序号（学号）：222021335210174　　实验成绩：

西南大学人工智能学院

实验报告

学年学期	2023 学年第一学期
课程名称	模式识别
姓　名	杨东山
学　院	人工智能学院
专　业	智能科学与技术
班　级	21 级智科 3 班
任课教师	**周跃**

2023　年　12　月　18　日

实验项目	聚类算法实现

一、实验目的：

1.知识目标 掌握基本的聚类算法，理解聚类算法的设计和实现。

2.能力目标 能够实现基本的聚类算法，对聚类的结果做出合理的解释。

3.素质目标 了解聚类算法当前的研究现状。

二、算法原理概述：

聚类即非监督模式识别。

1. K 均值聚类是最基础常用的聚类算法。它的基本思想是，通过迭代寻找 K 个簇（Cluster）的一种划分方案，使得聚类结果对应的损失函数最小。其中，损失函数可以定义为各个样本距离所属簇中心点的误差平方和。

- 基本概念：
  - 要得到簇的个数，需要指定K值
  - 质心：均值，即向量各维取平均即可
  - 距离的度量：常用欧几里得距离和余弦相似度（先标准化）
  - 优化目标：$\min \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2$

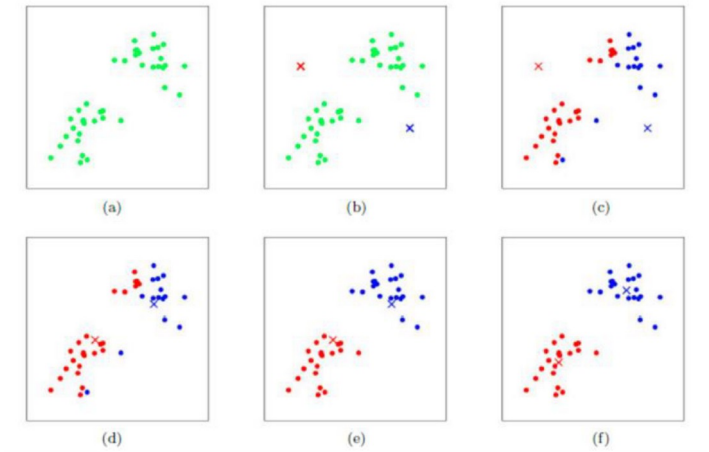
i:第几个簇；

c_i:第 i 个簇的质心；

C_i:第 i 的簇里面所有的点的集合

dist()：两点间的欧氏距离

工作流程：



装

订

线

2. DBSCAN（Density-Based Spatial Clustering of Applications with Noise）是一种基于密度的聚类算法，python 中的 sklearn.cluster 库可以实现 DBSCAN 聚类。

✓ 基本概念：（Density-Based Spatial Clustering of Applications with Noise）

- 核心对象：若某个点的密度达到算法设定的阈值则其为核心点。（即 r 邻域内点的数量不小于 minPts）
- ϵ-邻域的距离阈值：设定的半径r
- 直接密度可达：若某点p在点q的 r 邻域内，且q是核心点则p-q直接密度可达。
- 密度可达：若有一个点的序列q0、q1、...qk，对任意qi-qi-1是直接密度可达的，则称从q0到qk密度可达，这实际上是直接密度可达的“传播”。
- 密度相连：若从某核心点p出发，点q和点k都是密度可达的,则称点q和点k是密度相连的。
- 边界点:属于某一个类的非核心点,不能发展下线了
- 直接密度可达：若某点p在点q的 r 邻域内，且q是核心点则p-q直接密度可达。
- 噪声点：不属于任何一个类簇的点，从任何一个核心点出发都是密度不可达的

算法原理：由密度可达关系导出的最大密度相连的样本集合,即为我们最终聚类的一个类别,或者说一个簇

工作流程：伪码如下

算法：DBSCAN，一种基于密度的聚类算法
输入：
D：一个包含 n 个对象的数据集
ε：半径参数
MinPts：领域密度阀值
输出：基于密度的簇的集合
方法：
1. 标记所有对象为 unvisited;
2. Do
3. 随机选择一个 unvisited 对象 p;
4. 标记 p 为 visited;
5. If p 的 ε－领域至少有 MinPts 个对象
6. 创建一个新簇 C，并把 p 添加到 C;
7. 令 N 为 p 的 ε－领域 中的对象集合
8. For N 中每个点 p'
9. If p'是 unvisited;
10. 标记 p'为 visited;
11. If p'的 ε－领域至少有 MinPts 个对象，把这些对象添加到 N;
12. 如果 p'还不是任何簇的成员，把 p'添加到 C;
13. End for;
14. 输出 C;
15. Else 标记 p 为噪声;
16. Until 没有标记为 unvisited 的对象;

装 订 线

三、算法优缺点分析

1. k-means 算法

优势：简单，快速，适合常规数据集

劣势：

K 值难确定

复杂度与样本呈线性关系

很难发现任意形状的簇

2. DBSCAN 算法

优势：

不需要指定簇个数

擅长找到离群点（检测任务）

可以发现任意形状的簇

两个参数就够了

劣势：

高维数据有些困难（可以做降维）

Sklearn 中效率很慢（数据削减策略）

参数难以选择（参数对结果的影响非常大）

四、程序设计：

1. K-means 算法

（1） 定义 KMeans 类，详细代码见附录 1，主要模块如下：

1 def __init__(self, data, num_clustres)：进行类的初始化。

2 def train(self, max_iterations):进行训练。

3 def centroids_init(data, num_clustres):初始化质心。

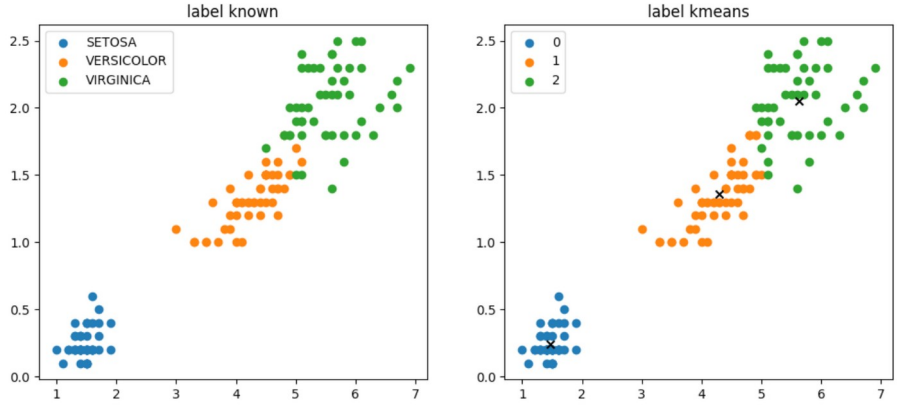
4 def centroids_find_closest(data, centroids):为每个点找到最近的质心。

5 def centroids_compute(data, closest_centroids_ids,

I

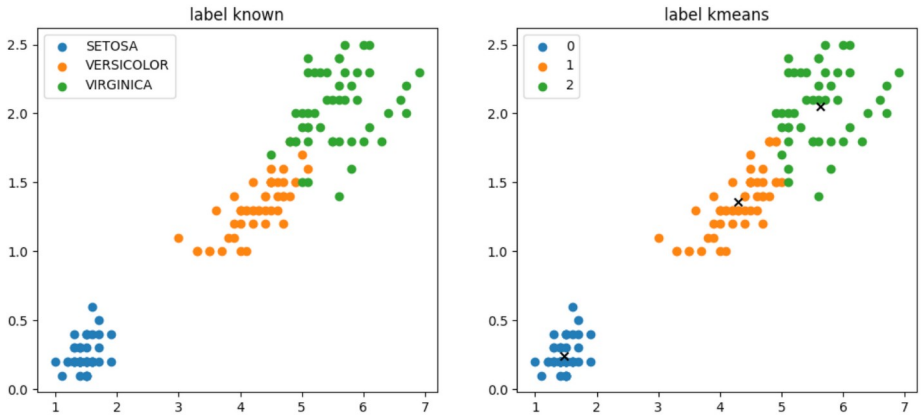
（2 略

标签



装 订 线

（3） 指定参数，簇数量设置为 3，最大迭代次数设置为 50，经过训练后，结果如图所示：

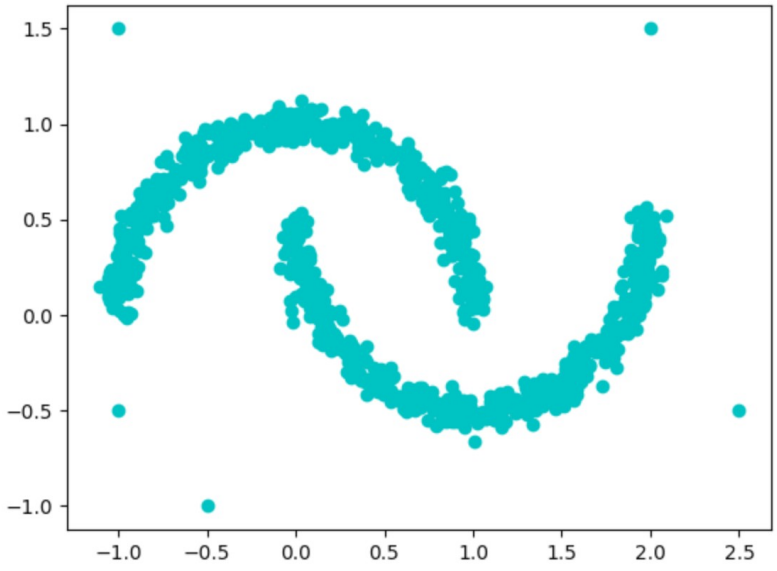


由图可知，程序运行成功，虽训练结果与真实数据有所偏差，但在可接受范围内，算法性能较高。

2. DBSCAN 算法

（1） 本次实验中，选择调用 sklearn 库中的 DBSCAN 类进行实验。

（2） 使用 sklearn 中的 make_moons()函数制作月亮型数据，初始数据如下：



装 订 线

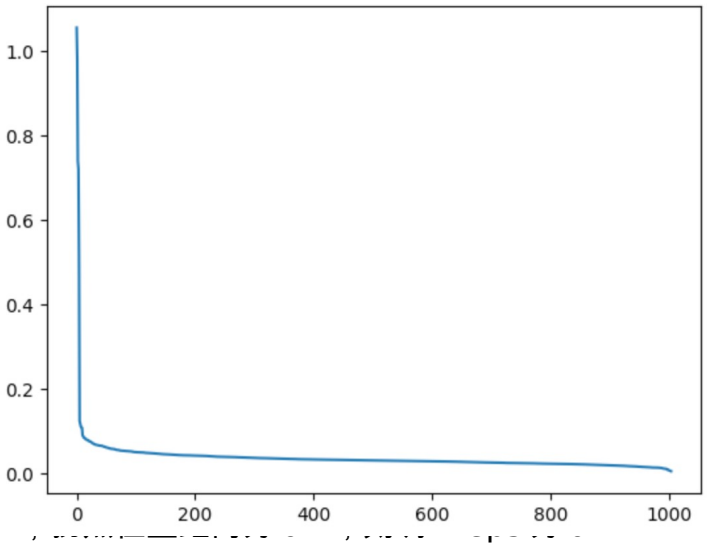
（3） 选择 eps（邻域半径）和 min_samples（最少样本数），对于邻域半径的选取我们可以利用 k-distance 碎石图来实现：

① 选取 k 值，建议取 k 为 2*维度-1。（其中维度为特征数）

② 计算并绘制 k-distance 图。（计算出每个点到距其第 k 近的点的距离，然后将这些距离从大到小排序后进行绘图。）

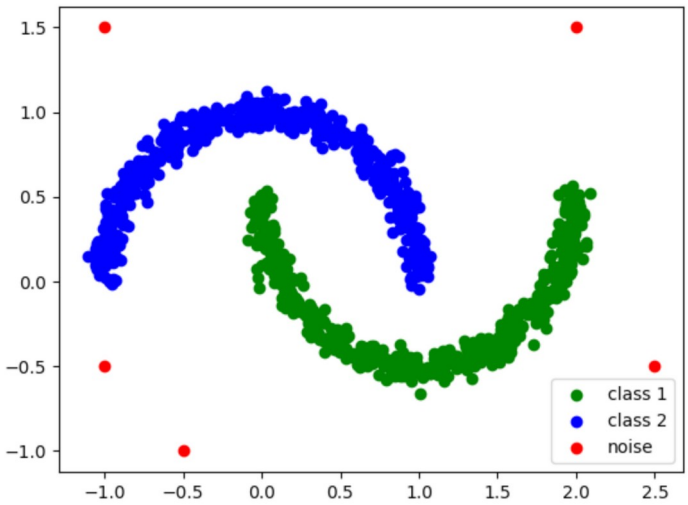
③ 找到拐点位置的距离，即为 Eps 的值。

min_samples 的取值为上述 k 值加 1，即：min_samples=k + 1 。



可[illegible]

（4） 建立模型，进行训练，结果如图：



可知，程序运行成功，如图中，红色点为噪点。

装 订 线

五、**总结分析**：

本次实验分析实现了 k-means 与 dbscan 两种聚类算法，对算法的原理进行了深入的了解，根据原理编写了代码并分别对鸢尾花数据集与月亮型数据集进行了聚类实验，结果都相当客观，同时编程能力也有了极大的提升。

附录 源代码：

k_means.py

```
import numpy as np


class KMeans:
    def __init__(self, data, num_clustres):
        self.data = data
        self.num_clustres = num_clustres

    def train(self, max_iterations):
        # 1.先随机选择K个中心点
        centroids = KMeans.centroids_init(self.data, self.num_clustres)
        # 2.开始训练
        num_examples = self.data.shape[0]
        closest_centroids_ids = np.empty((num_examples, 1))
        for _ in range(max_iterations):
            # 3 得到当前每一个样本点到K个中心点的距离，找到最近的
            closest_centroids_ids = KMeans.centroids_find_closest(self.data,
centroids)
            # 4.进行中心点位置更新
            centroids = KMeans.centroids_compute(self.data,
closest_centroids_ids, self.num_clustres)
        return centroids, closest_centroids_ids

    # 初始化质心
    @staticmethod
    def centroids_init(data, num_clustres):
        num_examples = data.shape[0]
        random_ids = np.random.permutation(num_examples)
        centroids = data[random_ids[:num_clustres], :]
        return centroids

    # 为每个点找到最近的质心
    @staticmethod
    def centroids_find_closest(data, centroids):
        num_examples = data.shape[0]
        num_centroids = centroids.shape[0]
        closest_centroids_ids = np.zeros((num_examples, 1))
        for example_index in range(num_examples):
            distance = np.zeros((num_centroids, 1))
            for centroid_index in range(num_centroids):
```

```
                distance_diff = data[example_index, :] -
centroids[centroid_index, :]
                distance[centroid_index] = np.sum(distance_diff ** 2)
            closest_centroids_ids[example_index] = np.argmin(distance)
        return closest_centroids_ids

    # 计算，更新质心
    @staticmethod
    def centroids_compute(data, closest_centroids_ids, num_clustres):
        num_features = data.shape[1]
        centroids = np.zeros((num_clustres, num_features))
        for centroid_id in range(num_clustres):
            closest_ids = closest_centroids_ids == centroid_id
            centroids[centroid_id] =
np.mean(data[closest_ids.flatten(), :], axis=0)
        return centroids
```

k_means_main.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from k_means import KMeans

# 读取鸢尾花数据集
data = pd.read_csv('data/iris.csv')
iris_types = ['SETOSA','VERSICOLOR','VIRGINICA']

x_axis = 'petal_length'
y_axis = 'petal_width'

# 绘制已知标签分类数据集与未分类数据集
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
for iris_type in iris_types:
    plt.scatter(data[x_axis][data['class']==iris_type],data[y_axis]
[data['class']==iris_type],label = iris_type)
plt.title('label known')
plt.legend()

plt.subplot(1,2,2)
plt.scatter(data[x_axis][:],data[y_axis][:])
plt.title('label unknown')
plt.show()

num_examples = data.shape[0]
x_train = data[[x_axis,y_axis]].values.reshape(num_examples,2)

#指定好训练所需的参数
num_clusters = 3
max_iteritions = 50

k_means = KMeans(x_train,num_clusters)
centroids,closest_centroids_ids = k_means.train(max_iteritions)

# 对比结果，绘制已知标签分类数据集 与 使用算法进行分类后的数据
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
for iris_type in iris_types:
    plt.scatter(data[x_axis][data['class']==iris_type],data[y_axis]
```

```
[data['class']==iris_type],label = iris_type)
plt.title('label known')
plt.legend()

plt.subplot(1,2,2)
for centroid_id, centroid in enumerate(centroids):
   current_examples_index = (closest_centroids_ids ==
centroid_id).flatten()
   plt.scatter(data[x_axis][current_examples_index],data[y_axis]
[current_examples_index],label = centroid_id)

for centroid_id, centroid in enumerate(centroids):
   plt.scatter(centroid[0],centroid[1],c='black',marker = 'x')
plt.legend()
plt.title('label kmeans')
plt.show()
```

dbscan.py

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

np.random.seed(2021)

# 整理数据
data = np.ones([1005, 2])
data[:1000] = make_moons(n_samples=1000, noise=0.05, random_state=2022)[0]
data[1000:] = [[-1, -0.5],
               [-0.5, -1],
               [-1, 1.5],
               [2.5, -0.5],
               [2, 1.5]]
print(data.shape)
plt.scatter(data[:, 0], data[:, 1], color="c")
plt.show()


# 选择 eps 和 min_samples
def select_MinPts(data, k):
   k_dist = []
   for i in range(data.shape[0]):
       dist = (((data[i] - data) ** 2).sum(axis=1) ** 0.5)
       dist.sort()
       k_dist.append(dist[k])
   return np.array(k_dist)


k = 3  # 此处 k 取 2*2 -1
k_dist = select_MinPts(data, k)
k_dist.sort()
plt.plot(np.arange(k_dist.shape[0]), k_dist[::-1])
plt.show()  # 展示原始数据

# 建立模型
dbscan_model = DBSCAN(eps=0.1, min_samples=k + 1)
label = dbscan_model.fit_predict(data)
class_1 = []
class_2 = []
noise = []
```

```
for index, value in enumerate(label):
    if value == 0:
        class_1.append(index)
    elif value == 1:
        class_2.append(index)
    elif value == -1:
        noise.append(index)
plt.scatter(data[class_1, 0], data[class_1, 1], color="g", label="class
1")
plt.scatter(data[class_2, 0], data[class_2, 1], color="b", label="class
2")
plt.scatter(data[noise, 0], data[noise, 1], color="r", label="noise")
plt.legend()
plt.show()
```