北京郵電大學

机器学习

姓　　名 王明瑞

学　　号 2018211807

班　　级 2018211406

学　　院 现代邮政学院(自动化学院)

专　　业 自动化

内　　容 第九章作业

课　　程 机器学习

2021 年 6 月 18 日

目录

第九章 聚类

一、数据集介绍

本次实验使用到一个数据集，为西瓜数据集 4.0。

西瓜数据集 4.0 包含 30 条信息，每条信息对应西瓜的 2 种属性：密度和含糖率。西瓜数据集 4.0 的具体内容如下图所示。

表 9.1 西瓜数据集 4.0

编号	密度	含糖率	编号	密度	含糖率	编号	密度	含糖率
1	0.697	0.460	11	0.245	0.057	21	0.748	0.232
2	0.774	0.376	12	0.343	0.099	22	0.714	0.346
3	0.634	0.264	13	0.639	0.161	23	0.483	0.312
4	0.608	0.318	14	0.657	0.198	24	0.478	0.437
5	0.556	0.215	15	0.360	0.370	25	0.525	0.369
6	0.403	0.237	16	0.593	0.042	26	0.751	0.489
7	0.481	0.149	17	0.719	0.103	27	0.532	0.472
8	0.437	0.211	18	0.359	0.188	28	0.473	0.376
9	0.666	0.091	19	0.339	0.241	29	0.725	0.445
10	0.243	0.267	20	0.282	0.257	30	0.446	0.459

图 1 西瓜数据集 4.0

二、K-means 算法

2.1 K-means 算法介绍

k-means 算法是一种聚类算法，所谓聚类，即根据相似性原则，将具有较高相似度的数据对象划分至同一类簇，将具有较高相异度的数据对象划分至不同类簇。聚类与分类最大的区别在于，聚类过程为无监督过程，即待处理数据对象没有任何先验知识，而分类过程为有监督过程，即存在有先验知识的训练数据集。

2.2K-means 算法原理

K-means 算法中的 k 代表类簇个数，means 代表类簇内数据对象的均值（这种均值是一种对类簇中心的描述），因此，K-means 算法又称为 K-均值算法。K-means 算法是一种基于划分的聚类算法，以距离作为数据对象间相似性度量的标准，即数据对象间的距离越小，则它们的相似性越高，则它们越有可能在同一个类簇。数据对象间距离的计算有很多种，K-means 算法通常采用闵可夫斯基距离来计算数据对象间的距离。

给定样本集$D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m\}$，“k 均值”(K-means)算法针对聚类所得簇划分$\mathcal{C} = \{C_1, C_2, \dots, C_k\}$最小化平方误差：

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|_2^2 \qquad (1-1)$$

其中$\boldsymbol{\mu}_i = \frac{1}{|C_i|}\sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$是簇$C_i$的均值向量，直观来看，式(1-1)在一定程度上刻画了簇内样本围绕簇均值向量的紧密程度，E 越小则簇内样本相似度越高。

K-means 算法流程如下：

输入： 样本集 $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$;
聚类簇数 k.

过程：

从 D 中随机选择 k 个样本作为初始均值向量 $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$
repeat
  令 $C_i = \varnothing \ (1 \leqslant i \leqslant k)$
  for $j = 1, 2, \ldots, m$ **do**
    计算样本 $\boldsymbol{x}_j$ 与各均值向量 $\boldsymbol{\mu}_i \ (1 \leqslant i \leqslant k)$ 的距离: $d_{ji} = \|\boldsymbol{x}_j - \boldsymbol{\mu}_i\|_2$;
    根据距离最近的均值向量确定 $\boldsymbol{x}_j$ 的簇标记: $\lambda_j = \arg\min_{i \in \{1,2,\ldots,k\}} d_{ji}$;
    将样本 $\boldsymbol{x}_j$ 划入相应的簇: $C_{\lambda_j} = C_{\lambda_j} \bigcup \{\boldsymbol{x}_j\}$;
  end for
  for $i = 1, 2, \ldots, k$ **do**
    计算新均值向量: $\boldsymbol{\mu}_i' = \frac{1}{|C_i|}\sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$;
    if $\boldsymbol{\mu}_i' \neq \boldsymbol{\mu}_i$ **then**
      将当前均值向量 $\boldsymbol{\mu}_i$ 更新为 $\boldsymbol{\mu}_i'$
    else
      保持当前均值向量不变
    end if
  end for
until 当前均值向量均未更新

输出： 簇划分 $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$

三、K-means 算法代码

1. 读入要分类的西瓜数据集 4.0。

使用 pandas 的 pandas.read_csv()函数读取 watermelon.csv 文件，并将数据转化为 nparray 格式，方便使用。

```
data = pd.read_csv(r'Data/watermelon4.0.csv', index_col=0)
X = np.array(data)
```

2. 从样本中随机选择 K 个样本作为初始均值向量

```
X_len = len(X)
index = list(range(len(X)))
random.shuffle(index)
random_init_avg_vec_index = index[:n_class]   # 从 X 中随机选择 n_class 个样本作为初始均值向量
print(str(n_class)+"个中心位置分别为:", random_init_avg_vec_index)
random_init_avg_vec = X[random_init_avg_vec_index, :]
```

```
clusters = {}    # 以字典的形式保存每个簇
```

3. 计算样本与均值向量之间的距离，根据最近均值向量确定样本的簇标记

```
for i in range(X_len):       # 计算每个样本与均值向量之间的距离
    dists = []
    for item in random_init_avg_vec:
        dist = np.sqrt(((X[i] - item) ** 2).sum())  # 计算样本和均值向量的闵可夫斯基距离
        dists.append(dist)
    clusters[int(np.argmin(dists))].append(X[i].tolist())   # 根据最近均值向量确定样本的簇标记
```

4. 图形化输出结果，将不同的簇标记为不同颜色，并标记出均值向量位置

```
def plot_k_means(clusters):
    mid = []
    for key in clusters.keys():
        mid.append(clusters[key][0])
        array = np.asarray(clusters[key])
        plt.scatter(x=array[:, 0], y=array[:, 1], marker='o')

    mid = np.asarray(mid)
    plt.scatter(x=mid[:, 0], y=mid[:, 1], marker='+', s=500)
    plt.show()
```

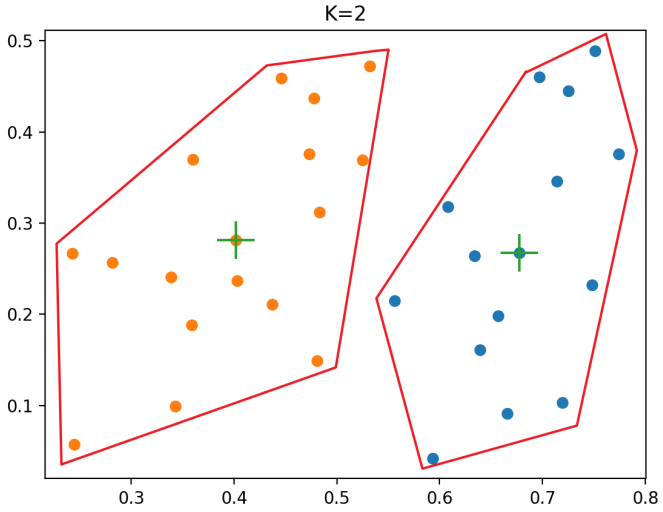
四、习题解答

4.1 习题 9.4

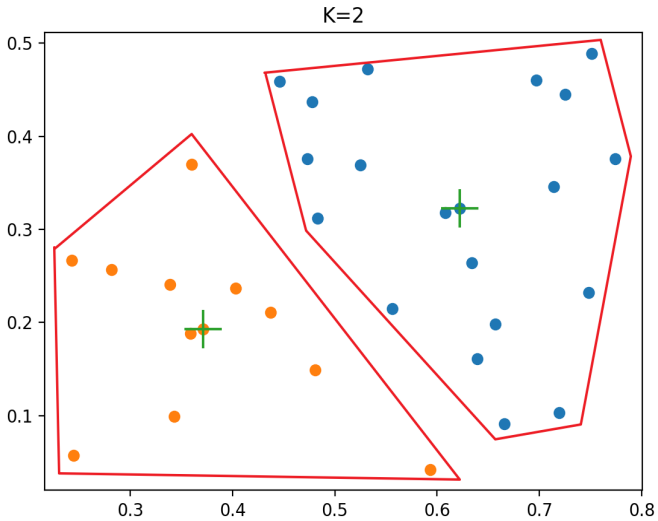
【9.4】试编程实现 k 均值算法，设置三组不同的 k 值、三组不同初始中心点，在西瓜数据集 4.0 上进行实验比较，并讨论什么样的初始中心有利于取得好结果。

实验结果如下：

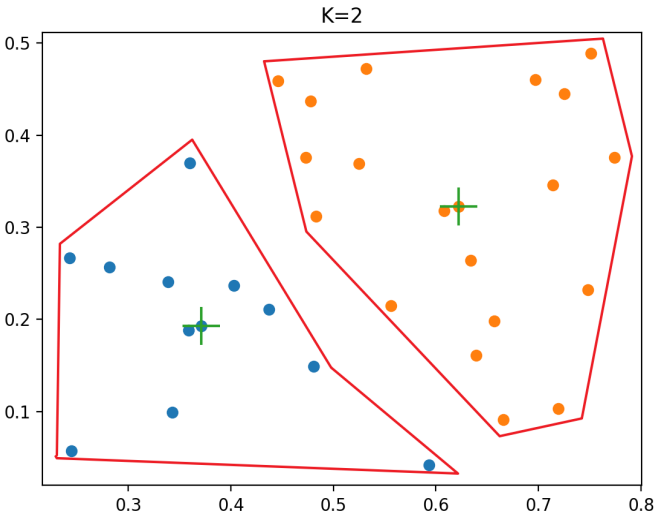
1.1 K=2, 初始中心样本[9, 25]，中心位置坐标[(0.437,0.211), (0.525,0.369)]



1.2　K=2，初始中心样本[21, 16]，中心位置坐标[(0.748,0.232), (0.593,0.042)]



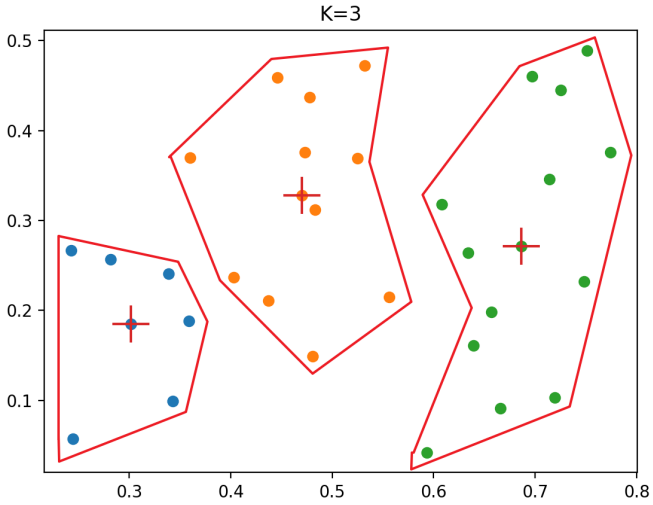
1.3　K=2，初始中心样本[21, 16]，中心位置坐标[(0.343,0.099), (0.714,0.346)]



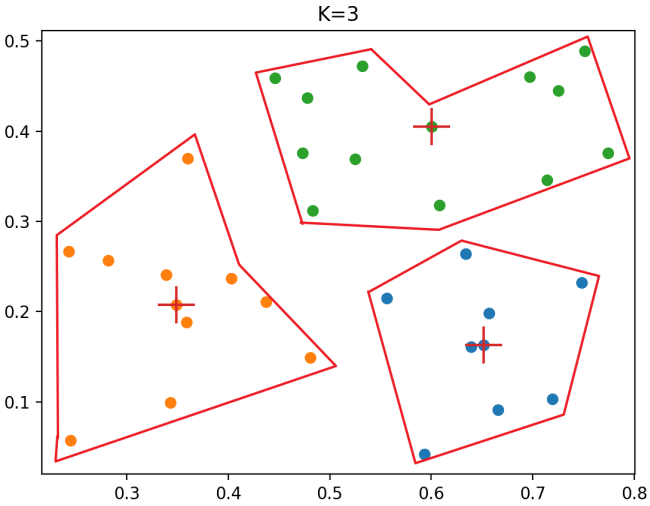
K=2 时划分为 2 个簇，可以看到不同的中心点划分出来的结果不同。

2.1　K=3 初始中心样本[10, 7, 21]

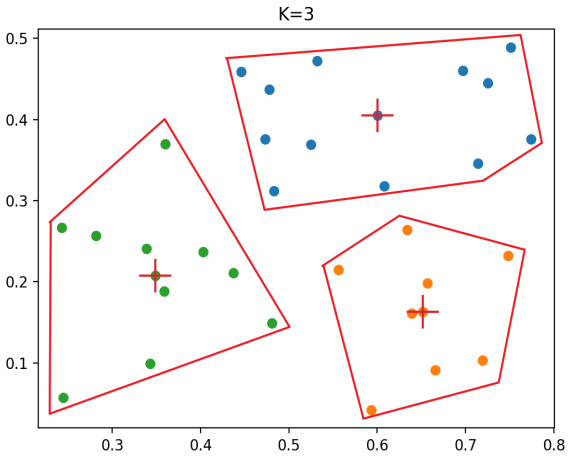
中心位置坐标[(0.243, 0.267), (0.481, 0.149), (0.748, 0.232)]



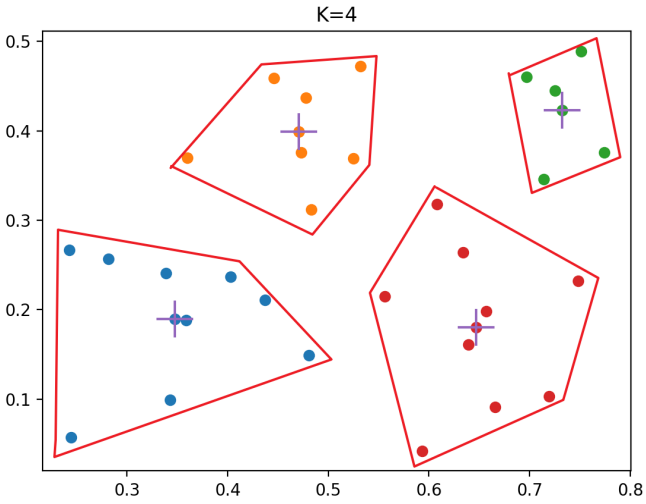
2.2　K=3，初始中心样本[29, 22, 11]
中心位置坐标[(0.725, 0.445), (0.714, 0.346), (0.245, 0.057)]



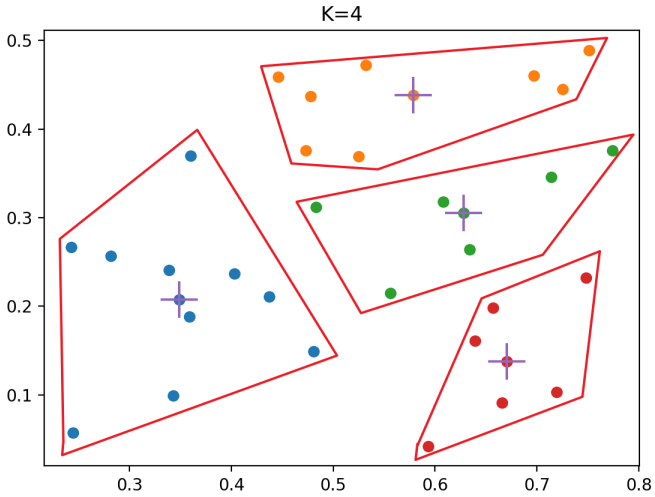
2.3　K=3，初始中心样本[116, 12, 4]
中心位置坐标[(0.593, 0.042), (0.343, 0.099), (0.608, 0.318)]



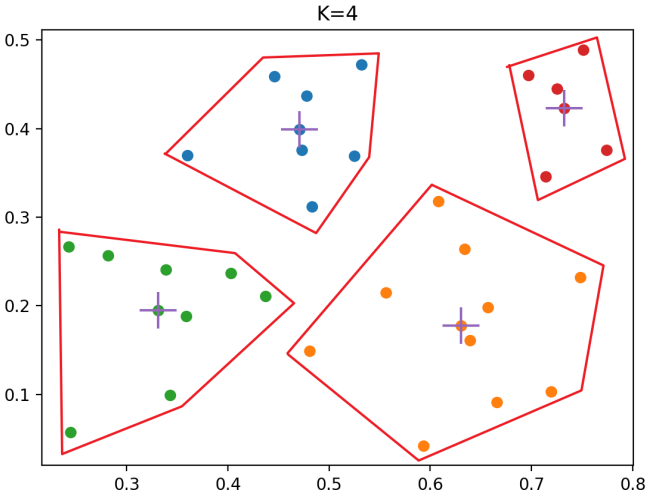
3.1　K=4 初始中心样本[7, 24, 29, 22]
中心位置坐标[(0.481, 0.149), (0.478, 0.437), (0.725, 0.445),(0.714, 0.346)]



3.2　K=4, 初始中心样本[19, 27, 3, 14]
中心位置坐标[(0.339, 0.241), (0.532, 0.472), (0.634, 0.264),(0.657, 0.198)]



3.3　K=3, 初始中心样本[27, 3, 11, 22]
中心位置坐标[(0.532, 0.472), (0.634, 0.264), (0.245, 0.057),(0.714, 0.346])]



上述结果在 K=2、3、4 都实现了收敛，并完成了划分，但是初始随机的中心点不同会导致算法的迭代次数与最终结果有很大的不同。一般来说，初始的中心点越集中且越靠近边缘，则会使得迭代次数更多。初始中心点越分散，迭代次数越少，结果越好。

五、附录

1. K-means.py

```
# -*- coding: utf-8 -*
# @Time: 2021/6/10 13:40
# @File : K-means.py
# 西瓜书习题 9.4 编程实现 K 均值算法

import random
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


def k_means(X, n_class=3):
    X_len = len(X)
    index = list(range(len(X)))
    random.shuffle(index)

    random_init_avg_vec_index = index[:n_class]   # 从 X 中随机选择
n_class 个样本作为初始均值向量
    print(str(n_class)+"个中心位置分别为:",
(np.array(random_init_avg_vec_index)+1).tolist())
    print(X[random_init_avg_vec_index])        # 打印初始中心位置
    random_init_avg_vec = X[random_init_avg_vec_index, :]
    clusters = {}   # 以字典的形式保存每个簇

    for i in range(n_class):
        clusters[i] = [random_init_avg_vec[i].tolist()]     # 初始化每个
簇的中心向量
    old_clusters = {0: np.random.normal(0, 0.01, (1, 2))}   # 随机生成
一组参考中心

    counter = 0
    while old_clusters != clusters:         # 只要每个簇的中心还在变化，
就继续迭代
        counter += 1
        old_clusters = clusters.copy()
        clusters = {}
        for i in range(n_class):
            clusters[i] = [random_init_avg_vec[i].tolist()]  # 记录每个
簇的中心向量

        for i in range(X_len):      # 计算每个样本与均值向量之间的距离
            dists = []
            for item in random_init_avg_vec:
                dist = np.sqrt(((X[i] - item) ** 2).sum())  # 计算样本和
均值向量的闵可夫斯基距离
```

```
                dists.append(dist)
            clusters[int(np.argmin(dists))].append(X[i].tolist())   #
根据最近均值向量确定样本的簇标记
        for key in clusters.keys():
            array = np.asarray(clusters[key])
            random_init_avg_vec[key] = array.sum(axis=0) / len(array)
# 更新每个簇的均值向量
    print("共迭代"+str(counter)+"次")
    return clusters


def plot_k_means(clusters):
    mid = []
    for key in clusters.keys():
        mid.append(clusters[key][0])
        array = np.asarray(clusters[key])
        plt.scatter(x=array[:, 0], y=array[:, 1], marker='o')

    mid = np.asarray(mid)
    plt.scatter(x=mid[:, 0], y=mid[:, 1], marker='+', s=500)
    plt.title("K="+str(len(clusters)))
    plt.show()


def main():
    data = pd.read_csv(r'Data/watermelon4.0.csv', index_col=0)
    X = np.array(data)

    clusters = k_means(X, n_class=4)
    plot_k_means(clusters)


if __name__ == '__main__':
    main()
```