# 摘 要

[聚类分析](http://baike.baidu.com/view/903740.htm)是根据指定的规则，将给定的数据集按照相似性进行分类的一种统计分析方法。聚类分析在许多领域得到广泛应用，例如计算视觉、生物医学、信息检索、数据挖掘和模式识别等。

[粗糙集理论](http://baike.baidu.com/view/223951.htm)是一种处理不完整性和不确定性问题的[数学工具](http://baike.baidu.com/view/750697.htm)，它能有效分析不精确、不一致、不完整的各类信息，能从数据本身提供的信息中发现有效的、潜在的知识。

Spark是由加州大学伯克利分校AMP实验室开发的内存计算分布式框架，主要针对海量数据处理和机器学习。Spark采用Scala语言实现，提供集成的语言编程接口，使用户可以非常容易地编写并行任务。Spark基于内存的计算比Hadoop的MapReduce快100倍，Spark基于硬盘的运算也比MapReduce要快10倍。

本文研究了基于粗糙集理论的MMR和MTMDP层次聚类算法，这两种算法都应用于分类数据。MMR算法应用了粗糙度的概念来选择分裂属性，选择对象数目多的叶子结点继续进行分裂。MTMDP算法应用了概率粗糙集模型中粗糙隶属度的概念来选择分裂属性，选择聚合度小的叶子结点继续进行聚类。这两种算法主要的优点如下：（1）能够处理聚类过程中的不确定性问题；（2）仅仅只需要用户输入希望生成的簇数目就能得到良好的聚类结果，具有良好的鲁棒性；（3）能够适用于大数据的处理。为了测试MMR和MTMDP算法的性能，我们在四个真实的数据集上进行了实验，并和一些不稳定的聚类算法进行了比较。传统单机的聚类算法无论是从效率上，还是从计算复杂度上都已无法满足海量信息的处理需要，云计算技术的发展为聚类分析提供了新的研究方向。为了突破面对海量数据时的计算瓶颈，本文在Spark平台上实现了MMR和MTMDP算法的并行化。实验结果表明MMR和MTMDP并行算法均有较好的性能，并且都适合海量数据的处理。

关键词：聚类分析； 粗糙集理论； 分类数据； 云计算

# Abstract

Clustering analysis refers to dividing a given dataset into similar groups according to given rules, and the technique is widely applied in many domains, such as computer vision, biology, medicine, information retrieval, data mining, and pattern recognition.

Rough Set Theory is a mathematical tool to deal with incomplete and uncertain problems. It can effectively analyze the inaccurate, inconsistent and incomplete information. It can find valid and potentially useful knowledge in large amount of data.

Spark is a memory computing distributed framework developed by California Berkeley AMP Lab, which mainly aims at the mass data processing and machine learning. Spark provides a language-integrated programming interface in the Scala programming language, making it easy for users to write parallel jobs. Spark runs programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

This thesis studies two hierarchical clustering algorithms, MMR and MTMDP, for categorical data based on Rough Set Theory. The MMR algorithm employs the concept of roughness to search the partitioning attribute and choose the leaf node with more objects for further splitting. The MTMDP algorithm employs the concept of rough membership degree based on probabilistic rough set theory to search the partitioning attribute and determines the further clustering node with less cohesion degree. The main advantages of the two algorithms are as follows: (1) it is capable of handling the uncertainty in the clustering process; (2) it is a robust clustering algorithm because it enables the users to obtain stable results by only one input: the number of clusters; (3) it has the capability of handling large datasets. In order to evaluate performance of MMR and MTMDP, we carried out experiments on four real datasets. In addition, we compared the two algorithms with some unstable clustering algorithms. In terms of efficiency or computational complexity, the traditional clustering algorithms conducted on a single machine are unable to meet the needs of massive information processing. The development of cloud computing technology provides a new direction for clustering analysis. In order to break through the bottleneck of massive data computing, the parallelization of MMR and MTMDP is implemented on the Spark platform. Experimental results show that the MMR and MTMDP parallel algorithms achieve better performance, and they are suitable for handling large amounts of data.

**Keywords:** Cluster analysis; Rough Set Theory; Categorical data; Cloud Computing

目 录

[摘 要 III](#_Toc422787915)

[Abstract IV](#_Toc422787916)

[第1章 绪论 1](#_Toc422787917)

[1.1 本论文的背景和意义 1](#_Toc422787918)

[1.2 国内外研究现状 2](#_Toc422787919)

[1.2.1 聚类分析的研究现状 2](#_Toc422787920)

[1.2.2 粗糙集理论的研究现状 2](#_Toc422787921)

[1.2.3 云计算的研究现状 3](#_Toc422787922)

[1.3 本论文的主要内容 3](#_Toc422787923)

[1.4 本论文的结构安排 3](#_Toc422787924)

[1.5 本章小结 4](#_Toc422787925)

[第2章 相关知识概述 5](#_Toc422787926)

[2.1 数据挖掘 5](#_Toc422787927)

[2.1.1 数据挖掘的定义 5](#_Toc422787928)

[2.1.2 数据挖掘的基本过程 5](#_Toc422787929)

[2.2 聚类分析 5](#_Toc422787930)

[2.2.1 聚类分析的数据类型 5](#_Toc422787931)

[2.2.2 聚类分析中的相似度量方法 7](#_Toc422787932)

[2.2.3 聚类分析方法简介 8](#_Toc422787933)

[2.2.4 聚类分析的有效性 9](#_Toc422787934)

[2.3 粗糙集理论 10](#_Toc422787935)

[2.4 Spark平台介绍 11](#_Toc422787936)

[2.4.1 Spark的体系结构 12](#_Toc422787937)

[2.4.2 Spark的核心概念 12](#_Toc422787938)

[2.4.3 Spark编程接口 14](#_Toc422787939)

[2.5 本章小结 15](#_Toc422787940)

[第3章 算法设计与分析 16](#_Toc422787941)

[3.1 MMR算法介绍与分析 16](#_Toc422787942)

[3.1.1 MMR算法介绍 16](#_Toc422787943)

[3.1.2 MMR算例分析 18](#_Toc422787944)

[3.1.3 MMR算法分析 21](#_Toc422787945)

[3.2 MTMDP算法介绍与分析 21](#_Toc422787946)

[3.2.1 MTMDP算法介绍 21](#_Toc422787947)

[3.2.2 MTMDP算例分析 25](#_Toc422787948)

[3.2.3 MTMDP算法分析 28](#_Toc422787949)

[3.3 MMR和MTMDP算法的对比分析 29](#_Toc422787950)

[3.4 基于Spark的MMR和MTMDP并行算法设计 32](#_Toc422787951)

[3.4.1 Spark中的RDD操作 32](#_Toc422787952)

[3.4.2 MMR和MTMDP并行化的基本思想 32](#_Toc422787953)

[3.4.3 基于Spark的MMR和MTMDP并行算法的设计 34](#_Toc422787954)

[3.5 本章小结 39](#_Toc422787955)

[第4章 实验结果与分析 40](#_Toc422787956)

[4.1 实验平台 40](#_Toc422787957)

[4.1.1 硬件支持 40](#_Toc422787958)

[4.1.2 软件支持 40](#_Toc422787959)

[4.1.3 实验平台的部署 40](#_Toc422787960)

[4.2 实验过程与分析 43](#_Toc422787961)

[4.2.1 实验一：聚类结果分析 43](#_Toc422787962)

[4.2.2 实验二：串行和并行对比分析 47](#_Toc422787963)

[4.2.3 实验三：集群并行化加速比分析 49](#_Toc422787964)

[4.3 本章小结 51](#_Toc422787965)

[结 论 52](#_Toc422787966)

[致 谢 53](#_Toc422787967)

[参考文献 54](#_Toc422787968)

# 第1章 绪论

## 1.1 本论文的背景和意义

在日常生活、生产、科研工作中，经常要对被研究对象进行分类。研究和处理给定对象的分类常用的方法就是聚类分析。聚类分析是一种重要的人类行为，是根据指定规则，将给定的数据集按照相似性进行分类的一种统计分析方法。聚类[1]是使得同一个簇中的对象之间具有较高的相似性，而不同簇中的对象具有较大的相异性。聚类分析在许多领域得到广泛应用，例如计算视觉、生物医学、信息检索、数据挖掘和模式识别等[2]。

近年来聚类分析方面的研究发展迅速，诞生了许多优秀的聚类算法，但是许多已有的聚类算法存在一个共同的不足之处，它们在聚类过程中需要依赖以往的经验数据或者先验知识，否则无法达到好的聚类效果。[粗糙集理论](http://baike.baidu.com/view/223951.htm)是一种刻划不完整性和不确定性的[数学工具](http://baike.baidu.com/view/750697.htm)，它能有效分析不精确、不一致、不完整的各类信息，还可以对数据进行分析和推理，从中发现隐含的知识，揭示潜在的规律[3]。因此非常适合处理聚类分析这样的不确定性问题。粗糙集理论的最大优势在于，它在处理问题过程中不需要依赖任何以往经验数据或者先验知识。

随着互联网和信息行业的发展，数据正在迅速膨胀。比如一分钟内，微博和Twitter上新发的数据量就超过了10万，社交网络Facebook的浏览量超过600万，大数据时代对数据的处理又提出了新的需求。面对大规模数据集和高维数据类型时，传统的单机处理无论是从运算能力或时间效率都无已法胜任实际需求，随着云计算技术的发展为聚类分析提供了新的研究方向。

本文研究了基于粗糙集模型的两种分层聚类算法MMR（Min-Min-Roughness）和MTMDP（Maximum Total Mean Distribution Precision），这两种聚类算法应用于分类数据，克服了传统聚类算法在聚类过程中需要依赖以往的经验数据或者先验知识的缺点，仅仅根据输入的数据信息和希望的簇数目就能得到很好的聚类结果。本文实现了这两种聚类算法，并在四个真实的数据集上进行了测试，分析了两种算法的性能。本文还分析了在Spark平台上并行实现这两种算法的可行性，同时在云计算平台Spark[4]上并行实现了这两种聚类算法，并在Spark平台上测试了两种算法的加速比。

## 1.2 国内外研究现状

### 1.2.1 聚类分析的研究现状

在数据挖掘研究方向，聚类分类数据获得了人们大量的关注。Huang等人根据简单匹配度测量分类数据的相异性，扩展了标准的k-means算法，提出了k-modes聚类算法[5]。分类数据相异度测量是简单地根据不同属性值的数目，计算两个对象的相异性。k-modes算法改进了k-means，使用一种基于频率的方法更新中心，使聚类的代价最小。k-modes算法包含了所有k-means的优点，并且能够解释聚类的结果。但是k-modes算法依赖于初始模型和处理顺序，产生局部的最优结果。另外k-modes算法规定一个对象只能属于一个簇，并且属于每个簇的置信度都相同。在现实世界中，数据的边界是很难区分的，所以每个簇之间往往没有很明确的边界。所以Huang和Ng提出了基于k-modes算法基础的fuzzy-k-modes[6]算法，分配隶属度给不同的簇。但是该算法依然受到初始模型和处理顺序的影响，因此fuzzy-k-modes需要调整一个控制参数来获得更好的结果。在实际应用中我们不知道如何获得最优的参数，而最优参数的值往往依赖以前的经验和先验信息。

近年来粗糙集(Rough Set)理论在聚类分析方面引起了广泛的研究。比如Cheng和Wang基于粗糙集理论和香农理论改进了聚类算法，Lingras和West提出了粗糙的k-means算法，并且用该算法来分析学生Web的访问日志。Maji和pal提出了一种基于粗糙集理论的选择最佳生物属性的fuzzy c-medoids算法。以上提到的这些算法不是有收敛缺陷，就是有稳定性问题。Chen等人提出了一种适用于分类数据的基于粗糙集的层次聚类算法，但是该算法的时间复杂度达到了，其中*n*是指对象数目，*m*是指属性数目。

### 1.2.2 粗糙集理论的研究现状

粗糙集理论是Pawlak教授于1982年提出的一种能够定量分析处理不精确、不一致、不完整信息与知识的数学工具[7]。粗糙集理论最初的原型来源于比较简单的信息模型，它的基本思想是通过关系数据库分类归纳形成概念和规则，通过等价关系的分类以及分类对于目标的近似实现知识发现。由于粗糙集理论思想新颖、方法独特，已成为一种重要的智能信息处理技术，该理论已经在机器学习与知识发现、数据挖掘、决策支持与分析等方面得到广泛应用。

粗糙集理论发展三十多年来，无论在理论研究还是应用研究上均取得了良好的成果。从1992年至今,每年都召开以粗糙集为主题的国际会议，主要有RSCTC、RSFDGrC和RSKT。在国内从2001年开始每年举办粗糙集与软计算学术会议。此外还有很多基于粗糙集理论的KDD(Knowledge Discovery in Database)系统已经建立，其中最具有代表性的有KDD-R，RoughDAS&RoughClass[8]，LERS[9]，Rosetta和RIDAS[10]等。

### 1.2.3 云计算的研究现状

云计算[11]是一种新的服务模式和应用，它基于分布式的拓扑结构和通过网络连接大量的计算机节点，这样可以使用户得到超大规模的且具有伸缩性、高可靠性的计算机资源，从而能以低廉的价格来处理大规模数据或者得到充足的存储空间。面向企业级的云计算服务包括：基础设施服务（IaaS）、平台即服务（PaaS）和软件即服务（SaaS）三种，面向普通用户的云计算服务则更倾向于云存储和云安全。[云计算](http://baike.baidu.com/view/1316082.htm)是[分布式计算](http://baike.baidu.com/view/30655.htm)、[并行计算](http://baike.baidu.com/view/1666.htm)、[效用计算](http://baike.baidu.com/view/5246792.htm)、[网络存储](http://baike.baidu.com/view/600264.htm)、[虚拟化](http://baike.baidu.com/view/729629.htm)、[负载均衡](http://baike.baidu.com/view/51184.htm)、热备份[冗余](http://baike.baidu.com/subview/104445/10102121.htm)等传统[计算机](http://baike.baidu.com/view/3314.htm)和[网络技术](http://baike.baidu.com/view/25363.htm)发展融合的产物。

## 1.3 本论文的主要内容

本论文的研究内容如下：

（1）聚类分析的基本概念和主要方法，重点研究了层次聚类算法。

（2）粗糙集理论的基本概念：等价类、上近似、下近似、粗糙度、粗糙隶属度等。

（3）了解了Spark云计算平台的编程模型，重点研究了RDD的变换和动作。

（4）基于粗糙集模型的MMR层次聚类算法的基本概念和算法的实现，分析了MMR算法的优缺点。根据MMR算法的特点，结合Spark的编程模型实现了MMR算法的并行化。

（5）基于粗糙集模型的MTMDP层次聚类算法的基本概念和算法的实现，分析了MTMDP算法的优缺点，和MMR算法进行了对比分析。根据MTMDP算法的特点，结合Spark的编程模型实现了MTMDP算法的并行化。

（6）在Soybean、Zoo、Wisconsin Breast Cancer和Mushroom四个数据集上进行了实验，并且选择了三个常用的聚类结果评价指标Overall purity、ARI、NMI对MMR和MTMDP算法的聚类结果进行评估。

（7）在Spark云平台上对并行的MMR和MTMDP算法进行性能分析，完成了串行和并行对比分析和集群加速比分析两个实验。

## 1.4 本论文的结构安排

1. 简要地介绍了论文的研究背景、目的及意义。阐述了本文的主要方法和论文的主要内容与结构安排。
2. 概述了本文相关的理论知识。介绍了数据挖掘的定义和基本过程，详细介绍了聚类分析的数据类型、相似度量、主要方法和评价指标，还介绍了粗糙集理论的基本概念，最后介绍了实验平台Spark的体系结构和特点。
3. 详细介绍了MMR和MTMDP算法的设计与实现。首先介绍了MMR算法相关的定义，然后介绍了MMR算法的主要思想，然后给出了一个具体的实例分析MMR的具体过程。再详细介绍了MTMDP算法并与MMR算法进行了对比分析。最后介绍了基于Spark平台的并行算法设计与实现。
4. 首先介绍了Spark集群的搭建过程，实验的硬件环境和软件配置。然后介绍了基于Spark的MMR和MTMDP算法聚类结果分析实验、串行和并行对比分析实验、集群加速比分析实验三个实验的详细过程，并且分析了实验结果。

最后，对本文的研究工作进行总结，对未来的研究工作进行了展望

## 1.5 本章小结

本章首先介绍了本课题的背景和意义，说明了聚类分析在各领域的广泛应用，介绍了传统聚类分析方法的不足以及本论文拟研究的两种层次聚类算法MMR和MTMDP的特点。接着，介绍了聚类分析、粗糙集理论和云计算的国内外研究现状。最后概述了本论文的主要内容和结构安排。

# 第2章 相关知识概述

## 2.1 数据挖掘

数据挖掘(Data Mining)一般是指从大量的数据中通过算法搜索隐藏于其中信息的过程，数据挖掘又称数据库中的知识发现，是人工智能和数据库领域研究的热点问题。数据挖掘主要基于机器学习、模式识别、统计学、人工智能、可视化技术等，分析大量的数据，挖掘潜在的模式，获取的信息和知识可以广泛地应用于市场分析、欺诈检测、产品控制和科学探索等。

### 2.1.1 数据挖掘的定义

数据挖掘就是从大量的、不完整的、有噪声的、不一致的真是世界中的数据，提取出未知的、有用的信息和知识的过程[12]。其中数据源必须是真实的、大量的、含噪声的，挖掘出的知识是可理解、可运用的。从商业的角度上，数据挖掘可以描述为，按照企业既定业务目标，对大量的企业数据进行探索和分析，揭示隐藏的、未知的规律，并进一步将其模型化的先进有效方法。

### 2.1.2 数据挖掘的基本过程

数据挖掘包括三个阶段：数据准备、数据挖掘、结果的表达与解释。其中数据准备阶段又包括：信息收集、数据清理、数据集成、数据选择、数据变换、数据挖掘、模式评估和知识表示等步骤[13]。

## 2.2 聚类分析

将物理或抽象对象的集合分成相似的对象类的过程称为聚类，簇是数据对象的集合，这些对象与同一个簇中的对象彼此相似，而与其他簇中的对象相异[14]。尽管分类是一种有效的手段，但是它常常需要高昂的代价收集和标记大量训练元组集或模式，以便分类法使用它们对每个组建模。

在聚类分析中主要涉及三个要素，分别是相似性度量、聚类算法、聚类准则函数，一般的聚类分析方法是通过多次迭代执行找出使得聚类准则函数达到最优的聚类结果。一般聚类分析的流程图如图2-1所示。

### 2.2.1 聚类分析的数据类型

假设聚类的数据集合包含个数据对象，这些对象可能代表人、房子、文档、国家等，基于内存的聚类算法通常可以采用以下两种数据结构运行：

图2-1 聚类分析流程图

Y

数据预处理

定义距离函数

按给定规则进行聚类

评估和输出

满意

开始

结束

调整参数

N

（1）数据矩阵（data matrix，或称为对象与变量结构）：它用*p*个变量（也称为度量或属性）来表现*n*个对象，例如用年龄、身高、体重、性别、种族等属性来表现对象“人”。这种数据结构是关系表的形式，或者看成*n*×*p*（*n*个对象×*p*个变量）矩阵：

 (2-1)

（2）相异度矩阵（dissimilarity matrix，或称为对象－对象结构）：存储所有成对的*n*个对象两两之间的近似性，通常用一个*n*×*n*维的矩阵表示，其中是对象和对象之间的测量差或相异度，通常是一个非负的数值，当对象和越相似或“接近”时，其值越接近0；两个对象越不同，其值越大。其中，并且。

数据矩阵的行和列代表不同的实体，而相异度矩阵的行和列代表相同的实体。因而，数据矩阵经常称为二模(two-mode)矩阵，而相异度矩阵称为单模(one-mode)矩阵。很多的聚类算法都使用相异度矩阵进行聚类。

(2-2)

### 2.2.2 聚类分析中的相似度量方法

在数据挖掘中聚类分析中经常出现的数据变量类型有区间标度变量、二元变量、分类变量、序数型变量、比例标度变量、混合类型变量和向量变量等[15]。不同的数据类型之间有不同的相异度测量方法。

（1）区间标度变量：区间标度变量是一个粗略现行标度的连续度量。典型的例子包括重量和高度、经度和纬度坐标（如聚类房屋），以及大气温度。其距离度量一般包括欧几里得距离、曼哈顿距离以及闵可夫斯基距离。

欧几里得距离定义如下：

 (2-3)

其中和表示两个维的数据对象。

曼哈顿（或城市块）距离定义如下：

 (2-4)

闵可夫斯基距离定义如下：

 (2-5)

（2）二元变量：只有两个状态：0或1，0表示该变量不出现，1表示该变量出现。例如，给出一个描述病人的变量smoker，1表示病人抽烟，而0表示病人不抽烟。如果像对待区间标量那样处理二元变量会产生误导的聚类结果，所以二元变量要采用特定的方法来计算其相异度。如果一个二元变量的两个状态具有同等价值和相同的权重，则该二元变量是对称的。如果输出的状态不是同等重要的，比如疾病的判断，这样的二元变量是非对称的。

对称二元相异度定义如下：

 (2-6)

非对称相异度定义如下：

 (2-7)

其中表示两个对象，是对象和值都为1的变量数目，是对象值为1，但是对象值为0的变量数目，是对象值为0，但是对象值为1的变量数目，t是对象和值都为0的变量数目。

（3）分类变量：分类变量是二元变量的推广，它可以取于两个的状态值，例如颜色是一个分类变量，它可能有多种状态：红、绿、蓝等。

分类变量的相异度定义如下：

 (2-8)

其中，是匹配的数目（即对象和对象取值相同状态的变量的数目），表示总的变量数目。

（4）向量对象：在某些应用中，如在信息检索、文本文档聚类和生物学分类中，需要对包含了大量符号实体（如关键词和短语）的复杂对象（如文档）进行比较和聚类。为了测量复杂对象间的距离，通常期望放弃传统的度量距离计算，而引入非度量的相似度函数。

向量对象的余弦相似度定义如下：

 (2-9)

其中表示两个向量，是向量的转置，是向量的欧几里得范数，是向量的欧几里得范数，本质上是向量之间的夹角的余弦值。

### 2.2.3 聚类分析方法简介

没有任何一种聚类算法可以普遍适用于揭示各种多维数据集所呈现出来的多种多样的结构[16]。根据数据的积聚规则以及应用这些规则的方法，有多种聚类算法。主要的聚类算法包括以下几种。

（1）基于划分的方法：划分式聚类算法需要预先指定聚类数目或聚类中心，通过反复迭代运算，逐步降低目标函数的误差值，当目标函数值收敛时，得到最终聚类结果。典型的划分聚类算法有k-means和k-modes。

（2）基于层次的方法：它使用数据的联接规则，透过一种层次架构方式，反复将数据进行分裂或凝聚，将数据对象集的层次分解。典型的层次聚类算法有CURE和Chameleon。

分裂的层次聚类算法：使用自顶向下的策略。开始它把所有对象置于一个簇中，该簇是层次结构的根。然后它把根上的簇划分成许多个较小的子簇，并且递归地把这些簇划分成更小的簇，划分过程继续，直到每个对象自成一簇，或者达到了某个终止条件，例如达到了某个希望的簇数目，或者某个簇的直径都在某个阈值内。

凝聚的层次聚类：使用自底向上的策略，首先每个对象作为其簇，然后合并这些原子簇为越来越大的簇，直到所有的对象都在一个簇中，或者满足某个终止条件。

本文中采用分裂的层次聚类算法，并且一旦分裂执行就不能修正，所以如何选择分裂的属性和选择继续分裂的节点显得特别关键。

（3）基于密度的方法：只要“邻域”中的密度（对象或数据点的数目）超过某个阈值，就继续增长给定的簇。也就是说，对给定类中的每个数据点，在一个给定的区域内必须至少包含某个数目的点。这样的方法可以用来过滤“噪声”孤立点数据，可以发现任意形状的簇。典型的基于密度的聚类算法有DBSCAN、OPTICS和DENCLUE。

（4）基于网格的方法：该方法使用一种多分辨率的网格数据结构。它将对象空间量化为有限数目的单元，形成网格结构，所有的聚类操作都在网格上进行。基于网格的聚类算法常常与其它方法相结合，特别是与基于密度的聚类方法相结合。这种方法的主要优点是处理速度快，其处理时间独立于数据对象的数目，仅依赖于量化空间中每一维上的单元数目。典型的网格聚类算法有STING和CLIQUE。

（5）基于模型的方法：试图优化给定数据和数据模型之间的拟合。这类方法经常基于这样的假设：数据根据潜在的混合概率分布生成。典型的基于模型的聚类算法有EM（期望最大）。

目前针对聚类算法的研究主要着重于两个方面：第一，聚类对象的输入顺序以及初始聚类中心的选择方面。这是聚类研究中的一个重点，因为这两个因素会直接影响到聚类结果的好坏，如何能使最终的聚类结果达到全局最优是进一步研究的方向。第二，关于算法执行效率的方面。除了聚类的质量，效率也是算法优劣至关重要的考量因素。随着数据量的急剧增加，如何改进现有聚类算法，提高算法效率，使其能够快速有效处理大规模数据成为亟待解决的另一个研究课题。

### 2.2.4 聚类分析的有效性

聚类分析是一个富有挑战性的研究领域，它的潜在应用提出了各自特殊的要求，数据挖掘对聚类的要求很多，下面介绍一些常用的聚类分析评价指标。

（1）可伸缩性：能否无论是在小数据集还是包含百万个对象的大数据集上，都能够取得良好的聚类结果。

（2）处理不同数据类型属性的能力：能否处理区间变量、二元变量、分类变量、序数变量等混合的数据类型。

（3）发现任意形状的聚类：许多聚类算法基于欧几里得或曼哈顿距离度量来确定簇，基于这种距离度量的算法只能发现球状形的簇，但是簇的形状可能是任意的。所以能否发现不同形状的簇是很重要的。

（4）对于决定输入参数的领域知识需求最小：许多的聚类算法在聚类分析中要求用户输入一定的参数，比如期望的簇的数目，聚类结果又对于输入参数非常敏感，而面对实际的问题时恰当参数的选取很困难，对参数依赖过大会使聚类质量难以控制。所以能通过算法对相应数据判断自行聚类的算法更好。

（5）处理带噪声数据的能力：绝大多数真实世界中的数据库都包含了离群点、缺失、未知或者错误的数据。一些聚类算法对于这样的数据敏感，可能会导致低质量的聚类。

（6）增量聚类和对输入记录的次序不敏感：一些聚类算法不能讲新插入的数据合并到已经的聚类结构中去，只能重新聚类。一些算法以不同的输入顺序提供对象时可能会产生不同的聚类结果。开发增量聚类算法和对输入数据不敏感的算法具有重要的意义。

## 2.3 粗糙集理论

[粗糙集理论](http://baike.baidu.com/view/223951.htm)是一种刻划不完整性和不确定性的[数学工具](http://baike.baidu.com/view/750697.htm)，它能有效分析不精确、不一致、不完整的各类信息，还可以对数据进行分析和推理，从中发现隐含的知识，揭示潜在的规律[17]。粗糙集理论的最大优势在于，它在处理问题过程中不需要依赖任何以往经验数据或者先验知识。粗糙集近年来受到越来越多的关注，应用越来越广泛，已经成为人工智能理论及其应用领域中的研究热点之一。

（1）信息系统：一般情况下，结构化数据都可以用一张表来存储，每一行记录一个对象相关的事实。一张数据表又叫做一个信息系统。更准确的说，信息系统(IS)，通常可以按照下面的格式来描述：



是一个有限的非空的对象的集合，叫做全集；

是非空的有限的属性的集合；

，是指属性*a*的值域，是一个有限的无序的集合；

是一个映射，是对象关于属性的一个映射；

（2）不可分辨关系：对于任何的一个集合，都有一种不可分辨关系，不可分辨关系表示如下：

，和分别是对象*x*和对象*y*在条件属性*a*上的取值。在某属性集*B*上的不可分辨关系，就是在属性集*B*上取相同值的对象集合。

（3）等价类：不可分辨关系把全集划分成互斥的几个类，是根据划分的等价类，其中。明显地，如果两个对象属于一个相同的等价类，相对于属性集*B*这两个对象是不可分辨的。等价类也称为基础集。

（4）上近似、下近似：对任意的集合集合可以近似的用两个关于属性*B*的集合来描述，和，和分别叫做属性集*B*关于集合*X*的下近似和上近似。下近似表明了根据知识*B*判定肯定属于*X*的*U*中元素组成的集合，即如果一个对象，那么这个对象一定属于集合*X*。上近似表明了根据知识*B*判断可能属于集合*X*的*U*中的元素组成的集合，即如果一个对象，那么这个对象有可能属于集合*X*。

（5）粗糙度：集合的不精确性是由于边界区域的存在而引起的，集合的边界区域越大，其精确性就越低，为了更精确的表达这一点，引入了粗糙度这个概念，*X*是*U*的一个子集，集合*X*相对于属性集*B*的粗糙度定义如下：

 (2-10)

其中表示集合的对象数目，显然，如果，那么，集合*X*关于属性集*B*是没有边界的，即集合*X*关于属性集*B*是清晰的。如果，显然，那么集合*X*关于属性集*B*就是有边界的，*X*关于属性*B*是粗糙的。粗糙度反映了我们根据知识*B*对集合*X*的了解程度。

（6）粗糙隶属度：给定一个属性集*B*，给定一个对象，粗糙隶属度是指对象属于集合*X*的概率()，对象*x*属于集合*X*的粗糙隶属度用表示，计算公式如下：

 (2-11)

## 2.4 Spark平台介绍

Spark是近几年发展迅速的云平台，已经成为了Apache顶级的孵化项目，是由UC Berkeley AMP实验室开发的开源通用并行云计算平台，目的是加快数据分析的速度。Spark基于MapReduce[18]思想实现分布式计算，但是采用了内存分布式数据集，引进了内存集群计算的概念，可以将运算的中间结果存储在内存中，从而不再需要读写HDFS，因此Spark能更好地运行图应用、机器学习等迭代式的算法和交互式的数据挖掘算法。Spark内核是用Scala语言开发的，Scala是一门集成了面向对象编程和函数式编程的各种优点的语言，能将操作分布式数据集像操作本地集合对象一样容易，使用户可以非常便捷地编写并行任务。

### 2.4.1 Spark的体系结构

目前，Spark已经发展成为包含各式各样子项目的大数据计算平台，Spark的体系结构如图2-2所示。其核心框架是Spark，同时涵盖了支持结构化数据SQL查询与分析的查询引擎Spark SQL，提供了Hive命令接口的Shark方便了用户对数据的操作，提供了底层的分布式机器学习库Mlib、基于BSP模型的并行计算图框架GraphX、流计算框架Streaming，实现了Google的PageRank算法的Bagel等子项目。这些子项目在Spark上层提供了更高层、更丰富的计算框架。



图2-2 Spark的体系结构

### 2.4.2 Spark的核心概念

Spark是一个基于内存计算的大数据分布式编程框架，Spark的核心是RDD（弹性分布式数据集），RDD是Spark最基本的抽象，是对分布式内存的抽象使用，实现了以操作本地集合的方式来操作分布式数据集。RDD表示已经被分区，不可变的能够被并行操作的数据集合，RDD必须是可序列化的，可以缓存到内存中，省去了MapReduce大量的磁盘IO操作，提升了很大的效率。

RDD只能从持久存储或通过Transformations变换操作产生，相比于分布式共享内存（DSM）可以更高效实现容错，对于丢失部分数据分区只需根据创建该RDD的一系列转换记录即Lineage就可重新计算出来，而不需要做特定的Checkpoint。RDD和共享内存(DSM)的对比结果如表2-1所示。RDD的不变性，可以实现类似Hadoop MapReduce的推测式执行。RDD的数据分区特性，可以通过数据的本地性来提高性能，这与Hadoop MapReduce是一样的。RDD都是可序列化的，在内存不足时可自动降级为磁盘存储，把RDD存储于磁盘上，这时性能会有大的下降但不会差于现在的MapReduce。

总结RDD的特点如下：

（1）在集群结点上不可变的、已分区的集合对象；

（2）通过并行转换的方式创建如map、filter、join、groupBy等；

（3）失败时自动重建；

（4）可以控制存储级别，根据内存、磁盘等进行重用；

（5）必须是可序列化的；

（6）是静态类型的。

在RDD空间中，每个RDD都可以使用5个方面的特性来表示：

（1）分区列表（数据块列表）；

（2）计算每个分片的函数，根据父RDD计算出此RDD；

（3）对父RDD的依赖列表；

（4）对Key-Value RDD的Partitioner；

（5）每个数据分片的预定义地址列表（如HDFS上的数据块的地址）。

表2-1 RDD与DSM对比

|  |  |  |
| --- | --- | --- |
| 对比项目 | RDD | 分布式共享内存（DSM） |
| 读 | 批量或细粒度操作 | 细粒度操作 |
| 写 | 批量转换操作 | 细粒度操作 |
| 一致性 | 不重要（RDD是不可更改的） | 取决于应用程序或运行时 |
| 容错性 | 细粒度，低开销（使用Lineage） | 需要检查点操作和程序回滚 |
| 落后任务的处理 | 任务备份 | 很难处理 |
| 任务安排 | 基于数据存放的位置自动实现 | 取决于应用程序（通过运行时实现透明性） |
| 如果内存不够 | 与已有的数据流系统类似 | 性能较差 |

对于RDD可以有两种操作方式：变换（返回值仍然是一个RDD）与动作（返回值不是一个RDD）。变换如map、filter、groupBy和join等。变换操作是Lazy的，也就是说从一个RDD转换生成另一个RDD的操作不是马上执行，当Spark在遇到变换操作时只会记录需要这样的操作，并不会去执行，直到有动作操作的时候才会真正启动计算过程进行计算。动作是向应用程序返回值，或向存储系统导出数据的那些操作，例如，count（返回RDD中的元素个数），collect（返回元素本身），save（将RDD输出到存储系统）。在Spark中，只有在动作第一次使用RDD时，才会计算RDD（即延迟计算）。这样在构建RDD的时候，运行时通过管道的方式传输多个变换。

### 2.4.3 Spark编程接口

Spark用Scala语言实现了RDD的API。Scala是一种基于JVM的静态类型、函数式、面向对象的语言。Scala语言简洁（特别适合交互式使用）、有效（JVM上的强制静态类型），方便用户编写并行程序。

Spark由Master主节点和Workers从节点组成，用户编写的Spark程序被称为Driver程序，Dirver程序会连接到集群以运行Worker，Driver程序定义了一个或多个RDD，并调用RDD上的变换与动作。对RDD的变换与动作通过Scala闭包（字面量函数）来表示，Scala使用Java对象来表示闭包，并且都是可序列化的，以此把对RDD的闭包操作发送到各Workers节点。Workers存储着数据分块和享有集群内存，是运行在工作节点上的守护进程，当它收到对RDD的操作时，根据数据分片信息进行本地化数据操作，生成新的数据分片、返回结果或把RDD写入存储系统。如图2-3所示，用户的driver程序启动多个worker，worker从分布式文件系统中读取数据块，并将计算后的RDD分区缓存在内存中。

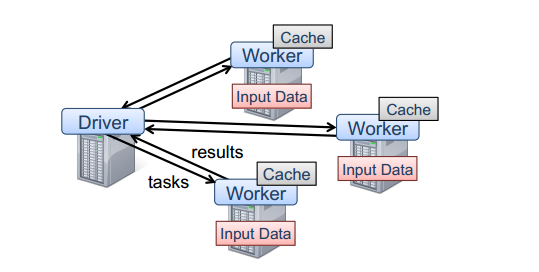


图2-3 Spark程序的运行图[19]

## 2.5 本章小结

本章首先介绍了数据挖掘的定义和基本过程。然后介绍了聚类分析的数据类型、不同数据类型的距离度量，再简要概述了主要的聚类分析方法，以及聚类分析常用的评价指标。本章还介绍了粗糙集的信息系统、等价类、上近似、下近似、粗糙度和粗糙隶属度等基本概念。最后介绍了本次实验平台Spark的体系结构、核心概念RDD和编程接口。

# 第3章 算法设计与分析

## 3.1 MMR算法介绍与分析

目前有很多不同的聚类分析方法，但是由于现实数据库中的数据大多是多值的分类数据，增加了聚类分析的难度，虽然目前已经提出了一些针对分类数据的聚类算法，但是仍然面临着各式各样的挑战，比如无法在聚类过程中无法处理不确定性问题，或者算法没有很好的稳定性等问题。本文介绍了一种适合分类数据的聚类算法，该算法在粗糙集理论的基础上提出了聚类规则，能够处理聚类过程中的不确定性问题，叫做二重最小粗糙度MMR(Min-Min-Roughness)算法[22]。

### 3.1.1 MMR算法介绍

#### 3.1.1.1 相关定义

**定义3.1**[22]给定一个属性，是属性所有可能取值的集合，*X*是的一个等价类，这说明了*X*是一个有明确值的集合的子集。集合*X*关于属性且的粗糙度定义为，计算如下所示：

 (3-1)

**定义3.2**[22] 属性关于属性的平均粗糙度()定义如下：

 (3-2)

**定义3.3**[22] (Min-Roughness，MR)最小粗糙度。给定一个分类信息系统，信息系统如所示，属性的MR被定义为该属性和相对属性中最小的平均粗糙度的值，定义如下：

 (3-3)

**定义3.4**[22] MMR(Min-Min-Roughness)二重最小粗糙度。是一个信息系统，MMR是所有属性中最小的Min-Roughness值，定义如下：

 (3-4)

#### 3.1.1.2 算法主要步骤

定义3.3中MR值表明了每个属性能够达到的最大清晰度，定义3.4中MMR值确定所有属性中的最佳分裂属性。MMR算法规定从一系列的候选属性中选择一个具有最小值的Min-Roughness的属性作为分裂属性。在上面提到的这些定义的基础上，Parmar等人提出了一个自上而下的分裂层次聚类算法，叫做MMR（二重最小粗糙度）算法[22]。这个算法不断地根据Min-Min-Roughness选择分裂属性进行分裂，直到达到预先设定生成的簇数目。这个算法仅仅要求输入所有的对象和一个预先给定的参数*k*值（最终生成的聚类数目），就能得到很好的聚类结果，算法的伪代码如表3-1所示。

MMR算法的主要步骤如下：

输入：数据集*U*和最终生成的簇数目*k*

Repeat

Step1：计算每个属性的等价类

Step2：根据等价类计算每个属性的最小平均粗糙度MR

Step3：根据MMR值选择最佳的分裂属性

Step4：选择分裂属性的最佳分裂值。

Step5：根据结点对象数目选择继续分裂的结点

Until当前簇数目>=*k*

输出：*k*个簇的集合

表3-1 MMR算法伪代码

|  |
| --- |
| Procedure MMR(*U*, *k*)  Begin  //设置初始簇数目，初始化全部对象为一个簇  Set current number of cluster CNC=1  Set ParentNode=*U*  While（CNC< *k*）  //计算当前结点所有属性的等价划分类  For each  Determine partition CNode/  End  //计算所有结点的平均粗糙度  For each  For each  Calculate  End  Calculate  End  //选择具有最小值MR的属性作为分裂属性  Determine Splitting attributecorresponding to Min-Min-Roughness  //根据分裂属性等价类的平均粗糙度选择分裂属性值  Determine Splitting attribute val  //根据分裂的属性值进行二分裂  Do binary splitting attribute=val  CNC=CNC+1  //如果当前簇数目仍未达到预先设定的*k*值，选择对象数目多的结点继续进行分裂  if CNC< *k* then  CNode=leaf node has more objects  end if  end while  End |

### 3.1.2 MMR算例分析

下面我们提出了一个具体的例子来说明MMR算法一次迭代的过程。如表3-2所示的信息系统，一共有10个对象(=10)，每个对象都包含了6个属性(=6)，每个分类属性最多取4个不同的值(=4)。

表3-2 信息系统

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| U |  |  |  |  |  |  |
| 1 | Big | Blue | Hard | Infinite | Plastic | Negative |
| 2 | Medium | Red | Moderate | Smooth | Wood | Neutral |
| 3 | Small | Yellow | Soft | Fuzzy | Plush | Positive |
| 4 | Medium | Blue | Moderate | Fuzzy | Plastic | Negative |
| 5 | Small | Yellow | Soft | Indefinite | Plastic | Neutral |
| 6 | Big | Green | Hard | Smooth | Wood | Positive |
| 7 | Small | Yellow | Hard | Indefinite | Metal | Positive |
| 8 | Small | Yellow | Soft | Indefinite | Plastic | Positive |
| 9 | Big | Green | Hard | Smooth | Wood | Neutral |
| 10 | Medium | Green | Moderate | Smooth | Plastic | Neutral |

Step1：计算每个属性的等价类

以计算属性*a*1，*a2*的等价类为例。该信息系统根据属性划分可以得到如下三个等价类。



该信息系统根据属性划分可以得到如下四个等价类。



Step2：根据等价类计算每个属性的最小平均粗糙度MR

以属性为例，计算属性关于属性的平均粗糙度。根据上近似、下近似的定义，求出的上、下近似结果如表3-3所示，根据公式(3-2)平均粗糙度的定义计算出属性关于属性的平均粗糙度是(0+5/6+10)/3=0.611，重复相同的步骤计算属性关于属性,,,的平均粗糙度，计算结果如表3-4所示。根据表3-4求出属性的最小平均粗糙度MR()=0.5238。重复相同的步骤，计算出属性,,,,,的MR值如表3-5所示。

表3-3 属性的等价类关于属性的平均粗糙度

|  |  |  |  |
| --- | --- | --- | --- |
| 集合 | *X*1(Small) | *X*2(Medium) | *X*3(Big) |
| 下近似 | {3,5,7,8} | {2} | {} |
| 上近似 | {3,5,7,8} | {1,2,4,6,9,10} | {1,4,6,9,10} |
|  | 1 | 0.177 | 0 |
|  | 0 | 0.8333 | 1 |

表3-4 属性的平均粗糙度

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 相对属性 | *X*1(Small) | *X*2(Medium) | *X*3(Big) | Mean roughness |
|  | 0 | 0.8333 | 1 | 0.6111 |
|  | 0.5714 | 0 | 1 | 0.5238 |
|  | 1 | 1 | 1 | 1 |
|  | 0.7143 | 1 | 1 | 0.9048 |
|  | 1 | 1 | 1 | 1 |
| 平均值 | 0.65714 | 0.76666 | 1 | 0.80814 |

Step3：根据MMR值选择最佳的分裂属性。

根据表3-5的计算结果和可以得到属性*a*1和属性*a*3的最小平均粗糙度MR值相同，都是0.5238。在MMR算法中如果根据公式(3-4)得到了多个相同的MMR值，则继续比较该属性的次小值，直到这种平衡被打破，如果这种平衡一直没有打破就随机地选择一个属性进行分裂。比如在这次迭代过程中，属性*a*1和*a*3的最小平均粗糙度值相同，则继续比较属性*a*1和*a*3的平均粗糙度的次小值，次小值分别是0.6111和0.9074，由于属性*a*1平均粗糙度的次小值更小，所以选择属性*a*1作为分裂属性。

表3-5 MMR算法计算结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 基本属性 | 平均粗糙度 | | | | | |  | |
|  |  |  |  |  |  |  | 次小值 |
|  |  | 0.6111 | 0.5238 | 1 | 0.9048 | 1 | 0.5238 | 0.6111 |
|  | 0.7500 |  | 0.8929 | 1 | 0.9286 | 0.7500 | 0.7500 |  |
|  | 0.5238 | 0.9444 |  | 1 | 0.9074 | 1 | 0.5238 | 0.9074 |
|  | 1 | 0.6667 | 1 |  | 0.7639 | 1 | 0.6667 |  |
|  | 1 | 0.8820 | 1 | 1 |  | 0.9500 | 0.8820 |  |
|  | 1 | 0.6250 | 1 | 1 | 0.9333 |  | 0.6250 |  |

Step4：选择分裂属性的最佳分裂值。根据Step3已经确定了属性*a*1作为分裂属性，但是属性*a*1可以取Small、Medium、Big三个取值，选择最优的分裂属性值进行二分裂是一个NP难的问题。MMR算法根据平均粗糙度来简化问题的复杂度，选择具有最小值的平均粗糙度的属性值进行分裂。根据表3-4所示，属性值*a*1=Small具有最小平均粗糙度0.65714，所以选择属性*a*1=Small进行二分裂，分裂结果如图3-1所示。

图3-1 MMR算法第一次分裂结果

Medium-Big

(1, 2, 4, 6, 9, 10)

Small

(3, 5, 7, 8)

Step5：根据结点对象数目选择继续分裂的结点。

MMR算法选择对象数目多的结点继续进行分裂，比如第一次分裂后产生了图3-1所示的分裂结果，左叶子结点*a*1=*Big*∪*Medium*一共有6个对象，右叶子结点*a*1=Small一共有4个对象，根据MMR算法要求，选择对象数目更多的左叶子结点继续分裂。

重复Step1~Step5相同的计算过程，直到叶子结点数目等于预先设定的*k*值，算法结束。

### 3.1.3 MMR算法分析

给定这样的一个数据集，有*n*个对象，*m*个属性，每个属性最多有*l*个取值，*k*是最终要生成的簇数目。要产生*k*个簇就要进行*k-*1次迭代。每次迭代的过程，计算出所有属性的等价类的时间是*nm*，计算平均粗糙度的时间近似是*m*2*l*(*n*2)，计算出MR和MMR的时间是2*m*。所以MMR聚类算法的时间复杂度是O(*knm+km*2*ln*2)。MMR算法的时间复杂度还是很高的，随着数据规模的增大，算法的性能会越来越低，所以我们考虑利用云平台实现并行的MMR算法，以提高该算法的效率。

## 3.2 MTMDP算法介绍与分析

MMR算法应用粗糙度的基本概念来选择分裂的属性，选择对象数目多的叶子结点继续进行分裂。MMR算法两个主要的缺点：(1)粗糙度并不能有效地反映边界对象的分辨度。(2)选择对象数目多的叶子结点继续进行分裂可能会导致一些不理想的聚类结果。

受到了MMR算法的启发，Li等人提出了一种基于概率的粗糙集理论的适用于分类数据的聚类算法MTMDP(Max Total Mean Distribution Precision)[23]。该算法不但包含了MMR算法的所有优点，还针对算法的两个关键步骤进行了改进。第一，MTMDP算法根据所有属性的平均分布精度(Mean Distribution Precision)来选择分裂的属性，这个指标比MMR算法中的MR(Min-Roughness)要好，平均分布精度更能有效地反映集合的边界。第二，MTMDP算法根据所有叶子结点的聚合度来选择继续进行分裂的结点，比MMR算法中选择对象更多的叶子结点方法更合理。在真实世界数据集上的实验结果表明MTMDP算法比MMR算法产生了更好的聚类结果。

### 3.2.1 MTMDP算法介绍

#### 3.2.1.1相关定义

**定义3.5**[23] 概率分布近似集合。是一个分类的信息系统，并且，集合*X*关于属性集*B*的概率分布近似集合定义如下：

 (3-5)

子上标*d*代表了分布近似，该定义表明了集合*X*关于属性集*B*的概率分布近似集是集合*X*的一个概率粗糙集，并且在这个集合中每个对象*x*的隶属度是用根据属于的等价类中的对象关于集合*X*的粗糙隶属度来计算。

**定义3.6**[23] 分布近似精度。是一个分类的信息系统，并且*X*是*U*的非空一个子集，集合*X*关于属性集*B*的分布近似精度定义如下：

 (3-6)

从该定义可以知道，集合*X*关于属性集*B*的分布近似精度就是集合*X*中所有对象的粗糙隶属度之和，显然。如果，那么集合*X*是清晰的，否则集合*X*和其他等价类之间一定有相同的对象。公式(3-6)和计算公式(3-7)是等价的。

 (3-7)

从该定义可以了解到分布近似精度根据对象的粗糙隶属度来推算这个对象是可能属于集合*X*还是一定属于集合*X*。因此，即使在的条件下仍然可以用来比较两个条件属性集*P*和*B*的分辨率。公式(3-7)是公式(3-6)的一种快速计算方法，因为如果把所有对象根据属性集*B*按照字典顺序排序，这样就能轻松地计算出的所有等价类。

**定义3.7**[23] 平均分布精度。是一个分类数据的信息系统，是指属性所有取值的集合，属性关于属性的MDP平均分布精度定义如下：

 (3-8)

其中是属性能够取不同值的数目，表示根据属性划分的等价类关于属性的平均分布精度。取值范围是0到1，如果，那么根据属性划分的所有等价类关于属性都是清晰的，否则根据属性划分的等价类之间存在耦合。的值越大，说明根据属性划分的等价类之间的耦合度越小，因此可以作为一个衡量等价类关于属性的耦合度的指数。

此外，一个较大值的表明了关于属性的等价类很可能和的等价类分布相同，近而在等价类中的对象有更多的相似性。

**定义3.8**[23] 总平均分布精度是一个关于分类数据的信息系统，属性的总平均分布精度TMDP定义如下：

 (3-9)

表示了根据属性划分成等价类的总平均分布精度，取值范围从0到1，显然体现了根据划分的等价类之间的总耦合度。的值越大，根据划分的等价类之间的总耦合度越小，所以可以作为选择分裂属性的参数。

**定义3.9**[23] MTMDP(Max Total Mean Distribution Precision)最大总平均分布精度。是一个分类信息系统，MTMDP是所有属性中最大的总平均分布精度，定义如下：

 (3-10)

**定义3.10**[23] 我们用粒度的概念来描述一个集合的确定度。集合，根据属性*a*划分的等价类，集合*X*关于属性*a*的粒度定义如下：

 (3-11)

描述了从集合*X*中选择两个对象，这两个对象属于同一个等价类的概率。显然。的值越大，那么集合*X*关于属性*a*的确定性就越大，

**定义3.11**[23] 聚合度。我们介绍一下MTMDP算法中聚合度的概念，聚合度是选择哪个叶子结点继续进行聚类的参数。集合*X*的聚合度定义如下：

 (3-12)

显然，的值越大，集合*X*的聚合度越大。最大总平均分布精度MTMDP算法选择具有最小聚合度的叶子结点继续进行分裂。

#### 3.1.1.2 算法主要步骤

MTMDP聚类算法的主要目标就是把所有对象分成几个组，使每个组中的对象间具有最大的相似性，和其他组中的对象具有最大的差异性。换句话说，一个簇的聚合度越大越好，簇和簇之间的耦合度越小越好。在总平均粗糙度TMDP定义的基础上，Li等人构建了一个自上而下的层次聚类算法，叫做MTMDP（Maximum Total Mean Distribution Precision）最大总平均分布精度算法[23]。

MTMDP算法的工作流程如下：首先把所有的对象*U*作为根结点，选择具有最大值的，将属性作为分裂属性，根节点进行一次二分裂产生两个新的叶子结点。接着，选择具有最小聚合度的叶子结点继续进行分裂直到叶子结点的数目达到了预先设定的簇数目*k*值。表3-6是MTMDP算法的伪代码。在选择分裂属性之前，MTMDP算法把只具有单值的属性剔除了，因为单值属性不可能作为分裂的属性。

MTMDP算法的主要步骤如下：

输入：数据集*U*和最终生成的簇数目*k*

Repeat

Step1：计算每个属性的等价类

Step2：根据等价类计算每个属性的总平均分布精度TMDP

Step3：根据MTMDP值选择最佳的分裂属性

Step4：选择分裂属性的最佳分裂值。

Step5：根据结点的Cohesion聚合度选择继续分裂的结点

Until当前簇数目>=*k*

输出：*k*个簇的集合

表3-6 MTMDP算法伪代码

|  |
| --- |
| Procedure MTMDP(*U*, *k*)  Begin  //设置初始簇数目，初始化全部对象为一个簇  Set current number of cluster CNC=1  Set ParentNode=*U*  While（CNC< *k*）do  B=A  //计算当前结点所有属性的等价划分类  For each  Determine partition CNode/  //如果当前属性只有一个取值，则不可能作为分裂属性  if|CNode/|=1  B=B-  end if  End  //计算所有结点的总平均分布度TMDP  For each  For each  Calculate  End  Calculate  End  //选择具有最大值TMDP的属性作为分裂属性  Determine Splitting attributecorresponding to Max Total Mean Distribution Precision  //选择分裂属性值  Determine Splitting attribute val  //根据分裂的属性值进行二分裂  Do binary splitting attribute with *a*i=val  CNC=CNC+1  //如果当前簇数目仍未达到预先设定的*k*值，选择具有最小聚合度的结点继续进行分裂  if CNC< *k* then  CNode=leaf node has the minimal cohesion degree  end if  end while  End |

### 3.2.2 MTMDP算例分析

下面我们提出了一个具体的例子来说明MTMDP算法如何如确定最佳分裂属性值过程。如表3-2所示的信息系统，一共有10个对象(=10)，每个对象都包含了6个属性(=6)，每个分类属性最多取4个不同的值(=4)。接下来给出MTMDP算法第一次迭代的过程。

Step1：计算所有属性的等价类，以属性为例。根据属性划分的等价类如下：



Step2：根据等价类计算每个属性的总平均分布精度TMDP。

以属性为例。我们可以根据属性排序，然后就能快速计算出属性的平均分布精度。在表3-7中的前三行显示的是根据属性字典顺序的排列结果，从排序的结果我们可以得到根据属性划分的所有等价类，表格3-7中不同颜色代表关于属性的一个等价划分。在这样的基础之上我们可以根据粗糙隶属度的定义，直接计算所有对象的，表3-7的第四行显示了部分的计算结果。

表3-7 计算属性关于的平均分布精度

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| U | 1 | 4 | 6 | 9 | 10 |
|  | Blue | Blue | Green | Green | Green |
|  | Big | Medium | Big | Big | Medium |
|  | 1/2 | 1/2 | 2/3 | 2/3 | 1/3 |
| U | 2 | 3 | 5 | 7 | 8 |
|  | Red | Yellow | Yellow | Yellow | Yellow |
|  | Medium | Small | Small | Small | Small |
|  | 1/1 | 4/4 | 4/4 | 4/4 | 4/4 |

接下来根据属性划分的等价类的平均分布精度计算如下：



属性关于属性的平均分布精度计算如下：



重复相似的计算过程，我们得到了属性关于属性和的平均分布精度，计算结果如表3-8所示，表格的最后一列记录了每个属性的平均分布精度，表格的最后一行记录了等价类的平均分布近似精度值。

根据公式(3-9)可以计算，类似地可以计算出所有属性的总平均分布精度，计算结果如表3-9所示。

表3-8 属性的平均分布精度

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 相对属性 | *X*1(*Small*) | *X*2(*Medium*) | *X*3(*Big*) | MDP |
|  | 1 | 0.611 | 0.611 | 0.741 |
|  | 0.8125 | 1 | 0.75 | 0.854 |
|  | 0.688 | 0.5 | 0.417 | 0.535 |
|  | 0.7 | 0.283 | 0.511 | 0.498 |
|  | 0.833 | 0.5 | 0.333 | 0.555 |
| 平均值 | 0.807 | 0.579 | 0.524 |  |

表3-9 全部属性的MDP和TMDP

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 关于相对属性的MDP(Mean Distribution Precision) | | | | | | TMDP |
| 基本属性 |  |  |  |  |  |  |  |
|  |  | 0.741 | 0.854 | 0.535 | 0.498 | 0.555 | 0.6366 |
|  | 0.556 |  | 0.429 | 0.5 | 0.486 | 0.573 | 0.5088 |
|  | 0.854 | 0.627 |  | 0.5 | 0.539 | 0.452 | 0.5944 |
|  | 0.535 | 0.687 | 0.5 |  | 0.7 | 0.479 | 0.5083 |
|  | 0.397 | 0.486 | 0.403 | 0.525 |  | 0.392 | 0.4406 |
|  | 0.479 | 0.757 | 0.451 | 0.479 | 0.522 |  | 0.5376 |

Step3：根据MTMDP值选择最佳的分裂属性。

根据表3-9所示，属性的TMDP值0.6366是所有属性TMDP中的最大值，根据公式(3-10)选择属性作为分裂属性，根据属性进行二分裂。

Step4：选择分裂属性的最佳分裂值。

属性是一个多值属性，选出最优的二分裂属性是一个NP难的问题。MTMDP算法受到分布精度的启发来选择分裂点，降低了计算的复杂性。从前面的分析发现如果一个对象集合有更大值的分布精度，那么这个集合和其他对象的集合之间就具有较小的耦合度，所以有最大值的分布精度的等价类更应该组成一个簇。具体而言，MTMDP算法把具有最大值的平均分布近似精度的属性值的对象作为一个簇，把剩下的对象作为另一个簇。在这个例子中，从表3-8可以发现有最大值的平均分布近似精度值0.807，因此产生了两个叶子结点={3,5,7,8}和={1,2,4,6,9,10}。

Step5：根据结点的Cohesion聚合度选择继续分裂的结点。

根据公式(3-12)分别计算这两个叶子结点的聚合度，再根据聚合度来选择继续聚类的结点。以={1,2,4,6,9,10}结点为例，计算，首先要得到*X*关于属性的等价类，如表3-10所示，不同的颜色表示关于属性不同等价划分。

表3-10 计算关于属性的粒度

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 对象 | 1 | 4 | 6 | 9 | 10 | 2 |
|  | Blue | Blue | Green | Green | Green | Red |
|  |  | |  | | |  |

根据公式(3-11)计算=0.389，接着再计算关于属性和的粒度，最后根据公式(3-12)求出该结点的聚合度。

重复类似的过程，分别计算两个叶子结点的聚合度，计算结果如表3-11所示。选择具有最小聚合度的叶子结点继续进行聚类，结点{1,2,4,6,9,10}有更小的聚合度0.426，所以选择{1,2,4,6,9,10}继续进行分裂。

Step1~Step5这个过程不断地重复，直到叶子结点的数目等于预先设定的聚类数目*k*值，算法结束。

表3-11 计算叶子结点聚合度

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 叶子结点 |  |  |  |  |  |  | *cohesion*(*X*) |
| {3,5,7,8} | 1 | 1 | 0.625 | 0.625 | 0.375 | 0.25 | 0.646 |
| {1,2,4,6,9,10} | 0.5 | 0.389 | 0.5 | 0.5 | 0.5 | 0.167 | 0.426 |

### 3.2.3 MTMDP算法分析

假设在一个分类信息系统中，一共有*n*个对象，*m*个属性，每个属性最多有*l*个取值，*k*是指预先设定的最终簇数目。要产生*k*个聚类就要进行*k-1*次迭代，每次迭代的过程，计算出所有属性的等价类的时间是*nm*，计算平均分布精度MDP值的时间近似是*m*2*ln*2，计算出所有属性TMDP值和MTMDP的时间是。另外MTMDP算法还需要计算*2*(*k-2*)次的聚合度来选择叶子结点进行分裂，计算聚合度的总时间少于*2*(*k-2*)*nm*。综上所述MTMDP算法的时间复杂度是一个多项式*O*(*km*2*ln*2*+kmn*)。

## 3.3 MMR和MTMDP算法的对比分析

表3-12描述了一个分类数据的信息系统，一共有6个对象，两个分类属性。

表3-12 信息系统

|  |  |  |
| --- | --- | --- |
| U | *a*1 | *a*2 |
| 1 | M | F |
| 2 | S | F |
| 3 | S | T |
| 4 | S | T |
| 5 | S | T |
| 6 | H | T |

对于等价类和，相对于属性的精确近似度分别计算如下：



由以上计算可知，所以根据精确近似集合和集合有同样的不确定性。

等价类关于属性的分布近似精度计算如下所示：



以上计算结果可知，使用分布近似精度的结果表明集合比集合有更小的不确定度，使用分布近似精度得到的结果更能符合真实的结果。

对于任何集合和任意属性集*B*，分布近似精度和精确近似度都是不确定的方法。但是分布近似精度是一种更有效的不确定性方法。这个例子说明了分布近似精度是一种比精确近似更有效的不确定性方法。

**定义3.12**[23] 对一个分类数据的信息系统而言，预先给定的*k*值(*k*<|*U*|)最终生成的簇数目，最后的聚类结果是，代表一次聚类产生的簇。聚类的结果可以用平均聚合度来评价，平均聚合度的定义如下：

 (3-13)

考虑表3-12所示的信息系统，如果预先设定最终的聚类数目*k*=3，MMR和MTMDP算法工作过程如下所示：

第一，MMR和MTMDP算法把所有的对象*U*作为分裂的根结点，MMR算法根据MR来选择最佳的分裂属性，MTMDP算法根据TMDP来选择最佳的分裂属性。计算之后，MMR和MTMDP算法都选择了属性作为分裂属性。因此产生了两个叶子结点和。

第二，MMR算法选择对象数目多的叶子结点继续进行聚类（分裂）。因此，选择结点继续进行聚类，最后的聚类结果是图3-2所示的一颗树。然而MTMDP算法选择具有最小聚合度的叶子结点继续进行聚类。叶子结点聚合度的计算过程如下：





根据计算结果可知叶子结点有更小的聚合度，所以选择该结点继续进行聚类，聚类的最后结果如图3-3所示。









图3-2 MMR算法聚类结果

图3-3 MTMDP算法聚类结果









下面我们用平均聚合度（公式(3-13)）来评价一下MMR和MTMDP算法的聚类结果。如图3-2所示，MMR算法的聚类结果是，这三个簇的聚合度计算如下：



平均聚合度计算如下：



如图3-3所示，MTMDP算法的聚类结果是，这三个簇的聚合度计算如下：



这三个聚类的平均聚合度计算如下：



计算结果表明MTMDP算法得到的聚类结果含有较大值的平均聚合度，这说明了MTMDP算法产生了一个更好的聚类结果，MTMDP算法比MMR算法更优。

MTMDP和MMR算法都有相似的优点，都能够处理聚类过程中的不确定问题，都只需要输入最终生成的聚类数目*k*值一个参数，不依赖初始值和输入的顺序，就能够得到一个稳定的聚类结果（重复运行程序，得到相同的聚类结果）。

和MMR算法相比，MTMDP算法从三个方面有所提高[23]：

（1）MTMDP算法采用了一种更能有效反映对象集合的不确定性的分布近似精度指数，和MMR算法中使用的粗糙度不同。

（2）MTMDP算法是根据总平均分布精度来排列候选属性，而不是选择最大值的平均分布精度。合理的分裂属性应该反映在所有属性上，而不是只在某一个属性上效果最佳。

（3）MTMDP算法选择聚合度最小的叶子结点继续进行分裂，这样能够提高聚类结果的准确性，比MMR算法中选择含对象数目多的叶子结点进行分裂更好，MMR算法很容易产生不理想的聚类结果，尤其是某些不平衡的数据集。

## 3.4 基于Spark的MMR和MTMDP并行算法设计

随着云计算概念的推广，为提高聚类效率出现了多种并行处理数据方式，典型的并行方法是将海量数据集无差别地划分为多个数据子集，将数据子集发放到各个子处理机器上并行地对数据进行处理，最后汇合局部数据得出全局结果。相比传统的单机模式，采用多节点并行化处理可以大幅度提高聚类速度，本文以下将介绍如何在Spark平台上实现MMR和MTMDP算法的并行化。

### 3.4.1 Spark中的RDD操作

Spark平台在集群中实现并行化是依靠RDD这种粗粒度的数据集，基于RDD概念的并行化都是数据并行而非任务并行，所以在Spark平台上实现并行，要从整个集群的数据流出发。Spark的并行化首先要基于算法，设计文件从被吸入到RDD空间中到最后脱离RDD空间生成目标文件之间的数据类型转换，因为RDD不支持细粒度的操作例如遍历，所以算法的实现自然要将RDD转化成Scala中常用的集合类型，再以一种细粒度的方式来对这些数据进行操作。要利用Spark编程，首先要学会RDD的基本操作，Spark中RDD的转换和动作如表3-13所示。

### 3.4.2 MMR和MTMDP并行化的基本思想

前面我们已经分析了MMR和MTMDP算法的实现的具体步骤，我们可以发现MMR和MTMDP算法的大致流程是相同的，两种算法的关键步骤：选择分裂的属性和选择继续分裂的结点。这两种算法都是先计算所有属性的等价类，然后根据等价类计算属性之间的粗糙度或分布精度，根据计算结果选择出分裂的属性值进行一次二分裂，然后再根据结点的对象数目或聚合度选择继续分裂的结点。

表3-13 Spark中RDD的变换和动作

|  |  |
| --- | --- |
| 变换 | map(f:T)U):RDD[T])RDD[U] filter(f:T)Bool):RDD[T])RDD[T] flatMap(f:T)Seq[U]):RDD[T])RDD[U] sample(fraction:Float):RDD[T])RDD[T](Deterministic sampling) groupByKey():RDD[(K,V)])RDD[(K,Seq[V])] reduceByKey(f:(V;V))V):RDD[(K,V)])RDD[(K,V)] union():(RDD[T];RDD[T]))RDD[T] join():(RDD[(K,V)];RDD[(K,W)]))RDD[(K,(V,W))] cogroup():(RDD[(K,V)];RDD[(K,W)]))RDD[(K,(Seq[V],Seq[W]))] crossProduct():(RDD[T];RDD[U]))RDD[(T,U)] mapValues(f:V)W):RDD[(K,V)])RDD[(K,W)](Preserves partitioning) sort(c:Comparator[K]):RDD[(K,V)])RDD[(K,V)] partitionBy(p:Partitioner[K]):RDD[(K,V)])RDD[(K,V)] |
| 动作 | count():RDD[T])Long collect():RDD[T])Seq[T] reduce(f:(T;T))T):RDD[T])T lookup(k:K):RDD[(K,V)])Seq[V](On hash/range partitioned RDDs) save(path:String):Outputs RDD to a storage system,e.g.,HDFS |

由算例分析过程我们可以发现，MMR和MTMDP算法在一次迭代的过程中，计算属性的等价类之间并没有互相影响的情况，属性和属性的等价类其实是可以并行计算的。当我们得到全部的属性等价类时再计算属性之间的粗糙度或分布精度，计算属性和属性的粗糙度或分布精度也是可以并行的，属性之间的计算结果并不存在相互依赖的关系，它们之间不会互相更新。MMR和MTMDP并行算法的流程图如图3-4所示。

综上分析，MMR和MTMDP算法中可以并行实现的主要有三个步骤：

（1）并行计算属性的等价类

（2）并行计算属性的平均粗糙度或平均分布精度

（3）并行计算分裂属性的不同取值的平均粗糙度或平均分布精度

图3-4 并行算法的流程图

开始

输入数据集*U*

生成簇数目*k*

初始化*U*为一个簇

初始化CNC=1

(当前簇数目)

CNC< *k*

选择继续分裂的簇

计算属性等价类

（并行）

计算属性的平均粗糙度

或平均分布精度（并行）

选择分裂属性

计算分裂属性值的平均粗糙度

或平均分布精度（并行）

根据分裂属性值进行二分裂

CNC=CNC+1

结束

输出*k*个簇

Y

N

### 3.4.3 基于Spark的MMR和MTMDP并行算法的设计

#### 3.4.3.1 并行计算属性的等价类

并行计算属性等价类的流程图如图3-5所示。

（1）获得分裂的簇，转化为一个RDD对象，MMR算法选择对象数目多的结点继续分裂，MTMDP算法选择聚合度最小的结点继续分裂。

图3-5 并行计算属性等价类

结束

groupBy（属性名）

属性名，

（属性值，id）

属性名，

（属性值，id）

属性名，

（属性值，id）

属性1的全部信息

属性2的全部信息

属性*n*的全部信息

groupBy（属性值）

groupBy（属性值）

groupBy（属性值）

属性1等价类

属性2等价类

属性*n*等价类

开始

flatMap

选择继续分裂的簇

|  |
| --- |
| var curCluster = getSplitCluster(allCluster) |
| var curObjects\_rdd = sc.parallelize(curObjects) |

（2）对每个对象执行一次flatMap操作，得到多个（属性名，（属性值，id））的键值对，便于并行地计算属性的等价类。

|  |
| --- |
| val allattr\_rdd = curObjects\_rdd.flatMap(x => x.attrArray) |

（3）根据属性名对所有的（属性名，（属性值，id））进行划分，得到一个属性的全部信息。

|  |
| --- |
| val attrGroup\_rdd = allattr\_rdd.groupBy(getAttrGroup(\_)) |

（4）对每个关于属性名划分过的元素再根据属性值进行划分

|  |
| --- |
| val attrValGroup\_rdd = attrGroup\_rdd.map  (x =>x.\_2.groupBy(getAttrValGroup(\_))) |

（5）计算出每个属性的等价类，并且缓存到内存中。

|  |
| --- |
| val attrEqual\_rdd = attrValGroup\_rdd.map(getAttrEqual(\_)) |
| attrEqual\_rdd.cache() |

#### 3.4.3.2 并行计算每个属性的平均粗糙度或平均分布精度

（等价类，等价类）

（等价类，等价类）

（等价类，等价类）

（等价类，粗糙度）

（等价类，粗糙度）

（等价类，粗糙度）

（等价类1，最小平均粗糙度）

计算MMR

结束

（等价类2，最小平均粗糙度）

（等价类*n*，最小平均粗糙度）

开始

属性等价类

Cartesian

reduceByKey

图3-6 MMR算法并行计算属性粗糙度

由上面的分析可知attrEqual\_rdd保存了所有属性的等价类，该RDD中的一个元素，就是一个属性的等价类。MMR算法并行计算属性粗糙度的流程图如图3-6所示。

（1）过滤掉只有一个属性值的属性。如果一个属性只有一个取值，那么这个属性不可能作为分裂属性，因为单值属性不能进行二分裂。

|  |
| --- |
| attrEqual\_rdd2= attrEqual\_rdd.filter(deletOneValAttr) |

（2）生成（属性等价类，属性等价类）等价类键值对。过滤了单值属性后，要计算计算两个属性之间的平均粗糙度，首先要得到这个属性与其它所有属性的键值对，作为函数getAttrOneRoughness和getAttrOneMDP的输入参数。

|  |
| --- |
| val twoAttrEqual\_rdd = attrEqual\_rdd2.cartesian(attrEqual\_rdd2) |

（3）并行计算每一组等价类键值对的平均粗糙度或平均分布精度。

|  |
| --- |
| val attrOneRough\_rdd = twoAttrEqual\_rdd.map  (x => getAttrOneRoughness(x.\_1, x.\_2)) |
| val attrOneMDP = twoAttrEqual\_rdd.map  (x => getAttrOneMDP(x.\_1, x.\_2)) |

（4）MMR算法计算每个属性的MR最小平均粗糙度值，MTMDP算法计算每个属性的TMDP总平均分布精度值。

|  |
| --- |
| val minRoughness\_rdd = attrOneRough\_rdd.reduceByKey(getMR(\_, \_)) |
| val totalMDP\_rdd = attrOneMDP\_rdd.reduceByKey( (\_+\_)) |

（5）MMR算法中计算所有属性中最小的MR值，选择出最佳的分裂属性。在RDD中计算min(key, value)时是选择出具有最小key值的元素，在MMR算法中要选择具有最小MR值的属性，所以要将RDD中的(key, value)互换，再求出具有最小MR值的分裂属性。MTMDP算法中选择具有最大TMDP值的属性作为分裂属性。

|  |
| --- |
| val mmr\_rdd = minRoughness\_rdd.map(x => (x.\_2, x.\_1)) |
| var SplitAttr= mmr\_rdd.min().\_2 |
| val mtmdp\_rdd = totalMDP\_rdd.map(x => (x.\_2, x.\_1))  var SplitAttr= mtmdp\_rdd.min().\_2 |

#### 3.4.3.3并行计算分裂属性值的总平均粗糙度或总平均分布精度

（1）生成（属性值，等价类）键值对，便于并行地计算属性值的粗糙度或平均分布精度。

|  |
| --- |
| val twoSplitAttrVal\_rdd = splitAttrVal\_rdd.cartesian(attrEqual\_rdd) |

（2）对每一个（属性值，等价类）执行一个map操作，计算属性值关于该等价类的粗糙度或平均分布精度。

|  |
| --- |
| val splitAttrValOneRough\_rdd = twoSplitAttrVal\_rdd.map  (x => (x.\_1.attrVal, getAttrValOneRoughness(x.\_1, x.\_2))) |
| val splitAttrValOneMDP\_rdd = twoSplitAttrVal\_rdd.map  (x => (x.\_1.attrVal, getAttrValMDP(x.\_1, x.\_2))) |

（3）对所有的（属性值，粗糙度）和（属性值，平均分布精度）键值对，进行reduceByKey操作，根据属性值进行reduce操作，得到该属性值对应的总平均粗糙度或总平均分布精度。

图3-7 MMR算法并行计算分裂属性值的粗糙度

（属性值，等价类）

（属性值，等价类）

（属性值，等价类）

（属性值，粗糙度）

（属性值，粗糙度）

（属性值，粗糙度）

（属性值，平均粗糙度）

计算分裂属性值

结束

（属性值，平均粗糙度）

（属性值，平均粗糙度）

开始

分裂属性等价类

Cartesian

reduceByKey

|  |
| --- |
| val splitAttrValAllRough\_rdd = splitAttrValOneRough\_rdd.reduceByKey  (\_ + \_).map(x=>(x.\_2,x.\_1)) |
| val splitAttrValAllMDP\_rdd = splitAttrValOneMDP\_rdd.reduceByKey  (\_ + \_).map(x=>(x.\_2,x.\_1)) |

（4）MMR算法中计算分裂属性值中最小的MR值，选择出最佳的分裂属性值。MTMDP算法中选择具有最大TMDP值的属性值作为分裂属性。

|  |
| --- |
| val splitAttrVal = splitAttrValAllRough\_rdd.min().\_2 |
| val splitAttrVal = splitAttrValAllMDP\_rdd.max().\_2 |

## 3.5 本章小结

本章主要有四个部分。第一部分介绍了MMR层次聚类算法的思路，并且以一个实际的算例分析了MMR层次聚类算法一次迭代过程，最后分析了MMR层次聚类算法的时间复杂度。第二部分介绍了MTMDP层次聚类算法的思路，并且以一个实际的算例分析了MTMDP层次聚类算法一次迭代过程，最后分析了MTMDP层次聚类算法的时间复杂度。第三部分分析了MMR和MTMDP两种算法的优缺点，并且对两种算法进行了对比分析。第四部分介绍了MMR和MTMDP算法在Spark平台上的并行化设计。