目 录

1. 题目 1
1.1. 实验目的 1
1.2. 实验要求 1
1.3. 实验内容 1
1.4. 实验数据 1
2. 算法阐述或实验步骤说明 2
2.1. 算法阐述 2
2.1.1. K-Means 定义 2
2.1.2. K-Means 原理 2
2.1.3. K-Means 算法步骤 2
2.2. 实验步骤说明 4
3. 实验结果与截图 7
4. 总结 9
5. 参考文献 9

1. 题目

1.1. 实验目的

通过使用 K-Means 算法实现对鸢尾花数据的聚类，帮助学生掌握 K-Means 算法的原理和使用场景、掌握导入数据、预处理数据、模型训练、模型评估和可视化方法，使研究生们尽快熟悉人工智能、机器学习与大数据领域的基础算法。

1.2. 实验要求

(1) 理解算法本身，用程序设计语言(不限语言，但建议选择 Python、C、C++或 Java)实现算法。

(2) 除了数学运算(例如求函数值、矩阵求逆等)可以调用现有的库函数，其他一律不准调库。

1.3. 实验内容

Iris 也称鸢尾花卉数据集[1]，是常用的分类实验数据集，由 R. A. Fisher 于 1936 年收集整理的。其中包含 3 种植物种类，分别是山鸢尾(setosa)、变色鸢尾(versicolor)和维吉尼亚鸢尾(virginica)，每类 50 个样本，共 150 个样本。该数据集包含 4 个特征变量，1 个类别变量。iris 每个样本都包含了 4 个特征：花萼长度，花萼宽度，花瓣长度，花瓣宽度，以及 1 个类别变量(label)。

本实验需要通过自行实现 K-Means 算法，通过利用数据集中的特征，实现数据的聚类，并掌握导入数据、预处理数据、模型训练、模型评估和可视化方法。

1.4. 实验数据

本实验所用数据集是鸢尾花卉数据集，来源为 sklearn.datasets 中的 iris 数据集。

iris 数据集有 150 个样本数据，分为 3 个类别，分别用整数表示：0(Setosa)，1(Versicolour)，2(Virginica)。每个样本包含 4 个特征变量：花萼长度(sepal_length)、花萼宽度(sepal_width)、花瓣长度(petal_length)、花瓣宽度(petal_width)。

在实验中，使用花萼宽度(sepal_width)、花瓣长度(petal_length)和花瓣宽度(petal_width)作为特征变量，实现数据的聚类效果。

2. 算法阐述或实验步骤说明

2.1. 算法阐述

2.1.1. K-Means 定义

K-Means 算法是基于样本集合划分的聚类算法，是一种无监督学习。其中，K 表示分类的数量，Means 指样本均值[2]。目前，K-Means 算法已经广泛应用于各种商业，包括：

(1) 客户分割：可以对客户进行分组，以便更好地定制产品。

(2) 文本、文档或搜索结果聚类：分组以查找文本中的主题。

(3) 图像分组或图像压缩：图像或颜色相似的组。

(4) 异常检测：从集群中找出不相似的地方或异常值

(5) 半监督式学习：将集群与一组较小的已标记数据和监督式机器学习相结合，以获得更有价值的结果。

2.1.2. K-Means 原理

(1) 主要思想：将样本划分为 k 个子集，再将 n 个样本分到这 k 个类别中，使得每个样本到所属类的中心距离最小。

(2) 假设：一个样本只属于一个类(或者不同类之间的交集为空)。

(3) K-Means 算法使用欧式距离平方判断数据之间的相似程度，使用样本的均值中心作为类别，使用样本与其类中心之间的距离的总和作为损失函数，使用迭代算法作为聚类训练过程的关键方法。

2.1.3. K-Means 算法步骤

(1) 主要步骤：

① 初始化聚类中心：随机选取 k 个样本作为初试的聚类中心；

② 给聚类中心分配样本：计算每个样本与各个聚类中心之间的距离，把每个样本分配给距离它最近的聚类中心；

③ 更新聚类中心：将聚类中心更新为这个聚类所有样本的平均值处；

④ 停止更新：当聚类中心不再变化，则停止更新聚类中心。

(2) 如何确定 k 值？

① 多选取几个 k 值，对比聚类效果，选择最优的 k 值；

② 结合业务特点设定 k 值；

③ 根据 SSE 和轮廓系数，SSE 越小、轮廓系数越大，聚类效果越好。

(3) 如何评估？

① 误差平方和 SSE：给定一个包含 n 个数据对象的数据集合 $D=\{x_1,x_2,...,x_n\}$，定义经由 K-Means 算法进行聚类后产生的类别集合为 $C=\{C_1,C_2,...,C_K\}$，则 SSE 计算公式如公式(2-1)所示。

$$SSE(C)=\sum_{k=1}^{K}\sum_{x_i\in C_k}\left|x_i-m_k\right|^2. \tag{2-1}$$

其中，m_k 是簇 C_k 的中心点，计算方式如公式(2-2)所示。

$$m_k=\frac{\sum_{x_i\in C_k}x_i}{\left|C_k\right|}. \tag{2-2}$$

② 轮廓系数：轮廓系数由两部分组成，即样本与同一簇类中其他样本点的平均距离(量化凝聚度)和样本与距离最近簇类中所有样本点的平均距离(量化分离度)。因此，每个样本的轮廓系数定义如公式(2-3)所示。

$$S=\frac{b-a}{\max(a,b)}. \tag{2-3}$$

其中，$S\in[-1,1]$。

(4) 算法伪代码

算法 2-1K-Means 算法伪代码[3]

算法 2.1K-Means **算法**

输入：样本集 $D=\{x_1,x_2,...,x_n\}$；聚类簇数 k；

输出：簇划分 $C=\{C_1,C_2,...,C_k\}$；

1: 从 D 中随机选择 k 个样本作为初始均值向量 $\{\mu_1,\mu_2,...,\mu_k\}$；

2: **Repeat**{

3: **Set** $C_i=\varnothing(1\le i\le k)$；

4: **For** $j=1,2,...,n$ **do**

5: 计算样本 x_j 与各簇中心向量 $\mu_i(1\le i\le k)$ 的欧式距离 $d_{ji}=\left\|x_j-\mu_i\right\|^2$；

6: 根据距离最近的簇中心向量确定 x_j 的簇标记 $\lambda_j=\arg\min_{i\in(1,2,...,k)}d_{ji}$；

7: 将样本 x_j 划入相应的簇 $C_{\lambda_j}=C_{\lambda_j}\cup\{x_j\}$；

8: **End for**

9: **For** $i=1,2,...,k$ **do**

10: 计算新簇中心向量 $\mu_i^{'}=\frac{1}{|C_i|}\Sigma_{x\in C_i}x$；

11: **If** $\mu_i^{'}\neq\mu_i$ **then**

12: 将当前簇中心向量 μ_i 更新为 $\mu_i^{'}$

13: **Else**

14: 保持当前簇中心向量不变

15: **End if**

16: **End for**

17: **Until** 当前簇中心向量均未更新

18: };

2.2. 实验步骤说明

(1) 搭建实验环境

① 操作系统：Windows10 64 位，CPU 为 Intel(R) Core(TM) i7-8550U；

② 开发语言及软件：Python，PyCharm 2021.3.2 (Community Edition)；

③ 编译器及软件包版本：Python 3.6，scikit-learn 0.19；

(2) 引入库函数

```
from definition import *
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
import numpy as np
```

(3) 数据载入及预处理

```
# test
# load data
iris = load_iris()
x = iris.data[:, 1:]
feature_names = iris.feature_names[1:]
# 将数据转换为矩阵的形式
data = np.mat(x.tolist())
```

(4) 可视化描述数据信息

```
# 输出数据集散点分布图
plt.scatter(x[:, 0], x[:, 1], c='r')
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
```

```
plt.show()
```

(5) 模型训练

① 定义欧式距离

```
# 定义两点间的距离—欧氏距离
def distance(self, dot_1, dot_2):
    return np.sqrt(np.sum(np.power((dot_1 - dot_2), 2)))
```

② 定义随机产生 k 个质心

```
# 定义随机产生 k 个质心
def generateCenter(self, data, k):
    # 获取数据集的特征数
    num_features = np.shape(data)[1]
    # 生成 k 个质心矩阵
    center_dot = np.mat(np.zeros((k, num_features)))
    for i in range(num_features):
        # 找出每个特征的最小值
        min_dot = np.min(data[:, i])
        # 找出每个特征的范围
        range_dot = float(np.max(data[:, i]) - min_dot)
        # 在该范围中随机产生质心
        center_dot[:, i] = min_dot + range_dot * np.random.rand(k, 1)
    return center_dot
```

③ 定义 K-Means 训练过程

```
def KMeansTraining(self, data, k):
    # 获取数据集的数据数量
    num_training = np.shape(data)[0]
    # 生成数据的误差记录列表
    error_list = np.mat(np.zeros((num_training, 2)))
    center_dot = self.generateCenter(data, k)
    flag = True
    while flag:
        flag = False
        # 寻找每一个数据距离哪一个质心最近
        for i in range(num_training):
            min_distance = np.Inf
            min_index = -1
            for j in range(k):
                distance_i_to_j = self.distance(center_dot[j, :], data[i, :])
                if distance_i_to_j < min_distance:
                    min_distance = distance_i_to_j
                    min_index = j
            # 检测每个数据对应的质心下标是否被更新为当前最小距离对应的质心下标
            if error_list[i, 0] != min_index:
```

```
                flag = True
            error_list[i, :] = min_index, min_distance**2
        # 更新质心的位置
        for m in range(k):
            # 找到以下标 m 为质心的所有点的集合
            dots_for_m = data[np.nonzero(error_list[:, 0].A == m)[0]]
            # 计算这些点的均值
            center_dot[m, :] = np.mean(dots_for_m, axis=0)
    # 返回新的质心列表以及误差列表
    return center_dot, error_list
```

(6) 进行数据测试并输出测试结果

在这里，我们设定质心为 3 个，即 $k=3$。

```
K_Means = KMeans()
center_dot, error_list = K_Means.KMeansTraining(data, 3)
print("==================== 聚类结果 ====================")
print()
print("质心列表: ")
print(center_dot)
print("误差列表: ")
print(error_list)
print()
print("================================================")
```

(7) 可视化描述测试结果

```
# 画出聚类图
color = ['r', 'b', 'g']
x = np.array(data)
center_dot_arr = np.array(center_dot)
error_list_arr = np.array(error_list[:,0])
# 根据质心，画出对应聚类数据点
for i, n in enumerate(error_list_arr):
    plt.scatter(x[i][0], x[i][1], c=color[int(n[0])])
# 画出质心
plt.scatter(
    center_dot_arr[:, 0], center_dot_arr[:, 1],
    marker='*', s=160, c='m', label='Center of Mass')
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
plt.show()
```

3. 实验结果与截图

(1) 载入数据后，对数据进行可视化描述。如图 3-1 所示。

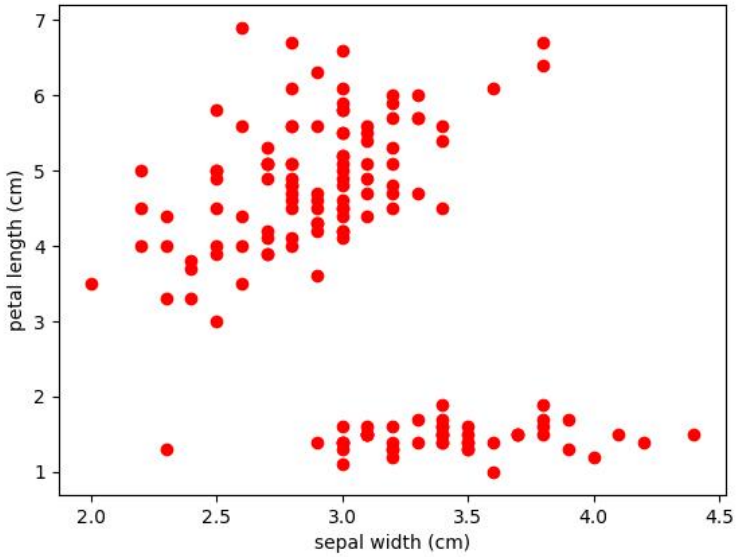
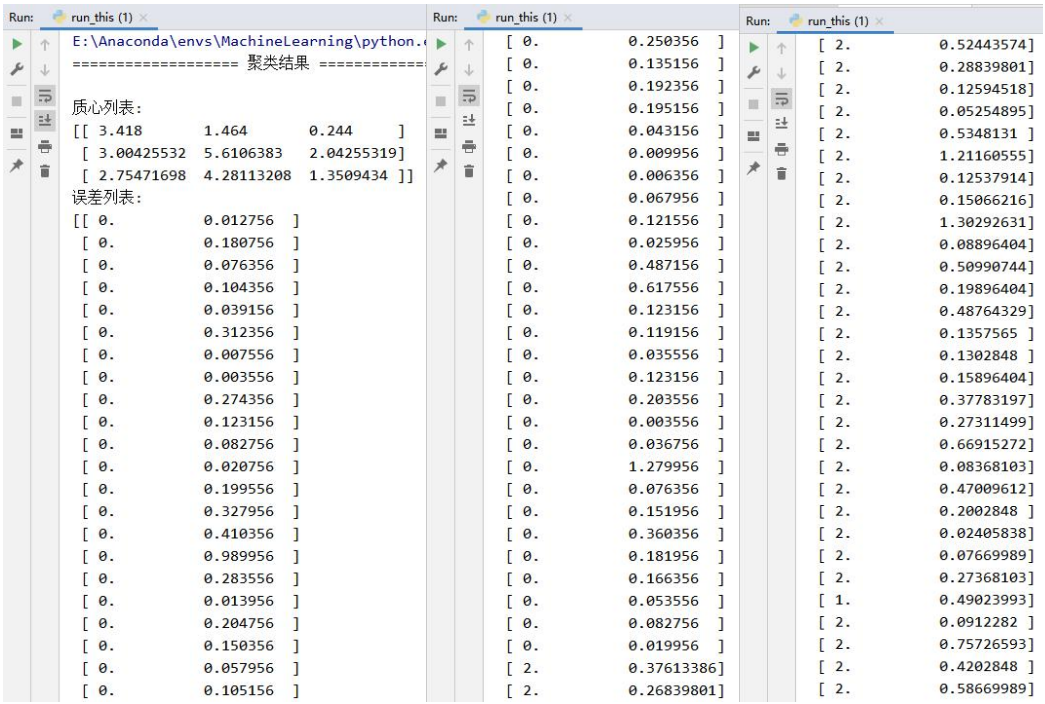


图 3-1 数据集中载入部分数据散点图

(2) 针对上述载入数据，设定质心个数为 3($k=3$)，使用 K-Means 算法对数据集进行训练后，输出训练结果，即最终质心的坐标以及对应索引的误差列表，如图 3-2 所示。



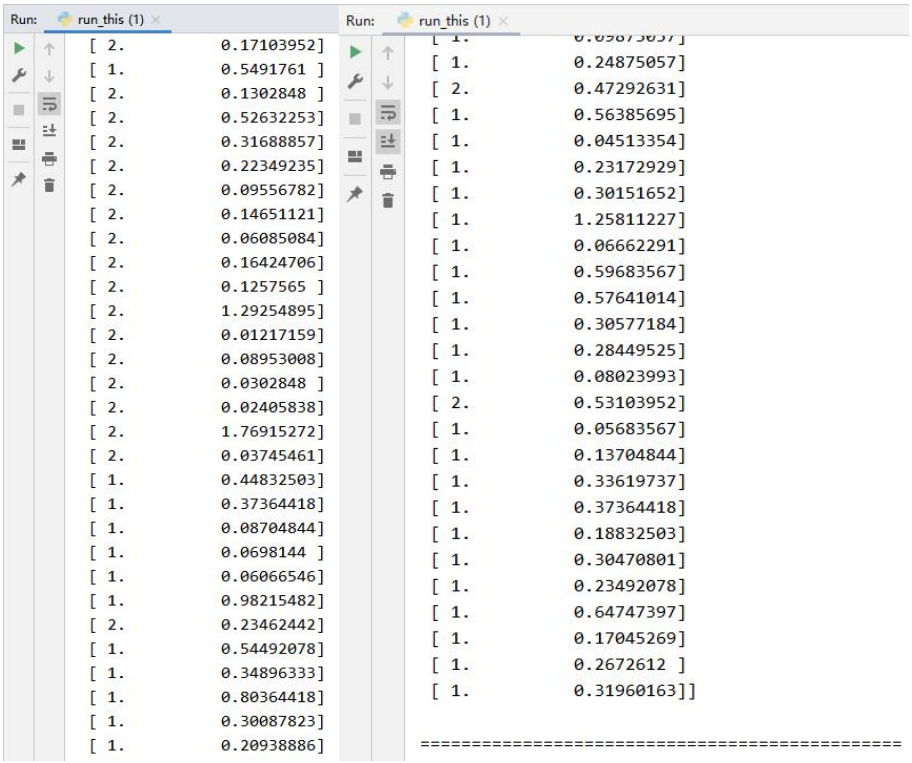


图 3-2K-Means 算法训练结果输出

(3) 根据训练结果找到的 3 个质心，对数据集中的数据进行聚类，输出的散点图如图 3-3 所示。

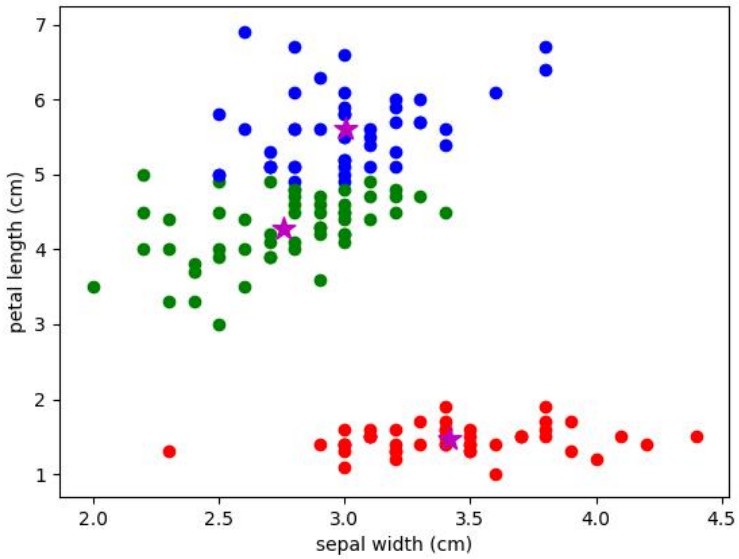


图 3-3K-Means 聚类结果分布散点图

4. 总结

K-Means 算法在实验中虽然往往是局部最优的求解，但其算法复杂度低，所求得的局部最优解也能具有较好的聚类分类效果。但通过实验我们可以发现，K-Means 算法的聚类效果会受到特征数量以及选定质心数量的影响。同时训练过程对初试簇中心的选取极为敏感，不同的选取方式将得到最终不同的结果，对异常值也及其敏感。

5. 参考文献

[1] R. A. Fisher. Iris Data Set[DB/OL]. (1936)[2022-07-08]. http://archive.ics.uci.edu/ml/datasets/Iris.

[2] MacQueen J. Classification and analysis of multivariate observations[C]//5th Berkeley Symp. Math. Statist. Probability. 1967: 281-297.

[3] 周志华. 机器学习及其应用[M]. 清华大学出版社有限公司, 2006.

[4] 吴恩达. 吴恩达机器学习[EB/OL]. (2019-04-28)[2022-07-01]. https://www.bilibili.com/video/BV164411b7dx?p=32&t=0.7.