这两个类别中进行选择时采取更好的方式。查看包含输入文档的 wikipedia/rd/prepared 目录，我们发现两个训练文件在训练示例方面更加平衡了。 此外，与 “历史/科学” 结果相比，得到了示例也少了很多，因为每个文件都比历史或科学训练集小很多。总的来说，结果至少表明平衡性得到了显著改善。更大的训练集可能会抵消 Republicans 和 Democrats 之间的差异，即便不行也可以暗示某个分组坚持其在 Wikipedia 上的消息是较好的选择 — 但是，我选择将这留给政治学者来决定。

现在，我已经展示了如何在独立模式中执行分类，接下来需要将代码添加到云中，并在 Hadoop 集群上运行。与集群代码相同，您需要 Mahout Job JAR。除此之外，我之前提到的所有算法都是支持 Map-Reduce 的，并且能够在 Hadoop 教程所述的 Job 提交流程中运行。

回页首

结束语

Apache Mahout 在一年多的时间中走过了漫长的道路，为集群、分类和 CF 提供了许多重要的功能，但它还存在很大的发展空间。日益强大起来的还有 Map-Reduce 的随机决策实现，它提供了分类、关联规则、用于识别文档主题的 Latent Dirichlet Allocation 以及许多使用 HBase 和其他辅助存储选项的类别选项。除了这些新的实现之外，还可以找到许多演示、文档和 bug 修复包。

最后，就像实际驱象者（mahout）利用大象的力量一样，Apache Mahout 也可以帮助您利用小黄象 Apache Hadoop 的强大功能。下次在需要集群、分类或推荐内容时，特别是规模很大时，一定要考虑使用 Apache Mahout。

致谢

特别感谢 Ted Dunning 和 Sean Owen 对本文的审阅和建议。

参考资料

学习

- 机器学习
  - 机器学习：Wikipedia 页面提供了一些有用的入门信息和优秀的参考资料，可帮助您了解关于机器学习（包含监管学习等方法）的更多信息。
  - *Programming Collective Intelligence*（Toby Segaran，O'Reilly，2007 年）：本书可以帮助您迅速掌握许多机器任务。
  - Artificial Intelligence | Machine Learning：使用斯坦福大学教授 Andrew Ng 开发的这个类。
  - Evaluation of clustering：了解关于评估集群的更多信息。另请参阅 Mahout 邮件列表上的 讨论。
  - Bayes Theorem：了解 Bayes Theorem 的运行原理。
  - L^p 空间：理解 L^p 格式。
- Apache Mahout 和 Apache Lucene
  - Mahout 项目主页：搜索关于 Mahout 的所有内容。
  - “Map-Reduce for Machine Learning on Multicore”：这篇文章将帮助您启动 Mahout。
  - “MapReduce: Simplified Data Processing on Large Clusters”（Google Research Publications）：阅读关于 Map-Reduce 初级文章。
  - Taste：阅读 Taste 文档。
  - Apache Lucene：了解关于 Lucene 的更多信息。

  - Apache Lucene on developerWorks：通过这些文章探索 Lucene 的世界。
  - Creating Vectors from Text：阅读 Mahout Wiki 中的这个条目，了解如何将数据转换为 Mahout 的 `Vector` 类。
  - Cluster Your Data：阅读此 Mahout Wiki 页面，了解关于如何实现数据集群的更多信息。

- Apache Hadoop：
  - Apache Hadoop：了解关于 Hadoop 的更多信息。
  - Hadoop 快速入门教程：了解如何运行 Hadoop Job。
  - HBase：理解 Hadoop 数据库。
- 浏览 技术书店，阅读有关这些主题和其他技术主题的图书。
- Cloud Computing：访问 developerWorks 云计算空间。
- developerWorks Java 技术专区：数百篇关于 Java 编程各个方面的文章。

获得产品和技术

- 下载 Hadoop 0.20.0。
- 下载 Wikipedia 的子集 。
- 下载 Wikipedia 的子集 作为矢量。
- 从 GroupLens 项目获取真实的电影评分数据。

讨论

- 加入 Mahout 社区：mahout-user@lucene.apache.org。
- 加入 My developerWorks 社区。

关于作者

Grant Ingersoll 是 Lucid Imagination 的创始人及技术人员之一。Grant 的编程兴趣包括信息检索、机器学习、文本分类和提取。Grant 是 Apache Mahout 机器学习项目的创始人之一，同时还是 Apache Lucene 和 Apache Solr 项目的热衷者和演讲者。他还与人共同编写了 *Taming Text*（Manning，即将出版）一书，该书概述了用于自然语言处理的开源工具。

建议

回页首

打印此页面　分享此页面　关注 developerWorks

技术主题
AIX and UNIX
Information Management
Lotus
Rational
WebSphere
Java technology
Linux
Open source
SOA and web services
Web development
查找软件
IBM 产品
评估方式（下载，在线试用，Beta 版，云）
行业
社区
群组
博客
Wiki
文件
使用条款与条件
关于 developerWorks
反馈意见
在线投稿
投稿指南
网站导航
请求转载内容
IBM
解决方案
软件
支持门户
产品文档
红皮书 (英语)