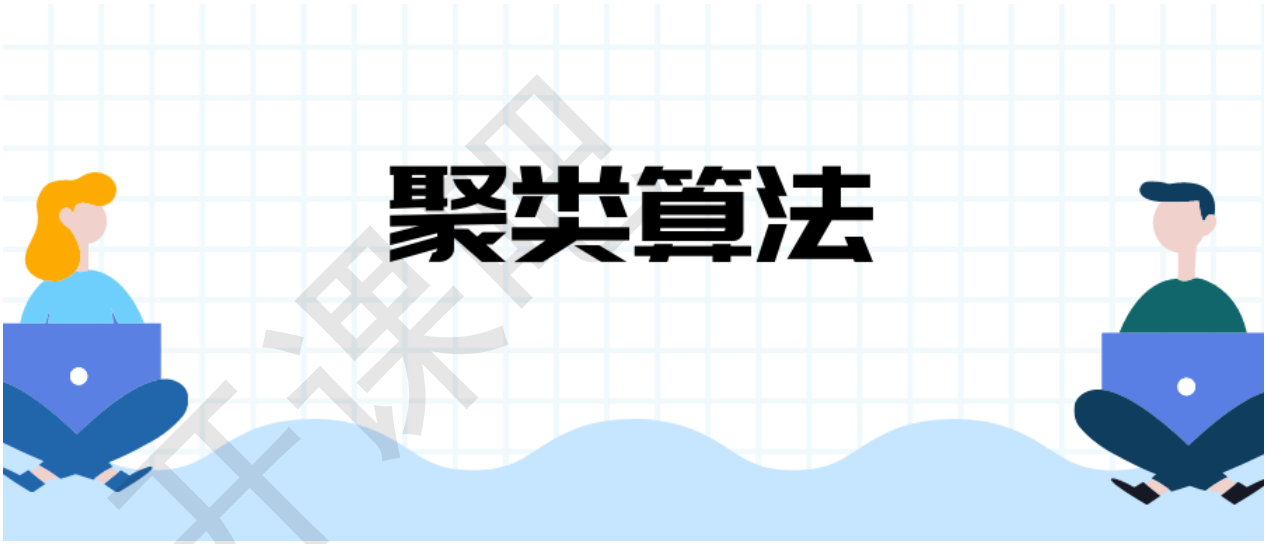



# 聚类算法

## 一、课前准备

- 熟悉python的基本用法
- 熟悉监督学习和无监督学习的区别

## 二、课堂主题

- K-means算法的原理
- K-means算法的应用
- 层次聚类的原理
- 层次聚类的使用

## 三、课堂目标

1. 理解机器学习中非监督学习概念及应用场景
2. 聚类中距离的计算方式
3. k-means算法原理
4. K-means算法中K值求解、算法性能评估的方法
5. 了解层次聚类的用法

## 四、知识要点

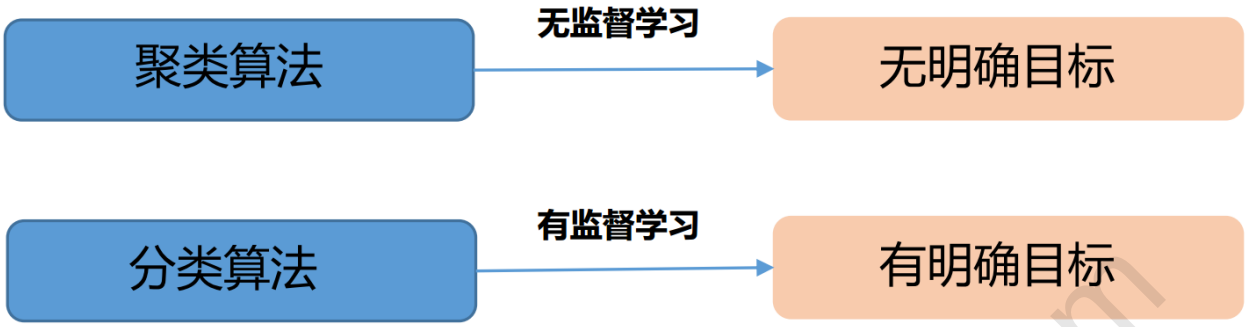
### 4.1 机器学习中的非监督学习

非监督学习又叫无监督学习，事先不需要为数据打上标签，需要机器自己学会数据特征，通常作为数据规约技术，将数据进行聚类或者对数据进行降维。

本节课我们学习继续学习中非监督问题在聚类方面的应用，聚类广泛应用在生物和行为科学、市场以及互联网中，通过将数据分为几类，探究原因或者制定策略。如在本节课中，我们通过对用户行为进行聚类，得到用户的行为特征，进而制定相应的策略。在聚类算法中，最常用到的是K-means算法。

- **“物以类聚，人以群分”**。对事物进行分类，是人们认识事物的出发点，也是人们认识世界的一种重要方法。
- **无监督学习也称聚类分析。** 无监督学习源于许多研究领域，受到很多应用需求的推动。**例如**，
- **在复杂网络分析**中，人们希望发现具有内在紧密联系的社团
- 在**图像分析**中，人们希望将图像分割成具有类似性质的区域
- 在**文本处理**中，人们希望发现具有相同主题的文本子集
- 在**有损编码技术**中，人们希望找到信息损失最小的编码
- 在**顾客行为分析**中，人们希望发现消费方式类似的顾客群，以便制订有针对性的客户管理方式和提高营销效率。
- 这些情况都可以在适当的条件下归为聚类分析。

聚类与分类的区别




聚类是没有明确目标，需要我们定性或定量去描述数据间相似或相异的度，这个度我们一般使用距离来进行表示。

## 4.2 聚类的基本概念

聚类的目标是将相似的研究目标聚集在同一个类别，相异的目标分布在不同的类别。度量分类效果好坏的标准是类内距离最小，类间距离最大。

### 4.2.1 距离的定义

- 闵可夫斯基距离

  闵可夫斯基距离是衡量数值点之间距离的一种非常常见的方法，假设数值点P和Q坐标如下：

  $P = (x_1, x_2, x_3, \ldots, x_n)$

$Q=(y_1,y_2,y_3,\ldots,y_n)$

那么闵可夫斯基距离的定义为：

$(\sum_{i=1}^{n}|x_i-y_i|^p)^{\frac{1}{p}}$

- 欧式距离

  对于闵可夫斯基距离，当p=2时，我们称之为欧式距离（欧几里得距离）。

  $d(x,y)=\sqrt{\sum_{k=1}^{n}(x_k-y_k)^2}$

- 曼哈顿距离

  对于闵可夫斯基距离，当p=1时，我们称之为曼哈顿距离。

  $d(x,y)=\sum_{k=1}^{n}|x_k-y_k|$

- 切比雪夫距离

  对于闵可夫斯基距离，当P趋近于无穷大时，可以转化为切比雪夫距离。

  $(\lim_{\infty}\sum_{i=1}^{n}|x_i-y_i|^p)^{\frac{1}{p}}=max_{i=1}^{n}|x_i-y_i|$

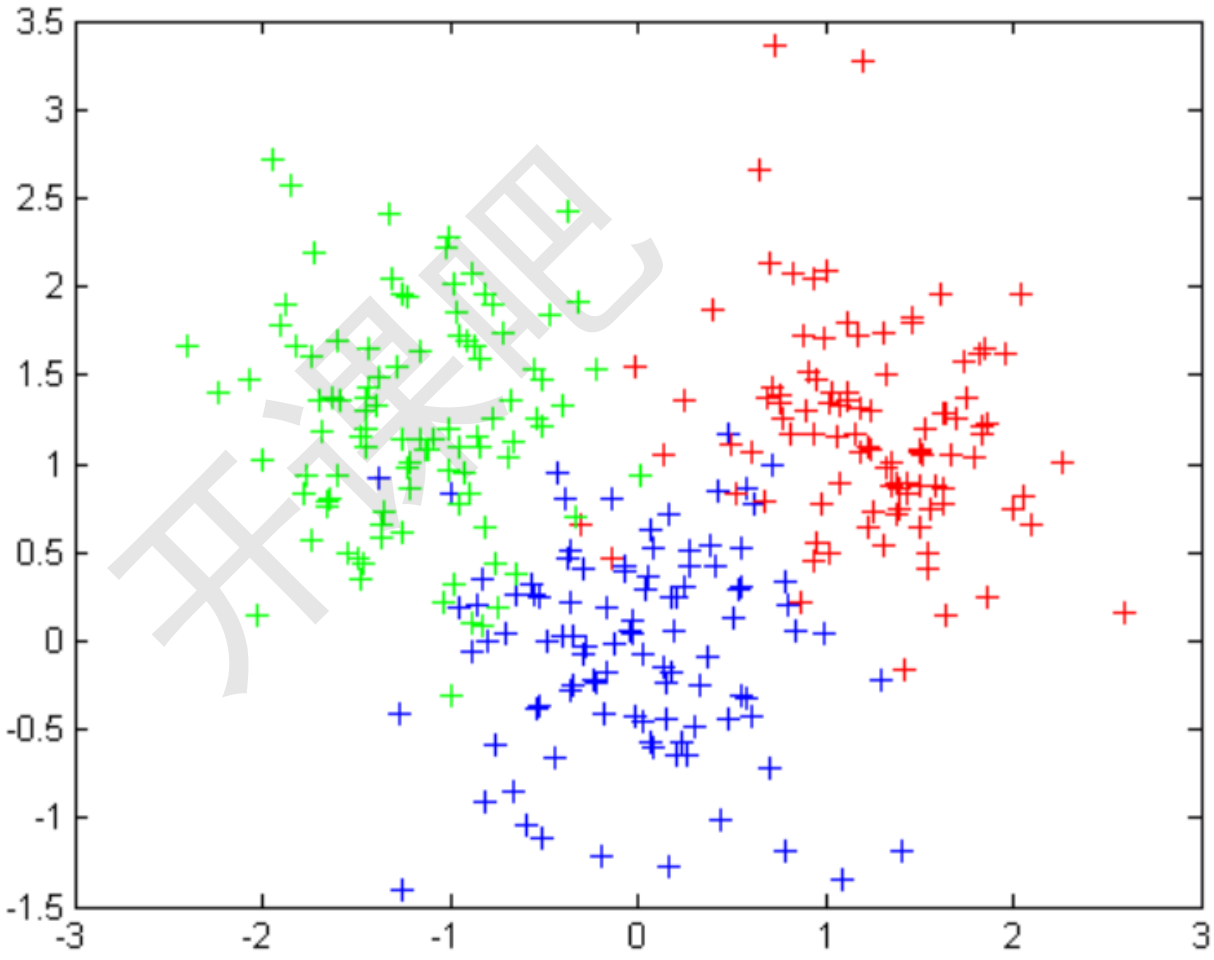
# 4.3 k-means算法

注意:K-means算法通常可以应用于维数、数值都很小且连续的数据集

## 4.3.1 k-means算法概念

K-means算法的目标是在数据中查找一个个组，组的数量由变量K表示。根据数据所提供的特征，通过迭代运算将每个数据点分配给K个组中的其中一个组。K-means算法用在文档分类、用户分类、保险欺诈检测等场景。

k-means聚类算法效果如下图所示




## 4.3.2 k-means算法原理

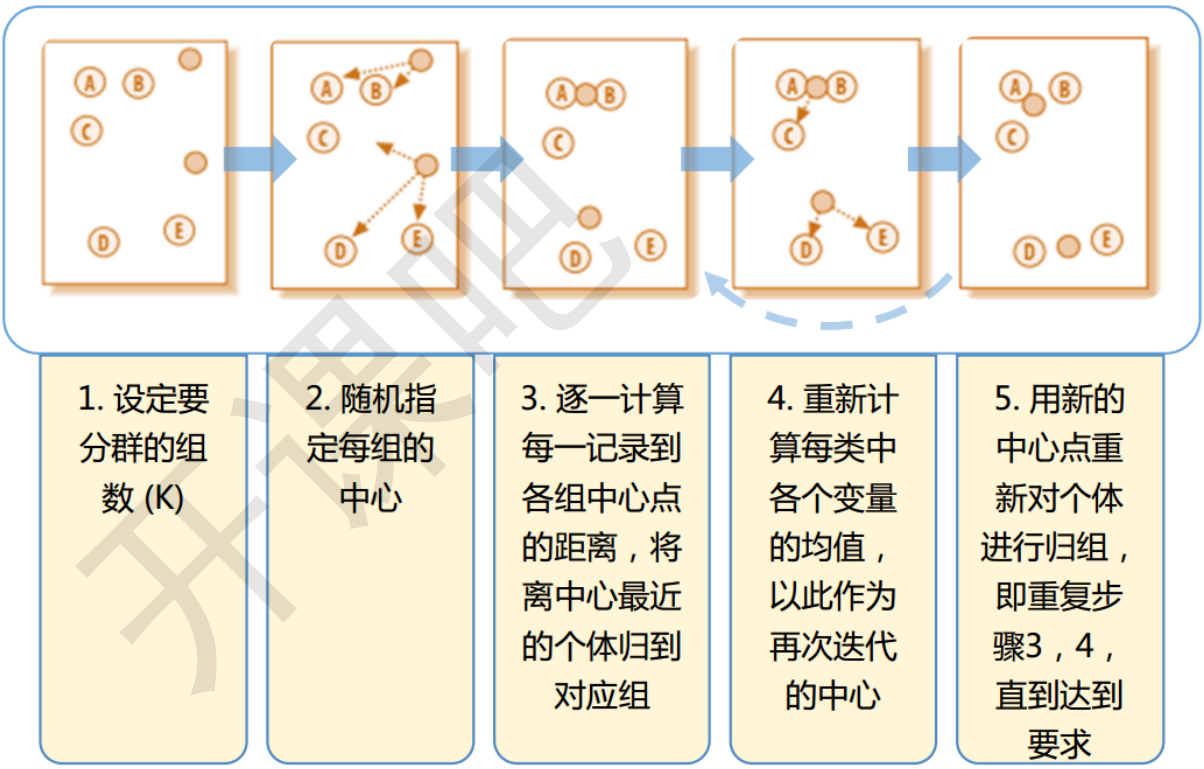
k-means算法使用欧式距离来计算数据之间的距离，算法的具体过程如下。

输入:簇的数目k和包含n个对象的数据库

输出:k个簇

算法步骤:

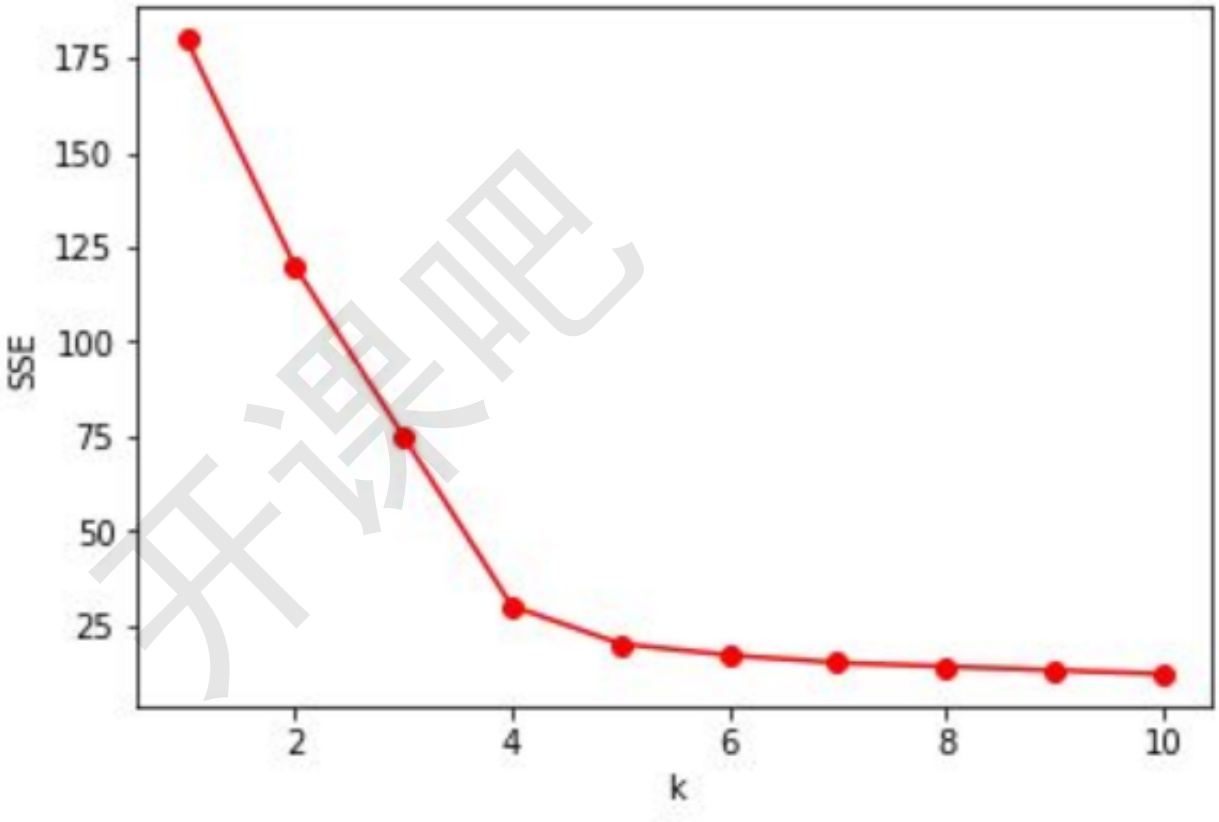
1. 为每个聚类确定一个初始聚类中心，共k个。
2. 将样本集中的样本按照最小距离准则分配到最邻近聚类。
3. 使用每个聚类中的样本均值作为新的聚类中心。
4. 重复步骤2、步骤3，直到聚类中心不再变化。
5. 结束，得到k个聚类。




### 4.3.3 k值的确定

- 肘部法

肘部法则会把不同K值的成本函数值画出来。随着K值的增大，平均畸变程度会减小;每个类包含的样本数会减少，于是样本离其重心会更近。但是，随着K值继续增大，平均畸变程度的改善效果会不断减低。K值增大过程中，畸变程度的改善效果下降幅度最大的位置对应的K值就是肘部。 其中需要解释的一点是: 每个类的畸变程度等于该类重心与其内部成员位置距离的平方和。也即我们前面所说的每个类的类内离差平方和。若类内部的成员彼此间越紧凑则类的畸变程度越小，反之，若类内部的成员彼此间越分散则类的畸变程度越大。




- 轮廓系数

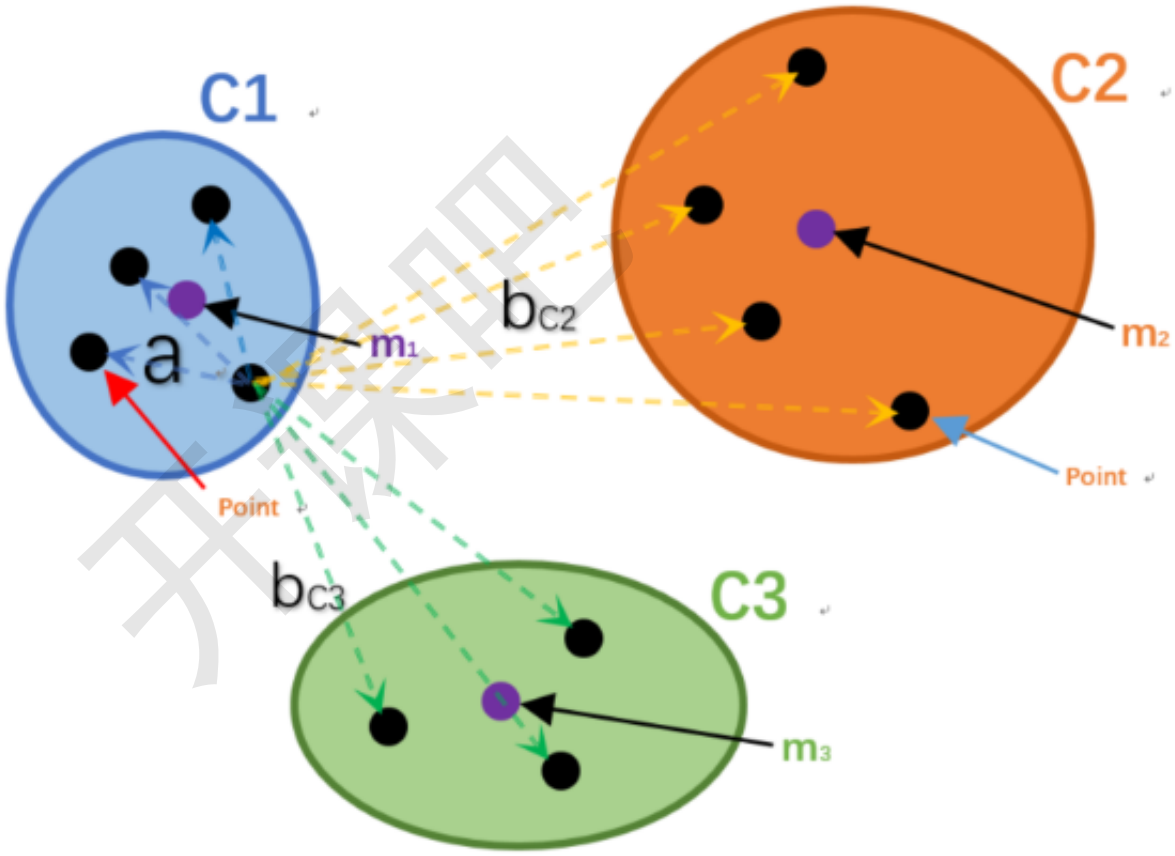
轮廓系数，是聚类效果好坏的一种评价方式。最早由 Peter J. Rousseeuw 在 1986 提出。它结合内聚度和分离度两种因素。可以用来在相同原始数据的基础上用来评价不同算法、或者算法不同运行方式对聚类结果所产生的影响。

假设我们已经通过聚类算法将待分类数据进行了聚类，分为了 k 个簇 。对于簇中的每个向量。分别计算它们的轮廓系数。对于其中的一个点 i 来说:

计算 a(i) = average(i向量到所有它属于的簇中其它点的距离)

计算 b(i) = min (i向量到所有非本身所在簇的点的平均距离) ,那么 i 向量轮廓系数就为:

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




- a表示C1簇中的某一个样本点Xi到自身簇中其他样本点的距离总和的平均值。
- bC2表示样本点Xi 到C2簇中所有样本点的距离总和的平均值。
- bC3表示样本点Xi 到C3簇中所有样本点的距离总和的平均值。
- 我们定义b = min(bC2 ,bC3)

可见轮廓系数的值是介于 [-1,1] ，越趋近于1代表内聚度和分离度都相对较优。

最后将所有样本点的轮廓系数求平均，就是该聚类结果总的轮廓系数。

### 4.3.4 k-means算法的优缺点

优点:

1. 原理简单，容易实现
2. 内存占用小

缺点:

1. K值需要预先给定，属于预先知识，很多情况下K值的估计是非常困难的，对于像计算全部微信用户的交往圈这样的场景就完全的没办法用K-Means进行。
2. K-Means算法对初始选取的聚类中心点是敏感的，不同的随机种子点得到的聚类结果完全不同。
3. K均值算法并不适合所有的数据类型。
4. 对离群点的数据进行聚类时，K均值也有问题，这种情况下，离群点检测和删除有很大的帮助。

## 4.4 K-Means++算法

K-Means++（初始化优化）

根据K-Means算法的原理我们不难发现，最初的质心选择对聚类的结果和运行时间有着很大的影响，因此我们需要选择合适的K个质心，K-Means++就使用了更优化的方法来初始化质心，让我们来看一下K-Means++的优化策略：

（1）从输入的数据点集合中随机选择一个点作为第一个聚类中心$\mu_1$；

（2）对于数据集中的每一个点$x_i$，计算它与最近聚类中心(指已选择的聚类中心)的距离D(x)；

（3）选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大；

（4）重复（2）（3）步骤直到选择出k个聚类质心；

（5）利用这k个质心来作为初始化质心去运行标准的K-Means算法。

过程中提到的D(x)计算方法如下：

$$D(x_i) = argmin||x_i - \mu_r||_2^2, r = 1, 2, \ldots, k$$

# 4.5 KMeans参数说明

```
from sklearn.cluster import KMeans

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=None,
       algorithm='auto')
```

```
KMeans(n_jobs=None, precompute_distances='auto')
```

| 参数 | 说明 |
| --- | --- |
| n-cluster | 分类簇的数量 |
| max_iter | 最大的迭代次数（默认300） |
| n_init | 算法的运行次数（默认10） |
| init | 接收待定的strings。 kmeans++表示该初始化策略选择的初始均值向量之间都距离比较远，他的效果越好。random表示从数据中随机选择k个样本作为初始均值向量。还可以提供一个数组（n_cluster,n_feature）,该数组作为初始均值向量。 |
| precompute_distances | 三个可选值，‘auto’，True 或者 False。 预计算距离，计算速度更快但占用更 多内存。 |
| tol | 接收float，表示算法收敛的阈值 |
| n_jobs | 表示任务使用CPU数量 |
| random_state | 表示随机数生成器种子 |
| verbose | 0表示不输出日志信息，1表示每隔一段时间打印一次日志，大于1打印次数频繁 |

## 4.6 K-Means算法的使用

```
# 导入包
import numpy as np
import sklearn
from sklearn.datasets import make_blobs  # 导入产生模拟数据的方法
from sklearn.cluster import KMeans  # 导入kmeans类
```

```
# 1. 产生模拟数据；random_state此参数让结果容易复现，随机过程跟系统时间有关
N = 100
centers = 4

X, Y = make_blobs(n_samples=N, n_features=2, centers=centers, random_state=28)
print(Y)
```

```
[0 1 3 1 2 0 3 0 0 3 0 2 1 1 3 1 0 2 0 1 0 1 3 1 3 0 1 3 1 1 0 0 1 3 1 0 3
 2 3 1 3 3 0 2 2 0 2 3 0 1 3 3 3 3 2 2 0 0 2 2 2 3 1 2 0 3 3 1 2 0 3 0 1 0
 2 2 3 1 1 1 0 3 3 2 3 2 1 0 2 2 2 1 2 1 2 2 0 2 1 0]
```

```
# 2. 模型构建;init初始化函数的意思
km = KMeans(n_clusters=centers, init='random', random_state=28)
km.fit(X)
```

```
KMeans(init='random', n_clusters=4, random_state=28)
```

```
# 实际的y值
Y
```

```
array([0, 1, 3, 1, 2, 0, 3, 0, 0, 3, 0, 2, 1, 1, 3, 1, 0, 2, 0, 1, 0, 1,
       3, 1, 3, 0, 1, 3, 1, 1, 0, 0, 1, 3, 1, 0, 3, 2, 3, 1, 3, 3, 0, 2,
       2, 0, 2, 3, 0, 1, 3, 3, 3, 3, 2, 2, 0, 0, 2, 2, 2, 3, 1, 2, 0, 3,
       3, 1, 2, 0, 3, 0, 1, 0, 2, 2, 3, 1, 1, 1, 0, 3, 3, 2, 3, 2, 1, 0,
       2, 2, 2, 1, 2, 1, 2, 2, 0, 2, 1, 0])
```

```
# 模型的预测
y_hat = km.predict(X[:10])
y_hat
```

```
array([0, 2, 1, 2, 3, 0, 1, 0, 0, 1], dtype=int32)
```

```
print("所有样本距离所属簇中心点的总距离和为:%.5f" % km.inertia_)
print("所有样本距离所属簇中心点的平均距离为:%.5f" % (km.inertia_ / N))
```

```
所有样本距离所属簇中心点的总距离和为:184.64263
所有样本距离所属簇中心点的平均距离为:1.84643
```

```
print("所有的中心点聚类中心坐标:")
cluter_centers = km.cluster_centers_
print(cluter_centers)  #4组
```

```
所有的中心点聚类中心坐标:
[[ 4.63158330e+00  1.81989519e+00]
 [-6.61167883e+00  6.91472919e+00]
 [-7.38206071e+00 -2.32141230e+00]
 [ 5.54777181e+00 -6.72218901e-03]]
```

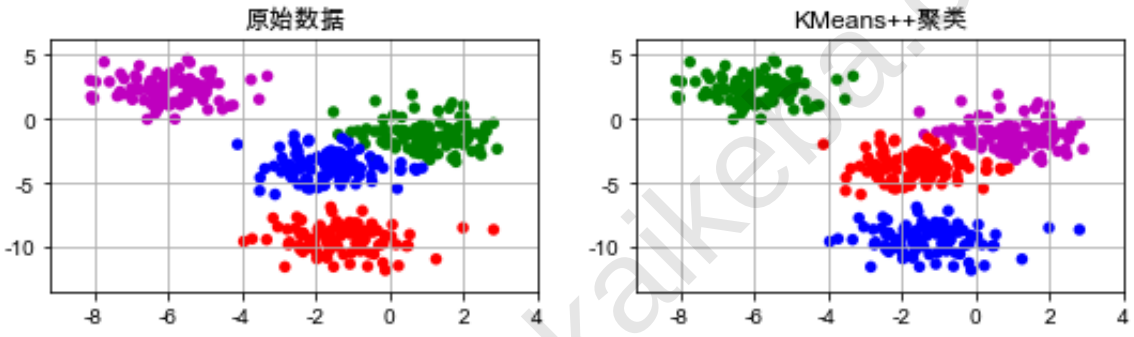
```
print("score其实就是所有样本点离所属簇中心点距离和的相反数:")
print(km.score(X))
```

```
score其实就是所有样本点离所属簇中心点距离和的相反数:
-184.64263227954362
```

```
# !/usr/bin/python
# -*- coding:utf-8 -*-

import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as ds
import matplotlib.colors
from sklearn.cluster import KMeans
import matplotlib as mpl
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')  #忽视


def expand(a, b):
    d = (b - a) * 0.1
    return a - d, b + d


if __name__ == "__main__":
    N = 400
    centers = 4
    data, y = ds.make_blobs(N, n_features=2, centers=centers, random_state=2)
    data2, y2 = ds.make_blobs(N,
                              n_features=2,
                              centers=centers,
                              cluster_std=(1, 2.5, 0.5, 2),
                              random_state=2)
    data3 = np.vstack((data[y == 0][:], data[y == 1][:50], data[y == 2][:20],
                       data[y == 3][:5]))
```

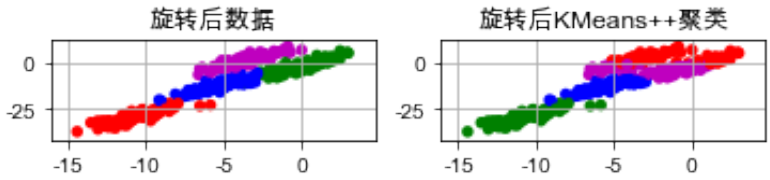
```
    y3 = np.array([0] * 100 + [1] * 50 + [2] * 20 + [3] * 5)

    cls = KMeans(n_clusters=4, init='k-means++')
    y_hat = cls.fit_predict(data)
    y2_hat = cls.fit_predict(data2)
    y3_hat = cls.fit_predict(data3)

    m = np.array(((1, 1), (1, 3)))
    data_r = data.dot(m)
    y_r_hat = cls.fit_predict(data_r)

    # 设置字符集，防止中文乱码
    # mpl.rcParams["font.sans-serif"] = [u'simHei'] #Win自带的字体
    plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']  #Mac自带的字体
    mpl.rcParams["axes.unicode_minus"] = False
    cm = matplotlib.colors.ListedColormap(list('rgbm'))
```

```
plt.figure(figsize=(9, 10), facecolor='w')
plt.subplot(421)
plt.title(u'原始数据')
plt.scatter(data[:, 0], data[:, 1], c=y, s=30, cmap=cm, edgecolors='none')
x1_min, x2_min = np.min(data, axis=0)
x1_max, x2_max = np.max(data, axis=0)
x1_min, x1_max = expand(x1_min, x1_max)
x2_min, x2_max = expand(x2_min, x2_max)
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)

plt.subplot(422)
plt.title(u'KMeans++聚类')
plt.scatter(data[:, 0], data[:, 1], c=y_hat, s=30, cmap=cm, edgecolors='none')
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)
```

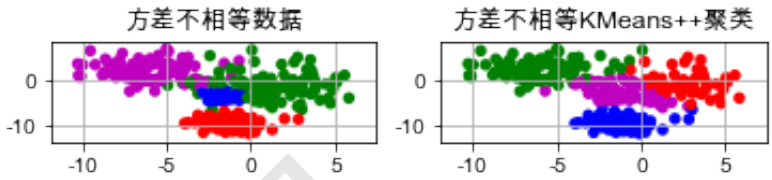



```
plt.subplot(423)
```

```
plt.title(u'旋转后数据')
plt.scatter(data_r[:, 0], data_r[:, 1], c=y, s=30, cmap=cm, edgecolors='none')
x1_min, x2_min = np.min(data_r, axis=0)
x1_max, x2_max = np.max(data_r, axis=0)
x1_min, x1_max = expand(x1_min, x1_max)
x2_min, x2_max = expand(x2_min, x2_max)
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)

plt.subplot(424)
plt.title(u'旋转后KMeans++聚类')
plt.scatter(data_r[:, 0], data_r[:, 1], c=y_r_hat, s=30, cmap=cm,
edgecolors='none')
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)
```

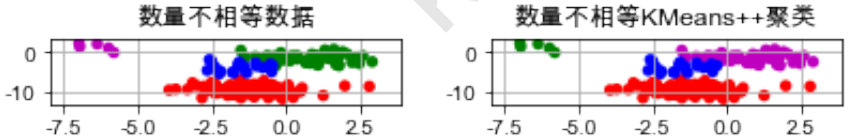



```
plt.subplot(425)
plt.title(u'方差不相等数据')
plt.scatter(data2[:, 0], data2[:, 1], c=y2, s=30, cmap=cm, edgecolors='none')
x1_min, x2_min = np.min(data2, axis=0)
x1_max, x2_max = np.max(data2, axis=0)
x1_min, x1_max = expand(x1_min, x1_max)
x2_min, x2_max = expand(x2_min, x2_max)
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)

plt.subplot(426)
plt.title(u'方差不相等KMeans++聚类')
plt.scatter(data2[:, 0], data2[:, 1], c=y2_hat, s=30, cmap=cm,
edgecolors='none')
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)
```




```
plt.subplot(427)
plt.title(u'数量不相等数据')
plt.scatter(data3[:, 0], data3[:, 1], s=30, c=y3, cmap=cm, edgecolors='none')
x1_min, x2_min = np.min(data3, axis=0)
x1_max, x2_max = np.max(data3, axis=0)
x1_min, x1_max = expand(x1_min, x1_max)
x2_min, x2_max = expand(x2_min, x2_max)
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)

plt.subplot(428)
plt.title(u'数量不相等KMeans++聚类')
plt.scatter(data3[:, 0], data3[:, 1], c=y3_hat, s=30, cmap=cm,
edgecolors='none')
plt.xlim((x1_min, x1_max))
plt.ylim((x2_min, x2_max))
plt.grid(True)

plt.tight_layout(2)
plt.suptitle(u'数据分布对KMeans聚类的影响', fontsize=18)
plt.subplots_adjust(top=0.92)
plt.show()
```

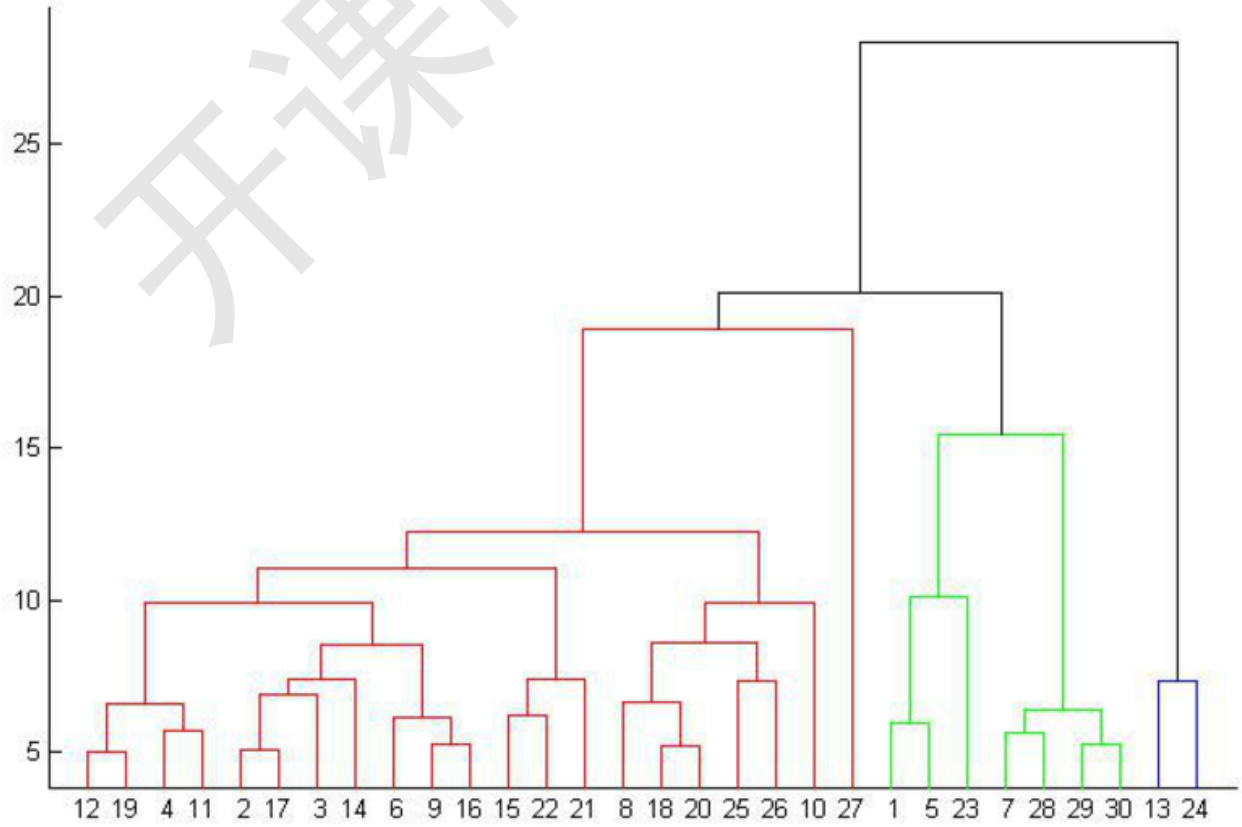
数据分布对KMeans聚类的影响



# 4.7 层次聚类

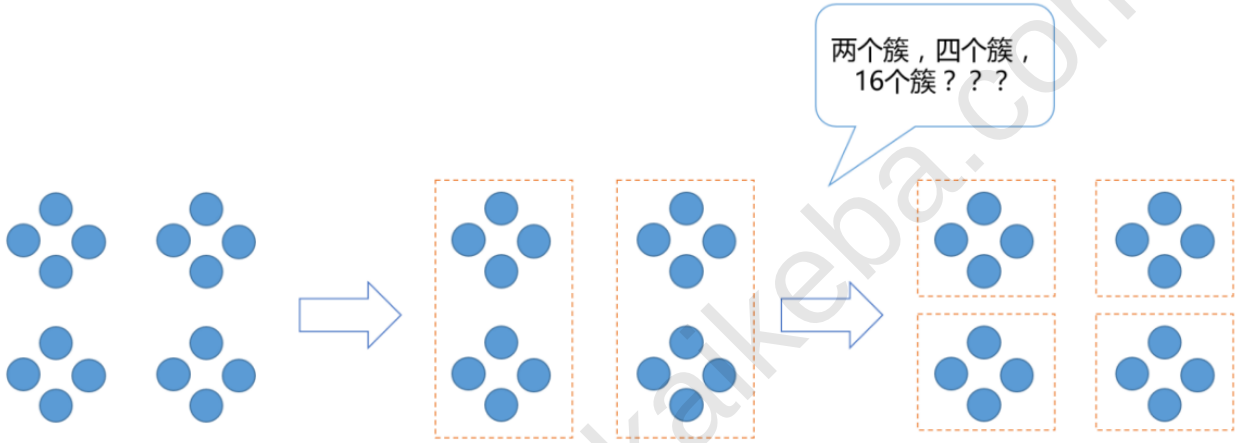
## 4.7.1 层次聚类的概念

层次聚类(Hierarchical Clustering)是聚类算法的一种，通过计算不同类别数据点间的相似度来创建一棵有层次的嵌套聚类树。在聚类树中，不同类别的原始数据点是树的最低层，树的顶层是一个聚类的根节点。创建聚类树有自下而上合并和自上而下分裂两种方法。




## 4.7.2 层次聚类的划分

**怎么分更合适?**




**自底向上的合并算法**

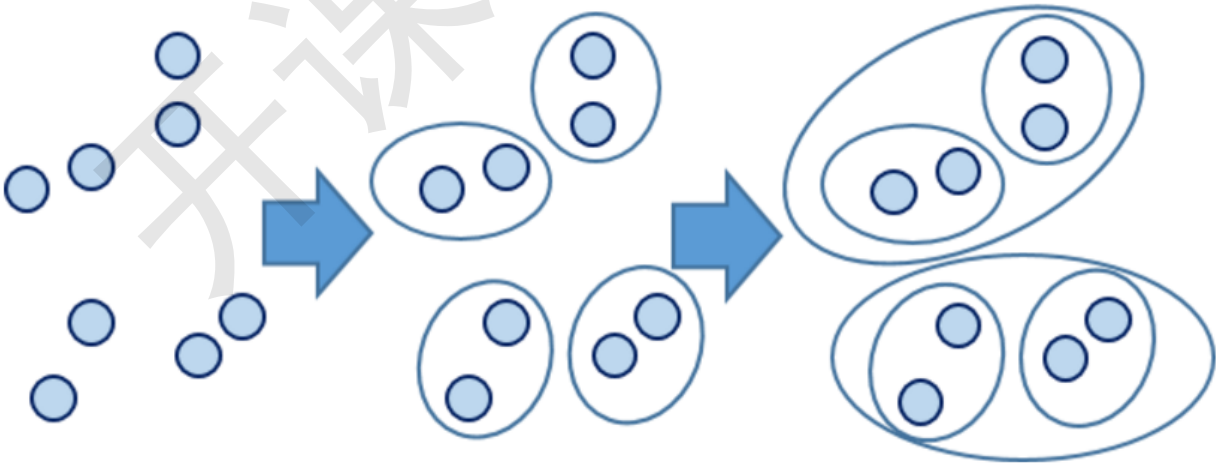
- 相似度的计算

在进行层次聚类的划分时，我们使用欧式距离来计算不同类别数据点间的距离（相似度）

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2}$$

- 合并方法

层次聚类的合并算法通过计算两类数据点间的相似性，对所有数据点中最为相似的两个数据点进行组合，并反复迭代这一过程。简单的说层次聚类的合并算法是通过计算每一个类别的数据点与所有数据点之间的距离来确定它们之间的相似性，距离越小，相似度越高。并将距离最近的两个数据点或类别进行组合，生成聚类树。



## 4.7.3 层次聚类实例

- 有如下的数据点

| | |
|---|---|
| A | 16.9 |
| B | 38.5 |
| C | 39.5 |
| D | 80.8 |
| E | 82 |
| F | 34.6 |
| G | 116.1 |

- 通过点间的距离计算得到如下的形式

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 21.60 | 22.60 | 63.90 | 65.10 | 17.70 | 99.20 |
| B | 21.60 | 0 | 1.00 | 42.30 | 43.50 | 3.90 | 77.60 |
| C | 22.60 | **1.00** | 0 | 41.30 | 42.50 | 4.90 | 76.60 |
| D | 63.90 | 42.30 | 41.30 | 0 | 1.20 | 46.20 | 35.30 |
| E | 65.10 | 43.50 | 42.50 | 1.20 | 0 | 47.40 | 34.10 |
| F | 17.70 | 3.90 | 4.90 | 46.20 | 47.40 | 0 | 81.50 |
| G | 99.20 | 77.60 | 76.60 | 35.30 | 34.10 | 81.50 | 0 |

找到距离最小的两个点后，我们就可以先把这两个点聚到一个类别中，表格中D（B，C）最小。

下一次计算的时候，对于已经分到一个类别中的点，我们采用距离均值的方式进行计算。

比如：A到(B,C)的距离。

$$D = \frac{\sqrt{(B-A)^2}+\sqrt{(C-A)^2}}{2} = \frac{21.6+22.6}{2}$$

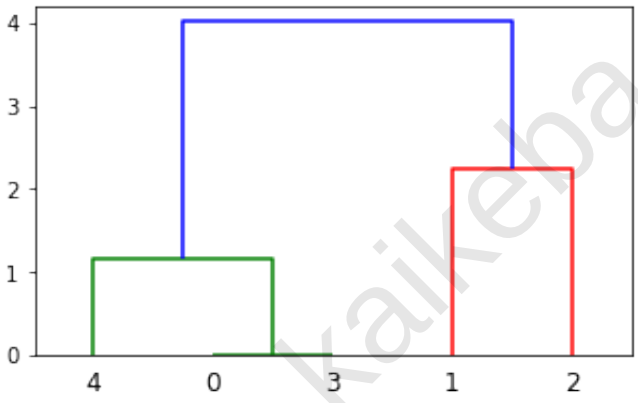
当一层聚类结束后需要进行二层聚类的时候我们同样计算的是两类点的距离均值。

比如：(A,F)和(B,C)的距离。

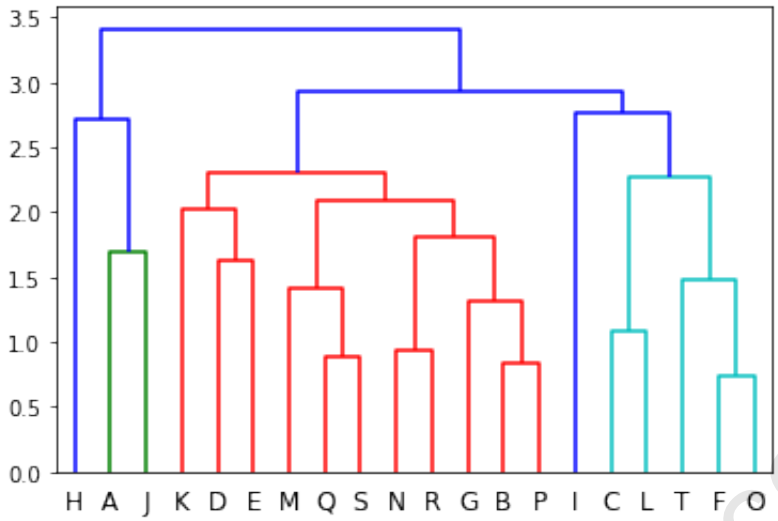
$$D = \frac{\sqrt{(A-B)^2}+\sqrt{(A-C)^2}+\sqrt{(F-B)^2}+\sqrt{(F-C)^2}}{2}$$

## 4.8 层次聚类算法的使用

```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from matplotlib import pyplot as plt
X = [[1, 2], [3, 2], [4, 4], [1, 2], [1, 3]]
Z = linkage(X, 'ward')
f = fcluster(Z, 4, 'distance')
fig = plt.figure(figsize=(5, 3))
dn = dendrogram(Z)
plt.show()
```



```
###cluster.py
#导入相应的包
import scipy
import scipy.cluster.hierarchy as sch
from scipy.cluster.vq import vq, kmeans, whiten
import numpy as np
import matplotlib.pylab as plt

#生成待聚类的数据点,这里生成了20个点,每个点4维:
points = scipy.random.randn(20, 4)
#加一个标签进行区分
A = []
for i in range(20):
    a = chr(i + ord('A'))
    A.append(a)
#1. 层次聚类
#生成点与点之间的距离矩阵,这里用的欧氏距离:
disMat = sch.distance.pdist(points, 'euclidean')
#进行层次聚类:
Z = sch.linkage(disMat, method='average')
#将层级聚类结果以树状图表示出来并保存为plot_dendrogram.png
P = sch.dendrogram(Z, labels=A)
```




# 五、总结

- 聚类算法的思想
- K-means算法的思想
- K-means算法的优化
- K-means算法的应用
- 层次聚类的思想
- 层次算法的应用

# 六、作业

- 总结有监督和无监督算法的区别
- 梳理本节课两种聚类算法的原理及优化