三峡大学

计 算 机 与 信 息 学 院

《人工智能》课 程 作 业

2024 年 秋 季 学 期

课程类型：	专业核心课
学　　号：	202210120518
姓　　名：	胡国昌
专　　业：	计算机科学与技术
授课教师：	臧兆祥

完成日期：2024 年 10 月 31 日

K-means 聚类

一、 案例内容与要求

1.1. 实验内容与任务

现在有一批鸢尾花的数据，共包含 150 个样本，每个样本有四个属性，Sepal Length（花萼长度），Sepal Width（花萼宽度），Petal Length（花瓣长度），Petal Width（花瓣宽度）。同时，每个样本所属类别也已经标出，一共有 3 个类别：Iris Setosa（山鸢尾）、Iris Versicolour（杂色鸢尾），以及 Iris Virginica（维吉尼亚鸢尾）。要求学生根据样本的属性数据将鸢尾花用 K-Means 算法进行聚类，获得 3 个类别，并将每个样本分到一个类别中。然后将聚类所得的样本类别分布情况与原始数据中的样本类别分布情况进行对比，分析 K-Means 算法的性能。

1.2. 实验过程及要求

1. 实验环境要求：Windows/Linux 操作系统，Python 编译环境，numpy，matplotlib 等程序库。
2. 加载鸢尾花数据集，观察数据集特征。
3. 实现 K-Means 算法,运行并观察聚类结果。
4. 研究初始聚类中心的设置对 K-Means 算法收敛性的影响。
5. 研究参数 K 对聚类结果的影响。

二、 原理论述及解决方法

2.1. 原理概述

聚类是人类挖掘知识的重要手段，例如对自然界的生物进行类别和群体的划分。在商业活动中，对客户群体进行划分，能对客户特点进行分析并对不同群体进行针对性营销。在机器学习中，聚类属于无监督学习，直接在数据中挖掘类别关系。聚类跟有监督学习中的分类的区别是缺乏有标记的训练数据。聚类有两个任务，首先要确定将数据划分多少类，其次要将每个样本分到一个类别中。例如图 1 中的数据点，我们仅根据数据点的分布情况，可以考虑将其划分 4 个类，坐标比较相似的点处于同一个类中。完成聚类要基于两个原则：

1. 不同类别的样本之间相似性很小。
2. 同一类别的样本之间相似性很大。

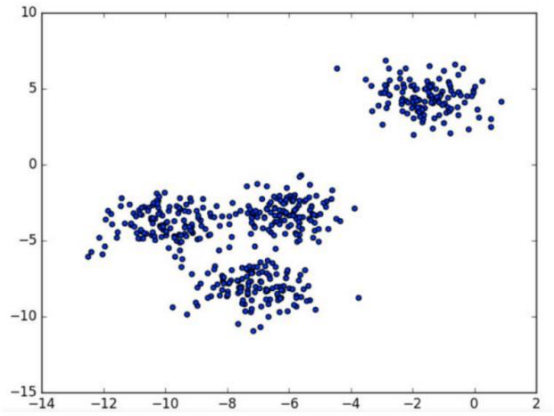


图 1 无标签数据

2.2. 解决方法

2.2.1. 数据加载

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

# 获取数据集并进行探索
iris = datasets.load_iris()
irisFeatures = iris['data']
irisFeaturesName = iris['feature_names']
irisLabels = iris['target']
```

2.2.2. K-Means 算法原理

K-Means 算法常用于对欧氏空间的样本点进行聚类。两个样本点x_i和x_j的相似度可用距离$\|x_i - x_j\|$来定义。假设聚类一共有 K 个类别，每个类别 Ck 定义一个聚类中心点u_k(注意，聚类中心点并不是样本点)，K-Means 算法规定，一个样本点根据其离每个聚类中心点的距离，划分到最近的类别中去。因此完成聚类的两个任务，只要能确定 K 个中心点即可。为了求出最好的中心点，定义一个损失函数，对聚类效果符合前述聚类原则的程度进行评价：

$$J(u_1, u_2, \cdots, u_k) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - u_k\|^2$$

从J的定义上看，各样本划分到距离最近聚类中心，能够达到更小的J值。另外，由于J是局部光滑的，可以通过对解求$\nabla J = 0$，计算出J最小时的聚类中心位置，

$$u_k^* = \frac{1}{|C_k|}\sum_{i \in C_k} x_i$$

可见最好的聚类中心u_k^*实际上是该类别的样本均值点。而将u_k移动到u_k^*后，如果所有样本点的划分不发生变化，则J到达一个局部最优值，K-Means 算法达到稳定。否则，可以继续求解在新的划分情况下的最优的中心点。

2.2.3. K-Means 算法实现

可以设想，如果某个聚类中心 k 远离所有的样本点，则样本点都会划分 到其他类别中。这个聚类中心 k 将会获得一个空的聚类，影响了聚类效果。 因此初始聚类点落到样本点中间较好。初始化时，限制聚类中心的坐标可 以达到这个目的。K-Means 算法实现如下：

```
def norm2(x):
  # 求 2 范数的平方值
  return np.sum(x * x)

class KMeans(object):
  def __init__(self,k = int,n = int):
    # k: 聚类数目，n: 数据集维度
    self.K = k
    self.N = n
    self.u = np.zeros((k,n))
    self.C = [[] for i in range(k)]
    self.labels = np.zeros(len(irisFeatures),dtype = int)
    # u[i]: 第i个类的中心点，C[i]: 第i类的数据点

  def fit(self,data: np.ndarray):
    # data: 每行是一个数据点
    self.select_u0(data)
    # 聚类中心初始化
    J = 0
    oldJ = 100
    while abs(J - oldJ) > 0.001:
      oldJ = J
      J = 0
      self.C = [[] for i in range(self.K)]
      # for x in data:
      #   nor = [norm2(self.u[i] - x) for i in range(self.K)]
      #   J += np.min(nor)
      #   self.C[np.argmin(nor)].append(x)
      # self.u = [np.mean(np.array(self.C[i]),axis = 0) for i in range(self.K)]
      for i,x in enumerate(data):
        nor = [norm2(self.u[j] - x) for j in range(self.K)]
```

```
            cluster_idx = np.argmin(nor)
            J += np.min(nor)
            self.C[cluster_idx].append(x)
            self.labels[i] = cluster_idx
        self.u = [np.mean(np.array(self.C[i]), axis = 0) for i in range(self.K)]

    def select_u0(self,data: np.ndarray):
        for j in range(self.N):
            # 得到该列数据的最大最小值
            minJ = np.min(data[:,j])
            maxJ = np.max(data[:,j])
            rangeJ = float(maxJ - minJ)
            # 聚类中心的第j维坐标随机初始化
            self.u[:,j] = minJ + rangeJ * np.random.rand(self.K)
```

2. 2. 4. 训练并显示聚类结果

设置 K=3,运行 K-Means 算法聚类后，用样本特征数据的前两个维度显示聚类效果。

```
model = KMeans(3,4)
# k = 3,n = 4
model.fit(irisFeatures)

# 绘制原始数据的分布图
plt.figure(figsize = (12,6))
plt.rcParams['font.sans-serif'] = ['KaiTi']

# 原始数据分布图
plt.subplot(1,2,1)
plt.scatter(irisFeatures[:,0],irisFeatures[:,1],c = irisLabels,cmap = 'viridis',marker
            = 'o',label = '原始数据')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.title('原始数据分布/origin data')
plt.legend(loc = 2)

# 聚类结果分布图
plt.subplot(1,2,2)
plt.scatter(irisFeatures[:,0],irisFeatures[:,1],c = model.labels,cmap
            = 'viridis',marker = 'o',label = '聚类结果')
u = np.array(model.u)
plt.scatter(u[:,0],u[:,1],c = 'red',marker = 'X',label = '中心点')
plt.xlabel('petal length')
```

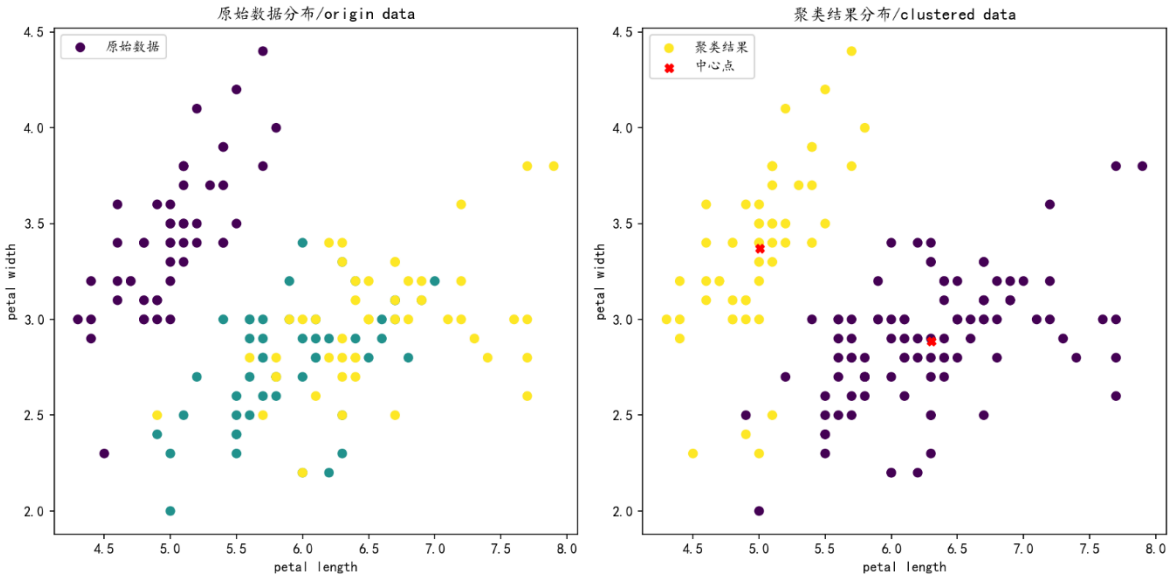
```
    plt.ylabel('petal width')
    plt.title('聚类结果分布/clustered data')
    plt.legend(loc = 2)

    plt.tight_layout()
    plt.show()
```

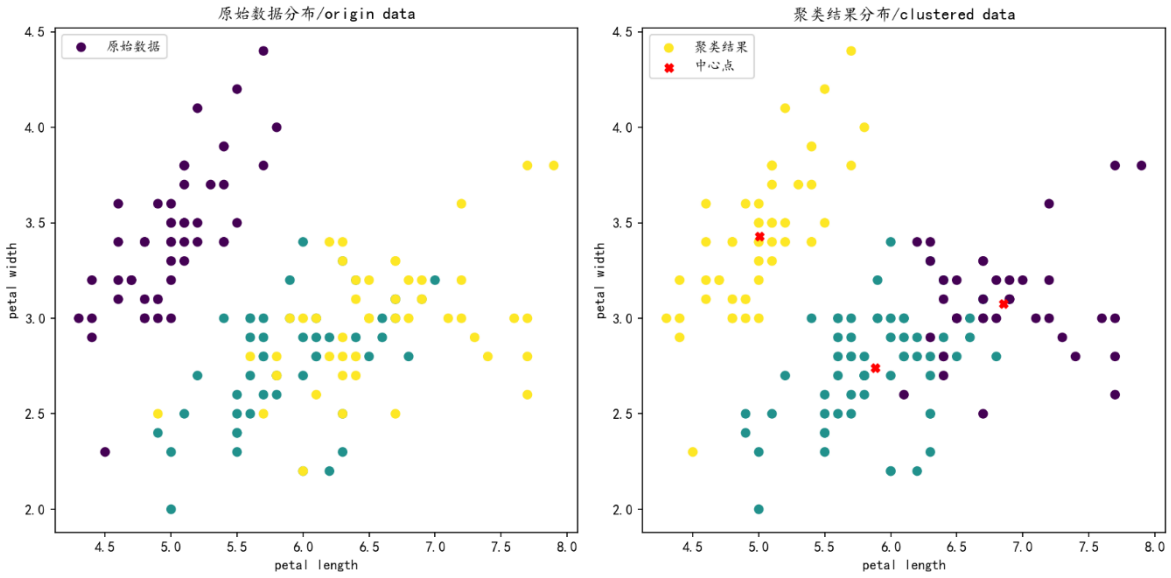
三、 计算结果与讨论

3.1. 计算结果

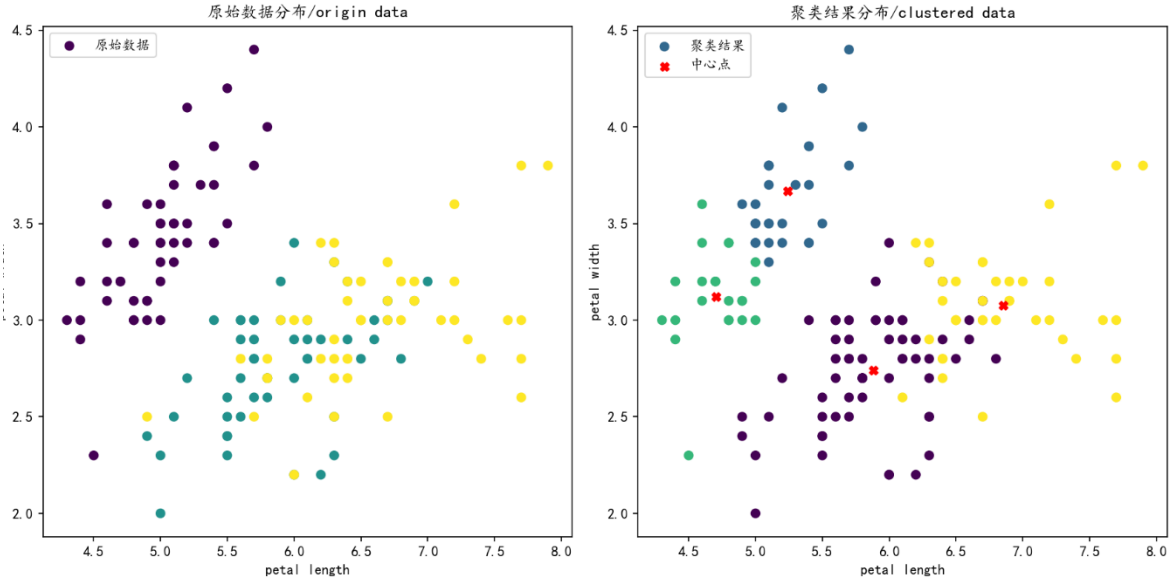
① 当 K=2, N=4 时，聚类结果：



② 当 K=3, N=4 时，聚类结果：



③ 当 K=4, N=4 时，聚类结果：



在实验中，采用鸢尾花数据集进行 K-means 聚类，设定不同的聚类数 K 值，并观察其对聚类结果的影响：

① 当 K=2，N=4 时，聚类结果显示大部分样本成功归类于两类中，但部分样本因特征相似性被错误分类。

② 当 K=3，N=4 时，聚类效果最佳，三个类别较好地与数据集中实际的三种鸢尾花类别对应，说明模型成功实现了聚类，分类准确性较高。

③ 当 K=4，N=4 时，算法将数据划分成四类，导致部分类中样本数较少，说明选择多于数据集实际类别数的 K 值会导致过拟合。

这些结果表明，适当的 K 值选择对于模型的分类效果至关重要，过大或过小的 K 值都会影响聚类的准确性。

在实验中，我们使用鸢尾花数据集进行 K-means 聚类，设定不同的聚类数 K 值，并观察其对聚类结果的影响：

① 当 K=2，N=4 时：聚类结果显示大部分样本成功地被归类为两个类别，但由于部分样本间特征的相似性，仍然存在一些错误分类的情况。

② 当 K=3，N=4 时：聚类效果最佳，三个类别与数据集中实际的三种鸢尾花类型高度对应，表明模型成功地实现了聚类，分类准确性较高。

③ 当 K=4，N=4 时：算法将数据划分为四类，导致某些类别中样本数量较少。这表明选择的 K 值大于数据集实际的类别数时，会产生过拟合现象。

这些结果表明，选择合适的 K 值对模型的分类效果至关重要；过大或过小的 K 值都会对聚类的准确性产生负面影响。

3.2. 实验讨论

实验结果表明，K-means 聚类算法对初始聚类中心的设置非常敏感，不同的初始点可能导致不同的聚类结果，这种现象称为“局部最优”。此外，数据集的类别数和特征分布同样影响模型的准确性。

1.初始聚类中心的影响：K-means 算法的初始聚类中心是随机选择的，这可能导致不同运行结果的差异。如果初始聚类中心位于样本的密集区域之外（即远离数据分布中心），K-means 算法容易陷入局部最优，部分样本可能被错误分类，进而影响聚类的准确性。为了减少这种情况，可以在初始化时采取策略，将聚类中心设定在样本点较为密集的区域，以提高算法的稳定性和收敛速度。

2.K 值对聚类结果的影响：：实验中考察了不同 K 值（如 K=2 和 K=4）对聚类结果的影响。K 值越大，划分的类别数越多，样本会被更细致地分配到不同的类别，但这可能导致过度划分，即将同一实际类别的样本分到多个类别。相反，K 值过小则可能将不同类别的样本归为同一类。因此，合理选择 K 值可以使聚类结果更符合实际类别的分布。实验结果显示，对于该数据集，K=3 是较为合适的选择，能够实现较高的聚类准确性，并与原始标签分布基本一致。

总之，实验结果和讨论揭示了 K-means 聚类在无监督分类中的应用效果，同时强调了初始聚类中心和 K 值选择对算法性能的重要影响。

四、 作业总结

在本次作业中，通过对鸢尾花数据集进行 K-means 聚类实验，我对无监督学习中的聚类算法有了更深入的理解。实验过程中，我使用 Python 实现了 K-means 算法，并对数据集进行了预处理和可视化分析，重点研究了初始聚类中心和 K 值对聚类效果的影响，同时对实验结果进行了详细的讨论。结合实验结果的可视化展示，我观察到不同类别样本的分布规律，以及距离计算在聚类中的关键作用。此外，这次实验也使我更加明确了聚类算法在实际应用中的优缺点及可能的改进方向，为后续的机器学习研究奠定了良好的基础。

本次实验的主要收获：

1.算法实现与理解：通过代码实现，我清晰理解了 K-means 算法的原理和步骤，包括初始化、迭代计算聚类中心及样本分配等过程。

2.数据预处理与可视化：在实验过程中，我学习了如何对数据进行标准化和可视化处理，这有助于更好地理解数据分布和聚类结果。

3.参数影响分析：通过改变初始聚类中心和 K 值，我观察到了其对聚类效果的显著影响，并学习了如何选择合适的参数以提高聚类的准确性。

4.算法优缺点：此次实验让我认识到 K-means 算法的优缺点，例如对初始中心的敏感性及容易陷入局部最优等问题。此外，我了解了改进算法性能的可能方法。

五、 参考文献

[1] 徐义春. 人工智能案例与实验[M]. 2024 年 5 月第 1 版. 清华大学出版社, 2024.

六、 实验代码

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

# 获取数据集并进行探索
iris = datasets.load_iris()
irisFeatures = iris['data']
irisFeaturesName = iris['feature_names']
irisLabels = iris['target']

def norm2(x):
  # 求 2 范数的平方值
  return np.sum(x * x)

class KMeans(object):
  def __init__(self,k = int,n = int):
    # k: 聚类数目，n: 数据集维度
    self.K = k
    self.N = n
    self.u = np.zeros((k,n))
    self.C = [[] for i in range(k)]
    self.labels = np.zeros(len(irisFeatures),dtype = int)
    # u[i]: 第i个类的中心点，C[i]: 第i类的数据点
```

```
    def fit(self,data: np.ndarray):
        # data: 每行是一个数据点
        self.select_u0(data)
        # 聚类中心初始化
        J = 0
        oldJ = 100
        while abs(J - oldJ) > 0.001:
            oldJ = J
            J = 0
            self.C = [[] for i in range(self.K)]
            # for x in data:
            #     nor = [norm2(self.u[i] - x) for i in range(self.K)]
            #     J += np.min(nor)
            #     self.C[np.argmin(nor)].append(x)
            # self.u = [np.mean(np.array(self.C[i]),axis = 0) for i in range(self.K)]
            for i,x in enumerate(data):
                nor = [norm2(self.u[j] - x) for j in range(self.K)]
                cluster_idx = np.argmin(nor)
                J += np.min(nor)
                self.C[cluster_idx].append(x)
                self.labels[i] = cluster_idx
            self.u = [np.mean(np.array(self.C[i]),axis = 0) for i in range(self.K)]

    def select_u0(self,data: np.ndarray):
        for j in range(self.N):
            # 得到该列数据的最大最小值
            minJ = np.min(data[:,j])
            maxJ = np.max(data[:,j])
            rangeJ = float(maxJ - minJ)
            # 聚类中心的第j维坐标随机初始化
            self.u[:,j] = minJ + rangeJ * np.random.rand(self.K)

model = KMeans(3,4)
# k = 3,n = 4
model.fit(irisFeatures)

# 绘制原始数据的分布图
plt.figure(figsize = (12,6))
plt.rcParams['font.sans-serif'] = ['KaiTi']

# 原始数据分布图
plt.subplot(1,2,1)
```

```
    plt.scatter(irisFeatures[:, 0], irisFeatures[:, 1], c = irisLabels, cmap = 'viridis', marker
                = 'o', label = '原始数据')
    plt.xlabel('petal length')
    plt.ylabel('petal width')
    plt.title('原始数据分布/origin data')
    plt.legend(loc = 2)

    # 聚类结果分布图
    plt.subplot(1, 2, 2)
    # x = np.array(model.C[0])
    # plt.scatter(x[:, 0], x[:, 1], c = 'red', marker = 'o', label = 'cluster1')
    # x = np.array(model.C[1])
    # plt.scatter(x[:, 0], x[:, 1], c = 'green', marker = '*', label = 'cluster2')
    # x = np.array(model.C[2])
    # plt.scatter(x[:, 0], x[:, 1], c = 'blue', marker = '+', label = 'cluster3')
    # u = np.array(model.u)
    # plt.scatter(u[:, 0], u[:, 1], c = 'black', marker = 'X', label = 'center')

    plt.scatter(irisFeatures[:, 0], irisFeatures[:, 1], c = model.labels, cmap
                = 'viridis', marker = 'o', label = '聚类结果')
    u = np.array(model.u)
    plt.scatter(u[:, 0], u[:, 1], c = 'red', marker = 'X', label = '中心点')
    plt.xlabel('petal length')
    plt.ylabel('petal width')
    plt.title('聚类结果分布/clustered data')
    plt.legend(loc = 2)

    plt.tight_layout()
    plt.show()
```