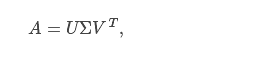
# 1.奇异值分解（SVD）

奇异值分解（SVD）将矩阵分解为三个矩阵：U，Σ和V，使得



其中,

* U是一个标准正交矩阵，其列称为左奇异向量，
* Σ是一个对角矩阵，非负对角线按降序排列，其对角线称为奇异值，
* V是一个标准正交矩阵，其列称为右奇异向量。

对于大型矩阵，通常我们不需要完全因子分解，只需要最大的若干个奇异值及其相关的奇异向量。这可以节省存储，去噪并恢复矩阵的低秩结构。

如果我们保留最大k个奇异值，那么得到的低秩矩阵的维数将是：



我们假设n小于m。 奇异值和右奇异向量是从Gramian矩阵ATA的特征值和特征向量导出的。如果用户通过computeU参数请求,存储右奇异向量U的矩阵是通过矩阵乘法U=A(VS-1)计算得到的。使用的实际方法是根据计算成本自动确定的：

* 如果n很小（n <100）或k与n（k> n / 2）相比较大，我们首先计算Gramian矩阵，然后在驱动器上本地计算其顶部特征值和特征向量。这需要在每个执行程序和驱动程序上单次传递O（n2）存储空间，并且在驱动程序上需要O（n2k）时间。
* 否则，我们以分布方式计算（ATA）v并将其发送到ARPACK以计算（ATA）在驱动器节点上的顶部特征值和特征向量。

|  |
| --- |
| **package** com.cb.spark.mllib;  **import** java.util.Arrays;  **import** java.util.List;  **import** org.apache.spark.SparkConf;  **import** org.apache.spark.api.java.JavaRDD;  **import** org.apache.spark.api.java.JavaSparkContext;  **import** org.apache.spark.mllib.linalg.Matrix;  **import** org.apache.spark.mllib.linalg.SingularValueDecomposition;  **import** org.apache.spark.mllib.linalg.Vector;  **import** org.apache.spark.mllib.linalg.Vectors;  **import** org.apache.spark.mllib.linalg.distributed.RowMatrix;  **public** **class** SVDExample {  **public** **static** **void** main(String[] args) {  SparkConf conf = **new** SparkConf().setAppName("SVDExample").setMaster("local");  JavaSparkContext sc = **new** JavaSparkContext(conf);  List<Vector> data = Arrays.*asList*(Vectors.*sparse*(5, **new** **int**[] { 1, 3 }, **new** **double**[] { 1.0, 7.0 }),  Vectors.*dense*(**new** **double**[] { 2.0, 0.0, 3.0, 4.0, 5.0 }),  Vectors.*dense*(**new** **double**[] { 4.0, 0.0, 6.0, 7.0 }));  JavaRDD<Vector> rows = sc.parallelize(data);  RowMatrix mat = **new** RowMatrix(rows.rdd());  SingularValueDecomposition<RowMatrix, Matrix> svd = mat.computeSVD(5, **true**, 1.0E-9d);  RowMatrix U = svd.U();  Vector s = svd.s();  Matrix v = svd.V();  System.***out***.println(U.numCols() + " " + U.rows());  System.***out***.println(s);  System.***out***.println(v);  }  } |

2.主成分分析

主成分分析（PCA）是一种统计方法，用于查找旋转，使得第一个坐标具有尽可能大的方差，并且每个后续坐标又具有可能的最大方差。旋转矩阵的列称为主要成分。 PCA广泛用于降维。

以下代码演示了如何计算RowMatrix上的主要成分，并使用它们将向量投影到低维空间。

|  |
| --- |
| **package** com.cb.spark.mllib;  **import** java.util.Arrays;  **import** java.util.List;  **import** org.apache.spark.SparkConf;  **import** org.apache.spark.api.java.JavaRDD;  **import** org.apache.spark.api.java.JavaSparkContext;  **import** org.apache.spark.mllib.linalg.Matrix;  **import** org.apache.spark.mllib.linalg.Vector;  **import** org.apache.spark.mllib.linalg.Vectors;  **import** org.apache.spark.mllib.linalg.distributed.RowMatrix;  **public** **class** JavaSVDExample {  **public** **static** **void** main(String[] args) {  SparkConf conf = **new** SparkConf().setAppName("SVDExample").setMaster("local");  JavaSparkContext sc = **new** JavaSparkContext(conf);  List<Vector> data = Arrays.*asList*(Vectors.*sparse*(5, **new** **int**[] { 1, 3 }, **new** **double**[] { 1.0, 7.0 }),  Vectors.*dense*(**new** **double**[] { 2.0, 0.0, 3.0, 4.0, 5.0 }),  Vectors.*dense*(**new** **double**[] { 4.0, 0.0, 0.0, 6.0, 7.0 }));  JavaRDD<Vector> rows = sc.parallelize(data);  RowMatrix mat = **new** RowMatrix(rows.rdd());  Matrix pc = mat.computePrincipalComponents(4);  System.***out***.println(pc);  RowMatrix projected = mat.multiply(pc);  projected.rows().toJavaRDD().foreach(v -> System.***out***.println(v));  }  } |