1.引言

聚类已经被广泛的应用，如视频和图像的处理，文本分析，影响因子，无线传感器网络，网页分析以及文档的聚类。

为了定义聚类，我们令I = {x1,x2,……，xn}表示n个元素的集合，并且每个元素包含m个属性：xi={xi1,xi2,xi,……,xim}。聚类的目的就是把集合I分为k个子集，并且k个子集两两之间没有交集，k个子集的并集为集合I。每个子集中的元素彼此之间是最相似的。

有许多聚类的算法已经被提出，如：DBSCAN, CURE, Chameleon, k-MS等。K-means算法在小数据集上的效果比较好。它将最小化每个子集中元素间的欧式距离的和，并取所有子集的平均值作为目标函数（SSEDM）。

输入一个集合I，SSEDM的输出为S，S = S1 ∪ S2 ∪ …… ∪Sk, 且Si ∩ Sj = ∅（i≠j），SSEDM的解S可以表述为：

SSEDM(S) = SSEDM(I,S,K) = (1)

在这篇论文中，SSEDM()被称为集合的偏微商(partial)， 可以通过下面的公式计算：

(2)

等式（2）中，dis(,)表示两个点之间的欧式距离，表示集合的中心点。

最小化SSEDM，既使元素是二维的（每个元素的属性个数为2），它也是一个NP难的优化问题。但是对于线性可分的的集合，k-means可以最小化SSEDM。 最小化包含k个集合的SSEDM，k-means算法有如下的方式：（1）随机选择k个点初始化聚类的中心。（2）将集合中的每个元素加入到距离其最近的中心集合中（3）更新每个簇的中心。（4）重复（2）（3）步直到中心收敛。尽管k-means算法在包含多个SSEDM的情况下非常有效，但是它对初始化中心点非常敏感。我们可以认为中心点如果初始化的好可以加快算法的收敛速度。因此，许多初始化k-means中心点的算法被提出。一些关于这些算法的评价在【24】中有所介绍。尽管使用好的初始化算法可以增加k-means的准确性，然而有时候使用这些算法k-means的准确性并不令人满意。尤其是在当k值特别大的时候。图1（B）展示了图1（A）人工数据集通过k-means++算法的一个聚类情况。我们可以看到，即使使用了k-means++这个比较好的初始化中心点算法，SSEDM的值在图1（B）的值也明显大于在图1（C）中的值。

图1的例子表明，即使使用了好的初始化中心点的方法，最后的结果仍然可能不是最优解，还需要进一步的优化。图2展示了k-means++产生的解如何进一步的优化。图2（A）中每一个椭圆形的集合可以被分成两个集合。每一个矩形集合包含了多个簇，可以移除一个或多个簇，其中的数据再重新分到其他合适的簇。在图2（A）中，我们可以通过移除以为中心的簇并分割以为中心的簇更新聚类结果。我们称这个过程为minus-plus阶段，在这个阶段一个已有的簇被移除，同时一个已有的簇被分割。在图2中，首先以为中心的簇被分割，并且以为中心的的簇被移除，然后minus-plus阶段将中心变为以为中心的簇中的一点，重新执行k-means算法。

本文提出了一个改进k-means结果的迭代方法。这个方法通过迭代这行多次minus-plus阶段，因此把它叫做I-k-means-+（iteratively k-means minus plus）。每次迭代，I-k-means-+使用一些启发式快速的找到一对合适的簇通过minus-plus阶段进行处理。实验的结果表明，通常情况下，就最小化SSEDM来说，I-k-means-+的准确度要高于其他的算法。此外，使用本文提出的启发式，I-k-means-+算法的速度与k-means相比是可以接受的，而且在一些情况下它的速度还要比k-means快。除了这些优点，这个方法可能也可以应用于目标函数类似的算法，比如fuzzy c-means（FCM），这留作进一步的工作。

因此，下一节我们先回顾k-means算法，第三节介绍本文提出的算法，第四节展示了算法的实验结果，最后一节是对本文的总结。

2.k-means目前的研究

因为k-means对初始化的簇中心非常敏感，许多研究尝试提出一个有效的方法去初始化k-means方法的簇中心，本节中的一部分就专门来回顾这些算法。另外，k-means算法速度比较慢，由于它在每一次迭代，每一个数据点都要计算与所有中心的距离并找到最近的一个簇中心加入该簇，这个步骤需要大量的计算。所以，本节第二个部分就回顾一些重要的加速k-means的算法。

**2.1初始化k-means簇中心的算法**

Maxmin算法随机选择一个数据点作为第一个簇中心，然后选择第i个中心时，找与之前选择的簇中心距离最大的数据点作为第i个中心。

论文【23】使用kd-trees将按空格分割的数据集放入一些桶里，然后确定每一个桶的密度。每一个桶用该桶中所有数据点的均值来表示。密度最大的桶被选为第一个簇中心。一个桶的均值点为能使密度增值最大，这个点通过计算计算与其最近簇中心的距离得到。则这个数据点被选为第i个中心。

k-means++算法随机在数据集中选一个数据点作为第一个中心，接下来，选择数据点xX作为第i个中心的可能性为，D(x)表示数据点到最近的簇中心的距离。

PCA-Part算法最开始初始化一个簇包含所有的数据点，然后执行k – 1步，每一步选择SSEDM值最大的一个簇，将这个簇分为2个子簇。分割这个簇时，使用一个超平面穿过簇的中心，这个超平面与协方差矩阵的主要的特征向量正交。

Global k-means算法

# 有待翻译

**2.2提升k-means速度的算法**

3.I-k-means-+

I-k-means-+通过使用【41】中初始化中心的k-means算法产生了一个聚类结果，然后通过迭代去优化这个结果，最小化SSEDM。每次迭代，移除一个簇中心（minus），分割另一个簇（plus），并且使用k-means的提升版本重新聚类。图3展示了I-k-means-+算法的流程，图4展示了minus-plus阶段（这个阶段移除一个簇中心，添加另一个簇中心并且重新聚类）如何优化最初的结果。S中的被分为两个簇，这个簇被移除，然后重新聚类后的结果为S’。minus-plus可以描述为，将簇中心变为簇中的任意一点，然后重新执行k-means聚类。但是，这里有两个主要的难点：

1. **如何找到合适的一对像，的簇。认为所有的组合都是不合理的，这就需要花费大量的时间。**
2. **如何加速k-means重新聚类的时间，因为只有一部分数据需要重新聚类。**

本节剩余的部分将描述如何解决这些难点，使用一些策略去寻找合适的一对簇，并使用k-means改进加速重新聚类的过程。最后一小部分解释了I-k-means-+的一些细节。

**3.1 寻找合适的一对簇**

合适的一对簇应该在minus-plus阶段前被找到，因为minus-plus阶段多次的试错方法表明，尤其是在重新聚类的过程，将要花费大量的时间。因此，在分割移除之前，应该确定或者估计出移除了后SSEDM增加了多少，这个值为移除的cost。分割后SSEDM能够取到多少，这个值被称为分割的gain。

**定义1.** 对于一个聚类结果S，移除的花费记为Cost（），表示从S中移除后SSEDM（S）的增量。

Cost（）最终的值是SSEDM（S）的增加值减去SSEDM（S）的取值。SSEDM（s）的取值就是在移除之前的SSEDM（）的值。我们可以通过计算中每个数据点到第二近的簇中心的距离和确定SSEDM(S)的增加值。因此Cost（）可以表示为下面的等式：

Cost（）= SSEDM() - (3)

为第二近的簇p的中心。应该注意的是第二个近的中心在k-means分配样本点属于哪个中心的阶段就可以确定，不用花费额外的时间。

**定义2.**对于一个聚类结果S，分割的收益记为Gain()，表示分割S中为两个簇后SSEDM（S）的值减少了多少。

当一个簇像一样被分割为两个簇和，当前SSEMD()的值被SSEDM(S)减去，相对的，SSEDM()，SSEDM()的和被加到SSEDM(S)中。因此，Gain()可以表示为如下的等式：

Gain() = SSEDM() – (SSEDM() + SSEDM()) (4)

对于这个等式，SSEDM()是已知的，SSEDM()和SSEDM()是未知的，我们需要估计它们的值。我们记中的数据点到中心的平均距离为：

X = (5)

为将要被分割的簇的中心，为中数据点的个数。

现在，对于分割出的两个簇和，我们估计它们包含的数据点到对应簇中心的距离为X/2，每个簇包含点的个数为/ 2, 所以我们有：

SSEDM() ≈ SSEDM() ≈ × (6)

如果我们使用下面的估计：

SSEDM() ≈ (7)

通过（4）、（6）和（7），我们有：

Gain() ≈ × – 2 × ×

= ≈ SSEDM() (8)

接下来，令α = ，我们估计分割的减少值为：

Gain() ≈ αSSEDM() (9)

**启发式 1.** 一对簇和可以执行minus-plus阶段，需要被分割，需要被移除，如果Gain() > Cost()。

Gain和Costs的值暂且不论， 这里有一个概念在选择合适的一对簇时要使用。图4中，通过minus-plus阶段的处理，我们可以观察到SSEDM() < SSEDM(S)，且分割是不合理的，因为是在接下来迭代过程中要移除集合的近邻集合。另外，同时移除和也是不合适的，它们已经有一个待分割集合。在通过启发式描述这些规则前，我们需要两个定义。

**定义3** **.** 以为中心的簇是以为中心的簇的**近邻**，当且仅当，中有一点P（距它最近的簇中心为）第二近的簇中心为。

当是的近邻，并不意味着是的近邻。

**定义4.** 如果是的**强近邻，**当且仅当是的近邻，同时是的近邻。

例如，在图4（A）中是的强近邻。是的近邻，但不是的近邻。

**启发式 2.** 如果一个簇是由另外一个簇在当前的minus-plus阶段分割出来的，它和它的强近邻在下次迭代时不能被移除。

**启发式 3.** 如果一个簇在当前的minus-plus阶段被移除，那么它的强近邻在下次迭代时不能被分割。

**3.2 topical-k-means（t-k-means）**

t-k-means算法是k-means的改进版，它只作用于那些受到影响的点。在I-k-means-+算法中的minus-plus阶段，只有一部分数据点在上一次迭代的过程中受到了影响。因此对所有的数据点重新聚类显然是没有必要的，t-k-means算法就很好的解决这一问题。如图5所示，在I-k-means算法的minus-plus阶段，中心为的簇将要被移除，中心为的簇将要被分割为两个簇，所以中心这个数据点就变为中心为簇中的随机一点。t-k-means在接下来重新聚类的阶段，只关心那些受到影响的点，从而加快聚类的速度。下面的定义描述了哪些簇和点受到影响的以及关于t-k-means更多的细节。

**定义5.**在一个聚类结果S下，如果一个数据点受簇中心的影响，当且仅当是距它最近或第二近的簇中心。

下面介绍了在当前聚类结果S下，当仅有簇的中心，变为中心为的簇中的随机一点的情况下，t-k-means的工作过程。

**Step#1.**

1．初始化AC（active centers）为一个空的集合

2．AC 🡨 AC ∪ {} ∪ {}

3．AC-Adjacent 🡨 中心为的簇的近邻

4．AP 🡨 在中心改变之前受其影响的数据点。

**Step#2.**

1. 如果AC为空转到最后一步
2. AC-Adjacent 🡨 AC集合中簇的近邻
3. Potential-AC 🡨 {}
4. AP 🡨 受AC集合中簇中心影响的数据点

**Step#3.**

对每个受AC集合中簇中心影响的数据点P

1. 在集合AC∪AC-Adjacent中，更新距P最近和第二近的簇中心
2. 如果距P最近的簇中心从变为，则Potential-AC🡨 Potential-AC ∪ {} ∪{}

**Step#4.** 更新簇中心

**Step#5.**  AC 🡨 Potential-AC

**Step#6.**  回到Step#2

**Step#7.**  算法结束

如图5所示，t-k-means的过程如下。第一次迭代，在经过Step#1，Step#2后，AC = {，}，AC-Adjacent={, , }，AP中保存的的是受集合AC中簇中心影响的点以及在中心改变前受其影响的点。这些点来自于中心为,,,和的簇。在执行完Step#3后，不仅更新了距AP集合中点最近和第二近的簇中心，我们也得到了Potential-AC = {, , }。Step#6回到Step#2直到AC为空集。

**3.3I-k-means-+的细节介绍**

I-k-means遵循如下的流程：

**Instruction#1.**用【41】中的k-means算法产生一个聚类结果S，S包含多个簇（）

**Instruction#2.**定义一个变量#success 🡨 0

**Instruction#3.** 在这些簇中选择一个满足如下条件的具有最大Gain的簇，如果没有转到算法最后一步。

1．必须是没有被标记为不可分割的簇。

**Instruction#4.**如果这里有k/2个簇的gain值大于的gain值，则跳转到算法最后一步。在这种情况下，k/2个簇都有较大的SSEDM，又因为这些簇是不可分割的，因此分割是不合理的。

**Instruction#5.**在这些簇中选择满足如下条件的cost值最小的簇，如果没有转到算法最后一步。

1. () < Gain()
2. 和必须是没有被标记为不可匹配的
3. 不是的近邻，也不是的近邻
4. 必须是没有被标记为不可移除的簇

**Instruction#6.**

1. 如果有k/2个簇cost值小于，并且满足条件（1），（2），（3），（4）和（5），跳转到最后一步。
2. 如果有k/2个簇cost值小于，并且没有一个可以配对儿的簇，则将标记为不可分割的集合，跳转到**Instruction#3.**（重新寻找一个可分割的簇）

**Instruction#7.**

1. 保存当前的结果： 🡨 S.
2. 对于结果，改变的簇中心为簇中的随机一点
3. 对重新通过t-k-means聚类

**Instruction#8.**

如果SSEDM(I, k,) > SSEDM(I, k, S)，那么说明聚类的结果没有改善，则将和标记为不匹配的簇对儿。

否则，如果SSEDM(I, k,) < SSEDM(I, k, S)则：

1. 标记和为不可移除的簇
2. 标记与移除之前的强近邻的簇为不可分割的簇
3. S 🡨
4. 标记与当前和的强近邻的簇为不可移除的簇
5. #success 🡨 #success + 1

**Instruction#9.**

如果#success > k / 2，跳转到最后一步

**Instruction#10.** 算法结束