Dimensionality Reduction - RDD-based API（降维）

- Singular value decomposition (SVD)（奇异值分解）
  - Performance（性能）
  - SVD Example（SVD例子）
- Principal component analysis (PCA)

原文链接 : https://spark.apache.org/docs/latest/mllib-dimensionality-reduction.html

译文链接 : http://www.apache.wiki/pages/viewpage.action?pageId=10027073

贡献者 : 程威 片刻 ApacheCN Apache中文网

降维是减少所考虑的变量数量的过程.它可以用于从原始和噪声特征中提取潜在特征,或者在保持结构的同时压缩数据.spark.mlib 提供对 RowMatrix class 降维的支持.

Singular value decomposition (SVD)（奇异值分解）

Singular value decomposition (SVD)（奇异值分解）将矩阵分解为三个矩阵: U, Σ, 和 V,例如

$$A = U\Sigma V^T,$$

解释 :

- U是一个正交矩阵（方阵）,其列被称为左奇异向量.
- Σ是一个对角线元素的值为非负数的对角矩阵,其对角线元素被称为奇异值.
- V是一个正交矩阵,其列称为右奇异向量.

对于大型的矩阵,通常我们不需要完整的因式分解,而只需要顶点奇异值及相关的奇异向量.这可以节省存储,去噪和恢复矩阵的低阶结构.

如果我们保留k个奇异值（将n维空间映射到k维空间）,则所得到的低阶矩阵的维数将是 :

- U: m×k
- Σ: k×k
- V: n×k

Performance（性能）

我们假设 n 的数量低于 m 的数量.奇异值和右奇异向量派生于 Gramian（格拉姆） 矩阵

$A^T A$.

的特征值和特征向量.如果用户通过 computeU 参数请求,矩阵储存左奇异向量U,通过矩阵相乘计算得到

$U = A(VS^{-1})$

.实际使用的方法是通过计算成本来自动确定 :

- 如果 n 是较小的（n<100）或者 k 是较大的(k>n/2)，我们先计算 Gramian（格拉姆）矩阵,然后在驱动程序上本地计算 top（明显的）特征值和特征向量.这需要一次运算伴随着在执行器和驱动程序上的

  $O(n^2)$

  的空间复杂度,和在驱动程序上的

  $O(n^2 k)$

  的时间复杂度.
- - 否则,我们以分发的方式计算

  $(A^T A)v$

  并发送到 ARPACK 来计算

  $A^T A$.

  在驱动节点上的 top（明显的）特征值和特征向量.这需要运算O(k)次,消耗执行器上的O(n)的空间复杂度和驱动程序上的O(nk)的

空间复杂度.

SVD Example（SVD例子）

spark.mllib 为面向行的矩阵提供SVD功能,在RowMatrix类中提供.

Scala

```
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
val data = Array(
  Vectors.sparse(5, Seq((1, 1.0), (3, 7.0))),
  Vectors.dense(2.0, 0.0, 3.0, 4.0, 5.0),
  Vectors.dense(4.0, 0.0, 0.0, 6.0, 7.0))
val dataRDD = sc.parallelize(data, 2)
val mat: RowMatrix = new RowMatrix(dataRDD)
// 5
val svd: SingularValueDecomposition[RowMatrix, Matrix] =
mat.computeSVD(5, computeU = true)
val U: RowMatrix = svd.U  // URowMatrix.
val s: Vector = svd.s  // .
val V: Matrix = svd.V  // V.
```

有关API的详细信息，请参阅SingularValueDecomposition Scala文档

在 Spark repo "examples/src/main/scala/org/apache/spark/examples/mllib/SVDExample.scala" 中查找完整示例代码.

如果 U 被定义为 IndexedRowMatrix ,则相同的代码适用于 IndexedRowMatrix.

Principal component analysis (PCA)

Principal component analysis (PCA)
是一种统计方法来找到一个旋转，使得第一个坐标具有最大的方差可能，每个后续坐标依次具有最大的方差。旋转矩阵的列称为主要组件。 PCA广泛用于降维。

spark.mllib 支持 PCA，用于以行为导向的格式和任何向量存储的高和低密度矩阵。

以下代码演示如何计算 RowMatrix 上的主要组件，并使用它们将向量投影到低维空间中

Scala

```
import org.apache.spark.mllib.linalg.Matrix
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.distributed.RowMatrix

val data = Array(
  Vectors.sparse(5, Seq((1, 1.0), (3, 7.0))),
  Vectors.dense(2.0, 0.0, 3.0, 4.0, 5.0),
  Vectors.dense(4.0, 0.0, 0.0, 6.0, 7.0))

val dataRDD = sc.parallelize(data, 2)

val mat: RowMatrix = new RowMatrix(dataRDD)

// Compute the top 4 principal components.
// Principal components are stored in a local dense matrix.
val pc: Matrix = mat.computePrincipalComponents(4)

// Project the rows to the linear space spanned by the top 4
principal components.
val projected: RowMatrix = mat.multiply(pc)
```

有关API的详细信息，请参阅 RowMatrix Scala docs

完整的示例代码在Spark repo 的"examples/src/main/scala/org/apache/spark/examples/mllib/PCAOnRowMatrixExample.scala" 文件中.

以下代码演示如何计算源向量中的主成分，并使用它们将向量投影到低维空间中，同时保留相关标签：

Scala

```
import org.apache.spark.mllib.feature.PCA
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.rdd.RDD

val data: RDD[LabeledPoint] = sc.parallelize(Seq(
  new LabeledPoint(0, Vectors.dense(1, 0, 0, 0, 1)),
  new LabeledPoint(1, Vectors.dense(1, 1, 0, 1, 0)),
  new LabeledPoint(1, Vectors.dense(1, 1, 0, 0, 0)),
  new LabeledPoint(0, Vectors.dense(1, 0, 0, 0, 0)),
  new LabeledPoint(1, Vectors.dense(1, 1, 0, 0, 0))))

// Compute the top 5 principal components.
val pca = new PCA(5).fit(data.map(_.features))

// Project vectors to the linear space spanned by the top 5 principal
// components, keeping the label
val projected = data.map(p => p.copy(features =
pca.transform(p.features)))
```

有关API的详细信息，请参阅 PCA Scala docs文档

完整的代码在Spark repo 的'examples/src/main/scala/org/apache/spark/examples/mllib/PCAOnSourceVectorExample.scala'文件中.

为了运行上述应用程序，请按照Spark快速入门指南的 Self-Contained Applications 部分中提供的说明进行操作。确保将spark-mllib包含在您的构建文件中作为依赖。