哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 考查课

实验题目： k-means 聚类

学号：
姓名：

一、实验目的

实现一个 k-means 算法，对自己生成或 UCI 下载的数据集进行聚类，并评价聚类的效果。

二、实验要求及实验环境

①实验要求：

用高斯分布产生 k 个高斯分布的数据（不同均值和方差）（其中参数自己设定），用 k-means 聚类，测试效果；

应用：可以 UCI 上找一个简单问题数据，用你实现的模型进行聚类。

②实验环境：

Windows10; python 3.9.12; Visual Studio Code; Anaconda.

三、设计思想（本程序中的用到的主要算法及数据结构）

问题背景：

给定训练样本 $X=\{x_1,x_2,\cdots\cdots,x_m\}$，和划分聚类的数量 k，给出一个簇划分 $C = C_1, C_2, \cdots\cdots, C_k$, 使得该划分的平方误差 E 最小化，即使下式取最小值：

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||_2^2$$

其中，$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x_i$.

它是簇 C_i 的均值向量。E 刻画了簇内样本围绕簇的均值向量的紧密程度，E 越小簇内样本的相似度越高。

① K-means 算法聚类

输入：n 个样本的集合 X，以及簇的数目 K

输出：样本集合的聚类 C^*

步骤：

1. 随机初始化 K 个中心点 $(m_1^{(0)}, m_2^{(0)}, \cdots\cdots, m_k^{(0)})$；

2. 对样本点进行聚类。对固定的类中心 $(m_1^{(t)}, m_2^{(t)}, \cdots\cdots, m_k^{(t)})$，计算每个样本到类中心的距离，将每个样本指派到与其最近的中心的类中，构成聚类结果 $C^{(t)}$；

3. 计算新的类中心，对聚类结果 $C^{(t)}$ 计算当前各个类中的样本的均值，作为新的类中心 $(m_1^{(t+1)}, m_2^{(t+1)}, \cdots\cdots, m_k^{(t+1)})$；

4. 重复 2 和 3 步，直到类中心收敛.

```
# k_means 算法计算分类中心和标签
def k_means(trainset, k, epsilon=1e-4):
    center = np.zeros((k, trainset[0].shape[1]))
    for i in range(k):
        center[i,:] = trainset[0][np.random.randint(0, high = trainset[0].shape[0]), :]
    while True:
        distance = np.zeros(k)
        # 根据中心重新给每个点贴分类标签
        for i in range(trainset[0].shape[0]):
            for j in range(k):
                distance[j] = np.linalg.norm(trainset[0][i] - center[j, :])
            trainset[1][i] = np.argmin(distance)
        # 根据每个点新的标签计算它的中心
        new_center = np.zeros((k, trainset[0].shape[1]))
        count = np.zeros(k)
        # 对每个类的所有点坐标求和
        for i in range(trainset[0].shape[0]):
            new_center[int(trainset[1][i]), :] += trainset[0][i]
```

```
            count[int(trainset[1][i])] += 1
        # 对每个类的所有点坐标求平均值
        for i in range(k):
            if count[i] != 0:
                new_center[i, :] = new_center[i, :] / count[i]
        #用差值的二范数衡量精度
        if np.linalg.norm(new_center - center) < epsilon:
            break
        else:
            center = new_center
    return center
```

② 通过兰德指数判断聚类效果

兰德指数（Rand index）需要给定实际类别信息 C，假设 K 是聚类结果，a 表示在 C 与 K 中都是同类别的元素对数，b 表示在 C 与 K 中都是不同类别的元素对数，则兰德指数为：

$$RI = \frac{a+b}{C_2^{n_{samples}}}$$

其中数据集中可以组成的总元素对数，RI 取值范围为[0,1]，值越大意味着聚类结果与真实情况越吻合。不过对于随机结果，RI 并不能保证分数接近零。

```
# 通过计算兰德指数 RI 判断聚类效果
def compute_RI(y_true, y_pred):
    n = len(y_true)
    a, b = 0, 0
    for i in range(n):
        for j in range(i+1, n):
            if (y_true[i] == y_true[j]) & (y_pred[i] == y_pred[j]):
                a += 1
            elif (y_true[i] != y_true[j]) & (y_pred[i] != y_pred[j]):
                b += 1
    RI = (a + b) / (n*(n-1)/2)
    return RI
```

③ 通过轮廓系数判断聚类效果

轮廓系数（Silhouette Coefficient），是聚类效果好坏的一种评价方式。最佳值为 1，最差值为-1。接近 0 的值表示重叠的群集。负值通常表示样本已分配给错误的聚类，因为不同的聚类更为相似。

首先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：

计算簇内不相似度 a(i) ：i 向量到同簇内其他点不相似程度的平均值，体现凝聚度。

计算簇间不相似度 b(i) ：i 向量到其他簇的平均不相似程度的最小值，体现分离度。

那么第 i 个对象的轮廓系数就为：

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

所有样本的 s i 的均值称为聚类结果的轮廓系数，定义为 S，是该聚类是否合理、有效的度量。聚类结果的轮廓系数的取值在[-1,1]之间，值越大，说明同类样本相距约近，不同样本相距越远，则聚类效果越好。

```
# 通过计算轮廓系数 Silhouette Coefficient 判断聚类效果
def compute_SC(result_set, k):
    n = len(result_set[0])
    #sc_lst 存储每个样本点的轮廓系数
    sc_lst = np.zeros(n)
    for i in range(n):
        sum_lst = np.zeros(k)
        count_lst = np.zeros(k)
        for j in range(n):
            sum_lst[result_set[1][j]] = np.linalg.norm(result_set[0][j, :] - result_set[0][i, :])
            count_lst[result_set[1][j]] += 1
        ave_lst = np.zeros(k)
```

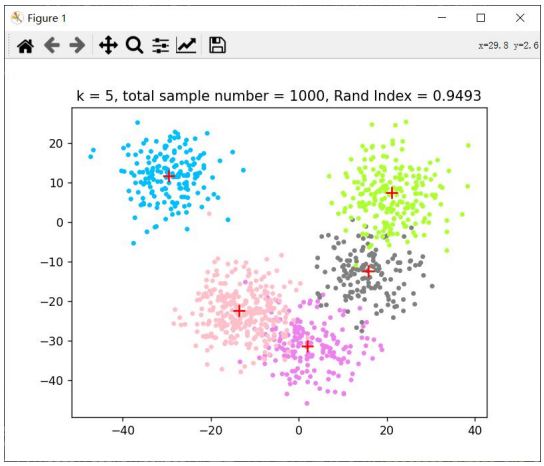
```
        ave_lst = sum_lst / count_lst
        # 当前点 i 与同簇内其他点的平均距离
        a = ave_lst[result_set[1][i]]
        # 当前点 i 与其他各个簇内点平均距离的最小值
        b = np.min(np.delete(ave_lst, result_set[1][i]))
        sc_lst[i] = (b - a) / max(a, b)
    return np.mean(sc_lst)
```

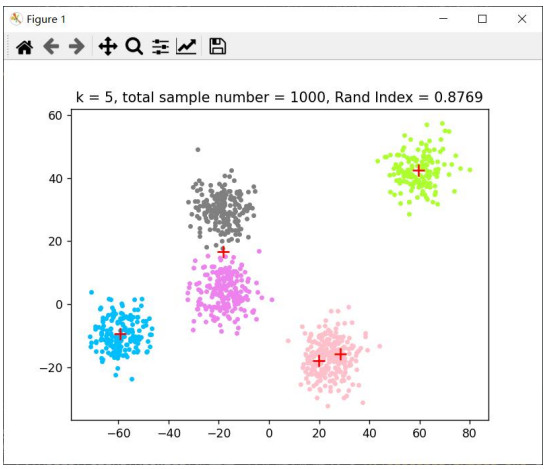
实验流程：

四、实验结果与分析

① 使用自己生成的数据集

1. 聚类数等于实际类别数时：





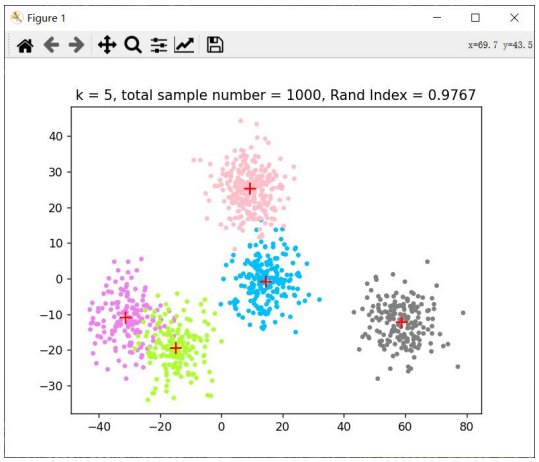


Figure 1
x=69.7 y=43.5
k = 5, total sample number = 1000, Rand Index = 0.9767

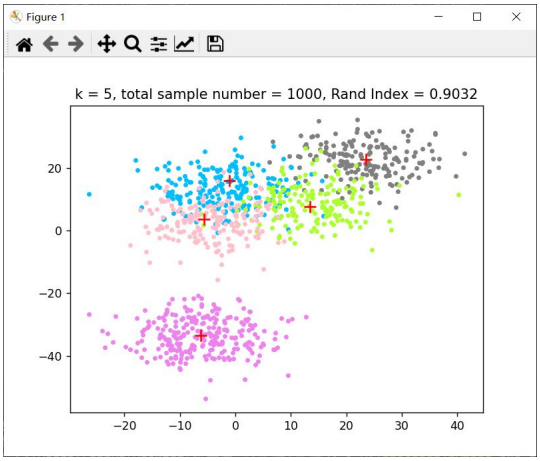


Figure 1
k = 5, total sample number = 1000, Rand Index = 0.9032

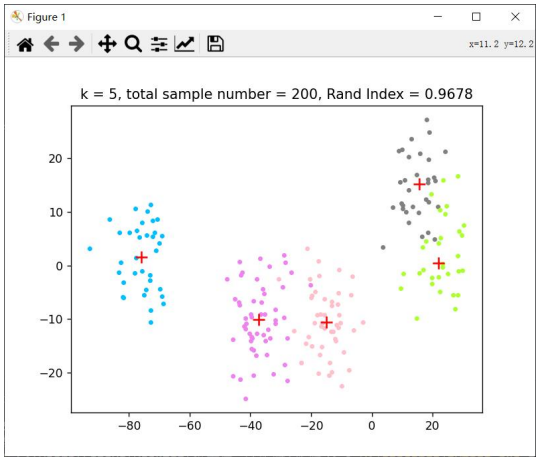


Figure 1
x=11.2 y=12.2
k = 5, total sample number = 200, Rand Index = 0.9678

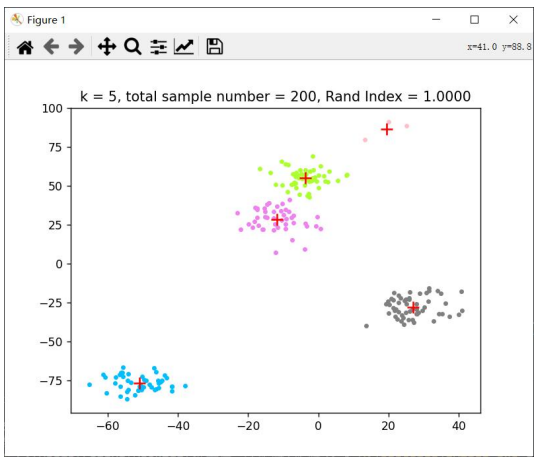


Figure 1
x=41.0 y=88.8
k = 5, total sample number = 200, Rand Index = 1.0000

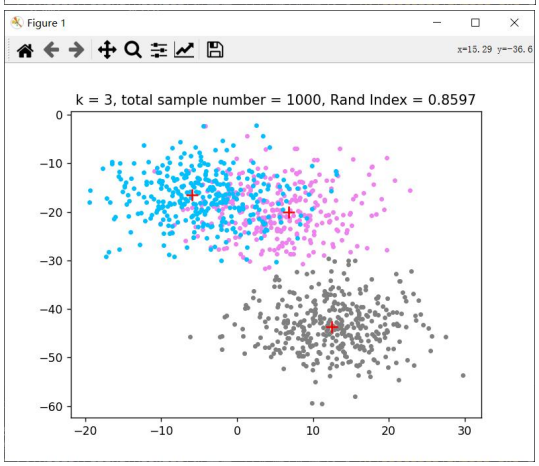


Figure 1
x=15.29 y=-36.6
k = 3, total sample number = 1000, Rand Index = 0.8597

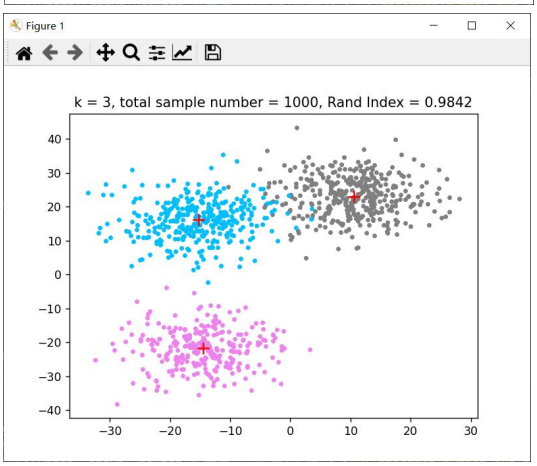


Figure 1
k = 3, total sample number = 1000, Rand Index = 0.9842

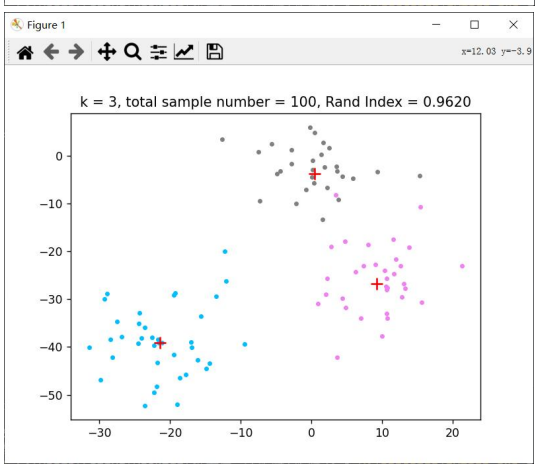


Figure 1
x=12.03 y=-3.9
k = 3, total sample number = 100, Rand Index = 0.9620

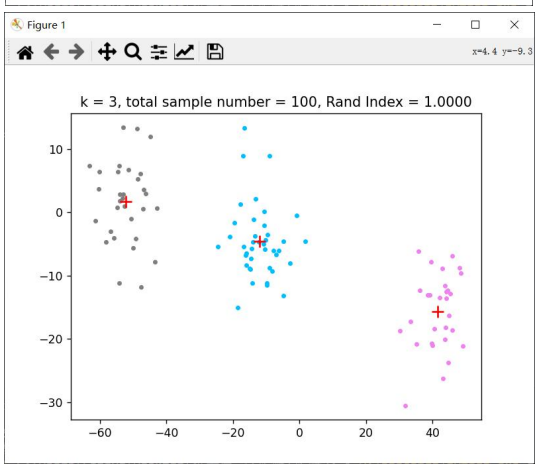
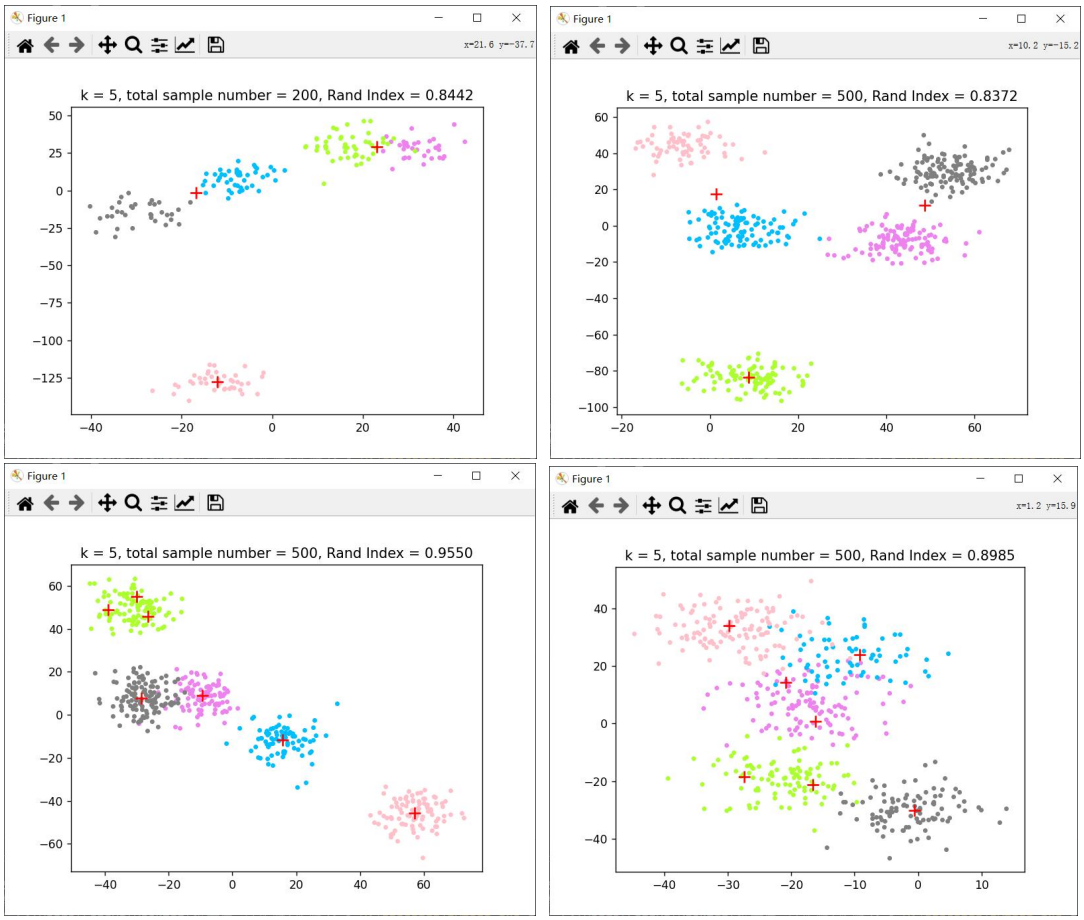
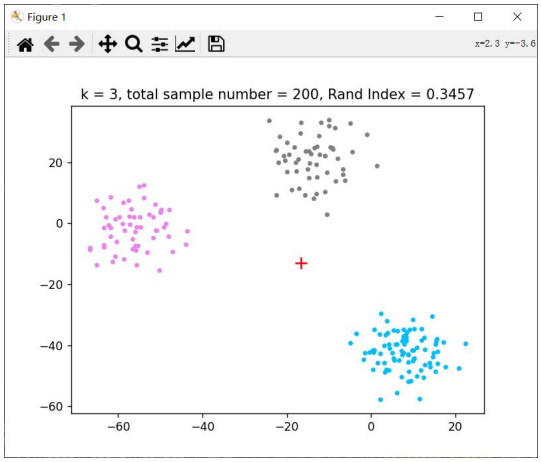


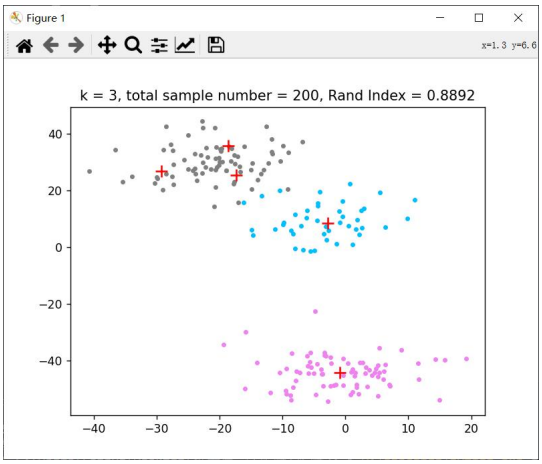
Figure 1
x=4.4 y=-9.3
k = 3, total sample number = 100, Rand Index = 1.0000

从中可以看出，当给定的聚类数等于实际类别数时，不论是样本量较小还是较大，聚类效果都相对较好，兰德指数都在 0.85 以上，很接近 1，与真实情况很吻合，即使有些情况下聚类样本本身不够分离也有不错的效果。但是由于 K-means 算法是随机选取初始聚类中心，因此有些时候部分聚类与真实情况会有不同，出现同一聚类出现两个中心、两个聚类共用一个中心等情况，导致兰德指数不够高，可以通过首先层次聚类选取初值的方法加以改进。这个问题本质上是因为 K-means 算法只会收敛到局部最优解导致的。

2. 聚类数不等于实际类别数时：







从中可以看出，当给定的聚类数不等于实际类别数时，即未知有几种类别的先验知识时，聚类效果有所下降，尤其是聚类数比实际类别数小的时候，兰德指数很低。

② 使用 UCI 下载的聚类数据集

1. iris 数据集聚类

数据集简介：

该数据集的 4 个属性：

1. sepal length in cm 2. sepal width in cm 3. petal length in cm 4. petal width in cm

可以被分类为以下 3 种聚类 class:-- Iris Setosa-- Iris Versicolour-- Iris Virginica

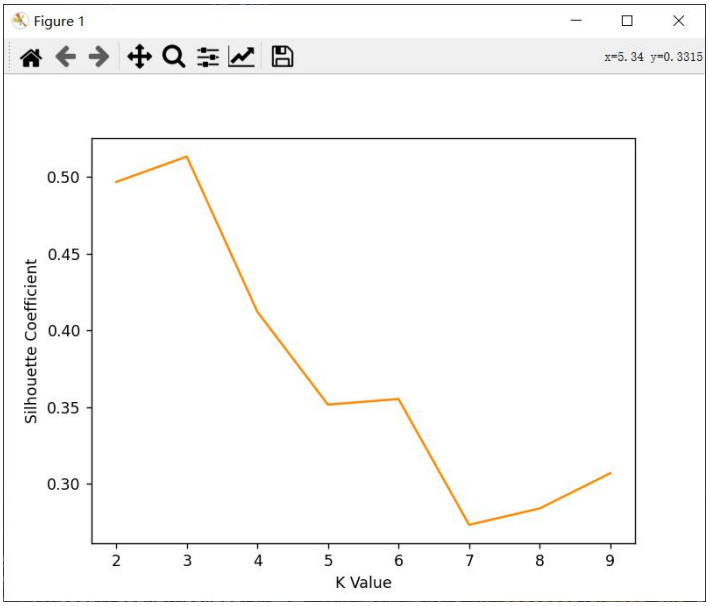
Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	4976324

部分数据概览：

iris.data

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
```

K-means 算法结果：



因为未知实际类别数量，通过一系列 K 值的尝试，根据聚类后的轮廓系数判断最佳 K 值。从中可以看出，当 K=3 时，聚类效果最好，说明应该把实际样本分类成 3 类，准确地判断出了与数据本来的结构。

2. 3D_spatial_network 数据集聚类

数据集简介：

This dataset was constructed by adding elevation information to a 2D road network in North Jutland, Denmark (covering a region of 185 x 135 km^2). Elevation values where extracted from a publicly available massive Laser Scan Point Cloud for Denmark (available at : [Web Link] (Bottom-most dataset)). This 3D road network was eventually used for benchmarking

various fuel and CO2 estimation algorithms. This dataset can be used by any applications that require to know very
accurate elevation information of a road network to perform more accurate routing for eco-routing, cyclist routes etc. For the data mining and machine learning community, this dataset can be used as 'ground-truth' validation in spatial mining techniques and satellite image processing. It has no class labels, but can be used in unsupervised learning and regression to guess some missing elevation information for some points on the road.
The work was supported by the Reduction project that is funded by the European Comission as FP7-ICT-2011-7 STREP project number 288254.

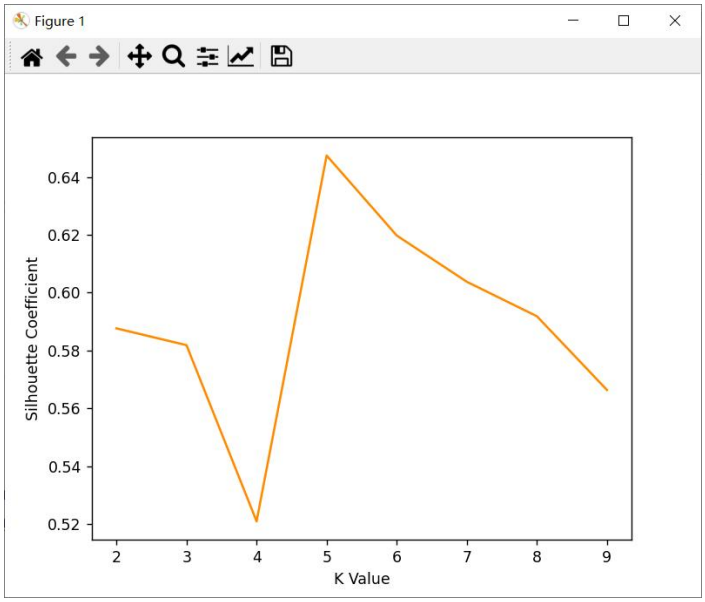
Abstract: 3D road network with highly accurate elevation information (+-20cm) from Denmark used in eco-routing and fuel/Co2-estimation routing algorithms.

Data Set Characteristics:	Sequential, Text	**Number of Instances:**	434874	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	2013-04-16
Associated Tasks:	Regression, Clustering	**Missing Values?**	N/A	**Number of Web Hits:**	227584

属性（其中未知真实类别数量）：

1. OSM_ID: OpenStreetMap ID for each road segment or edge in the graph.
2. LONGITUDE: Web Mercaptor (Google format) longitude
3. LATITUDE: Web Mercaptor (Google format) latitude
4. ALTITUDE: Height in meters.

K-means 算法（随机抽取其中的 1000 条数据）结果：



因为未知实际类别数量，通过一系列 K 值的尝试。从中可以看出，对于随机抽取的这 1000 条数据而言，分成 5 类的聚类效果最好，轮廓系数最高。

五、结论

1. K-means 算法是对初始值敏感的，初值选择不好容易陷入局部最优解，影响聚类效果。

2. 未知样本的实际类别数时，可以通过判断不同 K 值聚类得出的轮廓系数来判断类别数量。

3. 兰德指数和轮廓系数能够衡量聚类的准确程度，但是兰德指数需要已知分类标签，而轮廓系数则不需要。

4. 兰德指数在随机划分类别时并不为 0，可以用调整兰德指数优化。

六、参考文献

[1] Guenin B , Knemann J , Tunel L . A Gentle Introduction to Optimization. 2018.
[2] https://en.wikipedia.org/wiki/Silhouette_(clustering)

七、附录：源代码（带注释）

```python
import numpy as np
import matplotlib.pyplot as plt
import warnings

# 准备：生成训练集，其中 k 为簇的数量，total_num 为样本总数
def get_trainset(k, total_num):
    # np.random.seed(0)
    # 在实验中不打算让训练集各个簇的样本数量一致或手动指定
    # 因此使得每个簇样本数量 ~ N(total_num / k, total_num / (6 * k))
    num_temp_lst = (np.around(np.random.normal(total_num / k, total_num / (6 * k), k-1)))
    # 为了让总样本数量固定为 total_num，最后一个簇的样本数量不随机生成，
    # 指定为 total_num - 之前随机生成的样本数量和
    num_lst = np.append(num_temp_lst, total_num - np.sum(num_temp_lst)).astype(int)

    #np.random.seed(4)
    # 各簇的中心横坐标分布 x ~ U(0,50)
    x_mean_lst = np.random.normal(0, 30, size = k)
```

```
    # 各簇的中心横坐标分布 y ~ U(0,40)
    y_mean_lst = np.random.normal(0, 40, size = k)
    # 各簇的协方差均值方分布 CovXY = [[sigma1, 0], [0, sigma2]]
    # 其中：sigma1 ~ N(36, 25), sigma2 ~ N(25, 25)
    CovXY = np.zeros((k, 2, 2))
    CovXY[:, 0, 0] = np.random.normal(36, 5)
    CovXY[:, 1, 1] = np.random.normal(36, 5)

    x = np.zeros((np.sum(num_lst), 2))
    y = np.zeros(np.sum(num_lst)).astype(int)

    for i in range(k):
        temp_sum = np.sum(num_lst[:i])
        x[temp_sum : temp_sum + num_lst[i], :] = np.random.multivariate_normal([x_mean_lst[i],
y_mean_lst[i]],
                                                CovXY[i], size = num_lst[i])
        y[temp_sum : temp_sum + num_lst[i]] = i

    return [x, y]

# 绘制训练集散点图
def draw_trainset2D(trainset):
    color_lst = ['violet', 'grey', 'deepskyblue', 'greenyellow', 'pink', 'blue', 'darkorange',]
    i = 0
    for [x,y] in trainset[0]:
        plt.scatter(x, y, color = color_lst[trainset[1][i]], marker = '.')
        i += 1
```

```
# k_means 算法计算分类中心和标签
def k_means(trainset, k, epsilon=1e-4):
    center = np.zeros((k, trainset[0].shape[1]))
    for i in range(k):
        center[i,:] = trainset[0][np.random.randint(0, high = trainset[0].shape[0]), :]
    while True:
        distance = np.zeros(k)
        # 根据中心重新给每个点贴分类标签
        for i in range(trainset[0].shape[0]):
            for j in range(k):
                distance[j] = np.linalg.norm(trainset[0][i] - center[j, :])
            trainset[1][i] = np.argmin(distance)
        # 根据每个点新的标签计算它的中心
        new_center = np.zeros((k, trainset[0].shape[1]))
        count = np.zeros(k)
        # 对每个类的所有点坐标求和
```

```
        for i in range(trainset[0].shape[0]):
            new_center[int(trainset[1][i]), :] += trainset[0][i]
            count[int(trainset[1][i])] += 1
        # 对每个类的所有点坐标求平均值
        for i in range(k):
            if count[i] != 0:
                new_center[i, :] = new_center[i, :] / count[i]
        #用差值的二范数衡量精度
        if np.linalg.norm(new_center - center) < epsilon:
            break
        else:
            center = new_center
    return center
```

```
# 绘制聚类中心
def draw_center(point_set):
    for [x,y] in point_set:
        plt.scatter(x, y, s = 100, color = 'red', marker = '+')
```

```
# 通过计算兰德指数 RI 判断聚类效果
def compute_RI(y_true, y_pred):
    n = len(y_true)
    a, b = 0, 0
    for i in range(n):
        for j in range(i+1, n):
            if (y_true[i] == y_true[j]) & (y_pred[i] == y_pred[j]):
                a += 1
            elif (y_true[i] != y_true[j]) & (y_pred[i] != y_pred[j]):
                b += 1
    RI = (a + b) / (n*(n-1)/2)
    return RI
```

```
# 通过计算轮廓系数 Silhouette Coefficient 判断聚类效果
def compute_SC(result_set, k):
    n = len(result_set[0])
    #sc_lst 存储每个样本点的轮廓系数
    sc_lst = np.zeros(n)
    for i in range(n):
        sum_lst = np.zeros(k)
        count_lst = np.zeros(k)
        for j in range(n):
            sum_lst[result_set[1][j]] = np.linalg.norm(result_set[0][j, :] - result_set[0][i, :])
            count_lst[result_set[1][j]] += 1
        ave_lst = np.zeros(k)
```

```
        ave_lst = sum_lst / count_lst
        # 当前点 i 与同簇内其他点的平均距离
        a = ave_lst[result_set[1][i]]
        # 当前点 i 与其他各个簇内点平均距离的最小值
        b = np.min(np.delete(ave_lst, result_set[1][i]))
        sc_lst[i] = (b - a) / max(a, b)
    return np.mean(sc_lst)
```

```
if __name__ == '__main__':
    warnings.simplefilter('error')
    #self_made_data 变量指示：训练时使用自己编造的数据 or 从 UCI 下载的数据
    self_made_data = False

    #随机生成二维高斯分布的点作为数据集
    if self_made_data == True:
        k = 3
        total_num = 200
        trainset = get_trainset(k, total_num)
        y_true = np.zeros(total_num)
        y_true[:] = trainset[1][:]
        draw_trainset2D(trainset)
        center_point = k_means(trainset, 5)
        draw_center(center_point)
        #因为生成的点已知标签，可以用兰德指数 RI 评价效果
        plt.title('k = %d, total sample number = %d, Rand Index = %.4f'
                %(k, total_num, compute_RI(y_true,trainset[1][:] )))
        plt.show()

    #下载 UCI 的 iris 作为数据集
    if self_made_data == False:
        data_set = np.loadtxt('3D_spatial_network.txt', delimiter=',', encoding='utf-8',
usecols=(0,1,2,3))
        np.random.shuffle(data_set)
        data_set = data_set[:1000]
        print (data_set)
        print ('完成打乱！')
        y_set = np.zeros(data_set.shape[0]).astype(int)
        trainset = [data_set, y_set]
        sc_lst = []
        for k in range(2,10):
            temp_sc = []
            # 对同一个 k 值连续计算 20 次轮廓系数取平均值
            for i in range(20):
```

```
            try:
                k_means(trainset, k)
                temp_sc.append(compute_SC(trainset, k))
            except RuntimeWarning:
                pass
        #print (temp_sc)
        sc_lst.append(np.mean(temp_sc))
        plt.show()

    print (sc_lst)
    plt.plot(range(2,10), sc_lst, color = 'darkorange')
    plt.xlabel('K Value')
    plt.ylabel('Silhouette Coefficient')
    plt.show()
```