K-means 算法介绍和 K 值探讨

《用 Python 玩转数据》　　by 大壮@NJU

K-means 算法是典型的基于距离的聚类算法，采用距离作为相似性的评价指标，两个对象的距离越近，其相似度就越大。而簇是由距离靠近的对象组成的，因此算法目的是得到紧凑并且独立的簇。

假设要将对象分成 k 个簇，算法过程如下：

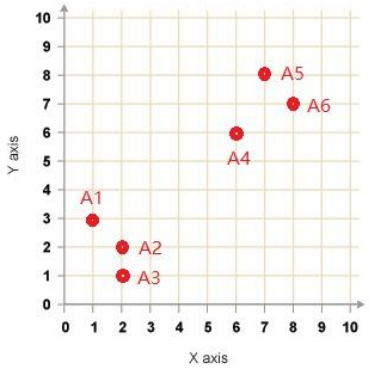
(1) 随机选取任意 k 个对象作为初始聚类的中心（质心，Centroid），初始代表每一个簇；

(2) 对数据集中剩余的每个对象根据它们与各个簇中心的距离将每个对象重新赋给最近的簇；

(3) 重新计算已经得到的各个簇的质心；

(4) 迭代步骤(2)-(3)直至新的质心与原来的质心相等或小于设定的阈值，算法结束。

随意找几个数据简单模拟算法如下：



假设有 6 个点 A1，A2，…，A6：

	X	Y
A1	1	3
A2	2	2
A3	2	1
A4	6	6
A5	7	8
A6	8	7

要聚成 2 类，算法过程如下：

(1) 假设选择 A1 和 A2 为初始质心；

(2) 计算 A3-A6 与 A1 和 A2 的距离，这里用欧氏距离公式 d = sqrt((x1-x2)²+(y1-y2)²)：

	A1	A2
A3	2.24	1
A4	5.83	5.66

A5	6.4	6.4
A6	8.06	7.81

(3) 根据与 A1 和 A2 距离的比较，A3、A4、A6 都离 A2 近，A5 与 A1 和 A2 距离相同，假设 A5 也分到 A2 这一簇，因此形成新的两簇：

簇 1：A1

簇 2：A2，A3，A4，A5，A6

(4) 计算新簇的质心

簇 1 质心：A1

簇 2：新质心“C_temp”计算用每个维度的平均值

((A2.x+A3.x+A4.x+A5.x+A6.x)/5 , (A2.y+A3.y+A4.y+A5.y+A6.y)/5)=(5, 4.8)

	A1	C_temp
A2	1.41	4.1
A3	2.24	4.84
A4	5.83	1.56
A5	6.4	3.77
A6	8.06	3.72

(5) 根据距离数据被分成了新的 2 簇：

簇 1：A1，A2，A3

簇 2：A4，A5，A6

新质心 1 “C_temp1”：((A1.x+A2.x+A3.x)/3 , (A1.y+A2.y+A3.y)/3)=(1.67, 2)

新质心 2 “C_temp2”：((A4.x+A5.x+A6.x)/3 , (A4.y+A5.y+A6.y)/3)=(7, 7)

	C_temp1	C_temp2
A1	1.2	7.21
A2	0.33	7.07
A3	1.05	7.81
A4	5.89	1.41
A5	6.66	1
A6	6.71	1

(6) 依据距离，簇 1 和簇 2 与前一轮一样，收敛，聚类结束，bingo ٩(◕‿◕)۶

簇 1：A1，A2，A3

簇 2：A4，A5，A6

提示：

(1) 在 K-means 算法 k 值通常取决于人的主观经验；

(2) 距离公式常用欧氏距离和余弦相似度公式，前者是根据位置坐标直接计算的，主要体现个体数值特征的差异，而后者更多体现了方向上的差异而不是位置上的，cos θ 越接近 1 个体越相似，可以修正不同度量标准不统一的问题；

(3) K-means 算法获得的是局部最优解，在算法中，初始聚类中心常常是随机选择的，一旦初始值选择的不好，可能无法得到有效的聚类结果，K-means 有一些优化方法，有兴趣可进一步研究。

关于模型 K 值的深入讨论：

对于一堆数据，K 值（簇数）的最优解如何确定呢？常见的有“肘”方法（Elbow method）和轮廓系数法（Silhouette Coeffient），对两种方法基本思想描述

如下，具体公式可查阅相关文档：

① “肘”方法：核心指标是 SSE（sum of the squared errors，误差平方和），即所有样本的聚类误差（累计每个簇中样本到质心距离的平方和），随着 K 的增大每个簇聚合度会增强，SSE 下降幅度会增大，随着 K 值继续增大 SSE 的下降幅度会减少并趋于平缓，SSE 和 K 值的关系图会呈现成一个手肘的形状，此肘部对应的 K 值就是最佳的聚类数。

② 轮廓系数法：结合聚类的凝聚度（Cohesion）和分离度（Separation）来考虑，凝聚度为样本与同簇其他样本的平均距离，分离度为样本与最近簇中所有样本的平均距离，该值处于-1~1 之间，值越大表示聚类效果越好。

以下以 iris 数据为例：

[Elbow method]

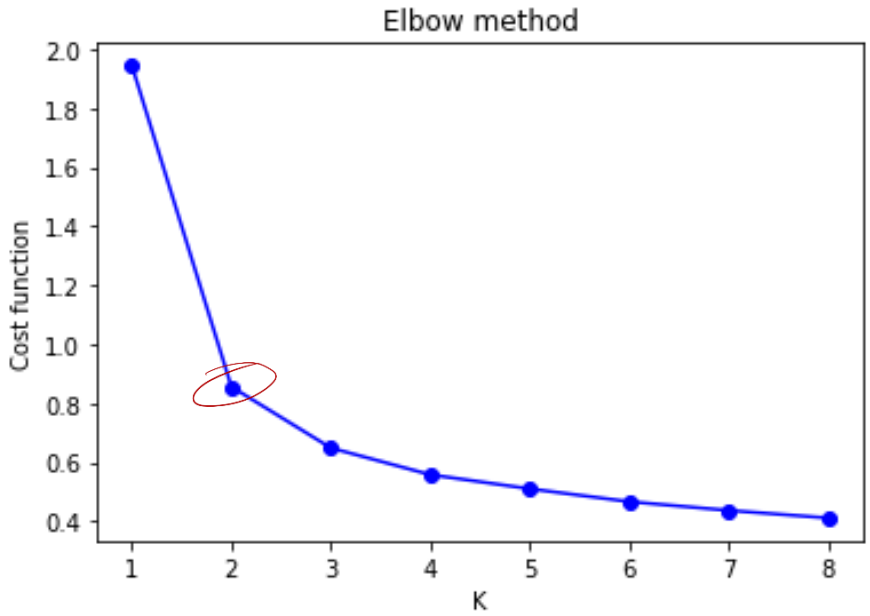
```
import numpy as np
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data

K = range(1, 9)       # 假设可能聚成 1~8 类
lst = []
for k in K:
    kmeans = KMeans(n_clusters = k)
    kmeans.fit(X)
    # 计算对应 K 值时最小值列表和的平均值
    lst.append(sum(np.min(cdist(X,kmeans.cluster_centers_,
      'euclidean'), axis = 1)) / X.shape[0])
    # cdist(X, kmeans.cluster_centers_, 'euclidean')  求 X 到各质心
      cluster_centers_之间的距离平方和，维度为(150, k), 'euclidean'表示使用
      欧式距离计算
    # np.min(cdist(X,kmeans.cluster_centers_, 'euclidean'), axis = 1)  计
      算每一行中的最小值
    # sum(np.min(cdist(X,kmeans.cluster_centers_, 'euclidean'), axis = 1))
      计算每一轮 K 值下最小值列表的和

plt.plot(K, lst, 'bo-')
plt.title('Elbow method')
plt.xlabel("K")
plt.ylabel("Cost function")
plt.show()
```

关系图如下所示，由图看出拐点在 K=2 处，K=3 次之，iris 实际数据分成了三类。



[Silhouette Coeffient]

```
from sklearn. cluster import KMeans
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import metrics

iris = datasets.load_iris()
X = iris.data

K = range(2, 9)     # 假设可能聚成 2~8 类
lst = []
for k in K:
    kmeans_model = KMeans(n_clusters = k).fit(X)
    sc_score = metrics.silhouette_score(X, kmeans_model.labels_,
               metric = 'euclidean')
    # silhouette_score() 计算所有样本的平均轮廓系数
    # kmeans_model.labels_ 每个样本预测的类标签
    # metric = 'euclidean' 使用欧式距离计算
    lst.append(sc_score)

plt.plot(K, lst, ' bo-')
plt.title('Silhouette Coefficient')
plt.xlabel("K")
plt.ylabel("Score")
plt.show()
```

绘制 K 和相应轮廓系数的关系图如下所示，表明 K=2 效果最佳，实际 iris 数据分成了三类。