# 数据挖掘入门

## 什么是数据挖掘

数据挖掘（Data Mining），也叫数据开采、数据采掘等，是从大量的、不完全的、有噪声的、模糊的、随机的实际应用数据中，自动提取隐含在其中的、人们事先不知道的，但又是潜在有用的信息的过程。这些信息的表现形式为规则、概念、规律及模式等。

https://blog.csdn.net/wzy0623/article/list/4

从上述定义可见数据挖掘明显有别于传统数据处理技术（如事务处理，OLTP）。首先数据挖掘面对的是大量的不完全的数据，所谓不完全指的是一个数据集合中的样本或观测数据。数据挖掘就是要通过对已知样本的分析发现规律，使用算法建立模型，再用模型预测未来的样本外数据的观测结果。其次输入的观测数据是有噪声的、模糊的和随机的。这里的随机指数据样本的选取方式，噪声简单说就是数据集合中无法解释的随机数据误差，即一些和其它数据不一致的数据。数据挖掘要能够在随机数据样本中识别和消除噪声，仅利用有助于实现挖掘目标的数据，从而提高模型的准确性。最后数据挖掘需要从数据中发现先前未知的有用信息，并且这个发现过程是自动的。

企业里的数据量非常大，而其中真正有价值的信息却很少，因此从大量数据中经过深层分析，获得有利于商业运作、提高竞争力的信息，就像从矿石中淘金一样，数据挖掘也因此得名。这种新式的商业信息处理技术，可以按商业既定业务目标，对大量数据进行探索和分析，揭示隐藏的、未知的或验证已知的规律性，并进一步将其模型化。在较浅的层次上，它利用现有数据库管理系统的查询、检索及报表功能，与多维分析、统计分析方法相结合，进行联机分析处理（OLAP），从而得出可供决策参考的统计分析数据。在深层次上，则从数据中发现前所未有的、隐含的知识。OLAP的出现早于数据挖掘，它们都是从数据中抽取有用信息的方法，就决策支持的需要而言两者是相辅相成的。OLAP可以看作是一种广义的数据挖掘方法，它旨在简化和支持联机分析，而数据挖掘的目的是使这一过程尽可能自动化。

并非所有的信息发现都被视为数据挖掘。例如，使用数据库管理系统查找个别的记录，或通过互联网搜索引擎查找特定的Web页面，则是信息检索（information retrieval）领域的任务。虽然这些任务非常重要，可能涉及使用复杂的算法和数据结构，但是它们主要依赖传统的计算机科学技术和数据的明显特征来创建索引结构，从而有效地组织和检索信息。下面列举一些属于或不属于数据挖掘任务的活动，帮助加深对数据挖掘概念的理解。

* 根据性别划分公司的顾客。这是简单的数据库查询，不是数据挖掘的任务。
* 根据消费金额划分公司的顾客。这是根据事先定义的阈值进行计算并对顾客已有消费行为进行分类，不属于数据挖掘。然而如果要预测顾客未来的消费金额，则属于数据挖据范畴。
* 计算公司的总销售额。这只是简单汇总，不需要数据挖掘。
* 按学号对学生排序。这是数据库查询，不属于数据挖掘。
* 预测掷骰子的结果。由于出现每种结果的概率是相等的，这是一个典型的概率计算问题，而不是数据挖掘问题。
* 使用历史记录预测某公司未来的股票价格。我们需要建立一个模型预测股票的后续价格。这是一个数据挖掘中预测模型的例子。
* 监视病人心率的异常变化。我们要以正常心率建模，并以此为依据，当出现异常心率时发出警告。这是一个数据挖掘中异常检测的例子。如果将正常和异常分为两类，本例也可看作是一个数据挖据的分类问题。
* 提取声波频率。这是信号处理问题，不属于数据挖掘。

## 数据挖掘与知识发现

数据挖掘起始于20世纪下半叶，是在当时多个学科的基础上发展起来的。随着数据库技术的广泛应用，数据的积累不断膨胀，导致简单的查询和统计已经无法满足企业的商业需求，急需一些革命性的技术去挖掘数据背后的信息。同时，这期间计算机领域的人工智能（Artificial Intelligence）也取得了巨大进展，进入了机器学习的阶段。因此，人们将两者结合起来，用数据库管理系统存储数据，用计算机分析数据，并且尝试挖掘数据背后的信息。这两者的结合促生了一门新的学科，即数据库中的知识发现（Knowledge Discovery in Databases，KDD）。1989年8月召开的第11届国际人工智能联合会议的专题讨论会上首次出现了知识发现这个术语，到目前为止，KDD的重点已经从发现方法转向了实践应用。数据挖掘则是知识发现（KDD）的核心部分。进入21世纪，数据挖掘已经成为一门比较成熟的学科，并且数据挖掘技术也伴随着信息技术的发展日益成熟起来。

KDD是将未加工的数据转换为有用信息的整个过程，包括一系列转换步骤，从数据的预处理到数据挖掘结果的后处理。数据挖掘在由数据转化为知识的过程中，所处的位置如图1所示。

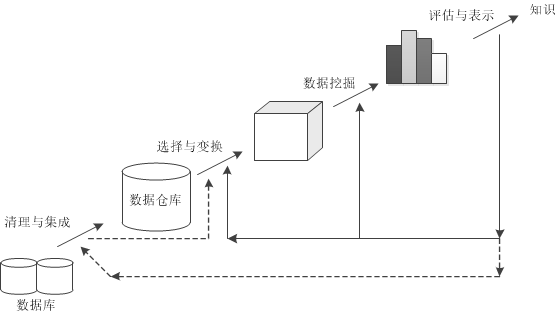


图1 数据转化为知识的过程

输入数据以各种形式存储，如平面文件、电子表格或数据库表等，基于的数据库类型主要有关系型数据库、面向对象数据库、事务数据库、演绎数据库、时态数据库、多媒体数据库、空间数据库及数据仓库（Data Warehouse）等。数据可以驻留在集中的数据存储库中，也可以分布在多个节点上。数据预处理（preprocessing）的目的是将未加工的输入数据转换成适合分析的形式。数据预处理的步骤包括融合来自多个数据源的数据，清洗数据以消除噪声和重复的观测值，选择与当前数据挖掘任务相关的记录和特征。由于收集和存储数据的方式多种多样，数据预处理可能是整个知识发现过程中最费力、最耗时的步骤。

数据挖掘后获得的知识包括关联规则、特征规则、区分规则、分类规则、总结规则、偏差规则、聚类规则、模式分析及趋势分析等。数据挖掘是一门交叉学科，它把人们对数据的应用从低层次的简单查询，提升到从数据中挖掘知识，提供决策支持。例如，在商业应用中，数据挖掘的结果所揭示的规律可以结合商业活动管理工具，从而开展或测试有效的商业促销活动。这样的结合需要后处理（postprocessing）步骤，确保只将那些有效的和有用的结果集成到决策支持系统中。后处理的一个例子是可视化，它使得数据分析者可以从各种不同的视角探查数据挖掘的结果。在后处理阶段，还能使用统计度量或假设检验，删除虚假的数据挖掘结果。

数据挖掘与传统的数据分析（如查询、报表、联机应用分析）的本质区别是数据挖掘是在没有明确假设的前提下去挖掘信息、发现知识，包括不公开的数据。数据挖掘使数据库技术进入了一个更高级的阶段。数据挖掘是要发现那些不能靠直觉发现的信息或知识，甚至是违背直觉的信息或知识，挖掘出的信息越是出乎意料，就可能越有价值。能够比市场提前知道这种信息，提前做出决策就有机会获得更多利润。

        相对于传统数据分析，概括来说，数据挖掘技术具有以下几个特点：

* 处理的数据规模十分庞大，达到GB、TB数量级，甚至更大。
* 查询一般是决策制定者（用户）提出的即时随机查询，往往不能形成精确的查询要求，需要靠系统本身寻找其可能感兴趣的东西。
* 在一些应用（如商业投资等）中，由于数据变化迅速，因此要求数据挖掘能快速做出相应反应以随时提供决策支持。
* 数据挖掘中，规则的发现基于统计规律。因此，所发现的规则不必适用于所有数据，而是当达到某一临界值时，即认为有效。因此，利用数据挖掘技术可能会发现大量的规则。
* 数据挖掘所发现的规则是动态的，它只反映了当前状态的数据具有的规则，随着新数据的不断加入，需要随时对其进行更新。

## 数据挖掘的原理

数据本身只是数据，直观上并没有表现出任何有价值的知识。当我们用数据挖掘方法，从数据中挖掘出知识后，这种知识是否值得信赖呢？为了说明这种知识是可信的，有必要了解一下数据挖掘的原理。

数据挖掘其实质是综合应用各种学科，对于业务相关的数据进行一系列科学的处理，这个过程中需要用到数据库、统计学、应用数学、机器学习、模式识别、数据可视化、信息科学、程序开发等多个领域的理论和技术，如图2所示。其核心是利用算法对处理好的输入和输出数据进行训练，并得到模型。然后再对模型进行验证，使得模型能够在一定程度上刻画出数据由输入到输出的关系。最终利用该模型，对新输入的数据进行计算，从而得到新的输出，对这个输出就可以进行解释和应用了。所以这种模型虽然不容易解释或很难看到，但它是基于大量数据训练并经过验证的，因此能够反映输入数据和输出数据之间的大致关系，这种关系也即模型，就是我们需要的知识。可以说，这就是数据挖掘的原理，从中可以看出，数据挖据是有一定科学依据的，这样挖掘的结果也是值得信任的。

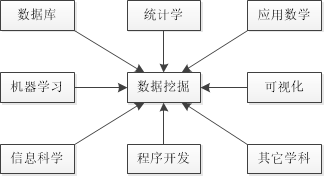


图2 数据挖据与其它学科的关系

## 数据挖掘的任务

通常可将数据挖掘任务分为预测和描述两大类。预测任务的目标是根据其它属性的值，预测特定属性的值。被预测的属性一般称为目标变量（target variable）或因变量（dependent variable），而用来做预测的属性称为说明变量（explanatory）或自变量（independent variable）。描述任务的目标是导出概括数据中潜在联系的模式（关联、趋势、聚类、轨迹和异常）。本质上，描述性数据挖掘任务大都是探查性的，并且常常需要后处理技术验证和解释结果。

对学术研究和产业应用进行归纳，会发现数据挖掘的任务总是集中在几个方面，即回归、分类、预测、关联、聚类、异常检测六个方面，前三个属于预测性任务，后三个属于描述性任务。它们不仅在挖掘的目标和内容上不同，所使用的技术也差别较大，所以通常也将数据挖掘的技术按照这六个方面分类。下面将逐一简要介绍这六类数据挖掘任务及相应的算法。

### 回归（regression）

回归是确定两种或两种以上变量间相互定量关系的一种统计分析方法。回归是数据挖掘中最为基础的方法，也是应用领域和应用场景最多的方法，只要是量化型问题，我们一般都会先尝试用回归方法来研究或分析。比如要研究某地区钢材消费量与国民收入的关系，那么就可以直接用这两个变量的数据进行回归，然后看它们之间的关系是否符合某种形式的回归关系。回归可用于预测连续的目标变量。

根据回归方法中因变量的个数和回归函数的类型（线性或非线性）可将回归方法分为以下几种：一元线性、一元非线性、多元线性、多元非线性。另外还有两种特殊的回归方式，一种是在回归过程中可以调整变量数的回归方法，称为逐步回归，另一种是针对因变量为定性变量（如0、1二值）的回归方法，称为Logisitic回归。

### 分类（classification）

分类是一个常见的问题，在我们的日常生活中就会经常遇到分类问题，比如垃圾分类。数据挖掘中的分类也是最为常见的问题，其典型的应用是根据事物在数据层面表现的特征，对事物进行科学的分类。分类可用于预测离散的目标变量。

对于分类问题，人们已经研究并总结出了很多有效的方法。到目前为止，已经研究出的分类方法包括：决策树方法（经典的决策树算法主要包括ID3算法、C4.5算法和CART算法等）、神经网络方法、贝叶斯分类、K-近邻算法、判别分析、支持向量机等分类方法。不同的分类方法有不同的特点。这些分类方法在很多领域都得到了成功的应用，比如决策树方法已经成功地应用到医学诊断、贷款风险评估等领域；神经网络则因为对噪声数据有很好的承受能力而在实际问题中得到了非常成功的应用，比如识别手写字符、语音识别和人脸识别等。但是由于每一种方法都有缺陷，再加上实际问题的复杂性和数据的多样性，使得无论哪一种方法都只能解决某一类问题。近年来，随着人工智能、机器学习、模式识别和数据挖掘等领域中传统方法的不断发展以及各种新方法和新技术的不断涌现，分类方法得到了长足的发展。

### 预测（forecasting）

预测是预计未来事件的一门科学，它包含采集历史数据并用某种数学模型来预测未来，它也可以是对未来的主观或直觉的预期，还可以是上述的综合。在数据挖掘中，预测是基于既有数据进行的，即以现有数据为基础，对未来的数据进行预测。

预测的重要意义就在于它能够在自觉地认识客观规律的基础上，借助大量的信息资料和现代化的计算手段，比较准确地揭示出客观事物运行中的本质联系及发展趋势，预见到可能出现的种种情况，勾画出未来事物发展的基本轮廓，提出各种可以互相替代的发展方案，这样就使人们具有了战略眼光，使得决策有了充分的科学依据。

预测方法有很多，可以分为定性预测方法和定量预测方法。从数据挖掘角度，我们用的方法显然属于定量预测方法。定量预测方法又可分为时间序列分析和因果关系分析两类，其中常用的时间序列分析法有移动平均（ARIMA）、指数平滑等，因果关系分析法有回归方法、计量经济模型、神经网络预测法、灰色预测法、马尔科夫预测法等。

### 关联（association）

关联分析用来发现描述数据中强关联特征的模式。所发现的模式通常用蕴涵规则或特征子集的形式表示。由于搜索空间是指数规模的，关联分析的目标是以有效的方式提取最有趣的模式。购物篮分析就是一种典型的关联分析。“尿布与啤酒”的故事大家都听过，它们之间属于关联关系，是通过对交易信息进行关联挖掘而得到的，这种类型的规则可以用来发现各类商品中可能存在的交叉销售的商机。

数据关联是一类重要的可被发现的知识。若两个或多个变量的取值之间存在某种规律性，就称为关联。关联可分为简单关联、时序关联、因果关联。关联分析的目的是找出数据之间隐藏的关联关系。关联分析生成的规则带有可信度，通过可信度来描述这种关系的确定程度。

关联规则挖掘就是要发现数据中项集之间存在的关联关系或相关关系。按照不同情况，关联挖掘可以分为以下几种情况：

（1）基于规则中处理的变量的类别，关联规则可以分为布尔型和数值型。

布尔型关联规则处理的值都是离散的、种类化的，它显示了这些变量之间的关系；而数值型关联规则可以和多维关联或多层关联规则结合起来，对数值型字段进行处理，将其进行动态地分隔，或者直接对原始的数据进行处理，当然数值型规则中也可以包含种类变量。例如：性别 = “女” => 职业 = “秘书”，是布尔型关联规则；性别 = “女” => avg（收入）=2300，涉及的收入是数值类型，所以是一个数值型关联规则。

（2）基于规则中数据的抽象层次，可以分为单层关联规则和多层关联规则。

在单层的关联规则中，所有的变量都没有考虑到现实的数据是具有多个不同层次的；而在多层关联规则中，对数据的多层性进行了充分的考虑。例如：IBM台式机 => Sony打印机，是一个细节数据上的单层关联规则；台式机 => Sony打印机，是一个较高层次和细节层次之间的多层关联规则。

（3）基于规则中涉及的数据的维数，关联规则可以分为单维的多维的。

在单维的关联规则中，我们只涉及数据的一个维，如用户购买的物品；而在多维的关联规则中，要处理的数据将会涉及多个维。换言之，单维关联规则是处理单个属性中的一些关系；多维关联规则是处理各个属性之间的某些关系。例如：啤酒 => 尿布，这条规则只涉及用户购买的物品；性别 = “女” => 职业 = “秘书”，这条规则就涉及两个字段的信息，是两个维上的一条关联规则。

常用的关联算法有Apriori、FP-tree、HotSpot等。

### 聚类（cluster）

聚类分析又称群分析，是根据“[物以类聚](https://www.baidu.com/s?wd=%E7%89%A9%E4%BB%A5%E7%B1%BB%E8%81%9A&tn=24004469_oem_dg)”的道理，对样品进行分类的一种多元统计分析方法。它讨论的对象是大量的样品，要求能合理地按各自特性来进行合理的分类，没有任何模式可供参考或依循，是在没有先验知识的情况下进行的。聚类是将数据分类到不同的类或者簇的过程，旨在发现紧密相关的观测对象群组，使得与属于不同簇的对象相比，属于同一簇的对象相互之间尽可能类似。

聚类分析起源于分类学，在古老的分类学中，人们主要依靠经验和专业知识来实现分类，很少利用数学工具进行定量的分类。随着人类科学技术的发展，对分类的要求越来越高，以至有时仅凭经验和专业知识难以确切地进行分类，于是人们逐渐地把数学工具引用到了分类学中，形成了数值分类学，之后又将多元分析的技术引入到数值分类学形成了聚类分析。更直接地说，聚类是看样品大致分成几类，然后再对样品进行分类，也就是说，聚类是为了更合理地分类。

在不同应用领域，很多聚类技术都得到了发展，这些技术方法被用作描述数据，衡量不同数据源间的相似性，以及把数据源分类到不同的簇中。在商业上，聚类分析被用来发现不同的客户群，并且通过购买模式刻画不同客户群的特征；在生物上，聚类分析被用来进行动植物分类和对基因进行分类，获取对种群固有结构的认识；在保险行业上，聚类分析通过一个高的平均消费来鉴定汽车保险单持有者的分组，同时根据住宅类型、价值、地理位置来鉴定一个城市的房产分组；在因特网应用上，聚类分析被用来在网上进行文档归类。

聚类问题的研究已经有很长的历史。迄今为止，为了解决各领域的聚类应用，已经提出的聚类算法有近百种。根据聚类原理，可将聚类算法分为以下几种：划分聚类、层次聚类、基于密度的聚类、基于网络的聚类和基于模型的聚类。虽然聚类的方法很多，在实践中用得比较多的还是K-means、层次聚类、神经网络聚类、模糊C-均值聚类、高斯聚类等几种常用的方法。

### 异常检测（anomalydetection）

异常检测的任务是识别其特征显著不同于其它数据的观测对象。这样的观测对象称为异常点（anomaly）或离群点（outlier）。离群点是不符合一般数据模型的点，它们与数据的其它部分不同或不一致。离群点可能是度量或执行错误所导致的。例如，一个人的年龄为-999可能是由于对年龄的缺省设置所产生的。离群点也可能是固有数据可变性的结果，例如，一个公司的首席执行官的工资远远高于公司其他雇员的工资，成为一个离群点。异常检测算法的目标是发现正真的异常点，而避免错误地将正常的对象标注为异常点。换言之，一个好的异常检测器必须具有高检测率和低误报率。

许多数据挖掘算法试图使离群点的影响最小化，或者排除它们。但是这可能导致重要的隐藏信息丢失，因为有时离群点本身可能是非常重要的，例如，信用卡公司记录每个持卡人所做的交易，同时也记录信用限度、年龄、年薪和地址等个人信息。由于与合法交易相比，欺诈行为的数目相对较少，因此异常检测技术可以用来构造用户的合法交易的轮廓。当一个新的交易达到时就与之比较，如果该交易的特性与先前所构造的轮廓很不相同，就把交易标记为可能是欺诈。这样，离群点探测和分析就是一个有趣的数据挖掘任务。

离群点检测有着广泛的应用。像上面所提到的，它能用于欺诈监测，如探测不寻常的信用卡使用或电信服务。此外，它在市场分析中可用于确定极低或极高收入的客户的消费行为，或者在医疗分析中用于发现对多种治疗方式的不寻常的反应。

目前，人们已经提出了大量关于离群点检测的算法。这些算法大致可以分为以下几类：基于统计学或模型的方法、基于距离或邻近度的方法、基于偏差的方法、基于密度的方法和基于聚类的方法，这些方法一般称为经典的离群点检测方法。近年来，有不少学者从关联规则、模糊集和人工智能等其它方面提出了一些新的离群点检测算法，比较典型的有基于关联的方法、基于模糊集的方法、基于人工神经网络的方法、基于遗传算法或克隆选择的方法等。

## 数据挖掘的过程

数据挖掘的过程是数据挖掘项目实施的方法论。数据挖掘能够从一堆杂乱的数据中挖掘出有价值的知识，但也需要一个过程。很多数据挖掘工具的厂商都对这个过程进行了抽象和定义，使之更加清晰。比如，SAS将数据挖掘过程划分为五个阶段：抽样（Sample）、探索（Explore）、处理（Manipulate）、建模（Model）、评估（Assess），即所谓的SEMMA过程模型；SPSS则提出了5A模型，即评估（Assess）、访问（Access）、分析（Analyze）、行动（Act）、自动化（Automate）。但对于商业项目，业界普遍采用CRISP-DM（Cross-Industry Standard Process for Data Mining）过程，所谓的“跨行业数据挖掘过程标准”，或者在其基础上改进的过程。CRISP-DM模型为一个KDD工程提供了完整的过程描述。一个数据挖掘项目的生命周期包含六个阶段：业务理解（Business Understanding）、数据理解（DataUnderstanding）、数据准备（Data Preparation）、建模（Modeling）、评估（Evaluation）、部署（Deployment）。

纵观这几个过程模型，大家会发现，其实质是一致的，所以也不必在意到底该用哪个数据挖掘流程，适合自己的就好。但从便于理解和操作的角度，可将数据挖掘过程描述为：1.挖掘目标的定义；2.数据的准备；3.数据的探索；4.模型的建立；5.模型的评估；6.模型的部署，简称为DPEMED（Definition、Preparation、Explore、Modeling、Evaluation、Deployment）模型，它们之间的关系如图3所示。

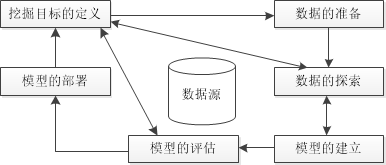


图3 数据挖掘过程示意图

### 挖掘目标的定义

企业或组织机构当想要实施数据挖掘时，十有八九是因为觉得积累的业务数据里有些有价值的东西，也就是说在潜意识里面已经有了大致的目标了。这种目标在无形之中会给随后的数据挖掘过程给出明确的目标，所谓有的放矢，这样数据挖掘就可以有意义地进行下去。因此，实施数据挖掘的第一步要确定数据挖掘的目标。

但要确定目标，就必须要了解数据和相关业务。比如，要分析电信领域的客户呼叫行为，需要了解电信的业务构成、业务运营以及其他诸多的行业知识。有关业务问题，指的是在业务过程中需要解决的问题、想要知道的答案并且认为这些问题的答案蕴藏在大量的数据中，但并不知道它们在哪里。可能涉及的业务问题很多，从数据挖掘的角度，所需要了解的业务问题至少包含以下三个方面：

（1）有关需要解决问题的明确定义；

（2）对有关数据的理解；

（3）数据挖掘结果对业务作用效力的预测。

如果无法确定哪些问题可用数据挖掘解决，一个好的方法是看它们的成功案例，不论所在行业是否相同。许多业务和研究的领域都被证实是数据挖掘能够得以成功应用的领域。它们包括金融服务、银行、保险、电信、零售、制造业、生物、化工等。

当对业务和数据有了一定的了解之后，就可以很容易地定义挖掘的目标，一般可以从以下两个方面定义数据挖掘的目标：

（1）数据挖掘需要解决的问题；

（2）数据挖掘完成后达到的效果，最好给出关键的评估参数及数值，比如数据挖掘结果在3个月内使得整体收益提高5个百分点。

### 数据的准备

数据的准备是数据挖掘中耗时最多的环节，因为数据挖掘的基础就是数据，所以足够、丰富、高质量的数据对数据挖掘的结果至关重要。数据的准备包括数据的选择、数据的质量分析和数据的预处理三个环节。

（1）数据的选择

选择数据就是从数据源中搜索所有与业务对象有关的内部和外部数据信息，并从中选择出适用于数据挖掘应用的数据。内部数据通常指的是现有数据，例如交易数据、调查数据、Web日志等。外部数据通常指需要购买的一些数据，比如股票实时交易数据。

从选择的数据类型来看，在大多数商业应用中都会包括交易数据、关系数据、人口统计数据三种类型的数据。交易数据是业务对象发生业务时产生的操作数据。它们一般有明显的时间和顺序特征，与业务发生有关联，如投资人的证券交易、客户的购物、电话的通话等。关系数据则是相对较少变化的数据，表达了客户、机构、业务之间的关系，如投资人与交易所，客户与电信公司等。人口统计数据表达与业务主题相关的描述信息，这些数据可能来自外部的数据源。这三种数据类型反映了三种数据信息，在数据挖掘的过程中，对知识的发现非常重要，所以选择数据的时候尽量包括业务相关的这三种类型的数据。

（2）数据的质量分析

数据几乎没有完美的。事实上，大多数数据都包含代码错误、缺失值或其它类型的不一致现象。一种避免可能出现缺陷的方法是在建模前对可用数据进行全面的质量分析。数据质量分析的目的是评估数据质量，同时为随后的数据预处理提供参考。数据的质量分析通常包括以下几个方面的内容：

缺失数据：包括空值或编码为无业务处理的缺省值（例如$null$、?或999）。

数据错误：通常是在输入数据时造成的人为错误。

度量标准错误：包括正确输入但却基于不正确的度量方法的数据。

编码不一致：通常包含非标准度量单位或不一致的值，例如同时使用M和male表示性别。

无效的元数据：包含字段的数据含义和字段名称或定义中陈述的意思不匹配。

（3）数据的预处理

经过数据质量分析往往会发现，数据总是存在这样或那样的问题，为了得到准确性、完整性和一致性较好的数据，必须要对数据进行预处理。根据数据质量的不同，数据预处理所用的技术也会有所不同，但通常会包括数据清洗、数据集成、数据规约和数据变换四个步骤，这四个步骤的作用效果如图4所示。

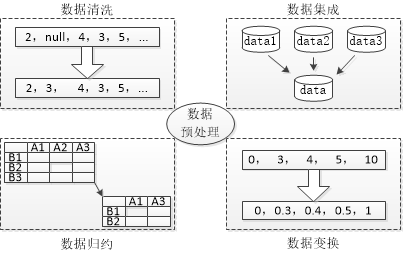


图4 数据预处理的内容

### 数据的探索

探索数据是对数据进行初步研究，以便更好地了解数据的特征，为建模的变量选择和算法选择提供依据。在数据挖掘的过程中，数据的准备和数据的探索是个双向的过程，也就是说，数据探索的结果也可以指导数据的准备，尤其是数据的预处理。更具体地说，在数据挖掘过程中，先进行数据的准备，包括收集、质量分析和预处理，然后进行数据的探索，如果在探索阶段发现数据量太少或数据质量不好或者区分度不好，那么就会返回数据的准备，重新进行数据的收集、质量分析和预处理，通常是直接返回到预处理环节，如对数据进行归一化等预处理操作。然后继续对数据进行探索，直到通过探索对数据比较满意为止，这样就可以转入到下个阶段了。

从广义上说，很少或没有得到理论支撑的数据分析均可以视为数据探索的范畴。数据探索更多的是对数据进行初步分析，有助于针对不同类型的数据或条件进行统计分析的一种技术。数据探索或探索性数据分析具有启发式、开放式等特点。

启发式在于，我们可能对数据的类型或特点知之甚少，需要通过统计技术来探索数据内部的东西，就是通常我们所说的“让数据说话”。这时一般是由于某种原因我们可能对数据背后的理论信息掌握得很少，或缺少这方面的资料等。

开放式在于数据探索以数据清理为先导。数据清理工作往往要参考学科背景知识，例如对缺失值的处理，如果该学科数据对异常值的反应很灵敏，这时如果使用均值去填补，可能会丢失大量的信息（假如缺失值很多）。所以如果仅仅是数据探索，则很少需要考虑上述情况，可以完全根据数据特点来选择相应的处理方法，开放性也体现于此。

下面从几个大的方向上来了解数据探索的方法：

（1）描述统计

描述统计包括均值、频率、众数、中位数、极差、方差和百分位数等，一般来说描述统计均可以用来探索数据结构，它们均用于探索数据的不同属性。

（2）数据可视化

数据可视化也是数据探索阶段常用的一种技术，这种技术概括起来就是将数据的总体特点以图形的方式呈现，用以发现其中的模式。并可以根据一定的规则（标准差、百分数等信息）去拆分、合并等进一步的处理。

毫无疑问图形简明易懂，很多难以表达的情况使用图表顿时使问题变得简单，这也许就是所谓的一图胜千言，这在数据探索中起到很重要的作用，比如常用的频次图、散点图、箱体图等。

（3）数据探索的建模活动

一切可以用于建模的统计方法或计量模型均可以用于数据探索，不过模型之所以是模型，是因为其背后的理论或学科性质的支撑，所以从这层意义上说，数据探索更多是为分析人员提供感性的认识，所有的结果都有待于理论的验证，而只有在认识的边缘，理论才渐渐被淡化。

### 模型的建立

模型的建立是数据挖掘的核心，在这一步要确定具体的数据挖掘模型（算法），并用这个模型原型训练出模型的参数，得到具体的模型形式。模型建立的操作流程如图5所示，在这一过程中，数据挖掘模型的选择往往是很直观的，例如对股票进行分类，则要选择分类模型。问题是分类模型又有多种模型（算法），这时就需要根据数据特征、挖掘经验、算法适应性等方面确定较为合适的算法，如果很难或不便选择哪种具体的算法，不妨对可能的算法都进行尝试，然后从中选择最佳的算法。

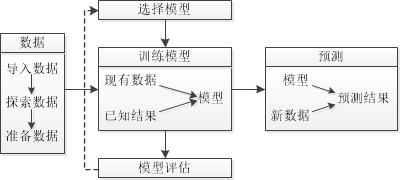


图5 模型建立的流程

数据挖掘的主要内容就是研究模型建立过程中可能用到的各种模型和算法，即关联、回归、分类、聚类、预测和异常检测六大类模型。如果从实现的角度，根据各种模型在实现过程中的人工监督（干预）程序，这些模型又可分为有监督模型和无监督模型。数据挖掘过程中，常用的模型结构如图6所示，根据这一结构，我们可以很清晰地知道模型建立过程中可供选择的模型。

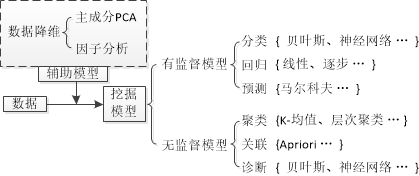


图6 数据挖掘常用的模型（算法）结构图

此处提到模型也提到算法，这两个概念容易混淆。一谈到算法就会想到通过历史数据建立模型，其实数据挖掘算法是创建挖掘模型的机制，对产生的最终挖掘输出结果有很大的决定性。随着数据挖掘新技术的层出不穷和商业数据挖掘产品的成熟与完善，对同一商业问题，通常在产品中有多种算法可供选择，而为特定的任务选择正确的算很有挑战性。可以使用不同的算法来执行同样的业务任务，每个算法会生成不同的结果。而且算法可以进行复合使用，在一个数据挖掘解决方案中可以使用一些算法来探析数据，而使用其它算法基于该数据预测特定结果。例如，可以使用聚类分析算法来识别模式，将数据细分成多少有点相似的组，然后使用分组结果来创建更好的决策树模型。也可以在一个解决方案中使用多个算法来执行不同的任务。例如，使用回归树算法来获取财务预测信息，使用基于规则的算法来执行购物篮分析。由此看出数据挖掘项目中，在明确挖掘目的和了解各种算法特点后，如何正确选择使用算法，得到期望的结果才是关键环节。

在模型建立这一环节，还有一项重要的工作是设置数据的训练集和测试集，训练集的数据用于训练模型，而测试集的数据则用于验证模型。因为这个环节的模型验证是在模型的训练过程中进行的验证，所以这部分模型的验证工作一般也认为隶属于模型的建立过程。为了保证得到的模型具有较好的准确度和健壮性，需要先用一部分数据建立模型，然后再用剩下的数据来测试得到的模型。有时还需要第三个数据集，称为验证集。因为测试集可能受模型特性的影响，还需要一个独立的数据集来验证模型的准确性。

训练和测试数据挖掘模型至少要把数据分成两个部分：一个用于模型训练，另一个用于模型测试。如果使用相同的训练和测试集，那么模型的准确度就很难使人信服。用训练集把模型建立出来之后，可以先在测试集数据上做实验，此模型在测试集上的预测准确度就是一个很好的指导数据，它表示将来与数据集合测试集类似的数据用此模型预测时正确的百分比。但并不能保证模型的正确性，它只是说明在相似数据集合的情况下用此模型会得出相似的结果。

常用的验证方法包括简单验证、交叉验证和N-维交叉验证。

（1）简单验证

简单验证是最基本的测试方法。它从原始数据集合中拿出一定百分比的数据作为测试数据，这个百分比在5%~33%。注意：在把数据集合分成几部分时，一定要保证选择的随机性，这样才能使分开的各部分数据的性质是一致的。先用数据集合的主体把模型建立起来，然后用此模型来预测测试集中的数据。出现错误的预测与预测总数之间的比称为错误率。对于分类问题，我们可以简单的下结论：“对”与“错”，此时错误率很容易计算。回归问题不能使用简单的“对”或“错”来衡量，但可以用方差来描述准确的程度。比如，用三年内预计的客户增长数量同三年内实际的数据进行比较。

在一次模型的建立过程中，这种最简单的验证通常要执行几十次。例如，在训练神经网络时，几乎每一个训练周期都要在测试集上运行一次，不断地训练测试，直到在测试集上的准确率不再提高为止。

（2）交叉验证

交叉验证（Cross Validation，CV）是用来验证模型性能的一种统计分析方法，其基本思想是在某种意义下将原始数据（DataSet）进行分组，一部分作为训练集（Training Set），另一部分作为验证集（Test Set）。首先用训练集对分类器进行训练，再利用验证集来测试训练得到的模型（Model），以此来作为评价模型的性能指标。交叉验证在实际应用中非常普遍，适应性非常广，根据不同的交叉方式，又可分为以下三种情况：

①   Hold-Out Method

将原始数据随机分为两组，一组作为训练集，一组作为验证集，利用训练集训练分类器，然后利用验证集验证模型，记录最后的分类准确率为此Hold-Out Method下分类器的性能指标。此种方法的好处是处理简单，只需随机把原始数据分为两组即可，其实严格意义上来说，Hold-Out Method并不能算是CV，因为这种方法没有达到交叉的思想，由于是随机地将原始数据分组，所以最后验证集分类准确率的高低与原始数据分组有很大的关系，因此这种方法得到的结果其实并不具有说服性。

②   K-fold Cross Validation（记为K-CV）

将原始数据分成K组（一般是均分），将每个子集数据分别做一次验证集，其余的K-1组子集数据作为训练集，这样会得到K个模型，用这K个模型最终的验证集的分类准确率的平均数作为此K-CV下分类器的性能指标。K一般大于等于2，实际操作时一般从3开始取，只有在原始数据集合量小的时候才会尝试取2。K-CV可以有效地避免过学习以及欠学习状态的发生，最后得到的结果也比较具有说服性。

③   Leave-One-Out Cross Validation（记为LOO-CV）

如果设原始数据有N个样本，那么LOO-CV就是N-CV，即每个样本单独作为验证集，其余的N-1个样本作为训练集，所以LOO-CV会得到N个模型，用这N个模型最终的验证集的分类准确率的平均数作为此LOO-CV分类器的性能指标。相比于前面的K-CV，LOO-CV有两个明细的优点：一是每一回合中几乎所有的样本皆用于训练模型，因此最接近原始样本的分布，这样评估所得的结果比较可靠；二是实验过程中没有随机因素会影响实验数据，确保实验过程是可以被复制的。但LOO-CV的缺点则是计算成本高，因为需要建立的模型数量与原始数据样本数量相同，当原始数据样本数量相当多时，LOO-CV就会非常困难，除非每次训练分类器得到模型的速度很快，或可以用并行化计算减少计算所需的时间。当然也可以认为，全集验证是一种特殊的交叉验证方式。

我们可以依据得到的模型和对模型的预期结果修改参数，再用同样的算法建立新的模型，甚至可以采用其它算法建立模型。在数据挖据中，要根据不同的商业问题采用效果更好的模型，在没有行业经验的情况下，最好用不同的方法（参数或算法）建立几个模型，从中选择最好的。通过上面的处理，就会得到一系列的分析结果和模式，它们是对目标问题多侧面的描述，这时需要对它们进行验证和评价，以得到合理的、完备的决策信息。对产生模型结果需要进行对比验证、准确度验证、支持度验证等检验以确定模型的价值。这个阶段需要引入更多层面和背景的用户进行测试和验证，通过对几种模型的综合比较，产生最后的优化模型。

### 模型的评估

模型评估阶段需要对数据挖掘过程进行一次全面的回顾，从而确定是否存在重要的因素或任务由于某些原因而被忽视，此阶段的关键目的是确定是否还存在一些重要的商业问题仍未得到充分的考虑。验证模型是处理过程中的关键步骤，可以确定是否成功地进行了前面的步骤。模型的验证需要利用未参与建模的数据进行，这样才能得到比较准确的结果。可以采用的方法有直接使用原来建立模型的样本数据进行检验，或另找一批数据对其进行检验，也可以在实际运行中取出新的数据进行检验。检验的方法是对已知客户状态的数据利用模型进行挖掘，并将挖掘结果与实际情况进行比较。在此步骤中若发现模型不够优化，还需要回到前面的步骤进行调整。

模型的预测精确度是检验模型好坏的一个重要指标，但不是唯一指标。一个良好的数据挖掘模型，在投入实际应用前，需要经过多方面的评估，从而确定它完全达到了商业目标。评估数据挖掘模型优劣的指标有许多，比如精确度、LIFT、ROC、Gain图等。

精确度是最基本和最简单的指标。但是要让用户接受一个模型的结果，仅靠这些评估指标是不够的，还需要从模型结果的可用性上进一步阐述，即数据挖掘模型到底能带来什么业务上的价值。这实际上也就是数据挖掘模型的可解释性。在实际数据挖掘项目中，模型的可解释性往往比评估指标更为重要。

在对模型进行评估时，既要参照评估标准，同时也要考虑到商业目标和商业成功的标准。片面地追求预测正确率就会忽视了数据挖掘的初衷。我们不是为了建立一个完美的数学模型而进行挖掘，而是为了解决实际商业问题。所以挖掘产生结果的可解释性与实用性，才是最根本的标准。例如，在解决客户流失问题中，预测模型捕获越多的流失客户，不一定就代表能够协助挽留较多的客户。关键在于预测结果对挽留营销活动的制定有多大的帮助。

### 模型的部署

模型的部署一般是数据挖掘过程的最后一步，是集中体现数据挖据成果的一步。顾名思义，模型的部署就是将通过验证的评估模型，部署到实际的业务系统中，这样就可以应用在数据中挖掘到的知识。

一般而言，完成模型创建并不意味着项目结束。模型建立并经验证后，有两种主要的使用方法。一种是提供给分析人员做参考，由分析人员通过查看和分析这个模型后提出行动方案建议；另一种是把此模型开发并部署到实际的业务系统中。在部署模型后，还要不断监控它的效果，并不断改进之。

## 认识数据

### 数据的类型

        通常，数据集可以看作数据对象的集合。数据对象有时也叫做记录、点、向量、模式、事件、案例、样本、观测或实体。数据对象用一组刻画对象基本特征（如物体质量或事件发生时间）的属性描述。属性有时也叫做变量、特性、字段、特征或维。基于记录的数据集存储在平面文件或关系数据库系统中是最常见的。

（1）  属性与度量

属性（attribute）是数据对象的性质或特性，它因对象而异，或随时间变化。例如，眼球颜色因人而异，而气温随时间变化。眼球颜色是一种符号属性，具有少量可能的值，而温度是数值属性，可以取无穷多个值。属性并非数字或符号，但为了讨论和精细分析对象的特性，我们为它们赋予了数字或符号。为了用一种明确定义的方式做到这一点，需要定义测量标度。

测量标度（measurement scale）是将数值或符号值与对象的属性相关联的规则（函数）。形式上，测量过程是使用测量标度将一个值与一个特定对象的特定属性相关联。显然，属性的性质不必与用来度量它的值的性质相同。或者说，用来代表属性的值可能具有不同于属性本身的性质，反之亦然。例如，用户ID和年龄这两个属性都可以用整数表示，但是讨论用户的平均年龄是有意义的，而用户的平均ID却毫无意义。

属性的类型告诉我们，属性的哪些性质反映在用于测量它的值中。知道属性的类型是重要的，因为它告诉我们测量值的哪些性质与属性的基本性质一致，从而使得我们可以避免诸如计算用户平均ID的行为。通常将属性的类型称作测量标度的类型。

（2）  属性的类型

一种指定属性类型的简单方法是，确定对应属性基本性质的数值的性质。例如，长度的属性可以有数值的许多性质。按照长度比较对象，确定对象的排序，或讨论长度的差和比例都是有意义的。数值的如下性质（操作）常常用来描述属性。

* 相异性： =和!=
* 序： <、<=、>和>=
* 加法： +和-
* 乘法： \*和/

给定这些性质，我们可以定义四种属性类型：标称（nominal）、序数（ordinal）、区间（interval）和比率（ratio）。表1给出了这些类型的定义，以及每种类型上有哪些合法的统计操作等信息。每种属性类型拥有其上方属性类型上的所有性质和操作，即属性类型的定义是累积的。

|  |  |  |  |
| --- | --- | --- | --- |
| **属性类型** | | **描述** | **操作** |
| 分类的  （定性的） | 标称 | 标称属性的值仅仅是不同的名字，即标称值只提供足够的信息以区分对象（=，!=） | 众数、熵、列联相关、检验 |
| 序数 | 序数属性的值提供足够的信息确定对象的序（<，>） | 中值、百分位、秩相关、游程检验、符号检验 |
| 数值的  （定量的） | 区间 | 对于区间属性，值之间的差是有意义的，即存在测量单位（+，-） | 均值、标准差、皮尔森相关、t和F检验 |
| 比率 | 对于比率变量，差和比率都是有意义的（\*，/） | 几何平均、调和平均百分比变差 |

表1 不同的属性类型

标称和序数属性统称为分类的（categorical）或定性的（qualitative）属性。定性属性不具有数的大部分性质，即便使用数表示，也应当像对待符号一样对待它们。其余两种类型的属性，即区间和比率属性，统称为数值的（numeric）或定量的（quantitative）属性。定量属性用数表示，并且具有数的大部分性质。

属性的类型也可以用不改变属性意义的变换来描述。对特定属性类型有意义的统计操作是这样一些操作，当使用保持属性意义的变换对属性进行变换时，它们产生的结果相同。表2给出表1中四种属性类型的允许的（保持意义的）变换。

|  |  |  |
| --- | --- | --- |
| **属性类型** | | **变换** |
| 分类的  （定性的） | 标称 | 任何一对一变化，例如值的一个排列 |
| 序数 | 值的保序变换，即：新值=f(旧值)，其中f是单调函数 |
| 数值的  （定量的） | 区间 | 新值=a\*旧值+b，其中a、b是常数 |
| 比率 | 新值=a\*旧值 |

表2 定义属性的变换

区分属性的一种独立方法是根据属性可能取值的个数来判断。

* 离散的（discrete）：离散属性具有有限个值或无限可数个值。这样的属性可以是分类的，如邮编或用户ID，也可以是数值的，如计数。通常，离散属性用整数变量表示。二元属性（binaryattribute）是离散属性的一种特殊情况，它只接受两个值。二元属性一般用布尔变量表示，或者用只取两个值0或1的整型变量表示。
* 连续的（continuous）：连续属性是取实数值的属性，通常用浮点变量表示。实践中，实数只能用有限的精度测量和表示。

        从理论上讲，任何测量标度类型（标称、序数、区间和比率）都可以与基于属性值个数的任意类型（二元、离散和连续）组合，但有些组合并不常出现，或者没有什么意义。通常，标称和序数是二元的或离散的，而区间和比率属性是连续的。然而，计数属性（count attribute）是离散的，也是比率属性。

（3）  数据集的类型

        数据集的类型有多种，并且随着数据挖掘的发展与成熟，会有更多类型的数据集用于分析。为了便于说明，我们将常见的数据集类型分成三组：记录数据、基于图形的数据和时序数据。大部分数据挖掘算法都是以记录数据或其变体设计的。通过从数据对象中提取特征，并使用这些特征创建对应于每个对象的记录，针对记录数据的技术也可以用于非记录数据。

**数据集合的一般特性**

        在提供特定类型数据集的细节之前，我们先讨论适用于数据集的三个一般特性，即维度、稀疏性和分辨率，它们对数据挖掘技术具有重要影响。

* 维度（dimensionality）：数据集的维度是数据集中对象具有的属性数目。低维度数据往往与中、高维度数据有质的不同。在分析高维数据有时会陷入所谓的维灾难（curse of dimensionality），正因如此，数据预处理的一个重要动机就是减少维度，称为维归约或降维（dimensionality reduction）。
* 稀疏性（sparsity）：有些数据集，如具有非对称特征的数据集，一个对象的大部分属性上的值都为0，在许多情况下，非零项还不到1%。实际上，稀疏性是一个优点，因为只有非零值才需要存储和处理，这将节省大量的计算时间和存储空间。此外，有些数据挖掘算法仅适合处理稀疏数据。
* 分辨率（resolution）：常常可以在不同的分辨率下得到数据，并且在不同分辨率下数据的性质也有所不同。例如，在几米的分辨率下，地球表面看上去很不平坦，但在数十公里的分辨率下却相对平坦。数据的模式也依赖于分辨率。如果分辨率太高，模式可能看不出，或者淹没在噪声中；如果分辨率太低，模式可能不出现。例如，几小时记录一下气压变化可以反映出风暴等天气系统的变化；而在月的标度下，这些现象就检测不到。

**记录数据**

        许多数据挖掘任务都假定数据集是记录（数据对象）的汇集，每个记录包含固定的数据字段（属性）集。对于记录数据的大部分基本形式，记录之间或数据字段之间没有明显的联系，并且每个记录（对象）具有相同的属性集。记录数据通常存放在平面文件或关系数据库中。下面介绍不同类型的记录数据。

事务数据（购物篮数据）：事务数据（transactiondata）是一种特殊的记录数据，其中每个记录（事务）涉及一系列的条目。电商用户一次购物所购买的商品的集合就构成一个事务，而购买的商品是条目。这种类型的数据也称作购物篮数据（market basket data），因为记录中的条目是用户“购物篮”中的商品。事务数据是记录的集合，其中记录的字段是非对称的属性。这些属性常常是二元的，指出商品是否已买。更一般地，这些属性还可以是离散的或连续的，例如购买的商品的数量或购买商品的花费。注意这里所说的事务与通常数据库系统中的事务不是一个概念。

数据矩阵：如果一个数据集合中的所有数据对象都具有相同的数值属性集，则数据对象可以看作多维空间中的点（向量），其中每个维度代表对象的一个不同属性。这样的数据对象集合可以用一个m×n的矩阵表示，其中m行，一个对象一行；n列，一个属性一列。（也可以将数据对象用列表示，属性用行表示。）这种矩阵称作数据矩阵（datamatrix）或模式矩阵（pattern matrix）。数据矩阵是记录数据的变体，由于它由数值属性组成，可以使用标准的矩阵操作对数据进行交换和处理。因此，对于大部分统计数据，数据矩阵是一种标准的数据格式。

稀疏数据矩阵：稀疏数据矩阵是数据矩阵的一种特殊情况，其中属性的类型相同并且是非对称的，即只有非零值才是重要的。事务数据是仅含0-1元素的稀疏数据矩阵的例子。另一个常见的例子是文档数据。如果忽略文档中词的次序，则文档可以用词向量表示，其中每个词是向量的一个分量（属性），而每个分量的值对应词在文档中出现的次数。文档集合的这种表示通常称作文档-词矩阵（document-term matrix）。实践应用时，只需存储稀疏矩阵的非零项。

**基于图形的数据**

        有时图形可以方便而有效地表示数据。我们考虑两种特殊情况：（1）图形捕获数据对象之间的联系；（2）数据对象本身用图形表示。

带有对象之间联系的数据：对象之间的联系常常携带重要信息。在这种情况下，数据常常用图形表示。一般把数据对象映射到图的节点，而对象之间的联系用对象之间的连接，诸如方向、权值等连接性质表示。考虑互联网上的网页，页面上包含文本和指向其它页面的链接。为了处理搜索查询，Web搜索引擎收集并处理网页，提取它们的内容。然而，指向或出自每个页面的链接包含了大量该页面与查询相关程度的信息，因而必须考虑。

具有图形对象的数据：如果对象具有结构，即对象包含具有联系的子对象，则这样的对象常常用图形表示。例如，化合物的结构可以用图形表示，其中节点是原子，节点之间的连接是化学键。图形表示可以确定何种结构频繁出现在化合物的集合中，并且查明这些子结构中是否有某种结构与其化学性质有关。

**时序数据**

        时序数据（sequential data）也称时间数据（temporal data），可以看作记录数据的扩充，其中每个记录包含一个与之相关联的时间。考虑存储事务发生时间的零售事务数据，时间信息可以帮助我们发现与时间相关的特定商品的销售峰值，从而利用这类模式进行促销活动。时间也可以与每个属性相关联，例如，每个记录可以是一位客户的购物历史，包含不同时间购买的商品列表。使用这些信息，就有可能发现购买商品时间顺序之类的模式。

        时间序列数据（time series data）是一种特殊的时序数据，其中每个记录都是一个时间序列（time series），即一段时间以来的测量序列。例如，金融数据集可能包含各种股票每日价格的时间序列对象。

# MADlib基础

## MADlib简介

        MADlib是Pivotal公司与伯克利大学合作开发的一个开源机器学习库，提供了多种数据转换、数据探索、统计、数据挖掘和机器学习方法，使用它能够简易地对结构化数据进行分析和挖掘。用户可以非常方便地将MADlib加载到数据库中，扩展数据库的分析功能。2015年7月MADlib成为Apache软件基金会的孵化器项目，经过两年的发展，于2017年8月毕业成为Apache顶级项目。其当前最新版本为MADlib 1.12，可以与PostgreSQL、Greenplum和HAWQ等数据库系统无缝集成。

### 基本概念

        无论是经典的SAS、SPSS，还是时下流行的MATLAB、R、Python，所有这些机器学习或数据挖掘软件，都是自成系统的。具体说就是具有一套完整的程序语言及其集成开发环境，提供了丰富的数学和统计分析函数，具备良好的人机交互界面，支持从数据准备、数据探索、数据预处理，到开发和实现模型算法、数据可视化，再到最终结果的验证与模型部署到应用的全过程。它们都是面向程序员的系统或语言，重点在于由程序员自己利用系统提供的基本计算方法或函数，通过编程的方式实现所需的模型算法。表1给出了5种常用数据挖掘工具在功能、特点、适用场景方面的比较，从中可以看出，每种工具都有自己的特点和适应条件。

|  |  |  |  |
| --- | --- | --- | --- |
| 工具名称 | 功能 | 特点 | 适用场景 |
| MATLAB | 不仅具有较强的数据统计、科学计算功能，还具有金融、经济等众多的行业应用工具箱 | 擅长矩阵计算和仿真模拟；  具有丰富的数学函数，适合算法开发或自主的程序开发；  具有强大的绘图功能 | 适合学习研究算法和灵活的产品开发 |
| SAS | 功能强大的统计分析软件 | 具有较强的大数据处理能力；  支持二次开发 | 有一些行业标准，适合工业使用 |
| SPSS | 侧重统计分析 | 使用方便，但不适合自己开发代码，就是说扩展上受限，如果要求不高，已经足够 | 界面友好，使用简单，但功能强大，也可以编程，能解决大部分统计学问题，适合初学者 |
| WEKA | 具有丰富数据挖掘函数，包括分类、聚类、关联分析等主流算法 | Java开发的开源数据分析、机器学习工具 | 适合于具有一定程序开发经验的工程师，尤其适合于用Java进行二次开发 |
| R | 类似MATLAB，具有丰富的数学和统计分析函数 | 开源并支持二次开发 | 适合算法学习、小项目的产品研发 |

表1 常用数据挖掘工具的比较

        MADlib具有与上述工具完全不同的设计理念，它不是面向程序员的，而是面向数据库开发或DBA的。如果用一句话说明什么是MADlib，那就是“SQL中的大数据机器学习库”。通常SQL查询能发现数据最明显的模式和趋势，但要想获取数据中最为有用的信息，需要的其实是完全不同的另一套技术，一套牢固扎根于数学和应用数学的技能，当然指的就是数据挖掘或机器学习，而具备这种技术的人才似乎只存在于学术界中。如果能将SQL的简单易用与数据挖掘的复杂算法结合起来，充分利用两者的优势和特点，那对于广大传统数据库应用技术人员来说，学习和从事数据挖掘工作的门槛将大大降低。现在，鱼和熊掌兼得的机会来了，DBAer不用现学Python、R或MATLAB，只要使用MADlib，用SQL就能实现简单的数据挖掘。

        对用户而言，MADlib仅提供了可在SQL查询语句中调用的函数。其中不但包括基本的线性代数运算和统计函数，而且还提供了常用的、现成的机器学习或数据挖掘模型函数。用户不需要深入了解算法的程序实现细节，只要搞清楚各函数中相关参数的含义，从而提供正确的入参，并且能够理解和解释函数的输出结果即可。这种使用方式无疑会极大地提高开发效率，节约开发成本。在MADlib的世界里，一切皆函数，就是这么简单。

        然而任何事物都具有两面性，MADlib提供了使用方便性，但相对于其它数据挖掘系统而言，灵活性、扩展性与功能完备性显然是其短板。这很好理解，首先，模型已经被封装在SQL函数中，性能优劣完全依赖于函数本身，基本没有留给用户进行性能调整的空间。其次，函数只能在SQL中调用，而SQL依赖于数据库系统，也就是说单独的MADlib函数库是无意义的，它必须与PostgreSQL、Greenplum和HAWQ等数据库系统结合使用。最后，既然MADlib是SQL中的机器学习库，注定它不关心数据可视化，本身不带数据的图形化表示功能。由此可见，MADlib作为工具，并不是传统意义上的数据挖掘系统软件，而只是一套可在SQL中调用的函数库，其出发点是让数据库技术人员用SQL快速完成简单的数据挖掘工作。

        即便如此，MADlib的易用性已经足以引起我们的兴趣。在了解了MADlib是什么及其优缺点后，用户就能根据自己的实际情况和需求，有针对性地选择和使用MADlib来实现特定业务目标。

### 架构

        MADlib架构如图1所示。

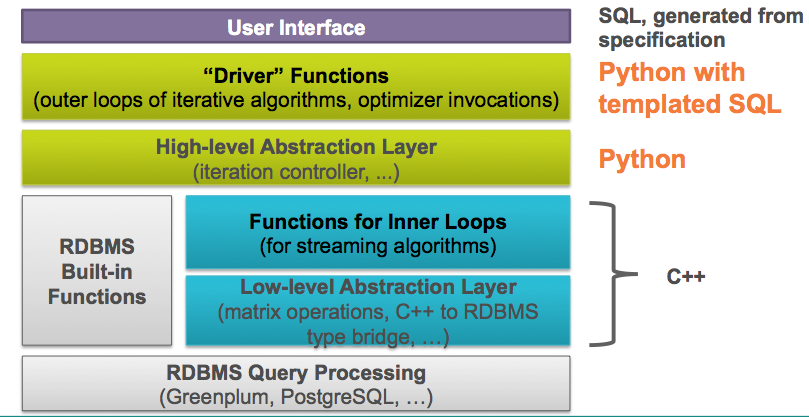


图1 MADlib架构

        处于架构最上面一层是用户接口。如前所述，用户只需通过在SQL查询语句中调用MADlib提供的函数来完成数据挖掘的工作。当然这里的SQL语法要与特定数据库管理系统相匹配。最底层则是Greenplum、PostgreSQL等数据库管理系统，最终由它们处理查询请求。

        从图1中看到，MADlib系统架构自上至下由以下四个主要组件构成：

* Python调用SQL模板实现的驱动函数
* Python实现的高级抽象层
* C++实现的核心函数
* C++实现的低级数据库抽象层

 （1）Python驱动函数

        驱动函数是用户输入的主入口点，调用优化器执行迭代算法的外层循环。

 （2）Python实现的高级抽象层

        高级抽象层负责算法的流程控制。与驱动函数一起实现输入参数验证、SQL语句执行、结果评估，并可能在循环中自动执行更多的SQL语句直到达到某些收敛标准。

 （3）C++实现的核心函数

        这部分函数是由C++编写的核心函数，在内层循环中实现特定机器学习或数据挖掘算法。出于性能考虑，这些函数使用C++而不是Python编写。

 （4）C++实现的低级数据库抽象层

        这些函数提供一个编程接口，将所有的Postgres数据库内核实现细节进行抽象。它们提供了一种机制，使得MADlib能够支持不同的后端平台，从而将关注点集中在内部功能而不是平台集成上。

### 设计思想

        驱动MADlib架构的主要设计思想与Hadoop是一致的，体现在以下方面：

* 操作数据库内的本地数据，不在多个运行时环境中进行不必要的数据移动。
* 充分利用数据库引擎功能，但将数据挖掘逻辑从特定数据库的实现细节中分离出来。
* 利用MPP无共享技术提供的并行性和可扩展性，如Greenplum或HAWQ数据库系统。
* 执行的维护活动对Apache社区和正在进行的学术研究开放。

## MADlib支持的模型类型

MADlib支持以下常用的数据挖掘与机器学习模型类型，其中大部分模型都包含训练和预测两组函数。

 （1）回归

        如果所需的输出具有连续性，我们使用回归方法建立模型，预测输出值。例如，如果有真实的描述房地产属性的数据，我们就可以建立一个模型，预测基于房屋已知特征的售价。因为输出反应了连续的数值而不是分类，所以该场景是一个回归问题。

（2）分类

        如果所需的输出实质上是分类的，可以使用分类方法建立模型，预测新数据会属于哪一类。分类的目标是能够将输入记录标记为正确的类别。例如，假设有描述人口统计的数据，以及个人申请贷款和贷款违约历史数据，那么我们就能建立一个模型，描述新的人口统计数据集合贷款违约的可能性。此场景下输出的分类为“违约”和“正常”两类。

（3）关联规则挖掘

        有时又叫做购物篮分析或频繁项集挖掘。相对于随机发生，确定哪些事项更经常一起发生，指出事项之间的潜在关系。例如，在一个网店应用中，关联规则挖掘可用于确定哪些商品倾向于被一起售出，然后将这些商品输入到客户推荐引擎中，提供促销机会，就像著名的啤酒与尿布的故事。

（4）聚类

        识别数据分组，一组中的数据项比其它组的数据项更相似。例如，在客户细分分析中，目标是识别客户行为相似特征组，以便针对不同特征的客户设计各种营销活动，以达到市场目的。如果提前了解客户细分情况，这将是一个受控的分类任务。当我们让数据识别自身分组时，这就是一个聚类任务。

（5）主题建模

        主题建模与聚类相似，也是确定彼此相似的数据组。但这里的相似通常特指在文本领域中，具有相同主题的文档。

（6）描述性统计

        描述性统计不提供模型，因此不被认为是一种机器学习方法。但描述性统计有助于向分析人员提供信息以了解基础数据，为数据提供有价值的解释，可能影响数据模型的选择。例如，计算数据集中每个变量内的数据分布，可以帮助分析理解哪些变量应被视为分类变量，哪些变量是连续性变量，以及值的分布情况。描述性统计通常是数据探索的组成部分。

（7）模型验证

        如果不了解一个模型的准确性就开始使用它，很容易导致糟糕的结果。正因如此，理解模型存在的问题，并用测试数据评估模型的精度尤为重要。需要将训练数据和测试数据分离，频繁进行数据分析，验证统计模型的有效性，评估模型不过分拟合训练数据。N-fold交叉验证方法经常被使用。

## MADlib的功能

        MADlib的功能特色如图2所示。

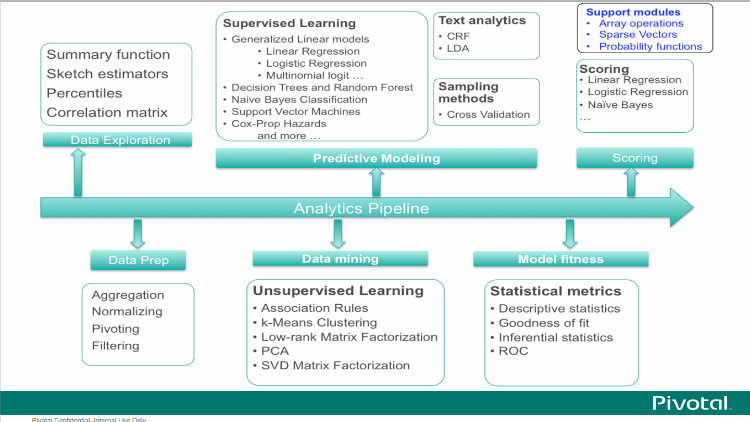


图2 MADlib功能

        下面基于MADlib 1.1.10版本，预览MADlib提供的具体模型算法或功能。

 （1）Data Types andTransformations（数据类型与转换）

        Arraysand Matrices（数组与矩阵）

                ArrayOperations（数组运算）

                MatrixOperations（矩阵运算）

                MatrixFactorization（矩阵分解）

                        Low-rankMatrix Factorization（低阶矩阵分解）

                        SingularValue Decomposition（SVD，奇异值分解）

                Normsand Distance functions（范数和距离函数）

                SparseVectors（稀疏向量）

        DimensionalityReduction（降维）

                PrincipalComponent Analysis（PCA主成分分析）

                PrincipalComponent Projection（PCP主成分投影）

        Pivot（透视表）

        EncodingCategorical Variables（分类变量编码）

        Stemming（词干提取）

（2）Graph（图）

        SingleSource Shortest Path（单源最短路径）

（3）Model Evaluation（模型评估）

        CrossValidation（交叉验证）

        PredictionMetrics（指标预测）

 （4）Statistics（统计）

        DescriptiveStatistics（描述性统计）

                Pearson’s Correlation（皮尔逊相关系数）

                Summary（摘要汇总）

        InferentialStatistics（推断性统计）

                HypothesisTests（假设检验）

        ProbabilityFunctions（概率函数）

 （5）Supervised Learning（监督学习）

        ConditionalRandom Field（条件随机场）

        RegressionModels（回归模型）

                ClusteredVariance（聚类方差）

                Cox-ProportionalHazards Regression（Cox比率风险回归）

                ElasticNet Regularization（弹性网络回归）

                GeneralizedLinear Models（广义线性回归）

                LinearRegression（线性回归）

                LogisticRegression（逻辑回归）

                MarginalEffects（边际效应）

                MultinomialRegression（多分类逻辑回归）

                OrdinalRegression（有序回归）

                RobustVariance（鲁棒方差）

        SupportVector Machines（支持向量机）

        TreeMethods（树方法）

                DecisionTree（决策树）

                RandomForest（随机森林）

 （6）Time Series Analysis（时间序列分析）

        ARIMA（自回归积分滑动平均）

 （7）UnsupervisedLearning（无监督学习）

        AssociationRules（关联规则）

                AprioriAlgorithm（Apriori算法）

        Clustering（聚类）

                k-MeansClustering（k-Means）

        TopicModelling（主题模型）

                LatentDirichlet Allocation（LDA）

 （8）Utility Functions（应用函数）

        DeveloperDatabase Functions（开发者数据库函数）

        LinearSolvers（线性求解器）

                DenseLinear Systems（稠密线性系统）

                SparseLinear Systems（稀疏线性系统）

        PathFunctions（路径函数）

        PMMLExport（PMML导出）

        Sessionize（会话化）

        TextAnalysis（文本分析）

                TermFrequency（词频）

## 安装与卸载MADlib

### 确定安装平台

        MADlib最新发布版本是1.12，可以安装在PostgreSQL、Greenplum和HAWQ中，在不同的数据库中安装过程也不尽相同。这里以在HAWQ2.1.1.0中安装MADlib为例，演示MADlib的安装与卸载过程。后续进行的一系列示例也都在此实验环境中进行。HAWQ的安装与部署参见“HAWQ技术解析（二） —— 安装部署”。

        数据挖掘需要数据库系统提供有效的存储、索引和查询处理支持。源于高性能（并行）计算的技术在处理海量数据集方面常常是重要的。分布式技术也能帮助处理海量数据，并且当数据不能集中到一起处理时更是至关重要。

        比照以上数据挖掘对数据库系统提出的要求，我们不妨简单考量一下HAWQ。先提出一点，HAWQ目前不支持索引。但是，对于存储在Hadoop集群上的“大数据”分析应用而言，实际执行的操作几乎都是表扫描，很少需要定位几行数据，因此传统的由用户定义的索引，其作用在此场景下微乎其微。而HAWQ使用的随机分布存储策略具有较好的数据本地化特性，优化器在制定查询计划时，内部实现已然利用了索引的思想。HAWQ使用专为HDFS量身打造的，基于成本的查询优化框架来增强其性能。所采用的MPP架构，使用户能够获益于基于MPP的分析功能及其查询性能，同时有效利用HDFS的分布式存储、容错机制、机架感知等功能，兼顾了低延时与高扩展。由此看来，在HAWQ上运行MADlib，实现大数据挖掘，是一个比较合理的选择。

### 下载MADlib二进制压缩包

        下载地址为：<https://network.pivotal.io/products/pivotal-hdb>。2.1.1.0版本的HAWQ提供了四个安装文件，如图3所示。经过测试，本环境只有MADlib 1.10.0版本的文件可以正常安装。

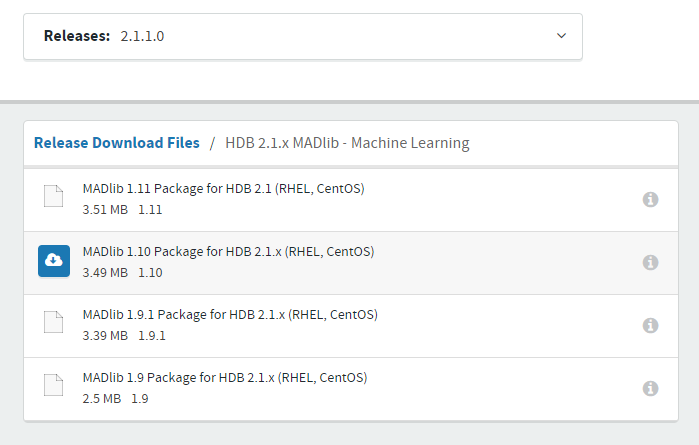


图3 下载MADlib安装文件

### 安装MADlib

        以下命令需要使用gpadmin用户，在HAWQ的Master主机上执行。

（1）解压缩

tar -zxvfmadlib-ossv1.10.0\_pv1.9.7\_hawq2.1-rhel5-x86\_64.tar.gz

（2）安装MADlib的gppkg文件

gppkg -imadlib-ossv1.10.0\_pv1.9.7\_hawq2.1-rhel5-x86\_64.gppkg

        该命令在HAWQ集群的所有节点（Master和Segment）上创建MADlib的安装目录和文件，缺省目录为/usr/local/hawq\_2\_1\_1\_0/madlib。gppkg是Greenplum的包管理器应用程序，用于在集群所有节点上安装Greenplum数据库扩展包及其依赖。

（3）在指定数据库中部署MADlib

$GPHOME/madlib/bin/madpack install -c /dm-s madlib -p hawq

        该命令在HAWQ的dm数据库中建立madlibschema，-p参数指定平台为HAWQ。命令执行后可以查看在madlib schema中创建的数据库对象。

dm=# set search\_path=madlib;

SET

dm=# \dt

List of relations

Schema | Name | Type | Owner | Storage

--------+------------------+-------+---------+-------------

madlib | migrationhistory | table | gpadmin | append only

(1 row)

dm=# \ds

List of relations

Schema | Name | Type | Owner | Storage

--------+-------------------------+----------+---------+---------

madlib | migrationhistory\_id\_seq | sequence | gpadmin | heap

(1 row)

dm=# select type,count(\*)

dm-# from (select p.proname as name,

dm(# case when p.proisagg then 'agg'

dm(# when p.prorettype

dm(# = 'pg\_catalog.trigger'::pg\_catalog.regtype

dm(# then 'trigger'

dm(# else 'normal'

dm(# end as type

dm(# from pg\_catalog.pg\_proc p, pg\_catalog.pg\_namespace n

dm(# where n.oid = p.pronamespace and n.nspname='madlib') t

dm-# group by rollup (type);

type | count

--------+-------

agg | 135

normal | 1324

| 1459

(3 rows)

        从查询结果可以看到，MADlib部署应用程序madpack首先创建数据库模式madlib，然后在该模式中创建数据库对象，包括一个表，一个序列，1324个普通函数，135个聚合函数。所有的机器学习和数据挖掘模型、算法、操作和功能都是通过调用这些函数实际执行的。

（4）验证安装

$GPHOME/madlib/bin/madpack install-check -c/dm -s madlib -p hawq

        该命令通过执行29个模块的77个案例，验证所有模块都能正常工作。命令输出如下，如果看到所有案例都已经正常执行，说明MADlib安装成功。这条命令需要执行较长时间。

[gpadmin@hdp3 Madlib]$ $GPHOME/madlib/bin/madpack install-check -c /dm -s madlib -p hawq

madpack.py : INFO : Detected HAWQ version 2.1.

TEST CASE RESULT|Module: array\_ops|array\_ops.sql\_in|PASS|Time: 1851 milliseconds

TEST CASE RESULT|Module: bayes|gaussian\_naive\_bayes.sql\_in|PASS|Time: 24222 milliseconds

TEST CASE RESULT|Module: bayes|bayes.sql\_in|PASS|Time: 70634 milliseconds

…

TEST CASE RESULT|Module: pca|pca.sql\_in|PASS|Time: 523230 milliseconds

TEST CASE RESULT|Module: validation|cross\_validation.sql\_in|PASS|Time: 33685 milliseconds

[gpadmin@hdp3 Madlib]$

### 卸载MADlib

        卸载过程基本上是安装的逆过程。

（1）删除madlib模式

        方法1，使用madpack部署应用程序删除模式。

$GPHOME/madlib/bin/madpack uninstall -c /dm-s madlib -p hawq

        方法2，使用SQL命令手工删除模式。

drop schema madlib cascade;

（2）删除其它遗留数据库对象

        删除模式

        如果模型验证过程中途出错，数据库中可能包含测试的模式，这些模式名称的前缀都是madlib\_installcheck\_，只能手工执行SQL命令删除这些模式，如：

drop schema madlib\_installcheck\_kmeanscascade;

        删除用户

        如果存在遗留的测试用户，则删除它，如：

drop user if existsmadlib\_1100\_installcheck;

（3）删除MADlib rpm包

        查询包名：

gppkg -q --all

        输出如下：

[gpadmin@hdp3 Madlib]$ gppkg -q --all

20170630:16:19:53:076493 gppkg:hdp3:gpadmin-[INFO]:-Starting gppkg with args: -q --all

madlib-ossv1.10.0\_pv1.9.7\_hawq2.1

        删除rpm包：

gppkg -rmadlib-ossv1.10.0\_pv1.9.7\_hawq2.1

# 数据类型之向量

        通常数据挖掘操作的数据集可以看作数据对象的集合。数据对象有时也叫做记录、点、向量、模式、事件、案例、样本、观测或实体。数据对象用一组刻画对象基本特征（如物体质量或事件发生的时间）的属性描述。属性有时也叫做变量、特性、字段、特征或维。而在数学上，向量和矩阵可以用来表示数据对象及其属性。

        和其它数据挖掘语言或工具一样，MADlib操作的基本对象也是向量与矩阵。对向量和矩阵的操作是通过一系列函数完成的。本篇将介绍MADlib中向量的概念，并举出一些简单的函数调用示例。用户可以使用psql的联机帮助，查看函数的参数、返回值和函数体等信息，例如：\df madlib.array\_add或\df+ madlib.array\_add。我们将侧重于应用，因为理解这些函数的意义和用法是使用MADlib进行数据挖掘的基础。

## 向量定义

        数学中的向量（vector，也称为欧几里得向量、几何向量、矢量），是一个具有大小（magnitude）和方向（direction）的值。它可以形象化地表示为带箭头的线段。箭头所指代表向量的方向，线段长度代表向量的大小。图1（a）给出了两个向量：向量u长度为1、平行于y轴，向量v长度为2、与x轴夹角为45º。

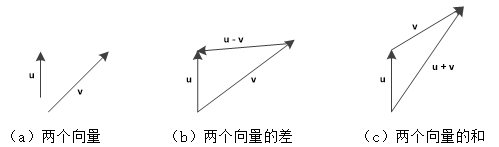


图1 两个向量以及它们的和与差

## MADlib中的向量操作函数

        在MADlib中，一维数组与向量具有相同的含义。MADlib的数组运算模块（array\_ops）提供了一组用C实现的基本数组操作，是数据挖掘算法的支持模块。数组运算函数支持以下数字类型：

* SMALLINT
* INTEGER
* BIGINT
* REAL
* DOUBLE PRECISION（FLOAT8）
* NUMERIC（内部被转化为FLOAT8，可能丢失精度）

         数组运算函数列表及功能描述如表1所示。

|  |  |
| --- | --- |
| **函数** | **描述** |
| array\_add() | 两个数组相加，需要所有值非空，返回与输入相同的数据类型。 |
| sum() | 数组元素求和，需要所有值非空，返回与输入相同的数据类型。 |
| array\_sub() | 两个数组相减，需要所有值非空，返回与输入相同的数据类型。 |
| array\_mult() | 两个数组相乘，需要所有值非空，返回与输入相同的数据类型。 |
| array\_div() | 两个数组相除，需要所有值非空，返回与输入相同的数据类型。 |
| array\_dot() | 两个数组点积，需要所有值非空，返回与输入相同的数据类型。 |
| array\_contains() | 检查一个数组是否包含另一个数组。如果右边数组中的每个非零元素都等于左边数组中相同下标的元素，函数返回TRUE。 |
| array\_max() | 返回数组中的最大值，忽略空值，返回数组元素的相同类型。 |
| array\_max\_index() | 返回数组中的最大值及其对应的下标，忽略空值，返回类型的格式为[max, index]，其元素类型与输入类型相同。 |
| array\_min() | 返回数组中的最小值，忽略空值，返回数组元素的相同类型。 |
| array\_min\_index() | 返回数组中的最小值及其对应的下标，忽略空值，返回类型的格式为[min, index]，其元素类型与输入类型相同。 |
| array\_sum() | 返回数组中值的和，忽略空值，返回与输入相同的数据类型。 |
| array\_sum\_big() | 返回数组中值的和，忽略空值，返回FLOAT8类型。该函数的意思是当汇总值可能超出元素类型范围时，替换array\_sum()。 |
| array\_abs\_sum() | 返回数组中绝对值的和，忽略空值，返回与输入相同的数据类型。 |
| array\_abs() | 返回由数组元素的绝对值组成的新数组，需要所有值非空。 |
| array\_mean() | 返回数组的均值，忽略空值。 |
| array\_stddev() | 返回数组的标准差，忽略空值。 |
| array\_of\_float() | 创建元素个数为参数值的FLOAT8数组，初始值为0.0。 |
| array\_of\_bigint() | 创建元素个数为参数值的BIGINT数组，初始值为0。 |
| array\_fill() | 将数组每个元素设置为参数值。 |
| array\_filter() | 过滤掉数组中的指定元素，要求所有值非空。返回与输入相同的数据类型。不指定被过滤元素时，该函数移除数组中的所有0值。 |
| array\_scalar\_mult() | 数组与标量相乘，返回结果数组。需要所有值非空，返回与输入相同的数据类型。 |
| array\_scalar\_add() | 数组与标量相加，返回结果数组。需要所有值非空，返回与输入相同的数据类型。 |
| array\_sqrt() | 返回由数组元素的平方根组成的数组，需要所有值非空。 |
| array\_pow() | 以数组和一个float8为输入，返回每个元素的乘幂（由第二个参数指定）组成的数组, 需要所有值非空。 |
| array\_square() | 返回由数组元素的平方组成的数组，需要所有值非空。 |
| normalize() | 该函数规范化一个数组，使它的元素平方和为1。要求所有值非空。 |

表1 MADlib数组运算函数

        下面用具体的例子说明函数的含义及用法。

（1）建立具有两个[整型](https://www.baidu.com/s?wd=%E6%95%B4%E5%9E%8B&tn=24004469_oem_dg)数组列array1和array2的数据库表并添加数据。

drop table if exists array\_tbl;

create table array\_tbl ( id integer, array1 integer[], array2 integer[] );

insert into array\_tbl values

( 1, '{1,2,3,4,5,6,7,8,9}','{9,8,7,6,5,4,3,2,1}' ),

( 2, '{1,1,0,0,1,2,3,99,8}','{0,0,0,-5,4,1,1,7,6}');

（2）查询array1列的最小值及下标、最大值及下标、平均值和标准差。

select id, madlib.array\_min(array1)min,

madlib.array\_max(array1) max,

madlib.array\_min\_index(array1) min\_idx,

madlib.array\_max\_index(array1) max\_idx,

madlib.array\_mean(array1) mean,

madlib.array\_stddev(array1) stddev

from array\_tbl;

        结果：

id | min | max | min\_idx | max\_idx | mean | stddev

----+-----+-----+---------+---------+------------------+------------------

1 | 1 | 9 | {1,1} | {9,9} | 5 | 2.73861278752583

2 | 0 | 99 | {0,3} | {99,8} | 12.7777777777778 | 32.4259840936932

(2 rows)

        说明：

* MADlib的数组下标从1开始。
* 标准差的计算公式为：https://img-blog.csdn.net/20171222161555739，其中μ为平均值。

         可以执行下面的查询验证标准差，结果同样是32.4259840936932。

select sqrt(sum(power(a-avg\_a,2))/(count(\*)-1))

from (select avg(a) avg\_a

from (select unnest(array1) a from array\_tbl where id=2) t) t1,

(select unnest(array1) a from array\_tbl where id=2) t2;

（3）执行数组加减运算。

select id,madlib.array\_add(array1,array2),

madlib.array\_sub(array1,array2)

from array\_tbl;

        结果：

id | array\_add | array\_sub

----+------------------------------+-------------------------

1 | {10,10,10,10,10,10,10,10,10} | {-8,-6,-4,-2,0,2,4,6,8}

2 | {1,1,0,-5,5,3,4,106,14} | {1,1,0,5,-3,1,2,92,2}

(2 rows)

        与数的加法一样，向量的加法也具有一些我们熟知的性质。如果u、v和w是3个向量，则向量的加法具有如下性质。

* 向量加法的交换律。加的次序不影响结果：u + v =v + u。
* 向量加法的结合律。相加时向量分组不影响结果：(u +v) + w = u + (v + w)。
* 向量加法单位元存在性。存在一个零向量（zero vector），简记为0，是单位元。对于任意向量u，有u + 0 = u。
* 向量加法逆元的存在性。对于每个向量u，都存在一个逆向量-u，使得u + (-u) = 0。

（4）数组乘以一个标量。

select id, madlib.array\_scalar\_mult(array1,3), madlib.array\_scalar\_mult(array1,-3)

from array\_tbl;

        结果：

id | array\_scalar\_mult | array\_scalar\_mult

----+---------------------------+------------------------------------

1 | {3,6,9,12,15,18,21,24,27} | {-3,-6,-9,-12,-15,-18,-21,-24,-27}

2 | {3,3,0,0,3,6,9,297,24} | {-3,-3,0,0,-3,-6,-9,-297,-24}

(2 rows)

        标量乘改变向量的量值，如果标量是正则方向不变，如果标量为负则方向相反。如果u和v是向量，α和β是标量（数），则向量的标量乘法具有如下性质。

* 标量乘法的结合律。被两个标量乘的次序不影响结果：α(βu)=(αβ)u。

select id,

madlib.array\_scalar\_mult(madlib.array\_scalar\_mult(array1,3),2),

madlib.array\_scalar\_mult(madlib.array\_scalar\_mult(array1,2),3)

from array\_tbl;

        结果：

id | array\_scalar\_mult | array\_scalar\_mult

----+-----------------------------+-----------------------------

1 | {6,12,18,24,30,36,42,48,54} | {6,12,18,24,30,36,42,48,54}

2 | {6,6,0,0,6,12,18,594,48} | {6,6,0,0,6,12,18,594,48}

(2 rows)

* 标量加法对标量与向量乘法的分配率。两个标量相加后乘以一个向量等于每个标量乘以该向量之后的结果向量相加：(α+β)u =αu + βu。

select id,

madlib.array\_scalar\_mult(array1,5),

madlib.array\_add

(madlib.array\_scalar\_mult(array1,2),madlib.array\_scalar\_mult(array1,3))

from array\_tbl;

        结果：

id | array\_scalar\_mult | array\_add

----+-----------------------------+-----------------------------

1 | {5,10,15,20,25,30,35,40,45} | {5,10,15,20,25,30,35,40,45}

2 | {5,5,0,0,5,10,15,495,40} | {5,5,0,0,5,10,15,495,40}

(2 rows)

* 标量乘法对向量加法的分配率。两个向量相加之后的和与一个标量相乘等于每个向量与该标量相乘然后相加：α(u + v) =αu +αv。

select id,

madlib.array\_scalar\_mult(madlib.array\_add(array1, array2),3),

madlib.array\_add

(madlib.array\_scalar\_mult(array1,3),madlib.array\_scalar\_mult(array2,3))

from array\_tbl;

        结果：

id | array\_scalar\_mult | array\_add

----+------------------------------+------------------------------

1 | {30,30,30,30,30,30,30,30,30} | {30,30,30,30,30,30,30,30,30}

2 | {3,3,0,-15,15,9,12,318,42} | {3,3,0,-15,15,9,12,318,42}

(2 rows)

* 标量单位元的存在性。如果α = 1，则对于任何向量u，有αu = u。

        由向量加法和标量与向量乘法引出了向量空间的概念。向量空间（vector space）是向量的集合，连同一个相关联的标量集（如实数集），满足上述性质，并且关于向量加法和标量与向量乘法是封闭的。封闭是指向量相加的结果、向量与标量相乘的结果都是原向量集中的向量。向量空间具有如下性质：任何向量都可以用一组称作基（basis）的向量线性组合（linear combination）表示。更明确地说，如果https://img-blog.csdn.net/20171222163119761是基向量，则对于任意向量v，都可以找到n个标量的集合https://img-blog.csdn.net/20171222163225545使得https://img-blog.csdn.net/20171222163353667。我们称基向量生成（span）了该向量空间。向量空间的维（dimension）是形成基所需要的最少向量数。通常，我们选取具有单位长度的基向量。

        基向量通常是正交的（orthogonal）。向量正交是直线垂直的二维概念的推广。从概念上讲，正交向量是不相关的或独立的。如果基向量是相互正交的，则将向量表示成基向量的线性组合事实上把该向量分解成一些独立分量（independent component）。

        因此，n维空间的向量可以看作标量（数）的n元组。为了具体地解释，考虑二维欧几里得空间，其中每个点都与一个表示该点到原点的位移的向量相关联。到任意点的位移向量都可以用x方向和y方向的位移和表示。这些位移分别是该点的x和y坐标。

        我们使用记号https://img-blog.csdn.net/20171222163703703引述向量v的分量。注意，https://img-blog.csdn.net/20171222163902665是向量v的一个分量，而https://img-blog.csdn.net/20171222163941088是向量集中的一个向量。从向量的分量角度看，向量的加法变得简单并易于理解。为了将两个向量相加，我们只需要简单地将对应的分量相加。例如，(2,3）+(4,2)=(6,5)。为了计算标量乘以向量，我们只要用标量乘以每个分量，如3×(2,3) = (6,9)。

（5）数组乘除。这里过滤掉了id=2的行，否则查询会因为除零错误而失败。

select id,madlib.array\_mult(array1,array2),

madlib.array\_div(array1,array2)

from array\_tbl

where 0 != all(array2);

        结果：

id | array\_mult | array\_div

----+----------------------------+---------------------

1 | {9,16,21,24,25,24,21,16,9} | {0,0,0,0,1,1,2,4,9}

(1 row)

        参与计算的两个数组都是整型，结果也是整型，因此除法运算的结果都被取整。与加法类似，数组乘除运算实际也就是向量分量上的乘除：

select array\_agg(a \* b), array\_agg(a/b)

from (select unnest(array1) a, unnest(array2) b

from array\_tbl where id=1) t;

        结果：

array\_agg | array\_agg

----------------------------+---------------------

{9,16,21,24,25,24,21,16,9} | {0,0,0,0,1,1,2,4,9}

(1 row)

（6）计算数组点积。

select id, madlib.array\_dot(array1,array2)

from array\_tbl;

        结果：

id | array\_dot

----+-----------

1 | 165

2 | 750

(2 rows)

        两个向量u和v的点积u·v的定义为：https://img-blog.csdn.net/20171222164529277。也就是说，两个向量的点积用向量对应分量的乘积的和来计算，如下面的查询结果为750。

select sum(a \* b)

from (select unnest(array1) a, unnest(array2) b

from array\_tbl where id=2) t;

        由点积的定义，我们定义何谓两个向量正交。在欧式空间中，可以证明两个非零向量的点积为0当且仅当它们是垂直的。从几何角度，两个向量定义一个平面，并且它们的点积为0当且仅当这两个向量在平面内的夹角等于90 º。我们说这样的两个向量是正交的（orthogonal）。

（7）向量规范化

select madlib.normalize(array1) from array\_tbl;

        结果：

normalize

----------------------------------------------------------------------------------------------------------

{0.0592348877759092,0.118469775551818,0.177704663327728,0.236939551103637,0.296174438879546,0.355409326655455,0.41464421

4431365,0.473879102207274,0.533113989983183}

{0.0100600363590491,0.0100600363590491,0,0,0.0100600363590491,0.0201200727180982,0.0301801090771473,0.995943599545862,0.

0804802908723929}

(2 rows)

        点积也可以用来计算欧式空间中的向量长度：https://img-blog.csdn.net/20171222164930710。向量长度又称范数（norm）,并记作‖u‖。给定一个向量u，我们可以通过用其长度除u的每个分量，即通过计算u/‖u‖，找到一个向量，它与u指向相同的方向，但是具有单位长度。这称作将该向量规范化，具有范数1。根据规范化的定义，下面的查询与规范化函数结果相同：

select madlib.array\_scalar\_mult

(array1::float[],1/sqrt(madlib.array\_dot(array1, array1)))

from array\_tbl;

        并且可以使用下面的查询验证范数为1：

select id,sum(a)

from (select id,power(unnest(madlib.normalize(array1)),2) a from array\_tbl) t

group by id;

        给定向量范数，向量的点积也可以写成：u·v = ‖u‖‖v‖cos(θ)其中，θ是两个向量之间的夹角。把项分组并重新排列，上式可以改写成：https://img-blog.csdn.net/20171222165358145其中，https://img-blog.csdn.net/20171222165544532表示向量v在u的方向上的长度，如图2所示。如果u是单位向量，则该点积是v在u的方向上的分量。我们称它为v在u上的正交投影（orthogonal projection）。当然，如果v是单位向量，则该点积也是u在v方向上的投影。

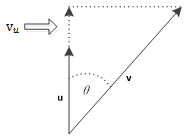


图2 向量v在向量u方向的正交投影

        一个与正交性密切相关的概念是线性独立性（linear independent）。如果一个向量集中的每个向量都不能表示成该集合中其它向量的线性组合，则该集合是线性独立的。如果一个向量集不是线性独立的，则它们是线性依赖的（linearly dependent）。我们希望基中每个向量都不线性依赖于其余的基向量。如果选择相互正交（独立的）基向量，则我们自动得到一个线性独立的基向量集，因为任意两个向量都正交的向量集是线性独立的。

 （8）构造一个9个元素的数组，数组元素的值设为1.3。

select madlib.array\_fill(madlib.array\_of\_float(9), 1.3::float);

        结果：

array\_fill

---------------------------------------

{1.3,1.3,1.3,1.3,1.3,1.3,1.3,1.3,1.3}

(1 row)

        array\_of\_float函数构造一个9个元素的数组，初始值为0，array\_fill填充数组元素值。array\_fill函数中第一个参数的数组元素数据类型需要与第二个参数的数据类型相同。

 （9）过滤掉数组中的指定元素

select madlib.array\_filter(array1),

madlib.array\_filter(array1,2),

madlib.array\_filter(array1,20)

from array\_tbl;

        结果：

array\_filter | array\_filter | array\_filter

---------------------+--------------------+----------------------

{1,2,3,4,5,6,7,8,9} | {1,3,4,5,6,7,8,9} | {1,2,3,4,5,6,7,8,9}

{1,1,1,2,3,99,8} | {1,1,0,0,1,3,99,8} | {1,1,0,0,1,2,3,99,8}

(2 rows)

        在没有给出第二个参数的情况下，madlib.array\_filter函数缺省过滤掉数组中的0元素，如果给出了第二个元素，则从第一个参数指定的数组中过滤掉该值。如果值在数组中不存在，则结果返回原数组。

（10）将二维数组列展开为一维数组集合。

        array\_unnest\_2d\_to\_1d是madlib 1.11版本的新增的函数，用于将二维数组展开为一维数组。1.10版本并无次函数，但可以创建一个UDF实现。

create or replace function madlib.array\_unnest\_2d\_to\_1d(anyarray)

returns table(unnest\_row\_id int,unnest\_result anyarray) as

$func$

select d1,array\_agg(val)

from (select $1[d1][d2] val,d1,d2

from generate\_series(array\_lower($1,1), array\_upper($1,1)) d1,

generate\_series(array\_lower($1,2), array\_upper($1,2)) d2

order by d1,d2) t

group by d1

$func$ language sql immutable;

        之后就可以调用函数展开二维数组：

select id,(madlib.array\_unnest\_2d\_to\_1d(val)).\*

from (select 1::int as id,

array[[1.3,2.0,3.2],[10.3,20.0,32.2]]::float8[][] as val

union all

select 2,

array[[pi(),pi()/2],[2\*pi(),pi()],[pi()/4,4\*pi()]]::float8[][])t

order by 1,2;

        结果：

id | unnest\_row\_id | unnest\_result

----+---------------+--------------------------------------

1 | 1 | {1.3,2,3.2}

1 | 2 | {10.3,20,32.2}

2 | 1 | {3.14159265358979,1.5707963267949}

2 | 2 | {6.28318530717959,3.14159265358979}

2 | 3 | {0.785398163397448,12.5663706143592}

(5 rows)

## 稀疏向量

        有些数据集，如具有非对称特征的数据集，一个对象的大部分属性上的值都为0，在许多情况下，非零项还不到1%。实际上，稀疏性（sparsity）是一个优点，因为只有非零值才需要存储和处理。这将节省大量的计算时间和存储空间。此外，有些数据挖掘算法仅适合处理稀疏数据。

### MADlib的稀疏向量

        MADlib的svec模块，实现了一种稀疏向量数据类型，能够为包含大量重复元素的向量提供压缩存储。浮点数组进行各种计算，有时会有很多的零或其它缺省值，在科学计算、零售优化、文本处理等应用中，这是很常见的。每个浮点数在内存或磁盘中占用8字节，节省多个零值的存储空间通常是有益的，而且，跳过零值对于很多向量计算也会提升性能。

        MADlib1.10版本仅支持float8稀疏向量类型，例如有如下float8[]数据类型的数组：

'{0, 33,...40000个0..., 12, 22 }'::float8[]

这个数组会占用320KB的内存或磁盘，而其中绝大部分存储的是0值。即使我们利用null位图，将0作为null存储，还是会得到一个5KB（40000/8）的null位图，内存使用效率还是不够高。何况在执行数组操作时，40000个零列上的计算结果并不重要。为了解决这个向量存储问题，svec类型使用行程长度编码（Run Length Encoding，RLE），即用一个数-值对数组表示稀疏向量。上面的数组以这种方式被存储为：

'{1,1,40000,1,1}:{0,33,0,12,22}'::madlib.svec

就是说1个0、1个33、40000个0等等，只使用5个整型和5个浮点数类型构成数组存储。除了节省空间，这种RLE表示也很容易实现向量操作，并使向量计算更快。svec模块提供了稀疏向量数据类型相关的函数库。

### 创建稀疏向量

        有以下四种方式可以创建稀疏向量。

 （1）直接使用常量表达式构建一个svec。

select'{n1,n2,...,nk}:{v1,v2,...vk}'::madlib.svec;

        其中n1、n2、...、nk分别指定值v1、v2、...、vk的个数，例如：

dm=# select'{1,3,5}:{2,4,6}'::madlib.svec;

svec

-----------------

{1,3,5}:{2,4,6}

(1 row)

（2）将一个float数组转换成svec。

select('{v1,v2,...vk}'::float[])::madlib.svec;

        例如：

dm=# select ('{2,4,4,4,6,6,6,6,6}'::float[])::madlib.svec;

svec

-----------------

{1,3,5}:{2,4,6}

(1 row)

（3）使用聚合函数创建一个svec，例如：

svec\_agg

----------------------------------------------

{1,1,1,1,1,1,1,1,1,1}:{1,2,3,4,5,6,7,8,9,10}

(1 row)

（4）利用madlib.svec\_cast\_positions\_float8arr()函数创建svec，例如：

dm=# select madlib.svec\_cast\_positions\_float8arr(array[1,3,5], array[2,4,6], 10, 0.0);

svec\_cast\_positions\_float8arr

-------------------------------

{1,1,1,1,1,5}:{2,0,4,0,6,0}

(1 row)

        此查询语句的含义是，生成一个10个元素的svec向量，其中1、3、5位置上的值分别是2、4、6，其它位置的值为0。svec模块的svec\_cast\_positions\_float8arr函数提供了从给定的位置数组和值数组声明一个稀疏向量的功能。下面再看一个例子：

dm=# select madlib.svec\_cast\_positions\_float8arr(

dm(# array[1,2,7,5,87],array[.1,.2,.7,.5,.87],90,0.0);

svec\_cast\_positions\_float8arr

-----------------------------------------------------

{1,1,2,1,1,1,79,1,3}:{0.1,0.2,0,0.5,0,0.7,0,0.87,0}

(1 row)

        第一个整数数组表示第二个浮点数数组的位置，即结果数组的第1、2、5、7、87下标对应的值分别为0.1、0.2、0.5、0.7、0.87。位置本身不需要有序，但要和值的顺序保持一致。第三个参数表示数组的最大维数。小于1最大维度将被忽略，此时数组的最大维度就是位置数组中的最大下标。最后的参数表示没有提供下标的位置上的值。

### 稀疏向量示例

（1）简单示例

        对svec类型可以应用<、>、\*、\*\*、/、=、+、SUM等操作和运算，并且具有典型的向量操作的相关含义。例如，加法（+）操作是对两个向量中相同下标对应的元素进行相加。为了使用svec模块中定义的运算符，需要将madlib模式添加到search\_path中。

dm=# -- 将madlib模式添加到搜索路径中

dm=# set search\_path="$user",public,madlib;

SET

dm=# -- 稀疏向量相加

dm=# select ('{0,1,5}'::float8[]::madlib.svec

dm(# + '{4,3,2}'::float8[]::madlib.svec)::float8[];

float8

---------

{4,4,7}

(1 row)

        如果最后不转换成float8[]，结果是一个svec类型：

dm=# select('{0,1,5}'::float8[]::madlib.svec

dm(# + '{4,3,2}'::float8[]::madlib.svec);

?column?

-------------

{2,1}:{4,7}

(1 row)

        两个向量的点积（%\*%）结果是float8类型，如(0\*4+ 1\*3 + 5\*2) = 13：

dm=# select'{0,1,5}'::float8[]::madlib.svec

dm-# %\*% '{4,3,2}'::float8[]::madlib.svec;

?column?

----------

13

(1 row)

        有些聚合函数对svec也是可用的，如svec\_count\_nonzero。

drop table if exists list;

create table list (a madlib.svec);

insert into list values

('{0,1,5}'::float8[]::madlib.svec),('{10,0,3}'::float8[]::madlib.svec),

('{0,0,3}'::float8[]::madlib.svec),('{0,1,0}'::float8[]::madlib.svec);

        svec\_count\_nonzero函数统计svec中每一列非0元素的个数，返回计数的svec。

dm=# select madlib.svec\_count\_nonzero(a)::float8[] from list;

svec\_count\_nonzero

--------------------

{1,2,3}

(1 row)

        svec数据类型中不应该使用NULL，因为NULL会显式表示为NVP（No Value Present）。

dm=# select'{1,2,3}:{4,null,5}'::madlib.svec;

svec

-------------------

{1,2,3}:{4,NVP,5}

(1 row)

        含有NULL的svec相加，结果中显示NVP。

dm=# select'{1,2,3}:{4,null,5}'::madlib.svec

dm-# + '{2,2,2}:{8,9,10}'::madlib.svec;

?column?

--------------------------

{1,2,1,2}:{12,NVP,14,15}

(1 row)

        可以使用svec\_proj()函数访问svec元素，该函数的参数为一个svec和一个元素下标。

dm=# select madlib.svec\_proj('{1,2,3}:{4,5,6}'::madlib.svec, 1)

dm-# + madlib.svec\_proj('{4,5,6}:{1,2,3}'::madlib.svec, 15);

?column?

----------

7

(1 row)

        通过svec\_subvec()函数可以访问一个svec的子向量，该函数的参数为一个svec，及其起止下标。

dm=# select madlib.svec\_subvec('{2,4,6}:{1,3,5}'::madlib.svec, 2, 11);

svec\_subvec

-----------------

{1,4,5}:{1,3,5}

(1 row)

        svec的元素/子向量可以通过svec\_change()函数进行改变。该函数有三个参数：一个m维的svec sv1，起始下标j，一个n维的svec sv2，其中j + n- 1 <= m，返回类似sv1的svec，但子向量sv1[j:j+n-1]被sv2所替换。

dm=# select madlib.svec\_change('{1,2,3}:{4,5,6}'

dm(# ::madlib.svec,3,'{2}:{3}'::madlib.svec);

svec\_change

---------------------

{1,1,2,2}:{4,5,3,6}

(1 row)

        还有处理svec的高阶函数，如svec\_lapply对应r语言中的lapply()函数。这里的所谓高阶函数，可以简单理解为函数（svec\_lapply）的参数是函数名（sqrt）。

dm=# select madlib.svec\_lapply('sqrt','{1,2,3}:{4,5,6}'::madlib.svec);

svec\_lapply

-----------------------------------------------

{1,2,3}:{2,2.23606797749979,2.44948974278318}

(1 row)

（2）扩展示例

        下面的示例是稀疏向量的一个具体应用，说明如何将文档转化为稀疏向量，并进一步对文档归类。假设有一个由若干单词组成的文本数组：

drop table if exists features;

create table features (a text[]);

insert into features values

('{am,before,being,bothered,corpus,document,i,in,is,me,

never,now,one,really,second,the,third,this,until}');

        同时有一个文档集合，每个文档表示为一个单词数组：

drop table if exists documents;

create table documents(a int,b text[]);

insert into documents values

(1,'{this,is,one,document,in,the,corpus}'),

(2,'{i,am,the,second,document,in,the,corpus}'),

(3,'{being,third,never,really,bothered,me,until,now}'),

(4,'{the,document,before,me,is,the,third,document}');

        如果忽略文档中词的顺序，则文档可以用词向量表示，其中每个词是向量的一个分量（属性），而每个分量的值对应词在文档中出现的次数。文档集合的这种表示通常称作文档-词矩阵（document-term matrix）文档是该矩阵的行，而词是矩阵的列。实践应用时，仅存放稀疏数据矩阵的非零项。

        现在有了字典和文档，我们要对每个文档中的出现单词的数量和比例应用向量运算，将文档进行分类。在开始处理前，需要找到每个文档中出现的字典中的单词。我们为每个文档创建一个稀疏特征向量（Sparse Feature Vector，SFV）。SFV是一个N维向量，N是字典单词的数量，SFV中的每个元素是文档中对每个字典单词的计数。svec模块中有一个函数可以从文档创建SFV：

dm=# select madlib.svec\_sfv((select a from features limit 1),b)::float8[]

dm-# from documents;

svec\_sfv

-----------------------------------------

{0,0,0,0,1,1,0,1,1,0,0,0,1,0,0,1,0,1,0}

{1,0,0,0,1,1,1,1,0,0,0,0,0,0,1,2,0,0,0}

{0,0,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,0,1}

{0,1,0,0,0,2,0,0,1,1,0,0,0,0,0,2,1,0,0}

(4 rows)

        注意，madlib.svec\_sfv()函数的输出是每个文档一个向量，元素值是相应字典顺序位置上单词在文档中出现的次数。通过对比特征向量和文档，更容易地理解这一点：

dm=# select madlib.svec\_sfv((select a from features),b)::float8[], b

dm-# from documents;

svec\_sfv | b

-----------------------------------------+--------------------------------------------------

{0,0,0,0,1,1,0,1,1,0,0,0,1,0,0,1,0,1,0} | {this,is,one,document,in,the,corpus}

{1,0,0,0,1,1,1,1,0,0,0,0,0,0,1,2,0,0,0} | {i,am,the,second,document,in,the,corpus}

{0,0,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,0,1} | {being,third,never,really,bothered,me,until,now}

{0,1,0,0,0,2,0,0,1,1,0,0,0,0,0,2,1,0,0} | {the,document,before,me,is,the,third,document}

(4 rows)

        可以看到文档"i am the second document in the corpus"，它的SFV为{1,3\*0,1,1,1,1,6\*0,1,2,3\*0}。单词“am”是字典中的第一个单词，并且在文档中只出现一次。单词“before”没有出现在文档中，所以它的值为0，以此类推。函数madlib.svec\_sfv()能够将大量文档高速并行转换为对应的SFV。

分类处理的其余部分都是向量运算。实际应用中很少使用实际计数值，而是将计数转为权重。最普通的权重叫做tf/idf，对应术语是Term Frequency / InverseDocument Frequency即词频-逆文件频率。对给定文档中给定单词的权重计算公式为：

{#Times in document} \* log {#Documents /#Documents the term appears in}

        例如，单词“document”在文档A中的权重为1\* log (4/3)，而在文档D中的权重为2 \* log(4/3)。在每个文档中都出现的单词的权重为0，因为log(4/4)= log(1) = 0，而仅出现在一个文档中的词具有最大权重log(文档数量)。TF-IDF是一种统计方法，用以评估一个词对于一个文件集或一个语料库中的其中一个文档重要程度。词的重要性随着它在文档中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。简单说就是，一个词在一篇文档中出现次数越多, 同时在所有文档中出现次数越少, 越能够代表该文章。

对于这部分处理，我们需要一个具有字典维数（19）的稀疏向量，元素值为：

log(#documents/#Documents each term appearsin)

        整个文档列表对应单一上述向量。#documents是文档总数，本例中是4，但对于每个字典单词都对应一个分母，其值为出现该单词的文档数。这个向量再乘以每个文档SFV中的计数，结果即为tf/idf权重。

drop table if exists corpus;

create table corpus

as (select a, madlib.svec\_sfv((select a from features),b) sfv

from documents);

drop table if exists weights;

create table weights

as (select a docnum, madlib.svec\_mult(sfv,logidf) tf\_idf

from (select madlib.svec\_log(madlib.svec\_div(

count(sfv)::madlib.svec,

madlib.svec\_count\_nonzero(sfv))) logidf

from corpus) foo, corpus order by docnum);

        查询权重：

dm=# select \* from weights;

docnum | tf\_idf

--------+----------------------------------------------------------------------------------------------------------------

------------------------------------------

1 | {4,1,1,1,2,3,1,2,1,1,1,1}:{0,0.693147180559945,0.287682072451781,0,0.693147180559945,0,1.38629436111989,0,0.287

682072451781,0,1.38629436111989,0}

2 | {1,3,1,1,1,1,6,1,1,3}:{1.38629436111989,0,0.693147180559945,0.287682072451781,1.38629436111989,0.69314718055994

5,0,1.38629436111989,0.575364144903562,0}

3 | {2,2,5,1,2,1,1,2,1,1,1}:{0,1.38629436111989,0,0.693147180559945,1.38629436111989,0,1.38629436111989,0,0.6931471

80559945,0,1.38629436111989}

4 | {1,1,3,1,2,2,5,1,1,2}:{0,1.38629436111989,0,0.575364144903562,0,0.693147180559945,0,0.575364144903562,0.6931471

80559945,0}

(4 rows)

        尽管文档具有具有数以百千计或数以万计的属性（词），但是每个文档向量都是稀疏的，因为它具有相对较少的非零属性值。这样，相似性不能依赖共享0的个数，因为任意两个文档多半不会包含许多相同的词，从而如果统计0-0匹配，则大多数文档都与其它大部分文档非常类似。因此文档的相似性度量需要忽略0-0匹配，而且还必须能处理非二元向量。下面定义的余弦相似度（cosinesimilarity）就是文档相似性最常用的度量之一。如果x和y是两个文档向量，则：

https://img-blog.csdn.net/20171222183520223

        现在就可以使用文档向量的点积的ACOS，获得一个文档与其它文档的“角距离”。下面计算第一个文档与其它文档的角距离：

dm=# select docnum, 180. \*

dm-# (acos(madlib.svec\_dmin(1., madlib.svec\_dot(tf\_idf, testdoc)

dm(# / (madlib.svec\_l2norm(tf\_idf)

dm(# \* madlib.svec\_l2norm(testdoc))))/3.141592654) angular\_distance

dm-# from weights,

dm-# (select tf\_idf testdoc from weights where docnum = 1 limit 1) foo

dm-# order by 1;

docnum | angular\_distance

--------+------------------

1 | 0

2 | 78.8235846096986

3 | 89.9999999882484

4 | 80.0232034288617

(4 rows)

        可以看到文档1与自己的角距离为0度，而文档1与文档3的角距离为90度，因为它们之间没有任何相同的单词。

## 向量与数据分析

        尽管最初引进向量是为了处理力、速度、加速度这样的量，但是实践证明它们也能用来表示和理解许多其它类型的数据。特别是，我们常常把一个数据对象或属性看作向量。例如，文档可以用向量表示，其中每个分量对应一个词，而每个分量的值是该词在文档中出现的次数。这会产生非常稀疏、高维的向量。这里，稀疏是指向量的大部分分量为0。

        一旦我们用向量表示数据对象，我们就可以在数据上执行各种向量计算。例如，我们可以计算两个向量的余弦相似度或距离。这种相似性度量不考虑向量的量值（长度），而只考虑两个向量在相同方向的程度。就文档而言，这意味如果两个文档以相同的比例包含相同词，则它们相同。两个文档中都不出现的词在计算相似性时不起作用。

        我们还可以简单地定义两个向量（点）之间的距离。如果u和v是向量，则这两个向量（点）之间的欧几里得距离简单地定义为：

https://img-blog.csdn.net/20171222183917885

如果在考虑相似性时，向量各分量的量值确实很重要，这种度量更合适。

        对于向量数据，计算向量集的均值也有意义。向量集的均值通过计算每个分量的均值实现。确实，有些聚类方法，如K-均值就是将数据对象划分成组（簇），并用数据对象（数据向量）的均值刻画每个簇。其基本思想是，簇中数据对象靠近均值的簇是好的簇。其中，对于象鸢尾花（最流行的机器学习数据集之一）这样的数据集，相似性用欧氏距离度量，而对于象文档这样的数据集用余弦相似性度量。

        数据上的其它常用运算也可以看作向量上的运算。考虑降维操作，在最简单的方法中，数据向量中的某些分量被删除，而保留其它分量不变。有些降维技术产生数据向量的新的分量（属性）集，这些新分量是原分量的线性组合。

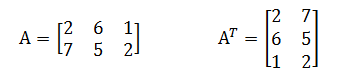
        对于某些数据分析领域（如统计学），分析技术用数据向量和包含这些数据向量的数据矩阵上的运算数学地表达。这样，向量表示带动了可以用来表示、变换和分析数据的强有力的数学工具。

# 数据类型之矩阵

 矩阵可以用来表示数据集，描述数据集上的变换，是MADlib中数据的基本格式，通常使用二维数组数据类型存储。MADlib中的向量是一维数组，可看作是矩阵的一种特殊形式。MADlib的矩阵运算模块（matrix\_ops）实现SQL中的矩阵操作。本篇介绍矩阵的概念，说明MADlib矩阵运算相关函数，并举出一些简单的函数调用示例。

## 矩阵定义

矩阵（matrix）是把数集合汇聚成行和列的一种表表示。术语https://img-blog.csdn.net/20171226173423445通常用来说明矩阵具有m行和n列。例如，下面所示的矩阵A是https://img-blog.csdn.net/20171226173534880。如果m=n，则我们称该矩阵为方阵（square matrix）。矩阵A的转置记作https://img-blog.csdn.net/20171226173701813，它通过交换A的行和列得到。



矩阵的元素用带小标的小写字母表示。例如，对于矩阵A，https://img-blog.csdn.net/20171226173916234是其第 i 行第 j 列的元素。行[自上而下](https://www.baidu.com/s?wd=%E8%87%AA%E4%B8%8A%E8%80%8C%E4%B8%8B&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)编号，列自左向右编号，编号从1开始。例如，https://img-blog.csdn.net/20171226174024469是矩阵A的第2行第1列的元素。

矩阵的每一行或列定义一个向量。对于矩阵A，其第 i 个行向量（row vector）可以用https://img-blog.csdn.net/20171227090814479表示，而第 j 个列向量（column vector）用https://img-blog.csdn.net/20171227090930339表示。使用前面的例子，https://img-blog.csdn.net/20171226174132174，而https://img-blog.csdn.net/20171226174304143。注意：行向量和列向量都是矩阵，必须加以区分，即元素个数相同并且值相同的行向量和列向量代表不同的矩阵。

## MADlib中的矩阵表示

        MADlib支持稠密和稀疏两种矩阵表示形式，所有矩阵运算都以任一种表示形式工作。

### 稠密

        矩阵被表示为一维数组的行集合，例如3x10的矩阵如下表：

row\_id | row\_vec

-------+-------------------------

1 | {9,6,5,8,5,6,6,3,10,8}

2 | {8,2,2,6,6,10,2,1,9,9}

3 | {3,9,9,9,8,6,3,9,5,6}

        row\_id列表示每一行的行号，是从1到N没有重复值的连续整型序列，N为矩阵的行数。row\_vec列对应构成矩阵每行的一维数组（行向量）。

### 稀疏

        使用行列下标指示矩阵中每一个非零项，例如：

row\_id | col\_id | value

-------+--------+-------

1 | 1 | 9

1 | 5 | 6

1 | 6 | 6

2 | 1 | 8

3 | 1 | 3

3 | 2 | 9

4 | 7 | 0

        常用这种方式表示包含多个零元素的稀疏矩阵。上面的例子只用6行表示一个4x7的矩阵中的非零元素。矩阵的行列元素个数分别由row\_id和col\_id的最大值指定。注意最后一行，即使value为0也要包含此行，它指出了矩阵的维度，而且指示矩阵的第4行与第7列的元素值都是0。

        对于稀疏矩阵表，row\_id和col\_id列逻辑类似于关系数据库的联合主键，要求非空且唯一。value列应该是标量（非数组）数据类型。上面矩阵对应的稠密表示如下：

row\_id | row\_vec

-------+-------------------------

1 | {9,0,0,0,6,6,0}

2 | {8,0,0,0,0,0,0}

3 | {3,9,0,0,0,0,0}

4 | {0,0,0,0,0,0,0}

## MADlib中的矩阵运算函数

        与数组操作相同，矩阵运算函数支持的元素数据类型也包括SMALLINT、INTEGER、BIGINT、FLOAT8和NUMERIC（内部被转化为FLOAT8，可能丢失精度）。

### 矩阵操作函数分类

        MADlib的矩阵操作函数可分为表示、计算、提取、归约、创建、转换六类。下面列出每一类中所包含的函数名称及其参数。

 （1）表示函数

-- 转为稀疏矩阵

matrix\_sparsify( matrix\_in, in\_args,matrix\_out, out\_args)

-- 转为稠密矩阵

matrix\_densify( matrix\_in, in\_args,matrix\_out, out\_args)

-- 获取矩阵的维度

matrix\_ndims( matrix\_in, in\_args )

（2）计算函数

-- 矩阵转置

matrix\_trans( matrix\_in, in\_args,matrix\_out, out\_args)

-- 矩阵相加

matrix\_add( matrix\_a, a\_args, matrix\_b,b\_args, matrix\_out, out\_args)

-- 矩阵相减

matrix\_sub( matrix\_a, a\_args, matrix\_b,b\_args, matrix\_out, out\_args)

-- 矩阵乘法

matrix\_mult( matrix\_a, a\_args, matrix\_b,b\_args, matrix\_out, out\_args)

-- 数组元素相乘

matrix\_elem\_mult( matrix\_a, a\_args,matrix\_b, b\_args, matrix\_out, out\_args)

-- 标量乘矩阵

matrix\_scalar\_mult( matrix\_in, in\_args,scalar, matrix\_out, out\_args)

-- 向量乘矩阵

matrix\_vec\_mult( matrix\_in, in\_args,vector)

（3）提取函数

-- 从行下标提取行

matrix\_extract\_row( matrix\_in, in\_args,index)

-- 从列下标提取列

matrix\_extract\_col( matrix\_in, in\_args,index)

-- 提取主对角线元素

matrix\_extract\_diag( matrix\_in, in\_args)

（4）归约函数（指定维度的聚合）

-- 获取指定维度的最大值。如果fetch\_index = True，返回对应的下标。

matrix\_max( matrix\_in, in\_args, dim,matrix\_out, fetch\_index)

-- 获取指定维度的最小值。如果fetch\_index = True，返回对应的下标。

matrix\_min( matrix\_in, in\_args, dim,matrix\_out, fetch\_index)

-- 获取指定维度的和

matrix\_sum( matrix\_in, in\_args, dim)

-- 获取指定维度的均值

matrix\_mean( matrix\_in, in\_args, dim)

-- 获取矩阵范数

matrix\_norm( matrix\_in, in\_args, norm\_type)

（5）创建函数

-- 创建一个指定行列维度的矩阵，用1初始化元素值。

matrix\_ones( row\_dim, col\_dim, matrix\_out,out\_args)

-- 创建一个指定行列维度的矩阵，用0初始化元素值。

matrix\_zeros( row\_dim, col\_dim, matrix\_out,out\_args)

-- 创建单位矩阵

matrix\_identity( dim, matrix\_out, out\_args)

-- 用给定对角元素初始化矩阵

matrix\_diag( diag\_elements, matrix\_out,out\_args)

（6）转换函数

-- 矩阵求逆

matrix\_inverse( matrix\_in, in\_args,matrix\_out, out\_args)

-- 广义逆矩阵

matrix\_pinv( matrix\_in, in\_args,matrix\_out, out\_args)

-- 矩阵特征提取

matrix\_eigen( matrix\_in, in\_args,matrix\_out, out\_args)

-- Cholesky分解

matrix\_cholesky( matrix\_in, in\_args,matrix\_out\_prefix, out\_args)

-- QR分解

matrix\_qr( matrix\_in, in\_args,matrix\_out\_prefix, out\_args)

-- LU分解

matrix\_lu( matrix\_in, in\_args, matrix\_out\_prefix,out\_args)

-- 求矩阵的核范数

matrix\_nuclear\_norm( matrix\_in, in\_args)

-- 求矩阵的秩

matrix\_rank( matrix\_in, in\_args)

        注意：矩阵转换函数仅基于内存操作实现。单一节点的矩阵数据被用于分解计算。这种操作只适合小型矩阵，因为计算不是分布到个多个节点执行的。

### 矩阵操作函数示例

        先执行下面的脚本创建两个稠密表示的矩阵测试表并添加数据。mat\_a矩阵4行4列，mat\_b矩阵5行4列。

drop table if exists mat\_a;

create table mat\_a (row\_id integer, row\_vec integer[]);

insert into mat\_a (row\_id, row\_vec)values

(1, '{9,6,5,8}'), (2, '{8,2,2,6}'), (3,'{3,9,9,9}'), (4, '{6,4,2,2}');

drop table if exists mat\_b;

create table mat\_b (row\_id integer, vector integer[]);

insert into mat\_b (row\_id, vector)values

(1, '{9,10,2,4}'), (2, '{5,3,5,2}'), (3,'{0,1,2,3}'), (4, '{2,9,0,4}'), (5,'{3,8,7,7}');

（1）由稠密矩阵表生成稀疏表示的表

drop table if exists mat\_a\_sparse;

select madlib.matrix\_sparsify('mat\_a','row=row\_id, val=row\_vec',

'mat\_a\_sparse','col=col\_id, val=val');

drop table if exists mat\_b\_sparse;

select madlib.matrix\_sparsify('mat\_b','row=row\_id, val=vector',

'mat\_b\_sparse','col=col\_id, val=val');

        madlib.matrix\_sparsify函数将稠密表示矩阵表转为稀疏表示的矩阵表，四个参数分别指定输入表名、输入表参数（代表行ID的列名、存储矩阵元素值的列名等）、输出表名、输出表参数（代表列ID的列名、存储矩阵元素值的列名等）。

        上面的例子将稠密矩阵转为稀疏表示，并新建表存储转换结果。源表的两列类型分别是整型和整型数组，输出表包含三列，行ID列名与源表相同，列ID列和值列由参数指定。由于mat\_a表的矩阵中不存在0值元素，生成的稀疏矩阵表共有16条记录，而mat\_b中有两个0值，因此稀疏表中只有18条记录。

dm=# select \* from mat\_a\_sparse order by row\_id, col\_id;

row\_id | col\_id | val

--------+--------+-----

1 | 1 | 9

1 | 2 | 6

…

4 | 3 | 2

4 | 4 | 2

(16 rows)

dm=# select \* from mat\_b\_sparse;

row\_id | col\_id | val

--------+--------+-----

1 | 1 | 9

1 | 2 | 10

…

4 | 2 | 9

4 | 4 | 4

(18 rows)

（2）矩阵转置

        matrix\_trans函数的第一个参数是源表名，第二个参数指定行、列或值的字段名，第三个参数为输出表名。

-- 稠密格式

drop table if exists mat\_a\_r;

select madlib.matrix\_trans('mat\_a','row=row\_id, val=row\_vec','mat\_a\_r');

select \* from mat\_a\_r order by row\_id;

        结果：

row\_id | row\_vec

--------+-----------

1 | {9,8,3,6}

2 | {6,2,9,4}

3 | {5,2,9,2}

4 | {8,6,9,2}

(4 rows)

-- 稀疏格式

drop table if exists mat\_b\_sparse\_r;

select madlib.matrix\_trans('mat\_b\_sparse', 'row=row\_id, col=col\_id, val=val','mat\_b\_sparse\_r');

select \* from mat\_b\_sparse\_r order byrow\_id, col\_id;

       结果：

col\_id | row\_id | val

--------+--------+-----

1 | 1 | 9

2 | 1 | 5

…

4 | 4 | 4

5 | 4 | 7

(18 rows)

        源矩阵5行4列，转置后的矩阵为4行5列。

 （3）提取矩阵的主对角线

select madlib.matrix\_extract\_diag('mat\_b', 'row=row\_id, val=vector'),

madlib.matrix\_extract\_diag

('mat\_b\_sparse\_r', 'row=row\_id, col=col\_id,val=val');

        结果：

matrix\_extract\_diag | matrix\_extract\_diag

---------------------+---------------------

{9,3,2,4} | {9,3,2,4}

(1 row)

        matrix\_extract\_diag函数的返回值是由对角线元素组成的数组。可以看到，矩阵和其对应的转置矩阵具有相同的主对角线。也就是说，矩阵转置实际上是沿着主对角线的元素对折操作。

 （4）提取指定下标的行或列

select madlib.matrix\_extract\_row('mat\_a','row=row\_id, val=row\_vec', 2) as row,

madlib.matrix\_extract\_col

('mat\_b\_sparse','row=row\_id, col=col\_id, val=val', 3) as col;

        结果返回两个向量，即mat\_a的第2行，mat\_b\_sparse的第3列：

row | col

-----------+-------------

{8,2,2,6} | {2,5,2,0,7}

(1 row)

（5）获取指定维度的最大最小值及其对应的下标

drop table if exists mat\_max\_r,mat\_min\_r;

select madlib.matrix\_max

('mat\_a','row=row\_id, val=row\_vec', 2, 'mat\_max\_r', true),

madlib.matrix\_min

('mat\_b\_sparse','row=row\_id, col=col\_id', 1, 'mat\_min\_r', true);

select \* from mat\_max\_r, mat\_min\_r;

        结果：

index | max | index | min

-----------+-----------+-----------+-----------

{1,1,2,1} | {9,8,9,6} | {3,3,4,2} | {0,1,0,2}

(1 row)

        matrix\_max和matrix\_min函数分别返回指定维度的最大值和最小值，其中维度参数的取值只能是1或2，分别代表行和列。返回值为数组类型，如果最后一个参数为‘true’，表示结果表中包含最大最小值对应的下标数组列。

（6）按指定维度求和

select madlib.matrix\_sum('mat\_b\_sparse', 'row=row\_id, col=col\_id,val=val', 1),

    madlib.matrix\_sum('mat\_b\_sparse', 'row=row\_id, col=col\_id,val=val', 2);

        结果：

matrix\_sum | matrix\_sum

---------------+-----------------

{19,31,16,20} | {25,15,6,15,25}

(1 row)

matrix\_sum函数按指定维度求和，第三个参数的值只能是1或2，分别表示按行或列求和。函数返回的结果是一个向量。

（7）按指定维度求均值

select madlib.matrix\_mean('mat\_b\_sparse', 'row=row\_id, col=col\_id,val=val', 1),

madlib.matrix\_mean('mat\_b\_sparse', 'row=row\_id, col=col\_id,val=val', 2);

        结果：

matrix\_mean | matrix\_mean

-----------------+---------------------------

{3.8,6.2,3.2,4} | {6.25,3.75,1.5,3.75,6.25}

(1 row)

        matrix\_mean函数按指定维度求均值，第三个参数的值只能是1或2，分别表示行或列。函数返回的结果是一个向量。

（8）创建对角矩阵

drop table if exists mat\_r;

select madlib.matrix\_diag(array[9,6,3,10],

'mat\_r', 'row=row\_id,col=col\_id, val=val');

select \* from mat\_r order by row\_id;

        结果：

row\_id | col\_id | val

--------+--------+-----

1 | 1 | 9

2 | 2 | 6

3 | 3 | 3

4 | 4 | 10

(4 rows)

        madlib.matrix\_diag函数输出的是一个稀疏表示的对角矩阵表，如果不指定“col=col\_id”，输出表中代表列的列名为col。

 （9）创建单位矩阵

drop table if exists mat\_r;

select madlib.matrix\_identity(4,'mat\_r');

select \* from mat\_r

        结果：

row | col | val

-----+-----+-----

4| 4 | 1

2| 2 | 1

1| 1 | 1

3| 3 | 1

(4 rows)

        matrix\_identity函数创建一个稀疏表示的单位矩阵表。主对角线上的元素都为1，其余元素全为0的方阵称为单位矩阵。

 （10）创建元素为全0的矩阵

drop table if exists mat\_r01, mat\_r02;

select madlib.matrix\_zeros(3, 2, 'mat\_r01','row=row\_id, col=col\_id, val=entry'),

madlib.matrix\_zeros(3, 2, 'mat\_r02', 'fmt=dense');

select \* from mat\_r01;

select \* from mat\_r02;

        结果分别为：

row\_id | col\_id | entry

--------+--------+-------

3 | 2 | 0

(1 row)

row| val

-----+-------

1| {0,0}

3| {0,0}

2| {0,0}

(3 rows)

        注意因为元素值全为0，稀疏表示的矩阵表只有1行。fmt=dense指示结果表是稠密格式。

 （11）创建元素为全1的矩阵

drop table if exists mat\_r11, mat\_r12;

select madlib.matrix\_ones(3, 2, 'mat\_r11','row=row\_id, col=col\_id, val=entry'),

madlib.matrix\_ones(3, 2, 'mat\_r12', 'fmt=dense');

select \* from mat\_r11 order by row\_id;

select \* from mat\_r12 order by row;

        结果分别为：

row\_id | col\_id | entry

--------+--------+-------

1 | 2 | 1

1 | 1 | 1

2 | 2 | 1

2 | 1 | 1

3 | 2 | 1

3 | 1 | 1

(6 rows)

row| val

-----+-------

1| {1,1}

2| {1,1}

3| {1,1}

(3 rows)

        注意因为元素值全为1，稀疏表示的矩阵表有6行。

 （12）获取行列维度数

select madlib.matrix\_ndims('mat\_a','row=row\_id, val=row\_vec'),

madlib.matrix\_ndims('mat\_a\_sparse', 'row=row\_id, col=col\_id');

        结果：

matrix\_ndims | matrix\_ndims

--------------+--------------

{4,4} | {4,4}

(1 row)

（13）矩阵相加

        与向量一样，矩阵也可以通过将对应元素（分量）相加来求和。MADlib的矩阵相加函数要求两个矩阵具有相同的行数和列数。更明确地说，假定A和B都是mXn的矩阵，A和B的和是mXn矩阵C，其元素由下式计算：https://img-blog.csdn.net/20171226180536803

drop table if exists mat\_r;

select madlib.matrix\_add('mat\_b','row=row\_id, val=vector',

'mat\_b\_sparse','row=row\_id, col=col\_id',

'mat\_r', 'val=vector,fmt=dense');

select \* from mat\_r order by row\_id;

        结果：

row\_id | vector

--------+--------------

1 | {18,20,4,8}

2 | {10,6,10,4}

3 | {0,2,4,6}

4 | {4,18,0,8}

5 | {6,16,14,14}

(5 rows)

        madlib.matrix\_add函数有三组参数，分别指示两个相加的矩阵表和结果矩阵表。相加的两个矩阵表不必有相同的表示形式，如上面的函数调用中，两个矩阵一个为稠密形式，一个为稀疏形式。但两个矩阵必须具有相同的行列数，否则会报如下错误：

Matrix error: The dimensions of the twomatrices don't match

        矩阵加法具有如下性质。

* 矩阵加法的交换律。加的次序不影响结果：A + B = B + A。
* 矩阵加法的结合律。相加时矩阵分组不影响结果：(A + B) + C = A + (B + C)。
* 矩阵加法单位元的存在性。存在一个零矩阵（zero matrix），其元素均为0并简记为0，是单位元。对于任意矩阵A，有A +0 = A。
* 矩阵加法逆元的存在性。对于每个矩阵A，都存在一个矩阵-A，使得A +(-A) = 0。-A的元素为https://img-blog.csdn.net/20171226180817058。

**（14）标量与矩阵相乘**

        与向量一样，也可以用标量乘以矩阵。标量α和矩阵A的乘积是矩阵B =αA，其元素由下式给出：https://img-blog.csdn.net/20171226180934579

        如下面matrix\_scalar\_mult函数执行结果是由原矩阵的每个元素乘以3构成的矩阵表。

drop table if exists mat\_r;

select madlib.matrix\_scalar\_mult('mat\_a','row=row\_id, val=row\_vec', 3, 'mat\_r');

select \* from mat\_r order by row\_id;

结果：

row\_id | row\_vec

--------+---------------

1 | {27,18,15,24}

2 | {24,6,6,18}

3 | {9,27,27,27}

4 | {18,12,6,6}

(4 rows)

        矩阵的标量乘法具有与向量的标量乘法非常相似的性质。

* 标量乘法的结合律。被两个标量乘的次序不影响结果：α(βA) = (αβ)A。
* 标量加法对标量与矩阵乘法的分配率。两个标量相加后乘以一个矩阵等于每个标量乘以该矩阵之后的结果矩阵相加：(α+β)A =αA +βA。
* 标量乘法对矩阵加法的分配率。两个矩阵相加之后的和与一个标量相乘等于每个矩阵与该标量相乘然后相加：α(A + B)=αA + αB。
* 标量单位元的存在性。如果α=1，则对于任意矩阵A，有αA =A。

         我们可以认为矩阵由行向量或列向量组成，因此矩阵相加或用标量乘以矩阵等于对应行向量或列向量相加或用标量乘它们。

**（15）矩阵乘法**

        我们可以定义矩阵的乘法运算。先定义矩阵与向量的乘法。

        矩阵与列向量的乘法mXn矩阵A乘以nX1的列矩阵u的积是mX1的列矩阵 v=Au，其元素由下式给出：https://img-blog.csdn.net/20171226181242394

        换言之，我们取A的每个行向量与u的转置的点积。注意，在下面的例子中，u的行数必然与A的列数相等。

https://img-blog.csdn.net/20171226181330862

        类似地，我们可以定义矩阵被行向量左乘。

        矩阵与行向量的乘法1Xm的行矩阵u乘以mXn矩阵A的积是1Xn的行矩阵v=uA，其元素由下式给出：https://img-blog.csdn.net/20171226181444417

        换言之，我们取该行向量与矩阵A的每个列向量的转置的点积。下面给出一个例子：

https://img-blog.csdn.net/20171226181516276

        MADlib的matrix\_vec\_mult函数用于计算一个mXn矩阵乘以一个1Xn的矩阵（向量），结果是一个1Xm的矩阵。如下面的5X4的矩阵mat\_b乘以一个1X4的矩阵，结果一个1X5的矩阵。

dm=# select \* from mat\_b;

row\_id | vector

--------+------------

1 | {9,10,2,4}

2 | {5,3,5,2}

3 | {0,1,2,3}

4 | {2,9,0,4}

5 | {3,8,7,7}

(5 rows)

dm=# select madlib.matrix\_vec\_mult('mat\_b','row=row\_id, val=vector',

dm(# array[1,2,3,4]);

matrix\_vec\_mult

------------------

{51,34,20,36,68}

(1 row)

        可以用下面的查询验证矩阵乘以向量的结果。

dm=# select array\_agg(madlib.array\_dot(vector,array[1,2,3,4])) from mat\_b;

array\_agg

------------------

{51,34,20,36,68}

(1 row)

        我们定义两个矩阵的乘积，作为上述概念的推广。mXn矩阵A与nXp矩阵B的积是mXp矩阵C=AB，其元素由下式给出：https://img-blog.csdn.net/20171226181820991

        换言之，C的第 ij 个元素是A的第 i 个行向量与B的第 j 个列向量转置的点积。

        matrix\_mult函数用于矩阵相乘。如前所述，第一组参数中的矩阵列数应该与第二组参数中的矩阵行数相同，否则会报错：

dm=# select \* from mat\_a;

row\_id | row\_vec

--------+-----------

1 | {9,6,5,8}

2 | {8,2,2,6}

3 | {3,9,9,9}

4 | {6,4,2,2}

(4 rows)

dm=# select \* from mat\_b;

row\_id | vector

--------+------------

1 | {9,10,2,4}

2 | {5,3,5,2}

3 | {0,1,2,3}

4 | {2,9,0,4}

5 | {3,8,7,7}

(5 rows)

dm=# drop table if exists mat\_r;

NOTICE: table"mat\_r" does not exist, skipping

DROP TABLE

dm=# select madlib.matrix\_mult('mat\_a', 'row=row\_id,val=row\_vec',

dm(# 'mat\_b', 'row=row\_id, val=vector',

dm(# 'mat\_r');

ERROR: plpy.Error: Matrixerror: Dimension mismatch for matrix multiplication. (plpython.c:4663)

DETAIL: Left matrix, coldimension = 4, Right matrix, row dimension = 5

CONTEXT: Traceback (mostrecent call last):

PL/Python function"matrix\_mult", line 26, in <module>

matrix\_out, out\_args)

PL/Python function"matrix\_mult", line 1633, in matrix\_mult

PL/Python function"matrix\_mult", line 49, in \_assert

PL/Python function "matrix\_mult"

dm=#

        可以对mat\_b先进行转置，再与mat\_a相乘。matrix\_mult 函数调用时的trans=true参数表示先对mat\_b表行列转置再进行矩阵乘法。这次的矩阵乘法计算将正常执行。

drop table if exists mat\_r;

select madlib.matrix\_mult('mat\_a', 'row=row\_id,val=row\_vec',

'mat\_b', 'row=row\_id, val=vector, trans=true',

'mat\_r');

select \* from mat\_r order by row\_id;

        结果是一个4X5矩阵：

row\_id | row\_vec

--------+----------------------

1 |{183,104,40,104,166}

2 |{120,68,24,58,96}

3 |{171,105,54,123,207}

4 |{106,56,14,56,78}

(4 rows)

        执行结果与下面的查询相同。

drop table if exists mat\_r;

select madlib.matrix\_mult('mat\_a', 'row=row\_id,val=row\_vec',

'mat\_b\_sparse\_r', 'row=row\_id, col=col\_id, val=val',

'mat\_r');

select \* from mat\_r order by row\_id;

        矩阵乘法具有如下性质。

* 矩阵乘法的结合律。矩阵乘的次序不影响计算结果：(AB)C=A(BC)。
* 矩阵乘法的分配率。矩阵乘法对矩阵加法是可分配的：A(B+ C) = AB + AC并且(B + C)A = BA + CA。
* 矩阵乘法单位元的存在性。如果https://img-blog.csdn.net/20171226182436239是pXp矩阵的单位矩阵，则对于任意mXn矩阵A，https://img-blog.csdn.net/20171226182514671并且https://img-blog.csdn.net/20171226182550895。

         一般地，矩阵乘法是不可交换的，即https://img-blog.csdn.net/20171226182656424。

        如果我们有一个nX1列向量u，则我们可以把mXn矩阵A被该向量右乘看作u到m维列向量v=Au的变换。类似地，如果我们用一个（行）向量https://img-blog.csdn.net/20171226182803417左乘A，则我们可以将它看作u到n维行向量v=uA的变换。这样，我们可以把一个任意mXn矩阵A看作一个把一个向量映射到另一个向量空间的函数。

        在许多情况下，可以用更容易理解的术语描述变换矩阵。

* 缩放矩阵（scaling matrix）不改变向量的方向，而是改变向量的长度。这等价于乘以一个乘了标量的单位矩阵得到的矩阵。
* 旋转矩阵（rotation matrix）改变向量的方向但不改变向量的量值。这相当于改变坐标系。
* 反射矩阵（reflection matrix）将一个向量从一个或多个坐标轴反射。这等价于用-1乘该向量的某些元素，而保持其它元素不变。
* 投影矩阵（projection matrix）把向量置于较低维子空间。最简单的例子是修改单位矩阵，将对角线上的一个或多个1改为0。这样的矩阵消除对应于0元素的向量分量，而保留其它分量。

         当然，单个矩阵可能同时进行两种类型的变换，如缩放和旋转。

 （16）两矩阵元素相乘

        与矩阵乘法定义不同，MADlib的两矩阵元素相乘定义为C=AB，A、B、C均为mXn矩阵，C的元素由下式给出：https://img-blog.csdn.net/20171226183110986

        MADlib的matrix\_elem\_mult函数执行两矩阵元素相乘，并输出结果矩阵。

drop table if exists mat\_r;

select madlib.matrix\_elem\_mult('mat\_b','row=row\_id, val=vector',

'mat\_b\_sparse','row=row\_id, col=col\_id, val=val',

'mat\_r','fmt=dense');

select \* from mat\_r order by row\_id;

        结果：

row\_id | vector

--------+---------------

1 | {81,100,4,16}

2 | {25,9,25,4}

3 | {0,1,4,9}

4 | {4,81,0,16}

5 | {9,64,49,49}

(5 rows)

（17）求矩阵的秩

select madlib.matrix\_rank('mat\_a','row=row\_id, val=row\_vec');

        结果：

matrix\_rank

-------------

4

(1 row)

        注意，当矩阵以稀疏形式表示，并且列数大于行数时，matrix\_rank函数会报错。

dm=#select madlib.matrix\_rank('mat\_b\_sparse\_r', 'row=row\_id, col=col\_id, val=val');

ERROR: plpy.SPIError: Function"madlib.\_\_matrix\_compose\_sparse\_transition(doubleprecision[],integer,integer,integer,integer,double precision)": Invalidcol id. (UDF\_impl.hpp:210) (seg20hdp4:40000 pid=123035) (plpython.c:4663)

CONTEXT: Traceback (most recent call last):

PL/Python function "matrix\_rank",line 23, in <module>

returnmatrix\_ops.matrix\_rank(schema\_madlib, matrix\_in, in\_args)

PL/Python function "matrix\_rank",line 2702, in matrix\_rank

PL/Python function "matrix\_rank",line 2672, in matrix\_eval\_helper

PL/Pythonfunction "matrix\_rank"

dm=#

        矩阵的秩（rank of a matrix）常常用来刻画矩阵。设矩阵https://img-blog.csdn.net/20171226183451738，在A中任取 k 行 k 列交叉处元素按原相对位置组成的 k 阶行列式，称为A的一个 k 阶子式。mXn矩阵A共有https://img-blog.csdn.net/20171226183604743个 k 阶子式。若A有 r 阶子式不为0，任何 r+1 阶子式（如果存在的话）全为0，称 r 为矩阵A的秩，记作R(A)。

        矩阵的秩具有以下基本性质：

* 0矩阵的秩为0。
* 如果R(A)=r，则A中至少有一个 r 阶子式https://img-blog.csdn.net/20171227103433959，所有 r+1 阶子式为0，且更高阶子式均为0，r 是A中非零的子式的最高阶数。
* 矩阵转置，秩不变。0<=R(A)<=min(m,n)。
* 如果A是nXn方阵，并且|A|≠0，则R(A)=n；反之，如果R(A)=n，则|A|≠0。

        矩阵的秩是行空间和列空间的最小维度，此维度中的向量组是线性无关的。例如，如果把一个1Xn的行向量复制m次，产生一个mXn的矩阵，则我们只有一个秩为1的矩阵。

 （18）求逆矩阵

drop table if exists mat\_r;

select madlib.matrix\_inverse('mat\_a','row=row\_id, val=row\_vec', 'mat\_r');

select row\_vec from mat\_r order byrow\_id;

        结果：

row\_vec

--------------------------------------------------------------

{-1.2,0.900000000000001,0.333333333333334,0.600000000000001}

{3.20000000000001,-2.4,-1,-1.1}

{-5.00000000000001,3.50000000000001,1.66666666666667,2}

{2.2,-1.4,-0.666666666666668,-1.1}

(4 rows)

        设A、B是两个矩阵，若AB=BA=E，则称B是A的逆矩阵，而A则被称为可逆矩阵。其中E是单位矩阵。

        一个实际和理论问题是矩阵是否像实数一样具有乘法逆元。首先，由于矩阵乘法的性质（即维必须匹配），如果矩阵具有逆矩阵（inverse matrix），它必须是方阵。这样，对于一个mXm的矩阵A，我们会问是否可以找到一个矩阵https://img-blog.csdn.net/20171226184131339使得https://img-blog.csdn.net/20171226184233248。答案是某些方阵有逆矩阵，而有些没有。

        一个mXm矩阵A有逆矩阵，当且仅当矩阵的秩R(A)=m，此时方阵A的行列式不为零，即|A|≠0，称A为非奇异矩阵或满秩矩阵，否则称A为奇异矩阵或降秩矩阵。满秩方阵的行、列向量组都是线性无关的。从概念上讲，一个mXm矩阵有逆矩阵，当且仅当它把每个非零m维行（列）向量都映射到一个唯一的非零m维行（列）向量。在求解各种矩阵方程时，逆矩阵的存在性是很重要的。

        下面看一个不可逆矩阵的例子。

create table t1 (a int, b int[]);

insert into t1 values

(1,'{1,2,3}'),(2,'{2,4,6}'),(3,'{3,6,9}');

select madlib.matrix\_rank('t1', 'row=a,val=b');

select madlib.matrix\_inverse('t1', 'row=a,val=b', 't2');

select \* from t2 order by a;

        3阶矩阵t1的秩为1，用matrix\_inverse求t1的逆矩阵，结果如下：

a | b

---+--------------------------

1 |{NaN,NaN,NaN}

2 |{-Infinity,Infinity,NaN}

3 |{Infinity,-Infinity,NaN}

(3 rows)

        如果求逆的矩阵不是方阵，则matrix\_inverse函数会报如下错误：

Matrix error: Inverse operation is onlydefined for square matrices

（19）求广义逆矩阵

        把逆矩阵推广到不可逆方阵（奇异矩阵）或长方矩阵上，这就是所谓的广义逆矩阵。广义逆矩阵具有逆矩阵的部分性质，并且在方阵可逆时，它通常与逆矩阵一致。

drop table if exists mat\_r;

select madlib.matrix\_pinv('mat\_a','row=row\_id, val=row\_vec', 'mat\_r');

select row\_vec from mat\_r order byrow\_id;

        结果：

row\_vec

---------------------------------------------------------------------------

{-1.20000000000001,0.900000000000004,0.333333333333335,0.600000000000003}

{3.20000000000002,-2.40000000000001,-1,-1.10000000000001}

{-5.00000000000003,3.50000000000002,1.66666666666667,2.00000000000001}

{2.20000000000001,-1.40000000000001,-0.66666666666667,-1.1}

(4rows)

        matrix\_pinv函数用于求矩阵的广义逆矩阵。还以上面的不可逆方阵为例，求它的广义逆矩阵。

drop table if exists t1,t2;

create table t1 (a int, b int[]);

insert into t1 values

(1,'{1,2,3}'),(2,'{2,4,6}'),(3,'{3,6,9}');

select madlib.matrix\_pinv('t1', 'row=a,val=b', 't2');

select \* from t2 order by a;

        结果：

a | b

---+-------------------------------------------------------------

1 |{0.00510204081632653,0.0102040816326531,0.0153061224489796}

2 |{0.0102040816326531,0.0204081632653061,0.0306122448979592}

3 |{0.0153061224489796,0.0306122448979592,0.0459183673469388}

(3 rows)

        再看一个长方矩阵的例子。

drop table if exists mat\_r;

select madlib.matrix\_ndims('mat\_b','row=row\_id, val=vector'),

madlib.matrix\_pinv('mat\_b', 'row=row\_id, val=vector', 'mat\_r');

select \* from mat\_r order by row\_id;

        mat\_b是一个5X4矩阵，它的广义逆矩阵如下：

row\_id| vector

--------+----------------------------------------------------------------------------------------------------

1|{0.169405974490348,-0.000368687326811998,0.153584606426279,-0.123375654346853,-0.0920196750284563}

2|{-0.0977762692158761,0.0690615737096675,-0.292887943436732,0.173906300749372,0.0622886509652684}

3|{-0.145985550968097,0.18130052991488,-0.238906461684316,0.0186947883412873,0.123325910818632}

4|{0.167425011631207,-0.222818819439771,0.534239640910248,-0.134386530622994,-0.0413193154120082}

(4rows)

        设A为m×n矩阵，如果存在n×m阶矩阵G，满足条件 ① AGA=A，② GAG=G, ③(AG)\*=AG, ④ (GA)\*=GA，式中\*表示共轭后再转置，则称G为A的广义矩阵。

（20）提取矩阵的特征值

drop table if exists mat\_r;

select madlib.matrix\_eigen('mat\_a','row=row\_id, val=row\_vec', 'mat\_r');

select \* from mat\_r order by row\_id;

        结果：

row\_id | eigen\_values

--------+------------------------

1 | (22.2561699851212,0)

2 | (-0.325748023524478,0)

3 | (2.91179834025418,0)

4 | (-2.8422203018509,0)

(4 rows)

        特征值和特征向量，连同相关的奇异值和奇异向量概念，捕获了矩阵的结构，使得我们可以分解矩阵，并用标准形式表示它们。因此，这些概念可以用于数学方程求解、维归约和降低噪声。

        n阶方阵A的特征值和特征向量分别是标量值λ和向量u，它们是如下方程的解：Au=λu

        换言之，特征向量（eigenvector）是被A乘时除量值外并不改变的向量。特征值（eigenvalue）是缩放因子。该方程也可以写成(A-λE)u = 0，其中E为单位矩阵。|A-λE|是一个n次多项式，它的全部根就是n阶方阵A的全部特征值。如果n阶矩阵A的全部特征值为https://img-blog.csdn.net/20171226185357291，则https://img-blog.csdn.net/20171226185540855。

（21）求矩阵范数

        matrix\_norm函数用于求矩阵范数，支持的类型值有‘fro’、‘one’、‘inf’、‘max’、‘spec’，分别代表frobenius范数、1范数、infinity范数、max范数和spectral范数。缺省为frobenius范数。

select madlib.matrix\_norm('mat\_b\_sparse','row=row\_id, col=col\_id, val=val','fro');

        结果：

matrix\_norm

---------------

23.4520787991

(1 row)

        F-范数的公式为：https://img-blog.csdn.net/20171226185728406。依据公式下面查询的结果与matrix\_norm函数的返回值相等。

select sqrt(sum(power(val,2))) frommat\_b\_sparse;

（22）求矩阵核范数

select madlib.matrix\_nuclear\_norm('mat\_a','row=row\_id, val=row\_vec');

        结果：

matrix\_nuclear\_norm

---------------------

34.322635238

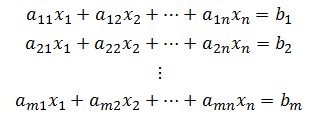
(1 row)

        矩阵的核范数是指矩阵奇异值的和，关于矩阵奇异值，在讨论MADlib的矩阵分解函数时再进行详细说明。

### 矩阵与数据分析

        我们可以把数据集表示成数据矩阵，其中每一行存放一个数据对象，而每一列是一个属性。（同样，我们也可以用行表示属性，列表示对象。）矩阵表示为我们的数据提供了紧凑、结构良好的表示，使得我们可以很容易地通过各种矩阵运算对数据对象或属性进行操作。

        线性方程组是使用数据的矩阵表示的很常见的例子。线性方程组可以写成一个矩阵方程Ax=b，并使用矩阵运算求解。



        特殊地，如果A有逆矩阵，则该方程组的解为https://img-blog.csdn.net/20171226190042818。如果A没有逆矩阵，则该方程组或者没有解，或者有无穷多个解。注意，在这种情况下，行（数据对象）是方程，列是变量（属性）。

        对于许多统计学和数据分析问题，我们希望解线性方程组，但是这些线性方程组不能使用刚才介绍的方法求解。例如，我们可能有一个数据矩阵，其中行代表病人，而列代表病人的特征（身高、体重和年龄）和他们对特定药物治疗的反应（如血压的变化）。我们想把血压（因变量）表示成其它（自）变量的线性函数，并且可以用上面的方法写一个矩阵方程。然而，如果我们的病人比变量多（通常如此），则矩阵的逆不存在。

        在这种情况下，我们仍然想找出该方程的最好解。这意味着我们想找出自变量的最好线性组合来预测因变量。使用线性代数的术语，我们想找尽可能接近向量b的向量Ax；换句话说，我们希望最小化向量b-Ax的长度‖b-Ax‖。这称作最小二乘（least square）问题。许多统计学技术（例如线性回归）都需要解最小二乘问题。可以证明，方程Ax=b的最小二乘解是https://img-blog.csdn.net/20171226190220381。

        在分析数据时，特别是对于维归约，奇异值和特征向量分解也非常有用。维归约还可以带来降低噪声的效果。

# 数据转换之邻近度

MADlib的线性代数模块（linalg module）包括基本线性代数操作的实用函数，其中包括多种范式、距离、相似度、向量均值、矩阵聚合等函数。本篇先从讨论相似性和相异性的基本概念，然后对照概念说明MADlib的线性代数函数，并用简单示例描述这些函数的用法。

## 邻近度的度量

        相似性要和相异性是重要的概念，因为它们被许多数据挖掘技术所使用，如聚类、最邻近分类和异常检测等。在许多情况下，一旦计算出相似性或相异性，就不再需要原始数据了。这种方法可以看作将数据变换到相似性（相异性）空间，然后进行分析。为方便起见，我们使用术语邻近度（proximity）表示相似性或相异性。

        两个对象之间的相似度（similarity）是指这两个对象相似程度的数值度量。两个对象越相似，它们的相似度就越高。通常，相似度是非负的，并常常在0（不相似）和1（完全相似）之间取值。两个对象之间的相异度（dissimilarity）是这两个对象差异程度的数值度量。对象越类似，它们的相异度就越低。术语距离（distance）经常用作相异度的同义词，用来表示特定类型的相异度。有时，相异度在区间[0,1]中取值，但相异度在0和∞之间取值也很常见。

        通常使用变换把相似度转换成相异度或相反，或者把邻近度变换到一个特定区间，如[0,1]。例如，我们可能有相似度，其值域从1到10，但是我们打算使用的算法或软件只能处理相异度，或只能处理[0,1]区间的相似度。这种变换相对独立于特定的邻近度度量方法。

        邻近度度量（特别是相似度）常被定义为或变换到区间[0,1]中的值。这样做的动机是使用一种适当的尺度，由邻近度的值表明两个对象之间的相似（或相异）程度。这种变换通常是比较直接的。例如，如果对象之间的相似度在1和10之间变化，则我们可以使用如下变换将它变换到[0,1]区间：s'=(s-1)/9，其中s和s'分别是相似度的原值和新值。一般来说，相似度到[0,1]区间的变换由如下表达式给出：s'=(s-min\_s)/(max\_s-min\_s)，其中max\_s和min\_s分别是相似度的最大值和最小值。类似地，有限值域的相异度也能用d'=(d-min\_d)/(max\_d-min\_d)映射到[0,1]区间。这种方法有时也称为Min-Max标准化法。当使用诸如神经网络、最近邻分类或聚类这种基于距离的挖掘算法进行建模或挖掘时，如果待分析的数据已经标准化，即按比例映射到一个较小的区间（如[0,1]），则这些方法将得到更好的结果。

        然而，将邻近度映射到[0,1]区间也可能非常复杂。例如，如果邻近度度量原来在区间[0,∞]上取值，则需要使用非线性变换，并且在新的尺度上，值之间不再具有与原来相同的联系。对于从0变化到∞的相异度度量，考虑变换d'=d/(1+d)，相异度0、0.5、2、10、100和1000分别被变换到0、0.33、0.67、0.90、0.99和0.999。在原来相异性尺度上较大的值被压缩到1附近，但是否希望如此取决于应用。另一个问题是邻近度度量的含义可能会被改变。例如，相关性是一种相似性度量，在区间[-1,1]上取值，通过取绝对值将这些值映射到[0,1]区间丢失了符号信息，而对于某些应用，符号信息可能是重要的。

        将相似度变换成相异度或相反也是比较直接的，尽管我们可能再次面临保持度量的含义问题和将线性尺度改变成非线性尺度的问题。如果相似度（相异度）落在[0,1]区间，则相异度（相似度）可以定义为d=1-s（或s=1-d）。另一种简单的方法是定义相似度为负的相异度（或相反）。例如，相异度0、1、10和100可以分别换成相似度0、-1、-10和-100。

        负变换产生的相似度不必局限于[0,1]区间，但如果希望的话，则可以使用变换s=1/(d+1)、https://img-blog.csdn.net/20171229173331985或https://img-blog.csdn.net/20171229173413891。

        一般来说，任何单调减函数都可以用来将相异度转换到相似度（或相反）。当然，在将相似度变换到相异度（或相反），或者在将邻近度的值变换到新的尺度时，也必须考虑一些其它因素，如前面提到过的涉及保持意义、扰乱标度和数据分析工具的需要等等问题。

## MADlib的邻近度相关函数

### 函数概览

        利用MADlib提供的邻近度相关函数，可以很方便地实现新算法。这些函数操作的对象是向量（1维FLOAT8数组）和矩阵（2维FLOAT8数组）。注意，这类函数只接受FLOAT8数组参数，因此在调用函数时，需要将其它类型的数组转换为FLOAT8[]。表1列出了相关函数的简要说明。

|  |  |  |  |
| --- | --- | --- | --- |
| **函数名称** | **描述** | **参数** | **返回值** |
| norm1() | 向量的1范数https://img-blog.csdn.net/20171229173500626 | 向量https://img-blog.csdn.net/20171229173537548 | https://img-blog.csdn.net/20171229173615053 |
| norm2() | 向量的2范数https://img-blog.csdn.net/20171229173824511 | 向量https://img-blog.csdn.net/20171229173850405 | https://img-blog.csdn.net/20171229173924804 |
| dist\_norm1() | 两个向量之差的1范数  https://img-blog.csdn.net/20171229173959807 | 向量https://img-blog.csdn.net/20171229174014269  向量https://img-blog.csdn.net/20171229174047196 | https://img-blog.csdn.net/20171229174115500 |
| dist\_norm2() | 两个向量之差的2范数  https://img-blog.csdn.net/20171229174229374 | 向量https://img-blog.csdn.net/20171229174241785  向量https://img-blog.csdn.net/20171229174313054 | https://img-blog.csdn.net/20171229174400833 |
| dist\_pnorm() | 两个向量之差的p范数  https://img-blog.csdn.net/20171229174508987 | 向量https://img-blog.csdn.net/20171229174421029  向量https://img-blog.csdn.net/20171229174431073  标量p，p>0 | https://img-blog.csdn.net/20171229174537026 |
| dist\_inf\_norm() | 两个向量之差的无穷范数https://img-blog.csdn.net/20171229174604308 | 向量https://img-blog.csdn.net/20171229174616415  向量https://img-blog.csdn.net/20171229174628195 | https://img-blog.csdn.net/20171229174701315 |
| squared\_dist\_norm2() | 两个向量之差的2范数平方https://img-blog.csdn.net/20171229174732410 | 向量https://img-blog.csdn.net/20171229174745693  向量https://img-blog.csdn.net/20171229174755258 | https://img-blog.csdn.net/20171229174831369 |
| cosine\_similarity() | 两个向量的余弦相似度  https://img-blog.csdn.net/20171229174905719 | 向量https://img-blog.csdn.net/20171229174916480  向量https://img-blog.csdn.net/20171229174927013 | https://img-blog.csdn.net/20171229174959644 |
| dist\_angle() | 欧氏空间中两个向量之间的角距离https://img-blog.csdn.net/20171229175038646 | 向量https://img-blog.csdn.net/20171229175047924  向量https://img-blog.csdn.net/20171229175056386 | https://img-blog.csdn.net/20171229175134560 |
| dist\_tanimoto() | 两个向量间的谷本距离 | 向量https://img-blog.csdn.net/20171229175203575  向量https://img-blog.csdn.net/20171229175214010 | https://img-blog.csdn.net/20171229175255725 |
| dist\_jaccard() | 两个字符向量集之间的杰卡德距离 | 向量https://img-blog.csdn.net/20171229175317587  向量https://img-blog.csdn.net/20171229175325710 | https://img-blog.csdn.net/20171229175353074 |
| get\_row() | 返回矩阵的行 | 二维数组行下标 | 二维数组的一行 |
| get\_col() | 返回矩阵的列 | 二维数组列下标 | 二维数组的一列 |
| avg() | 计算向量的平均值 | m个n维向量  https://img-blog.csdn.net/20171229175411394 | https://img-blog.csdn.net/20171229175509967 |
| normalized\_avg() | 计算向量的归一化平均值（欧氏空间中的单位向量） | m个n维向量  https://img-blog.csdn.net/20171229175524854 | https://img-blog.csdn.net/20171229175556003 |
| matrix\_agg() | 将向量合并进一个矩阵 | 向量https://img-blog.csdn.net/20171229175610443 | 包含https://img-blog.csdn.net/20171229175729554列的矩阵 |

表1 MADlib邻近度相关函数

### 函数示例

**（1）范数**

select madlib.norm1('{1,-2,3}'),madlib.norm2('{1,-2,3}');

        结果：

norm1 | norm2

-------+------------------

6 | 3.74165738677394

(1 row)

        1范数的定义为向量各元素绝对值的和，2范数的定义是向量各元素平方和的平方根。根据定义下面的查询与范数函数的结果相同。

select sum(norm1) norm1, sqrt(sum(norm2))norm2

from (select abs(unnest('{1,-2,3}'::float[])) norm1 ,

power(unnest('{1,-2,3}'::float[]),2) norm2) t;

**（2）欧几里得距离（L2范数）**

select madlib.dist\_norm2('{1,-2,3}','{4,-5,6}');

        结果：

dist\_norm2

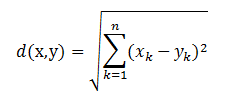
------------------

5.19615242270663

select madlib.dist\_pnorm('{1,-2,3}','{4,-5,6}', 5);

(1 row)

        2范数与欧几里得距离相对应。一维、二维、三维或高维空间中两个点x和y之间的欧几里得距离（Euclideandistance）d由如下公式定义：



其中，n是维数，而https://img-blog.csdn.net/20171229180242756和https://img-blog.csdn.net/20171229180401785分别是x和y的第k个属性值（分量）。

**（3）闵可夫斯基距离（Lp范数）**

select madlib.dist\_pnorm('{1,-2,3}','{4,-5,6}', 5);

        结果：

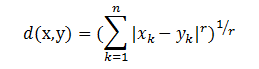
dist\_pnorm

------------------

3.73719281884655

(1 row)

        欧几里得距离可以用闵可夫斯基距离（Minkowski distance）来推广：



其中r是标量参数。注意不要将参数r与维数（属性数）n混淆。欧几里得距离、曼哈顿距离和上确界距离是对n的所有值（1,2,3…）定义的，并且指定了将每个维（属性）上的差的组合成总距离的不同方法。

**（4）曼哈顿距离（L1范数）**

select madlib.dist\_norm1('{1,-2,3}', '{4,-5,6}');

        结果：

dist\_norm1

------------

9

(1 row)

        当闵可夫斯基距离的r = 1时称为曼哈顿距离。r = 2，就是欧几里得距离。

**（5）上确界距离（Lmax或L∞范数）。**

select madlib.dist\_inf\_norm('{1,-2,3}','{4,-5,6}');

        结果：

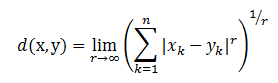
dist\_inf\_norm

---------------

3

(1 row)

        当闵可夫斯基距离的r = ∞，称为上确界距离。这是对象属性之间的最大距离。更正式地，https://img-blog.csdn.net/20171229181018036距离由下面的公式定义：



**（6）平方欧几里得距离**

select madlib.squared\_dist\_norm2('{1,-2,3}', '{4,-5,6}');

        结果：

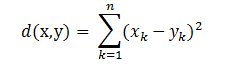
squared\_dist\_norm2

--------------------

27

(1 row)

        平方欧几里得距离即：



        距具有一些基本性质。如果d(x,y)是两个点x和y之间的距离，则如下性质成立：

* 非负性。（a）对于所有x和y，d(x,y)≥0，（b）仅当x=y时d(x,y)=0。
* 对称性。对于所有x和y，d(x,y)=d(y,x)。
* 三角不等式。对于所有x、y和z，d(x,z) ≤ d(x,y) + d(y,z)。

         对于相似度，三角不等式（或类似的性质）通常不成立，但是对称性和非负性通常成立。更明确地说，如果s(x,y)是数据点x和y之间的相似度，则相似度具有如下典型性质。

* 仅当x=y时s(x,y)=1。（0≤s≤1）
* 对于所有x和y，s(x,y)=s(y,x)。（对称性）

     对于相似度，没有与三角不等式对应的一般性质。然而，有时可以将相似度简单地变换成一种度量距离。

**（7）Jaccard距离**

select madlib.dist\_jaccard('{1,-2,3}','{4,-5,6}');

        结果：

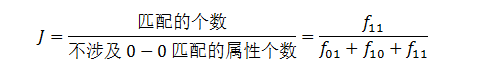
dist\_jaccard

--------------

1

(1 row)

        Jaccard距离的定义是1- Jaccard系数（Jaccard Coefficient）。Jaccard 系数定义为A与B交集的大小与A与B并集的大小的比值。假定x和y是两个数据对象，代表两个事务。如果每个二元属性对应于商店的一种商品，1表示该商品被购买，而0表示该商品未被购买。由于未被顾客购买的商品数远远大于被其购买的商品数，常常使用Jaccard系数来处理这种仅包含非对称二元属性的对象。Jaccard系数通常用符号J表示，由如下等式定义：



其中：https://img-blog.csdn.net/20171229181642010=x取0并且y取0的属性个数

https://img-blog.csdn.net/20171229181713146=x取0并且y取1的属性个数

https://img-blog.csdn.net/20171229181747339=x取1并且y取0的属性个数

https://img-blog.csdn.net/20171229181822150=x取1并且y取1的属性个数

**（8）Tanimoto距离**

select madlib.dist\_tanimoto('{1,-2,3}','{4,-5,6}');

        结果：

dist\_tanimoto

-------------------

0.457627118644068

(1 row)

        Tanimoto距离（谷本距离）定义为1-Tanimoto系数。Tanimoto系数又称广义Jaccard系数，可以用于文档数据，并在二元属性情况下归约为Jaccard系数。该系数用EJ表示，由下式定义：

https://img-blog.csdn.net/20171229181951329

**（9）余弦相似度**

select madlib.cosine\_similarity('{1,-2,3}','{4,-5,6}');

        结果：

cosine\_similarity

-------------------

0.974631846197076

(1 row)

        在介绍MADlib的稀疏向量时，曾经举了一个tf/idf算法的例子。当时我们使用了反余弦函数计算文档的角距离，从而以此判断文档的相似度（参见<http://blog.csdn.net/wzy0623/article/details/78874176>）。文档的相似性度量不仅应当像Jaccard度量一样需要忽略0-0匹配，而且还必须能够处理非二元向量。文档相似性最常用的度量之一就是余弦相似度，其定义如下。如果x和y是两个文档向量，则

https://img-blog.csdn.net/20171229182233918

其中，“.”表示向量点积，https://img-blog.csdn.net/20171229182407403，https://img-blog.csdn.net/20171229182516724是向量x的长度，https://img-blog.csdn.net/20171229182642027。

        余弦相似度实际上是x和y之间夹角（余弦）的度量。这样，如果余弦相似度为1，则x个y之间的夹角为0度，并且除大小（长度）之外，x和y是相同的；如果余弦相似度为0，则x和y之间的夹角为90度，以文档为例，说明它们不包含任何相同的词（术语）。

        余弦相似度公式可以写成下面的形式：

https://img-blog.csdn.net/20171229182734396

其中，https://img-blog.csdn.net/20171229182829686，而https://img-blog.csdn.net/20171229182909972。x和y被它们的长度除，将它们规范化成具有长度1。这意味着在计算相似度时，余弦相似度不考虑两个数据对象的量值。（当量值是重要的时，欧几里得距离可能是一种更好的选择。）对于长度为1的向量，余弦度量可以通过简单地取点积计算。从而，在需要大量对象之间的余弦相似度时，将对象规范化，使之具有单位长度可以减少计算时间。

**（10）角距离**

select madlib.dist\_angle('{1,-2,3}','{4,-5,6}');

        结果：

dist\_angle

-------------------

0.225726128552734

(1 row)

        角距离定义为余弦相似度上的反余弦函数：

dm=# selectacos(madlib.cosine\_similarity('{1,-2,3}', '{4,-5,6}'));

acos

-------------------

0.225726128552734

(1 row)

**（11）取矩阵的行列**

drop table if exists matrix;

create table matrix(id integer, mfloat8[]);

insert into matrix values (1,'{{4,5},{3,5},{9,0}}');

select madlib.get\_row(m, 1) as row\_1,

madlib.get\_row(m, 2) as row\_2,

madlib.get\_row(m, 3) as row\_3,

madlib.get\_col(m, 1) as col\_1,

madlib.get\_col(m, 2) as col\_2

from matrix;

        结果：

row\_1 | row\_2 | row\_3 | col\_1 | col\_2

-------+-------+-------+---------+---------

{4,5} | {3,5} | {9,0} | {4,3,9} | {5,5,0}

(1 row)

**（12）求向量平均值**

drop table if exists vector;

create table vector(id integer, vfloat8[]);

insert into vector values (1, '{4,1}'), (2,'{8,-6}'), (3, '{5,9}');

select madlib.avg(v) from vector;

        结果：

avg

-------------------------------------

{5.66666666666667,1.33333333333333}

(1 row)

        平均值函数madlib.avg对一组向量个分量求平均值，返回由分量平均值构成的结果向量。

dm=# select avg(v[1]),avg(v[2]) fromvector;

avg | avg

------------------+------------------

5.66666666666667 | 1.33333333333333

(1 row)

**（13）求向量的归一化平均值**

select madlib.normalized\_avg(v) fromvector;

        结果：

normalized\_avg

---------------------------------------

{0.974756579834322,0.223270262394468}

(1 row)

        madlib.normalized\_avg的计算过程为：

1. 将原数据中的向量做标准差归一化。
2. 对归一化后的数据求向量平均值。
3. 对结果向量再做一次标准差归一化，返回结果向量。

dm=# select c1/sqrt(power(c1,2) +power(c2,2)) c1,

dm-# c2/sqrt(power(c1,2) + power(c2,2)) c2

dm-# from (select avg(v[1]/sqrt(power(v[1],2) + power(v[2],2))) c1,

dm(# avg(v[2]/sqrt(power(v[1],2) +power(v[2],2))) c2

dm(# from vector) t;

c1 | c2

-------------------+-------------------

0.974756579834322 | 0.223270262394468

(1 row)

        注意madlib.normalized\_avg函数只做归一化，不做中心化（均值不为0）。

**（14）合并向量**

select madlib.matrix\_agg(v) from vector;

        结果：

matrix\_agg

----------------------

{{4,1},{8,-6},{5,9}}

(1 row)

        madlib.matrix\_agg函数将参数中的一组向量合并为一个矩阵（2维数组）。

## 距离度量的中心化和标准化

        距离度量的一个重要问题是当属性具有不同的值域时如何处理。（这种情况通常称作“变量具有不同的尺度。”）例如，基于年龄和收入两个属性来度量人之间的欧几里得距离，除非这两个属性是标准化的，否则两个人之间的距离将被收入所左右。

        当属性具有不同值域（不同的方差）、并且数据分布近似于正太分布时，需要标准化步骤对数据进行预处理。通过中心化和标准化处理，可以得到均值为0，标准差为1的服从正太分布的数据。

        假设样本集X的均值（mean）为m，标准差（Standard Deviation）为s，那么X的标准化变量表示为：

https://img-blog.csdn.net/20171229183759854

假设有一组数值https://img-blog.csdn.net/20171229183857062，其算数平均值为m，则标准差的计算公式为：

https://img-blog.csdn.net/20171229183939027

        标准差是一组数据平均值分散程度的一种度量。较大的标准差表示大部分数值和其平均值之间差异较大，标准差较小，代表这些数值比较接近平均值。

        通过简单的推导可得，两个向量x和y的标准化欧几里得距离的计算公式为：

https://img-blog.csdn.net/20171229184043454

其中，https://img-blog.csdn.net/20171229184105657是向量x的第k个分量，https://img-blog.csdn.net/20171229184120833向量y的第k个分量，https://img-blog.csdn.net/20171229184213477是第k个分量上的标准差。这样，在计算距离时，不同特征的影响程度就一样了。如果将方差的倒数看成是权重，标准化欧氏距离公式也可以看作是一种加权欧氏距离。

        标准化欧几里得距离解决了不同属性的尺度（值域）不一致的问题，但当某些属性之间相关时，可能需要使用马氏距离。

## 选取正确的邻近度度量

        首先，邻近度度量的类型应该与数据类型相适应。对于稠密的、连续的数据，通常使用距离度量，如欧几里得距离。数据挖掘中，取实数值的数据是连续的数据，而具有有限个值或无限但可数个值的数据称为离散数据。连续属性之间的邻近度通常用属性值的差来表示，并且距离度量提供了一种将这些差组合到总邻近性度量的良好的方法。尽管属性可能有不同的取值范围和不同的重要性，但这些问题通常可以使用标准化或加权的方法处理。

        对于稀疏数据，常常包含非对称的属性，通常使用忽略0-0匹配的相似性度量。从概念上讲，这反映了如下事实：对于一对复杂对象，相似度依赖于它们共同具有的性质数目，而不是依赖于它们都缺失的性质数目。在特殊情况下，对于稀疏的、非对称的数据，大部分对象都只具有少量被属性描述的性质，因此如果考虑它们都不具有的性质的话，它们都高度相似。余弦、Jaccard和广义Jaccard度量对于这类数据是合适的。

        在某些情况下，为了得到合适的相似性度量，数据的变换或规范化是重要的，因为这种变换并非总能在邻近性度量中提供，例如，时间序列数据可能具有显著影响相似性的趋势或周期模式。此外，正确地计算相似度还需要考虑时间延迟。最后，两个时间序列可能只在特定的时间周期上相似，例如，气温与天然气的用量之间存在很强的关联，但是这种联系仅出现在取暖季节。

        实践考虑也是重要的。有时，一种或多种邻近度度量已经在某个特定领域使用，因此，其他人已经回答了应当使用何种邻近度度量的问题；另外，所使用的软件包或聚类算法可能完全限制了选择；如果关心效率，则我们可能希望选择具有某些性质的邻近性度量，这些性质（如三角不等式）可以用来降低邻近度计算量。然而，如果通常的实践或实践限制并未规定某种选择，则正确地选择邻近性度量可能是一项耗时的任务，需要仔细地考虑领域知识和度量使用的目的。可能需要评估许多不同的相似性度量，以确定哪些结果最有意义。

# [数据转换之矩阵分解](https://blog.csdn.net/wzy0623/article/details/78971328)

        矩阵分解（Matrix Factorization）简单说就是将原始矩阵拆解为数个矩阵的乘积。在一些大型矩阵计算中，其计算量大，化简繁杂，使得计算非常复杂。如果运用矩阵分解，将大型矩阵分解成简单矩阵的乘积形式，则可大大降低计算的难度以及计算量。这就是矩阵分解的主要目的。而且，对于矩阵的秩的问题，奇异性问题，特征值问题，行列式问题等等，通过矩阵分解后都可以清晰地反映出来。另一方面，对于那些大型的数值计算问题，矩阵的分解方式以及分解过程也可以作为计算的理论依据。MADlib提供了低秩矩阵分解和奇异值分解两种矩阵分解方法。

## 低秩矩阵分解

        矩阵中的最大不相关向量的个数，叫做矩阵的秩，可通俗理解为数据有秩序的程度。秩可以度量相关性，而向量的相关性实际上又带有了矩阵的结构信息。如果矩阵之间各行的相关性很强，那么就表示这个矩阵实际可以投影到更低维度的线性子空间，也就是用几个特征就可以完全表达了，它就是低秩的。所以我们可以总结的一点是：如果矩阵表达的是结构性信息，例如图像、用户推荐表等，那么这个矩阵各行之间存在一定的相关性，这个矩阵一般就是低秩的。

        如果A是一个m行n列的数值矩阵，rank(A)是A的秩，假如rank(A)远小于m和n，则我们称A是低秩矩阵。低秩矩阵每行或每[列都](https://www.baidu.com/s?wd=%E5%88%97%E9%83%BD&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)可以用其它的行或列线性表示，可见它包含大量的冗余信息。利用这种冗余信息，可以对缺失数据进行恢复，也可以对数据进行特征提取。

        MADlib的lmf模块可用两个低维度矩阵的乘积逼近一个稀疏矩阵，逼近的目标就是让预测矩阵和原来矩阵之间的误差平方（RMSE）最小，从而实现所谓“潜在因子模型”。lmf模块提供的低秩矩阵分解函数，就是为任意稀疏矩阵A，找到两个矩阵U和V，使得https://img-blog.csdn.net/20180104152925497?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast的值最小化，其中https://img-blog.csdn.net/20180104153043863?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast代表Frobenius范数。换句话说，只要求得U和V，就可以用它们的乘积来近似[模拟](https://www.baidu.com/s?wd=%E6%A8%A1%E6%8B%9F&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)A。因此低秩矩阵分解有时也叫UV分解。假设A是一个m x n的矩阵，则U和V分别是m x r和n x r的矩阵，并且1<=r<=min(m,n)。

### MADlib低秩矩阵分解函数

        MADlib的lmf\_igd\_run函数能够实现低秩矩阵分解功能。本节介绍MADlib低秩矩阵分解函数的语法和参数含义，下一节用一个实例说明该函数的具体用法。

**（1）  lmf\_igd\_run函数语法**

lmf\_igd\_run( rel\_output,

rel\_source,

col\_row,

col\_column,

col\_value,

row\_dim,

column\_dim,

max\_rank,

stepsize,

scale\_factor,

num\_iterations,

tolerance )

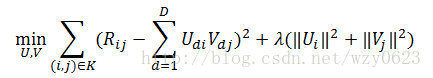
**（2）  参数说明**

|  |  |  |
| --- | --- | --- |
| **参数名称** | **数据类型** | **描述** |
| rel\_output | TEXT | 输出表名。输出的矩阵U和V是以二维数组类型存储。RESULT AS (   matrix\_u    DOUBLE PRECISION[],   matrix\_v    DOUBLE PRECISION[],   rmse        DOUBLE PRECISION)  行i对应的向量是matrix\_u[i:i][1:r]，列j对应的向量是matrix\_v[j:j][1:r]。 |
| rel\_source | TEXT | 输入表名。输入矩阵的格式如下：  {TABLE|VIEW} input\_table (   row   INTEGER,   col   INTEGER,   value DOUBLE PRECISION)  输入包含一个描述矩阵的表，数据被指定为（row、column、value）。输入矩阵的行列值大于等于1，并且不能有NULL值。 |
| col\_row | TEXT | 包含行号的列名。 |
| col\_column | TEXT | 包含列号的列名。 |
| col\_value | FLOAT8 | （row, col）位置对应的值。 |
| row\_dim（可选） | INTEGER | 指示矩阵中的行数，缺省为：“SELECT max(col\_row) FROM rel\_source”。 |
| column\_dim（可选） | INTEGER | 指示矩阵中的列数，缺省为：“SELECT max(col\_col) FROM rel\_source”。 |
| max\_rank | INTEGER | 期望逼近的秩数。 |
| stepsize（可选） | FLOAT8 | 缺省值为0.01。超参数，决定梯度下降法的步长。 |
| scale\_factor（可选） | FLOAT8 | 缺省值为0.1。超参数，决定初始缩放因子。 |
| num\_iterations（可选） | INTEGER | 缺省值为10。不考虑收敛情况下的最大迭代次数。 |
| tolerance（可选） | FLOAT8 | 缺省值为0.0001，收敛误差，小于该误差时停止迭代。 |

表1 lmf\_igd\_run函数参数说明

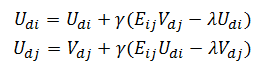
        矩阵分解一般不用数学上直接分解的办法，尽管直接分解出来的精度很高，但是效率实在太低！矩阵分解往往会转化为一个优化问题，通过迭代求局部最优解。但是有个问题是，通常原矩阵的稀疏度很大，分解很容易产生过拟合（overfitting），简单说就是为了迁就一些错误的偏僻的值导致整个模型错误的问题。所以现在的方法是在目标函数中增加一项正则化（regularization）参数，来避免过拟合问题。

        因此，一般的矩阵分解的目标函数（或者称为损失函数，loss function）为：



        前一项是预测后矩阵和原矩阵的误差，这里计算只针对原矩阵中的非空项。后一项就是正则化因子，用来解决过度拟合的问题。这个优化问题求解的就是分解之后的U、V矩阵的潜在因子向量。

        madlib.lmf\_igd\_run函数使用随机梯度下降法（stochastic gradient descent）求解这个优化问题，迭代公式为：



其中，https://img-blog.csdn.net/20180104153838705?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast。

        γ是学习速率，对应stepsize参数；λ是正则化系数，对应scale\_factor参数。这是两个超参数，对于最终结果影响极大。在机器学习的上下文中，超参数是在开始学习过程之前设置值的参数，而不是通过训练得到的参数数据。通常情况下，需要对超参数进行优化，以提高学习的性能和效果。γ的大小不仅会影响到执行时间，还会影响到结果的收敛性。γ太大的话会导致结果发散，一般都会把γ取得很小，不同的数据集取得值不同，但大概是0.001这个量级。这样的话训练时间会长一些，但结果会比较好。λ的值一般也比较小，大概取0.01这个量级。

        迭代开始前，需要对U、V的特征向量赋初值，这个初值很重要，会严重地影响到计算速度。一般的做法是在均值附近产生随机数作为初值。也正是由于这个原因，从数据库层面看，madlib.lmf\_igd\_run函数是一个非确定函数，也就是说，同样一组输入数据，多次执行函数生成的结果数据是不同的。迭代结束的条件一般是损失函数的值小于了某一个阈值（由tolerance参数指定），或者到达指定的最大迭代次数（由num\_iterations参数指定）。

### 低秩矩阵分解函数示例

        我们将通过一个简单示例，说明如何利用madlib.lmf\_igd\_run函数实现潜在因子（Latent Factor）推荐算法。该算法的主要思想是：每个用户（user）都有自己的偏好，比如一个歌曲推荐应用中，用户A喜欢带有小清新的、吉他伴奏的、王菲等元素，如果一首歌（item）带有这些元素，那么就将这首歌推荐给该用户，也就是用元素去连接用户和歌曲。每个人对不同的元素偏好不同，而每首歌包含的元素也不一样。

**（1）  潜在因子矩阵**

        我们希望能找到这样两个矩阵：

* 潜在因子-用户矩阵Q，表示不同的用户对于不同元素的偏好程度，1代表很喜欢，0代表不喜欢。比如图1所示：



图1 潜在因子-用户矩阵

* 潜在因子-歌曲矩阵P，表示每首歌曲含有各种元素的成分，比如图2中，音乐A是一个偏小清新的音乐，含有小清新这个潜在因子的成分是0.9，重口味的成分是0.1，优雅的成分是0.2等等。

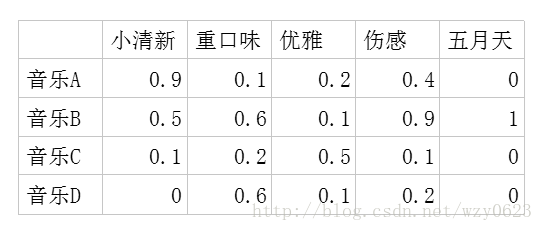


图2 潜在因子-音乐矩阵

        利用这两个矩阵，我们能得出张三对音乐A的喜欢程度是：张三对小清新的偏好\*音乐A含有小清新的成分+对重口味的偏好\*音乐A含有重口味的成分+对优雅的偏好\*音乐A含有优雅的成分+……即：0.6\*0.9+0.8\*0.1+0.1\*0.2+0.1\*0.4+0.7\*0=0.68。

        每个用户对每首歌都这样计算可以得到不同用户对不同歌曲的评分矩阵https://img-blog.csdn.net/20180104154608665?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，如图3所示。注意，这里的波浪线表示的是估计的评分，接下来我们还会用到不带波浪线的R表示实际的评分矩阵。

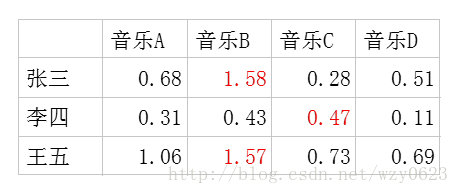


图3 估计的评分矩阵

        因此我们对张三推荐四首歌中得分最高的B，对李四推荐得分最高的C，王五推荐B。如果用矩阵表示即为：

https://img-blog.csdn.net/20180105092833150?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

**（2）  如何得到潜在因子**

        潜在因子是怎么得到的呢？面对大量用户和歌曲，让用户自己给歌曲分类并告诉我们其偏好系数显然是不现实的，事实上我们能获得的只有用户行为数据。假定使用以下量化标准：单曲循环=5, 分享=4, 收藏=3, 主动播放=2 , 听完=1, 跳过=-2 , 拉黑=-5，则在分析时能获得的实际评分矩阵R，也就是输入矩阵如图4所示：



图4 实际评分矩阵

        推荐系统的目标就是预测出空白对应位置的分值。推荐系统基于这样一个假设：用户对项目的打分越高，表明用户越喜欢。因此，预测出用户对未评分项目的评分后，根据分值大小排序，把分值高的项目推荐给用户。这是个非常稀疏的矩阵，因为大部分用户只听过全部歌曲中很少一部分。如何利用这个矩阵去找潜在因子呢？这里主要应用到的就是矩阵的UV分解，如图5所示。

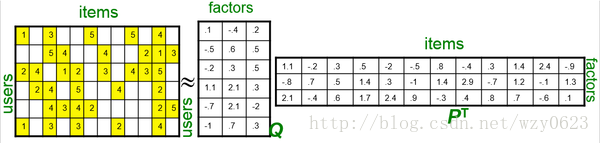
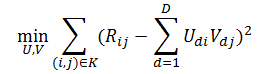


图5 矩阵的UV分解

        矩阵分解的想法来自于矩阵补全，即依据一个矩阵给定的部分数据，把缺失的值补全。一般假设原始矩阵是低秩的，我们可以从给定的值来还原这个矩阵。由于直接求解低秩矩阵从算法以及参数的复杂度来说效率很低，因此常用的方法是直接把原始矩阵分解成两个子矩阵相乘。例如将图5所示的评分矩阵分解为两个低维度的矩阵，用Q和P两个矩阵的乘积去估计实际的评分矩阵，而且我们希望估计的评分矩阵和实际的评分矩阵不要相差太多，也就是求解矩阵分解的目标函数：



        如前所述，实际应用中，往往还要加上2范数的罚项，然后利用梯度下降法就可以求得这P,Q两个矩阵的估计值。例如我们上面给出的那个例子可以分解成为这样两个矩阵（图6）：

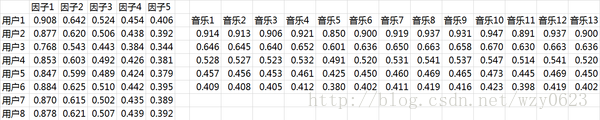


图6 分解后得到的UV矩阵

        这两个矩阵相乘就可以得到估计的得分矩阵（图7）：



图7 预测矩阵

        将用户已经听过的音乐剔除后，选择分数最高音乐的推荐给用户即可（红体字）。在这个例子里面用户7和用户8有强的相似性（图8）：

https://img-blog.csdn.net/20180104155822641?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

图8 评分相似的用户

        从推荐的结果来看，正好推荐的是对方评分较高的音乐（图9）：

https://img-blog.csdn.net/20180104155909884?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

图9对相似用户的推荐

        该算法假定我们要恢复的矩阵是低秩的，实际上这种假设是十分合理的，比如一个用户对某歌曲的评分是其他用户对这首歌曲评分的线性组合。所以，通过低秩重构就可以预测用户对其未评价过的音乐的喜好程度。从而对矩阵进行填充。

**（3）  利用madlib.lmf\_igd\_run函数实现**

1. **建立输入表并生成输入数据**

-- 创建用户索引表

drop table if exists tbl\_idx\_user;

create table tbl\_idx\_user (user\_idx bigserial, userid varchar(10));

-- 创建音乐索引表

drop table if exists tbl\_idx\_music;

create table tbl\_idx\_music (music\_idx bigserial, musicid varchar(10));

-- 创建用户行为表

drop table if exists lmf\_data;

create table lmf\_data (

row int,

col int,

val float8

);

-- 生成输入表数据

-- 用户表

insert into tbl\_idx\_user (userid)

values ('u1'),('u2'),('u3'),('u4'),('u5'),('u6'),('u7'),('u8'),('u9'),('u10');

-- 音乐表

insert into tbl\_idx\_music (musicid)

values ('m1'),('m2'),('m3'),('m4'),('m5'),('m6'),('m7'),('m8'),('m9'),('m10'),('m11'),('m12'),('m13'),('m14'),('m15');

-- 用户行为表

insert into lmf\_data values (1, 1, 5), (1, 6, -5), (1, 9, 5), (1, 11, 3), (1, 12, 1), (1, 13, 5);

insert into lmf\_data values (2, 4, 3), (2, 9, 3), (2, 13, 4);

insert into lmf\_data values (3, 3, 1), (3, 5, 2), (3, 6, -5), (3, 7, 4), (3, 11, -2), (3, 12, -2), (3, 13, -2);

insert into lmf\_data values (4, 2, 4), (4, 3, 4), (4, 4, 3), (4, 7, -2), (4, 9, -5), (4, 12, 3);

insert into lmf\_data values (6, 2, 5), (6, 3, -5), (6, 5, -5), (6, 7, 4), (6, 8, 3), (6, 11, 4);

insert into lmf\_data values (7, 3, 4), (7, 6, 3), (7, 9, 4);

insert into lmf\_data values (8, 2, -2), (8, 6, 5), (8, 11, 4), (8, 12, 4), (8, 13, -2);

insert into lmf\_data values (9, 2, -2), (9, 6, 5), (9, 8, 5), (9, 11, 4), (9, 13, -2);

        说明：

* 从前面的解释可以看到，推荐矩阵的行列下标分别表示用户和歌曲。然而在业务系统中，userid和musicid很可能不是按从1到N的规则顺序生成的，因此通常需要建立矩阵下标值与业务表ID之间的映射关系，这里使用HAWQ的BIGSERIAL自增数据类型对应推荐矩阵的索引下标。
* 在生成原始数据时对图4的例子做了适当的修改。用户表中u5和u10用户没有给任何歌曲打分，而音乐表中的m10、m14、m15无评分。我们希望看到的结果是，除了与打分行为相关的用户和歌曲以外，也能为u5、u10推荐歌曲，并可能将m10、m14、m15推荐给用户。

1. **调用lmf\_igd\_run函数分解矩阵**

-- 执行低秩矩阵分解

drop table if exists lmf\_model;

select madlib.lmf\_igd\_run( 'lmf\_model',

'lmf\_data',

'row',

'col',

'val',

11,

16,

7,

0.1,

1,

10,

1e-9

);

说明：

最大行列数可以大于实际行列数，如这里传入的参数是11和16，而实际的用户数与歌曲数是10和15。

max\_rank参数为最大秩数，要小于min(row\_dim, column\_dim)，否则函数会报错：

NOTICE: Matrix lmf\_data to be factorized: 11 x 16

ERROR: plpy.SPIError: Function "madlib.lmf\_igd\_transition(double precision[],integer,integer,double precision,double precision[],integer,integer,integer,double precision,double precision)": Invalid parameter: max\_rank >= row\_dim || max\_rank >= column\_dim (plpython.c:4663)

最大秩数实际可以理解为最大的潜在因子数，也就是例子中的最大量化指标个数。本例中共有7个指标，因此max\_rank参数传7。

stepsize和scale\_factor参数对于结果的影响巨大，而且不同的学习数据，参数值也不同。也就是说超参数的值是与输入数据相关的。在本例中，使用缺省值时RMSE很大。经过反复测试，对于测试矩阵，stepsize和scale\_factor分别为0.1和1时误差相对较小。

        函数执行结果的控制台输出如下：

NOTICE: Matrix lmf\_data to be factorized: 11 x 16

NOTICE: CREATE TABLE will create implicit sequence "lmf\_model\_id\_seq" for serial column "lmf\_model.id"

CONTEXT: SQL statement "

CREATE TABLE lmf\_model (

id SERIAL,

matrix\_u DOUBLE PRECISION[],

matrix\_v DOUBLE PRECISION[],

rmse DOUBLE PRECISION)"

PL/pgSQL function "lmf\_igd\_run" line 49 during exception cleanup

NOTICE:

Finished low-rank matrix factorization using incremental gradient

DETAIL:

\* table : lmf\_data (row, col, val)

Results:

\* RMSE = 0.0032755443518

Output:

\* view : SELECT \* FROM lmf\_model WHERE id = 1

lmf\_igd\_run

-------------

1

(1 row)

Time: 2477.339 ms

        可以看到，误差值为0.0033，分解用时2秒多。

1. **检查结果**

        从上一步的输出看到，lmf\_igd\_run()函数返回的模型ID是1，需要用它作为查询条件。

select array\_dims(matrix\_u) as u\_dims, array\_dims(matrix\_v) as v\_dims

from lmf\_model

where id = 1;

        结果：

u\_dims | v\_dims

-------------+-------------

[1:11][1:7] | [1:16][1:7]

(1 row)

Time: 158.163 ms

结果表中包含分解成的两个矩阵，U（用户潜在因子）矩阵11行7列，V（歌曲潜在因子）矩阵16行7列。

1. **查询结果值**

select matrix\_u, matrix\_v from lmf\_model where id = 1;

1. **矩阵相乘生成推荐矩阵**

        MADlib的矩阵相乘函数是matrix\_mult，支持稠密和稀疏两种矩阵表示。稠密矩阵需要指定矩阵对应的表名、row和val列，稀疏矩阵需要指定矩阵对应的表名、row、col和val列。现在要将lmf\_igd\_run函数输出的矩阵装载到表中再执行矩阵乘法。这里使用稀疏形式，只要将二维矩阵的行、列、值插入表中即可。

-- 建立用户稀疏矩阵表

drop table if exists mat\_a\_sparse;

create table mat\_a\_sparse as

select d1,d2,matrix\_u[d1][d2] val from

(select matrix\_u,

generate\_series(1,array\_upper(matrix\_u,1)) d1,

generate\_series(1,array\_upper(matrix\_u,2)) d2

from lmf\_model) t;

-- 建立音乐稀疏矩阵表

drop table if exists mat\_b\_sparse;

create table mat\_b\_sparse as

select d1,d2,matrix\_v[d1][d2] val from

(select matrix\_v,

generate\_series(1,array\_upper(matrix\_v,1)) d1,

generate\_series(1,array\_upper(matrix\_v,2)) d2

from lmf\_model) t;

-- 执行矩阵相乘

drop table if exists matrix\_r;

select madlib.matrix\_mult('mat\_a\_sparse', 'row=d1, col=d2, val=val',

'mat\_b\_sparse', 'row=d1, col=d2, val=val, trans=true',

'matrix\_r');

        结果：

matrix\_mult

-------------

(matrix\_r)

(1 row)

Time: 5785.798 ms

        这两个矩阵（11 x 3与16 x 3）相乘用时将近6秒。生成的结果表是稠密形式的11 x 16矩阵，这就是我们需要的推荐矩阵。为了方便与原始的索引表关联，将结果表转为稀疏表示。

drop table if exists matrix\_r\_sparse;

select madlib.matrix\_sparsify('matrix\_r', 'row=d1, val=val',

'matrix\_r\_sparse', 'col=d2, val=val');

        结果：

matrix\_sparsify

-------------------

(matrix\_r\_sparse)

(1 row)

Time: 2252.413 ms

        最后与原始的索引表关联，过滤掉用户已经已经听过的歌曲，选择分数最高的歌曲推荐。

select t2.userid,t3.musicid,t1.val

from (select d1,d2,val,row\_number() over (partition by d1 order by val desc) rn

from matrix\_r\_sparse t1

where not exists (select 1 from lmf\_data t2 where t1.d1 = t2.row and t1.d2 = t2.col)) t1,

tbl\_idx\_user t2, tbl\_idx\_music t3

where t1.rn = 1 and t2.user\_idx= t1.d1 and t3.music\_idx = t1.d2

order by t2.user\_idx;

        结果：

userid | musicid | val

--------+---------+------------------

u1 | m7 | 4.46508576767804

u2 | m1 | 3.45259331270197

u3 | m14 | 1.21085044974838

u4 | m15 | 2.87854763370041

u5 | m12 | 3.07991997971589

u6 | m9 | 2.8073075471921

u7 | m8 | 5.70400322091688

u8 | m8 | 4.229015803831

u9 | m12 | 4.14559001844873

u10 | m8 | 2.16407771238165

(10 rows)

Time: 743.415 ms

        这就是为每个用户推荐的歌曲。可以看到，用户u5、u10分别推荐了m12和m8，m14和m15也推荐给了用户。

        MADlib的低秩矩阵分解函数可以作为推荐类应用的算法实现。从函数调用角度看，madlib.lmf\_igd\_run函数是一个非确定函数，也就是说，同样一组输入数据，函数生成的结果数据却不同，结果就是对于同样的输入数据，每次的推荐可能不一样。在海量数据的应用中，推荐可能需要计算的是一个几亿 x 几亿的大型矩阵，如何保证推荐系统的性能将成为巨大挑战。

## 奇异值分解

### 奇异值分解简介

        低秩矩阵分解是用两个矩阵的乘积近似还原一个低秩矩阵。MADlib还提供了另一种矩阵分解方法，即奇异值分解。奇异值分解简称SVD（Singular Value Decomposition），可以理解为将一个比较复杂的矩阵用更小更简单的三个子矩阵的相乘来表示，这三个小矩阵描述了原矩阵重要的特性。SVD的用处有很多，比如推荐系统、数据降维等。

        要理解奇异值分解，先要知道什么是特征值和特征向量。m×n矩阵M的特征值和特征向量分别是标量值*λ*和向量u，它们是如下方程的解：

Mu=*λ*u

        换言之，特征向量（eigenvector）是被M乘时除量值外并不改变的向量。特征值（eigenvalue）是缩放因子。该方程也可以写成(M-*λ*E)u=0，其中E是单位矩阵。

        对于方阵，可以使用特征值和特征向量分解矩阵。假设M是n×n矩阵，具有n个独立的（正交的）特征向量https://img-blog.csdn.net/20180104161943075?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast和n个对应的特征值https://img-blog.csdn.net/20180104162038333?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast。设U是矩阵，它的列是这些特征向量，即https://img-blog.csdn.net/20180104162209847?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast；并且Λ是对角矩阵，它的对角线元素是https://img-blog.csdn.net/20180104162336924?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，1≤i≤n。则M可以被表示为：

https://img-blog.csdn.net/20180105093229664?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

        这样，M可以被分解成3个矩阵的乘积。U称为特征向量矩阵（eigenvector matrix），而Λ称为特征值矩阵（eigenvaluematrix）。

        更一般地，任意矩阵都可以用类似的方法分解。对于一个M x N（M>=N）的矩阵M，存在以下的SVD分解：

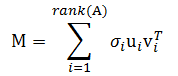
https://img-blog.csdn.net/20180105093254462?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

其中U是m×m矩阵，∑是m×n矩阵，V是n×n矩阵。U和V是标准正交矩阵，即它们的列向量都是单位长度，并且相互正交。这样，https://img-blog.csdn.net/20180104162820218?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，https://img-blog.csdn.net/20180104162900858?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，E是单位矩阵。∑是对角矩阵，其对角线元素非负，并且被排好序，使得较大的元素先出现，即https://img-blog.csdn.net/20180104162947603?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast。

        V的列向量https://img-blog.csdn.net/20180104163054371?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast是右奇异向量（rightsingular vector），U的列向量是左奇异向量（leftsingular vector）。奇异值矩阵（singular value matrix）∑的对角线元素通常记作https://img-blog.csdn.net/20180104163149458?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，称为M的奇异值（singular value）。最多存在rank(M)≤min(m,n)个非零奇异值。

        可以证明https://img-blog.csdn.net/20180104163540300?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast的特征向量是右奇异向量（即V的列），而https://img-blog.csdn.net/20180104163635118?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast的特征向量是左奇异向量（即U的列）。https://img-blog.csdn.net/20180104163700662?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast和https://img-blog.csdn.net/20180104163709942?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast的非零特征值是https://img-blog.csdn.net/20180104163810481?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast，即奇异值的平方。方阵的特征值分解可以看作奇异值分解的一个特例。

        矩阵的奇异值分解也可以用下面的等式表示。注意，尽管看上去像点积，但它并不是点积，其结果是秩为1的m×n矩阵。



        这种表示的重要性是每个矩阵都可以表示成秩为1矩阵的以奇异值为权重的加权和。由于以非递增序排列的奇异值通常下降很快，因此有可能使用少量奇异值和奇异值向量得到矩阵的很好的近似。这对于维归约是很有用的。

        数据矩阵的SVD分解具有如下性质。

* 属性中的模式被右奇异向量（即V的列）捕获。
* 对象中的模式被左奇异向量（即U的列）捕获。
* 矩阵M可以通过依次取公式https://img-blog.csdn.net/20180104164059149?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast中的项，以最优的方式不断逼近。就是说奇异值越大，该奇异值和其相关联的奇异向量决定矩阵的比例越大。

        很多情况下，前10%甚至更少的奇异值的平方就占全部奇异值平方的90%以上了，因此可以用前k个奇异值来近似描述原矩阵：

https://img-blog.csdn.net/20180105093402671?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

*k*的取值有下面的公式决定：

https://img-blog.csdn.net/20180105093415916?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

其中percentage称为“奇异值平方和占比的阈值”，一般取90%，k是一个远小于m和n的值，这样也就达到了降维的目的。

### MADlib奇异值分解函数

        MADlib的SVD函数可以对稠密矩阵和稀疏矩阵进行奇异值因式分解，并且还提供了一个稀疏矩阵的本地高性能实现函数。

**（1）  稠密矩阵的SVD函数**

        语法：

svd( source\_table,

output\_table\_prefix,

row\_id,

k,

n\_iterations,

result\_summary\_table );

        参数：

|  |  |  |
| --- | --- | --- |
| **参数名称** | **数据类型** | **描述** |
| source\_table | TEXT | 源表名（稠密矩阵数据表）。 |
| output\_table\_prefix | TEXT | 指定输出表名的前缀。 |
| row\_id | TEXT | 代表行ID的列名。 |
| k | INTEGER | 计算的奇异值个数。 |
| n\_iterations（可选） | INTEGER | 运行的迭代次数，必须在[k, 列维度数]范围内。 |
| result\_summary\_table（可选） | TEXT | 存储结果摘要的表的名称。 |

表2 svd函数参数说明

        source\_table表中含有一个row\_id列标识每一行，从数字1开始。其它列包含矩阵的数据。可以使用两种稠密格式的任何一个，例如下面示例的2 x 2矩阵。

格式一：

row\_id col1 col2

row1 1 1 0

row2 2 0 1

格式二：

row\_id row\_vec

row1 1 {1, 0}

row2 2 {0, 1}

**（2）  稀疏矩阵的SVD函数**

        表示为稀疏格式的矩阵使用此函数。为了高效计算，在奇异值分解操作之前，输入矩阵会被转换为稠密矩阵。

        语法：

svd\_sparse( source\_table,

output\_table\_prefix,

row\_id,

col\_id,

value,

row\_dim,

col\_dim,

k,

n\_iterations,

result\_summary\_table );

        参数：

|  |  |  |
| --- | --- | --- |
| **参数名称** | **数据类型** | **描述** |
| source\_table | TEXT | 源表名（稀疏矩阵数据表）。 |
| output\_table\_prefix | TEXT | 指定输出表名的前缀。 |
| row\_id | TEXT | 包含行下标的列名。 |
| col\_id | TEXT | 包含列下标的列名。 |
| value | TEXT | 包含值的列名。 |
| row\_dim | INTEGER | 矩阵的行数。 |
| col\_dim | INTEGER | 矩阵的列数。 |
| k | INTEGER | 计算的奇异值个数。 |
| n\_iterations（可选） | INTEGER | 运行的迭代次数，必须在[k, 列维度数]范围内。 |
| result\_summary\_table（可选） | TEXT | 存储结果摘要的表的名称。 |

表3 svd\_sparse函数参数说明

**（3）  稀疏矩阵的本地实现SVD函数**

        此函数在计算SVD时使用本地稀疏表示（不跨节点），能够更高效地计算稀疏矩阵，适合高度稀疏的矩阵。

        语法：

svd\_sparse\_native( source\_table,

output\_table\_prefix,

row\_id,

col\_id,

value,

row\_dim,

col\_dim,

k,

n\_iterations,

result\_summary\_table );

        参数：同svd\_sparse函数。

**（4）  输出表**

        三个SVD函数的输出都是以下三个表：

* 左奇异矩阵表：表名为<output\_table\_prefix>\_u。
* 右奇异矩阵表：表名为<output\_table\_prefix>\_v。
* 奇异值矩阵表：表名为<output\_table\_prefix>\_s。

        左右奇异向量表的格式为：

* row\_id：INTEGER类型。每个特征值对应的ID，降序排列。
* row\_vec：FLOAT8[]类型。该row\_id对应的特征向量元素，数组大小为k。

        由于只有对角线元素是非零的，奇异值表采用稀疏表格式，其中的row\_id和col\_id都是从1开始。奇异值表具有以下列：

* row\_id：INTEGER类型，第i个奇异值为i。
* col\_id：INTEGER类型，第i个奇异值为i（与row\_id相同）。
* value：FLOAT8类型，奇异值。

        除了矩阵分解得到的三个输出表外，奇异值分解函数还会输出一个结果摘要表，存储函数执行的基本情况信息，具有以下列：

* rows\_used：INTEGER类型，SVD计算使用的行数。
* exec\_time：FLOAT8类型，计算SVD使用的总时间。
* iter：INTEGER类型，迭代运行次数。
* recon\_error：FLOAT8类型，质量得分（近似精度）。公式为：https://img-blog.csdn.net/20180104165153630?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast
* relative\_recon\_error：FLOAT8类型，相对质量分数。计算公式为：https://img-blog.csdn.net/20180104165229237?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvd3p5MDYyMw==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

**（5）  联机帮助**

        可以执行下面的查询获得SVD函数的联机帮助。

select madlib.svd();

-- 用法

select madlib.svd('usage');

-- 示例

select madlib.svd('example');

### 奇异值分解函数示例

        本节我们使用稀疏SVD函数解决前面低秩矩阵分解示例中的歌曲推荐问题，但使用的不是潜在因子算法，而是另一个推荐系统的常用算法——协同过滤。

**（1）  建立输入表并生成输入数据**

        建立用户索引表。

drop table if exists tbl\_idx\_user;

create table tbl\_idx\_user (user\_idx bigserial, userid varchar(10));

        建立音乐索引表。

drop table if exists tbl\_idx\_music;

create table tbl\_idx\_music (music\_idx bigserial, musicid varchar(10));

        建立用户行为数据表。

drop table if exists source\_data;

create table source\_data (

userid varchar(10), -- 用户ID

musicid varchar(10), -- 歌曲ID

val float8 -- 用户评分

);

        建立用户评分矩阵表。

drop table if exists svd\_data;

create table svd\_data (

row\_id int, -- 行ID，从1开始，表示用户

col\_id int, -- 列ID，从1开始，表示作品

val float8 -- 分数

);

        生成用户行为数据表数据。

insert into source\_data values

('u1', 'm1', 5), ('u1', 'm6', -5),

('u2', 'm4', 3),

('u3', 'm3', 1), ('u3', 'm5', 2), ('u3', 'm7', 4),

('u4', 'm2', 4), ('u4', 'm3', 4), ('u4', 'm4', 3), ('u4', 'm7', -2),

('u5', 'm2', 5), ('u5', 'm3', -5), ('u5', 'm5', -5),

('u5', 'm7', 4), ('u5', 'm8', 3),

('u6', 'm3', 4), ('u6', 'm6', 3),

('u7', 'm2', -2), ('u7', 'm6', 5),

('u8', 'm2', -2), ('u8', 'm6', 5), ('u8', 'm8', 5),

('u9', 'm3', 1), ('u9', 'm5', 2), ('u9', 'm7', 4) ;

        从行为数据表生成用户索引表数据。

insert into tbl\_idx\_user (userid)

select distinct userid from source\_data order by userid;

        从行为数据表生成歌曲索引表数据。

insert into tbl\_idx\_music (musicid)

select distinct musicid from source\_data order by musicid;

        这里从业务数据生成有过打分行为的9个用户，以及被打过分的8首歌曲。注意查询中的排序子句，作用是便于业务ID与矩阵里的行列ID对应。

        从行为数据表生成评分矩阵表数据。

insert into svd\_data

select t1.user\_idx, t2.music\_idx, t3.val

from tbl\_idx\_user t1, tbl\_idx\_music t2, source\_data t3

where t1.userid = t3.userid and t2.musicid = t3.musicid;

        之所以要用用户行为表作为数据源，是因为矩阵中包含所有有过打分行为的用户和被打过分的歌曲，但不包括与没有任何打分行为相关的用户和歌曲。与低秩矩阵分解不同的是，如果包含无行为记录的用户或歌曲，会在计算余弦相似度时出现除零错误。正因如此，如果要用奇异值分解方法推荐没有被评过分的歌曲，或者为没有评分行为的用户形成推荐，需要做一些特殊处理，如将一个具有特别标志的虚拟用户或歌曲，用平均分数赋予初值，手工添加到评分矩阵表中。

**（2）  执行SVD**

        调用svd\_sparse\_native函数。

drop table if exists svd\_u, svd\_v, svd\_s, svd\_summary cascade;

select madlib.svd\_sparse\_native

( 'svd\_data', -- 输入表

'svd', -- 输出表名前缀

'row\_id', -- 行索引列名

'col\_id', -- 列索引列名

'val', -- 矩阵元素值

9, -- 矩阵行数

8, -- 矩阵列数

7, -- 计算的奇异值个数，小于等于最小行列数

NULL, -- 使用缺省的迭代次数

'svd\_summary' -- 概要表名

);

        选择svd\_sparse\_native函数的原因是测试数据比较稀疏，矩阵实际数据只占1/3（25/72），该函数效率较高。这里给出的行、列、奇异值个数分别为9、8、7。svd\_sparse\_native函数要求行数大于等于列数，而奇异值个数小于等于列数，否则会报错。结果U、V矩阵的行数由实际的输入数据所决定，例如测试数据最大的行值为9，最大列值为8，则结果U矩阵的行数为9，V矩阵的行数为8，而不论行、列参数的值是多少。U、V矩阵的列数、S矩阵的行列数均由奇异值个数参数所决定。

        查看SVD结果。

select array\_dims(row\_vec) from svd\_u;

select \* from svd\_s order by row\_id, col\_id;

select array\_dims(row\_vec) from svd\_v;

select \* from svd\_summary;

        结果：

dm=# select array\_dims(row\_vec) from svd\_u;

array\_dims

------------

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

(9 rows)

dm=# select \* from svd\_s order by row\_id, col\_id;

row\_id | col\_id | value

--------+--------+------------------

1 | 1 | 10.6650887159422

2 | 2 | 10.0400685494281

3 | 3 | 7.26197376834847

4 | 4 | 6.5227892843447

5 | 5 | 5.11307075598297

6 | 6 | 3.14838515537081

7 | 7 | 2.67251694708377

7 | 7 |

(8 rows)

dm=# select array\_dims(row\_vec) from svd\_v;

array\_dims

------------

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

[1:7]

(8 rows)

dm=# select \* from svd\_summary;

rows\_used | exec\_time (ms) | iter | recon\_error | relative\_recon\_error

-----------+----------------+------+----------------+----------------------

9 | 10364.79 | 8 | 0.116171249851 | 0.0523917951113

(1 row)

        可以看到，结果U、V矩阵的维度分别是9 x 7和8 x 7，奇异值是一个7 x 7的对角矩阵。这里还有一点与低秩矩阵分解函数不同，低秩矩阵分解函数由于引入了随机数，是不确定函数，相同参数的输入，可能得到不同的输出结果矩阵。但奇异值分解函数是确定的，只要入参相同，输出的结果矩阵就是一样的。

**（3）  对比不同奇异值个数的逼近程度**

        让我们按k的取值公式计算一下奇异值的比值，验证k设置为6、8时的逼近程度。

-- k=8

drop table if exists svd8\_u, svd8\_v, svd8\_s, svd8\_summary cascade;

select madlib.svd\_sparse\_native

('svd\_data', 'svd8', 'row\_id', 'col\_id', 'val', 9, 8, 8, NULL, 'svd8\_summary');

-- k=6

drop table if exists svd6\_u, svd6\_v, svd6\_s, svd6\_summary cascade;

select madlib.svd\_sparse\_native

('svd\_data', 'svd6', 'row\_id', 'col\_id', 'val', 9, 8, 6, NULL, 'svd6\_summary');

        对比逼近程度。

select \* from svd6\_summary;

select \* from svd\_summary;

select \* from svd8\_summary;

select s1/s3, s2/s3

from (select sum(value\*value) s1 from svd6\_s) t1,

(select sum(value\*value) s2 from svd\_s) t2,

(select sum(value\*value) s3 from svd8\_s) t3;

        结果：

dm=# select \* from svd6\_summary;

t3;

rows\_used | exec\_time (ms) | iter | recon\_error | relative\_recon\_error

-----------+----------------+------+----------------+----------------------

9 | 3051.51 | 8 | 0.335700790666 | 0.151396899541

(1 row)

dm=# select \* from svd\_summary;

rows\_used | exec\_time (ms) | iter | recon\_error | relative\_recon\_error

-----------+----------------+------+----------------+----------------------

9 | 6122.36 | 8 | 0.116171249851 | 0.0523917951113

(1 row)

dm=# select \* from svd8\_summary;

rows\_used | exec\_time (ms) | iter | recon\_error | relative\_recon\_error

-----------+----------------+------+-------------------+----------------------

9 | 4182.38 | 8 | 1.52006310774e-15 | 6.85529638348e-16

(1 row)

dm=# select s1/s3, s2/s3

dm-# from (select sum(value\*value) s1 from svd6\_s) t1,

dm-# (select sum(value\*value) s2 from svd\_s) t2,

dm-# (select sum(value\*value) s3 from svd8\_s) t3;

?column? | ?column?

-------------------+-------------------

0.977078978809393 | 0.997255099805013

(1 row)

        可以看到，随着k值的增加，误差越来越小。在本示例中，奇异值个数为6、7的近似度分别为97.7%和99.7%，当k=8时并没有降维，分解的矩阵相乘等于原矩阵。后面的计算都使用k=7的结果矩阵。

**（4）  基于用户的协同过滤算法UserCF生成推荐**

        所谓UserCF算法，简单说就是依据用户的相似程度形成推荐。

        定义基于用户的协同过滤函数。

create or replace function fn\_user\_cf(user\_idx int)

returns table(r2 int, s float8, col\_id int, val float8, musicid varchar(10)) as

$func$

select r2, s, col\_id, val, musicid

from

(select r2,s,col\_id,val,row\_number() over (partition by col\_id order by col\_id) rn

from

(select r2,s,col\_id,val

from

(select r2,s

from

(select r2,s,row\_number() over (order by s desc) rn

from

(select t1.row\_id r1, t2.row\_id r2,

(madlib.cosine\_similarity(v1, v2)) s

from

(select row\_id, row\_vec v1

from svd\_u where row\_id = $1) t1,

(select row\_id, row\_vec v2 from svd\_u) t2

where t1.row\_id <> t2.row\_id) t) t

where rn <=5 and s < 1) t1, svd\_data t2

where t1.r2=t2.row\_id and t2.val >=3) t

where col\_id not in (select col\_id from svd\_data where row\_id = $1)) t1,

tbl\_idx\_music t2

where t1.rn = 1 and t1.col\_id = t2.music\_idx

order by t1.s desc, t1.val desc limit 5;

$func$

language sql;

        说明：

* 最内层查询调用madlib.cosine\_similarity函数返回指定用户与其他用户的余弦相似度。

select t1.row\_id r1, t2.row\_id r2, (madlib.cosine\_similarity(v1, v2)) s

from (select row\_id, row\_vec v1 from svd\_u where row\_id = $1) t1,

(select row\_id, row\_vec v2 from svd\_u) t2

where t1.row\_id <> t2.row\_id

* 外面一层查询按相似度倒序取得排名。

select r2,s,row\_number() over (order by s desc) rn from …

* 外面一层查询取得最相近的5个用户，同时排除相似度为1的用户，因为相似度为1说明两个用户的歌曲评分一模一样，而推荐的应该是用户没有打过分的歌曲。

select r2,s from … where rn <=5 and s < 1

* 外面一层查询取得相似用户打分在3及其以上的歌曲索引ID。

select r2,s,col\_id,val from … wheret1.r2=t2.row\_id and t2.val >=3

* 外面一层查询取得歌曲索引ID的排名，目的是去重，避免相同的歌曲推荐多次，并且过滤掉被推荐用户已经打过分的歌曲。

select r2,s,col\_id,val,

row\_number() over (partition by col\_id order by col\_id) rn

from … where col\_id not in (select col\_id from svd\_data where row\_id =$1)

* 最外层查询关联歌曲索引表取得歌曲业务主键，并按相似度和打分推荐前5个歌曲。

select r2, s, col\_id, val, musicid ...

where t1.rn = 1 and t1.col\_id =t2.music\_idx

order by t1.s desc, t1.val desc limit5

        定义接收用户业务ID的函数。

create or replace function fn\_user\_recommendation(i\_userid varchar(10))

returns table (r2 int, s float8, col\_id int, val float8, musicid varchar(10)) as

$func$

declare

v\_rec record;

v\_user\_idx int:=0;

begin

select user\_idx into v\_user\_idx from tbl\_idx\_user where userid=i\_userid;

for v\_rec in (select \* from fn\_user\_cf(v\_user\_idx)) loop

r2:=v\_rec.r2;

s:=v\_rec.s;

col\_id:=v\_rec.col\_id;

val :=v\_rec.val;

musicid:=v\_rec.musicid;

return next;

end loop;

return;

end;

$func$

language plpgsql;

        通常输入的用户ID是业务系统的ID，而不是索引下标，因此定义一个接收业务系统的ID函数，内部调用fn\_user\_cf函数生成推荐。

        测试推荐结果。

select \* from fn\_user\_recommendation('u1');

select \* from fn\_user\_recommendation('u3');

select \* from fn\_user\_recommendation('u9');

        结果：

dm=# select \* from fn\_user\_recommendation('u1');

r2 | s | col\_id | val | musicid

----+---------------------+--------+-----+---------

6 | 0.0446031939822096 | 3 | 4 | m3

2 | 0.0304067953459913 | 4 | 3 | m4

5 | 0.00259120409706406 | 2 | 5 | m2

5 | 0.00259120409706406 | 7 | 4 | m7

5 | 0.00259120409706406 | 8 | 3 | m8

(5 rows)

dm=# select \* from fn\_user\_recommendation('u3');

r2 | s | col\_id | val | musicid

----+---------------------+--------+-----+---------

6 | 0.109930597010835 | 6 | 3 | m6

2 | 0.0749416547815916 | 4 | 3 | m4

5 | 0.00638637254275688 | 2 | 5 | m2

5 | 0.00638637254275688 | 8 | 3 | m8

(4 rows)

dm=# select \* from fn\_user\_recommendation('u9');

r2 | s | col\_id | val | musicid

----+---------------------+--------+-----+---------

6 | 0.109930597010835 | 6 | 3 | m6

2 | 0.0749416547815916 | 4 | 3 | m4

5 | 0.00638637254275688 | 2 | 5 | m2

5 | 0.00638637254275688 | 8 | 3 | m8

(4 rows)

        可以看到，因为u3和u9的评分完全相同，相似度为1，所以为他们生成的推荐也完全相同。

**（5）  基于歌曲的协同过滤算法ItemCF生成推荐**

        所谓ItemCF算法，简单说就是依据歌曲的相似程度形成推荐。

        定义基于歌曲的协同过滤函数。

create or replace function fn\_item\_cf(user\_idx int)

returns table(r2 int, s float8, musicid varchar(10)) as

$func$

select t1.r2, t1.s, t2.musicid

from (select t1.r2,t1.s,row\_number() over (partition by r2 order by s desc) rn

from (select t1.\*, row\_number() over (partition by r1 order by s desc) rn

from (select t1.row\_id r1, t2.row\_id r2, (madlib.cosine\_similarity(v1, v2)) s

from (select row\_id, row\_vec v1

from svd\_v

where row\_id in (select col\_id from svd\_data where row\_id=$1)) t1,

(select row\_id, row\_vec v2

from svd\_v

where row\_id not in (select col\_id from svd\_data where row\_id=$1)) t2

where t1.row\_id <> t2.row\_id) t1) t1

where rn <=3) t1, tbl\_idx\_music t2

where rn = 1 and t1.r2 = t2.music\_idx

order by s desc;

$func$

language sql;

        说明：

* 最内层查询调用madlib.cosine\_similarity函数返回指定用户打过分的歌曲与没打过分的歌曲的相似度。

select t1.row\_id r1, t2.row\_id r2, (madlib.cosine\_similarity(v1, v2)) s

from (select row\_id, row\_vec v1

from svd\_v

where row\_id in (select col\_id from svd\_data where row\_id=$1)) t1,

(select row\_id, row\_vec v2

from svd\_v

where row\_id not in (select col\_id from svd\_data where row\_id=$1)) t2

where t1.row\_id <> t2.row\_id

* 外面一层查询按相似度倒序取得排名。

select t1.\*, row\_number() over (partitionby r1 order by s desc) rn …

* 外面一层查询取得与每个打分歌曲相似度排前三的歌曲，并以歌曲索引ID分区，按相似度倒序取得排名，目的是去重，避免相同的歌曲推荐多次。

select t1.r2,t1.s,row\_number() over(partition by r2 order by s desc) rn

from … where rn <=3

* 最外层查询关联歌曲索引表取得歌曲业务主键并推荐。

select t1.r2, t1.s, t2.musicid

from ... where rn = 1 and t1.r2 = t2.music\_idx order by s desc

        定义接收用户业务ID的函数。

create or replace function fn\_item\_recommendation(i\_userid varchar(10))

returns table (r2 int, s float8, musicid varchar(10)) as

$func$

declare

v\_rec record;

v\_user\_idx int:=0;

begin

select user\_idx into v\_user\_idx from tbl\_idx\_user where userid=i\_userid;

for v\_rec in (select \* from fn\_item\_cf(v\_user\_idx)) loop

r2:=v\_rec.r2;

s:=v\_rec.s;

musicid:=v\_rec.musicid;

return next;

end loop;

return;

end;

$func$

language plpgsql;

        通常输入的用户ID是业务系统的ID，而不是索引下标，因此定义一个接收业务系统的ID函数，内部调用fn\_item\_cf函数生成推荐。

        测试推荐结果。

select \* from fn\_item\_recommendation('u1');

select \* from fn\_item\_recommendation('u3');

select \* from fn\_item\_recommendation('u9');

        结果：

dm=# select \* from fn\_item\_recommendation('u1');

r2 | s | musicid

----+--------------------+---------

3 | 0.120167300602806 | m3

7 | 0.063463750764015 | m7

4 | 0.0474991338483946 | m4

(3 rows)

dm=# select \* from fn\_item\_recommendation('u3');

r2 | s | musicid

----+--------------------+---------

2 | 0.211432380780157 | m2

4 | 0.125817242051348 | m4

1 | 0.120167300602806 | m1

6 | 0.115078854619122 | m6

8 | 0.0747298357682603 | m8

(5 rows)

dm=# select \* from fn\_item\_recommendation('u9');

r2 | s | musicid

----+--------------------+---------

2 | 0.211432380780157 | m2

4 | 0.125817242051348 | m4

1 | 0.120167300602806 | m1

6 | 0.115078854619122 | m6

8 | 0.0747298357682603 | m8

(5 rows)

        可以看到，因为u3和u9的评分作品完全相同，相似度为1，所以按作品相似度为他们生成的推荐也完全相同。

**（6）  为新用户寻找相似用户**

        假设一个新用户u10的评分向量为'{0,4,5,3,0,0,-2,0}'，要利用已有的奇异值矩阵找出该用户的相似用户。

        1. 添加行为数据

insert into source\_data

values ('u10', 'm2', 4), ('u10', 'm3', 5), ('u10', 'm4', 3), ('u10', 'm7', -2);

insert into tbl\_idx\_user (userid)

select distinct userid

from source\_data

where userid not in (select userid from tbl\_idx\_user)

order by userid;

        2. 确认从评分[向量计算](https://www.baidu.com/s?wd=%E5%90%91%E9%87%8F%E8%AE%A1%E7%AE%97&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)svd\_u向量的公式

u10[1:8] x svd\_v[8:7] x svd\_s[7:7]^-1

        3. 生成u10用户的向量表和数据

drop table if exists mat\_u10;

create table mat\_u10(row\_id int, row\_vec float8[]);

insert into mat\_u10 values (1, '{0,4,5,3,0,0,-2,0}');

        4. 根据计算公式，先将前两个矩阵相乘

drop table if exists mat\_r\_10;

select madlib.matrix\_mult('mat\_u10', 'row=row\_id, val=row\_vec',

'svd\_v', 'row=row\_id, val=row\_vec',

'mat\_r\_10');

        5. 根据公式，求奇异值矩阵的逆矩阵

drop table if exists svd\_s\_10;

create table svd\_s\_10 as

select row\_id, col\_id,1/value val from svd\_s where value is not ;

        6. 根据公式，将4、5两步的结果矩阵相乘。注意 4 的结果mat\_r\_10是一个稠密矩阵，5 的结果svd\_s\_10是一个稀疏矩阵。

drop table if exists matrix\_r\_10;

select madlib.matrix\_mult('mat\_r\_10', 'row=row\_id, val=row\_vec',

'svd\_s\_10', 'row=row\_id, col=col\_id, val=val',

'matrix\_r\_10');

        7. 查询与u10相似的用户

select t1.row\_id r1, t2.row\_id r2, madlib.cosine\_similarity(v1, v2) s

from (select row\_id, row\_vec v1 from matrix\_r\_10 where row\_id = 1) t1,

(select row\_id, row\_vec v2 from svd\_u) t2

order by s desc;

        结果：

dm=# select t1.row\_id r1, t2.row\_id r2, madlib.cosine\_similarity(v1, v2) s

dm-# from (select row\_id, row\_vec v1 from matrix\_r\_10 where row\_id = 1) t1,

dm-# (select row\_id, row\_vec v2 from svd\_u) t2

dm-# order by s desc;

r1 | r2 | s

----+----+----------------------

1 | 4 | 0.989758250631095

1 | 6 | 0.445518586781384

1 | 2 | 0.117185108937363

1 | 8 | 0.00673214822878822

1 | 1 | -0.0026200076051774

1 | 5 | -0.00988637492741542

1 | 3 | -0.0276611552976064

1 | 9 | -0.0276611552976064

1 | 7 | -0.253951334956948

(9 rows)

        可以看到，u10与u4的相似度高达99%，从原始的评分向量可以得到验证：

u4：'{0,4,4,3,0,0,-2,0}'

u10：'{0,4,5,3,0,0,-2,0}'

        8. 将结果向量插入svd\_u矩阵

insert into svd\_u

select user\_idx, row\_vec from matrix\_r\_10, tbl\_idx\_user where userid = 'u10';

# [数据转换之其它转换](https://blog.csdn.net/wzy0623/article/details/79008881)

# [数据探索之描述性统计](https://blog.csdn.net/wzy0623/article/details/79071818)

# [数据探索之概率统计](https://blog.csdn.net/wzy0623/article/details/79088215)

# [数据探索之主成分分析](https://blog.csdn.net/wzy0623/article/details/79160959)

# [回归之线性回归](https://blog.csdn.net/wzy0623/article/details/79196758)

# [回归之广义线性模型](https://blog.csdn.net/wzy0623/article/details/79205296)

# [回归之逻辑回归](https://blog.csdn.net/wzy0623/article/details/79217198)

# [回归之多类回归](https://blog.csdn.net/wzy0623/article/details/79227278)

# [回归之序数回归](https://blog.csdn.net/wzy0623/article/details/79237961)

# [回归之弹性网络回归](https://blog.csdn.net/wzy0623/article/details/79260968)

# [回归之Cox比例风险回归](https://blog.csdn.net/wzy0623/article/details/79268589)

# [回归之稳健方差](https://blog.csdn.net/wzy0623/article/details/79279689)

# [回归之聚类方差](https://blog.csdn.net/wzy0623/article/details/79288581)

# [时间序列分析之ARIMA](https://blog.csdn.net/wzy0623/article/details/79310475)

# [分类之KNN](https://blog.csdn.net/wzy0623/article/details/79442414)

# [分类之朴素贝叶斯](https://blog.csdn.net/wzy0623/article/details/79456926)

# [分类之SVM](https://blog.csdn.net/wzy0623/article/details/79480880)

# [分类之决策树](https://blog.csdn.net/wzy0623/article/details/79504754)

# [分类之随机森林](https://blog.csdn.net/wzy0623/article/details/79529590)

# [聚类之k-means方法](https://blog.csdn.net/wzy0623/article/details/79537352)

# [关联规则之Apriori算法](https://blog.csdn.net/wzy0623/article/details/79550159)

# [图算法之单源最短路径](https://blog.csdn.net/wzy0623/article/details/79564814)

# [模型评估之交叉验证](https://blog.csdn.net/wzy0623/article/details/79578574)

# [模型评估之预测度量](https://blog.csdn.net/wzy0623/article/details/79579081)