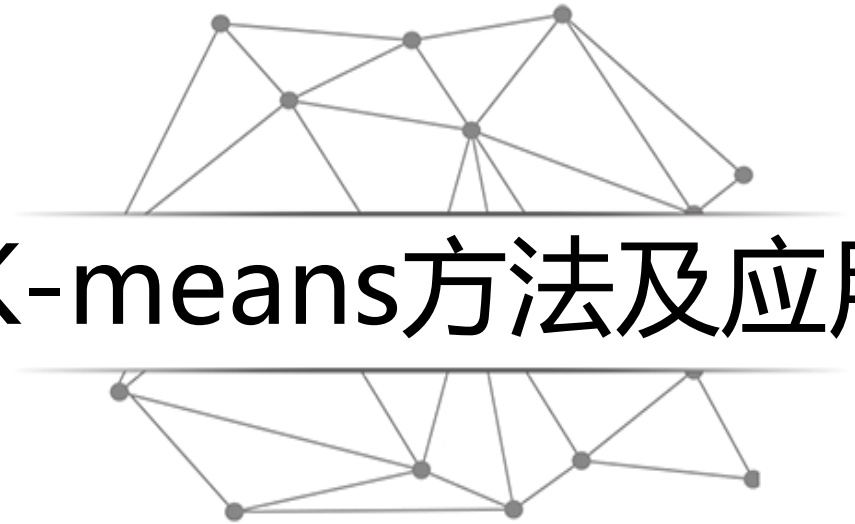
# 无监督学习-聚类

*ML05*


礼欣
www.python123.org



# K-means方法及应用

# K-means聚类算法

k-means算法以k为参数，把n个对象分成k个簇，使簇内具有较高的相似度，而簇间的相似度较低。

其处理过程如下：

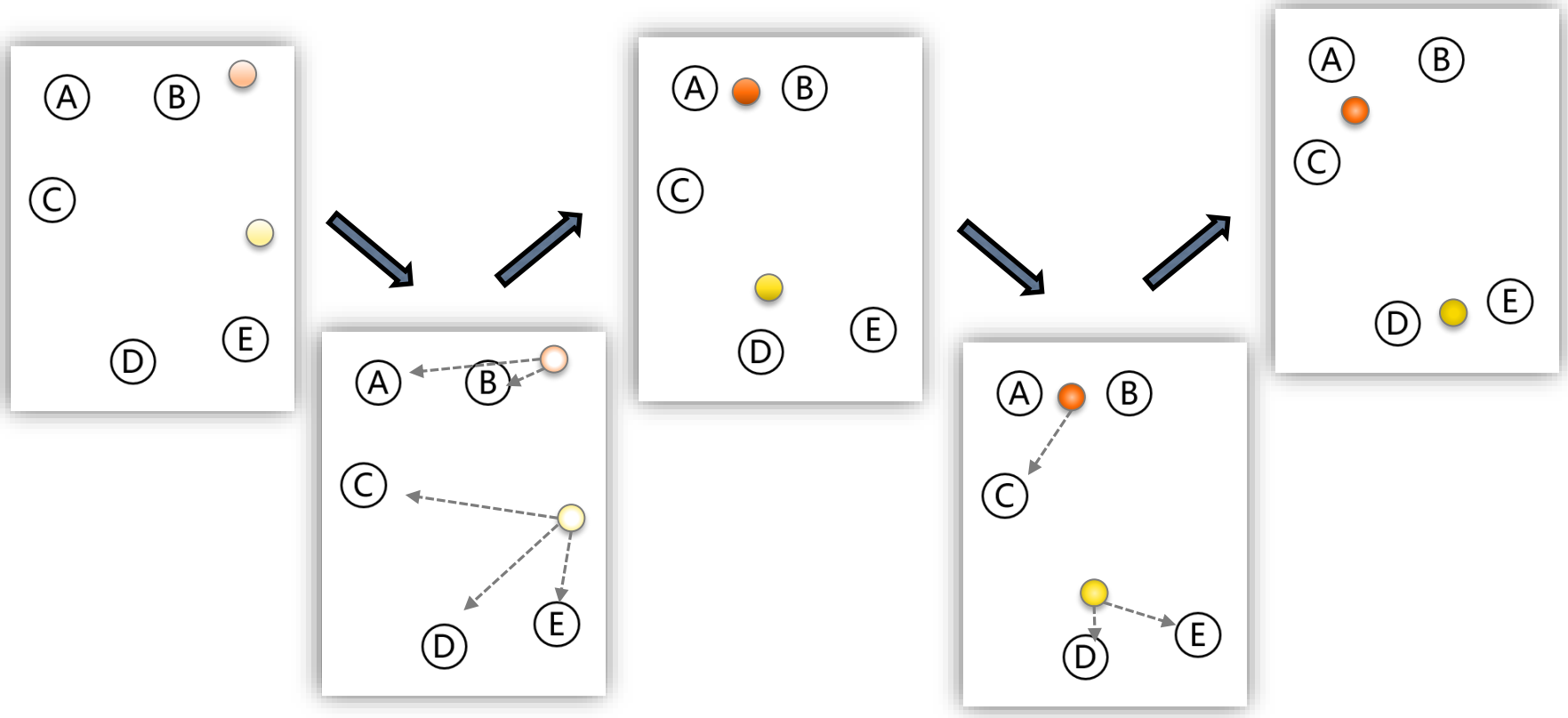
1.随机选择k个点作为初始的聚类中心；

2.对于剩下的点，根据其与聚类中心的距离，将其归入最近的簇

3.对每个簇，计算所有点的均值作为新的聚类中心

4.重复2、3直到聚类中心不再发生改变

# K-means聚类算法



# K-means的应用

**数据介绍：**

现有1999年全国31个省份城镇居民家庭平均每人全年消费性支出的八个主要变量数据，这八个变量分别是：食品、衣着、家庭设备用品及服务、医疗保健、交通和通讯、娱乐教育文化服务、居住以及杂项商品和服务。利用已有数据，对31个省份进行聚类。

**实验目的：**

通过聚类，了解1999年各个省份的消费水平在国内的情况。

技术路线：`sklearn.cluster.Kmeans`

# 数据实例：

1999年全国31个省份城镇居民家庭平均每人全年消费性支出数据

| 城市 | 食品 | 衣着 | 家庭设备用品及服务 | 医疗保健 | 交通和通讯 | 娱乐教育文化服务 | 居住 | 杂项商品和服务 |
|---|---|---|---|---|---|---|---|---|
| 北京 | 2959 | 730.79 | 749.41 | 513.34 | 467.87 | 1141.82 | 478.42 | 457.64 |
| 天津 | 2460 | 495.47 | 697.33 | 302.87 | 284.19 | 735.97 | 570.84 | 305.08 |
| 河北 | 1496 | 515.9 | 362.37 | 285.32 | 272.95 | 540.58 | 364.91 | 188.63 |
| 山西 | 1406 | 477.77 | 290.15 | 208.57 | 201.5 | 414.72 | 281.84 | 212.1 |
| 内蒙古 | 1304 | 524.29 | 254.83 | 192.17 | 249.81 | 463.09 | 287.87 | 192.96 |
| 辽宁 | 1731 | 553.9 | 246.91 | 279.81 | 239.18 | 445.2 | 330.24 | 163.86 |
| 吉林 | 1562 | 492.42 | 200.49 | 218.36 | 220.69 | 459.62 | 360.48 | 147.76 |
| 黑龙江 | 1410 | 510.71 | 211.88 | 277.11 | 224.65 | 376.82 | 317.61 | 152.85 |
| 上海 | 3712 | 550.74 | 893.37 | 346.93 | 527 | 1034.98 | 720.33 | 462.03 |
| 江苏 | 2208 | 449.37 | 572.4 | 211.92 | 302.09 | 585.23 | 429.77 | 252.54 |
| 浙江 | 2629 | 557.32 | 689.73 | 435.69 | 514.66 | 795.87 | 575.76 | 323.36 |
| 安徽 | 1845 | 430.29 | 271.28 | 126.33 | 250.56 | 513.18 | 314 | 151.39 |
| 福建 | 2709 | 428.11 | 334.12 | 160.77 | 405.14 | 461.67 | 535.13 | 232.29 |
| 江西 | 1564 | 303.65 | 233.81 | 107.9 | 209.7 | 393.99 | 509.39 | 160.12 |
| 山东 | 1676 | 613.32 | 550.71 | 219.79 | 272.59 | 599.43 | 371.62 | 211.84 |
| 河南 | 1428 | 431.79 | 288.55 | 208.14 | 217 | 337.76 | 421.31 | 165.32 |
| 湖南 | 1942 | 512.27 | 401.39 | 206.06 | 321.29 | 697.22 | 492.6 | 226.45 |
| 湖北 | 1783 | 511.88 | 282.84 | 201.01 | 237.6 | 617.74 | 523.52 | 182.52 |
| 广东 | 3055 | 353.23 | 564.56 | 356.27 | 811.88 | 873.06 | 1082.82 | 420.81 |
| 广西 | 2034 | 300.82 | 338.65 | 157.78 | 329.06 | 621.74 | 587.02 | 218.27 |
| 海南 | 2058 | 186.44 | 202.72 | 171.79 | 329.65 | 477.17 | 312.93 | 279.19 |
| 重庆 | 2303 | 589.99 | 516.21 | 236.55 | 403.92 | 730.05 | 438.41 | 225.8 |
| 四川 | 1974 | 507.76 | 344.79 | 203.21 | 240.24 | 575.1 | 430.36 | 223.46 |
| 贵州 | 1674 | 437.75 | 461.61 | 153.32 | 254.66 | 445.59 | 346.11 | 191.48 |
| 云南 | 2194 | 537.01 | 369.07 | 249.54 | 290.84 | 561.91 | 407.7 | 330.95 |
| 西藏 | 2647 | 839.7 | 204.44 | 209.11 | 379.3 | 371.04 | 269.59 | 389.33 |
| 陕西 | 1473 | 390.89 | 447.95 | 259.51 | 230.61 | 490.9 | 469.1 | 191.34 |
| 甘肃 | 1526 | 472.98 | 328.9 | 219.86 | 206.65 | 449.69 | 249.66 | 228.19 |
| 青海 | 1655 | 437.77 | 258.78 | 303 | 244.93 | 479.53 | 288.56 | 236.51 |
| 宁夏 | 1375 | 480.89 | 273.84 | 317.32 | 251.08 | 424.75 | 228.73 | 195.93 |
| 新疆 | 1609 | 536.05 | 432.46 | 235.82 | 250.28 | 541.3 | 344.85 | 214.4 |

# 实验过程：

- 使用算法： K-means聚类算法
- 实现过程：

1. 建立工程，导入sklearn相关包

```
import numpy as np
from sklearn.cluster import KMeans
```

关于一些相关包的介绍：

- NumPy是Python语言的一个扩充程序库。支持高级大量的维度数组与矩阵运算，此外也针对数组运算提供大量的数学函数库。
- 使用sklearn.cluster.KMeans可以调用K-means算法进行聚类

## 2. 加载数据，创建K-means算法实例，并进行训练，获得标签：

```
if __name__ == '__main__':
    data,cityName = loadData('city.txt')
    km = KMeans(n_clusters=3)
    label = km.fit_predict(data)
    expenses = np.sum(km.cluster_centers_,axis=1)
    #print(expenses)
    CityCluster = [[],[],[]]
    for i in range(len(cityName)):
        CityCluster[label[i]].append(cityName[i])
    for i in range(len(CityCluster)):
        print("Expenses:%.2f" % expenses[i])
        print(CityCluster[i])
```

1.利用loadData方法读取数据

2.创建实例

3.调用Kmeans（ ）fit_predict()方法进行计算

## 2. 加载数据，创建K-means算法实例，并进行训练，获得标签：

- data,cityName = loadData('city.txt')
- km = KMeans(n_clusters=3)
- label = km.fit_predict(data)
- expenses = np.sum(km.cluster_centers_,axis=1)

调用KMeans方法所需参数：

- n_clusters：用于指定聚类中心的个数
- init：初始聚类中心的初始化方法
- max_iter：最大的迭代次数
- 一般调用时只用给出n_clusters即可，init默认是k-means++，max_iter默认是300

其它参数：

- data：加载的数据
- label：聚类后各数据所属的标签
- axis: 按行求和
- fit_predict()：计算簇中心以及为簇分配序号

## 2. 加载数据，创建K-means算法实例，并进行训练，获得标签：

重点方法解释：data,cityName = loadData('city.txt')

```
def loadData(filePath):
    fr = open(filePath,'r+')
    lines = fr.readlines()
    retData = []
    retCityName = []
    for line in lines:
        items = line.strip().split(",")
        retCityName.append(items[0])
        retData.append([float(items[i]) for i in range(1,len(items))])
    for i in range(1,len(items))])
        return retData,retCityName
```

r+：读写打开一个文本文件

.read() 每次读取整个文件，它通常用于将文件内容放到一个字符串变量中

.readlines() 一次读取整个文件（类似于 .read() ）

.readline() 每次只读取一行，通常比 .readlines() 慢得多。仅当没有足够内存可以一次读取整个文件时，才应该使用 .readline()。

## 2. 加载数据，创建K-means算法实例，并进行训练，获得标签：

重点方法解释：data,cityName = loadData('city.txt')

```
def loadData(filePath):
    fr = open(filePath,'r+')
    lines = fr.readlines()
    retData = []
    retCityName = []
    for line in lines:
        items = line.strip().split(",")
        retCityName.append(items[0])
        retData.append([float(items[i]) for i in range(1,len(items))])
     for i in range(1,len(items))])
        return retData,retCityName
```

retCityName：用来存储城市名称

retData：用来存储城市的各项消费信息

返回值：返回城市名称，以及该城市的各项消费信息

## 2. 加载数据，创建K-means算法实例，并进行训练，获得标签：

```
if __name__ == '__main__':
    data,cityName = loadData('city.txt')
    km = KMeans(n_clusters=3)
    label = km.fit_predict(data)
    expenses = np.sum(km.cluster_centers_,axis=1)
    #print(expenses)
    CityCluster = [[],[],[]]
    for i in range(len(cityName)):
        CityCluster[label[i]].append(cityName[i])
    for i in range(len(CityCluster)):
        print("Expenses:%.2f" % expenses[i])
        print(CityCluster[i])
```

将城市按label分成设定的簇
将每个簇的城市输出
将每个簇的平均花费输出

## 3. 输出标签，查看结果

- 将城市按照消费水平n_clusters类，消费水平相近的城市聚集在一类中
- expense：聚类中心点的数值加和，也就是平均消费水平

聚成2类：km = KMeans(n_clusters=2)

```
Expenses:4040.42
['河北', '山西', '内蒙古', '辽宁', '吉林', '黑龙江', '江苏', '安徽', '江西', '山东',
'河南', '湖南', '湖北', '广西', '海南', '四川', '贵州', '云南', '陕西', '甘肃', '青
海', '宁夏', '新疆']
Expenses:6457.13
['北京', '天津', '上海', '浙江', '福建', '广东', '重庆', '西藏']
```

关于函数参数文档：http://scikit-learn.org/stable/

聚成3类：km = KMeans(n_clusters=3)

```
Expenses:3827.87
['河北', '山西', '内蒙古', '辽宁', '吉林', '黑龙江', '安徽', '江西', '山东',
'河南', '湖北', '贵州', '陕西', '甘肃', '青海', '宁夏', '新疆']
Expenses:5113.54
['天津', '江苏', '浙江', '福建', '湖南', '广西', '海南', '重庆', '四川', '云
南', '西藏']
Expenses:7754.66
['北京', '上海', '广东']
```

聚成4类：km = KMeans(n_clusters=4)

```
Expenses:3788.76
['河北', '山西', '内蒙古', '辽宁', '吉林', '黑龙江', '江西', '山东', '河南', '贵州',
'陕西', '甘肃', '青海', '宁夏', '新疆']
Expenses:5678.62
['天津', '浙江', '福建', '重庆', '西藏']
Expenses:4512.27
['江苏', '安徽', '湖南', '湖北', '广西', '海南', '四川', '云南']
Expenses:7754.66
['北京', '上海', '广东']
```

从结果可以看出消费水平相近的省市聚集在了一类，例如消费最高的“北京” “上海” “广东”聚集在了消费最高的类别。聚4类时，结果可以比较明显的看出消费层级。

# 拓展&&改进

计算两条数据相似性时，Sklearn 的K-Means默认用的是欧式距离。虽然还有余弦相似度，马氏距离等多种方法，但没有设定计算距离方法的参数。

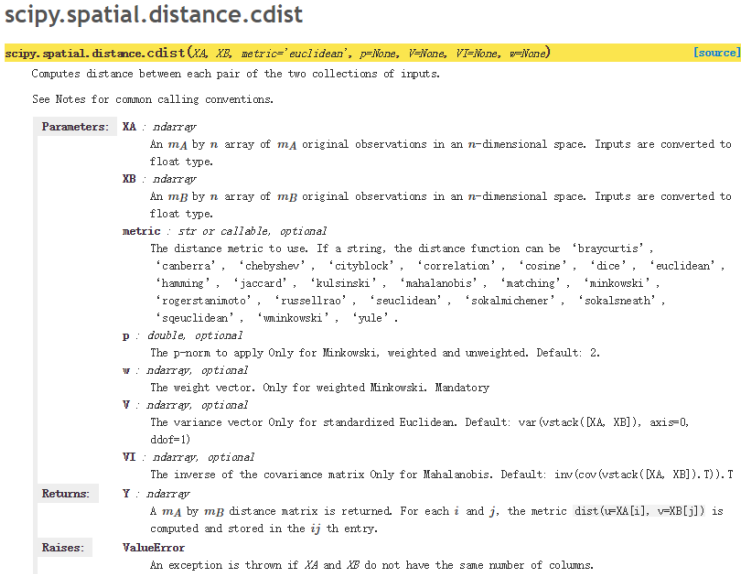
```
161  # Pairwise distances
162  def euclidean_distances(X, Y=None, Y_norm_squared=None, squared=False,
163                          X_norm_squared=None):
164      """
165      Considering the rows of X (and Y=X) as vectors, compute the
166      distance matrix between each pair of vectors.
167
168      For efficiency reasons, the euclidean distance between a pair of row
169      vector x and y is computed as::
170
171          dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
172
```

- 如果想要自定义计算距离的方式时，可以更改此处源码。
- 建议使用scipy.spatial.distance.cdist方法

源码地址：https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/metrics/pairwise.py

# 拓展&&改进

使用形式：scipy.spatial.distance.cdist(A, B, metric= ‘cosine’ )



scipy.spatial.distance.cdist

scipy.spatial.distance.cdist(*XA, XB, metric='euclidean', p=None, V=None, VI=None, w=None*) [source]

Computes distance between each pair of the two collections of inputs.

See Notes for common calling conventions.

**Parameters:**

**XA** : *ndarray*
An $m_A$ by $n$ array of $m_A$ original observations in an $n$-dimensional space. Inputs are converted to float type.

**XB** : *ndarray*
An $m_B$ by $n$ array of $m_B$ original observations in an $n$-dimensional space. Inputs are converted to float type.

**metric** : *str or callable, optional*
The distance metric to use. If a string, the distance function can be ‘braycurtis’, ‘canberra’, ‘chebyshev’, ‘cityblock’, ‘correlation’, ‘cosine’, ‘dice’, ‘euclidean’, ‘hamming’, ‘jaccard’, ‘kulsinski’, ‘mahalanobis’, ‘matching’, ‘minkowski’, ‘rogerstanimoto’, ‘russellrao’, ‘seuclidean’, ‘sokalmichener’, ‘sokalsneath’, ‘sqeuclidean’, ‘wminkowski’, ‘yule’.

**p** : *double, optional*
The p-norm to apply Only for Minkowski, weighted and unweighted. Default: 2.

**w** : *ndarray, optional*
The weight vector. Only for weighted Minkowski. Mandatory

**V** : *ndarray, optional*
The variance vector Only for standardized Euclidean. Default: var(vstack([XA, XB]), axis=0, ddof=1)

**VI** : *ndarray, optional*
The inverse of the covariance matrix Only for Mahalanobis. Default: inv(cov(vstack([XA, XB])).T)).T

**Returns:**

**Y** : *ndarray*
A $m_A$ by $m_B$ distance matrix is returned. For each $i$ and $j$, the metric `dist(u=XA[i], v=XB[j])` is computed and stored in the $ij$ th entry.

**Raises:**

**ValueError**
An exception is thrown if *XA* and *XB* do not have the same number of columns.

重要参数：

- A：A向量
- B：B向量
- metric: 计算A和B距离的方法，更改此参数可以更改调用的计算距离的方法

详细：
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html#scipy.spatial.distance.cdist