专业课程实验报告

课程名称： 模式识别

开课学期： 2022 至 2023 学年 第 1 学期

专　　业： 智能科学与技术

年级班级： 20 级 3 班

学生姓名： 严中圣

学生学号： 222020335220177

实验教师： 杨颂华

人工智能学院

《模式识别》实验课报告书

实验编号：	4	**实验名称:**	聚类算法实现
姓　　名：	严中圣	**学　　号：**	222020335220177
日　　期：	2022 年 12 月 31 日	**教师打分：**	

1 实验目标

聚类即非监督模式识别，一般需要对算法给出的结果给出进一步的分析。通过本实验旨在掌握典型的聚类原理和实现算法，将其应用到自定义的数据集。

2 实验环境

- PyCharm 2022.1.3 (Professional Edition)
- python 3.7.13, numpy 1.21.5
- OS: Windows 11 22H2, CPU:12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz

3 实验原理

聚类是针对给定的样本，依据它们特征的相似度或距离，将其归并到若干个类的数据分析问题。一个类是样本的一个子集。直观上，相似的样本聚集在相同的类，不相似的样本分散在不同的类。这里，样本之间的相似度或距离起着重要作用。聚类的目的是通过得到的类或簇来发现数据的特点或对数据进行处理，在数据挖掘、模式识别等领域有着广泛的应用。聚类属于无监督学习，因为只是根据样本的相似度或距离将其进行归类，而类事先并不知道。

聚类算法很多，本文介绍两种最常用的聚类算法: 层次聚类和 k 均值聚类。层次聚类分为聚合和分裂两种方法。聚合法开始将每个样本各自分到一个类：之后将相距最近的两类合并，建立一个新的类，重复此操作直到满足停止条件；得到层次化的类别。分裂法开始将所有样本分到一个类：之后将已有类中相距最远的样本分到两个新的类，重复此操作直到满足停止条件。k 均值聚类是基于中心的聚类方法，通过迭代，将样本分到 k 个类中，使得每个样本与其所属类的中心或均值最近，构成对空间的划分。

3.1 K-means 聚类方法

K-means 聚类方法基于最小误差平方和准则，给定 n 个样本的集合 $X=\{x_1,x_2,\cdots,x_n\}$。k 均值聚类的目标是将 n 个样本分到 k 个不同的类中，这里假设 $k<n$ 。k 个类 $G_1,G_2,\cdots,G_k$ 形成对样本集合 X 的划分，其中 $G_i\cap G_j=\varnothing,\bigcup_{i=1}^{k}G_i=X$ 。用 C 表示划分，一个划分对应着一个聚类结果。

k 均值聚类归结为样本集合 X 的划分，或者从样本到类的函数的选择问题。k 均值聚类的策略是通过损失函数的最小化选取最优的划分。然后, 定义样本与其所属类的中心之间的距离的总和为损失函数, 即

$$W(C)=\sum_{l=1}^{k}\sum_{C(i)=l}\|x_i-\bar{x}_l\|^2 \tag{1}$$

式中 $\bar{x}_l = (\bar{x}_{1l}, \bar{x}_{2l}, \cdots, \bar{x}_{ml})^{\mathrm{T}}$ 是第 l 个类的均值或中心,$n_l = \sum_{i=1}^{n} I(C(i) = l), I(C(i) = l)$ 是指示函数,取值为 1 或 0 。k 均值聚类就是求解最优化问题:

$$\begin{aligned} C^* &= \arg\min_C W(C) \\ &= \arg\min_C \sum_{l=1}^{k} \sum_{C(i)=l} \|x_i - \bar{x}_l\|^2 \end{aligned} \tag{2}$$

上式最优解求解问题是 NP 难问题，故无法用解析的方式求解最优解，只能用迭代的方式，每次迭代包括两个步骤。首先选择 k 个类的中心，将样本逐个指派到与其最近的中心的类中，得到一个聚类结果；然后更新每个类的样本的均值，作为类的新的中心：重复以上步骤，直到收敛为止。

考查下面的调整：如果把 $\boldsymbol{y}$ 从 C_k 类移到 C_j 类中，则这两个类别发生了变化，C_k 类少了一个样本而变成 $\tilde{C}_k$，C_j 类多了一个样本而变成 $\tilde{C}_j$，其余类别不受影响。这样调整后，两类的均值分别变为

$$\begin{aligned} \tilde{\boldsymbol{x}}_k &= \bar{\boldsymbol{x}_k} + \frac{1}{N_k - 1} [\bar{\boldsymbol{x}}_k - \boldsymbol{y}] \\ \tilde{\boldsymbol{x}}_j &= \bar{\boldsymbol{x}_j} + \frac{1}{N_j + 1} [\boldsymbol{y} - \bar{\boldsymbol{x}_j}] \end{aligned} \tag{3}$$

相应地,两类各自的误差平方和也分别变为

$$\begin{aligned} \widetilde{W}_k &= W_k - \frac{N_k}{N_k - 1} \|\boldsymbol{y} - \bar{\boldsymbol{x}_k}\|^2 \\ \widetilde{W}_j &= W_j + \frac{N_j}{N_j + 1} \|\boldsymbol{y} - \bar{\boldsymbol{x}_j}\|^2 \end{aligned} \tag{4}$$

总的误差平方和的变化只取决于这两个变化。显然,移出一个样本会带来 C_k 类均方误差的减小,移入这个样本又会导致 C_j 类的均方误差增大。如果隊小量大于增加量,即

$$\frac{N_j}{N_j + 1} \|\boldsymbol{y} - \bar{\boldsymbol{x}_j}\|^2 < \frac{N_k}{N_k - 1} \|\boldsymbol{y} - \bar{\boldsymbol{x}_k}\|^2 \tag{5}$$

则进行这一步搬运就有利于总体误差平方和的减少,否则不移动。同样道理,如果类别数 $c > 2$,则可以考虑 C_k 类之外的所有其他类,考查其中均方误差增加量最小的类别,如果最小增加量小于 C_k 类均方误差的减小量,则把样本 $\boldsymbol{y}$ 从 C_k 类移到均方误差增加量最小的类别中。以上就是 k 均值算法的核心思想。

3.2 层次聚类方法

分级聚类开始首先将每个样本各自分到一个类：之后将相距最近的两类合并，建立一个新的类，重复此操作直到满足停止条件：得到层次化的类别。分级聚类的具体过程如下:对于给定的样本集合，开始将每个样本分到一个类;然后按照一定规则，例如类间距离最小，将最满足规则条件的两个类进行合并：如此反复进行，每次减少一个类，直到满足停止条件，如所有样本聚为一类。分级聚类的算法步如下：

(1) 初始化，每个样本形成一个类；

(2) 合并：计算任意两个类之间的距离（或相似性），把距离最小（或相似性最大）的两个类合并为一类，记录下这两个类之间的距离（或相似性），其余类不变；

(3) 重复 (2)，直到所有样本被合并到两个类中。

4 实验步骤

4.1 数据集来源

本次实验采用经典鸢尾花数据集，Iris 数据集是一种常用的分类实验数据集，由 Fisher 于 1936 年收集和组织。数据集包含 150 个数据样本，分为 3 类，每类 50 个数据，每个数据包含 4 个属性。

4.2 K-means 聚类

K-means 聚类过程具体如下：首先确定超参数 k，其次随机选择 k 个点作为初始的聚类中心；对于剩下的点，根据其与聚类中心的距离，将其归入最近的簇；然后对于每个簇，计算所有点的均值最为新的聚类中心；重复以上步骤直到聚类中心不再发生改变即完成聚类；代码见附录一。

选择 K=3 的聚类结果如下所示，可见良好的达到了预期效果：

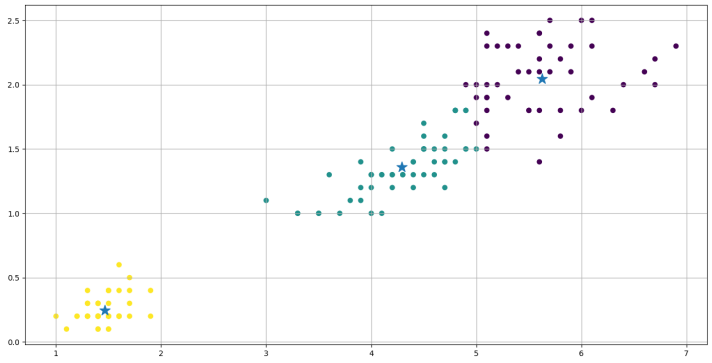

图 1: k-means 聚类结果

同时我们利用每次聚类的误差平方和确定聚类数目 K，如下图所示，可见 k=3 是最佳的聚类数目。

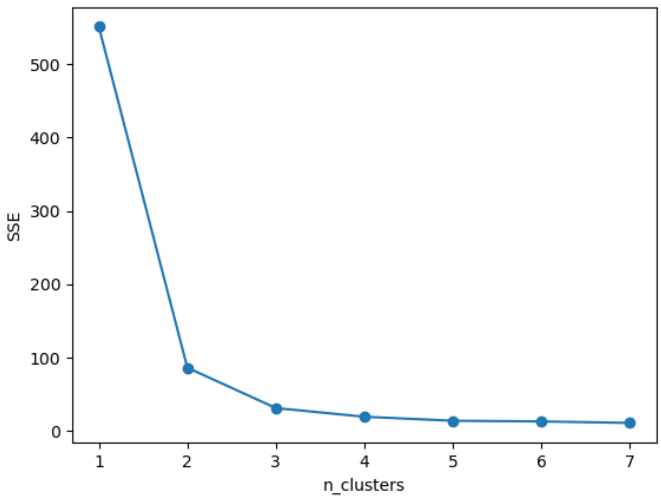

图 2: 确定聚类数目曲线图

4.3 分层聚类

层次聚类的程序设计思路如下：首先，从原始数据集中加载特征和标签，并将其存储到数据框中。然后，定义欧氏距离函数，用于计算两个样本之间的距离。接下来，编写层次聚类函数，输入特征和聚类数量，输出聚类结果和聚类中心。在这一步中首先将每个样本分配给一个初始簇，并初始化簇和聚类中心。然后，计算簇之间的距离矩阵，找到最小距离。接着，合并最近的两个簇，并更新聚类中心。重复上述步骤，直到聚类数量达到指定数量为止，返回聚类结果和聚类中心。最后，将聚类结果添加到原始数据框中，并可视化聚类结果和层次聚类树。详细代码见附录 2。

设定聚类数目为 3，分层聚类结果如下，较好地完成了分类任务：

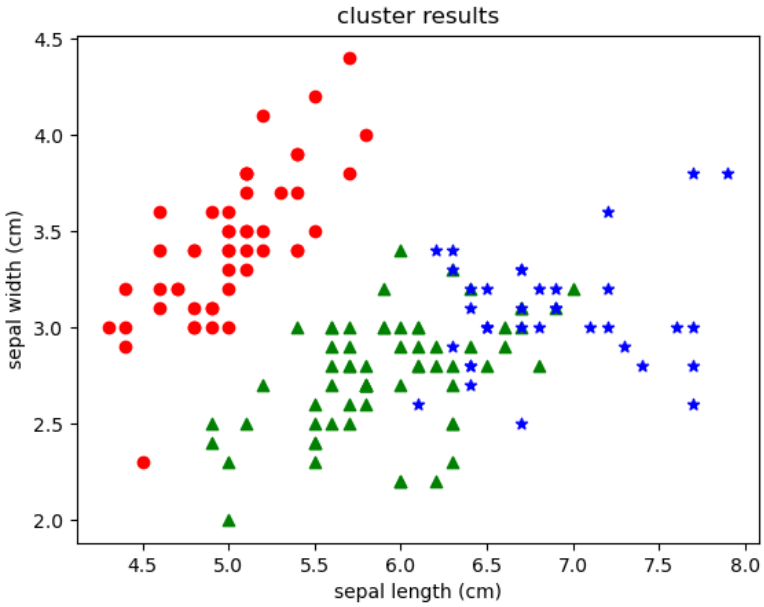

图 3: 分层聚类结果

分层聚类树如下所示：

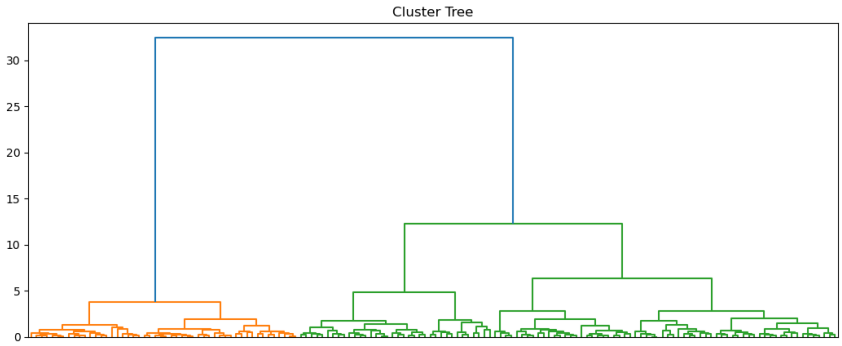

图 4: 分层聚类树

5 实验分析

本次实验发现 K-means 聚类和层次聚类在实现目标分类方面都很有效，通过比较 K-means 聚类和层次聚类的性能，发现 K-means 聚类能够快速聚类数据，并且能够找出数据的中心点；而层次聚类能够根

据数据之间的相似性划分成多个类别，并将数据依据深度和层次有序地展示出来，但是，K-means 聚类使用的是距离度量，容易受到离群点的影响，而层次聚类则使用的是相似度度量，能够更好地处理离群点。因此，K-means 聚类和层次聚类都是有效的聚类算法，可以根据实际情况选择合适的算法。

6 参考文献

[1] 周志华. 机器学习 [M]. 清华大学出版社, 2016.

[2] 李航. 统计学习方法 [M]. 清华大学出版社, 2019

[3] Christopher M. Bishop: Pattern Recognition and Machine Learning, Chapter 4.3.4

A 附录 1 – K-means 实验程序代码

```
import numpy as np
import random

class KMeans:
    def __init__(self, n_clusters=3, random_state=0):
        assert n_clusters >=1, " must be valid"
        self._n_clusters = n_clusters
        self._random_state = random_state
        self._X = None
        self._center = None
        self.cluster_centers_ = None

    def distance(self, M, N):
        return (np.sum((M - N) ** 2, axis = 1))** 0.5

    def _generate_labels(self, center, X):
        return np.array([np.argmin(self.distance(center, item)) for item
   in X])

    def _generate_centers(self, labels, X):
        return np.array([np.average(X[labels == i], axis=0) for i in np.
   arange(self._n_clusters)])

    def fit_predict(self, X):
        k = self._n_clusters
        if self._random_state:
            random.seed(self._random_state)
        center_index = [random.randint(0, X.shape[0]) for i in np.arange(
   k)]
        center = X[center_index]
        n_iters = 1e3
        while n_iters > 0:
            last_center = center
```

```
            labels = self._generate_labels(last_center, X)
            self.labels_ = labels
            center = self._generate_centers(labels, X)
            self.cluster_centers_ = center
            if (last_center == center).all():
                self.labels_ = self._generate_labels(center, X)
                break
            n_iters = n_iters - 1
        sses = []
        for i in range(len(X)):
            sses.append(((X[i] - center[labels[i]])**2).sum())
        sses = np.array(sses)
        self.sse = sses.sum()
        return self
```

B 附录 2 – 层次聚类实验完整代码

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
iris = load_iris()
X = iris.data
features = iris.feature_names
y = iris.target
target_names = iris.target_names
df = pd.DataFrame(X, columns=features)
df['label'] = y

def euclidean_distance(x1, x2):
    dist = np.sqrt(np.sum((x1 - x2) ** 2))
    return dist

def hierarchical_clustering(X, n_clusters):
    # 初始化簇和聚类中心
    clusters = [[i] for i in range(len(X))]
    centroids = np.array(X)

    while len(clusters) > n_clusters:
        dist_matrix = np.zeros((len(clusters), len(clusters)))
        for i in range(len(clusters)):
            for j in range(i+1, len(clusters)):
                dist_matrix[i][j] = euclidean_distance(centroids[i],
   centroids[j])
                dist_matrix[j][i] = dist_matrix[i][j]
```

```
        # 找出最短距离
        min_dist = 10000
        x = 0
        y = 0
        for i in range(len(clusters)):
            for j in range(i+1,len(clusters)):
                if  dist_matrix[i][j] < min_dist:
                    x = i
                    y = j
                    min_dist = dist_matrix[i][j]

        # 合并簇
        clusters[x].extend(clusters[y])
        del clusters[y]

        # 更新聚类中心
        centroid_x = np.mean(X[clusters[x]], axis=0)
        centroids[x] = centroid_x
        centroids = list(centroids)
        del centroids[y]
        centroids = np.array(centroids)

    return clusters, centroids

clusters, centroids = hierarchical_clustering(X, 3)
df['cluster'] = 0
for i in range(len(clusters)):
    for j in range(len(clusters[i])):
        df.loc[clusters[i][j], 'cluster'] = i
markers = ['o', '^', '*']
colors = ['r', 'g', 'b']
plt.figure()
for i in range(len(clusters)):
    x = df[df['cluster'] == i]
    plt.scatter(x['sepal length (cm)'], x['sepal width (cm)'], marker=
   markers[i], c=colors[i])
plt.title('cluster results')
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.show()

# 层次聚类树
from scipy.cluster.hierarchy import dendrogram, linkage
Z = linkage(X, 'ward')
plt.figure(figsize=(20, 5))
dendrogram(Z, labels=y, leaf_rotation=90, leaf_font_size=8)
plt.title('Cluster Tree')
```

```
plt.show()
```