第 1 章

◂ MADlib 基础 ▸

MADlib 是一个基于 SQL 的数据库内置的开源机器学习库，具有良好的并行度和可扩展性，有高度的预测精准度。MADlib 最初由 Pivotal 公司与伯克利大学合作开发，提供了多种数据转换、数据探索、概率统计、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学习，以满足各行各业的应用需求。用户可以非常方便地将 MADlib 加载到数据库中，从而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。MADlib 1.14 可以与 PostgreSQL、Greenplum 和 HAWQ 等数据库系统无缝集成。

本章首先介绍 MADlib 的一些基本概念及其有别于其他机器学习工具包的特点。为了更好地使用 MADlib，我们将简要说明它的设计思想、工作原理、执行流程和基础架构，还将罗列 MADlib 支持的模型和主要功能模块，最后说明 MADlib 软件包的安装与卸载。

1.1 基本概念

1.1.1 MADlib 是什么

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，具体来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及应用的全过程。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应用需求。

MADlib 具有与上述工具完全不同的设计理念，不是面向程序员，而是面向数据库开发人员或 DBA。如果要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一套技术，一套牢固扎根于数学和应用数学的技能（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，对于广大传统数据库应用技术人员来说，就可将他们长期积累的数据库操作技能复用到机器学习领域，使转型更加轻松。现在，鱼和熊掌兼得的机会来了，DBA 只要使用 MADlib，就能用 SQL 查询实现简单的机器学习。

对用户而言，MADlib 提供了可在 SQL 查询语句中调用的函数，即可以用 select + function name 的方式来调用这个库。这就意味着，所有的数据调用和计算都在数据库内完成而不需要数据的导入导出。MADlib 不仅包括基本的线性代数运算和统计函数，还提供了常用的、现成的机器学习模型函数。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、提供正确的入参并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，虽然 MADlib 提供了使用方便性、降低了学习和使用门槛，但是相对于其他机器学习系统而言，其灵活性与功能完备性显然是短板。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 函数库是毫无意义的，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，就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机器学习系统软件，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 快速完成简单的机器学习工作，比较适合做一些简单的、特征相对明显的机器学习。

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。

1.1.2 MADlib 的设计思想

驱动 MADlib 架构的关键设计思想体现在以下方面：

- 操作数据库内的本地数据，避免在多个运行时环境之间不必要地移动数据。
- 充分利用数据库引擎功能，但将机器学习逻辑从数据库特定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。
- 开放实施，保持与 Apache 社区的积极联系和持续的学术研究。

操作本地数据的思想与 Hadoop 是一致的。为了使全局的带宽消耗和 I/O 延迟降到尽可能小，在选择数据时，MADlib 总是选择距离读请求最近的存储节点。如果在读请求所在节点的同一个主机上有需要的数据副本，那么 MADlib 会尽量选择它来满足读请求。如果数据库集群跨越多个数据中心，那么存储在本地数据中心的副本会优先于远程副本被选择。

MADlib 库表现为数据库内置的函数。当函数在 SQL 语句中执行时，可以充分利用数据库引擎提供的功能。例如，在 HAWQ 中执行 MADlib 函数时，每个物理 Segment 在执行查询的时候会启动多个查询执行器（Query Executor，QE），使得单一物理 Segment 看起来就像多个虚拟 Segment，从而使 HAWQ 能够更好地利用所有可用资源。虚拟 Segment 是内存、CPU 等资源的容器，每个虚拟 Segment 都含有为查询启动的一个 QE。查询就是在虚拟 Segment 中被 QE 所执行的。

MADlib 利用 Greenplum 或 HAWQ 数据库系统使用的 MPP（Massively Parallel Processing，大规模并行处理）架构，使用户能够获益于经过锤炼的基于 MPP 的分析功能及其查询性能，兼顾了低延时与高扩展。

MADlib 现已成为 Apache 顶级项目，其整个项目和代码是在 Apache 上开源的，开发是在 Pivotal 的支持下基于 Apache 社区的，与社区有很好的互动。

1.1.3　MADlib 的工作原理

现以 HAWQ 上的 MADlib 为例解释它的工作原理。图 1-1 是 HAWQ 的架构。当一个客户端查询向 HAWQ 发出请求时，Master 节点会对查询进行处理，根据查询成本、资源队列定义、数据局部化和当前系统中的资源使用情况，为查询规划资源分配。之后查询被分发到 Segment 节点所在的物理主机并行处理，可能是节点子集或整个集群。每个 Segment 节点上的资源实施器监控着查询对资源的实时使用情况，避免异常资源占用。查询处理完成后，最后的结果再通过 Master 返回客户端。

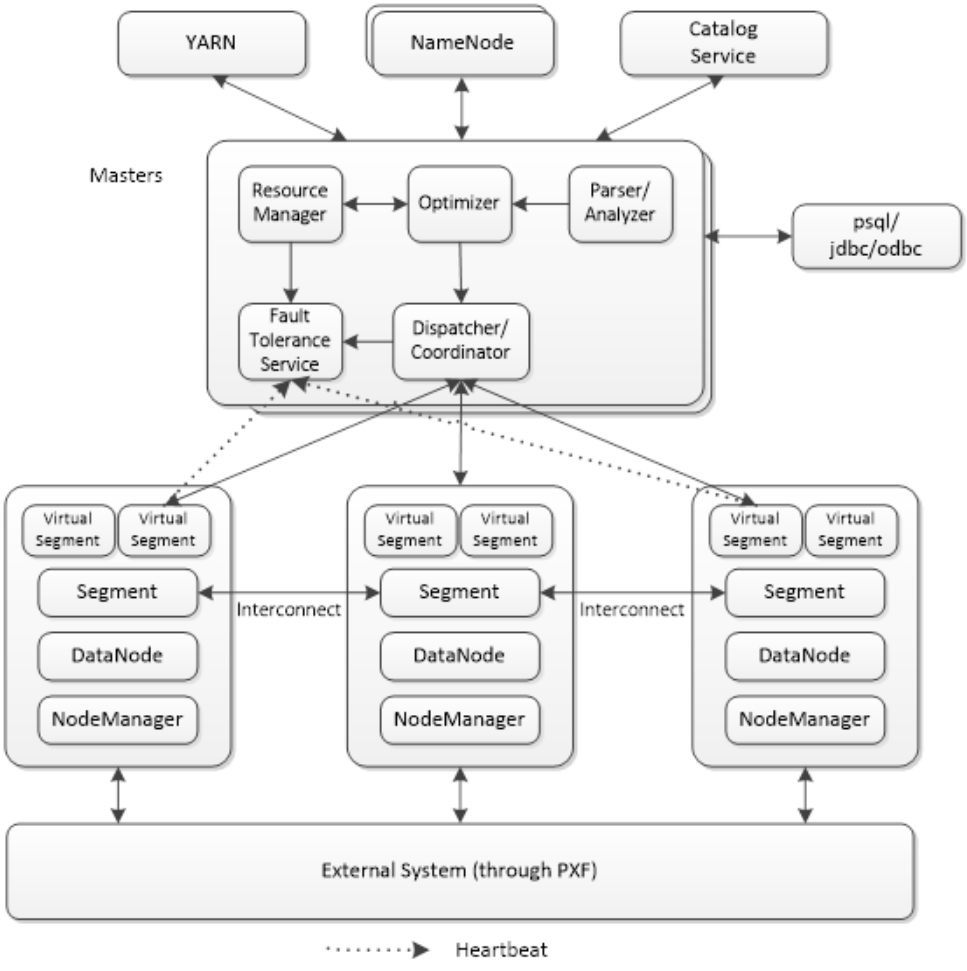


图 1-1　HAWQ 架构

MADlib 就是构建在 HAWQ 架构之上的，通过定义 HAWQ 上的 UDA 和 UDF 建立 In-Database Functions。当我们使用 SQL 调用 MADlib 时，MADlib 会首先进行输入的有效性判断和数据的预处理，将处理后的查询传给 HAWQ，之后所有的计算即等同于普通的查询处理请求在 HAWQ 内执行。图 1-2 显示了 MADlib 在 HAWQ 上的工作原理。

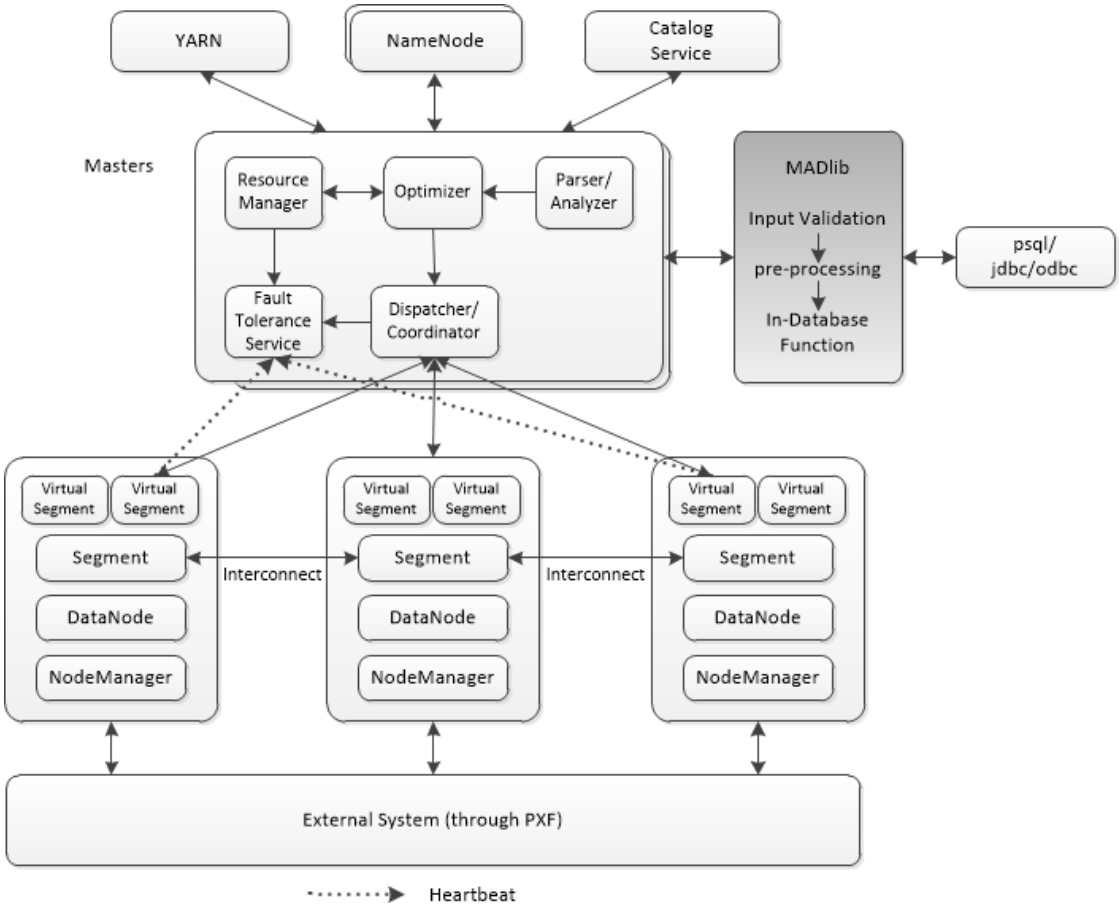


图 1-2　MADlib 在 HAWQ 上的工作原理

1.1.4　MADlib 的执行流程

图 1-3 中是整个 MADlib 函数调用过程的执行流程。在客户端，我们可以使用 Jupyter、Zeppelin、psql 等工具连接数据库并调用 MADlib Function。MADlib 预处理后根据具体算法生成多个查询传入数据库服务器，之后数据库服务器执行查询并返回 String（一般是一个或多个存放结果的表）。

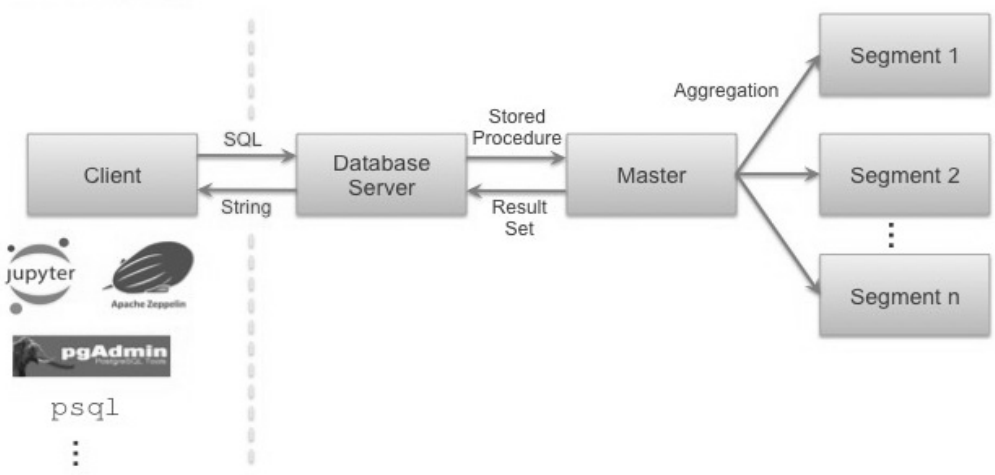


图 1-3　MADlib 执行流程

1.1.5　MADlib 架构

MADlib 架构如图 1-4 所示。

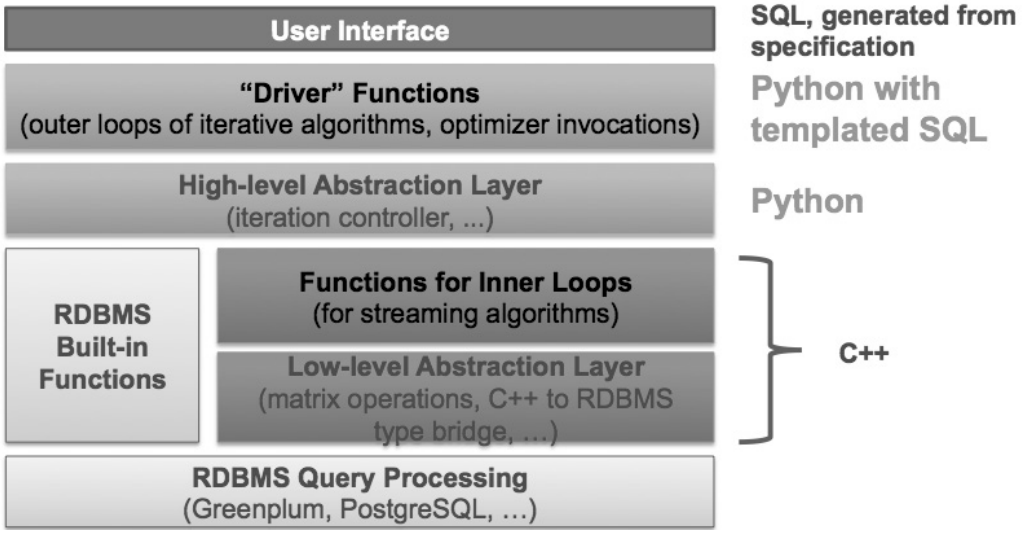


图 1-4　MADlib 架构

处于架构最上面一层的是用户接口。如前所述，用户只需通过在 SQL 查询语句中调用 MADlib 提供的函数来完成机器学习工作。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、HAWQ 等数据库管理系统，最终由它们处理查询请求。中间四层是构成 MADlib 的组件。从图 1-4 中可以看到，MADlib 系统架构自上而下由四个主要组件构成：

（1）Python 调用 SQL 模板实现的驱动函数

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 语句直至达到某些收敛标准。

（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，对所有的 PostgreSQL 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，从而使用户将关注点集中在内部功能而不是平台集成上。

1.2 MADlib 的功能

1.2.1 MADlib 支持的模型类型

MADlib 支持以下常用机器学习模型类型，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，我们通常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，就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

关联规则有时又叫作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他组的数据项更相似。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。这里的相似通常特指在文本领域中具有相同主题的文档。注意，MADlib 的当前实现并不支持中文分词。

（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，但是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，可能会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有助于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

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，所以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用于模型验证。

1.2.2　MADlib 的主要功能模块

MADlib 的主要功能模块如图 1-5 所示。

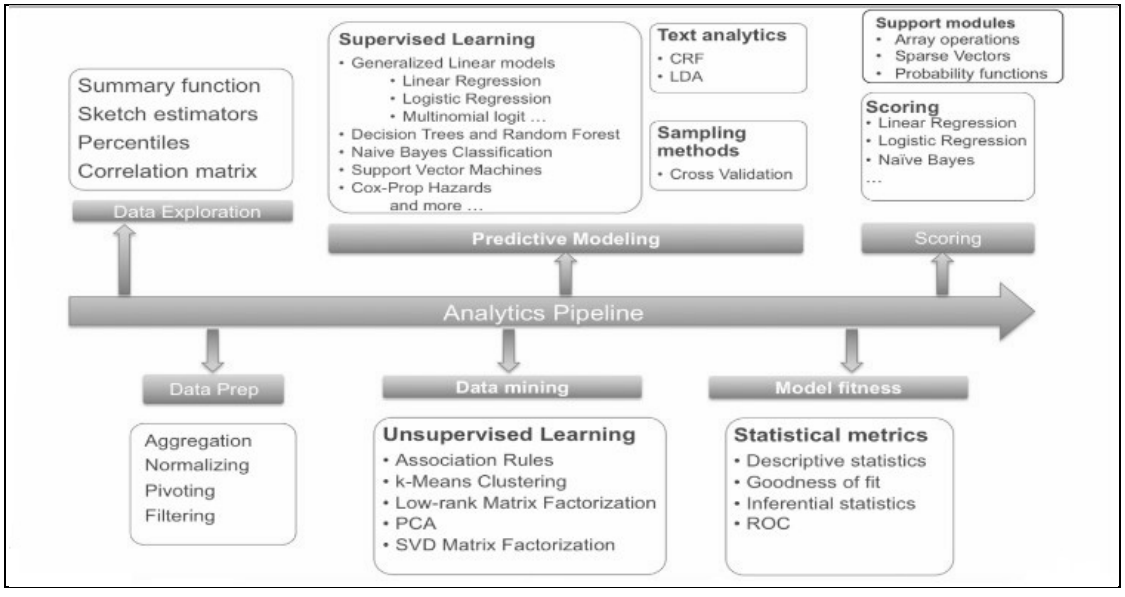


图 1-5　MADlib 主要功能模块

下面基于 MADlib 1.10 版本预览 MADlib 提供的具体模型算法或功能。

（1）Data Types and Transformations（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（皮尔森相关系数）
    - 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）

1.3 MADlib 的安装与卸载

1.3.1 确定安装平台

MADlib 1.14 可以安装在 PostgreSQL、Greenplum 和 HAWQ 中。在不同的数据库系统，安装过程不尽相同。这里以在 HAWQ 2.1.1.0 中安装 MADlib 为例，演示 MADlib 的安装与卸载过程。后面章节进行的一系列示例也都在此实验环境中进行的。HAWQ 的安装部署过程从略。

机器学习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。

比照以上机器学习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。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用在此场景下的作用非常有限。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优异的查询性能，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 是实现大数据机器学习比较合理的选择。

1.3.2 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 MADlib 安装文件，如图 1-6 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。

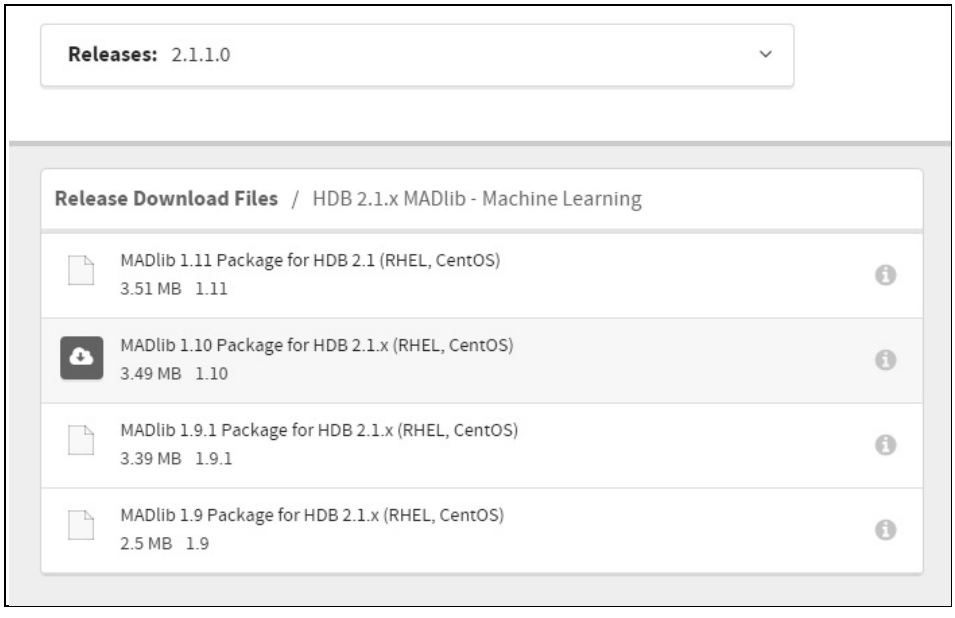


图 1-6 下载 MADlib 安装文件

1.3.3 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，默认目录为/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 madlib schema，-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。所有机器学习的模型、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，就说明 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]$
```

1.3.4 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）删除其他遗留数据库对象

① 删除模式。如果模型验证过程中途出错，那么数据库中可能包含测试的模式（模式名称前缀都是 madlib_installcheck_），只能手工执行 SQL 命令删除，例如：

```
drop schema madlib_installcheck_kmeanscascade;
```

② 删除用户。如果存在遗留的测试用户，就将其删除，例如：

```
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
```

1.4 小结

不同于其他机器学习工具，MADlib 是一个基于 SQL 的数据库内置的可扩展机器学习库。其语法是基于 SQL 的，也就是说，可以用 select + function name 的方式来调用这个库。这意味着 MADlib 需要在数据库系统中使用，所有的数据调用和计算都在数据库内完成而不需要数据的导入导出。MADlib 是一个运行在大规模并行处理数据库系统上的应用，因此可扩展性非常好，能够处理较大量级的数据，目前支持 PostgreSQL、Greenplum 和 HAWQ。MADlib 具有强大的数据分析能力，支持大量的机器学习、数据分析和统计算法。MADlib 项目和代码在 Apache 社区开源，现已成为 Apache 软件基金会的顶级项目。

第 2 章

◄ 数据类型 ►

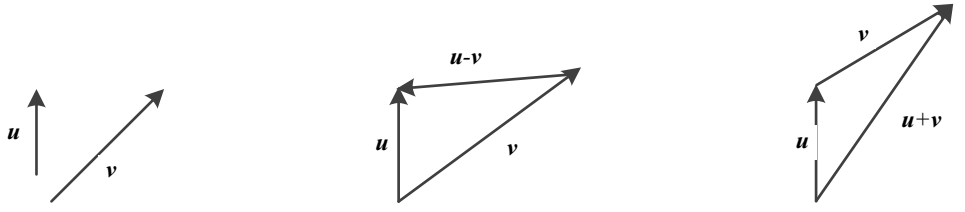
通常机器学习操作的数据集可以看作数据对象的集合。数据对象有时也叫作记录、点、向量、模式、事件、案例、样本、观测或实体。数据对象用一组刻画对象基本特征的属性描述，如物体质量、事件发生的时间等。属性有时也叫作变量、字段、特征或维。在数学上，向量和矩阵可以用来表示数据对象及其属性。

和其他机器学习语言或工具一样，MADlib 操作的基本对象也是向量与矩阵。在 MADlib 中，对向量和矩阵的操作是通过一系列函数完成的。

本章将介绍 MADlib 中向量和矩阵的概念，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。这里侧重于应用，因为理解这些函数的意义和用法是使用 MADlib 进行机器学习的基础。

2.1 向量

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，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 2-1（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 45°。图 2-1（b）和图 2-1（c）分别以有向线段表示两个向量的差与和。



（a）两个向量　　（b）两个向量的差　　（c）两个向量的和

图 2-1　两个向量以及它们的和与差

2.1.1　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，是机器学习算法的支持模块。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 2-1 所示。

表 2-1　MADlib 数组运算函数

函数名称	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，就返回 TRUE
array_max()	返回数组中的最大值，忽略空值，返回与输入相同的数据类型
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，返回与输入相同的数据类型
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。该函数的作用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()	该函数创建元素个数为参数值的 FLOAT8 数组，初始值为 0.0
array_of_bigint()	该函数创建元素个数为参数值的 BIGINT 数组，初始值为 0

（续表）

函数名称	描述
array_fill()	该函数将数组每个元素设置为参数值
array_filter()	该函数过滤掉数组中等于指定值的元素，要求是一维数组并且所有值非空，返回与输入相同的数据类型，默认移除所有 0 值
array_scalar_mult()	该函数将一个数组作为输入，元素与第二个参数指定的标量值相乘，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()	该函数将一个数组作为输入，元素与第二个参数指定的标量值相加，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()	该函数以数组和一个 FLOAT8 为输入，返回由每个元素的乘幂（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，使元素平方和为 1，要求是一维数组并且所有值非空

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$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$。其中，$\mu$表示数组元素平均值。

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 $\boldsymbol{u}$、$\boldsymbol{v}$ 和 $\boldsymbol{w}$ 是 3 个向量，那么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：$\boldsymbol{u}+\boldsymbol{v}=\boldsymbol{v}+\boldsymbol{u}$。
- 向量加法的结合律，相加时向量分组不影响结果：$(\boldsymbol{u}+\boldsymbol{v})+\boldsymbol{w}=\boldsymbol{u}+(\boldsymbol{v}+\boldsymbol{w})$。
- 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 $\boldsymbol{u}$，有 $\boldsymbol{u}+0=\boldsymbol{u}$。
- 向量加法逆元存在性，对于每个向量 $\boldsymbol{u}$，都存在一个逆向量$-\boldsymbol{u}$，使得 $\boldsymbol{u}+(-\boldsymbol{u})=0$。

（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，若标量是正则方向不变，若标量为负则方向相反。假设 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 是向量、α 和 β 是标量（数），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：$\alpha(\beta\boldsymbol{u})=(\alpha\beta)\boldsymbol{u}$。

```
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：$(\alpha+\beta)\boldsymbol{u}=\alpha\boldsymbol{u}+\beta\boldsymbol{u}$。

```
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：$\alpha(\boldsymbol{u}+\boldsymbol{v})=\alpha\boldsymbol{u}+\alpha\boldsymbol{v}$。

```
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 $\alpha = 1$，那么对于任何向量 $\boldsymbol{u}$ 都有 $\alpha\boldsymbol{u} = \boldsymbol{u}$。

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$\boldsymbol{u_1}$，…，$\boldsymbol{u}_n$是基向量，那对于任意向量 $\boldsymbol{v}$，都可以找到 n 个标量的集合{α_1，…，α_n}使得 $\boldsymbol{v} = \sum_{i=1}^{n} \alpha_i \boldsymbol{u}_i$。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，那么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 $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{n-1}, \boldsymbol{v}_n)$引述向量 $\boldsymbol{v}$ 的分量。注意，$\boldsymbol{v}_i$是向量 $\boldsymbol{v}$ 的一个分量，而$\boldsymbol{v}_i$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，我们只需要用标量乘以每个分量即可，如 3×(2,3) = (6,9)。

（5）数组乘除。注意，这里过滤掉 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 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 的点积 $\boldsymbol{u} \cdot \boldsymbol{v}$ 的定义为：$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} \boldsymbol{u}_i \boldsymbol{v}_i$。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;
```

由点积的定义来说明何谓两个向量正交。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 90º。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：

```
{0.0592348877759092,0.118469775551818,0.177704663327728,0.236939551103637,0.29
6174438879546,0.355409326655455,0.41464421
4431365,0.473879102207274,0.533113989983183}
{0.0100600363590491,0.0100600363590491,0,0,0.0100600363590491,0.020120072718090
82,0.0301801090771473,0.995943599545862,0.
0804802908723929}
(2 rows)
```

点积也可以用来计算欧式空间中的向量长度：length($\boldsymbol{u}$)=$\sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}$。向量长度又称范数（norm），并记作 $\|\boldsymbol{u}\|$。给定一个向量 $\boldsymbol{u}$，我们可以通过用其长度除 $\boldsymbol{u}$ 的每个分量（通过计算 $\boldsymbol{u}/\|\boldsymbol{u}\|$）找到一个向量，它与 $\boldsymbol{u}$ 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L_2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：

$$\boldsymbol{u} \cdot \boldsymbol{v} = \| \boldsymbol{u} \| \| \boldsymbol{v} \| \cos(\theta)$$

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：

$$\boldsymbol{u} \cdot \boldsymbol{v} = (\| \boldsymbol{v} \| \cos(\theta)) \| \boldsymbol{u} \| = \boldsymbol{v}_{\boldsymbol{u}} \| \boldsymbol{u} \|$$

其中，$\boldsymbol{v}_{\boldsymbol{u}} = \| \boldsymbol{v} \| \cos(\theta)$表示向量 $\boldsymbol{v}$ 在 $\boldsymbol{u}$ 的方向上的长度，如图 2-2 所示。如果 $\boldsymbol{u}$ 是单位向量，那么该点积是 $\boldsymbol{v}$ 在 $\boldsymbol{u}$ 方向上的分量，称为 $\boldsymbol{v}$ 在 $\boldsymbol{u}$ 上的正交投影（orthogonal projection）。当然，如果 $\boldsymbol{v}$ 是单位向量，那么该点积也是 $\boldsymbol{u}$ 在 $\boldsymbol{v}$ 方向上的投影。

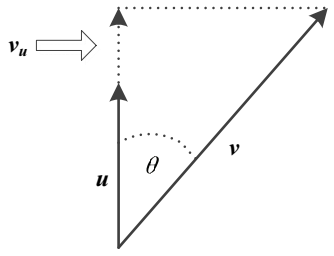


图 2-2　向量 $\boldsymbol{v}$ 在向量 $\boldsymbol{u}$ 方向的正交投影

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他向量的线性组合，那么该集合是线性独立的。如果一个向量集不是线性独立的，那么它们是线性依赖的（linear 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，就会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 个元素的数组并将数组元素的值设为 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 函数构造一个包含 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 函数默认过滤掉数组中的 0 元素，如果给出了第二个元素，就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，就返回原数组。

（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 版本并无此函数，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)
```

2.1.2　稀疏向量

有些数据集具有非对称特征，一个对象的大部分属性值都为 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，有些机器学习算法仅适合处理稀疏数据。

1. 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。浮点数组可进行各种计算，有时会有很多的零或其他默认值，在科学计算、零售优化、文本处理等应用中是很常见的。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。

MADlib 1.10 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，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。

2. 创建稀疏向量

可以利用以下四种方式创建稀疏向量。

（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}
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}
row)
```

（3）使用聚合函数创建一个 svec，例如：

```
 dm=# select madlib.svec_agg(v1) from generate_series(1,10) v1;
                svec_agg
-----------------------------------------------
 {1,1,1,1,1,1,1,1,1,1}:{1,2,3,4,5,6,7,8,9,10}
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，其他位置的值为 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。

3. 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[]，结果就是一个 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 $j + n - 1 \leqslant m$，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}
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}');
```

如果忽略词的顺序，文档就可以用词向量表示，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）。文档是矩阵的行，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 中的每个元素都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，更容易理解这一点：

```
dm=# \x
Expanded display is on.
dm=# select madlib.svec_sfv((select a from features),b)::float8[], b
dm-#   from documents;
-[ RECORD 1 ]-------------------------------------------
svec_sfv | {0,0,0,0,1,1,0,1,1,0,0,0,1,0,0,1,0,1,0}
b        | {this,is,one,document,in,the,corpus}
-[ RECORD 2 ]-------------------------------------------
svec_sfv | {1,0,0,0,1,1,1,1,0,0,0,0,0,0,1,2,0,0,0}
b        | {i,am,the,second,document,in,the,corpus}
-[ RECORD 3 ]-------------------------------------------
svec_sfv | {0,0,1,1,0,0,0,0,0,1,1,1,0,1,0,0,1,0,1}
b        | {being,third,never,really,bothered,me,until,now}
-[ RECORD 4 ]-------------------------------------------
svec_sfv | {0,1,0,0,0,2,0,0,1,1,0,0,0,0,0,2,1,0,0}
b        | {the,document,before,me,is,the,third,document}
```

可以看到文档"i am the second document in the corpus"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。最普通的权重叫作 tf/idf，对应术语是 Term Frequency / InverseDocument Frequency（词频-逆文档频率）。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，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资料库其中一个文档的重要程度。词的重要性随着它在文档中出现的次数成正比增加，但同时会随着它在资料库中出现的频率成反比下降。简单来说就是一个词在一篇文档中出现的次数越多，同时在所有文档中出现的次数越少，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}:{0,0.693147180559945,0.287682072451781,0,0.69314718
0559945,0,1.38629436111989,0,0.287682072451781,0,1.38629436111989,0}
      2 |
{1,3,1,1,1,1,6,1,1,3}:{1.38629436111989,0,0.693147180559945,0.287682072451781,
1.38629436111989,0.693147180559945,0,1.38629436111989,0.575364144903562,0}
      3 | {2,2,5,1,2,1,1,2,1,1,1}:{0,1.38629436111989,0,0.693147180559945,
1.38629436111989,0,1.38629436111989,0,0.693147180559945,0,1.38629436111989}
```

```
     4 | {1,1,3,1,2,2,5,1,1,2}:{0,1.38629436111989,0,0.575364144903562,0,
0.693147180559945,0,0.575364144903562,0.693147180559945,0}
(4 rows)
```

尽管文档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，如果统计 0-0 匹配，那么大多数文档都会与其他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 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 是两个文档向量，则：

$$\cos(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}\cdot\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}$$

其中，“·”表示向量点积，$\boldsymbol{x}\cdot\boldsymbol{y} = \sum_{k=1}^{n}\boldsymbol{x}_k\boldsymbol{y}_k$，$\|\boldsymbol{x}\|$是向量 $\boldsymbol{x}$ 的长度，$\|\boldsymbol{x}\| = \sqrt{\sum_{k=1}^{n}\boldsymbol{x}_k^2} = \sqrt{\boldsymbol{x}\cdot\boldsymbol{x}}$。

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

```
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。

2.2 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。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。本节将介绍矩阵的概念，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。

2.2.1　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“$m \times n$ 矩阵”通常用来说明矩阵具有 m 行和 n 列。下面所示的矩阵 $\boldsymbol{A}$ 是 2×3 矩阵。如果 m=n，则我们称该矩阵为方阵（square matrix）。矩阵 $\boldsymbol{A}$ 的转置记作 $\boldsymbol{A}^{\mathrm{T}}$（通过交换 $\boldsymbol{A}$ 的行和列得到）。

$$\boldsymbol{A} = \begin{bmatrix} 2 & 6 & 1 \\ 7 & 5 & 2 \end{bmatrix} \qquad \boldsymbol{A}^{\mathrm{T}} = \begin{bmatrix} 2 & 7 \\ 6 & 5 \\ 1 & 2 \end{bmatrix}$$

矩阵的元素用带下标的小写字母表示。对于矩阵 $\boldsymbol{A}$，a_{ij}是其第 i 行第 j 列的元素。行自上而下编号，列自左向右编号，编号从 1 开始。例如，a_{21}是矩阵 $\boldsymbol{A}$ 的第 2 行第 1 列的元素，该元素的值是 7。

矩阵的每一行或列定义一个向量。对于矩阵 $\boldsymbol{A}$，其第 i 个行向量（row vector）可以用a_{i*}表示，第 j 个列向量（column vector）用a_{*j}表示。使用前面的例子，a_{2*} = [7 5 2]，而a_{*3} = $[1\ 2]^{\mathrm{T}}$。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。

2.2.2　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。

1. 稠密

矩阵被表示为一维数组的行集合，例如 3×10 的矩阵如下：

```
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，即行向量。

2. 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 4×7 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 列应该是标量简单数据类型。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}
```

2.2.3 MADlib 中的矩阵运算函数

与向量操作相同，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）。

1. 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)
-- 矩阵相乘
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)
```

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多个节点执行的。

2. 矩阵操作函数示例

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 by row_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）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。

（5）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。

（6）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)
```

（7）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’，表示结果中包含最大最小值对应的下标数组列。

（8）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，元素值全为 0，所以稀疏表示的矩阵表只有 1 行。

（9）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)
```

因为元素值全为 1，所以稀疏表示的矩阵表有 6 行。

（10）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)
```

（11）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 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 都是 $m \times n$ 的矩阵，$\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的和是 $m \times n$ 矩阵 $\boldsymbol{C}$，其元素由下式计算：

$$c_{ij} = a_{ij} + b_{ij}$$

```
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，分别是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，一个矩阵为稠密形式，一个矩阵为稀疏形式，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 two matrices 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：$\boldsymbol{A}+\boldsymbol{B}=\boldsymbol{B}+\boldsymbol{A}$。
- 矩阵加法的结合律。相加时矩阵分组不影响结果：$(\boldsymbol{A}+\boldsymbol{B})+\boldsymbol{C}=\boldsymbol{A}+(\boldsymbol{B}+\boldsymbol{C})$。
- 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 $\boldsymbol{A}$，有 $\boldsymbol{A}+0=\boldsymbol{A}$。
- 矩阵加法逆元的存在性。对于每个矩阵 $\boldsymbol{A}$，都存在一个矩阵-$\boldsymbol{A}$，使得 $\boldsymbol{A}+(-\boldsymbol{A})=0$。-$\boldsymbol{A}$ 的元素为$-a_{ij}$。

（12）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 $\boldsymbol{A}$ 的乘积是矩阵 $\boldsymbol{B}=\alpha\boldsymbol{A}$，其元素由下式给出：

$$b_{ij} = \alpha a_{ij}$$

例如，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：$\alpha(\beta\boldsymbol{A})=(\alpha\beta)\boldsymbol{A}$。
- 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：$(\alpha+\beta)\boldsymbol{A}=\alpha\boldsymbol{A}+\beta\boldsymbol{A}$。
- 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：$\alpha(\boldsymbol{A}+\boldsymbol{B})=\alpha\boldsymbol{A}+\alpha\boldsymbol{B}$。
- 标量单位元的存在性，如果 α=1，则对于任意矩阵 $\boldsymbol{A}$，有 $\alpha\boldsymbol{A}=\boldsymbol{A}$。

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。

（13）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：$m\times n$ 矩阵 $\boldsymbol{A}$ 乘以 $n\times1$ 的列矩阵 $\boldsymbol{u}$ 的积是 $m\times1$ 的列矩阵 $\boldsymbol{v}=\boldsymbol{A}\boldsymbol{u}$，其元素由下式给出：

$$v_i=\mathrm{a}_{i*}\cdot\boldsymbol{u}^{\mathrm{T}}$$

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

$$\begin{bmatrix}3 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}5\\ 2\end{bmatrix}=\begin{bmatrix}17\\ 9\end{bmatrix}$$

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：$1\times m$ 的行矩阵 $\boldsymbol{u}$ 乘以 $m\times n$ 矩阵 $\boldsymbol{A}$ 的积是 $1\times n$ 的行矩阵 $\boldsymbol{v}=\boldsymbol{u}\boldsymbol{A}$，其元素由下式给出：

$$v_i=\boldsymbol{u}\cdot(\mathrm{a}_{*j})^{\mathrm{T}}$$

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

$$\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}5 & 4\\ 2 & 9\end{bmatrix}=\begin{bmatrix}9 & 22\end{bmatrix}$$

MADlib 的 matrix_vec_mult 函数用于计算一个 $m\times n$ 矩阵乘以一个 $1\times n$ 的矩阵（向量），结果是一个 $1\times m$ 的矩阵。如下面的 5×4 的矩阵 mat_b 乘以一个 1×4 的矩阵，结果是一个 1×5 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。$m \times n$ 矩阵 $\boldsymbol{A}$ 与 $n \times p$ 矩阵 $\boldsymbol{B}$ 的积是 $m \times p$ 矩阵 $\boldsymbol{C}$（$\boldsymbol{C}=\boldsymbol{AB}$），其元素由下式给出：

$$c_{\mathrm{ij}} = \mathrm{a}_{i*} \cdot (b_{*j})^{\mathrm{T}}$$

换言之，$\boldsymbol{C}$ 的第 ij 个元素是 $\boldsymbol{A}$ 的第 i 个行向量与 $\boldsymbol{B}$ 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: Matrix error: Dimension mismatch for matrix
multiplication. (plpython.c:4663)
DETAIL:  Left matrix, col dimension = 4, Right matrix, row dimension = 5
CONTEXT:  Traceback (most recent 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 4×5 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：$(\boldsymbol{AB})\boldsymbol{C}=\boldsymbol{A}(\boldsymbol{BC})$。
- 矩阵乘法的分配率，矩阵乘法对矩阵加法是可分配的：$\boldsymbol{A}(\boldsymbol{B}+\boldsymbol{C})=\boldsymbol{AB}+\boldsymbol{AC}$ 并且$(\boldsymbol{B}+\boldsymbol{C})\boldsymbol{A}=\boldsymbol{BA}+CA$。
- 矩阵乘法单位元的存在性，若$\boldsymbol{I}_p$是 $p\times p$ 矩阵的单位矩阵，则对于任意 $m\times n$ 矩阵 $\boldsymbol{A}$，$\boldsymbol{AI}_n=\boldsymbol{A}$ 并且$\boldsymbol{I}_m\boldsymbol{A}=\boldsymbol{A}$。

一般地，矩阵乘法是不可交换的，即 $\boldsymbol{AB}\neq\boldsymbol{BA}$。

如果我们有一个 $n\times 1$ 列向量 $\boldsymbol{u}$，我们就可以把 $m\times n$ 矩阵 $\boldsymbol{A}$ 被该向量右乘看作 $\boldsymbol{u}$ 到 m 维列向量 $\boldsymbol{v}=\boldsymbol{Au}$ 的变换。类似地，如果我们用一个（行）向量 $\boldsymbol{u}=[\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m]$左乘 $\boldsymbol{A}$，我们可以将它看作 $\boldsymbol{u}$ 到 n 维行向量 $\boldsymbol{v}=\boldsymbol{uA}$ 的变换。这样，我们可以把一个任意 $m\times n$ 矩阵 $\boldsymbol{A}$ 看作一个把一个向量映射到另一个向量空间的函数。

（14）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 $\boldsymbol{C}=\boldsymbol{AB}$，$\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 均为 $m\times n$ 矩阵，$\boldsymbol{C}$ 的元素由下式给出：

$$c_{\mathrm{ij}}=a_{\mathrm{ij}}\times b_{\mathrm{ij}}$$

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)
```

（15）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(double
precision[],integer,integer,integer,integer,double precision)": Invalid col
id. (UDF_impl.hpp:210)  (seg20 hdp4:40000 pid=123035) (plpython.c:4663)
CONTEXT:  Traceback (most recent call last):
  PL/Python function "matrix_rank", line 23, in <module>
    return matrix_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/Python function "matrix_rank"
dm=#
```

矩阵的秩（rank of a matrix）常常用来刻画矩阵。设矩阵 $\boldsymbol{A}=(a_{ij})_{m\times n}$，在 $\boldsymbol{A}$ 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 $\boldsymbol{A}$ 的一个 k 阶子式。$m\times n$ 矩阵 $\boldsymbol{A}$ 共有 $c_m^k c_n^k$ 个 k 阶子式。若 $\boldsymbol{A}$ 有 r 阶子式不为 0，任何 r+1 阶子式（如果存在的话）全为 0，则称 r 为矩阵 $\boldsymbol{A}$ 的秩，记作 $R(\boldsymbol{A})$。

矩阵的秩具有以下基本性质：

- 0 矩阵的秩为 0。
- 若 $R(\boldsymbol{A})=r$，则 $\boldsymbol{A}$ 中至少有一个 r 阶子式 $D_r \neq 0$，所有 r+1 阶子式为 0，且更高阶子式均为 0，r 是 $\boldsymbol{A}$ 中非零子式的最高阶数。
- 矩阵转置，秩不变。
- $0 \leqslant R(\boldsymbol{A}) \leqslant \min(m,n)$。
- 若 $\boldsymbol{A}$ 是 $n\times n$ 方阵，并且 $|\boldsymbol{A}|\neq 0$，则 $R(\boldsymbol{A})=n$；反之，若 $R(\boldsymbol{A})=n$，则 $|\boldsymbol{A}|\neq 0$。

矩阵的秩（rank of a matrix）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，把一个 $1\times n$ 的行向量复制 m 次，产生一个 $m\times n$ 的矩阵，则我们只有一个秩为 1 的矩阵。

（16）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 by row_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。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，那 matrix_inverse 函数会报如下错误：

```
Matrix error: Inverse operation is only defined for square matrices
```

（17）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 by row_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}
(4 rows)
```

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 5 ×4 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}
(4 rows)
```

（18）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)
```

（19）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。默认为 frobenius 范数。

```
select madlib.matrix_norm('mat_b_sparse', 'row=row_id, col=col_id, val=val');
```

结果：

```
  matrix_norm
---------------
 23.4520787991
(1 row)
```

F-范数的公式为：$\|\boldsymbol{A}\|_{\mathrm{F}}=(\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{\mathrm{ij}}|^{2})^{\frac{1}{2}}$。依据公式可知下面查询的结果与 matrix_norm 函数的返回值相等。

```
select sqrt(sum(power(val,2))) from mat_b_sparse;
```

（20）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。

2.3 小结

本章介绍了 MADlib 的基本数据类型，向量与矩阵。在 MADlib 中，向量和矩阵分别用数据库中的一维数组和二维数组数据类型存储。MADlib 提供了丰富的向量和矩阵操作函数，这里对大部分函数给出了例子说明。这些函数虽然简单，但是作为 MADlib 的基础模块非常重要。数据分析的对象通常被抽象为向量或矩阵，而许多机器学习的复杂算法内部也是对向量或矩阵进行计算。