实验三　K-means 聚类算法实验

胡祝华 2022.12.12

1　实验目的 2 学时

1.理解聚类的基本概念，分类和聚类的区别，有监督聚类和无监督聚类的区别。

2.理解并掌握 K-means 聚类算法的基本原理

3.学会用 python 实现 K-means 聚类算法

2　实验环境

软件环境：Windows 7 以上版本的操作系统

开发环境：（1）Python 3.6 以上版本

（2）Anaconda3 集成环境

（3）Pycharm IDE

硬件环境：PC 机

3　实验原理

3.1 聚类分析简介

聚类分析是一种典型的无监督学习，用于对未知类别的样本进行划分。将它们按照一定的规则划分成若干个类簇，把相似（距高相近）的样本聚在同一个类簇中，把不相似的样本分为不同类簇，从而揭示样本之间内在的性质以及相互之间的联系规律。聚类分析中，组内相似性越大，组间差距越大，说明聚类效果越好。聚类的目标就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。

3.2 K-means 算法简介

K-means 算法是基于距离的聚类算法，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。其中 k 代表类簇个数，means 代表类簇内数据对象的均值。K-means 算法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。数据对象间距离的计算有很多种，K-means 算法通常采用欧式距离来计算数据对象间的距离。

欧式距离：衡量 m 维空间中两个点之间的真实距离。

$$dist(X,Y)=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

3.3 K-means 工作步骤

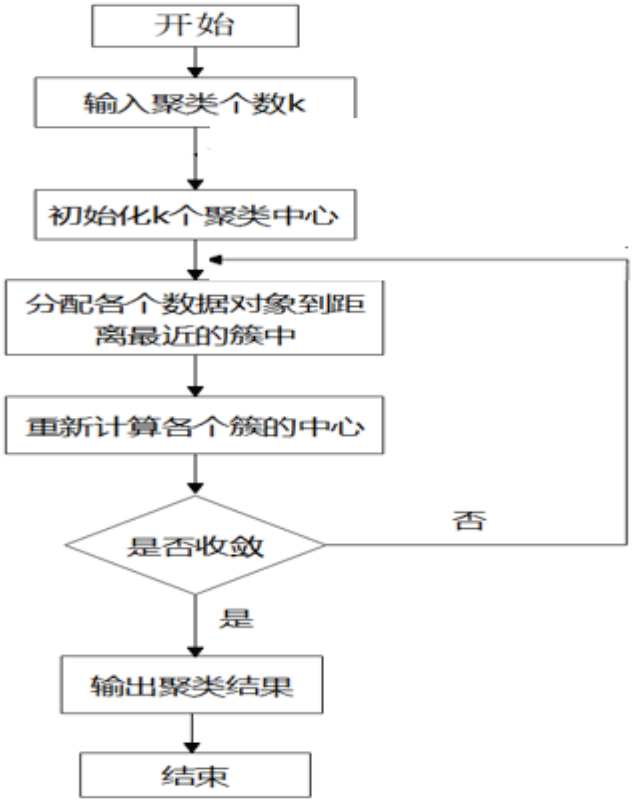
（1）随机选择 K 个对象，每个对象代表一个簇的初始均值或质心；

（2）对剩余的每个对象，计算它们到各簇质心的欧式距离，并将其归入到相互间距离最近的质心所在的簇；

（3）计算簇中所有点的均值并将均值作为更新后的簇质心；

（4）不断重复（2）（3）步骤，直到准则函数收敛。

K-means 算法流程图如下所示：



4　实验内容

注意：和前两次实验类似的做法：（1） 在实验中要求把模型和测试解耦开。即模型是一个.py 文件，对 k-means 模型的测试是另外一个.py 文件。（2）参考前两次实验，要求将给出的函数封装在一个 class 中，类名为 MyKmeans。

4.1 实验步骤

（1）导入工具库

```
# 导入标准库
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

（2）定义一个函数计算欧式距离

```
# 定义函数：计算欧式距离
def euclDistance(point1, point2):
    # 计算两点point1、point2之间的欧式距离
    distance = np.sqrt(sum(pow(point2-point1,2)))
    return distance
```

（3）定义函数：初始化质心

```
def initCentroids(dataSet, k):
    # dataSet为数据集
    # k是指用户设定的k个簇
    numSamples, dim = dataSet.shape # numSample: 数据集数量; dim: 特征维度
    centroids = np.zeros((k, dim))  # 存放质心坐标，初始化k行、dim列零矩阵
    for i in range(k):
        #index = int(np.random.uniform(0, numSamples))  # 给出一个服从均匀分布的在0~numSamples之间的整数
        index = np.random.randint(0, numSamples)  # 给出一个随机分布在0~numSamples之间的整数
        centroids[i, :] = dataSet[index, :]  # 第index行作为质心
    return centroids
```

（4）定义核心函数实现 kmeans 聚类

```
def kmeans(dataSet, k):
    # dataSet 为数据集
    # k 是指用户设定的 k 个簇
    numSamples = dataSet.shape[0]
    clusterAssment = np.zeros((numSamples, 2)) #clusterAssment 第 1 列存放所属的簇，第 2 列存放与质心的距离
    clusterChanged = True #clusterChanged=False 时迭代更新终止

    ## step 1: 初始化质心 centroids
    centroids = initCentroids(dataSet, k)
    # 循环体：是否更新质心
    while clusterChanged:
        clusterChanged = False #关闭更新
```

```
        # 对每个样本点
        for i in range(numSamples):
            minDist = 100000.0  # 最小距离
            minIndex = 0  # 最小距离对应的簇
            ## step2: 找到距离每个样本点最近的质心
            # 对每个质心
            for j in range(k):
                distance = euclDistance(centroids[j, :], dataSet[i, :])  # 计算每个样本点到质心的欧式距离
                if distance < minDist:  # 如果距离小于当前最小距离 minDist
                    minDist = distance  # 最小距离更新
                    minIndex = j  # 样本所属的簇也会更新

            ## step 3: 更新样本所属的簇
            if clusterAssment[i, 0] != minIndex:  # 如当前样本不属于该簇
                clusterChanged = True  # 聚类操作需要继续
            clusterAssment[i, :] = minIndex, minDist

        ## step 4: 更新质心
        # 对每个质心
        for j in range(k):
            pointsInCluster = dataSet[np.nonzero(clusterAssment[:,0] == j)[0]]
            # pointsInCluster 存储的是当前所有属于簇 j 的 dataSet 样本点
            centroids[j, :] = np.mean(pointsInCluster, axis=0)  # 更新簇 j 的质心

    print("cluster complete!")
    return centroids, clusterAssment
```

（5）定义函数：选择不同的 K 值进行交叉验证

```
def selectK(dataSet, k_list):
    # dataSet为数据集
    # k_list不同k值列表
    distanceK = [] #存储不同k值下每个样本点到质心的平均欧式距离
    for i, k in enumerate(k_list):
        centroids, clusterAssment = kmeans(dataSet, k) #调用kmeans函数
        distance = np.mean(clusterAssment[:,1], axis=0) # clusterAssment所有minDist的平均
        distanceK.append(distance)

    #best_k = k_list[np.argmin(distanceK)] #能够让距离最小的k值

    return distanceK
```

(6) 定义函数：作图可视化 kmeans 聚类效果

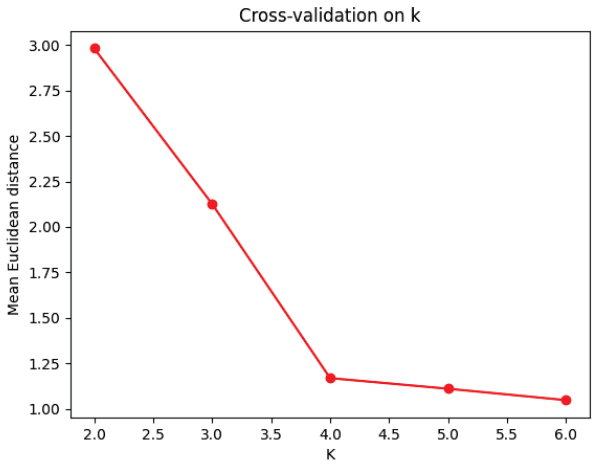
```
def showCluster(dataSet, k, centroids, clusterAssment):
    # dataSet 为数据集
    # k 是指用户设定的 k 个簇
    # centroids 存放质心坐标
    # clusterAssment 第 1 列存放所属的簇，第 2 列存放与质心的距离
    numSamples, dim = dataSet.shape  # numSample: 数据集数量；dim: 特征维度
    if dim != 2:
        print("The dimension of data is not 2!")
        return 1

    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']
    if k > len(mark):
        print("K is too large!")
        return 1
    # 画所有的样本
    plt.figure()
    for i in range(numSamples):
        markIndex = int(clusterAssment[i, 0])
        plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])

    mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']
    # 画所有的质心
```

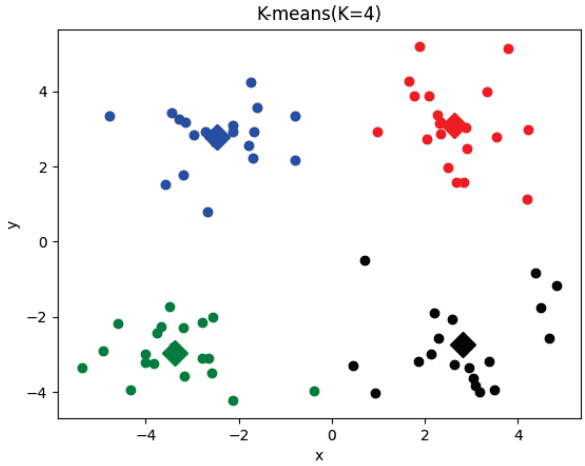
```
        for i in range(k):
            plt.plot(centroids[i, 0], centroids[i, 1], mark[i], ms=12.0)
        plt.title('K-means(K={})'.format(k))
        plt.xlabel('x')
        plt.ylabel('y')
    plt.show()
```

(7) 对模型进行测试

```
# 导入数据集
dataSet = pd.read_csv('dataSet.csv')
dataSet = dataSet.values #(80,2)


# 选择不同的k值对比
k_list = [2, 3, 4, 5, 6]
disK = selectK(dataSet, k_list)
# 画图
plt.figure()
plt.plot(k_list, disK, 'ro-')
plt.title('Cross-validation on k')
plt.xlabel('K')
plt.ylabel('Mean Euclidean distance')
plt.show()


# 使用K-means算法进行聚类
k = 4
centroids, clusterAssment = kmeans(dataSet, k)
# 作图可视化kmeans聚类效果
showCluster(dataSet, k, centroids, clusterAssment)
```

4.2 实验结果



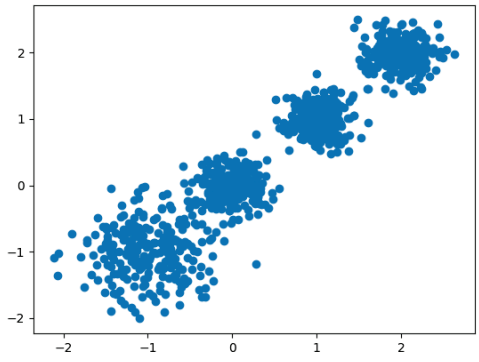


4.3 直接使用 sklearn 库里面封装好的 k-means 算法进行实验。

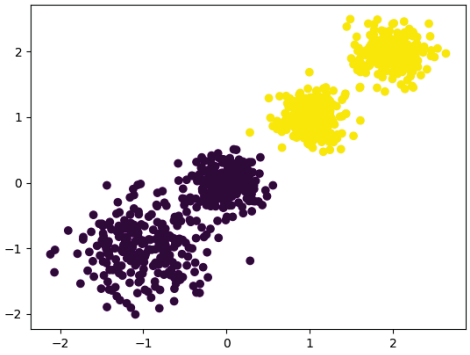
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# X 为样本特征，Y 为样本簇类别， 共 1000 个样本，每个样本 4 个特征，共 4 个簇，簇中心在[-1,-1], [0,0],[1,1], [2,2]， 簇方差分别为[0.4, 0.2, 0.2]
X, y = make_blobs(n_samples=1000, n_features=2, centers=[[-1, -1], [0, 0], [1, 1], [2, 2]],
                  cluster_std=[0.4, 0.2, 0.2, 0.2],
                  random_state=9)
plt.scatter(X[:, 0], X[:, 1], marker='o')
plt.show()
print(X)

# k = 2
y_pred = KMeans(n_clusters=2, random_state=9).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()

# k=3
y_pred = KMeans(n_clusters=3, random_state=9).fit_predict(X)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()


# k=4
y_pred = KMeans(n_clusters=4, random_state=9).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()
```

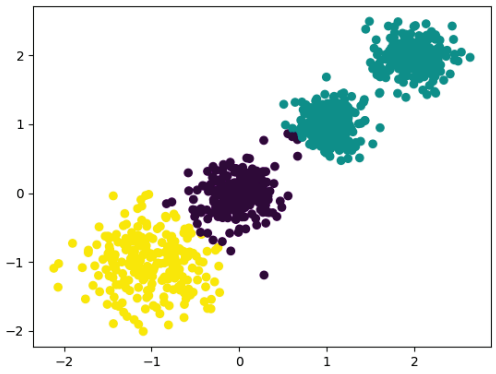
实验结果：



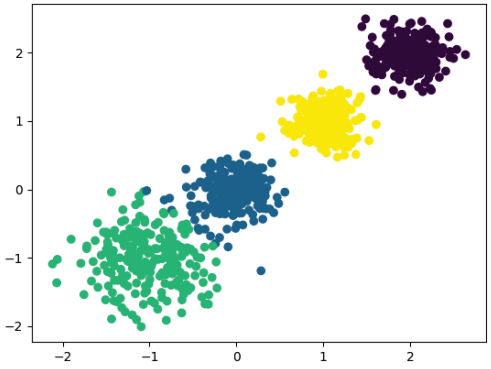
（a）数据集展示



（b）k=2



（a）k=3



（b）k=4

5　实验提升

（1）K-means 算法原理简单，容易实现，且运行效率比较高；聚类结果容易解释，适用于高维数据的聚类。

（2）K-means 算法采用贪心策略，容易导致局部收敛，在大规模数据集上求解较慢；对离群点和噪声点非常敏感，少量的离群点和噪声点可能对算法求平均值产生极大影响，从而影响聚类结果。

（3）K-means 算法中初始聚类中心的选取也对算法结果影响很大，不同的初始中

心可能会导致不同的聚类结果。

（4）关于 k 值的选取

a.手肘法：核心指标是 SSE（误差平方和）

$$SSE=\sum_{i=1}^{k}\sum_{p\in C_i}\left|p-m_i\right|^2$$

其中，C_i 是第 i 个簇，p 是 C_i 中的样本点，m_i 是 C_i 的质心（C_i 中所有样本的均值），SSE 是所有样本的聚类误差，代表了聚类效果的好坏。

手肘法的核心思想是：随着类簇个数 k 的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和 SSE 会逐渐变小。并且，当 k 小于真实聚类数时，由于 k 的增大会大幅增加每个簇的聚合程度，所以 SSE 的下降幅度会很大，而当 k 到达真实聚类数时，再增加 k 所得到的聚合程度会迅速变小，所以 SSE 的下降幅度会骤减，然后随着 k 值的继续增大而趋于平缓，也就是说 SSE 和 k 的关系图是一个手肘的形状，而这个肘部对应的 k 值就是数据的真实聚类数。

b.轮廓系数法：核心指标是轮廓系数，某个样本点 X_i 的轮廓系数定义如下：

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

其中，a 是 X_i 与同簇的其他样本的平均距离，称为凝聚度，b 是 X_i 与最近簇中所有样本的平均距离，称为分离度。而最近簇的定义是：

$$C_j=\arg\min_{C_k}\frac{1}{n}\sum_{p\in C_k}\left|p-X_i\right|^2$$

其中 p 是某个簇 C_k 中的样本。事实上，就是用 X_i 到某个簇所有样本平均距离作为衡量该点到该簇的距离后，选择离 X_i 最近的一个簇作为最近簇。求出所有样本的轮廓系数后再求平均值就得到了平均轮廓系数。平均轮廓系数的取值范围为[-1,1]，且簇内样本的距离越近，簇间样本距离越远，平均轮廓系数越大，聚类效果越好。