聚类算法

陈 鑫

2023 年 4 月 29 日

目录

目录

聚类分析（Cluster Analysis）是一组将研究对象分为相对同质的群组（Clusters）的统计分析技术。依据研究对象（样本或指标）的特征，对其进行分类以减少研究对象的数目。聚类分析是一种典型的无监督学习，用于对未知类别的对象进行划分，根据在数据中发现的描述对象及其关系的信息，将数据对象分组。组内的对象相互之间是相似的（相关的），而不同组中的对象是不同的（不相关的）。组内相似度越大，组间差距越大，说明聚类效果越好。也就是说，聚类的目标是得到较高的群组内相似度和较低的群组间相似度，使得群组间的距离尽可能大，群组内样本与群组中心的距离尽可能小。

聚类得到的群组可以用聚类中心、群组大小、群组密度和群组描述等来表示。聚类中心是一个群组中所有样本点的均值（质心），群组大小表示群组中所含样本的数量，群组密度表示群组中样本点的紧密程度，群组描述是群组中样本的业务特征。

常见的聚类有：基于划分的聚类，如 k-均值算法、k-medoids 算法、k-prototype 算法；基于层次的聚类，如合并聚类算法；基于密度的聚类，如 DBSCAN 算法、OPTICS 算法、DENCLUE 算法；基于网格的聚类以及基于模型的聚类，如模糊聚类、Kohonen 神经网络聚类。

目录

K 均值聚类是基于划分的聚类算法，计算样本点与群组质心的距离，与群组质心相近的样本点划分为同一群组。K 均值通过样本间的距离来衡量它们之间的相似度，两个样本距离越远，则相似度越低，否则相似度越高。

给定 m 个数据样本 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)} \in \mathbb{R}^n$。每个数据样本可看为 n 维空间中的一个点。假设需要将 m 个数据样本聚成 k 个类。K 均值算法的基本思想是：选取 $\mathbb{R}^n$ 中的 k 个点 $\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \cdots, \boldsymbol{c}^{(k)}$ 作为中心，并将每个数据样本分配至与其距离最近的中心，使得所有样本到分配到的中心的距离之和最小。这样，分配到同一中心的样本就聚成一类，采用这种方法，就可以将 m 个样本聚成 k 个类。

K 均值聚类算法：先随机选取 k 个点（不一定是样本点）作为初始中心（质心），并以它们为基础进行样本的初始分类，每一个质心为一个类；对每个样本点，计算它们到各个质心的距离，并将其归入到相互间距离最小的质心所在的群组。计算各个新群组的质心。在所有样本点都划分完毕后，根据划分情况重新计算各个群组的质心所在位置，然后迭代计算各个样本点到各个群组质心的距离，对所有样本点重新进行划分，重复此过程直到质心不再发生变化时或者到达最大迭代次数。

K 均值算法

Algorithm 1: K 均值算法

随机选择 $\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \cdots, \boldsymbol{c}^{(k)}$ 作为聚类的初始中心点

for $t = 1, 2, \cdots, N$ **do**

  设置质心集合 $C_1, C_2, \cdots, C_k = \emptyset$

  for $i = 1, 2, \cdots, m$ **do**

    $j^* = \arg\min_{1 \le j \le k} ||\boldsymbol{x}^{(i)} - \boldsymbol{c}^{(j)}||$

    $C_{j^*} \leftarrow C_{j^*} \cup \{\boldsymbol{x}^{(i)}\}$

  调整质心

  for $j = 1, 2, \cdots, k$: **do**

    $\boldsymbol{c}^{(j)} \leftarrow \frac{1}{|C_j|} \sum_{\boldsymbol{x} \in C_j} \boldsymbol{x}$

return $\boldsymbol{c}^{(1)}, \boldsymbol{c}^{(2)}, \cdots, \boldsymbol{c}^{(k)}$

在算法1中，每一轮循环，样本归类的计算时间为 m 个样本，每个样本有 n 个特征，要计算其到 k 个质心的距离并挑出最小的，因此时间复杂度为 $O(mnk)$，中心调整时间为 $O(mnk)$，所以 k 均值算法的整体时间复杂度为 $O(mnkN)$，其中 N 为迭代的轮次。

K 均值算法实现

K 均值算法实现参考 k_means.py
案例实战 9.1 墨渍数据的聚类。参考 blobs_4means.py

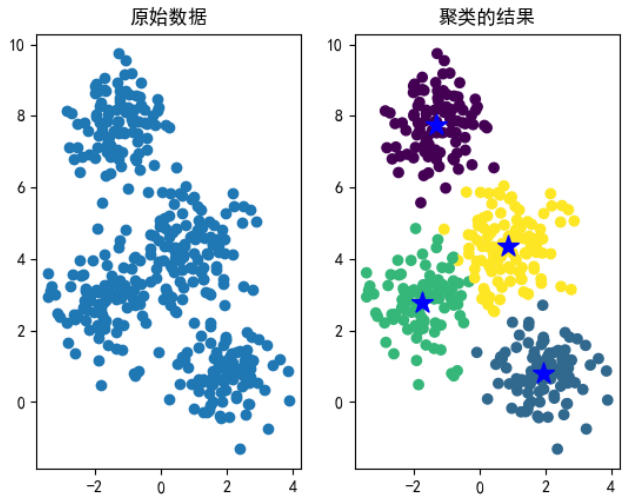


图 1: Kmeans 聚类可视化

案例实战 9.2 使用 kmeans 算法对鸢尾花进行聚类。参考 iris_3means.py

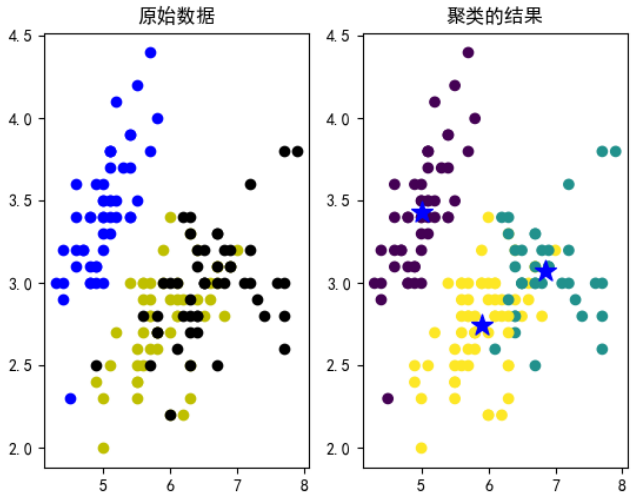


图 2: Kmeans 聚类可视化

K 均值算法并不适用于所有的聚类问题。如果数据集是由凸集构成，则 k 均值算法能有较好的效果，收敛速度也比较快，但如果数据分布不满足以上条件，则 k 均值算法可能无法取得理想的结果。此外，k 均值算法要求预先确定聚类的类别数 k，这也是它的一个不足之处。

目录

合并聚类算法是一种经典的层次聚类算法，它的思想有点类似于哈夫曼树的合并过程。假设有 m 个数据样本聚为 k 个类。用合并算法聚类时，先将每个数据样本看成一个独立的类，然后每次合并距离最小的两个类成为一个新的类，并在后面的合并中，将原来的两个类排除，直至 m 个数据样本聚合成 k 个类为止。

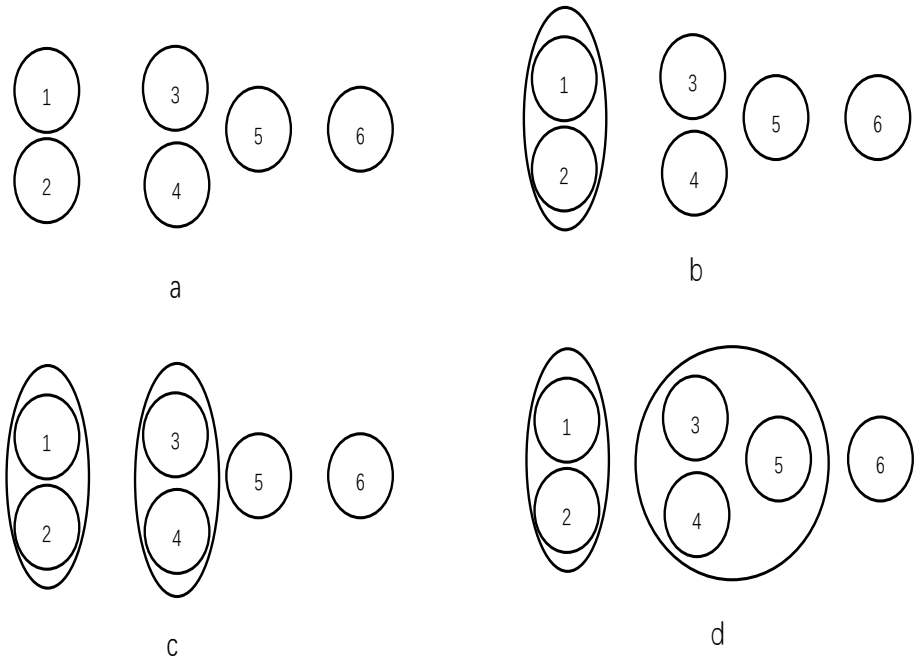


图 3: 合并聚类算法聚为 3 类的过程

合并聚类算法

Algorithm 2: 合并聚类算法

for $i = 1, 2, \cdots, m$ **do**

  $C_i = \{\boldsymbol{x}^{(i)}\}$

$\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$

$new\_id = m + 1$

while $|\mathcal{C}| > k$ **do**

  选择两个类距离最小的类 $C_{i_1}, C_{i_2} \in \mathcal{C}$

  $C_{new\_id} = C_{i_1} \cup C_{i_2}$

  $\mathcal{C} \leftarrow \mathcal{C} - C_{i_1} - C_{i_2} + C_{new\_id}$

  $new\_id \leftarrow new\_id + 1$

return $\mathcal{C}$

假设算法的输入为 m 个样本 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}$，聚类的类数 k 是算法的一个参数，用 $\mathcal{C}$ 表示当前聚成的类组成的集合，初始时每个样本 $\boldsymbol{x}^{(i)}$ 自成一类 C_i，算法以迭代的方式合并 $\mathcal{C}$ 中的类，在每一次迭代循环中，算法选取 $\mathcal{C}$ 中类间距离最小的两个类 C_{i_1} 和 C_{i_2} 进行合并，在循环过程中，用 new_id 表示当前未被使用过的最小可用的类编号，生成新类之后，算法将进行合并操作的两个类 C_{i_1} 和 C_{i_2} 删除，并将新生成的新类 C_{new_id} 合并进 $\mathcal{C}$ 中，因此，每循环一次，$\mathcal{C}$ 中的类别数减 1，以此类推，经过 $m - k$ 次循环之后，$\mathcal{C}$ 中就只含 k 个类，这时算法结束。

合并聚类的实现

为了更高效实现算法2，将各个类对 (C_i, C_j)（初始时为 $\binom{m}{2}$ 对）存进一个优先队列中，类对 (C_i, C_j) 的值为这两个类之间的距离，则优先队列的队首元素就是 $\mathcal{C}$ 中类间距离最小的类对 (C_{i_1}, C_{i_2})。其实现代码参 agglomerative_clustering.py

案例实战 9.3 使用合并聚类算法对鸢尾花进行聚类。 iris_agglomerative_clustering.py

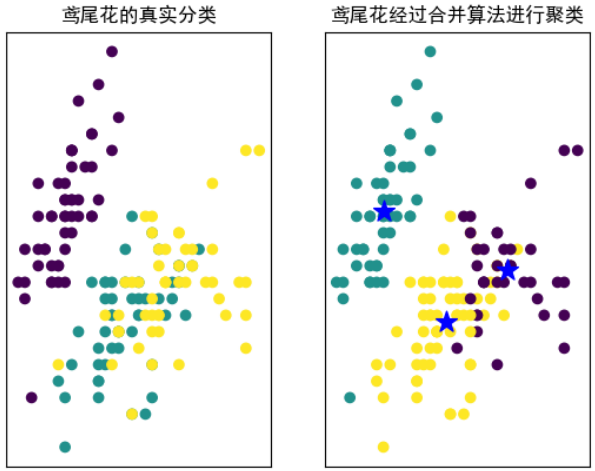
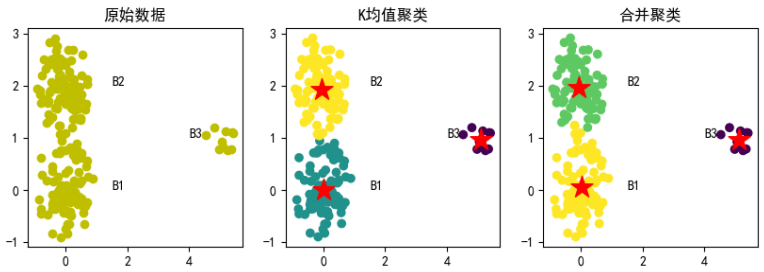
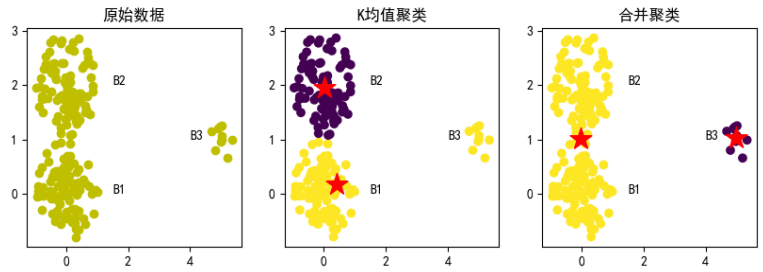


图 4: 鸢尾花真实分类与合并聚类算法分类

案例实战 9.4 k 均值聚类和合并聚类的对比。参考代码 kmeans_vs_agglomerative.py



(a) 聚成 3 类



(b) 聚成 2 类

图 5: k 均值聚类和合并聚类对比

目录

基于距离的聚类算法的聚类结果是球状的类别，当数据集中的聚类结果是非球状结构时，基于距离的聚类算法的聚类效果并不好。与基于距离的聚类算法不同的是，基于密度的聚类算法可以发现任意形状的聚类。在基于密度的聚类算法中，通过在数据集中寻找被低密度区域分离的高密度区域，将分离出的高密度区域作为一个独立的类别。

DBSCAN（Density-Based Spatial Clustering of Applications with Noise，带噪声的基于密度的聚类方法）是一种基于密度的空间聚类算法。该算法将具有足够密度的区域划分为一类，并在具有噪声的空间数据集中发现任意形状的类别，它将类别定义为密度相连的点的最大集合。

DBSCAN 算法的思想是，首先从任一个样本开始向外扩充出一个类，算法不断地将类中样本的 ϵ 邻域内的样本点加入到类中，直至没有新的样本可以加入其中为止。一个样本的 ϵ 邻域定义为与该样本的距离不超过 ϵ 的样本集合。ϵ 的取值由算法设计者指定。这样就生成了第一个类。然后算法再任选一个不属于第一个类的样本点，重复上述过程，生成第二个类。如此重复，直到所有样本都被归类完毕为止。

在上述扩充过程中，由于可能存在噪声的干扰，因此，算法还进一步指定了样本的 ϵ 稠密邻域，即当一个样本的 ϵ 邻域中含有多于指定数目的样本时，就是一个 ϵ 稠密邻域。DBSCAN 算法生成类时，只将稠密邻域中的样本加入类中，而忽略可能是噪声的非稠密邻域中的样本。

给定 m 个样本 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)} \in \mathbb{R}^n$，数组 assignments 用于记录每个样本所属的类编号，$grow\_cluster(i, id, eps, assignments)$ 函数用于生成一个以当前 id 值为编号的类，其中，搜索邻域的半径 ϵ 和稠密邻域的样本数下限 min_sample 需要由算法设计者事先根据实际问题和经验指定。

Algorithm 3: DBSCAN 算法

for $i = 1, 2, \cdots, m$ **do**
  $assignments[i] \leftarrow 0$
$id = 1$
for $i = 1, 2, \cdots, m$ **do**
  if $assignments[i] == 0$ *and* $|N(\boldsymbol{x}^{(i)}, \epsilon)| > min\_sample$ **then**
    $grow\_cluster(i, id, eps, assignments)$
    $id \leftarrow id + 1$
return $assignments$

初始化时，将数组 $assignments[i]$ 设置为 0，表示样本尚未归入任何一个类，随后算法开始循环逐一扫描样本 $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \cdots, \boldsymbol{x}^{(m)}$。当扫描到一个未被归类的样本 $\boldsymbol{x}^{(i)}$ 时，算法判断 $\boldsymbol{x}^{(i)}$ 的 ϵ 邻域是否为稠密邻域，如果是，则从 $\boldsymbol{x}^{(i)}$ 开始调用 $grow\_cluster(i, id, eps, assignments)$ 函数，生成一个以当前 id 值为编号的类，同时 id 值加 1，为下一个类做准备。函数 $grow\_cluster(i, id, eps, assignments)$ 的实现如下：

Algorithm 4: 增长同一类的元素

def $grow\_cluster(i, id, eps, assignments)$**:**
    $assignments[i] \leftarrow id$
    $Q = N(\boldsymbol{x}^{(i)}, \epsilon)$ #Q 为队列。
    while $|Q| > 0$ **do**
        $j = Q.pop()$
        if $assignments[j] == 0$ **then**
            $assignments[j] \leftarrow id$
            if $|N(\boldsymbol{x}^{(j)}, \epsilon)| > min\_sample$ **then**
                $Q.push(N(\boldsymbol{x}^{(j)}, \epsilon))$

类的生成算法先将 $\boldsymbol{x}^{(i)}$ 归入以 id 为编号的类中，然后用一个先进先出的队列 Q 来扩充这个类。先将所有 $\boldsymbol{x}^{(i)}$ 的 ϵ 邻域 $N(\boldsymbol{x}^{(i)}, \epsilon)$ 中的样本加入 Q 中，只要队列 Q 非空，就让队首元素 j 出队。如果样本 $\boldsymbol{x}^{(j)}$ 尚未被归类，则将其归入当前类。此时，如果 $\boldsymbol{x}^{(j)}$ 的 ϵ 邻域 $N(\boldsymbol{x}^{(j)}, \epsilon)$ 是稠密的，则将 $N(\boldsymbol{x}^{(j)}, \epsilon)$ 中的样本加入到队列 Q 中。这个循环持续到队列 Q 被清空。此时，已经没有可选择的新样本可以加入类中了。至此，算法成功地生成了一个以 id 为编号的类。

从算法的描述可以看出，DBSCAN 算法对数据样本聚类时，不需要预先指定聚类的类别数，算法会自动地根据样本分布的密度，将数据聚类。

算法的实现 dbscan.py

案例实战 9.5 环形数据集和月亮形数据集的 DBSCAN 和 KMEANS 聚类。circle_dbscan_vs_kmeans.py

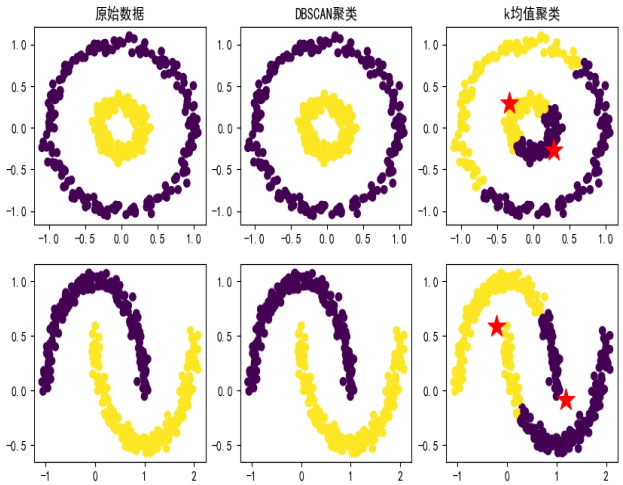


图 6: 环形数据集和月亮数据集的聚类

生成的原始数据是带标签的，展示了其真实的分类；中间的 DBSCAN 算法运行结果表明该算法成功地区分了环形数据集和月亮形数据集，但右边的 k 均值算法却无法区分这两类数据集。当然，这与选择适当的算法超参数 eps 和 min_sample 也有关系。

这个例子说明，当每个类中的数据都不是凸集时，k 均值算法不能保证有合理的聚类结果，而 DBSCAN 算法则不同，它可以处理任何不规则的数据分布。与传统的 k 均值算法相比，DBSCAN 最大的不同就是不需要输入类别数 k，当然它最大的优势是可以发现任意形状的群组，而不是像 k 均值算法，一般仅仅使用于凸的样本集聚类。同时它在聚类的时还可以找出异常点。

什么时候需要用 DBSCAN 来聚类呢？一般来说，如果数据集是稠密的，并且数据集不是凸的，那么用 DBSCAN 会比 k 均值算法聚类效果好很多。如果数据集不是稠密的，则不推荐用 DBSCAN 来聚类。

DBSCAN 的主要优点有：

1. 可以对任意形状的稠密数据集进行聚类，而 k 均值之类的聚类算法一般只适用于凸数据集。
2. 不需要预先指定聚类的类别数，它可以根据数据分布的密度自行决定。
3. 可以在聚类的同时发现异常点，对数据集中的异常点不敏感。
4. 聚类结果没有偏倚（多次运行结果稳定），而 k 均值之类的聚类算法初始值对聚类结果有很大影响。

DBSCAN 的主要缺点有：

1. 如果样本集的密度不均匀、聚类间距差相差很大时，聚类质量较差，这时用 DBSCAN 聚类一般不适合。
2. 如果样本集较大时，聚类收敛时间较长，此时需要对算法进行一些改进。
3. 调参相对于传统的 k 均值之类的聚类算法稍复杂，主要需要对邻域阈值 eps，邻域样本数 min_sample 联合调参，不同的参数组合对最后的聚类效果有较大影响。

目录

Sklearn 提供了多种聚类的算法，功能也比前面几节更为丰富和强大。未标记的数据聚类可以使用模块 sklearn.cluster 来实现。例如：Kmeans（k 均值）、Affinity propagation（近邻传播算法或者亲和力传播算法）、Mean-shift（均值漂移算法）、谱聚类（Spectral clustering）、合并聚类（Agglomerative clustering）和带噪声的基于密度的空间聚类算法（DBSCAN）等等。

案例实战 9.6：使用 kmeans 算法聚类手写数字 digits_kmeans_sklearn.py

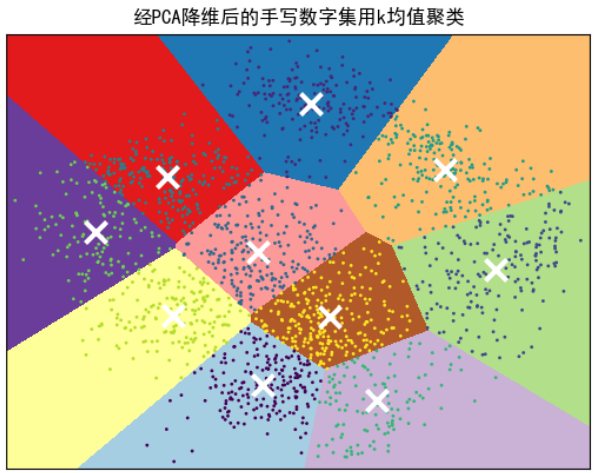


图 7: 手写数字降维后的 k 均值聚类结果

案例实战 9.7：KMeans 与 MiniBatchKMeans 对比。
kmeans_vs_minibatchkmeans.py

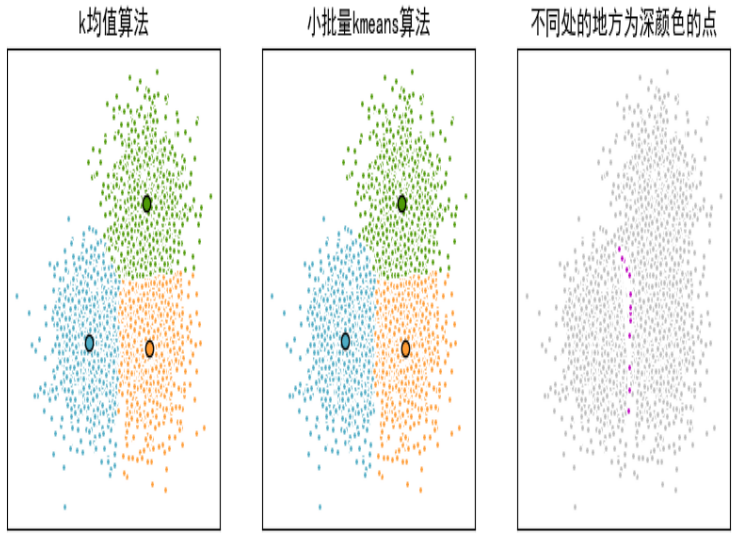


图 8: k 均值算法与小批量 k 均值算法对比可视化

两种算法几乎没什么差别，聚类结果只有少数不同的点，在第三个子图中以深颜色刻画了出来。

近邻传播算法 Affinity Propagation

AP 算法是通过在样本之间发送消息直到收敛的方式来创建聚类。使用少量模范样本作为聚类中心来描述数据集，这些模范样本被认为是最能代表数据集中其他数据的样本。在样本对之间发送消息表示一个样本作为另一个样本的模范样本的适合程度，其值根据通信的反馈不断更新迭代直至收敛，完成聚类中心的选取。

AP 算法可以根据提供的数据决定聚类的数目。有两个比较重要的参数，偏好（preference）决定使用多少个模范样本，阻尼因子（damping factor）减少吸引信息和归属信息以减少更新这些信息时的数据振荡。

AP 聚类算法主要的缺点是算法的复杂度，其时间复杂度是 $O(m^2T)$，其中 m 是样本的个数，T 是收敛之前迭代的次数。如果使用密集的相似性矩阵空间复杂度是 $O(m^2)$，如果使用稀疏的相似性矩阵空间复杂度可以降低。这使得 AP 聚类最适合中小型数据集。

案例实战 9.8 近邻传播聚类算法示例。
affinity_propagation_sklearn.py

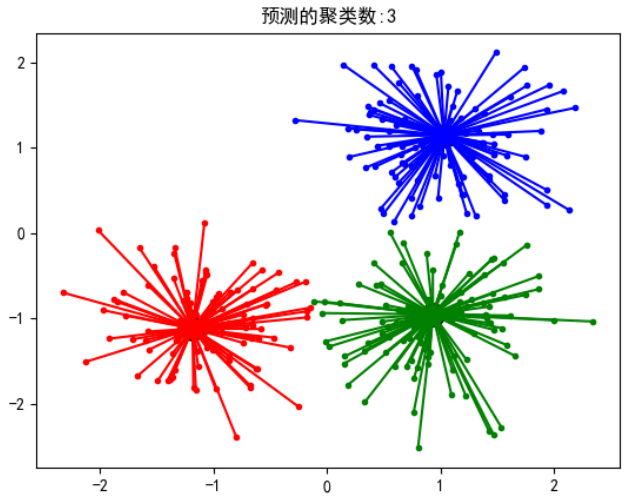


图 9: 近邻传播算法聚类示例可视化

均值漂移算法（Mean-shift）

均值漂移算法旨在于发现一个样本密度平滑的“斑点”（墨渍）。均值漂移算法是基于质心的算法，通过更新质心的候选位置，这些候选位置通常是所选定区域内点的均值。然后，这些候选位置在后处理阶段被过滤以消除近似重复的对象，从而形成最终质心集合。给定第 t 次迭代中的候选质心 $\boldsymbol{x}_i$，候选质心的位置将被按照如下公式更新：$\boldsymbol{x}_i^{t+1} = m(\boldsymbol{x}_i^t)$，$m(\boldsymbol{x}_i)$ 的表达式如下：

$$m(\boldsymbol{x}_i) = \frac{\sum_{\boldsymbol{x}_j \in N(\boldsymbol{x}_i)} K(\boldsymbol{x}_j - \boldsymbol{x}_i)\boldsymbol{x}_j}{\sum_{\boldsymbol{x}_j \in N(\boldsymbol{x}_i)} K(\boldsymbol{x}_j - \boldsymbol{x}_i)}$$

其中，其中 $N(\boldsymbol{x}_i)$ 是围绕 $\boldsymbol{x}_i$ 周围一个给定距离范围内的样本邻域，m 是均值偏移向量，该向量是所有质心中指向点密度增加最多的区域的偏移向量。使用上式计算，有效地将质心更新为其邻域内样本的平均值，K 为核函数。算法自动设定聚类的数目，而不是依赖参数带宽，带宽是决定搜索区域大小的参数。

案例实战 9.9：随机数据的均值漂移聚类示例。参考 mean_shift_sklearn.py

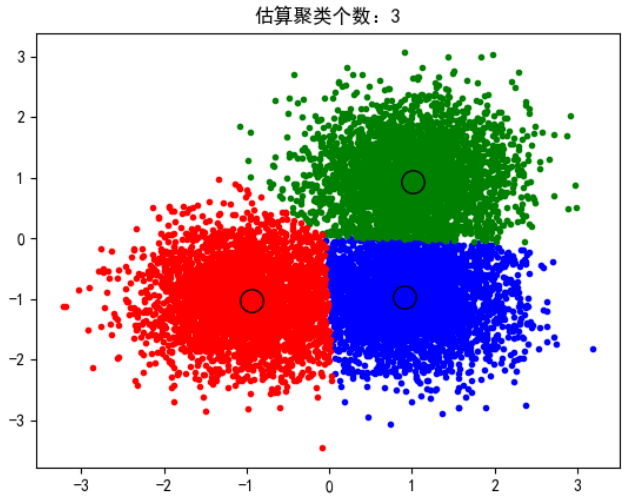


图 10: 均值漂移算法聚类示例可视化