MLlib支持存储在单个机器上的局部向量和矩阵，以及由一个或多个RDD支持的分布式矩阵。局部向量和局部矩阵是用作公共接口的简单数据模型。其底层线性代数运算由Breeze提供。在监督学习中使用的训练示例在MLlib中称为“labeled point”。

局部向量(Local Vector)

局部向量具有整数类型和基于0的索引和双精度浮点型，存储在单个机器上。MLlib支持两种类型的局部向量：密集(dense)和稀疏(sparse)。密集向量由表示其条目值的double数组支持，而稀疏向量由两个并行数组支持：索引数组和值数组。例如，矢量（1.0,0.0,3.0）可以以密集格式表示为[1.0,0.0,3.0]，或者以稀疏格式表示为（3，[0,2]，[1.0,3.0]），其中3是 矢量的大小。

局部向量的基类是Vector，我们提供了两种实现：DenseVector和SparseVector。我们建议使用Vector中实现的工厂方法来创建局部向量。

|  |
| --- |
| 代码示例 |
| **package** com.cb.spark.mllib;  **import** org.apache.spark.mllib.linalg.Vectors;  **import** org.apache.spark.mllib.linalg.Vector;  **import** org.apache.spark.sql.SparkSession;  **public** **class** DataType {  **public** **static** **void** main(String[] args) {  SparkSession session=SparkSession.*builder*().appName("DataType").master("local").getOrCreate();    //创建一个密集向量  Vector dv=Vectors.*dense*(1.0,0.0,3.0);    //创建一个稀疏向量，需要指定相关的索引苏数组和值数组  Vector sv=Vectors.*sparse*(3, **new** **int**[]{0,2},**new** **double**[]{1.0,3.0});    System.***out***.println(dv);  System.***out***.println(sv);  }  } |

**2.标记点(Labeled point)**

**标记点(Labeled point)是与标签/响应(response)相关联的密集或稀疏的局部矢量。**在MLlib中，标记点用于监督学习算法。我们使用double**类型**来存储标签，因此我们可以在回归和分类中使用标记点。对于二进制分类，标签应为0（负）或1（正）。 对于多类分类，标签应该是从零开始的类索引：0,1,2，....

标记点由LabeledPoint**类**表示。

|  |
| --- |
| //使用正标签和一个密集特征向量创建一个标记点  LabeledPoint pos=**new** LabeledPoint(1.0, Vectors.*dense*(1.0,0.0,3.0));    //使用负标签和一个稀疏特征向量创建一个标记点  LabeledPoint neg=**new** LabeledPoint(0.0, Vectors.*sparse*(3, **new** **int**[]{0,2},**new** **double**[]{1.0,3.0})); |

稀疏数据：

在实践中很常见的是稀疏的训练数据。MLlib支持读取以LIBSVM格式存储的训练样例，这是LIBSVM和LIBLINEAR使用的默认格式。它是一种文本格式，其中每一行使用以下格式表示标记的稀疏特征向量：

label index1:value1 index2:value2 ...

其中索引是一个基于升序的。加载后，**特征**索引将转换为从零开始。

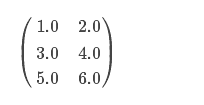
MLUtils.loadLibSVMFile读取以LIBSVM格式存储的训练样例。

|  |
| --- |
| RDD<LabeledPoint> rdd = MLUtils.*loadLibSVMFile*(context.sc(), "./src/main/resources/libsvm\_data\_test"); |

**3.**局部矩阵

局部矩阵具有整数类型的行和列索引以及**double**类型值，存储在单个机器上。MLlib支持密集矩阵，其条目值以列主要顺序存储在单个**double类型**数组中**;也支持**稀疏矩阵，其非零条目值以列主要顺序存储在压缩稀疏列（CSC）格式中。

例如，以下密集矩阵



存储在具有矩阵大小（3,2）的一维数组[1.0,3.0,5.0,2.0,4.0,6.0]中。

Matrix的基类是Matrix，我们提供两种实现：DenseMatrix和SparseMatrix。

我们建议使用Matrices中实现的工厂方法来创建局部矩阵。 请记住，MLlib中的本地矩阵以列主顺序存储。

|  |
| --- |
| //创建一个密集矩阵，3行2列  Matrix dm=Matrices.*dense*(3, 2, **new** **double**[]{1.0,3.0,5.0,2.0,4.0,6.0});  System.***out***.println(dm);  //创建一个稀疏矩阵  Matrix sm=Matrices.*sparse*(3, 2, **new** **int**[]{0,1,3},**new** **int**[]{0,2,1},**new** **double**[]{9,6,8});  System.***out***.println(sm); |

**4.分布式矩阵**

**分布式矩阵具有long类型的行和列索引以及double类型值，分布式地存储在一个或多个RDD中。选择正确的格式来存储大型和分布式矩阵非常重要。将分布式矩阵转换为不同的格式可能需要全局混洗(shuffle)，这需要很大的代价。到目前为止已经实现了四种类型的分布式矩阵。**

**基本类型称为RowMatrix。RowMatrix是面向行的分布式矩阵，没有有意义的行索引，例如特征向量的集合。它由行(row)的RDD支持，其中每行(row)是本地向量。我们假设RowMatrix的列数不是很大，因此单个本地向量可以合理地传递给驱动程序，也可以使用单个节点进行存储/操作。IndexedRowMatrix类似于RowMatrix但具有行索引，可用于标识行和执行连接。CoordinateMatrix是以坐标列表（COO）格式存储的分布式矩阵，由其条目的RDD支持。BlockMatrix是由MatrixBlock的RDD支持的分布式矩阵，它是（Int，Int，Matrix）的元组。**

**分布式矩阵的基础RDD必须是确定性的，因为我们缓存矩阵大小。 通常，使用非确定性RDD可能导致错误。**

**4.1 RowMatrix**

**RowMatrix是面向行的分布式矩阵，没有有意义的行索引，由其行的RDD支持，其中每行是本地向量。由于每一行都由局部向量表示，因此列数受整数范围的限制，但实际上它应该小得多。**

**可以从JavaRDD <Vector>实例创建RowMatrix。 然后我们可以计算其列摘要统计信息。**

|  |
| --- |
| JavaRDD<Vector> rows = context  .parallelize(Arrays.*asList*(Vectors.*dense*(1.0, 2.0, 3.0), Vectors.*dense*(2.0, 5.0, 4.0)));  RowMatrix matrix = **new** RowMatrix(rows.rdd());    //获取矩阵大小  **long** m=matrix.numRows(); //2  **long** n=matrix.numCols(); //3  System.***out***.println(m+" "+n);  // QR分解  QRDecomposition<RowMatrix, Matrix> result = matrix.tallSkinnyQR(**false**);  System.***out***.println(result.Q());    System.***out***.println(result.R()); |

### 4.2IndexedRowMatrix

**IndexedRowMatrix类似于RowMatrix但具有有意义的行索引。 它由索引行的RDD支持，因此每行由其索引（long-typed）和本地向量表示。**

|  |
| --- |
| JavaRDD<IndexedRow> rows = ... // a JavaRDD of indexed rows  // Create an IndexedRowMatrix from a JavaRDD<IndexedRow>.  IndexedRowMatrix mat = new IndexedRowMatrix(rows.rdd());  // Get its size.  long m = mat.numRows();  long n = mat.numCols();  // Drop its row indices.  RowMatrix rowMat = mat.toRowMatrix(); |

### 4.3 CoordinateMatrix

**CoordinateMatrix是由其条目的RDD支持的分布式矩阵。每个条目都是（i：Long，j：Long，value：Double）的元组，其中i是行索引，j是列索引，value是条目值。只有当矩阵的两个维度都很大且矩阵非常稀疏时，才应使用CoordinateMatrix。**

**可以从JavaRDD <MatrixEntry>实例创建CoordinateMatrix，其中MatrixEntry是一个包装器（long，long，double）。通过调用toIndexedRowMatrix，可以将CoordinateMatrix转换为具有稀疏行的IndexedRowMatrix。目前不支持CoordinateMatrix的其他计算。**

|  |
| --- |
| JavaRDD<MatrixEntry> entries = ... *// a JavaRDD of matrix entries*  *// Create a CoordinateMatrix from a JavaRDD<MatrixEntry>.*  CoordinateMatrix mat = **new** CoordinateMatrix(entries.rdd());  *// Get its size.*  long m = mat.numRows();  long n = mat.numCols();  *// Convert it to an IndexRowMatrix whose rows are sparse vectors.*  IndexedRowMatrix indexedRowMatrix = mat.toIndexedRowMatrix(); |

### 4.4 BlockMatrix

**BlockMatrix是由MatrixBlocks的RDD支持的分布式矩阵，其中MatrixBlock是（（Int，Int），Matrix）的元组，其中（Int，Int）是块的索引，而Matrix是给定索引处的子矩阵，其大小为rowsPerBlock x colsPerBlock。 BlockMatrix支持添加和乘以另一个BlockMatrix等方法。BlockMatrix还有一个辅助函数validate，可用于检查BlockMatrix是否设置正确。**

**通过调用toBlockMatrix，可以从IndexedRowMatrix或CoordinateMatrix轻松创建BlockMatrix。toBlockMatrix默认创建大小为1024 x 1024的块。 用户可以通过toBlockMatrix（rowsPerBlock，colsPerBlock）提供值来更改块大小。**

|  |
| --- |
| JavaRDD<MatrixEntry> entries = ... *// a JavaRDD of (i, j, v) Matrix Entries*  *// Create a CoordinateMatrix from a JavaRDD<MatrixEntry>.*  CoordinateMatrix coordMat = **new** CoordinateMatrix(entries.rdd());  *// Transform the CoordinateMatrix to a BlockMatrix*  BlockMatrix matA = coordMat.toBlockMatrix().cache();  *// Validate whether the BlockMatrix is set up properly. Throws an Exception when it is not valid.*  *// Nothing happens if it is valid.*  matA.validate();  *// Calculate A^T A.*  BlockMatrix ata = matA.transpose().multiply(matA); |