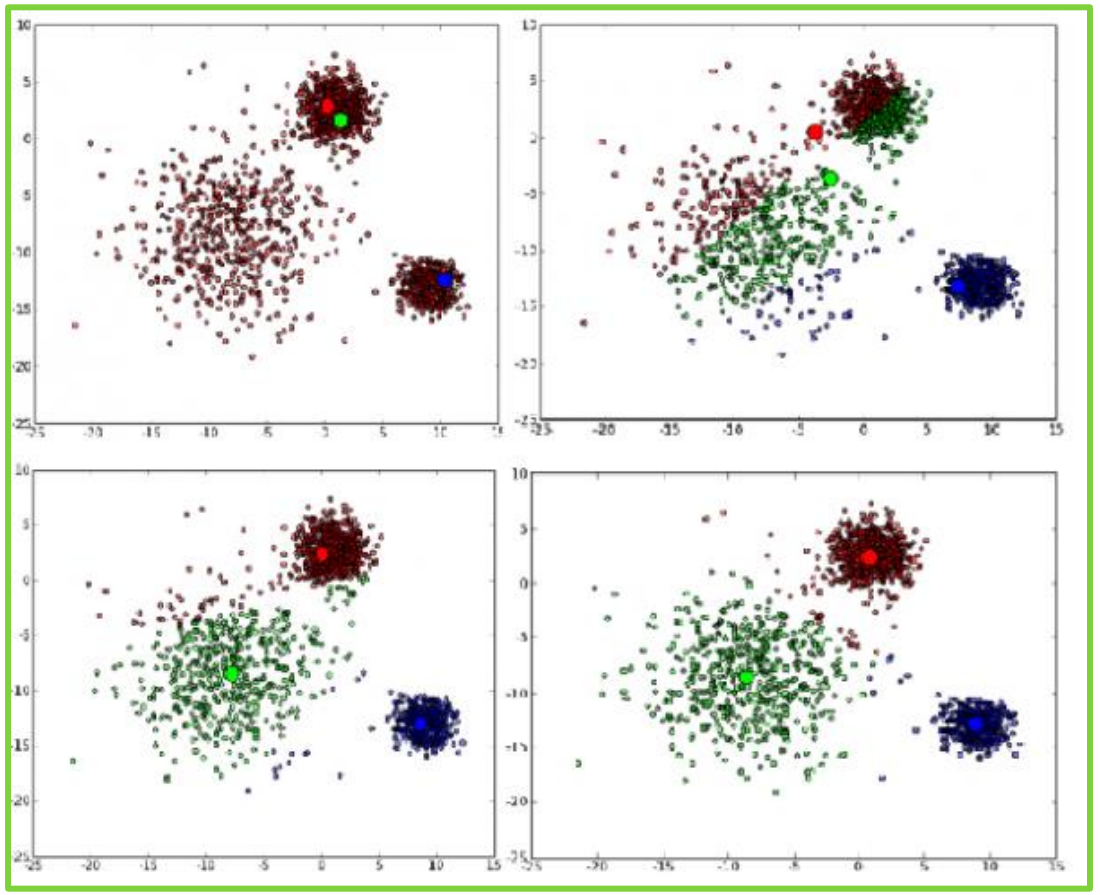


第七课 Spark ML KMeans聚类算法

目录

- KMeans聚类算法
- Kmeans 源码
- ML KMeans模型参数详解
- ML实例

目录

KMeans

KMeans 算法

- KMeans 算法的基本思想是初始随机给定K 个簇中心，按照最邻近原则把待分类样本点分到各个簇。然后按平均法重新计算各个簇的质心，从而确定新的簇心。一直迭代，直到簇心的移动距离小于某个给定的值。
- KMeans 聚类算法主要分为3 个步骤。

1）第1 步是为待聚类的点寻找聚类中心；

2）第2 步是计算每个点到聚类中心的距离，将每个点聚类到离该点最近的聚类中去；

3）第3 步是计算每个聚类中所有点的坐标平均值，并将这个平均值作为新的聚类中心。

反复执行2）、3），直到聚类中心不再进行大范围移动或者聚类次数达到要求为止。

KMeans 算法

- 展示了对n个样本点进行KMeans 聚类的效果，这里k取2。

（a）未聚类的初始点集；

（b）随机选取两个点作为聚类中心；

（c）计算每个点到聚类中心的距离，

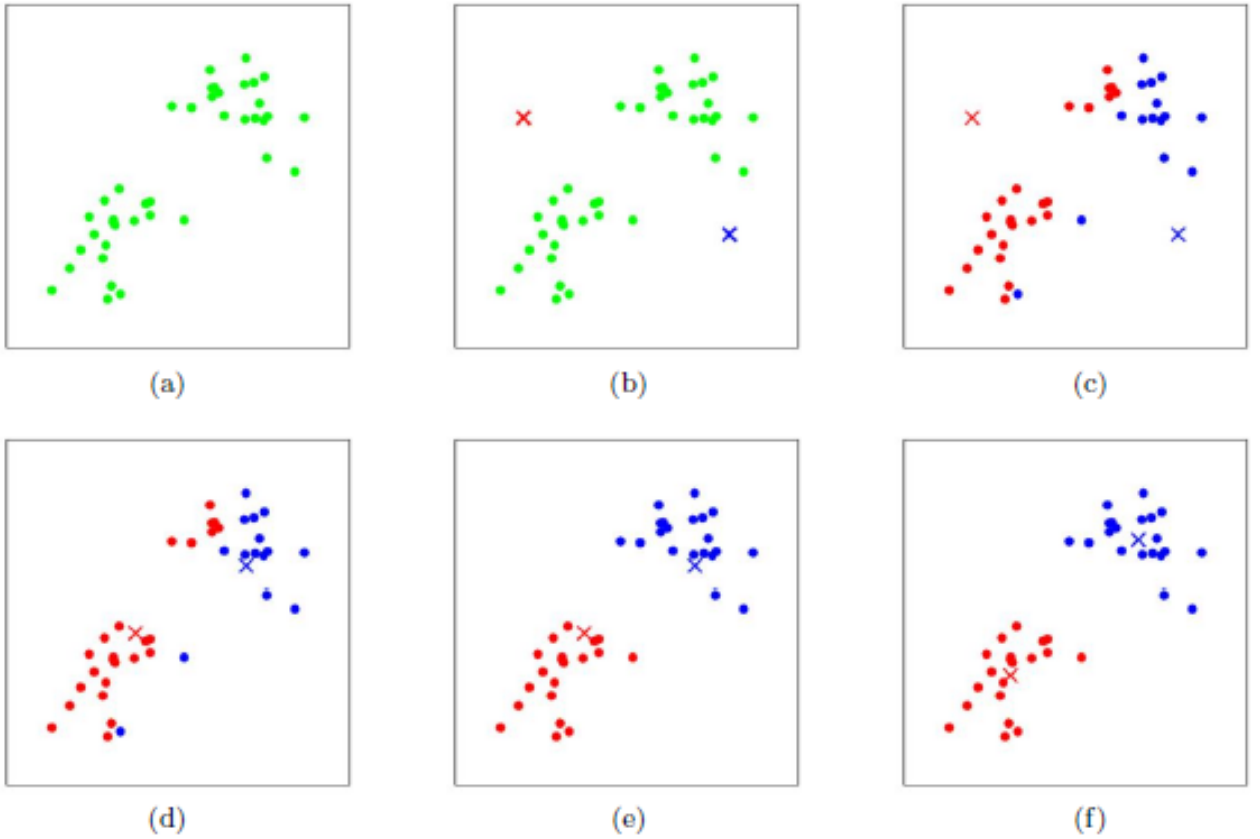
并聚类到离该点最近的聚类中去；

（d）计算每个聚类中所有点的坐标平均值，

并将这个平均值作为新的聚类中心；

（e）重复（c），计算每个点到聚类中心的距离，

并聚类到离该点最近的聚类中去；



（f）重复（d），计算每个聚类中所有点的坐标平均值，并将这个平均值作为新的聚类中心。

初始化聚类中心点

- k-means++算法选择初始中心点的基本思想就是：初始的聚类中心之间的相互距离要尽可能远。

  初始化过程如下。

  1）从输入的数据点集合中随机选择一个点作为第一个聚类中心；

  2）对于数据集中的每一个点x，计算它与最近聚类中心（指已选择的聚类中心）的距离D(x)；

  3）选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大；

  4）重复2）和3），直到k 个聚类中心被选出来；

  5）利用这k 个初始的聚类中心来运行标准的KMeans 算法。

初始化聚类中心点

- 从上面的算法描述可以看到，算法的关键是第3 步，如何将$D(x)$反映到点被选择的概率上。一种算法如下。

  1）随机从点集D 中选择一个点作为初始的中心点。

  2）计算每一个点到最近中心点的距离Si ，对所有Si 求和得到sum。

  3）然后再取一个随机值，用权重的方式计算下一个“种子点”。取随机值random（0<random<sum），对点集D 循环，做random - = Si 运算，直到random < 0，那么点i 就是下一个中心点。

  4）重复2）和3），直到k 个聚类中心被选出来。

  5）利用这k 个初始的聚类中心来运行标准的KMeans 算法。

源码分析

- MLlib 实现KMeans 聚类算法：首先随机生成聚类中心点，支持随机选择样本点当作初始中心点，还支持k-means++方法选择最优的聚类中心点。然后迭代计算样本的中心点，迭代计算中心点的分布式实现是：首先计算每个样本属于哪个中心点，之后采用聚合函数统计属于每个中心点的样本值之和以及样本数量，最后求得最新中心点，并且判断中心点是否发生改变。

- MLlib 的KMeans 聚类模型的runs 参数可以设置并行计算聚类中心的数量，runs 代表同时计算多组聚类中心点，最后取计算结果最好的那一组中心点作为聚类中心点。

源码分析

MLlib 的 KMeans 聚类模型对于计算样本属于哪个中心点，采用了一种快速查找、计算距离的方法，其方法如下。

首先定义 lowerBoundOfSqDist 距离公式，假设中心点 center 是 (a_1,b_1)，需要计算的点 point 是 (a_2,b_2)，那么 lowerBoundOfSqDist 是：

$$\begin{aligned}\text{lowerBoundOfSqDist} &= (\sqrt{a_1^2+b_1^2}-\sqrt{a_2^2+b_2^2})^2 \\ &= a_1^2+b_1^2+a_2^2+b_2^2-2\sqrt{(a_1^2+b_1^2)(a_2^2+b_2^2)}\end{aligned}$$

对比欧氏距离：

$$\begin{aligned}\text{EuclideanDist} &= (a_1-a_2)^2+(b_1-b_2)^2 \\ &= a_1^2+b_1^2+a_2^2+b_2^2-2(a_1a_2+b_1b_2)\end{aligned}$$

可轻易证明 lowerBoundOfSqDist 将会小于或等于 EuclideanDist，因此在进行距离比较的时候，先计算很容易计算的 lowerBoundOfSqDist（只需要计算 center、point 的 L2 范数）。如果 lowerBoundOfSqDist 都不小于之前计算得到的最小距离 bestDistance，那真正的欧氏距离也不可能小于 bestDistance 了。因此在这种情况下就不需要去计算欧氏距离了，省去了很多计算工作。

源码分析

如果 lowerBoundOfSqDist 小于 bestDistance，则进行距离的计算，调用 fastSquaredDistance，该方法是一种快速计算距离的方法。fastSquaredDistance 方法会先计算一个精度，有关精度的计算：precisionBound1 = 2.0 * EPSILON * sumSquaredNorm / (normDiff * normDiff + EPSILON)。如果精度满足条件，则欧氏距离为 EuclideanDist = sumSquaredNorm − 2.0 * v1.dot(*v2*)，其中 sumSquaredNorm 为 $a_1^2 + b_1^2 + a_2^2 + b_2^2$，2.0 * v1.dot(v2)为 $2(a_1a_2 + b_1b_2)$。这里可以直接利用之前计算的 L2 范数。如果精度不满足要求，则进行原始的距离计算，公式为 $(a_1 - a_2)^2 + (b_1 - b_2)^2$。

源码分析

KMeans源码分解说明		
1、KMeans聚类伴生对象	KMeans	KMeans对象
1.1train静态方法	train	train是KMeans对象的静态方法，该方法是根据设置Kmeans聚类参数，新建Kmeans聚类，并执行run方法进行训练
2、KMeans聚类类	KMeans	KMeans类
2.1 run方法	run	run是KMeans类方法，该方法主要用runAlgorithm方法进行聚类中心点的计算。
3、聚类中心点计算	runAlgorithm	runAlgorithm方法
3.1 初始化中心	initRandom	初始化中心点方法支持支持随机选择中心点和k-means++方法生成中心点
3.2 迭代计算更新中心	iteration	迭代计算样本属于哪个计算中心点，并更新最新中心点
4、KMeans聚类模型	KMeansModel	KMeansModel类
4.1 预测计算	predict	预测样本属于哪个类

目录

ML 参数讲解

ML 参数

Parameter setters

def **setFeaturesCol**(value: String): KMeans.this.type

Annotations @Since("1.5.0")

def **setK**(value: Int): KMeans.this.type

Annotations @Since("1.5.0")

def **setMaxIter**(value: Int): KMeans.this.type

Annotations @Since("1.5.0")

def **setPredictionCol**(value: String): KMeans.this.type

Annotations @Since("1.5.0")

def **setSeed**(value: Long): KMeans.this.type

Annotations @Since("1.5.0")

def **setTol**(value: Double): KMeans.this.type

Annotations @Since("1.5.0")

ML 参数

(expert-only) Parameters

A list of advanced, expert-only (hyper-)parameter keys this algorithm can take. Users can set and get the parameter values through setters and getters, respectively.

final val **initMode**: Param[String]

Param for the initialization algorithm. This can be either "random" to choose random points as initial cluster centers, or "k-means||" to use a parallel variant of k-means
Default: k-means||.

Definition Classes KMeansParams
Annotations @Since("1.5.0")

final val **initSteps**: IntParam

Param for the number of steps for the k-means|| initialization mode. This is an advanced setting -- the default of 2 is almost always enough. Must be > 0. Default: 2.

Definition Classes KMeansParams
Annotations @Since("1.5.0")

目录

ML 实例

实例代码

```
import org.apache.spark.ml.clustering.{KMeans,KMeansModel}
import org.apache.spark.sql.SparkSession

import org.apache.spark.ml.clustering.{KMeans, KMeansModel}
import org.apache.spark.sql.SparkSession
```

实例代码

```
    // 读取样本
    val dataset = spark.read.format("libsvm").load("hdfs://[redacted]/sample_kmeans_data.txt")
    dataset.show(false)
```

```
dataset: org.apache.spark.sql.DataFrame = [label: double, features: vector]
+-----+-------------------------+
|label|features                 |
+-----+-------------------------+
|0.0  |(3,[],[])                |
|1.0  |(3,[0,1,2],[0.1,0.1,0.1])|
|2.0  |(3,[0,1,2],[0.2,0.2,0.2])|
|3.0  |(3,[0,1,2],[9.0,9.0,9.0])|
|4.0  |(3,[0,1,2],[9.1,9.1,9.1])|
|5.0  |(3,[0,1,2],[9.2,9.2,9.2])|
+-----+-------------------------+
```

实例代码

```
// 训练 a k-means model.
val kmeans = new KMeans().setK(2).setSeed(1L)
val model = kmeans.fit(dataset)
```

```
kmeans: org.apache.spark.ml.clustering.KMeans = kmeans_fcb5d8aac134
model: org.apache.spark.ml.clustering.KMeansModel = kmeans_fcb5d8aac134
```

```
// 模型指标计算.
val WSSSE = model.computeCost(dataset)
println(s"Within Set Sum of Squared Errors = $WSSSE")
```

```
WSSSE: Double = 0.11999999999994547
Within Set Sum of Squared Errors = 0.11999999999994547
```

实例代码

```
// 结果显示.
println("Cluster Centers: ")
model.clusterCenters.foreach(println)
```

```
Cluster Centers:
[0.1,0.1,0.1]
[9.1,9.1,9.1]
```

```
// 模型保存与加载
 model.save("hdfs:[redacted]/clustering/kmmodel")
 val load_treeModel = KMeansModel.load("hdfs://[redacted]/clustering/kmmodel")
```

```
load_treeModel: org.apache.spark.ml.clustering.KMeansModel = kmeans_fcb5d8aac134
```

Thanks

FAQ时间