

# 聚类

## 1. 模型的一些通用方法:

- `get_params([deep])`: 返回模型的参数。
  - `deep`: 如果为 `True`, 则可以返回模型参数的子对象。
- `set_params(**params)`: 设置模型的参数。
  - `params`: 待设置的关键字参数。
- `fit(X[, y, sample_weight])`: 训练模型。
  - `X`: 样本集合。通常是一个 `numpy array`, 每行代表一个样本, 每列代表一个特征。
  - `y`: 样本的标签集合。它与 `X` 的每一行相对应。
  - `sample_weight`: 样本的权重。其形状为 `[n_samples,]`, 每个元素代表一个样本的权重。
- `predict(X, sample_weight)`: 返回每个样本所属的簇标记。
  - `X`: 样本集合。通常是一个 `numpy array`, 每行代表一个样本, 每列代表一个特征。
  - `sample_weight`: 样本的权重。其形状为 `[n_samples,]`, 每个元素代表一个样本的权重。
- `fit_predict(X[, y, sample_weight])`: 训练模型并执行聚类, 返回每个样本所属的簇标记。
  - `X`: 样本集合。通常是一个 `numpy array`, 每行代表一个样本, 每列代表一个特征。
  - `y`: 样本的标签集合。它与 `X` 的每一行相对应。
  - `sample_weight`: 样本的权重。其形状为 `[n_samples,]`, 每个元素代表一个样本的权重。
- `transform(X)`: 将数据集 `X` 转换到 `cluster center space`。
 

在 `cluster center space` 中, 样本的维度就是它距离各个聚类中心的距离。

  - `X`: 样本集合。通常是一个 `numpy array`, 每行代表一个样本, 每列代表一个特征。
- `fit_transform(X[, y, sample_weight])`: 训练模型并执行聚类, 将数据集 `X` 转换到 `cluster center space`。
  - `X`: 样本集合。通常是一个 `numpy array`, 每行代表一个样本, 每列代表一个特征。
  - `y`: 样本的标签集合。它与 `X` 的每一行相对应。
  - `sample_weight`: 样本的权重。其形状为 `[n_samples,]`, 每个元素代表一个样本的权重。

## 2. 模型的一些通用参数:

- `n_jobs`: 一个正数, 指定任务并行时指定的 `CPU` 数量。  
如果为 `-1` 则使用所有可用的 `CPU`。
- `verbose`: 一个正数。用于开启/关闭迭代中间输出日志功能。
  - 数值越大, 则日志越详细。
  - 数值为0或者 `None`, 表示关闭日志输出。
- `max_iter`: 一个整数, 指定最大迭代次数。  
如果为 `None` 则为默认值 (不同 `solver` 的默认值不同)。
- `tol`: 一个浮点数, 指定了算法收敛的阈值。
- `random_state`: 一个整数或者一个 `RandomState` 实例, 或者 `None`。
  - 如果为整数, 则它指定了随机数生成器的种子。
  - 如果为 `RandomState` 实例, 则指定了随机数生成器。

- 如果为 `None`，则使用默认随机数生成器。

## 一、KMeans

### 1.1 KMeans

1. `KMeans` 是 `scikit-learn` 提供的 `k` 均值算法模型，其原型为：

```
class sklearn.cluster.KMeans(n_clusters=8, init='k-means++', n_init=10,
max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0,
random_state=None, copy_x=True, n_jobs=1, algorithm='auto')
```

- `n_clusters`：一个整数，指定分类簇的数量。
- `init`：一个字符串，指定初始均值向量的策略。可以为：
  - `'k-means++'`：该初始化策略选择的初始均值向量相互之间都距离较远，它的效果较好。
  - `'random'`：从数据集中随机选择  $K$  个样本作为初始均值向量。
  - 或者提供一个数组，数组的形状为 `(n_clusters, n_features)`，该数组作为初始均值向量。

`K` 均值算法总能够收敛，但是其收敛情况高度依赖于初始化的均值。有可能收敛到局部极小值。因此通常都是用多组初始均值向量来计算若干次，选择其中最优的那一次。而 `'k-means++'` 策略选择的初始均值向量可以一定程度上的解决这个问题。
- `n_init`：一个整数，指定了 `k` 均值算法运行的次数。  
每次都会选择一组不同的初始化均值向量，最终算法会选择最佳的分类簇来作为最终的结果。
- `max_iter`：一个整数，指定了单轮 `k` 均值算法中，最大的迭代次数。  
算法总的最大迭代次数为 `max_iter * n_init`。
- `precompute_distances`：指定是否提前计算距离。如果提前计算距离，则需要更多的内存，但是算法会运行的更快。  
可以为布尔值或者字符串 `'auto'`：
  - `'auto'`：如果 `n_samples * n_clusters > 12 million`，则不提前计算。
  - `True`：总是提前计算。
  - `False`：总是不提前计算。
- `tol`：指定收敛阈值。
- `n_jobs`：指定并行度。
- `verbose`：指定开启/关闭迭代中间输出日志功能。
- `random_state`：指定随机数种子。
- `copy_x`：布尔值，主要用于 `precompute_distances=True` 的情况。
  - 如果为 `True`，则预计算距离的时候，并不修改原始数据。
  - 如果为 `False`，则预计算距离的时候，会修改原始数据用以节省内存；然后当算法结束的时候，会将原始数据还原。但是可能会因为浮点数的表示，会有一些精度误差。
- `algorithm`：一个字符串，指定采用的算法。可以为：
  - `'full'`：使用经典的 `EM` 风格的算法。
  - `'elkan'`：使用 `'elkan'` 变种算法。它通过使用三角不等式来优化算法，但是不支持稀疏数据。

- 'auto': 自动选择算法。对于稀疏数据, 使用 'full'; 对于密集数据, 使用 'elkan'。

## 2. 属性:

- `cluster_centers_`: 一个形状为 `[n_clusters, n_features]` 的数组, 给出分类簇的均值向量。
- `labels_`: 一个形状为 `[n_samples,]` 的数组, 给出了每个样本所属的簇的标记。
- `inertia_`: 一个浮点数, 聚类平方误差  $err = \sum_{k=1}^K \sum_{\vec{x}_i \in C_k} \|\vec{x}_i - \vec{\mu}_k\|_2^2$ 。
- `n_iter_`: 一个整数, 指定运行的迭代次数。

## 3. 方法:

- `fit(X[, y, sample_weight])`: 训练模型。
- `fit_predict(X[, y, sample_weight])`: 训练模型并执行聚类, 返回每个样本所属的簇标记。
- `predict(X, sample_weight)`: 返回每个样本所属的簇标记。
- `transform(X)`: 将数据集 `X` 转换到 `cluster center space`。
- `fit_transform(X[, y, sample_weight])`: 训练模型并执行聚类, 将数据集 `X` 转换到 `cluster center space`。
- `score(X[, y, sample_weight])`: 一个浮点数, 给出了聚类平方误差的相反数:  

$$-\sum_{k=1}^K \sum_{\vec{x}_i \in C_k} \|\vec{x}_i - \vec{\mu}_k\|_2^2$$

# 1.2 MiniBatchKMeans

1. `MiniBatchKMeans` 是 `scikit-learn` 提供的 批量 `k` 均值算法模型, 其原型为:

```
class sklearn.cluster.MiniBatchKMeans(n_clusters=8, init='k-means++', max_iter=300,
batch_size=100, verbose=0, compute_labels=True, random_state=None, tol=0.0,
max_no_improvement=10, init_size=None, n_init=3, reassignment_ratio=0.01)
```

- `batch_size`: 一个整数, 指定 `batch` 大小。
- `compute_labels`: 一个布尔值, 指定当算法收敛时, 是否对全量数据集重新计算其完整的簇标记。
- `tol`: 一个浮点数, 指定收敛阈值。它可以用于早停。

当迭代前后聚类中心的变化小于它时, 执行早停。如果为 `0.0`, 则不开启这种早停。

- `max_no_improvement`: 一个整数, 用于控制早停的轮数。如果优化目标在连续 `max_no_improvement` 个 `batch` 内没有改善时, 执行早停。

这里的优化目标不是聚类中心的变化, 而是平方误差  $err = \sum_{k=1}^K \sum_{\vec{x}_i \in C_k} \|\vec{x}_i - \vec{\mu}_k\|_2^2$ 。

- `init_size`: 一个整数, 为加速初始化而随机采样的样本数。通常是 3 倍的 `batch_size`。它必须大于 `n_clusters`。

- `n_init`: 一个整数, 指定了初始化的尝试次数。

与 `KMeans` 不同, `MiniBatchKMeans` 只会运行一轮 (而不是多轮)。

- `reassignment_ratio`: 一个浮点数, 控制每次迭代中最多有多少个簇中心被重新赋值。

如果该值较大, 则模型可能收敛可能时间更长, 但是聚类效果也会更好。

- 其他参数参考 `sklearn.cluster.KMeans`。

2. 属性: 参考 `sklearn.cluster.KMeans`。

## 3. 方法:

- `partial_fit(X, y=None, sample_weight=None)`: 训练 `k means` 一个批次。

- 其它方法参考 `sklearn.cluster.KMeans` 。

## 二、DBSCAN

1. DBSCAN 是 `scikit-learn` 提供了一种密度聚类模型。其原型为：

```
class sklearn.cluster.DBSCAN(eps=0.5, min_samples=5, metric='euclidean',
metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None)
```

- `eps`： $\epsilon$  参数，用于确定邻域大小。
- `min_samples`：*MinPts* 参数，用于判断核心对象。
- `metric`：一个字符串或者可调用对象，用于计算距离。  
如果是字符串，则必须是 `metrics.pairwise.calculate_distance` 中指定的。
- `metric_params`：一个字典，当 `metric` 为可调用对象时，为 `metric` 提供关键字参数。
- `algorithm`：一个字符串，用于计算两点间距离并找出最近邻的点。可以为：
  - `'auto'`：由算法自动选取合适的算法。
  - `'ball_tree'`：用 `ball` 树来搜索。
  - `'kd_tree'`：用 `kd` 树来搜索。
  - `'brute'`：暴力搜索。
- `leaf_size`：一个整数，用于指定当 `algorithm=ball_tree` 或者 `kd_tree` 时，树的叶结点大小。  
该参数会影响构建树、搜索最近邻的速度，同时影响存储树的内存。
- `p`：一个浮点数，指定闵可夫斯基距离的  $p$  值。
- `n_jobs`：指定并行度。

2. 属性：

- `core_sample_indices_`：一个形状为 `[n_core_samples,]` 的数组，给出了核心样本在原始训练集中的位置。
- `components_`：一个形状为 `[n_core_samples, n_features]` 的数组，给出了核心样本的一份拷贝
- `labels_`：一个形状为 `[n_samples,]` 的数组，给出了每个样本所属的簇标记。

对于噪音样本，其簇标记为 -1。

3. 方法：

- `fit(X[, y, sample_weight])`：训练模型。
- `fit_predict(X[, y, sample_weight])`：训练模型并执行聚类，返回每个样本所属的簇标记。

4. 考察  $\epsilon$  参数的影响：

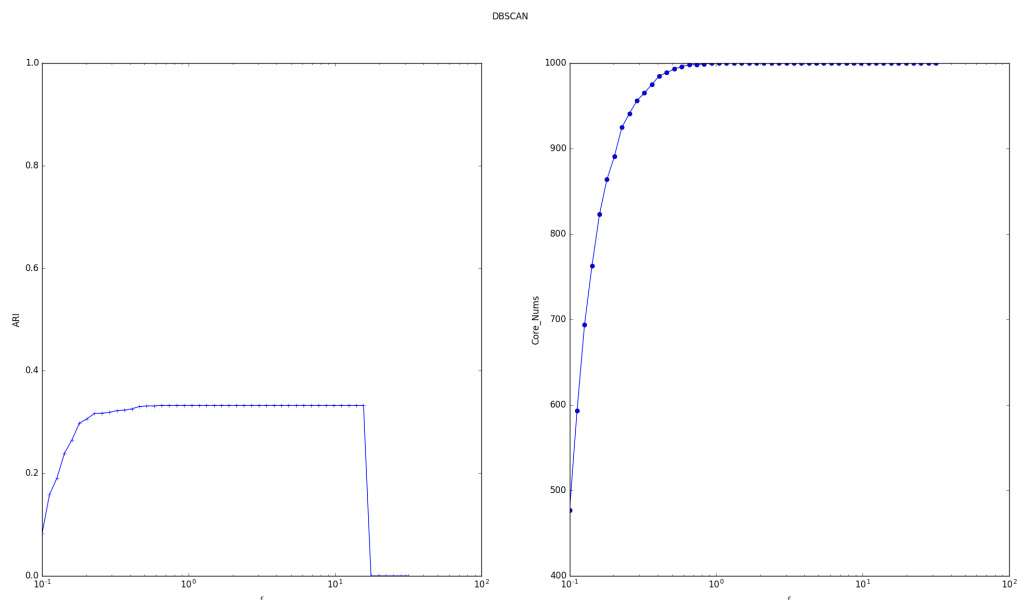
- `ARI` 指数随着  $\epsilon$  的增长，先上升后保持平稳最后断崖下降。

断崖下降是因为产生的训练样本的间距比较小，最远的两个样本点之间的距离不超过 30。当  $\epsilon$  过大时，所有的点都在一个邻域中。

- 核心样本数量随着  $\epsilon$  的增长而上升。

这是因为随着  $\epsilon$  的增长，样本点的邻域在扩展，则样本点邻域内的样本会更多，这就产生了更多满足条件的核心样本点。

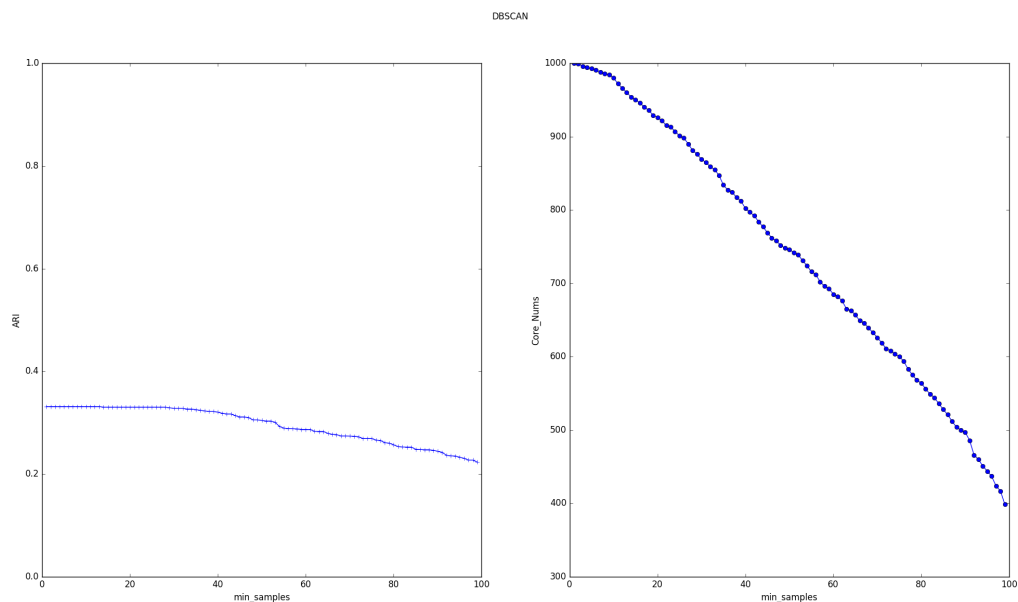
但是样本集中的样本数量有限，因此核心样本点数量的增长到一定数目后稳定。



### 5. 考察 *MinPts* 参数的影响：

- ARI 指数随着 *MinPts* 的增长，平稳的下降。
- 核心样本数量随着 *MinPts* 的增长基本上线性下降。

这是因为随着 *MinPts* 的增长，样本点的邻域中必须包含更多的样本才能使它成为一个核心样本点。因此产生的核心样本点越来越少。



## 三、MeanShift

1. `MeanShift` 是 `scikit-learn` 提供了一种密度聚类模型。其原型为：

```
class sklearn.cluster.MeanShift(bandwidth=None, seeds=None, bin_seeding=False,
min_bin_freq=1, cluster_all=True, n_jobs=None)
```

- `bandwidth`：一个浮点数，指定带宽参数。

如果未指定, 则通过 `sklearn.cluster.estimate_bandwidth()` 函数来自动计算。

- `seeds`: 一个形状为 `[n_samples, n_features]` 的数组, 用于初始化核函数。

如果未指定, 则通过 `sklearn.cluster.get_bin_seeds()` 函数来自动计算。

- `bin_seeding`: 一个布尔值。
    - 如果为 `True`, 则并不会使用所有的点来计算核函数, 而是使用网格边界上的点 (网格宽度为带宽) 来计算。这会加速算法的执行, 因为核函数的初始化需要的点大大降低。
    - 如果为 `False`, 则使用所有的点来计算核函数。
  - `min_bin_freq`: 一个整数值, 指定有效网格包含的数据点的最少数量。
- 当 `bin_seeding=True` 时, 仅仅接收那些网格内包含超过 `min_bin_freq` 个数据点的网格。
- `cluster_all`: 一个布尔值, 指定是否对所有数据点进行聚类。
    - 如果为 `False`, 则对离群点不聚类, 将离群点的簇标记设置为 `-1`。
    - 如果为 `True`, 则对离群点也聚类, 将离群点划分到离它最近的簇中。
  - `n_jobs`: 一个整数, 指定并行度。

## 2. 属性:

- `cluster_centers_`: 一个形状为 `[n_clusters, n_features]` 的数组, 给出了每个簇中心的坐标。
  - `labels_`: 一个形状为 `[n_samples,]` 的数组, 给出了每个样本所属的簇标记。
- 如果 `cluster_all=False`, 则对于离群样本, 其簇标记为 `-1`。

## 3. 方法:

- `fit(X[, y])`: 训练模型。
- `fit_predict(X[, y])`: 训练模型并执行聚类, 返回每个样本所属的簇标记。
- `predict(X)`: 对每个样本预测其簇标记。

每个样本距离最近的簇就是该样本所属的簇。

# 四、AgglomerativeClustering

1. `AgglomerativeClustering` 是 `scikit-learn` 提供的一种层次聚类模型。其原型为:

```
class sklearn.cluster.AgglomerativeClustering(n_clusters=2, affinity='euclidean',
memory=Memory(cachedir=None), connectivity=None, n_components=None,
compute_full_tree='auto', linkage='ward', pooling_func=<function mean>)
```

- `n_clusters`: 一个整数, 指定簇的数量。
  - `connectivity`: 一个数组或者可调用对象或者为 `None`, 用于指定连接矩阵。它给出了每个样本的可连接样本。
  - `affinity`: 一个字符串或者可调用对象, 用于计算距离。可以为: `'euclidean', 'l1', 'l2', 'manhattan', 'cosine', 'precomputed'`。
- 如果 `linkage='ward'`, 则 `'affinity'` 必须是 `'euclidean'`
- `memory`: 用于缓存输出的结果, 默认为不缓存。如果给定一个字符串, 则表示缓存目录的路径。
  - `n_components`: 将在 `scikit-learn v 0.18` 中移除

- `compute_full_tree`：通常当已经训练了 `n_clusters` 之后，训练过程就停止。  
但是如果 `compute_full_tree=True`，则会继续训练从而生成一颗完整的树。
- `linkage`：一个字符串，用于指定链接算法。
  - `'ward'`：采用方差恶化距离 `variance increass distance`。
  - `'complete'`：全链接 `complete-linkage` 算法，采用  $d_{max}$ 。
  - `'average'`：均链接 `average-linkage` 算法，采用  $d_{avg}$ 。
  - `'single'`：单链接 `single-linkage` 算法，采用  $d_{min}$ 。
- `pooling_func`：即将被废弃的接口。

## 2. 属性：

- `labels_`：一个形状为 `[n_samples,]` 的数组，给出了每个样本的簇标记。
- `n_leaves_`：一个整数，给出了分层树的叶结点数量。
- `n_components_`：一个整数，给除了连接图中的连通分量的估计值。
- `children_`：一个形状为 `[n_samples-1,2]` 数组，给出了每个非叶结点中的子节点数量。

## 3. 方法：

- `fit(X[, y])`：训练模型。
- `fit_predict(X[, y])`：训练模型并执行聚类，返回每个样本所属的簇标记。

# 五、BIRCH

1. `BIRCH` 是 `scikit-learn` 提供的一种层次聚类模型。其原型为：

```
class sklearn.cluster.Birch(threshold=0.5, branching_factor=50, n_clusters=3,
                             compute_labels=True, copy=True)
```

- `threshold`：一个浮点数，指定空间阈值  $\tau$ 。
- `branching_factor`：一个整数，指定枝平衡因子  $\beta$ 。叶平衡因子  $\lambda$  也等于该数值。
- `n_clusters`：一个整数或者 `None` 或者 `sklearn.cluster` 模型，指定最终聚类的数量。
  - 如果为 `None`，则由算法自动给出。
  - 如果为一个整数，则使用 `AgglomerativeClustering` 算法来对 `CF` 本身执行聚类，并将聚类结果返回。这使得最终的聚类数量就是 `n_clusters`。
  - 如果为一个 `sklearn.cluster` 模型，则该模型对 `CF` 本身执行聚类，并将聚类结果返回。
- `compute_labels`：一个布尔值，指定是否需要计算簇标记。
- `copy`：一个布尔值，指定是否拷贝原始数据。

## 2. 属性：

- `root_`：一个 `_CFNode` 对象，表示 `CF` 树的根节点。
- `subcluster_centers_`：一个数组，表示所有子簇的中心点。  
它直接从所有叶结点中读取。
- `subcluster_labels_`：一个数组，表示所有子簇的簇标记。  
可能多个子簇会有同样的簇标记，因为子簇可能会被执行进一步的聚类。
- `labels_`：一个形状为 `[n_samples,]` 的数组，给出了每个样本的簇标记。



如果执行分批训练，则它给出的是最近一批样本的簇标记。

### 3. 方法：

- `fit(X[, y])`：训练模型。
- `partial_fit(X[, y])`：分批训练模型（在线学习）。
- `fit_predict(X[, y])`：训练模型并执行聚类，返回每个样本所属的簇标记。
- `predict(X)`：对每个样本预测其簇标记。

根据每个子簇的中心点来预测样本的簇标记。

- `transform(X)`：将样本转换成子簇中心点的坐标：维度 $d$ 代表样本距离第 $d$ 个子簇中心的距离。
- `fit_transform(X[, y])`：训练模型并将样本转换成子簇中心点的坐标。

## 六、GaussianMixture

1. `GaussianMixture` 是 `scikit-learn` 给出的混合高斯模型。其原型为

```
class sklearn.mixture.GaussianMixture(n_components=1, covariance_type='full',
tol=0.001, reg_covar=1e-06, max_iter=100, n_init=1, init_params='kmeans',
weights_init=None, means_init=None, precisions_init=None, random_state=None,
warm_start=False, verbose=0, verbose_interval=10)
```

- `n_components`：一个整数，指定分模型的数量，默认为1。
- `covariance_type`：一个字符串，指定协方差的类型。必须为下列值之一：
  - `'spherical'`：球形，每个分模型的协方差矩阵都是各自的标量值。
  - `'tied'`：结点型，所有的分模型都共享一个协方差矩阵。
  - `'diag'`：对角型，每个分模型的协方差矩阵都是各自的对角矩阵
  - `'full'`：全型，每个分模型都有自己的协方差矩阵。
- `tol`：一个浮点数，用于指定收敛的阈值。EM 迭代算法中，当对数似然函数平均增益低于此阈值时，迭代停止。
- `reg_covar`：一个浮点数，添加到协方差矩阵对角线上元素，确保所有的协方差都是正数。
- `max_iter`：一个整数，指定 EM 算法迭代的次数。
- `n_init`：一个整数，用于指定初始化次数。即算法会运行多轮，只有表现最好的哪个结果作为最终的结果。
- `init_params`：一个字符串，可以为 `'kmeans'/'random'`，用于指定初始化权重的策略。
  - `'kmeans'`：通过 `kmeans` 算法初始化。
  - `'random'`：随机初始化。
- `weights_init`：一个序列，形状为 `(n_components,)`，指定初始化的权重。  
如果提供该参数，则不会使用 `init_params` 来初始化权重。
- `means_init`：一个数组，形状为 `(n_components, n_features)`，指定了初始化的均值。  
如果为 `None`，则使用 `init_params` 来初始化均值。
- `precision_init`：用户提供的初始 `precisions`（协方差矩阵的逆矩阵）。如果为 `None`，则使用 `init_params` 来初始化。



根据 `covariance_type` 的不同, 该参数值的形状为不同。

- `'spherical'` : 形状为 `[n_components,]`。
- `'tied'` : 形状为 `[n_features, n_features]`。
- `'diag'` : 形状为 `[n_components, n_features]`。
- `'full'` : 形状为 `[n_components, n_features, n_features]`。
- `random_state` : 指定随机数种子。
- `warm_start` : 一个布尔值。如果为 `True`, 则上一次训练的结果将作为本次训练的开始条件。此时忽略 `n_init`, 并且只有一次初始化过程发生。
- `verbose` : 一个整数, 指定日志打印级别。
- `verbose_interval` : 一个整数, 指定输出日志的间隔。

## 2. 属性:

- `weights_` : 一个形状为 `(n_components,)` 的数组, 表示每个分模型的权重。
- `means_` : 一个形状为 `(n_components, n_features)` 的数组, 表示每个分模型的均值  $\mu_k$ 。
- `covariances_` : 一个数组, 表示每个分模型的方差  $\sigma_k^2$ 。数组的形状根据方差类型有所不同。
  - `'spherical'` : 形状为 `[n_components,]`。
  - `'tied'` : 形状为 `[n_features, n_features]`。
  - `'diag'` : 形状为 `[n_components, n_features]`。
  - `'full'` : 形状为 `[n_components, n_features, n_features]`。
- `precision_` : 一个数组, 表示精度矩阵 (协方差矩阵的逆矩阵), 与 `covariances_` 形状相同。
- `precisions_cholesky_` : 一个数组, 表示精度矩阵的 Cholesky 分解。
 

Cholesky 分解: 如果  $\mathbf{A} \in \mathbb{R}^{n \times n}$  是对称正定矩阵, 则存在一个对角元为正数的下三角矩阵  $\mathbf{L} \in \mathbb{R}^{n \times n}$ , 使得  $\mathbf{A} = \mathbf{L}\mathbf{L}^T$  成立。
- `converged_` : 一个布尔值。当训练过程收敛时, 该值为 `True`; 不收敛时, 为 `False`。
- `n_iter_` : 一个整数, 给出 EM 算法收敛时的迭代步数。
- `lower_bound_` : 一个浮点数, 给出了训练集的对数似然函数的下界。

## 3. 方法:

- `fit(X[, y])` : 训练模型。
- `predict(X)` : 预测样本所属的簇标记。
- `fit_predict(X[, y])` : 训练模型并执行聚类, 返回每个样本所属的簇标记。
- `predict_proba(X)` : 预测样本所属各个簇的后验概率。
- `sample([n_samples])` : 根据模型来随机生成一组样本。 `n_samples` 指定待生成样本的数量。
- `score(X[, y])` : 计算模型在样本总体上的对数似然函数。
- `score_samples(X)` : 给出每个样本的对数似然函数。
- `aic(X)` : 给出样本集的 Akaike 信息准则。
- `bic(X)` : 给出样本集的贝叶斯信息准则。

## 4. AIC 和 BIC 都是用于评估模型好坏的准则, 用于衡量模型的复杂度和模型的精度。

- AIC: Akaike Information Criterion 准则的目标是: 选取 AIC 最小的模型。

其中 AIC 定义为:  $AIC = 2k - 2 \log L$ ,  $k$  为模型的参数个数,  $L$  为模型的似然函数。

- 当模型之间的预测能力存在较大差异时, 似然函数占主导地位。
- 当模型之间的预测能力差异不大时, 模型复杂度占主导地位。

- `BIC`: Bayesian Information Criterion 准则的目标是：选取 `BIC` 最小的模型。

其中 `BIC` 定义为： $BIC = k \log N - 2 \log L$ ， $k$  为模型的参数个数， $L$  为模型的似然函数， $N$  为训练样本的数量。

`BIC` 认为增加参数的数量也会增加模型复杂度。

$k$  并不是超参数的数量，而是训练参数的数量。

## 七、SpectralClustering

1. `SpectralClustering` 是 `scikit-learn` 给出的谱聚类模型。其原型为

```
class sklearn.cluster.SpectralClustering(n_clusters=8, eigen_solver=None,
random_state=None, n_init=10, gamma=1.0, affinity='rbf', n_neighbors=10,
eigen_tol=0.0, assign_labels='kmeans', degree=3, coef0=1, kernel_params=None,
n_jobs=None)
```

- `n_clusters`：一个整数，指定降维的维数，即  $k_1$ 。
- `eigen_solver`：一个字符串或者 `None`，指定求解特征值的求解器。可以为：
  - `'arpack'`：使用 `arpack` 求解器。
  - `'lobpcg'`：使用 `lobpcg` 求解器。
  - `'amg'`：使用 `amg` 求解器。它要求安装 `pyamg`。优点是计算速度快，支持大规模、稀疏数据，但是可能导致不稳定。
- `random_state`：指定随机数种子。
- `n_init`：一个整数，指定二次聚类时的 `k-means` 算法的 `n_init` 参数，它会重复执行 `k-means` 算法 `n_init` 轮，选择效果最好的那轮作为最终聚类的输出。
- `affinity`：一个字符串、数组或者可调用对象，指定相似度矩阵的计算方式。
  - 如果是字符串，则必须是 `'nearest_neighbors'`、`'precomputed'`、`'rbf'`、或者由 `sklearn.metrics.pairwise_kernels` 支持的其它核函数。
  - 如果是一个数组，则直接给出相似度矩阵。
  - 如果是可调用对象，则输入两个样本，输出其相似度。
- `gamma`：一个浮点数，它给出了 `rbf`, `poly`, `sigmoid`, `laplacian`, `chi2` 核的系数。  
如果 `affinity='nearest_neighbors'`，则忽略该参数。
- `degree`：一个整数，当使用多项式核时，指定多项式的度。
- `coef0`：一个整数，当使用多项式核和 `sigmoid` 核时，指定偏置。
- `kernel_params`：一个字典，当 `affinity` 是可调用对象时，传给该可调用对象的关键词参数。
- `n_neighbors`：一个整数，指定当计算相似度矩阵时，考虑样本周围近邻的多少个样本。  
如果 `affinity='rbf'`，则忽略该参数。
- `eigen_tol`：一个浮点数，当使用 `arpack` 求解器求解特征值时，指定收敛阈值。
- `assign_labels`：一个字符串，指定二次聚类的算法。
  - `'kmeans'`：使用 `k-means` 算法执行二次聚类。
  - `'discretize'`：使用 `discretize` 执行二次聚类。

- `n_jobs` : 指定并行度。

## 2. 属性:

- `affinity_matrix` : 一个形状为 `(n_samples,n_samples)` 的数组, 给出了相似度矩阵。
- `labels_` : 一个形状为 `(n_samples,)` 的数组, 给出了每个样本的簇标记。

## 3. 方法:

- `fit(X[,y])` : 训练模型。
- `fit_predict(X[, y])` : 训练模型并执行聚类, 返回每个样本所属的簇标记。