Spark mllib 分析报告-Kmeans

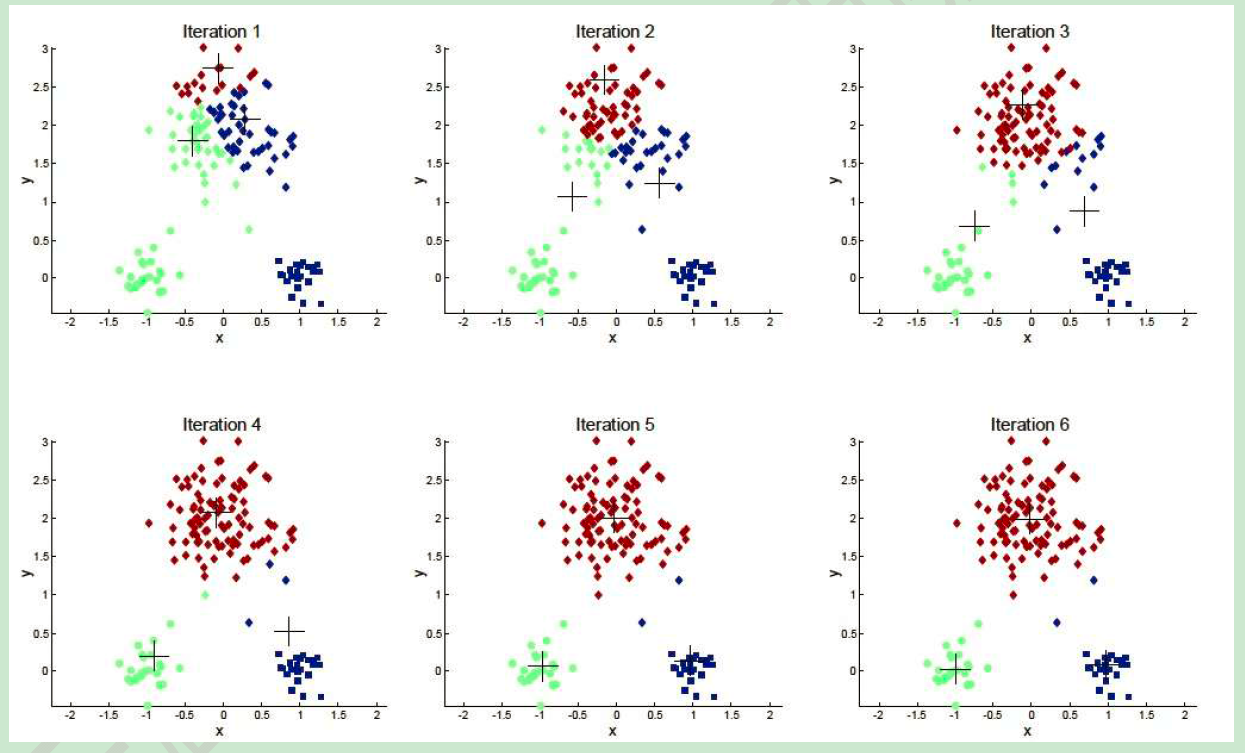
**一：算法介绍**

聚类分析(Cluster Analysis)试图将属性相似的实例划分到同一个类簇，“相似”

体现在实例在高维特征空间上具有更小的距离。图1 展示了示例性的聚类过程，

图中黑色十字表示类簇中心点，随着一步一步的迭代计算，类簇中心点不断被调

整，最终所有数据点被清晰划分为3 个类簇，每个类簇中的点与该类簇中心点之间的欧拉距离最小。Kmeans 是一种常见的聚类算法，是一种无监督学习方法。

 图1. 二维空间下聚类示例

大数据聚类是从大数据中发现价值必须面对的一个基础性问题，“物以类聚、

人以群分”也是人类基本认知能力在数据科学中的体现。大数据聚类能有效支撑

如客户群细分、文本主题发现、信息检索等大量实际应用。传统聚类方法的重要

假设是数据能够一次性载入内存，随着数据量的急剧增大，单机处理已经力不从

心，需要人们利用分布式计算系统进行并行处理。同时，由于大部分聚类算法都

是迭代型算法，下一轮计算依赖于上一轮的计算结果（如：K-means 新一轮距离

计算依赖于上一轮计算出的中心点），因此，Spark 的内存计算方式更适用于分布

**二：算法过程**

Kmeans 是一种迭代式算法。

算法初始化：

1>:可以随机初始化

2>:采用Kmeans|| , 是Kmeans++ 的变种

Kmeans|| 和Kmenas++具体见附录

算法过程：

主要计算过程是迭代以下两步：

1：计算每个样本点所属的中心点，得到K 个簇。

2：计算每个簇中的点的均值，得到新的中心点。

**三：算法评价**

评价标准：

1：收敛速度

单机实现：O(N\*K\*M\*D) 其中N为样本点个数，K为中心点个数，M为算法迭代次数，D是数据维数

并行化实现：比如Spark, O(N/P \* K \*M\*D) 其中P 为 集群中可以并行执行的Partition个数

2：正确性：SSE(sum of square error)

SSE:标准误差定义为各测量值误差的平方和的平均值的平方根。

设n个测量值的误差为ε1、ε2……εn，则这组测量值的标准误差σ等于：

[IMG_256](http://baike.baidu.com/pic/åæ¹è¯¯å·®/9024810/0/4651a7125b41e2d0c2fd7878?fr=lemma&ct=single)

目前Mllib实现了SSE评价标准。

**四：算法并行化**

**1>算法过程：**

Kmeans通过借助于RDD,从而实现了核心计算过程的并行化。

1:初始化：

初始化中心点

2:迭代直至收敛或者精度达到SSE指定的精度：

2.1：broadcast中心点

2.2：计算sum and count of each points mapping to each center

rdd.mapPartition().reduceByKey(mergeFunc).collectAsMap()

2.3：更新中心点：这一步在driver进行

KMeans应该没有driver瓶颈，因为每个executor 只返回K个（sum,count）这样的数据给driver。

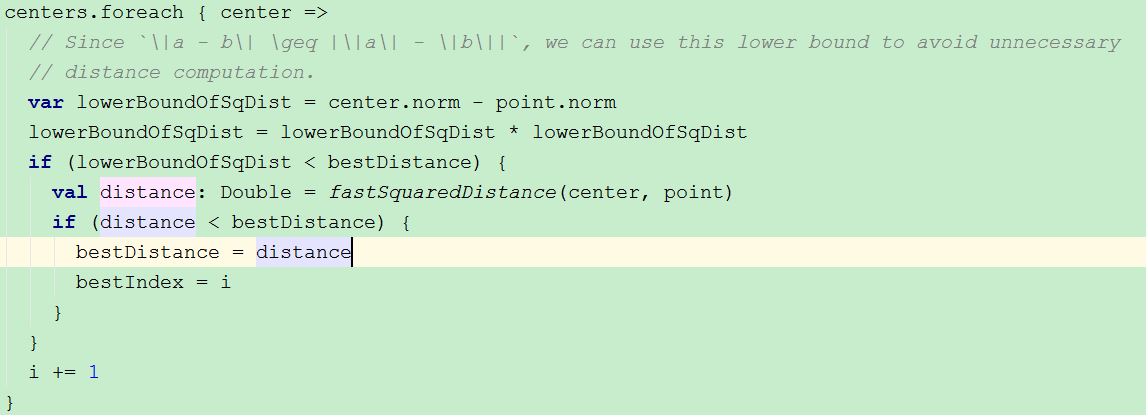
**2>Cache**

初始RDD: 应该被Cached,因为KMeans 是一个迭代式算法。

KMeans算法不生成中间RDD.

**3>算法中值得借鉴的地方：**

**通过三角不等式和L2范数优化欧式距离的计算,比如：计算每个点到中心点的距离：**



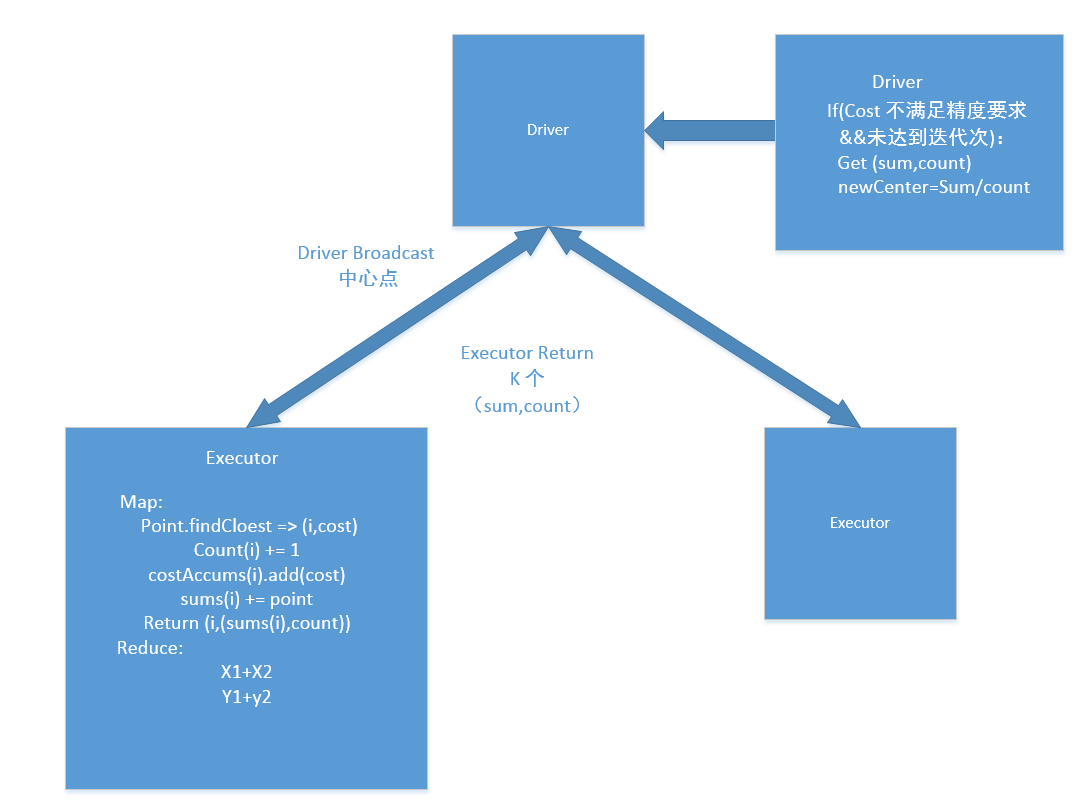
**利用了三角不等式 |a - b| >= |a| -|b|**

**可以通过L2 范式快速计算出 |a| -|b|**

**另外fastSquareDistance 还可以根据范式快速的计算距离**

**五：数据流图**

**算法过程流图如下：**



**六：测试**

|  |  |  |  |
| --- | --- | --- | --- |
| **DataSet Size** | **Configuration** | **Run time** | **Error** |
| 58M  10Million points  2 dims/point | 5Executor（10Cores）  4G/Executor | 1.3Mins | No |
| 290M  50Million points  2 dims/point | 5Executor(10Cores)  4G/Executor | 5.6Mins | No |
| 580M  100Million points  2 dims/point | 5 Executor(10cores)  4G/Executor | 40Mins | No |
| 2.9G  500Million points  2 dims/point | 8 Executor(16cores)  4G/Executor  Standalone mode | failed | JavaHeapError 非常多  作业失败率高 |
| 2.9G  500Million points  2 dims/point | 8 Executor(16cores)  12G/Executor  Standalone mode | Failed(多次尝试) | JavaHeapError 减少了，但是出现了大量的  Failed to fetch block after 1 fetch failures. Most recent failure cause: |
| 2.9G  500Million points  2 dims/point | 8 Executor(16cores)  12G/Executor  Yarn mode |  | RPC Failed |

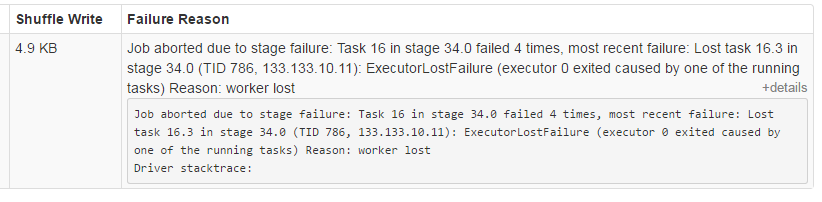
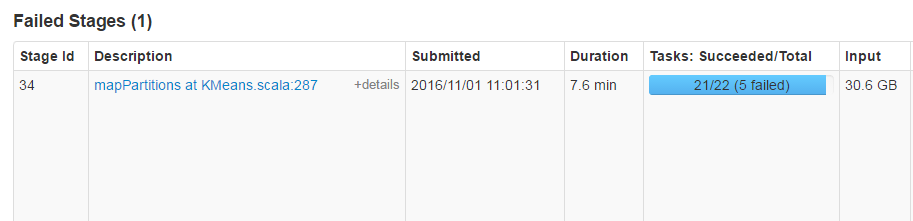
**存在的问题：**

1. **当数据集比较大的时候，中间数据集太大，容易导致作业失败**

**比如：2.9G 的 原始输入数据，往往在算法执行几个小时之后崩溃。监控页面却又显示作业SUCCESSED的信息。**

**Failure:**

**1. Standalone mode**



**算法可扩展性:**

**可以看到当数据量增加10倍的时候，计算时间增长是多于10倍的。**

**七：Others**

**Spark 还实现了 Bisecting KMeans和 StreamingKMeans**

**TODO：算法优化：**

**1：参考目前Kmeans 最好单机版实现：**

**<http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview>**

**高维数据稀疏性**

**2：Community**

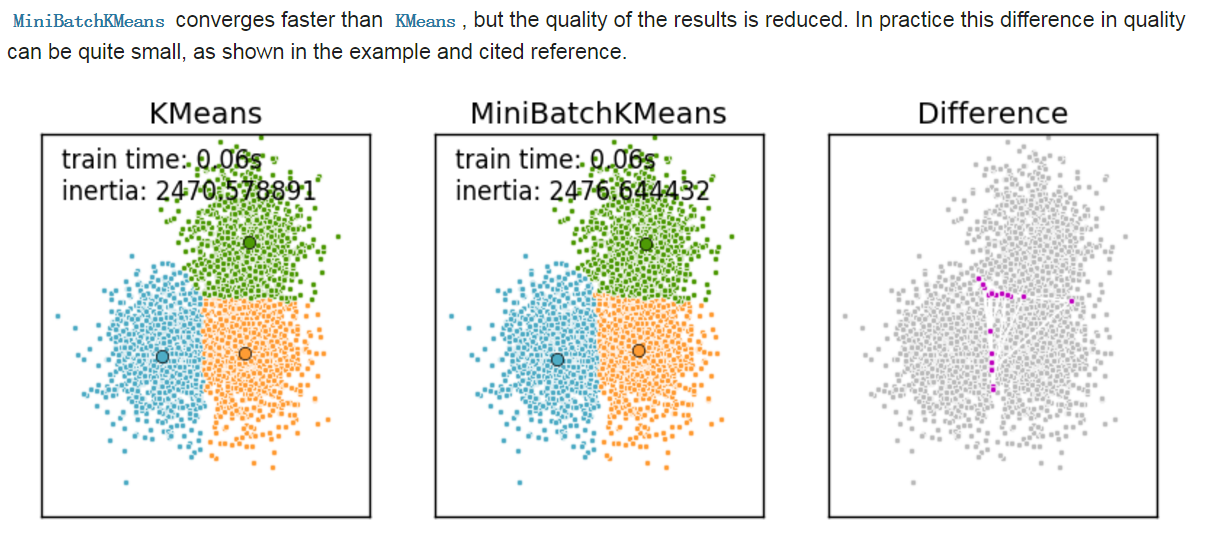
**SPARK-14174 Accelerate Kmeans wia Mini-Batch EM**

**思路：抽样**

**加速距离计算**

**保存状态，避免重复计算**

**关于Mini-Batch Kmeans,可以减少算法时间，准确性只比标准Kmeans 差一点，见下图。**



**附录：**

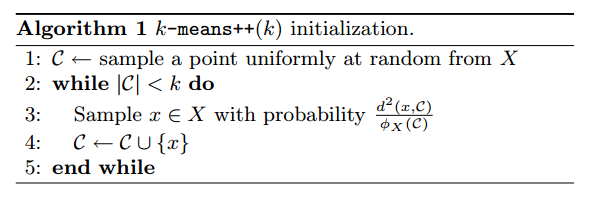
## **2 k-means++算法原理分析**

  k-means++算法选择初始聚类中心的基本原则是：初始的聚类中心之间的相互距离要尽可能的远。它选择初始聚类中心的步骤是：

（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[IMG_256](https://camo.githubusercontent.com/d3b70d64dd66dc12d8337cea2df1eb8108f3cb52/687474703a2f2f7777772e666f726b6f73682e636f6d2f6d6174687465782e6367693f7b637d5f7b317d)；

（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)，并根据概率选择新的聚类中心[IMG_257](https://camo.githubusercontent.com/4a6af13d055696286d358510632c6f6a8c760c34/687474703a2f2f7777772e666f726b6f73682e636f6d2f6d6174687465782e6367693f7b637d5f7b697d)。

（3）重复过程（2）直到找到k个聚类中心。



  第(2)步中，依次计算每个数据点与最近的种子点（聚类中心）的距离，依次得到D(1)、D(2)、...、D(n)构成的集合D，其中n表示数据集的大小。在D中，为了避免噪声，不能直接选取值最大的元素，应该选择值较大的元素，然后将其对应的数据点作为种子点。 如何选择值较大的元素呢，下面是spark中实现的思路。

求所有的距离和Sum(D(x))

取一个随机值，用权重的方式来取计算下一个“种子点”。这个算法的实现是，先用Sum(D(x))乘以随机值Random得到值r，然后用currSum += D(x)，直到其currSum > r，此时的点就是下一个“种子点”。

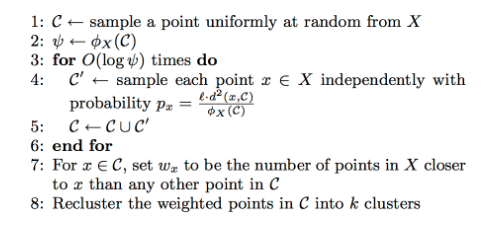
  为什么用这样的方式呢？我们换一种比较好理解的方式来说明。把集合D中的每个元素D(x)想象为一根线L(x)，线的长度就是元素的值。将这些线依次按照L(1)、L(2)、...、L(n)的顺序连接起来，组成长线L。L(1)、L(2)、…、L(n)称为L的子线。 根据概率的相关知识，如果我们在L上随机选择一个点，那么这个点所在的子线很有可能是比较长的子线，而这个子线对应的数据点就可以作为种子点。

### **2.1 k-means++算法的缺点**

  虽然k-means++算法可以确定地初始化聚类中心，但是显然它存在一个缺点，那就是它内在的有序性特性：下一个中心点的选择依赖于已经选择的中心点。 针对这种缺陷，k-means||算法提供了解决方法。

## **3 k-means||算法原理分析**

  k-means||算法是在k-means++算法的基础上做的改进，和k-means++算法不同的是，它采用了一个采样因子l，并且l=O(k)，在spark的实现中l=2k，。这个算法首先如k-means++算法一样，随机选择一个初始中心， 然后计算选定初始中心确定之后的初始花费phi(指与最近中心点的距离)。之后处理log(phi)次迭代，在每次迭代中，给定当前中心集，通过概率[IMG_258](https://camo.githubusercontent.com/bddd4fd1c31ee4110949de14d0e59c4589bb5a8b/687474703a2f2f7777772e666f726b6f73682e636f6d2f6d6174687465782e6367693f6c7b647d5e7b327d28782c43292f7b7068697d5f7b587d284329)来抽样x，将选定的x添加到初始化中心集中，并且更新[IMG_259](https://camo.githubusercontent.com/eff1041e609750b6d8c88d3fbc2da063b9bbac27/687474703a2f2f7777772e666f726b6f73682e636f6d2f6d6174687465782e6367693f7b7068697d5f7b587d284329)。该算法的步骤如下图所示：



  第1步随机初始化一个中心点，第2-6步计算出满足概率条件的多个候选中心点C，候选中心点的个数可能大于k个，所以通过第7-8步来处理。第7步给C中所有点赋予一个权重值[IMG_261](https://camo.githubusercontent.com/558bc93fcc0b4505bfe0251413793b7259ffd3ba/687474703a2f2f7777772e666f726b6f73682e636f6d2f6d6174687465782e6367693f7b777d5f7b787d)，这个权重值表示距离x点最近的点的个数。 第8步使用本地k-means++算法聚类出这些候选点的k个聚类中心。在spark的源码中，迭代次数是人为设定的，默认是5。

  该算法与k-means++算法不同的地方是它每次迭代都会抽样出多个中心点而不是一个中心点，且每次迭代不互相依赖，这样我们可以并行的处理这个迭代过程。由于该过程产生出来的中心点的数量远远小于输入数据点的数量， 所以第8步可以通过本地k-means++算法很快的找出k个初始化中心点。何为本地k-means++算法？就是运行在单个机器节点上的k-means++。

My Opinion:

KMeans|| 相对于Kmeans++ 就是抽取Log(InitCost) \* K 个 中心点，然后运行一次Local Kmeans, 即Kmeans++ & Lloyd’s Algorithm。

Lloyd’s Algorithm is a standard implement of Kmeans

My try:

