聚类是一种无监督的学习问题，目标是基于一些相似概念将实体的子集彼此分组。聚类通常用于探索性分析和/或作为分层监督学习管道的组成部分（其中针对每个聚类训练不同的分类器或回归模型）。

spark.mllib 包支持下列模型:

* [K-means](http://spark.apache.org/docs/latest/mllib-clustering.html#k-means)
* [Gaussian mixture](http://spark.apache.org/docs/latest/mllib-clustering.html#gaussian-mixture)

# 1.K-means

K-means是最常用的聚类算法之一，它将数据点聚类为预定义数量的簇。spark.mllib实现包括一个名为kmeans||的k-means ++方法的并行变体。spark.mllib中的实现具有以下参数：

* **K:**是所需簇的数量。 注意，可以返回少于k个簇，例如，如果用于训练的数据点少于k。
* **maxIterations**:是要运行的最大迭代次数。
* **initializationMode:**指定随机初始化或通过k-means初始化
* run:此参数自Spark 2.0.0起无效。
* **initializationSteps:**确定k-means算法||中的步数。
* **epsilon:**确定我们认为k-means已收敛的距离阈值。
* **initialModel**:是一组用于初始化的可选集群中心。 如果提供此参数，则仅执行一次运行。

MLlib的所有方法都使用Java友好类型，因此您可以像在Scala中一样导入和调用它们。唯一需要注意的是，这些方法使用Scala RDD对象，而Spark Java API使用单独的JavaRDD类。您可以通过在JavaRDD对象上调用.rdd（）将Java RDD转换为Scala。

|  |
| --- |
| **package** com.cb.spark.mllib;  **import** org.apache.spark.SparkConf;  **import** org.apache.spark.SparkContext;  **import** org.apache.spark.api.java.JavaRDD;  **import** org.apache.spark.mllib.clustering.KMeans;  **import** org.apache.spark.mllib.clustering.KMeansModel;  **import** org.apache.spark.mllib.linalg.Vector;  **import** org.apache.spark.mllib.linalg.Vectors;  **public** **class** JavaKMeansExample {  **public** **static** **void** main(String[] args) {  SparkConf conf = **new** SparkConf().setAppName("JavaKMeansExample").setMaster("local");  SparkContext sc = **new** SparkContext(conf);  String path = "F:\\Learning\\java\\project\\LearningSpark\\src\\main\\resources\\kmeans\_data.txt";  JavaRDD<String> data = sc.textFile(path, 2).toJavaRDD();  JavaRDD<Vector> parseData = data.map(s -> {  String[] sarray = s.split(" ");  **double**[] values = **new** **double**[sarray.length];  **for** (**int** i = 0; i < sarray.length; i++) {  values[i] = Double.*parseDouble*(sarray[i]);  }  **return** Vectors.*dense*(values);  });  parseData.cache();    **int** numClusters=2;//两个聚类  **int** numIterations=20; //迭代次数  KMeansModel clusters=KMeans.*train*(parseData.rdd(), numClusters, numIterations);  System.***out***.println("聚类中心:");  **for**(Vector center:clusters.clusterCenters())  {  System.***out***.println(" "+center);  }  **double** cost=clusters.computeCost(parseData.rdd());  System.***out***.println("cost:"+cost);    }  } |

# 2. Gaussian mixture

高斯混合模型表示复合分布，其中从k个高斯子分布之一绘制点，每个子分布具有其自己的概率。spark.mllib实现使用期望最大化算法在给定一组样本的情况下推导最大似然模型。该实现具有以下参数：

* **k**是所需簇的数量。
* **convergenceTol**是我们考虑收敛的对数似然的最大变化。
* **maxIterations**是在不达到收敛的情况下执行的最大迭代次数。
* **initialModel**是启动EM算法的可选起点。如果省略该参数，则将从数据构造随机起始点。

|  |
| --- |
| **package** com.cb.spark.mllib;  **import** org.apache.spark.SparkConf;  **import** org.apache.spark.SparkContext;  **import** org.apache.spark.api.java.JavaRDD;  **import** org.apache.spark.mllib.clustering.GaussianMixture;  **import** org.apache.spark.mllib.clustering.GaussianMixtureModel;  **import** org.apache.spark.mllib.linalg.Vector;  **import** org.apache.spark.mllib.linalg.Vectors;  **public** **class** JavaGaussianMixtureExample {  **public** **static** **void** main(String[] args) {  SparkConf conf = **new** SparkConf().setAppName("JavaKMeansExample").setMaster("local");  SparkContext sc = **new** SparkContext(conf);  String path = "F:\\Learning\\java\\project\\LearningSpark\\src\\main\\resources\\gmm\_data.txt";  JavaRDD<String> data = sc.textFile(path, 1).toJavaRDD();  JavaRDD<Vector> parsedData = data.map(s -> {  String[] sarray = s.trim().split(" ");  **double**[] values = **new** **double**[sarray.length];  **for** (**int** i = 0; i < sarray.length; i++) {  values[i] = Double.*parseDouble*(sarray[i]);  }  **return** Vectors.*dense*(values);  });  parsedData.cache();  GaussianMixtureModel gmm = **new** GaussianMixture().setK(2).run(parsedData.rdd());  // 输出高斯混合模型的参数  **for** (**int** j = 0; j < gmm.k(); j++) {  System.***out***.printf("weight=%f\nmu=%s\nsigma=\n%s\n", gmm.weights()[j], gmm.gaussians()[j].mu(),  gmm.gaussians()[j].sigma());  }  }  } |