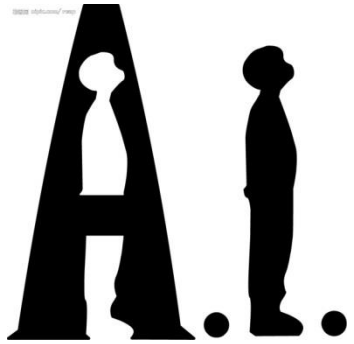
机器学习

讲师：曾江峰

E-mail：jfzeng@ccnu.edu.cn



聚类

outline

牧师-村民模型

- 有四个牧师去郊区布道，一开始牧师们随意选了几个布道点，并且把这几个布道点的情况公告给了郊区所有的村民，于是每个村民到离自己家最近的布道点去听课。
- 听课之后，大家觉得距离太远了，于是每个牧师统计了一下自己的课上所有的村民的地址，搬到了所有地址的中心地带，并且在海报上更新了自己的布道点的位置。
- 牧师每一次移动不可能离所有人都更近，有的人发现A牧师移动以后自己还不如去B牧师处听课更近，于是每个村民又去了离自己最近的布道点……
- 就这样，牧师每个礼拜更新自己的位置，村民根据自己的情况选择布道点，最终稳定了下来。

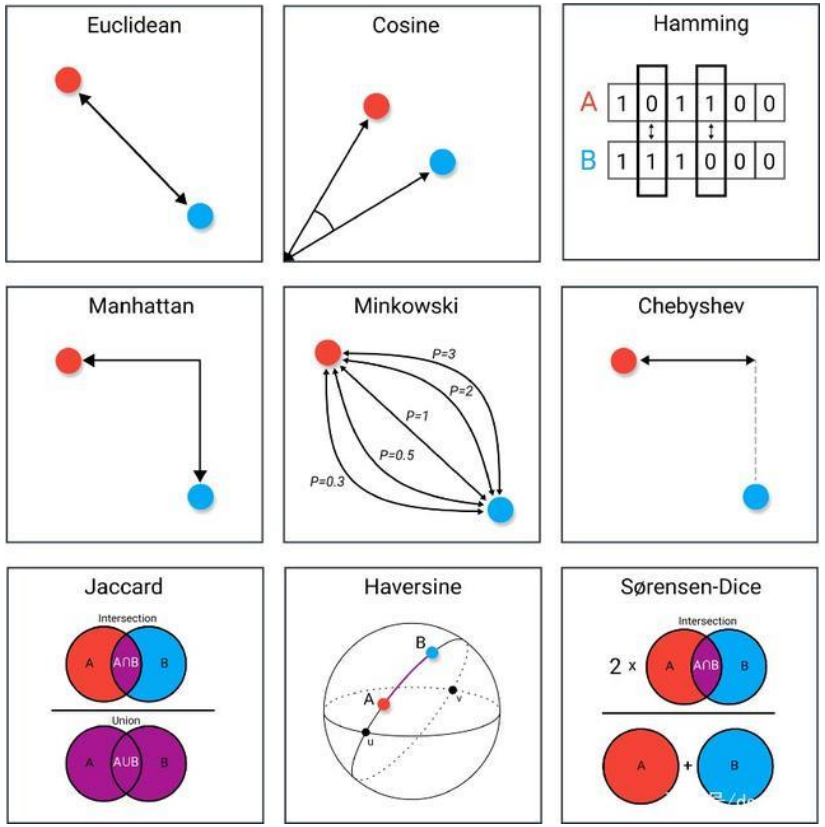
聚类分析

人以类聚，物以群分。

- 聚类分析是一种数据分析技术，属于无监督学习范畴，用于探索数据集中自然出现的群组，称为聚类。
- 聚类分析通过度量特征相似度或者距离，将样本自动划分为若干个类别。
- 聚类的目标是得到较高的簇内相似度和较低的簇间相似度，使得簇间的距离尽可能大，簇内样本与簇中心的距离尽可能小。

距离度量和相似度度量方式

9个数据科学中常见距离度量



距离度量和相似度度量方式

- 相关系数（Correlation coefficient）是度量样本相似度最常用的方式。
- 相关系数越接近1，表示两个样本越相似；相关系数越接近0，表示两个样本越不相似。
- 相关系数是最早由统计学家卡尔·皮尔逊设计的统计指标，是研究变量之间线性相关程度的量，一般用字母 r 表示。简单相关系数又称皮尔逊相关系数或者线性相关系数，其定义式为：

$$r=\frac{\sum\left(X-\overline{X}\right)\left(Y-\overline{Y}\right)}{\sqrt{\sum\left(X-\overline{X}\right)^2\sum\left(Y-\overline{Y}\right)^2}}$$

聚类的分类

- 基于划分的聚类
  - 基于划分的方法是简单、常用的一种聚类方法，它通过将对象划分为互斥的簇进行聚类， 每个对象属于且仅属于一个簇；划分结果旨在使簇之间的相似性低，簇内部的相似度高。
  - 常用算法有kmeans、k-medoids、k-prototype等
- 基于层次的聚类
  - 层次聚类的应用广泛程度仅次于基于划分的聚类，核心思想是通过对数据集按照层次，把数据划分到不同层的簇，从而形成一个树形的聚类结构。层次聚类算法可以揭示数据的分层结构，在树形结构上不同层次进行划分，可以得到不同粒度的聚类结果。
  - 按照层次聚类的过程分为自底向上的聚合聚类和自顶向下的分裂聚类。
  - 聚合聚类以AGNES、BIRCH、ROCK等算法为代表，分裂聚类以DIANA算法为代表。

聚类的分类

- 基于密度的聚类
  - 基于划分聚类和基于层次聚类的方法在聚类过程中根据距离来划分类簇，只能用于挖掘球状簇，但往往现实中会有各种形状。
  - 基于密度聚类算法利用密度思想，将样本中的高密度区域(即样本点分布稠密的区域)划分为簇，将簇看作是样本空间中被稀疏区域(噪声)分隔开的稠密区域。
  - 根据密度而不是距离来计算样本相似度，所以能够用于挖掘任意形状的簇，并且能够有效过滤掉噪声样本对于聚类结果的影响。
  - 常见的基于密度的聚类算法有DBSCAN、OPTICS、DENCLUE等。

聚类的分类

- 基于网格的聚类
  - 基于密度的算法一般时间复杂度较高，网格方法可以有效减少算法的计算复杂度，且同样对密度参数敏感。
  - 基于网格的聚类通常将数据空间划分成有限个单元的网格结构，所有的处理都是以单个的单元为对象。这样做起来处理速度很快，因为这与数据点的个数无关，而只与单元个数有关。
  - 代表算法有：STING，CLIQUE，WaveCluster。
- 基于模型的聚类
  - 基于模型的方法（Model-based methods）主要是指基于概率模型的方法和基于神经网络模型的方法。
  - 代表算法：高斯混合模型GMM、自组织神经网络SOM等。

outline

Kmeans

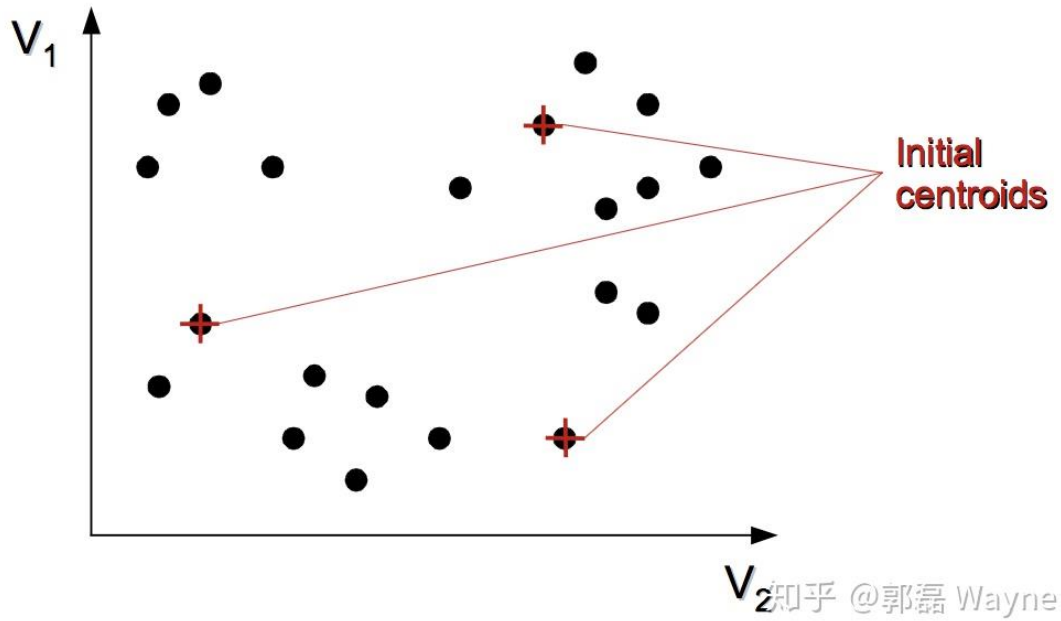
- k-均值聚类是基于划分的聚类算法，计算样本点与类簇质心的距离，与类簇质心相近的样本点划分为同一类簇。
- k-均值通过样本间的距离来衡量它们之间的相似度，两个样本距离越远，则相似度越低，否则相似度越高。

Kmeans 算法原理

- （1）初始化 K 个质心。初始质心随机选择即可，每一个质心为一个类。
- （2）按照样本与质心的距离对样本进行聚类。对剩余的每个样本点，计算它们到各个质心的欧式距离，并将其归入到相互间距离最小的质心所在的簇。
- （3）计算上一步聚类结果对应的新质心。在所有样本点都划分完毕后，根据划分情况重新计算各个簇的质心所在位置。
- 重复（2）和 （3），直到质心不再发生变化时或者到达最大迭代次数时。最终就确定了每个样本所属的类别以及每个类的质心。

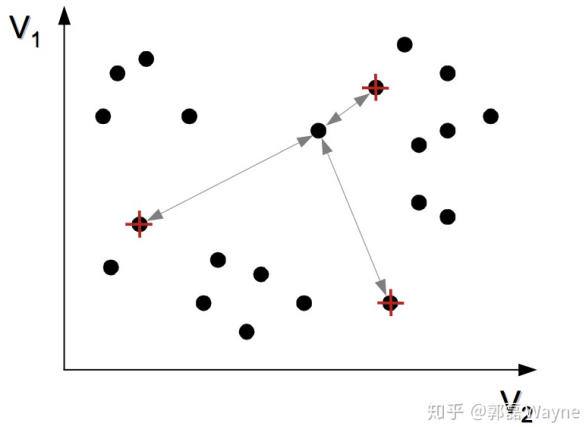
Kmeans 算法原理

- 有一个二维空间的一些点，我们要将它们分成3个类，即K=3。
- 首先随机选择3个初始质心，每一个质心为一类：

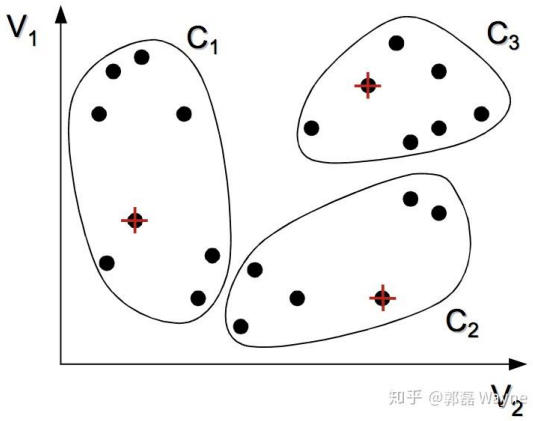


Kmeans 算法原理

- 然后计算每一个不是质心的点到这三个质心的距离：

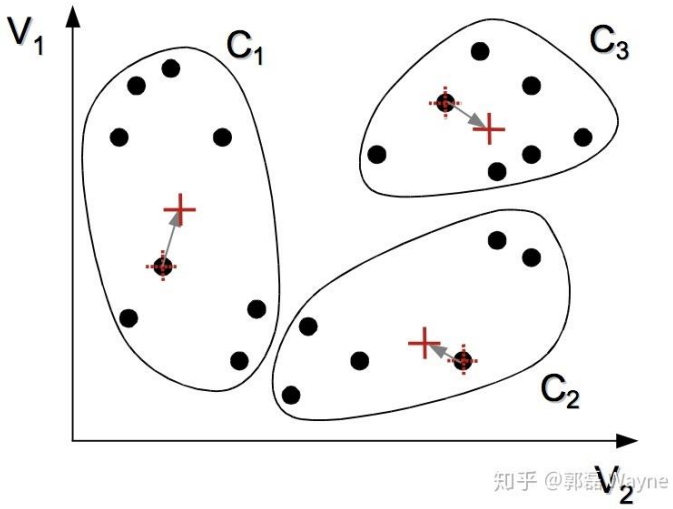


- 将这些点归类于距离最近的那个质心的一类：

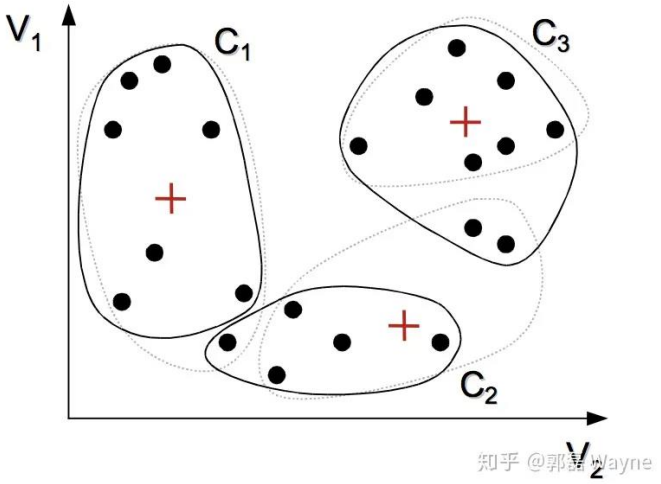


Kmeans 算法原理

- 重新计算这三个分类的质心：

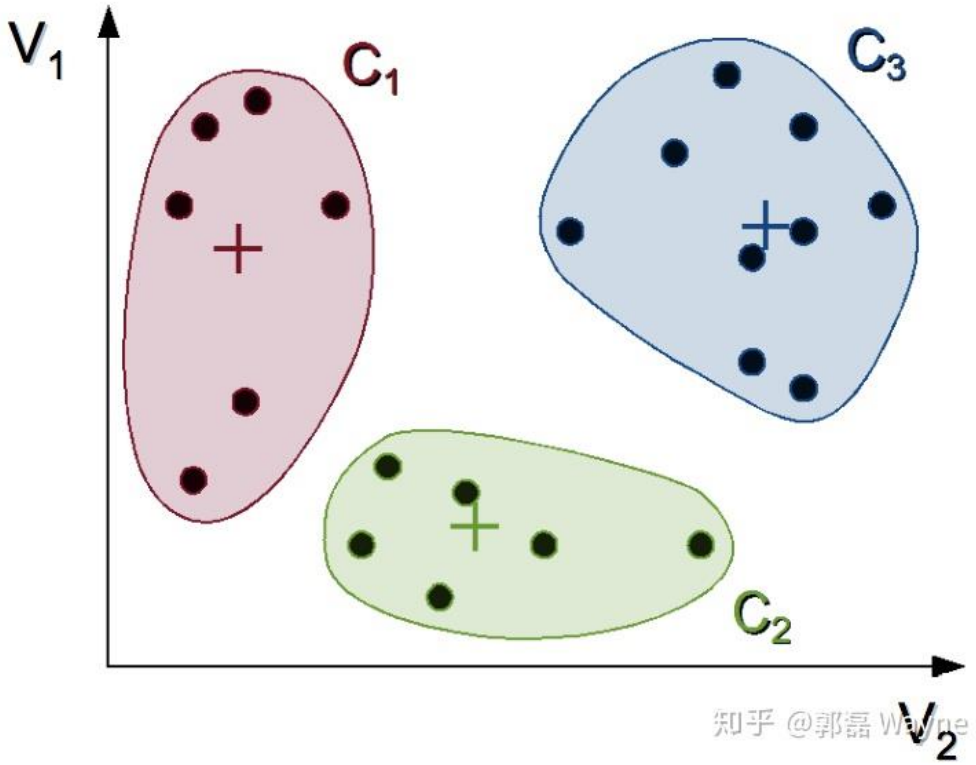


- 不断重复上述两步，更新三个类：



Kmeans 算法原理

- 当稳定以后，迭代停止，这时候的三个类就是得到的最后类簇：



Kmeans 数学推导

- 假设使用欧氏距离作为Kmeans的距离度量方式，则样本之间的距离为：

$$\boldsymbol{d_{ij}} = \left(\sum_{\boldsymbol{z}=1}^{\boldsymbol{d}} |\boldsymbol{x_{iz}} - \boldsymbol{x_{jz}}|^{2} \right)^{\frac{1}{2}} \qquad x_i = (x_{i1}, x_{i2}, \cdots, x_{i\boldsymbol{d}})$$

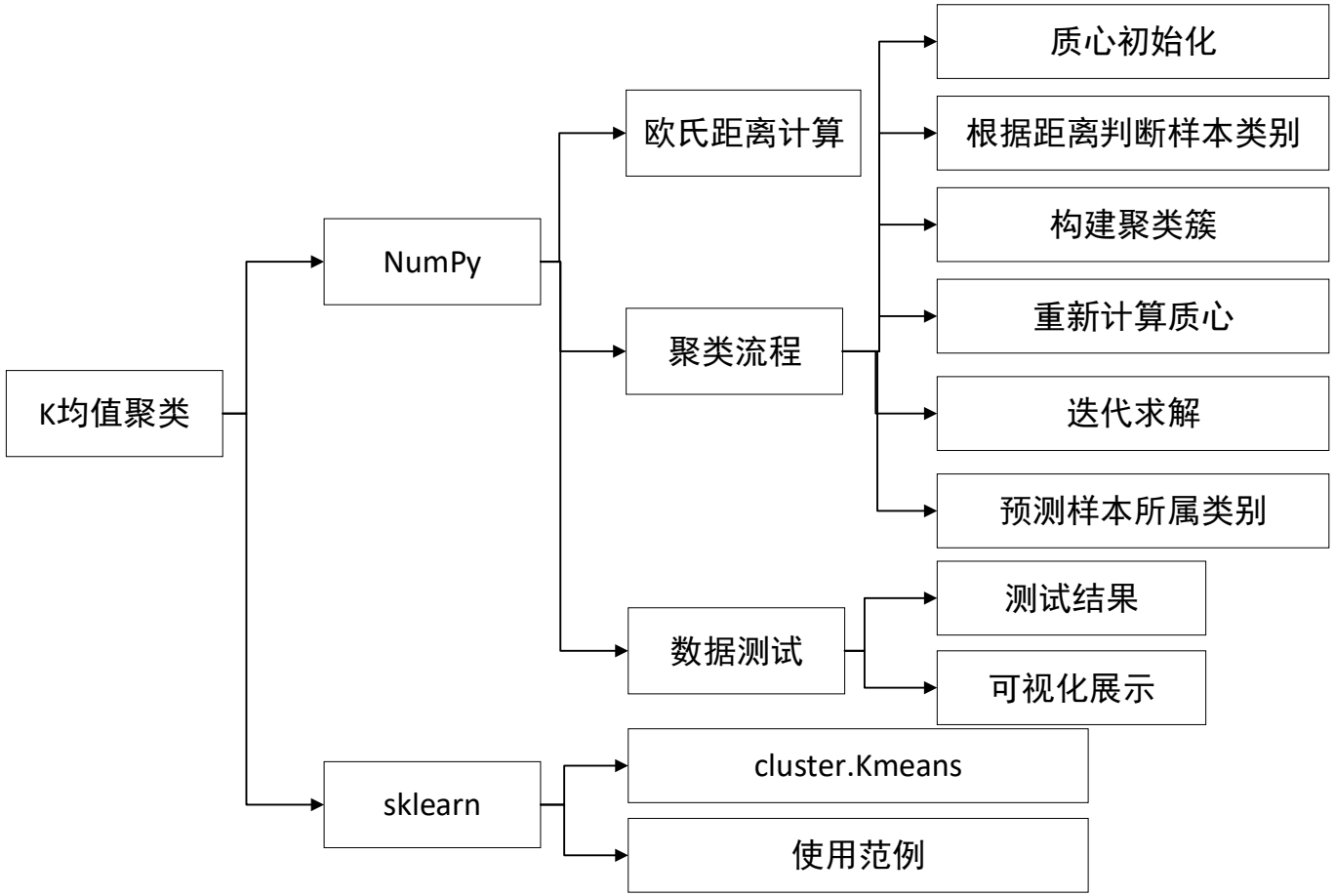
- 定义样本与其所属类中心的距离总和为最终损失函数：

$$\boldsymbol{L(C)} = \sum_{\boldsymbol{c}=1}^{\boldsymbol{k}} \sum_{\boldsymbol{x_i} \in \boldsymbol{C_c}} ||\boldsymbol{x_i} - \boldsymbol{u_c}||_2^2 \qquad \boldsymbol{u_c} = \frac{1}{\boldsymbol{n_c}} \sum_{\boldsymbol{x_i} \in \boldsymbol{C_c}} \boldsymbol{x_i} \quad \text{质心}$$

- 最终目标：

$$\boldsymbol{C}^* = \boldsymbol{arg} \min_{\boldsymbol{C}} \boldsymbol{L(C)}$$

Kmeans 算法实现


K均值聚类
NumPy
sklearn
欧氏距离计算
聚类流程
数据测试
质心初始化
根据距离判断样本类别
构建聚类簇
重新计算质心
迭代求解
预测样本所属类别
测试结果
可视化展示
cluster.Kmeans
使用范例

```python
import numpy as np
# 定义欧式距离
def euclidean_distance(x1, x2):
    distance = 0
    # 距离的平方项再开根号
    for i in range(len(x1)):
        distance += pow((x1[i] - x2[i]), 2)
    return np.sqrt(distance)
```

```python
# 定义中心初始化函数
def centroids_init(k, X):
    m, n = X.shape
    centroids = np.zeros((k, n))
    for i in range(k):
        # 每一次循环随机选择一个类别中心
        centroid = X[np.random.choice(range(m))]
        centroids[i] = centroid
    return centroids
```

```python
# 定义样本的最近质心点所属的类别索引
def closest_centroid(sample, centroids):
    closest_i = 0
    closest_dist = float('inf')
    for i, centroid in enumerate(centroids):
        # 根据欧式距离判断，选择最小距离的中心点所属类别
        distance = euclidean_distance(sample, centroid)
        if distance < closest_dist:
            closest_i = i
            closest_dist = distance
    return closest_i
```

```python
# 定义构建类别过程
def build_clusters(centroids, k, X):
    clusters = [[] for _ in range(k)]
    for x_i, x in enumerate(X):
        # 将样本划分到最近的类别区域
        centroid_i = closest_centroid(x, centroids)
        clusters[centroid_i].append(x_i)
    return clusters
```

```python
# 根据上一步聚类结果计算新的中心点
def calculate_centroids(clusters, k, X):
    n = X.shape[1]
    centroids = np.zeros((k, n))
    # 以当前每个类样本的均值为新的中心点
    for i, cluster in enumerate(clusters):
        centroid = np.mean(X[cluster], axis=0)
        centroids[i] = centroid
    return centroids
```

```python
# 获取每个样本所属的聚类类别
def get_cluster_labels(clusters, X):
    y_pred = np.zeros(X.shape[0])
    for cluster_i, cluster in enumerate(clusters):
        for X_i in cluster:
            y_pred[X_i] = cluster_i
    return y_pred
```

```python
# 根据上述各流程定义kmeans算法流程
def kmeans(X, k, max_iterations):
    # 1.初始化中心点
    centroids = centroids_init(k, X)
    # 遍历迭代求解
    for _ in range(max_iterations):
        # 2.根据当前中心点进行聚类
        clusters = build_clusters(centroids, k, X)
        # 保存当前中心点
        prev_centroids = centroids
        # 3.根据聚类结果计算新的中心点
        centroids = calculate_centroids(clusters, k, X)
        # 4.设定收敛条件为中心点是否发生变化
        diff = centroids - prev_centroids
        if not diff.any():
            break
    # 返回最终的聚类标签
    return get_cluster_labels(clusters, X)
```

```python
# 测试数据
X = np.array([[0,2],[0,0],[1,0],[5,0],[5,2]])
# 设定聚类类别为2个，最大迭代次数为10次
labels = kmeans(X, 2, 10)
# 打印每个样本所属的类别标签
print(labels)
```

```python
from sklearn.datasets import load_iris

iris = load_iris()
labels = kmeans(iris.data, 3, 100)
print(labels)
print(iris.target)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 2. 1. 2. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 2. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 2. 1. 2. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 1. 1. 2. 2. 2. 2. 1.
 2. 1. 2. 1. 2. 2. 1. 1. 2. 2. 2. 2. 2. 1. 2. 2. 2. 2. 1. 2. 2. 2. 1. 2.
 2. 2. 1. 2. 2. 1.]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

K值确定方法

- （1）拍脑袋法
- （2）肘部法则（Elbow Method）
- （3）间隔统计量（Gap Statistic）
- （4）轮廓系数（Silhouette Coefficient）
- （5）Canopy算法

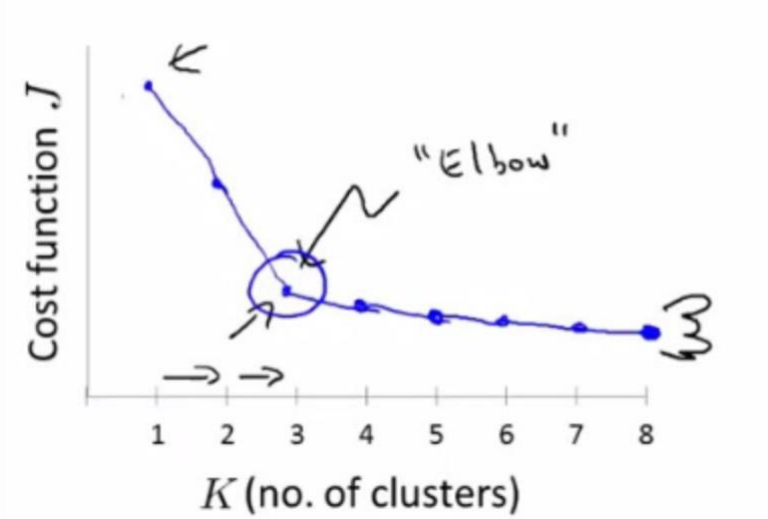
拍脑袋法

- 一个非常快速的，拍脑袋的方法是将样本量除以2再开方出来的值作为K值，具体公式为：

$$K \approx \sqrt{n/2}$$

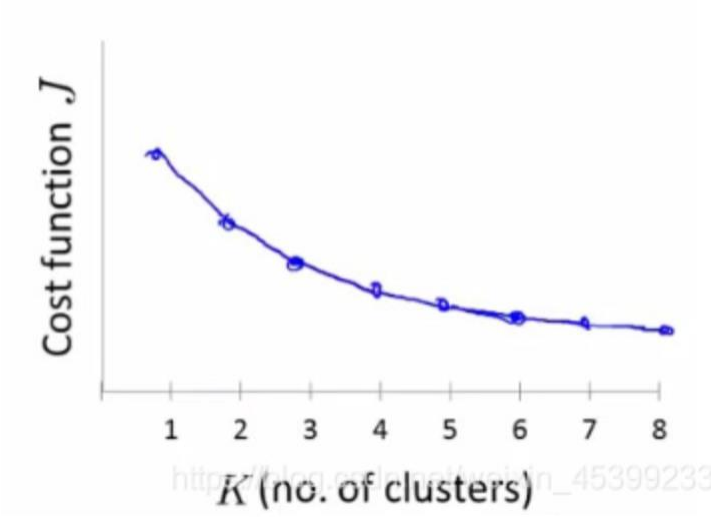
肘部法则（Elbow Method）

- 此种方法适用于 K 值相对较小的情况，当选择的K值小于最优值时，K每增加1，损失就会大幅的减小；当选择的K值大于最优值时，K每增加1，损失值的变化就不会那么明显。这样，最优K值就会在这个转折点，类似elbow的地方。

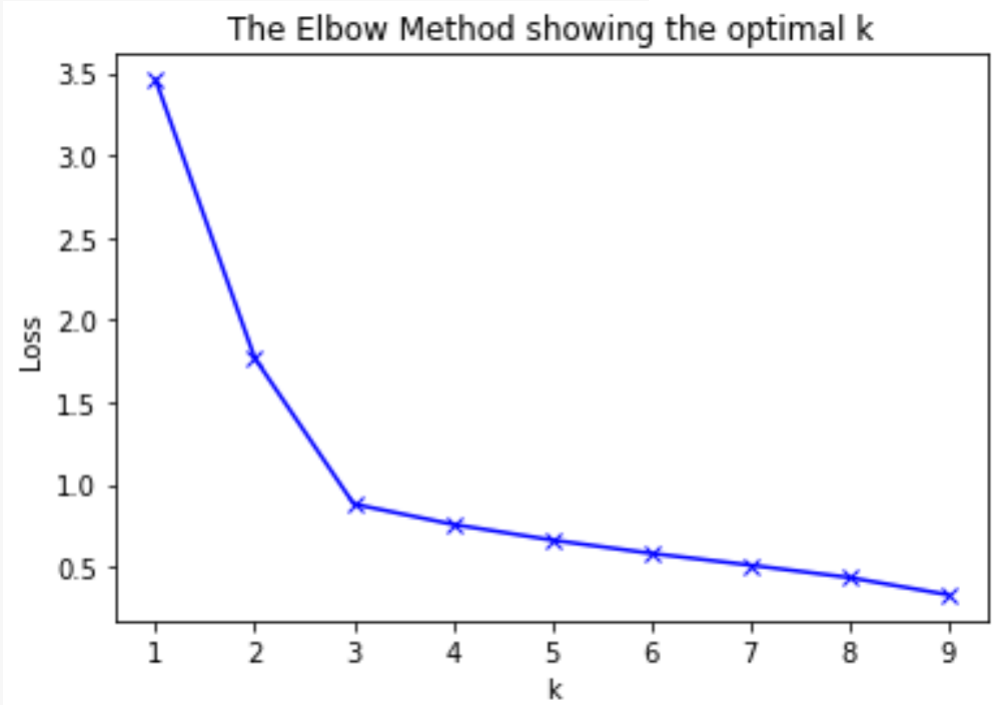


$$D_k = \sum_{i=1}^{K} \sum dist(x, c_i)^2$$

- 并不是所有的问题都可以通过画肘部图来解决，当肘点位置不明显（如图中肘点可以是3，4，5），这时就无法确定K值了。故肘部图是可以尝试的一种方法，但是并不是对所有的场景都有效。



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

x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 5, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])

plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)

# k means determine k
lossList = []
KList = range(1, 10)
for k in KList:
    kmeanModel = KMeans(n_clusters=k).fit(X)
    kmeanModel.fit(X)
    lossList.append(sum(np.min(cdist(X, kmeanModel.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0])

# Plot the elbow
plt.plot(KList, lossList, 'bx-')
plt.xlabel('k')
plt.ylabel('Loss')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



轮廓系数（Silhouette Coefficient）

- 轮廓系数衡量对象和所属簇之间的相似度，即内聚性（cohesion）。当把它与其他簇做比较，就称为分离性（separation）。
- 该对比通过轮廓系数值来实现，后者在 [-1, 1] 范围内。轮廓系数值接近 1，说明对象与所属簇之间有密切联系；反之则接近 -1。
- 若某聚类模型生成的数据簇，计算得到比较高的轮廓系数值，说明该模型是合适、可接受的。

轮廓系数（Silhouette Coefficient）

- 1）计算样本i到同簇其他样本的平均距离 a(i)。a(i) 越小，说明样本 i 越应该被聚类到该簇。
  - 将 a(i) 称为样本i的簇内不相似度。
  - 簇 C 中所有样本的 a(i) 均值称为簇 C 的簇不相似度。
- 2）计算样本 i 到其他某簇 C(j) 的所有样本的平均距离 b(ij)，称为样本 i 与簇 C(j) 的不相似度。
  - 样本 i 的簇间不相似度记为 b(i) = min{b(i1), b(i2), …, b(ik)}。
  - b(i) 越大，说明样本 i 越不属于其他簇。

轮廓系数（Silhouette Coefficient）

- 3）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：

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

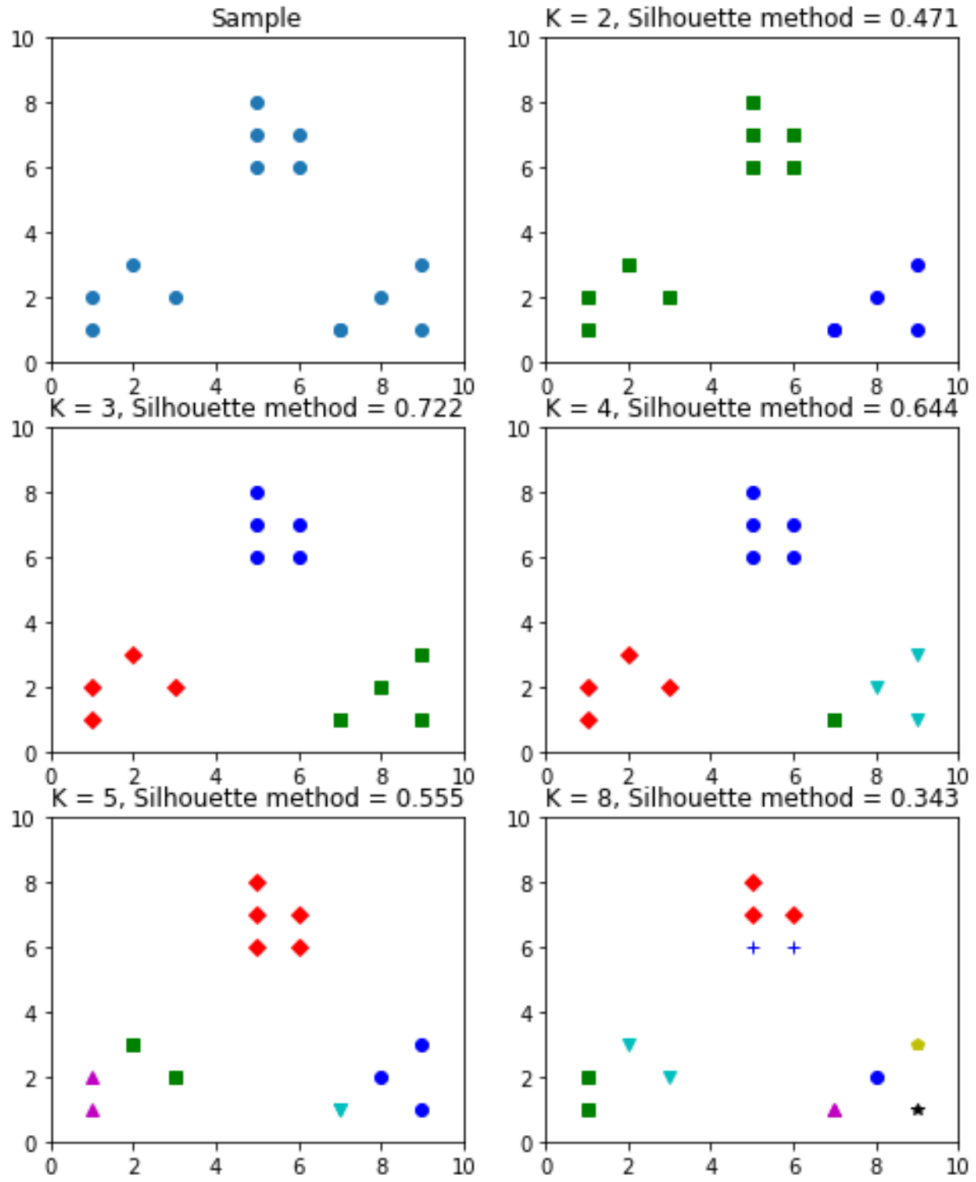
$$s(i)=\begin{cases}1-\dfrac{a(i)}{b(i)}, & a(i)<b(i)\\ 0, & a(i)=b(i)\\ \dfrac{b(i)}{a(i)}-1, & a(i)>b(i)\end{cases}$$

- 4）判断：
  - s(i) 接近1，则说明样本 i聚类合理。
  - s(i) 接近-1，则说明样本 i更应该分类到另外的簇。
  - 若 s(i) 近似为0，则说明样本 i 在两个簇的边界上。

轮廓系数（Silhouette Coefficient

- 所有样本的 s(i) 的均值称为聚类结果的轮廓系数，是该聚类是否合理、有效的度量。
- 但是，其缺陷是计算复杂度为 O(n^2)，需要计算距离矩阵，那么当数据量上到百万，甚至千万级别时，计算开销会非常巨大。

```python
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 10))
plt.subplot(3, 2, 1)
x1 = np.array([1, 2, 3, 1, 5, 6, 5, 5, 6, 7, 8, 9, 7, 9])
x2 = np.array([1, 3, 2, 2, 8, 6, 7, 6, 7, 1, 2, 1, 1, 3])
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Sample')
plt.scatter(x1, x2)
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'b']
markers = ['o', 's', 'D', 'v', '^', 'p', '*', '+']
tests = [2, 3, 4, 5, 8]
subplot_counter = 1
for t in tests:
    subplot_counter += 1
    plt.subplot(3, 2, subplot_counter)
    kmeans_model = KMeans(n_clusters=t).fit(X)
    for i, l in enumerate(kmeans_model.labels_):
        plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
        plt.xlim([0, 10])
        plt.ylim([0, 10])
        plt.title('K = %s, Silhouette method = %.03f' % (t, metrics.silhouette_score(X, kmeans_model.labels_,metric='euclidean')))
plt.show()
```



Kmeans 算法优点

- 原理简单，容易理解，聚类效果中上；
- 可解释度较强；
- 收敛速度快；
- 当簇近似高斯分布的时候，效果非常不错。

Kmeans 算法缺点

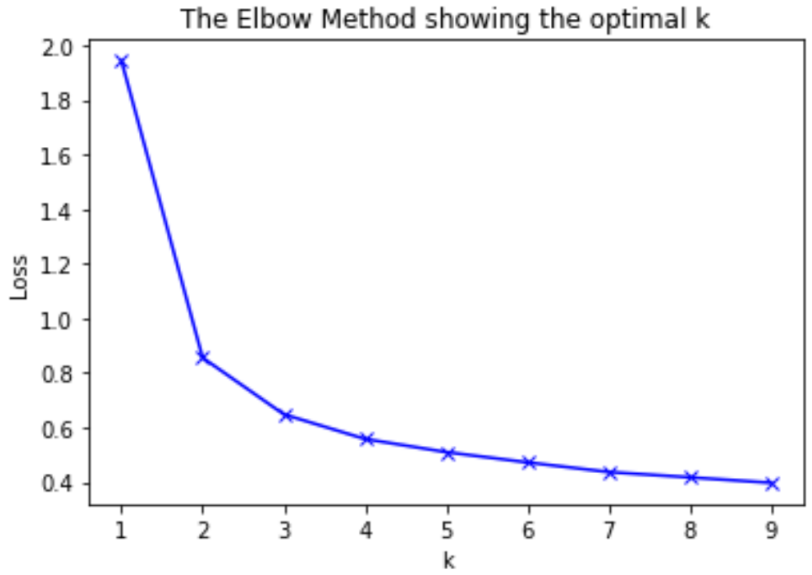
- K 值难确定，不同 K 值得到的结果不一样；
- 对初始的簇中心敏感，聚类效果依赖于聚类中心的初始化；
- 对噪声和异常值（离群点）敏感；
- 样本只能归为一类，不适合多分类任务；
- 可处理的数据类型有限，对于高维数据对象的聚类效果不佳；
- 对于非凸数据集或类别规模差异太大的数据效果不好。

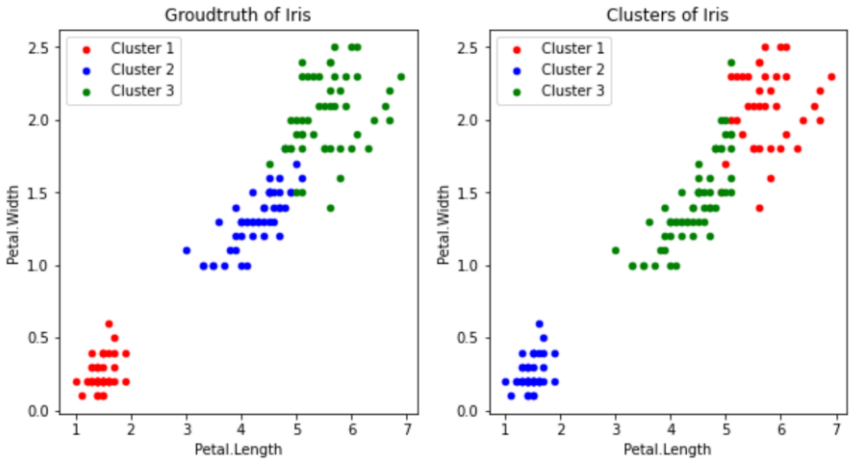
Kmeans Vs kNN

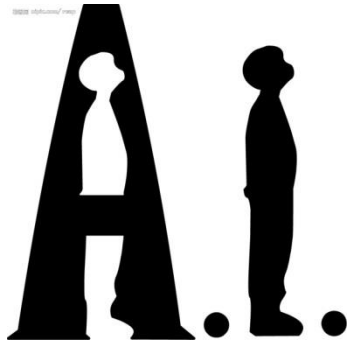
- KMeans是无监督学习算法，KNN是监督学习算法。
- KMeans算法的训练过程需要反复迭代的操作（寻找新的质心），但是KNN不需要（惰性算法）。
- KMeans中的K代表的是簇中心，KNN的K代表的是选择与新测试样本距离最近的前K个训练样本数。

基于scikit-learn的模型实现

- 课后作业：基于sklearn的鸢尾花聚类分析
- from sklearn.datasets import iris
- 要求：
  - 使用所有的数据做聚类；
  - 根据肘部规则方法确定K值；
  - 调用KMeans模型，参数init = ‘k-means++‘，表示可以避免选择较差起始点；
  - 以花瓣长度为x轴，花瓣宽度为y轴，绘制真实散点图和聚类散点图。







Thank you!