第9讲 聚类算法

- 聚类概念
- *k*Means
- GMM
- 聚类评估

聚类概念

- **聚类(Clustering)**：样本集分成不同“簇”，使簇与簇之间的区别尽可能大，簇内样本差异尽可能小，簇中所有样本的均值称为簇的质心(Centroids)

输入无标签样本集D，输出D的聚类结果：

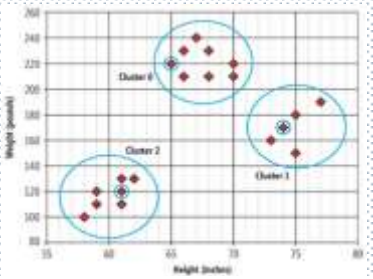
$$C = \{C_1, C_2, \ldots, C_k\}，$$

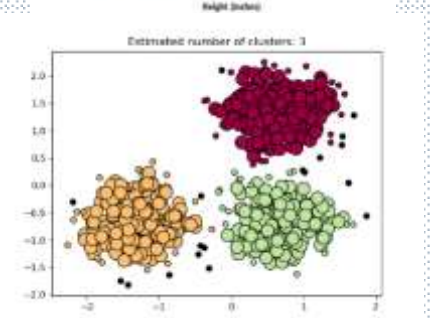
满足条件：

$$C_1 \cup C_2 \cup \ldots \cup C_k = D$$

$$C_i \cap C_j = \phi,\ i \neq j$$

C中的$C_1, C_2, \ldots, C_k$称簇，每个簇可通过一些特征描述





聚类概念

聚类要素

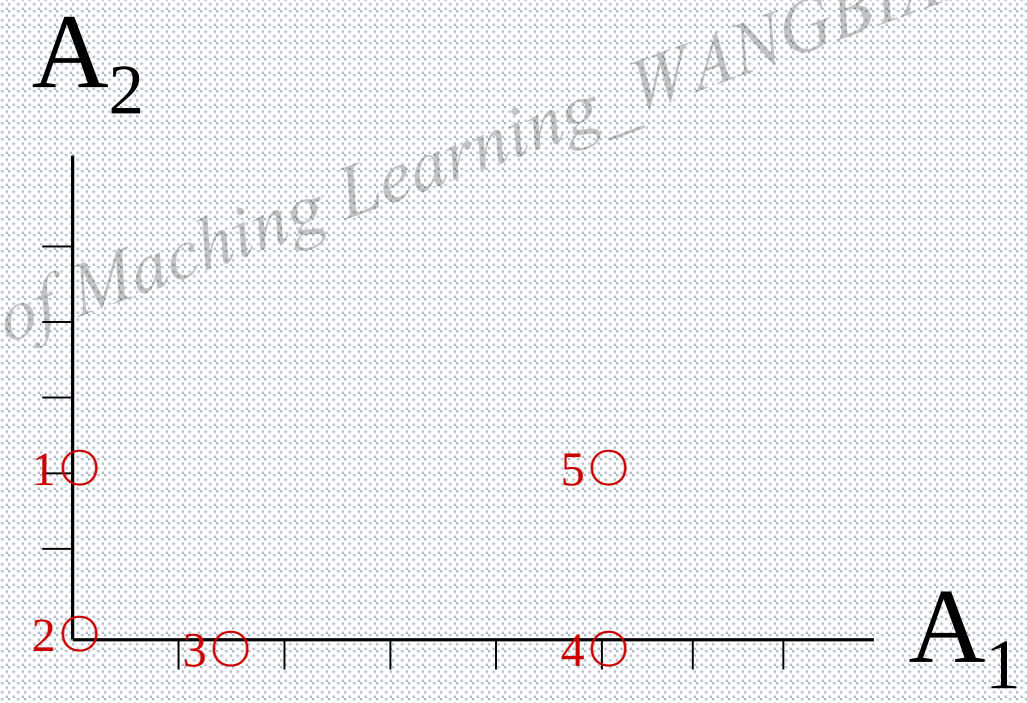
- 相似度度量：样本划分的依据，常用距离度量
- 目标函数：算法停止的判别条件。不同阶段得到不同划分结果
- 划分策略(算法)：使划分结果达到目标函数的方法

k-Means

- **示例1：***k*-Means

样本有二维属性

O	A_1	A_2
1	0	2
2	0	0
3	1.5	0
4	5	0
5	5	2



k-Means

- **示例1：*k*-Means**
  - 设簇数**k=2**, 初始簇中心： $M_1 = O_1 = (0,2)$ $M_2 = O_2 = (0,0)$
  - 对样本进行聚类
  - ✓ 计算O_3的距离：

$$d(M_1, O_3) = \sqrt{(0-1.5)^2 + (2-0)^2} = 2.5$$

$$d(M_2, O_3) = \sqrt{(0-1.5)^2 + (0-0)^2} = 1.5$$

$$d(M_2, O_3) \le d(M_1, O_3)$$

  - ✓ 计算O_4的距离

$$d(M_2, O_4) = \sqrt{(0-5)^2 + (0-0)^2} = 5$$

$$d(M_1, O_4) = \sqrt{(0-5)^2 + (2-0)^2} = \sqrt{29}$$

$$d(M_2, O_4) \le d(M_1, O_4)$$

k-Means

- ✓ 计算O_5的距离：

$$d(M_1,O_5)=\sqrt{(0-5)^2+(2-2)^2}=5$$

$$d(M_2,O_5)=\sqrt{(0-5)^2+(0-2)^2}=\sqrt{29}$$

$$d(M_1,O_5)\le d(M_2,O_5)$$

更新，得到新簇

$C_1=\{O_1,O_5\}$ $C_2=\{O_2,O_3,O_4\}$

单个簇距离平方和

$$E_1=\left[(0-0)^2+(2-2)^2\right]+\left[(0-5)^2+(2-2)^2\right]=25 \qquad E_2=27.25$$

总距离平方和：$E=E_1+E_2=25+27.25=52.25$

k-Means

- 计算新簇中心

$M_1 = ((0+5)/2, (2+2)/2) = (2.5, 2)$

$M_2 = ((0+1.5+5)/3, (0+0+0)/3) = (2.17, 0)$

$M_1 = O_1 = (0,2)$

$M_2 = O_2 = (0,0)$

重复之前步骤，得到O_1分配给C_1；O_2分配给C_2，O_3分配给C_2，O_4分配给C_2，O_5分配给C_1。更新，得到新簇$C_1 = \{O_1, O_5\}$ $C_2 = \{O_2, O_3, O_4\}$

新中心为：$M_1 = (2.5, 2)$ $M_2 = (2.17, 0)$

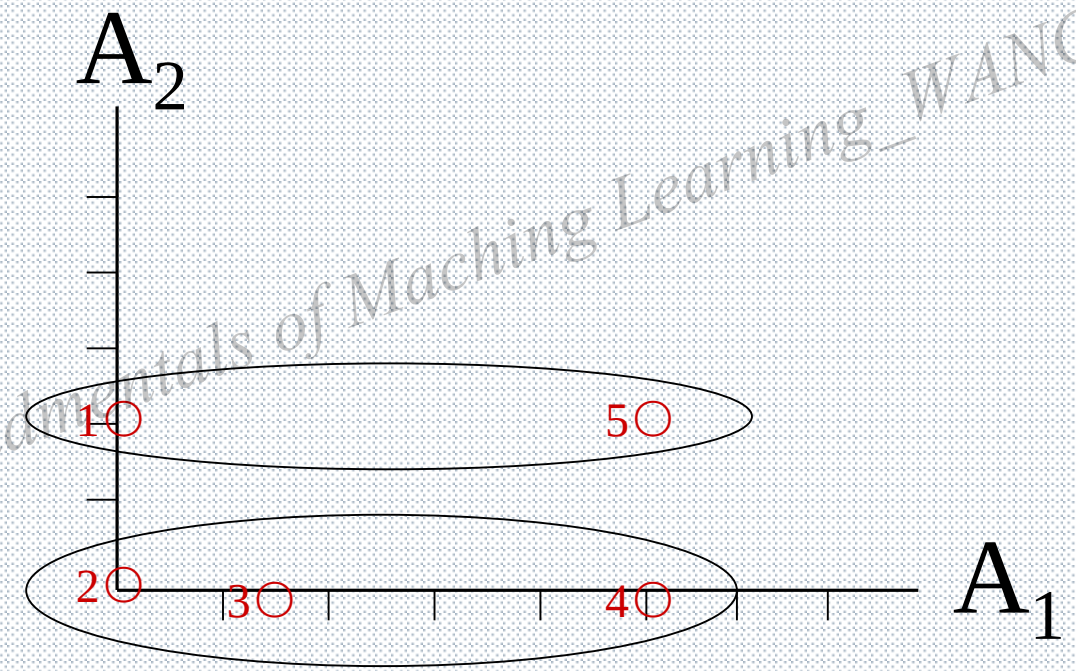
单簇距离平方和：

$E_1 = [(0-2.5)^2 + (2-2)^2] + [(2.5-5)^2 + (2-2)^2] = 12.5$ $E_2 = 13.15$

总距离平方和：$E = E_1 + E_2 = 12.5 + 13.15 = 25.65$

k-Means

- 如果迭代收敛或符合停止条件，输出结果



结果可视化

k-Means

- **思想原理**：指定簇数k，聚类结果由k 个簇中心表达（各个簇内的所有样本均值）
  - 通过迭代把样本集划分为不同簇，每次迭代过程都是向目标函数值减小的方向进行，直至其不发生变化(阈值)或达到迭代次数
  - 相似度量：距离度量
  - 目标函数：假设样本集D包含k个簇$C_1, C_2, \ldots, C_k$，簇的质心分别为$m_1, m_2, \ldots, m_k$，距离平方和:

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - m_i\|^2$$

k-Means

- **kMeans算法IOP描述：**

输入：簇数目k, 包含n个样本的数据集D

输出：k个簇, 使距离平方和最小

处理：

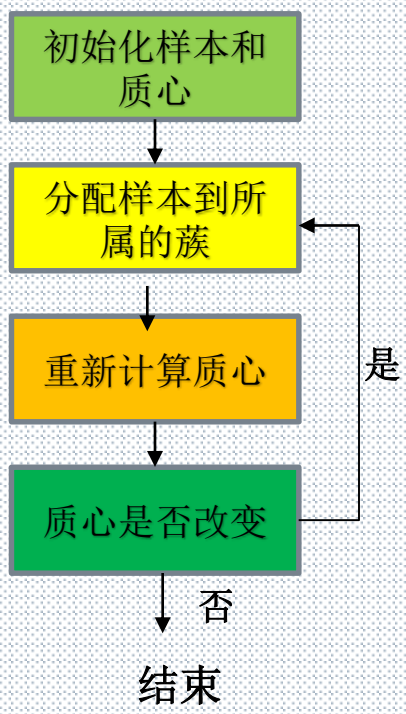
（1）从D中随机选择k个对象作为初始簇形心；

（2） repeat

（3）通过与质心距离度量，将每个样本分配到最近的簇；

（4）新簇质心，即重新计算每个簇中样本的均值；

（5）until不再发生变化或达迭代次数。



k-Means

k-Means性能

- 简单、快速，适合大数据集处理
- 时间复杂度：$O(tkmn)$，其中，t为迭代次数，k为簇数目，m为样本数，n为维数
- 空间复杂度：$O((m+k)n)$，其中，k为簇数目，m为样本数，n为维数
- 当结果簇密集，且簇与簇之间区别明显时，效果较好

k-Means

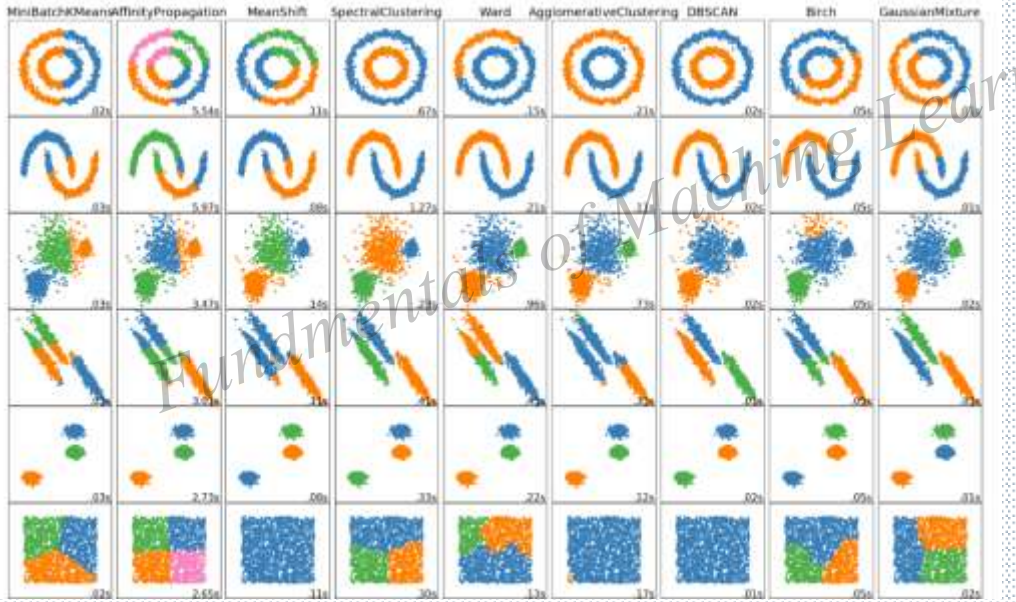
k-Means特点

- 在簇的平均值被定义的情况下才能使用，对于非数值属性的数据不适用。
- 给出*k*，而且对初值敏感，对不同初始值，可能导致不同结果，容易陷入局部最优。
- 对于“噪声”和孤立点数据敏感，少量的该类数据能够对平均值产生极大影响。
- 具有不同密度、非球形族、复杂形状族的数据点聚类效果差

k-Means

- **sklearn.cluster**

  https://scikit-learn.org/stable/modules/clustering.html#clustering



A comparison of the clustering algorithms in scikit-learn

k-Means

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

  https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans

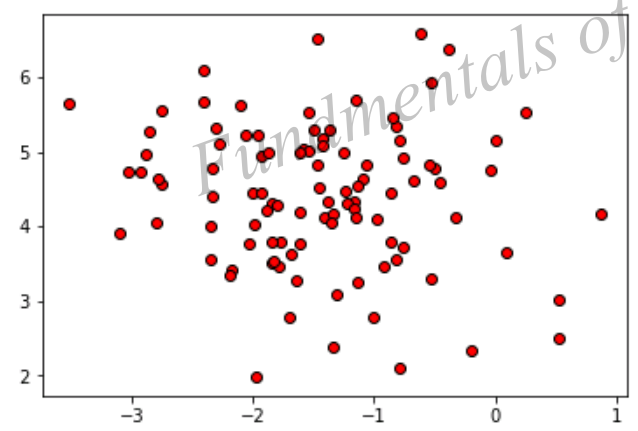
  parmeters: n_clusters, init
  Attributes: cluster_, centers_, labels_, inertia_, n_iter_
  Methods: fit(X[, y, sample_weight]), fit_predict(X[, y, sample_weight]), fit_transform(X[, y, sample_weight]), predict(X[, sample_weight]), score(X[, y, sample_weight]), transform(X)

k-Means

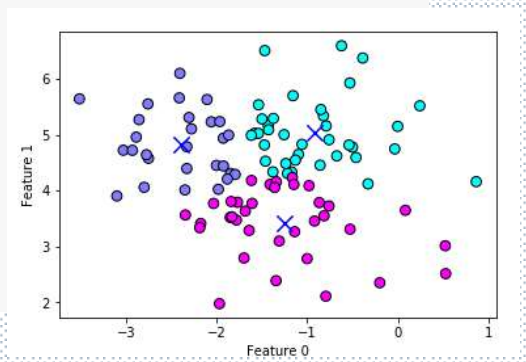
- **示例：sklearn.cluster.KMeans**

```
# 导入数据集生成工具
from sklearn.datasets import make_blobs
# 生成分类数为1的数据集
blobs = make_blobs(random_state=1,centers=1)
X_blobs = blobs[0]
# 绘制散点图
plt.scatter(X_blobs[:,0],X_blobs[:,1],c='r',edgecolor='k')
```

```
<matplotlib.collections.PathCollection at 0x9fa4b00>
```



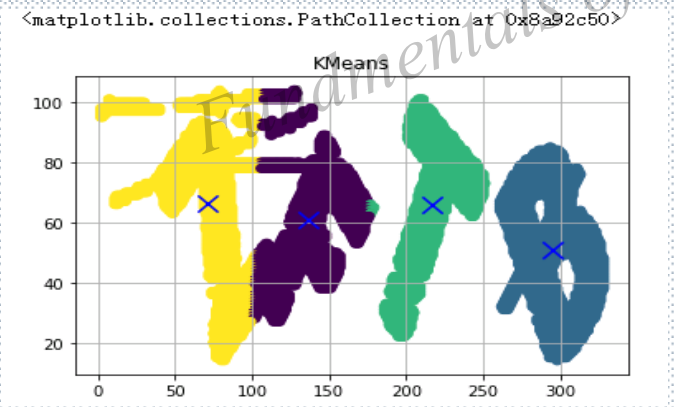

```
# 导入kMeans工具
from sklearn.cluster import KMeans
# k = 3
means = KMeans(n_clusters=3)
# 拟合数据
lusters = kmeans.fit_predict(X_blobs)
# 可视化结果
lt.scatter(X_blobs[:, 0], X_blobs[:, 1], c=clusters, cmap=plt.cm.cool,
           s=60,edgecolor='k')
lt.xlabel("Feature 0")
lt.ylabel("Feature 1")

# 用蓝色叉号代表聚类的中心
entroids = kmeans.cluster_centers_
lt.scatter(centroids[:, 0], centroids[:, 1],
           marker='x', s=150, linewidths=3,
           color='b', zorder=10)
```



k-Means

- **示例**：对简单验证码进行分割

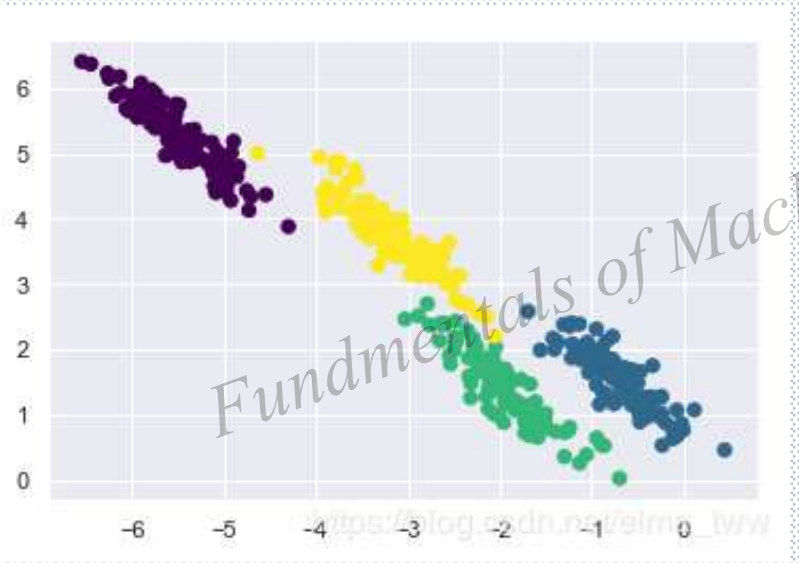
验证码有4个字符，即有4个簇



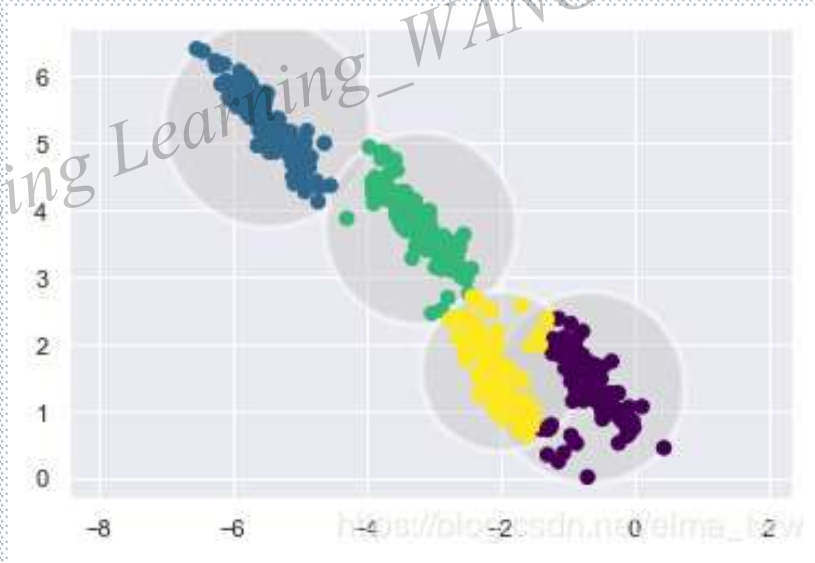

```
# 验证码分割识别
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# 导入PIL的Image类
from PIL import Image
# 打开图像
im = np.array(Image.open(r'zftb.jpg'))
# 获得图像3个维度
h,w,san = im.shape
X = [(h-x,y) for x in range(h) for y in range(w) if im[x][y][2] < 200]
# X被转换成数组
X = np.array(X)

# 导入kMeans工具
from sklearn.cluster import KMeans
n_clusters = 4
k_means = KMeans(init ='k-means++', n_clusters=4)
y_pred = k_means.fit_predict(X)

plt.scatter(X[:, 1], X[:, 0], c=y_pred)
plt.title('KMeans')
plt.grid(True)
```

k-Means

- ***k*-Means特点**



原始数据



kMeans聚类结果

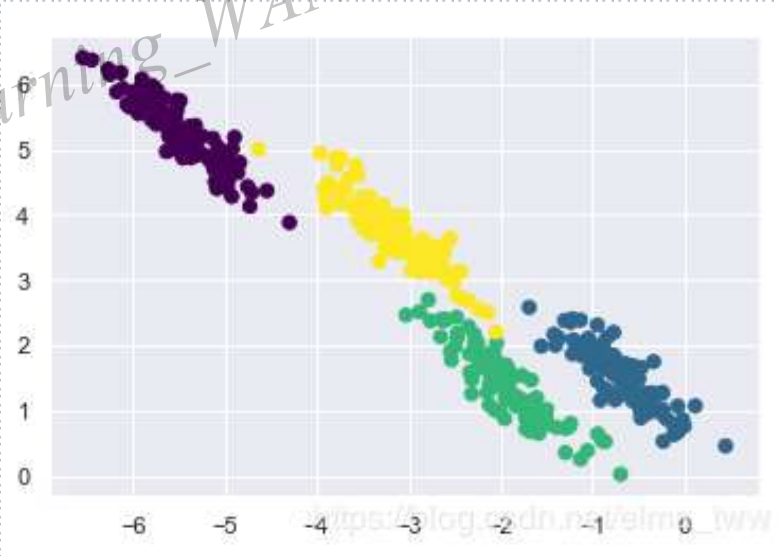
GMM

- 假设给定一个样本集

$$D = \{x_1, x_2, x_3, \ldots, x_N\},$$

由k个未知模型产生

- 通过样本集分别估计k个概率模型参数θ_k $(k=1, 2, 3\ldots.)$，使k个模型产生样本集的概率最大



GMM

- 问题：不知道x来自哪一个分布，只能对每一个观测数据都引入隐藏变量

$$\gamma_{jk}=\begin{cases}1 & \text{数据}x_j\text{来自第}k\text{个分布} \\ 0 & \text{数据}x_j\text{不是来自第}k\text{个分布}\end{cases} \qquad \begin{matrix}k=1,2,3.... \\ j=1,2,3...\end{matrix}$$

此时，观察值不再x_j

而是：$(x_j, r_{j1}, r_{j2}, \ldots, r_{jk})$, $j=1,2,3,\ldots,N$

GMM

- **EM(Expectation Maximization)算法**

  由 Dempster，Laind，Rubin 于 1977 年提出

  用于含有隐变量(latent variable)的概率模型参数的极大似然估计

  任何含隐变量的模型都可归纳为数据残缺问题

  EM算法是实际应用中解决数据残缺问题的一种方法

GMM

- EM算法
  - 不完整(或残缺)数据的估计
    例如，高斯混合模型参数的估计；
    隐马尔科夫模型参数的估计
  - 概率模型聚类
    如果估计的参数可以表明类别，则聚类问题
    也可以归结为参数估计问题

GMM

- **EM算法：**
  - 基本思想：首先根据已经给出的观测数据，估计出模型参数的值；然后再依据上一步估计出的参数值估计缺失数据的值，再根据估计出的缺失数据加上之前已经观测到的数据重新再对参数值进行估计，然后反复迭代，直至最后收敛，迭代结束。
  - 每次迭代包含两步：

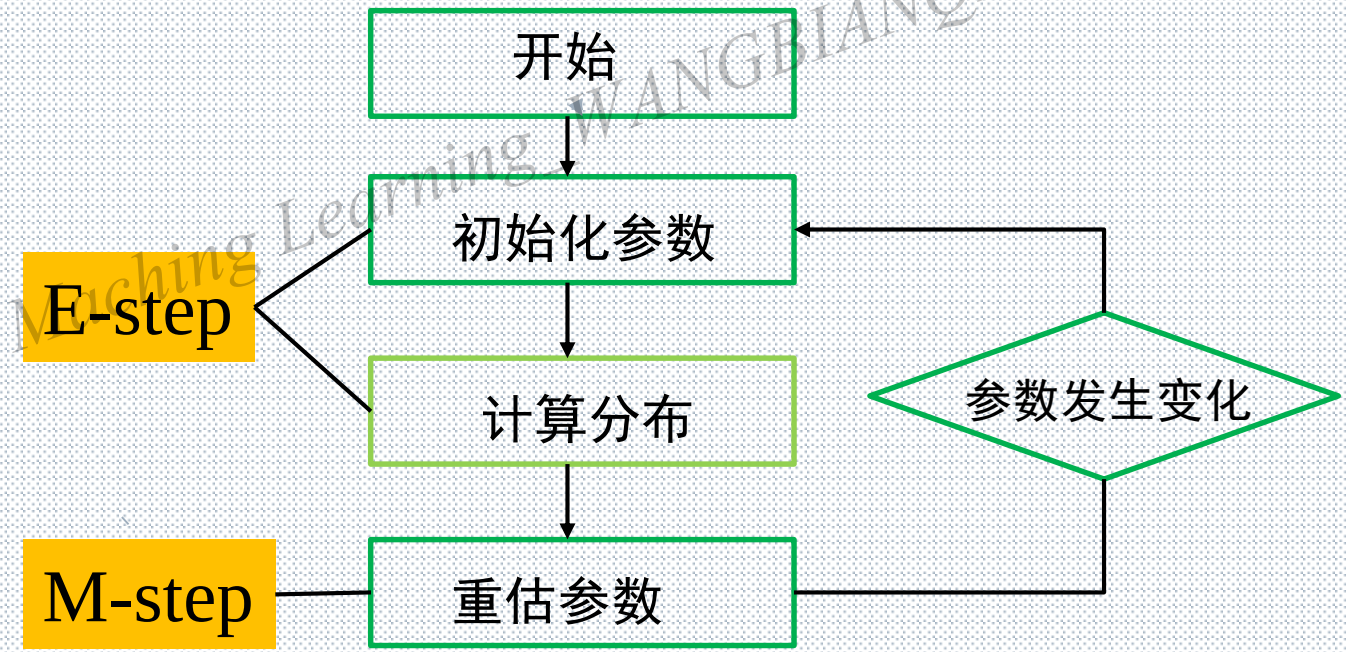
    E步，求期望值(expectation)

    M步，求极大(maximization)

GMM

- **EM算法：**

已知：观测数据
分布模型
未知：
每一个样本属于哪一个分布
模型参数



GMM

- 高斯混合模型（Gaussian mixture model, GMM）

被定义为k个高斯密度函数的线性组合：

$$P(x)=\sum_{i=1}^{k} w_i N_i(x;\mu_i,\sigma_i)$$

其中 $N_i(x;u_i,\sigma_i)$为均值 μ_i，标准差为 σ_i 的高斯分布，

w_i是混合参数，看成第i个高斯分布的权重，代表先验概率，且

$$\sum_{i=1}^{k} w_i = 1,\quad 0 \le w_i \le 1$$

GMM

- GMM参数估计-EM算法

第i个分模型 $N_i(x;\mu_i,\sigma_i)$ 的概率密度函数：

$$N_i(x)=\frac{1}{\sqrt{2\pi}\sigma_i}\exp\left\{-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right\}$$

设有n个样本，由k高斯混合分布产生，且各分布之间相互独立

利用EM算法估计高斯混合分布参数：

即，确定每个高斯分布的均值、方差(协方差)、先验概率

GMM

GMM参数估计-EM算法

输入：观测数据$x_1, x_2, ..., x_N$

输出：高斯混合模型参数

处理：

（1）取参数的初始值开始迭代

（2）E步：依据当前模型参数，计算分模型k对观测数据的响应度，即隐参数期望

（3）M步：计算新一轮的模型参数

（4）重复第(2)步和第(3)步，直到收敛

$$\hat{\gamma}_{jk} = \frac{a_k P(x_j \mid \theta_k)}{\sum_{k=1}^{K} a_k P(x_j \mid \theta_k)}, \quad j = 1, 2, ..., N; \; k = 1, 2, ..., K$$

$$\hat{\mu}_k = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk} x_j}{\sum_{j=1}^{N} \hat{\gamma}_{jk}}, \quad k = 1, 2, ..., K$$

$$\hat{\sigma}_k^2 = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk} (x_j - \mu_k)^2}{\sum_{j=1}^{N} \hat{\gamma}_{jk}}, \quad k = 1, 2, ..., K$$

$$\hat{a}_k = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk}}{N}, \quad k = 1, 2, ..., K$$

参考《统计学习方法》

GMM

- **示例2**：随机抽取10位同学测量身高，得到数据H。假设身高概率服从高斯分布，分别估算男生、女生身高的均值和方差(单位：CM)

H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
181	178	173	167	158	166	175	170	169	180

- 隐藏变量：性别G
- 观察数据：（H, (0, 1)）和（H, (1, 0)）
- 求每条个数据属于男生身高和属于女生身高的期望
- 似然函数：观察数据的联合概率表达式

GMM

- ✓ **E-step: ω1=ω2=0.5, μ1=177, μ2=160, $\alpha_1^2=\alpha_2^2$=总体方差**

$E(Z_m)$	0.98458	0.95927	0.81704	0.37782	0.02955	0.30336	0.89674	0.62218	0.54147	0.97863
$E(Z_f)$	0.01542	0.04073	0.18296	0.62218	0.97045	0.69664	0.10326	0.37782	0.45853	0.02137

- ✓ **M-step:**

ω	μ	α^2
0.65106	174.91237	29.56356
0.34894	165.70619	62.20912

GMM

- **E-step:** 利用上一步计算得到的参数

$E(Z_m)$	0.90455	0.88587	0.79599	0.48758	0.03342	0.41411	0.84420	0.67607	0.62051	0.90026
$E(Z_f)$	0.09545	0.11413	0.20401	0.51242	0.96658	0.58589	0.15580	0.32393	0.37949	0.09974

- **M-step:**

ω	μ	α^2
0.65626	174.34400	27.30195
0.34374	166.65222	44.41165

结束条件：参数不再变化或指定的迭代次数

GMM

- GMM特点
  - 对初值敏感，需要多次调整
  - 可能获得局部最优解，需要多次调整
  - 对孤立点敏感，有噪声时效果差
  - GMM比kMeans计算复杂，收敛也较慢，不适于大规模数据集和高维数据，但比kMeans结果稳定、准确
  - 两者都需要已知样本聚类数目

GMM

- **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*

  https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html?highlight=gmm

- Parmeters: n_components, covariance_type
- Attributes: weights_, means_, covariances_, precisions_, precisions_cholesky_, converged_, n_iter_, lower_bound_
- Methods

GMM

- **Methods**

aic(self, X)	Akaike information criterion for the current model on the input X.
bic(self, X)	Bayesian information criterion for the current model on the input X.
fit(self, X[, y])	Estimate model parameters with the EM algorithm.
fit_predict(self, X[, y])	Estimate model parameters using X and predict the labels for X.
predict(self, X)	Predict the labels for the data samples in X using trained model.
predict_proba(self, X)	Predict posterior probability of each component given the data.
sample(self[, n_samples])	Generate random samples from the fitted Gaussian distribution.
score(self, X[, y])	Compute the per-sample average log-likelihood of the given data X.
score_samples(self, X)	Compute the weighted log probabilities for each sample.

聚类评估

- 聚类模型评估：同簇样本尽可能相似，不同簇样本尽可能差异，即聚类结果“簇内相似度”(intra-cluster similarity)高，而“簇间相似度”(inter-cluster similarity)低
  - 有真值标签评估聚类
    调整兰德指数(adjusted rand index, ARI)，归一化互信息(normalized mutual information, NMI)等
  - 无真值标签评估聚类
    轮廓系数(silhouette coefficient)

聚类评估

- 轮廓系数(silhouette coefficient)：根据样本i的簇内不相似度a_i和簇间不相似度b_i，定义样本i的轮廓系数$s(i)$：

$$s(i)=\frac{b(i)-a(i)}{\max\{a(i),b(i)\}} \qquad s(i)=\begin{cases}1-\dfrac{a(i)}{b(i)}, & a(i)<b(i)\\ 0, & a(i)=b(i)\\ \dfrac{b(i)}{a(i)}-1, & a(i)>b(i)\end{cases}$$

其中a_i为样本i到同簇其它样本的平均距离，b_i为样本i到其它簇C_j所有样本的平均距离的最小值

  - 对于一个样本集合，它的轮廓系数是所有样本轮廓系数的平均值

聚类评估

- 示例：sklearn.metrics.cluster.silhouette_score

```
# 利用轮廓系数评估
from sklearn.metrics.cluster import silhouette_score
km_ss = silhouette_score(X_blobs,km_clusters)
db_ss = silhouette_score(X_blobs,db_clusters)
print('kmeans的轮廓系数',km_ss )
print('DBSCAN的轮廓系数',db_ss )
```

```
kmeans的轮廓系数 0.332790901261
DBSCAN的轮廓系数 0.362304371149
```