同济大学
TONGJI UNIVERSITY
王睿智 ruizhiwang@tongji.edu.cn

人工智能技术与应用

4.1 降维

4.2 聚类

4.3.1 k-means算法

聚类性能度量

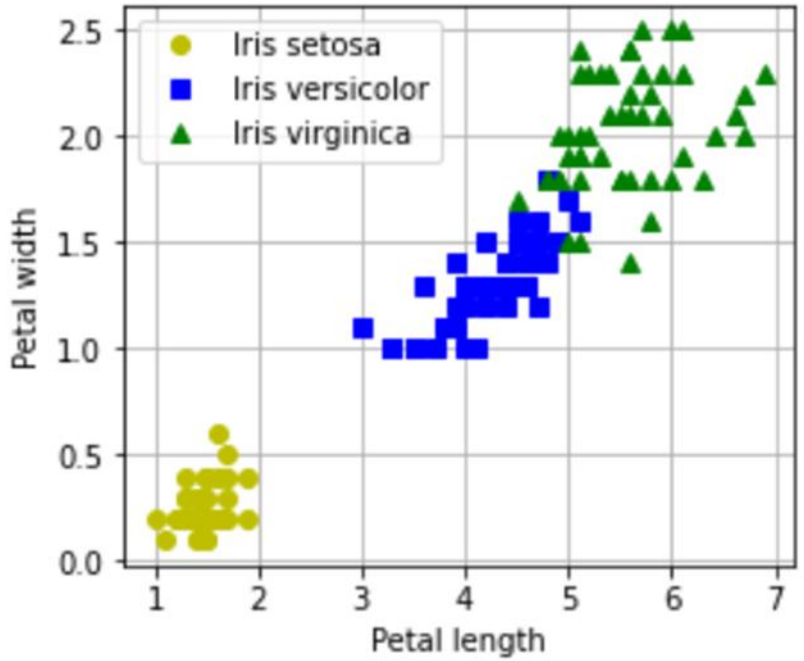
k均值的局限性

4.3.2 凝聚聚类算法

引例 分类与聚类

有两个数据集，如下图所示。 **要求将每个样本归入各自的某一组中。**

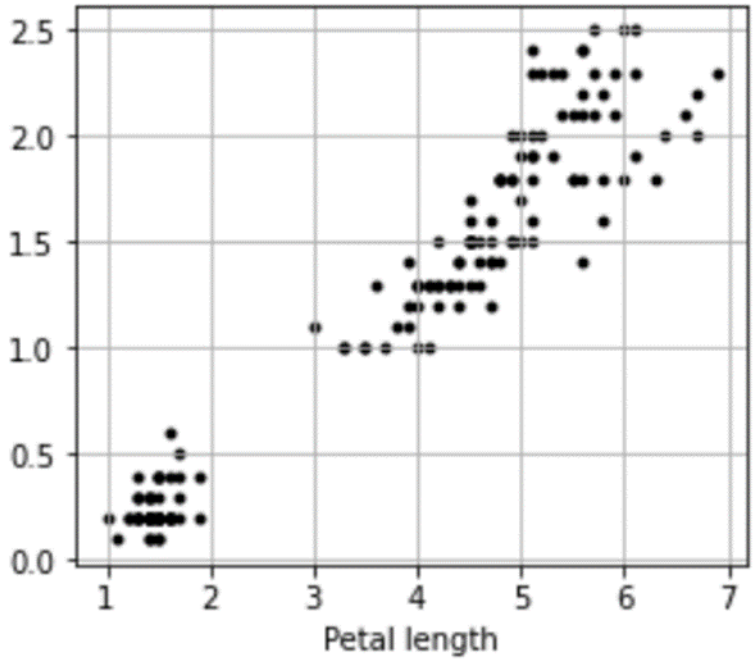
右边：数据集带类标签，是有监督的任务。
可用分类方法。



带类标签的iris数据

左边：数据集不带类标签，是无监督的任务。
用什么方法呢?

将相似的实例划到相应的组



不带类标签的iris数据

将相似的实例划到相应的组（簇，cluster），这样将数据集划分成不同的簇，每簇包含相似的实例。这类方法称为**聚类（clustering）**。

聚类是无监督学习中研究最多、应用最广的方法。常应用于

- **客户分组**：根据客户的购买习惯和网站的浏览行为，将客户划分成不同组。以便理解客户的需求，进而有针对性地进行市场营销。
- **降维**：对数据集聚类，然后样本的特征向量x可用样本与各个簇的邻近程度来替代。
- **异常检测**：与所有簇都不接近的样本很可能是异常点。异常检测多用于制造业中的次品检测，欺诈检测。
- **半监督学习**：若只有少量的标签数据，可以先聚类，然后将标签传播给同一簇中的所有样本。为监督学习算法提供更多的标签，以改善性能。
- **图像分割**：根据颜色对图像象素聚类，然后用簇颜色均值替代簇中各个象素的颜色。图像分割可用于目标检测和跟踪系统，它使得检测各个对象的轮廓更容易。

聚类算法有很多，大致可分为：

- **原型聚类**，也称为“基于原型的聚类”。此类算法假设聚类结构能通过一组原型（如簇中心）刻画。算法先对原型进行初始化，再对原型进行迭代更新求解。
  - 典型代表：k 均值算法
- **分层聚类**，是一类基于相同原则将簇组织成层次树的聚类算法。

  主要有两种：凝聚（agglomerative）分层聚类和分裂（divisive）分层聚类。
  - 分裂聚类：从一个包含所有样本的簇开始，然后每次逐步将簇分裂成更小的簇，直到每个簇只包含一个样本为止。
  - 凝聚聚类：将每个样本作为一个单独的簇，然后合并距离最近的一对簇，直到只剩下一个簇为止。
- **密度聚类**，也称“基于密度的聚类”，给样本的密度区域分配聚类标签。
  - 典型代表：DBSCAN（Density-Based Spatial Clustering of Applications with Noise）

  在DBSCAN中，密度被定义为指定半径为ε的球范围内包含的数据样本数量。

4.2.1 k均值算法

K均值 (k-means) 是最常用的聚类算法。属于基于原型的聚类算法。对离散数据，原型是簇中心；连续数据，原型是簇质心（均值）。

k均值算法分4步：

1. 从所有样本中随机选择k个样本作为初始簇的中心；
2. 将每个样本分配给最近的簇中心；
3. 更新簇中心，新簇中心为已分配样本的平均值（假设样本特征值是连续的）；
4. 重复步骤2和步骤3，直到簇中心不再变动为止，或者迭代次数得到用户设定的最大值。算法结束。

簇划分

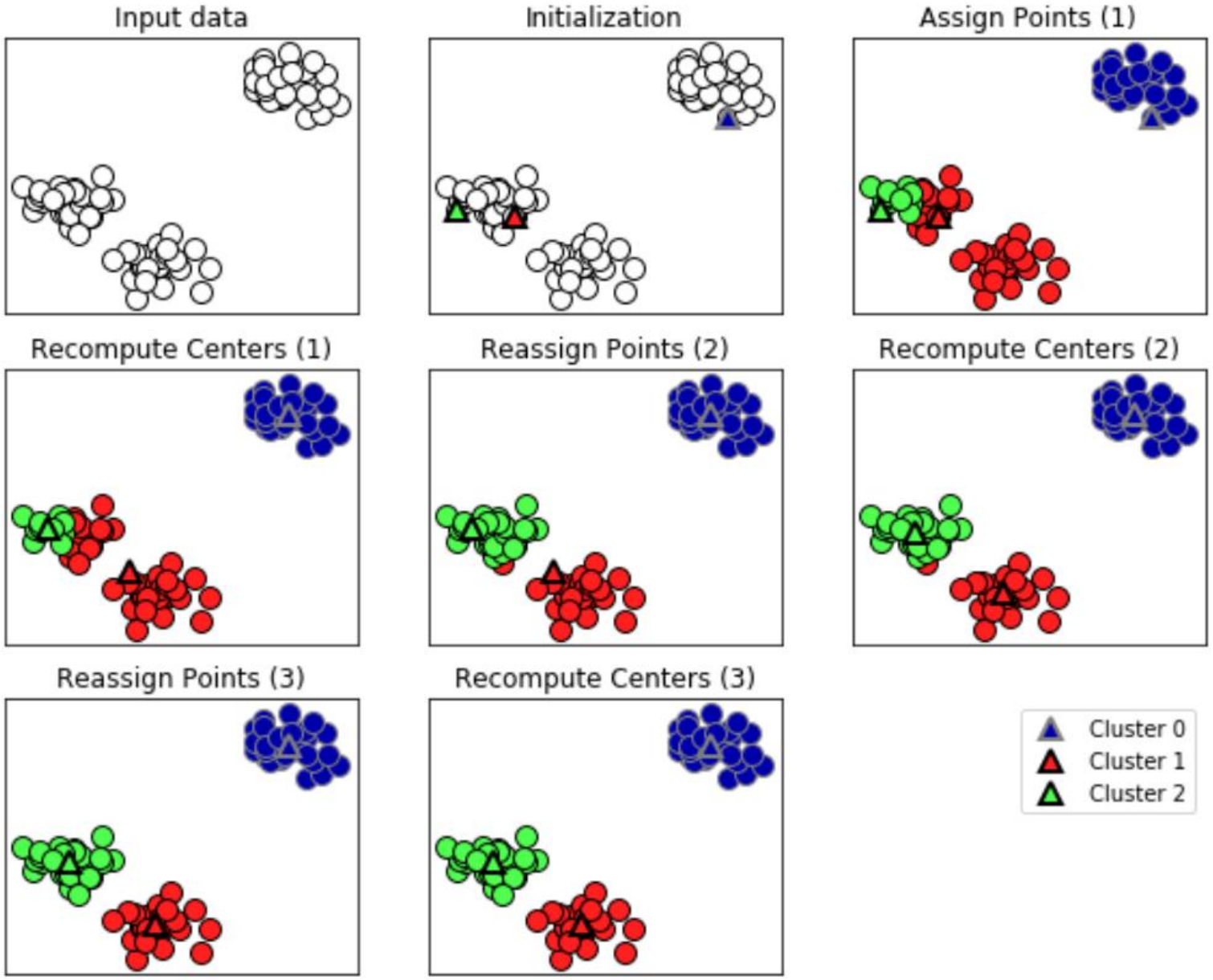
更新簇中心

k均值的2个关键点：

① k；

② 初始簇中心；

4.2.1 k均值算法 | *k* 均值聚类过程



- **两样本间的相似性**可定义为两样本之间距离的倒数。常用欧氏距离。
- **k均值算法可理解为优化问题：最小化簇内误差平方和**（Sum of Squared Error，SSE）。

$$\mathrm{SSE} = \sum_{i=1}^{m}\sum_{j=1}^{k} w^{(i,j)} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)} \right\|_2^2$$

其中，i和j分别代表样本索引和簇索引；

m是样本数目，k是簇数目；

$\boldsymbol{\mu}^{(j)}$为簇j的中心；

如果样本$\boldsymbol{x}^{(i)}$在簇j内，则$w^{(i,j)} = 1$，否则$w^{(i,j)} = 0$。

- 误差平方和也被称为**簇惯性**（cluster inertia, 也称模型惯性）。

Sklearn.cluster中提供了KMeans类：

```
KMeans(n_clusters=8, init='k-means++', max_iter=300,
       n_init=10,random_state=None,…)
```

参数	• ***n_clusters***: 簇的个数。int型，默认值8。 • ***init***：簇中心初始化方法，可选'*k-means++*','*random*'或一个*数组*，默认为'*k-means++*'。 '*random*': 从数据中随机选择k个观测值（行）作为初始中心； '*k-means++*': 首先随机选取第一个初始簇中心。若已选了n个初始簇中心(0<n<K)，则在选第n+1个中心时，距离当前n个中心越远的点有越高的概率被选为第n+1个中心。 *一个数组*: 其形状应为（n_clusters，n_features），它给出初始中心。 • ***n_init***：初始化簇中心的次数，用不同的随机簇中心多次运行算法，选最优（模型惯性最小）。默认值10。
方法	• **fit(X)** 计算k均值聚类，返回拟合的估计器。 • **predict(X)** 预测X中每个样本所属的最近簇，返回簇索引（标签）。 • **fit_predict(X)** 计算聚类中心并预测每个样本所属的簇，返回簇索引（标签）。 • **score(X)** 返回负惯性。
属性	• **labels_** 每个样本点的标签，即被分入簇的索引。 • **cluster_centers_** 簇中心坐标，[n_clusters, n_features] • **inertia_** float, 模型惯性

性能指标

等价于fit(X)后predict(X)

例4.6 鸢尾花k均值聚类

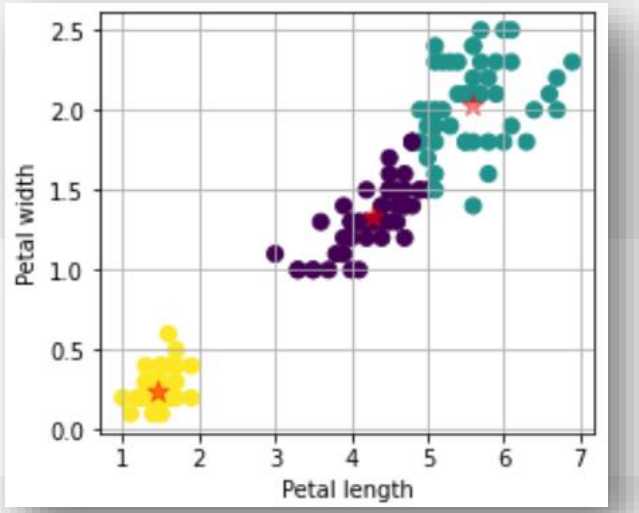
对iris的特征矩阵(不带类标签)，取其中petal length和petal width两个特征，构成新数据集X。在该数据集上做k=3的k均值聚类，然后可视化聚类结果。

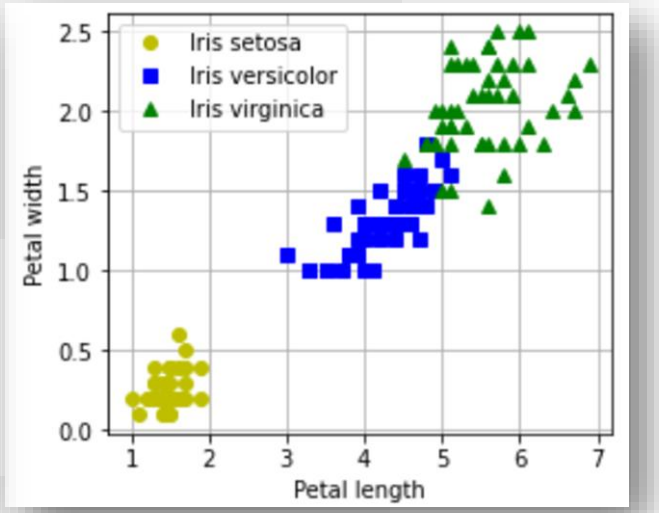
```
# 1.准备数据
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data[:,2:]   # 只取petal length和 petal width两个特征
y = iris.target
print(iris.feature_names[2:])
```

```
# 2. k均值模型训练、聚类
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=3, n_init='auto')
kmeans.fit(X)
y_kmeans=kmeans.predict(X) #预测X中每个样本所属的簇
```

```
# 3. 聚类结果可视化，画出散点图
plt.figure(figsize=(4, 3.5))
plt.scatter(X[:,0],X[:,1],c=y_kmeans,s=50,cmap='viridis')
centers=kmeans.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='r',marker='*',alpha=0.5)

plt.ylabel("Petal width")
plt.xlabel("Petal length")
plt.grid()
```






```
# 4.输出模型惯性
print(f"模型惯性为：{kmeans.inertia_},模型得分为：{kmeans.score(X)}")
```

模型惯性为：31.371358974358976,模型得分为：-31.371358974358976

1.k如何确定?

2.初始簇中心对聚类结果有何影响?

K均值聚类一定会收敛，但其收敛结果受簇中心初始值的影响。

例4.7 簇中心初始值对k均值结果的影响

生成由4个高斯分布簇构成的数据集，

簇中心坐标：(0.2,2.3),(-1.8,1.8),(-1.8,2.8), (-1.8,1.3)。

4个高斯分布的标准差分别为：0.4, 0.1, 0.1, 0.1。

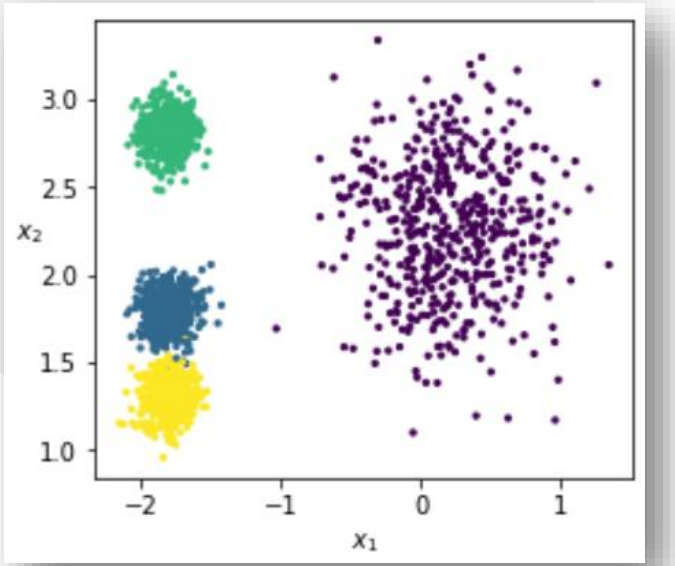
分别用随机初始化、k-means++初始化簇中心，比较并可视化聚类结果。

1.生成数据集

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import numpy as np
import matplotlib.pyplot as plt

blob_centers = np.array([[ 0.2, 2.3], [-1.8, 1.8],[-1.8, 2.8], [-1.8, 1.3]])
blob_std = np.array([0.4, 0.1, 0.1, 0.1])
X, y = make_blobs(n_samples=2000, centers=blob_centers, cluster_std=blob_std,
                  random_state=7)  # y：每个样本所属簇的整数标签{0,1,2,3}
```

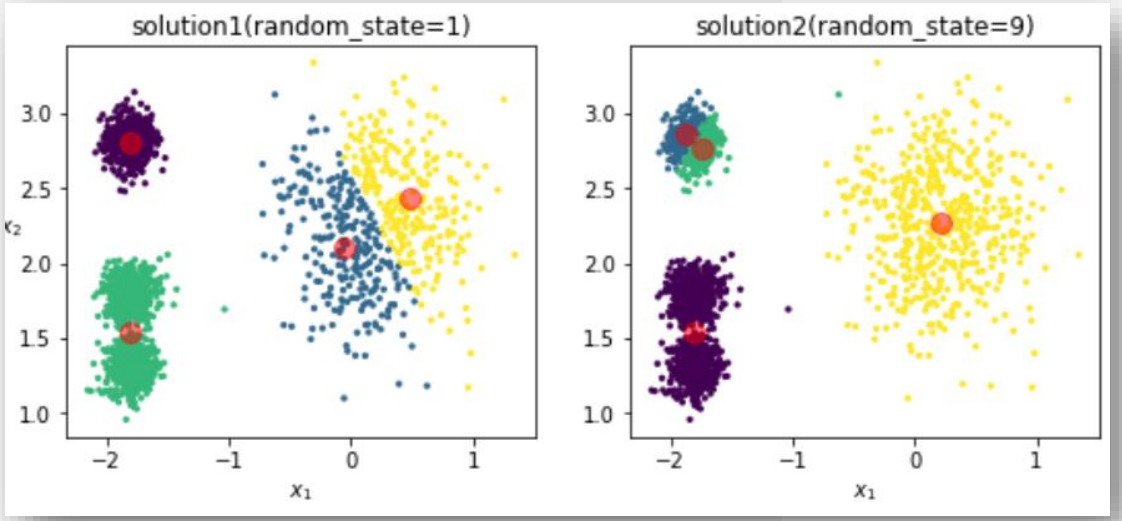
```
plt.figure(figsize=(4, 3.5))
plt.scatter(X[:, 0], X[:, 1],c=y,s=5,cmap='viridis')
plt.xlabel("$x_1$")
plt.ylabel("$x_2$", rotation=0)
plt.show()
```



make_blobs 参考：

2. 比较两种不同的随机初始化簇中心方案

```
# 随机初始化方案 1
k = 4
rnd_init1 = KMeans(n_clusters=k,init="random",n_init=1,random_state=1)
y_rnd_init1=rnd_init1.fit_predict(X)
plt.figure(figsize=(9, 3.5))
plt.subplot(121)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$", rotation=0)
plt.title('solution1(random_state=1)')
plt.scatter(X[:,0],X[:,1],s=5,c=y_rnd_init1,cmap='viridis')
centers=rnd_init1.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='r',s=100,alpha=0.5)
```



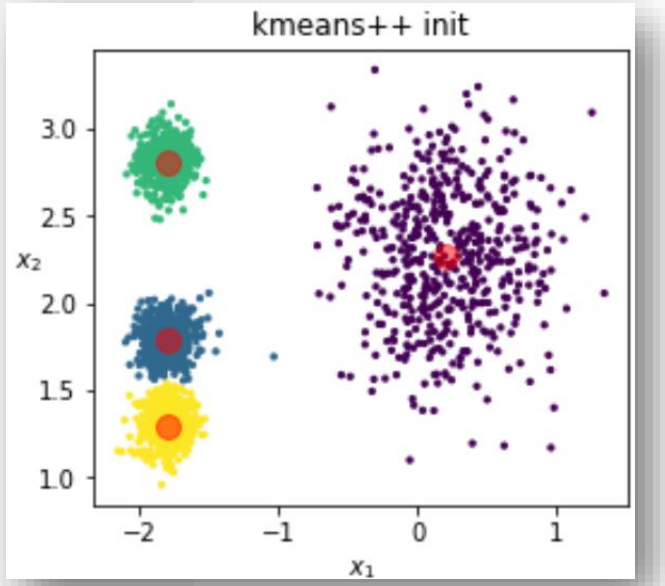

```
#随机初始化方案 2
rnd_init2 = KMeans(n_clusters=k,init="random",n_init=1,random_state=9)
y_rnd_init2=rnd_init2.fit_predict(X)
plt.subplot(122)
plt.xlabel("$x_1$")
plt.title('solution2(random_state=9)')
plt.scatter(X[:,0],X[:,1],s=5,c=y_rnd_init2,cmap='viridis')
centers=rnd_init2.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='r',s=100,alpha=0.5)
```

怎么评价两种方案得到的聚类结果？

3.采用kmeans++初始化簇中心

```
#方案3：kmeans++初始化
km = KMeans(n_clusters=k,init="k-means++", n_init=1,random_state=42)
y_pred = km.fit_predict(X)

plt.figure(figsize=(4, 3.5))
plt.xlabel("$x_1$")
plt.ylabel("$x_2$", rotation=0)
plt.title('kmeans++ init')
plt.scatter(X[:,0],X[:,1],s=5,c=y_pred,cmap='viridis')
centers=km.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='red',s=100,alpha=0.5)
plt.show()
```



怎么评价上述几种初始化中心点方案？看看各模型的惯性

4.比较随机初始化中心点和kmeans++初始化中心点

```
print(f"随机初始化方案1模型的惯性： {rnd_init1.inertia_:.2f}")
print(f"随机初始化方案2模型的惯性： {rnd_init2.inertia_:.2f}")
print(f"kmeans++模型的惯性： {km.inertia_:.2f}")
```

```
随机初始化方案1模型的惯性： 189.29
随机初始化方案2模型的惯性： 234.05
kmeans++模型的惯性： 176.19
```

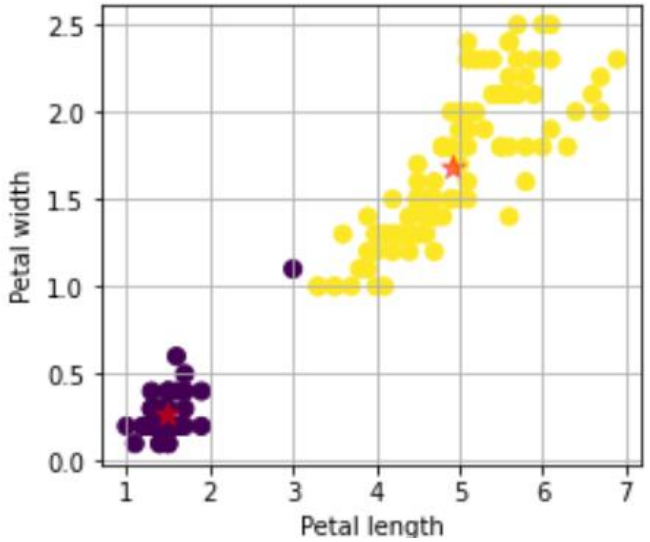
k均值++(k-means++)：让每个簇的初始中心彼此尽可能地远离，从而到达比经典k均值算法效果更好且结果一致的目的。

k均值++算法过程：

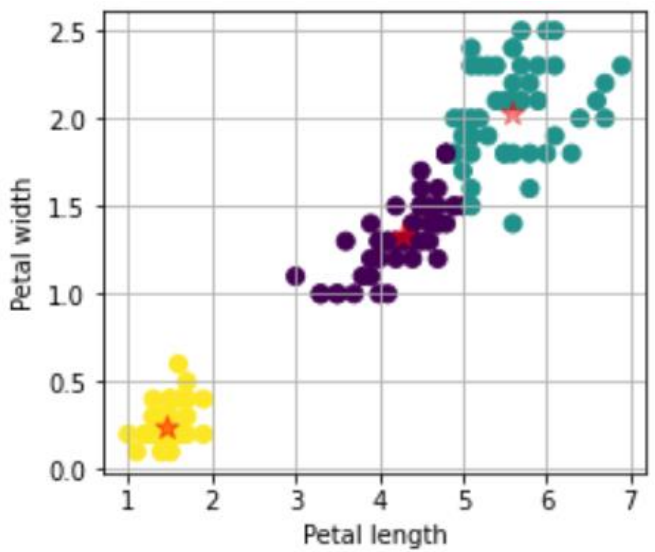
1. 初始化空集$\boldsymbol{M}$，用来存储k个簇中心；
2. 随机选择一个输入样本作为第一个簇中心$\boldsymbol{\mu}^{(1)}$并存储在$\boldsymbol{M}$中；
3. 对于不在$\boldsymbol{M}$中的每一个样本$\boldsymbol{x}^{(i)}$，计算其与M中所有簇中心距离平方最小的值，即$d(\boldsymbol{x}^{(i)}, \boldsymbol{M})^2$；
4. 根据权重概率分布随机选择下一个簇中心$\boldsymbol{\mu}^{(p)}$，其中权重概率分布为$\frac{d(\boldsymbol{x}^{(p)}, \boldsymbol{M})^2}{\sum_i d(\boldsymbol{x}^{(i)}, \boldsymbol{M})^2}$
5. 重复步骤3和步骤4，直至选出k个质心；
6. 继续运行经典的k均值算法。

4.2.1 k均值算法 | 确定最优的簇数目

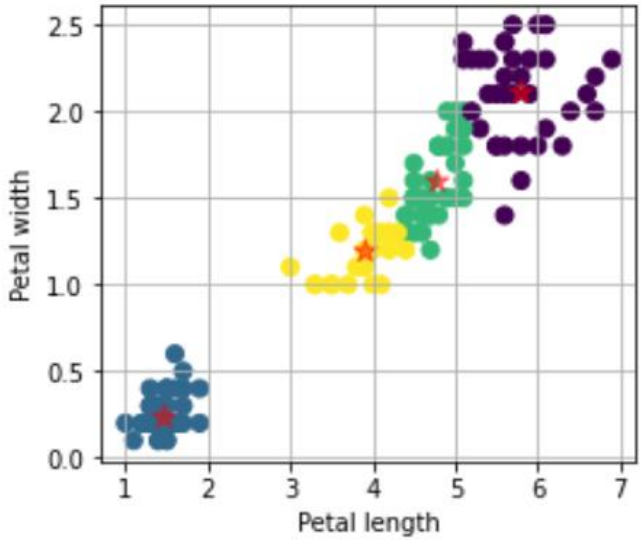
k=2



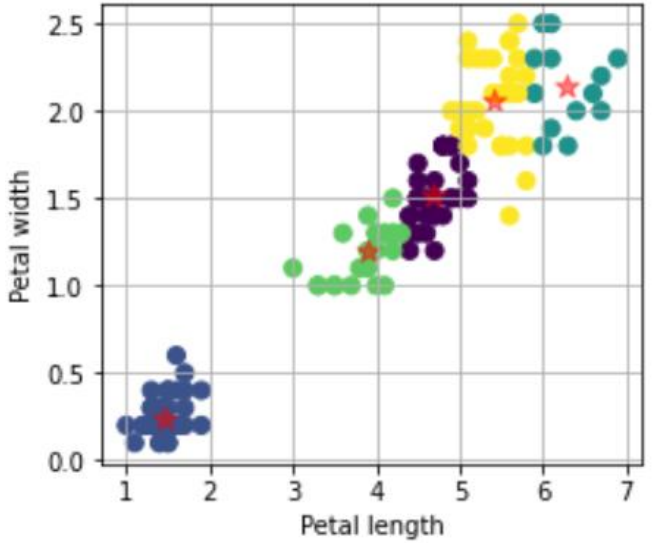
k=3



k=4



k=5



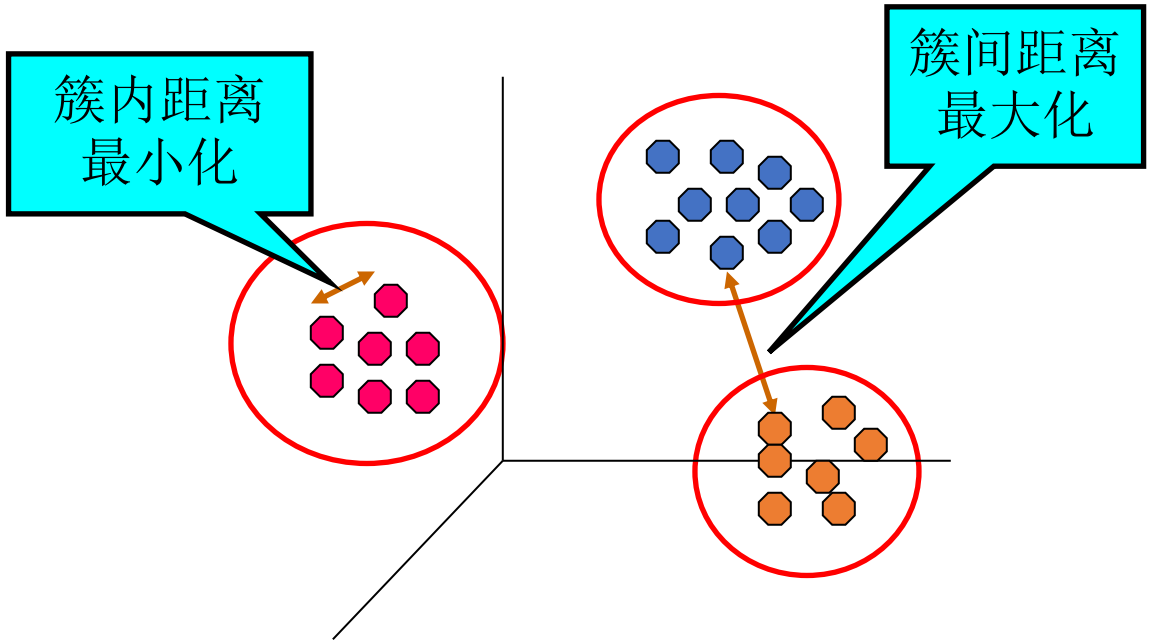
哪个聚类结果更好呢?

怎样找最优的簇数目k呢?

① 用肘方法

② 用轮廓系数

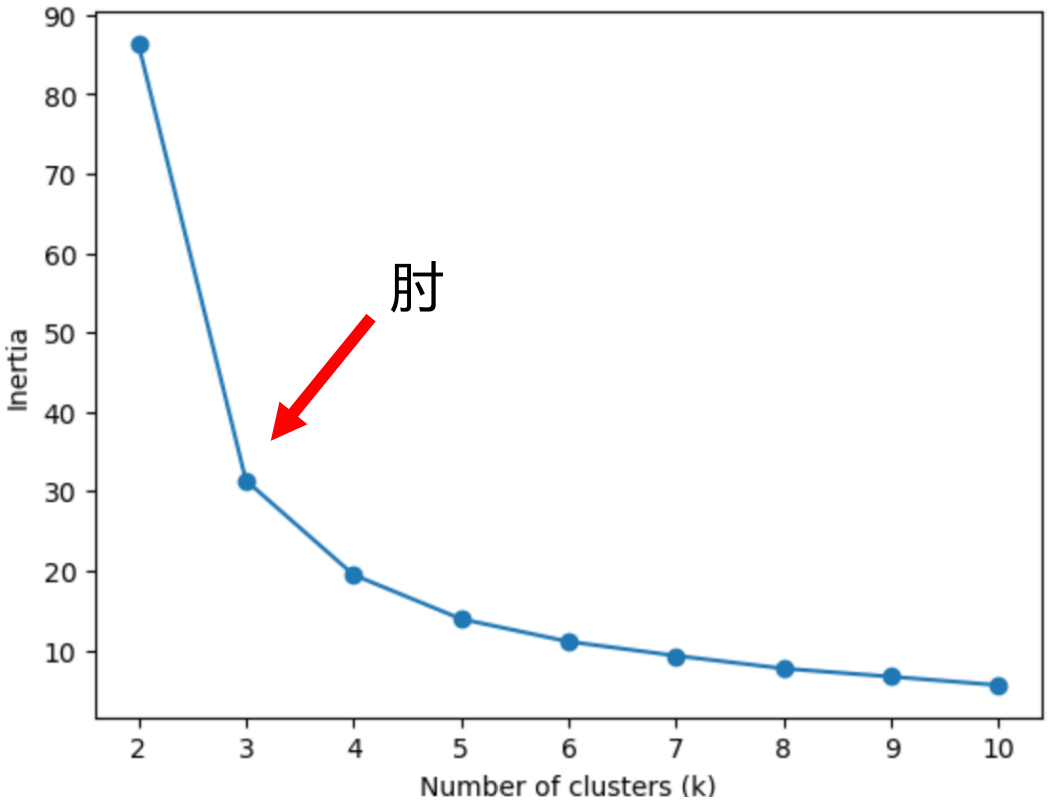
- 聚类质量度量，亦称聚类“**有效性指标**” （validity index）
- **总的原则**：“物以类聚”，即同一簇的样本尽可能彼此相似，不同簇的样本尽可能不同。即，聚类结果的“簇内相似度” 高，且“簇间相似度” 低，这样的聚类效果较好。



- 在簇数目未知情况下，常用**内部指标**（如簇内误差平方和，即**簇惯性**）来比较**不同的k**均值聚类模型的性能。
- Scikit-Learn的kMeans估计器，提供有**inertia_**属性，即簇内误差平方和。利用inertia_属性，可绘制不同的k对应的inertia_变化图，找出变换最迅速的拐点，从而确定k的值。这种方法被称为**肘方法**。

例4.6 鸢尾花k均值聚类 (续)｜用肘方法选择最优k

```
inertia_lst = []
for k in range(2,11):
    km = KMeans(n_clusters=k,
                init="k-means++",
                n_init=10,
                random_state=42)
    km.fit(X)
    inertia_lst.append(km.inertia_)
plt.plot(range(2,11),inertia_lst,marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.show()
```



肘部位于k=3处

肘方法相对粗糙，有没有更精确的方法?

轮廓系数

- 没标签的数据集，常用**轮廓系数** (Silhouette Coefficient) 度量聚类的质量。
- 单个样本的轮廓系数**计算步骤**：

1. 计算簇内凝聚度$a^{(i)}$，即样本$x^{(i)}$与簇内其他样本之间的平均距离；
2. 计算簇间分离度$b^{(i)}$，即样本$x^{(i)}$与距其最近的其他簇的所有样本之间的平均距离；
3. 计算轮廓系数$s^{(i)}$，即簇内凝聚度与簇间分离度之差除以两者中的最大值，

$$s^{(i)} = \frac{b^{(i)} - a^{(i)}}{max(b^{(i)}, a^{(i)})}$$

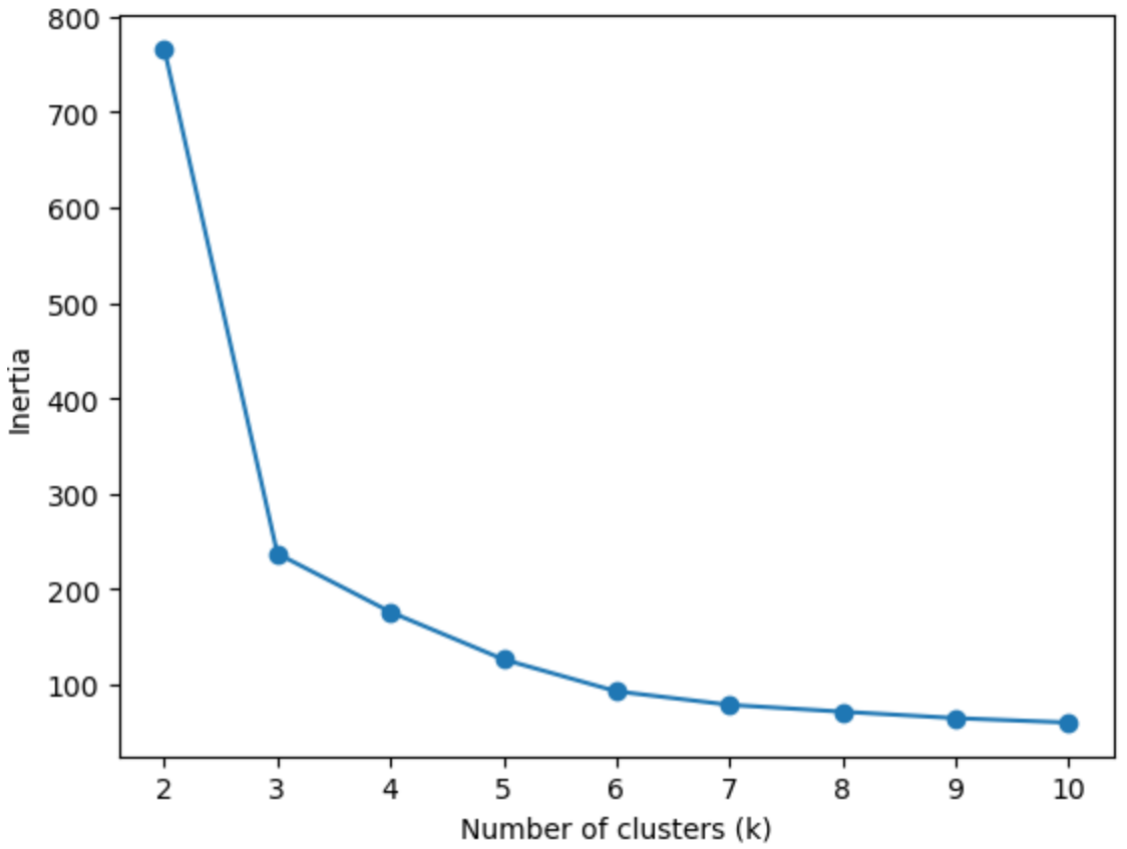
该系数取值[-1,1]，轮廓系数越大，表示聚类效果越好。$b^{(i)}$量化了一个样本与其他簇的样本的差异度，而$a^{(i)}$则表示了该样本与簇内其他样本的相似度。

- Scikit-learn的metrics类提供的silhouette_samples()即为轮廓系数。也提供了silhouette_score() 函数，来计算所有样本的平均轮廓系数，silhouette_score() 相当于numpy.mean(silhouette_samples(...))。

例4.7 簇中心初始值对k均值结果的影响(续) | 用肘方法选择最优k

5. 用肘方法

```
inertia_lst = []
for k in range(2,11):
    km = KMeans(n_clusters=k,
                init="k-means++",
                n_init=10,
                random_state=42)
    km.fit(X)
    inertia_lst.append(km.inertia_)
plt.plot(range(2,11),inertia_lst,marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.show()
```

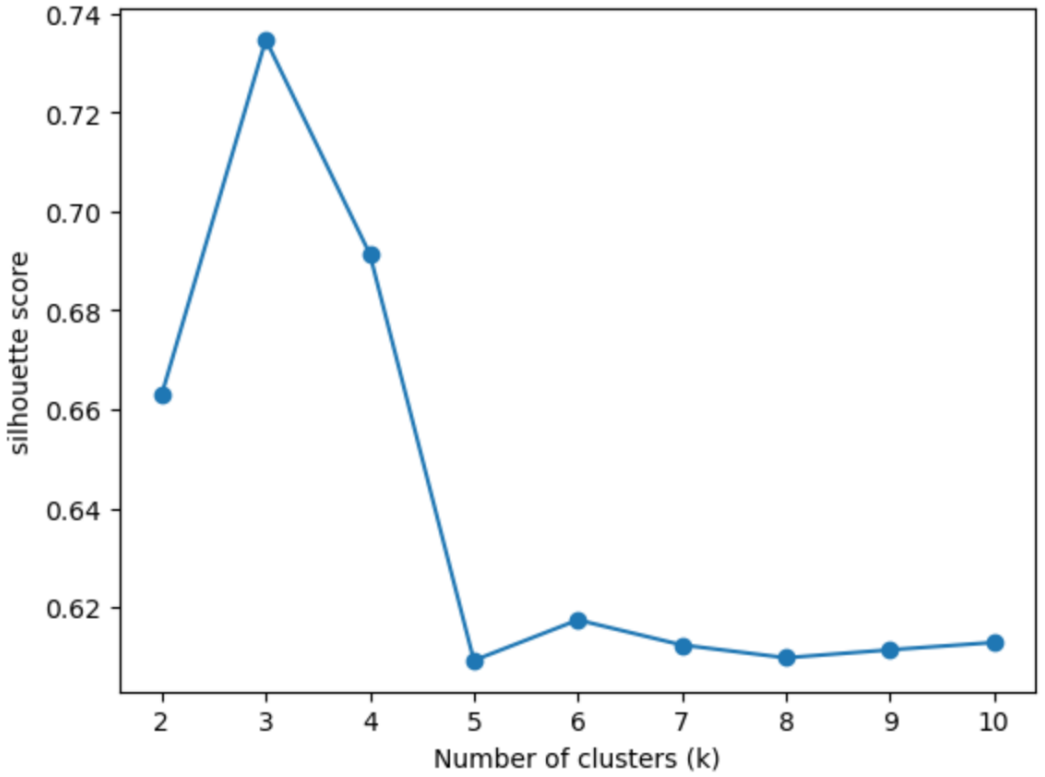


肘方法选k=3，实际数据是4个簇！！！

例4.7 续 | 用轮廓分数选择最优k

6. 用轮廓分数

```
from sklearn.metrics import silhouette_score
scores_lst = []
for k in range(2,11):
    km = KMeans(n_clusters=k,
                init="k-means++", n_init=10,
                max_iter=300,random_state=42)
    km.fit_predict(X)
    s_scores = silhouette_score(X, km.labels_)
    scores_lst.append(s_scores)
plt.plot(range(2,11),scores_lst,marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('silhouette score')
plt.show()
```



k=3，或k=4都不错

例4.8 玻璃数据聚类分析

玻璃数据集包含218个样本，有9个特征，分别对应各种氧化物的质量百分比。

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
折射率	钠的质量百分比	镁的质量百分比	铝的质量百分比	硅的质量百分比	钾的质量百分比	钙的质量百分比	钡的质量百分比	铁的质量百分比

（1）读取glass.csv文件，**kMeans聚类前，应对数据进行标准化缩放处理。**

（2）k-Means聚类分析该数据集，令k取2~10，利用肘方法选择最好的k

（3）用选定的k对该数据集进行k均值聚类，用平均轮廓系数评估聚类结果。

将预测的簇标签列与数量特征一起构成数据框，显示结果。

1.读取数据并缩放预处理

```
import pandas as pd
from sklearn.utils import shuffle
from sklearn.preprocessing import StandardScaler
```

```
df = pd.read_csv('glass.csv')
df_shuffled = shuffle(df, random_state=42)
scaler = StandardScaler() # z-score规范化
X = scaler.fit_transform(df_shuffled) # 缩放处理
```

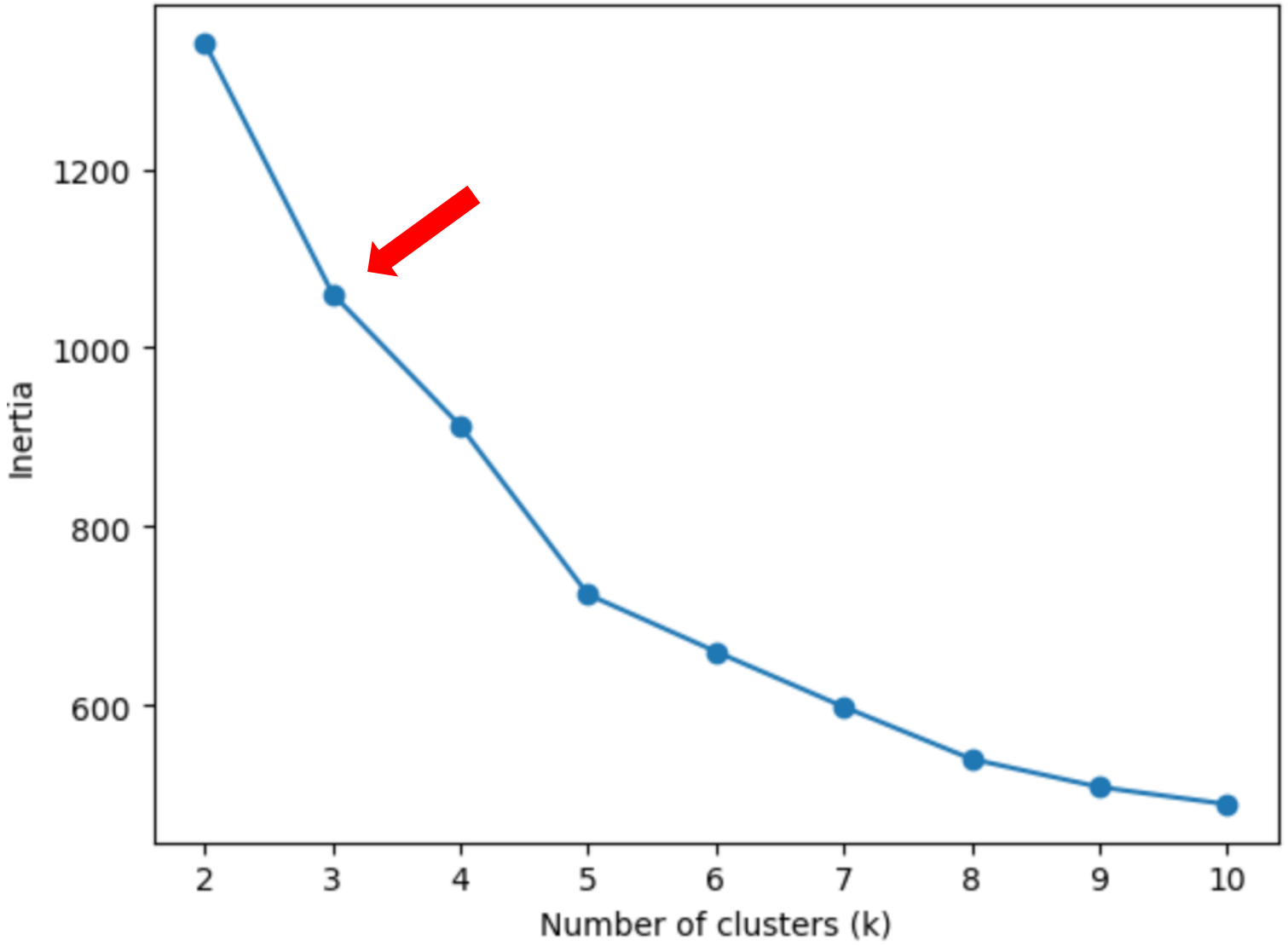
说明:

运行K均值之前，先缩放输入特征，否则簇可能变形，这样k均值的性能会差。缩放特征并不能保证所有簇都很好，但通常可以改善结果。

2.利用肘方法选簇数目

```
import warnings
warnings.filterwarnings('ignore') # 避免显示警告
```

```
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
inertia_lst = []
for k in range(2,11):
    km = KMeans(n_clusters=k,init="k-means++",n_init='auto',
                max_iter=400,random_state=42)
    km.fit(X)
    inertia_lst.append(km.inertia_)
plt.plot(range(2,11),inertia_lst,marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.show()
```



从图中可看出，k在2~3 惯性降速较快，3开始变缓，故k取3。

3. 用所选k建立k均值聚类模型

```
km = KMeans(n_clusters=3,init="k-means++",
            n_init='auto',max_iter=400,random_state=42)
y_km = km.fit_predict(X)
```

用平均轮廓系数评估结果

```
from sklearn.metrics import silhouette_score
silhouette_score(X,y_km,metric='euclidean')
```

```
0.5941200311629915
```

将预测标签与数据特征合并为dataframe

```
df_shuffled['Predicted_cluster'] = y_km
df_shuffled.head()
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Predicted_cluster
100	1.51655	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	0
215	1.51640	14.37	0.00	2.74	72.85	0.00	9.45	0.54	0.00	1
139	1.51674	12.87	3.56	1.64	73.14	0.65	7.99	0.00	0.00	0
178	1.52247	14.86	2.20	2.06	70.26	0.76	9.76	0.00	0.00	0
15	1.51761	12.81	3.54	1.23	73.24	0.58	8.39	0.00	0.00	0

例4.9聚类用于图像分割

图像分割是将图像分成多个区域的任务。

颜色分割：如果像素颜色相似，就将它们分配到同一个分割区域。

1.读入图像

```
from matplotlib.image import imread
import matplotlib.pyplot as plt

image = imread("ladybug.jpeg")
image.shape
```

```
(223, 337, 3)
```

2.颜色聚类

```
from sklearn.cluster import KMeans
X = image.reshape(-1, 3)                    # 对数组重构得到RGB颜色的长列表
kmeans = KMeans(n_clusters=8, n_init=10, random_state=42).fit(X) # 对颜色聚类
segmented_img = kmeans.cluster_centers_[kmeans.labels_] # 每个样本像素值与质心一样
segmented_img = segmented_img.reshape(image.shape)
print(f"聚类后像素最大值：{segmented_img.max()}\n像素最小值：{segmented_img.min()}")
```

```
聚类后像素最大值：244.21464540495134
像素最小值：1.976874933700923 3
```

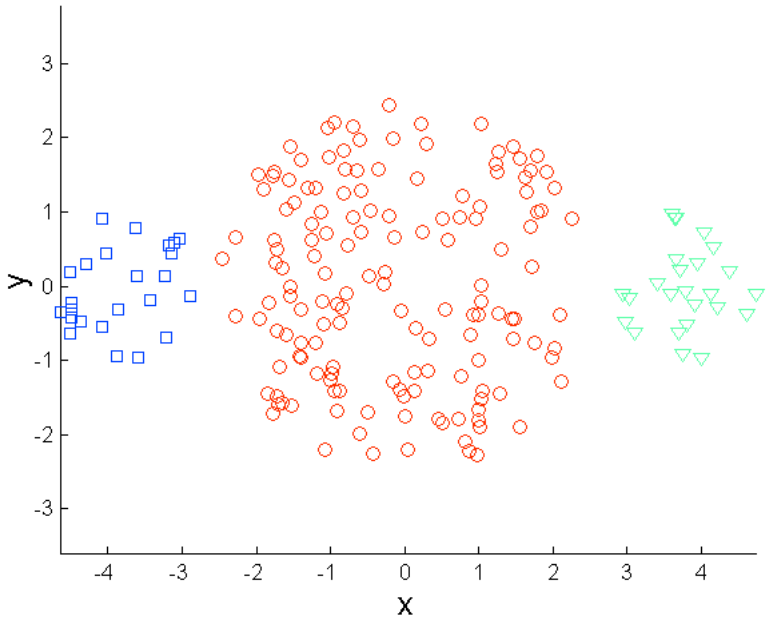
3.分割区域数分别为8、5、2

```
segmented_imgs = []
n_colors = ( 8, 5, 2)
for n_clusters in n_colors:
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=42).fit(X)
    segmented_img = kmeans.cluster_centers_[kmeans.labels_]
    segmented_imgs.append(segmented_img.reshape(image.shape))
plt.figure(figsize=(8, 6))
plt.subplots_adjust(wspace=0.05, hspace=0.1)
plt.subplot(2, 2, 1)
plt.imshow(image)
plt.title("Original image")
plt.axis('off')
for idx, n_clusters in enumerate(n_colors):
    plt.subplot(2, 2, 2 + idx)
    plt.imshow(segmented_imgs[idx] / 255)
    plt.title(f"{n_clusters} colors")
    plt.axis('off')
```

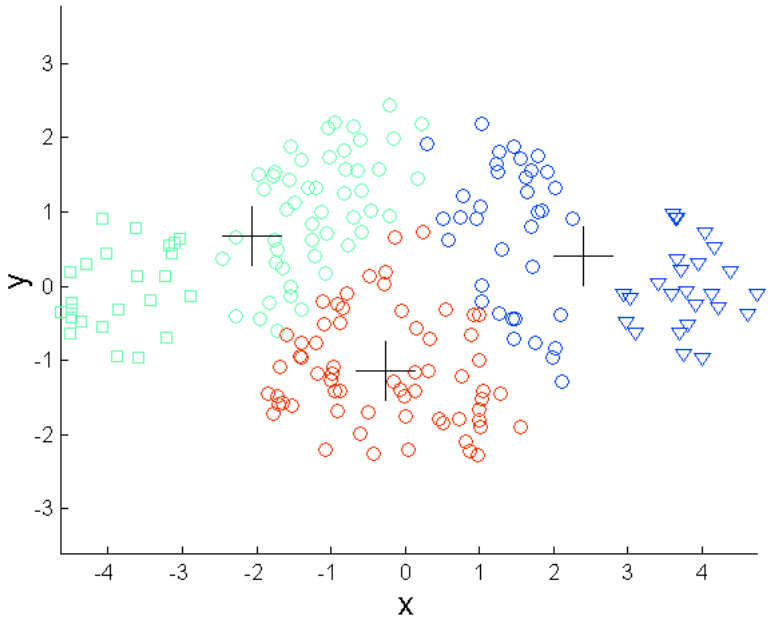
plt.imshow(X) 参数 X形状为(M, N, 3)， 对 RGB图像其值为 (0-1 float 或 0-255 int).
segmented_imgs[] 元素都是float， 故要除以255, 变成取值0-1

- k要事先指定；
- 对噪声敏感；
- 受初始中心影响；
- 簇具有不同的大小、不同的密度或非球形时，k均值效果都不好。

簇具有不同的大小

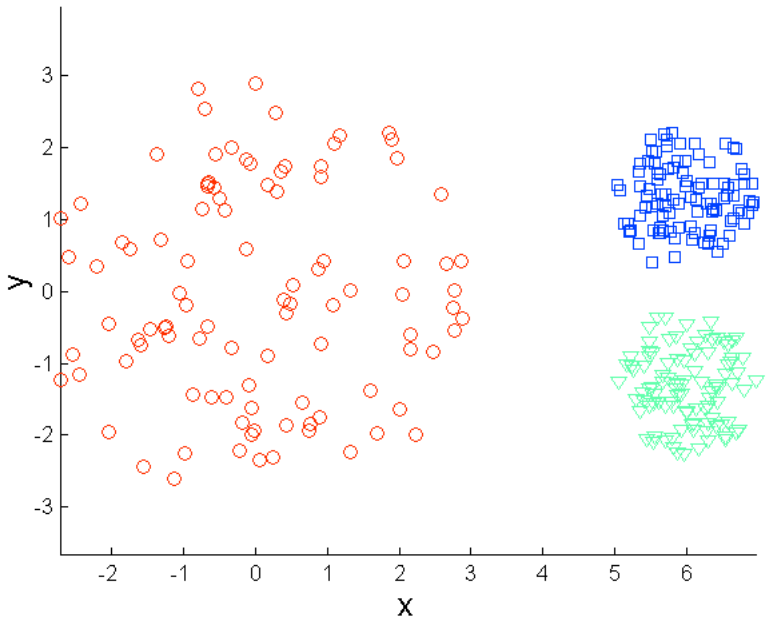


原始数据点

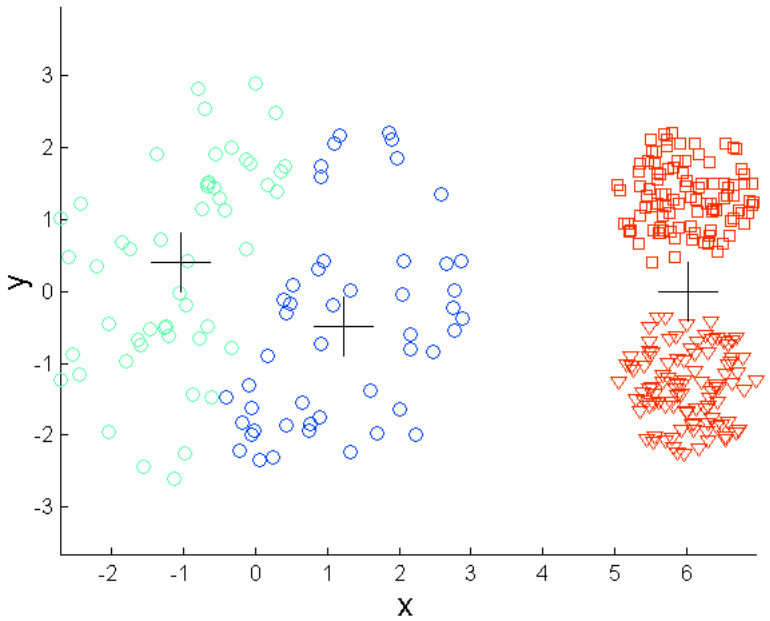


k均值 (3个簇)

簇具有不同的密度

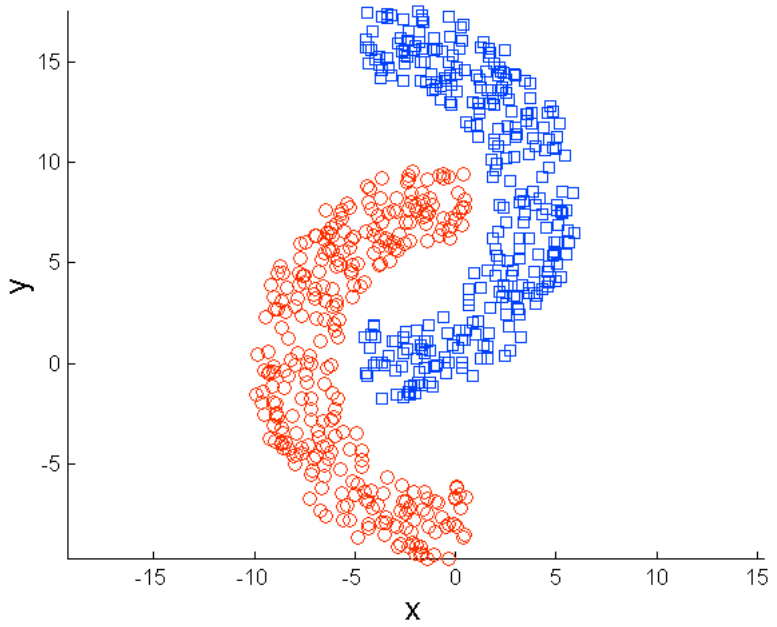


原始数据点

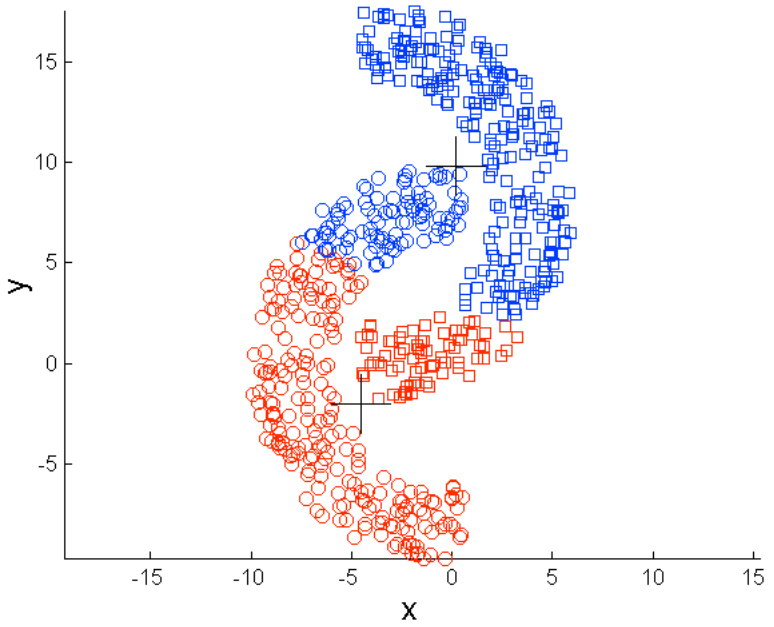


k均值 (3个簇)

簇非球形



原始数据点



k均值 (2个簇)

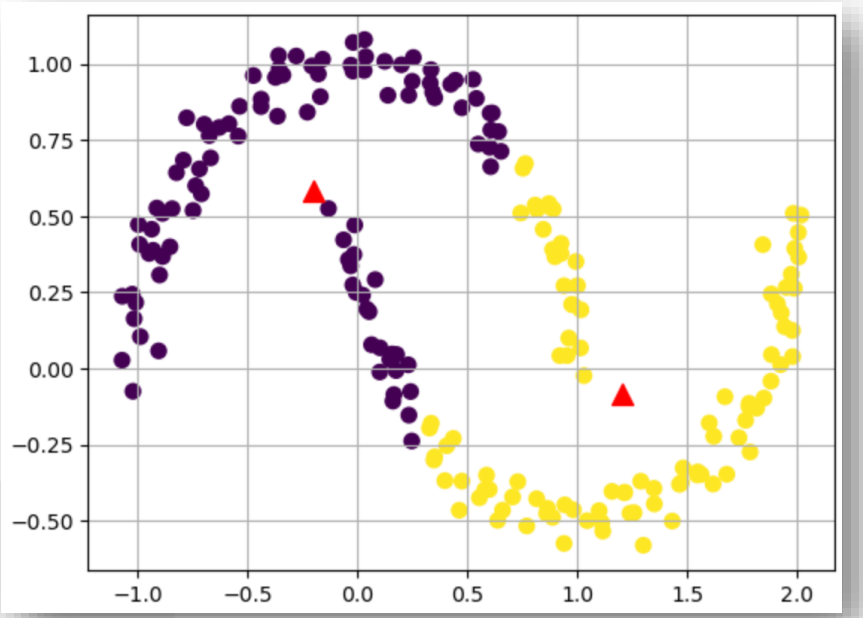
k 均值的基本假设（与其他簇的点相比，数据点更接近自己的簇中心）表明，当簇中心点呈现非线性的复杂形状时，该算法通常不起作用。

例4.10 k均值失败案例

k均值无法识别非球形簇

利用make_moons生成半月形的二维数据，共200点。
看看Kmeans聚类(k=2)找到的簇分配。

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# 生成半月数据
X,y = make_moons(n_samples=200,noise=0.05,random_state=0)
# 将数据聚类成2个簇
kmeans = KMeans(n_clusters=2,n_init = 'auto',random_state=0)
kmeans.fit(X)
y_pred = kmeans.predict(X)
# 画出簇分配和簇中心
plt.scatter(X[:,0],X[:,1],c=y_pred,cmap='viridis',s=60)
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],
            marker='^', c='red',s=100)
plt.grid()
```

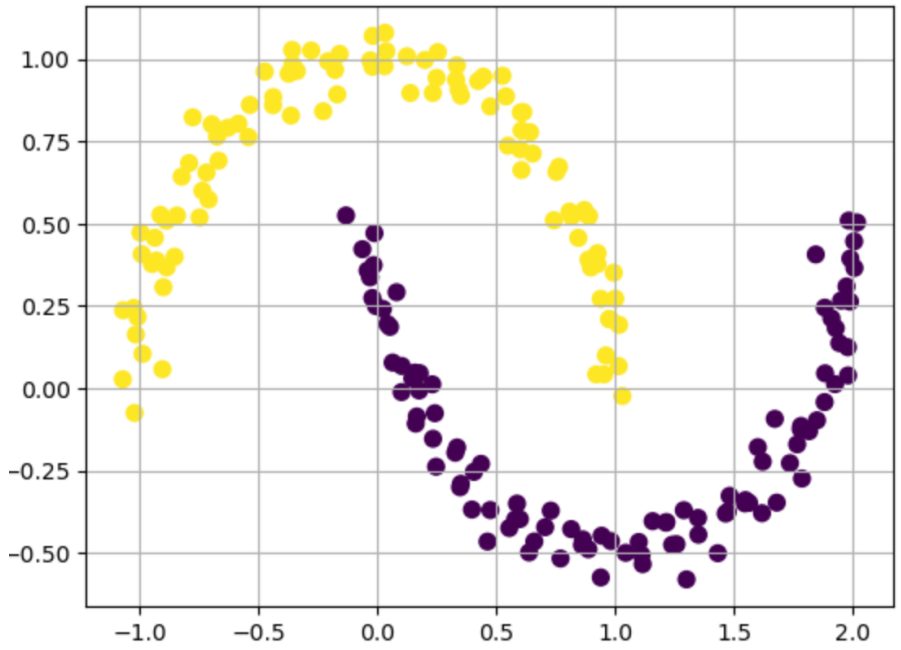


K-means聚类的边界总是线性的，这意味着当边界很复杂时，算法会失效。

- 通过核变换方法，核k-means能够找到簇之间的复杂的非线性边界。
- sklearn.cluster中的**SpectralClustering**评估器实现了核k-means算法，它使用最近邻图来计算数据的高维表示，然后用k-means算法分配标签。

```
# 对半月数据应用SpectralClustering
from sklearn.cluster import SpectralClustering

model = SpectralClustering(n_clusters = 2,
                           affinity='nearest_neighbors',
                           assign_labels = 'kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:,0],X[:,1],c = labels, s=50,cmap='viridis')
plt.grid()
```



4.2.2 凝聚聚类

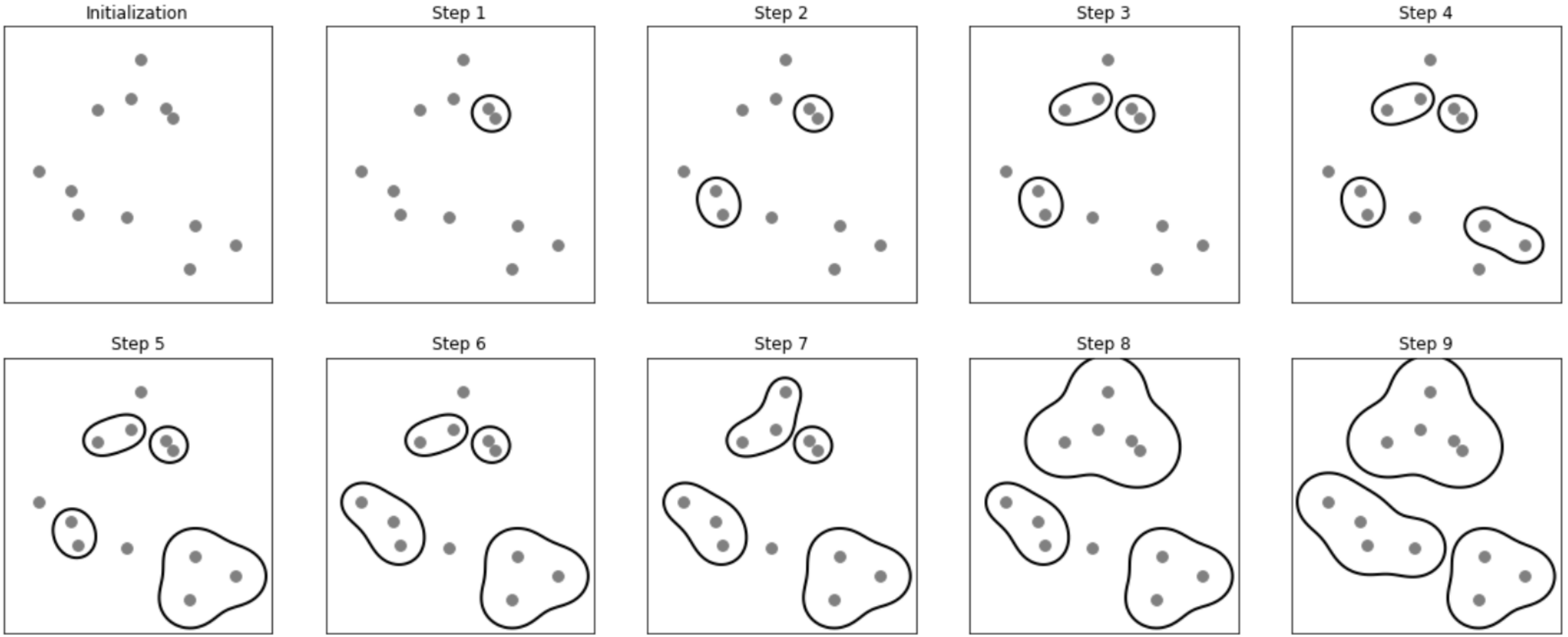
- **凝聚聚类**是采用自底向上策略，基于相同原则将簇组织成层次树的分层聚类算法。
- **算法思想：**

  首先声明每个点是自己的簇，

  然后合并两个最相似的簇，

  直到某种停止准则或只剩下一个簇为止。
- Scikit-learn中实现的停止准则是簇的个数。

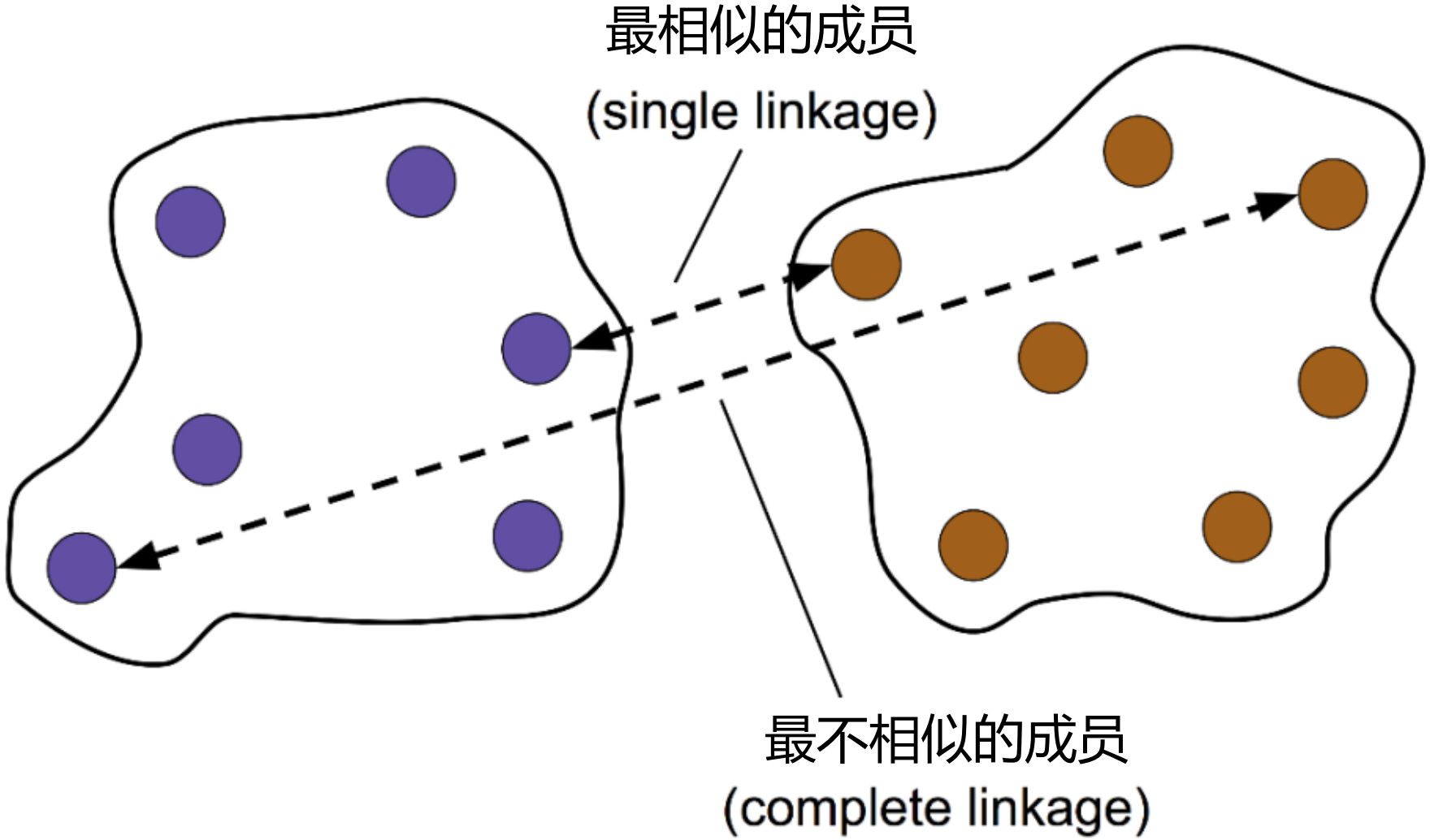
4.2.2 凝聚聚类 | 示意图



凝聚聚类用迭代的方式合并两个最近的簇

- 如何度量“最相似的簇”？ 根据连接(linkage)准则。
- Scikit-learn实现以下四种**连接准则**：
  - **单连接** (single linkage)：计算两个簇中最相似成员（样本）之间的距离，并将此作为两个簇的距离。合并距离最小的两个簇。
  - **全连接** (complete linkage)：寻找两个簇中最不相似的成员，并将这两个成员的距离作为两个簇的距离。合并距离最小的两个簇。
  - **平均连接** (average linkage)：将所有簇中两个平均距离最小的簇合并。
  - **ward连接**(ward linkage)：选择两个簇来合并，使得合并后的簇内误差平方和增加最小。Scikit-learn默认选项，通常会得到大小差不多的簇。

单连接与全连接



基于全连接的凝聚聚类算法，**具体步骤如下**：

1. 计算所有样本对距离，形成距离矩阵；
2. 将每个样本表示为一个簇；
3. 以两个簇中最不相似的样本距离作为两个簇的距离，合并距离最近的两个簇；
4. 更新簇的连接矩阵；
5. 重复步骤2~4，直到只剩下一个簇。

sklearn.cluster模块提供了AgglomerativeClustering，可以选择要返回的簇的数量。

```
AgglomerativeClustering (n_clusters=2, metric='euclidean',
                         linkage='ward', …)
```

参数	• *n_clusters*：int型，默认值2，簇的个数。 • *metric*：用于计算连接的测度方法，可选自{“euclidean”, “l1”, “l2”, “manhattan”, “cosine”, “precomputed”}，默认为“euclidean”。如果linkage取“ward”,这里只能取“euclidean”。 • *linkage*：使用哪种连接准则。{“ward”, “complete”, “average”, “single”},默认为“ward”。
方法	• **fit(X)** 模型学习，根据特征或距离矩阵拟合层次聚类。 • **fit_predict(X)** 拟合并返回每个样本的聚类分配结果。
属性	• **labels_** 每个样本点的标签 • **cluster_centers_** 簇中心坐标，[n_clusters, n_features]

注意：**凝聚聚类工作原理，决定了该类算法不能对新数据点做出预测，因此，AgglomerativeClustering没有predict方法。**

例4.11 凝聚聚类应用

(1) 用make_blobs生成20个样本点，有两个特征，3个簇，标准差为1.0。画出散点图。

(2) 计算全连接（complete）矩阵，画出树状图，观察图选出合适的簇个数k。

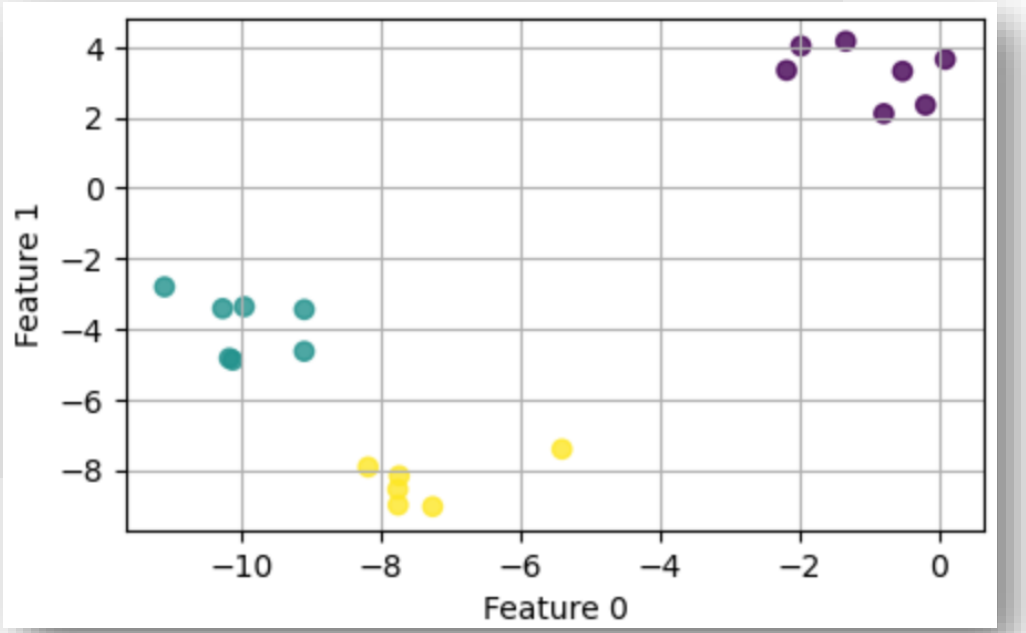
(3) 用凝聚聚类算法(n_clusters为k)，对数据集作层次聚类。画出聚类结果散点图。

(1) 生成数据，画出散点图。

```
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X,y = make_blobs(random_state=1,n_features=2,centers = 3,n_samples=20)

fig = plt.figure(figsize=(5,3))
plt.scatter(X[:,0],X[:,1],c=y,alpha=0.8,cmap='viridis')
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.grid()
```



凝聚聚类结果可视化为**树状图**(dendrogram)。

两步：

1. 调用scipy.cluster.hierarchy子模块的**linkage**函数，在数据上应用指定连接的凝聚聚类方法，结果将**返回一个连接矩阵**。

```
a = linkage(X, method= 'single',metric='euclidean') #计算连接距离，返回数组
```

参数：X为样本，

method，设置连接准则，可取'single', 'complete', 'ward', 'average'

metric，设置距离测度, 如 'euclidean'

2. 调用scipy.cluster.hierarchy的**dendrogram**函数，画出树状图。

```
dendrogram(a) #将层次聚类绘制为树状图。
```

参数：a为上一步计算得到的连接矩阵

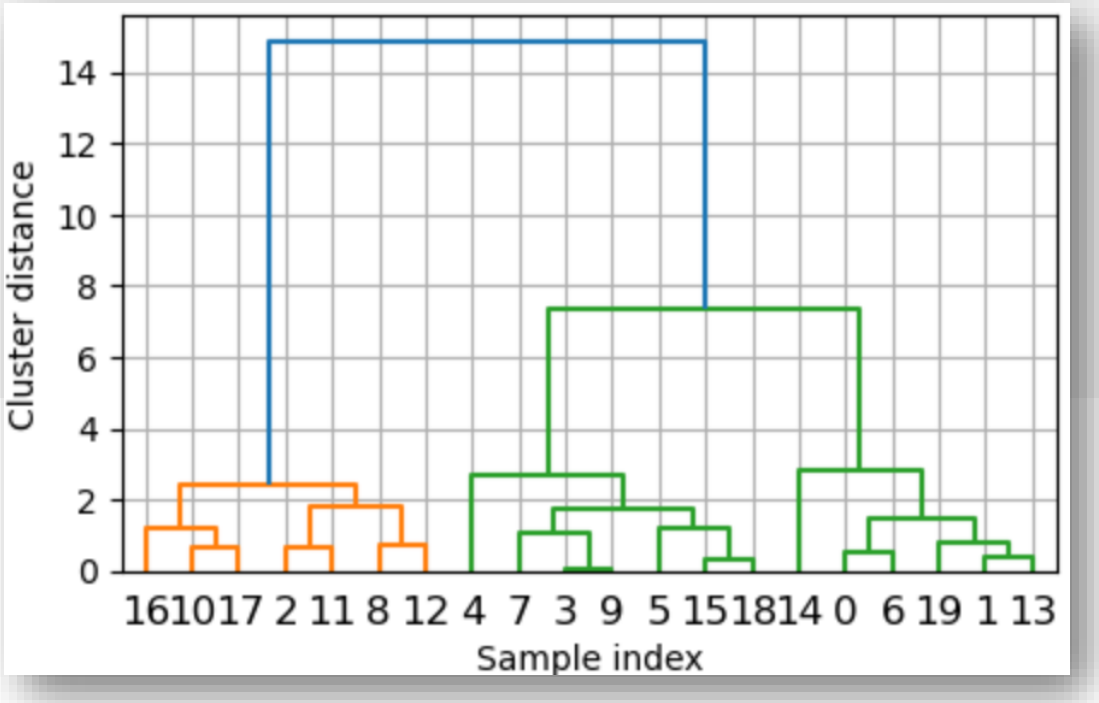
例4.11 凝聚聚类应用 | 代码

(2)计算全连接矩阵，画出树状图

```
from scipy.cluster.hierarchy import dendrogram,linkage

linkage_array = linkage(X, method='complete',metric='euclidean') #计算连接距离，返回数组


fig = plt.figure(figsize=(5,3))
dendrogram(linkage_array)
plt.xlabel("Sample index")
plt.ylabel("Cluster distance")
```



观察树状图，取n_clusters=3

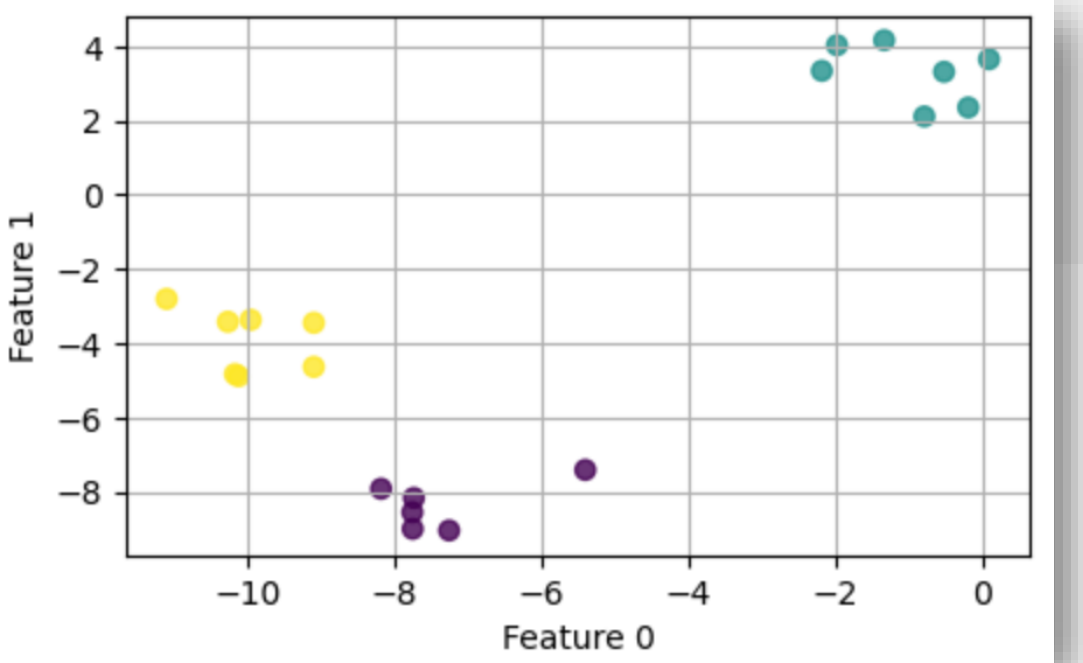
例4.11 凝聚聚类应用 | 代码

(3)用凝聚聚类算法(n_clusters为3)， 对数据集作层次聚类。画出聚类结果散点图。

```
agg = AgglomerativeClustering(n_clusters=3,metric='euclidean',linkage='complete')
assignment = agg.fit_predict(X)

fig = plt.figure(figsize=(5,3))
plt.scatter(X[:,0],X[:,1],c=assignment,alpha=0.8,cmap='viridis')
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.grid()
```

取n_clusters=3



作业十

10.1 k均值的应用

利用scikit-Learn的make_blobs函数，生成4个簇共300个样本，各个簇标准差为0.6。

对该数据集进行kMeans聚类分析，可视化聚类结果。

令k取1~10，画出惯性与k关系图，用肘方法选出最优k。

作业十

10.2 玻璃数据层次聚类分析

（1）读取glass.csv文件，存入一个数据框对象中，**对数据进行标准化缩放处理。**

（2）计算连接矩阵（取ward连接），画出凝聚聚类的树状图。

观察选出聚类簇数k。

（3）用AgglomerativeClustering聚类，n_clusters=k, linkage取ward，画出聚类结果的散点图。

（4）将聚类结果标签作为1列添加到（1）的数据框中。

补充：make_blobs函数

sklearn.datasets下的**make_blobs函数:**

```
make_blobs(n_samples=100, n_features=2,  centers=None,
           cluster_std=1.0, random_state=None, ……)
```

参数:

① n_samples: 若int类型，则为样本数，这些样本将均分到各个簇，默认值为100。
若为数组，则数组元素表示各个簇的样本数目。

② n_features: 特征数目，默认为2

③ centers: 中心个数(int)，或中心坐标，形状为（n_centers,n_features）。当n_samples为整数，且centers为None时，生成3个簇。

④ cluster_std: float，簇的标准差，默认值为1.0。

返回值: X，y —X是生成的样本，y是每个样本的簇成员标签。