

数据挖掘实验报告

学院: 软件学院

专业: 软件工程

|  |  |
| --- | --- |
| *作者:*  周治平 | *指导教师:*  陶乾 |
|  |  |
| *学号:*  202030483178 | *班级:*  20级软件工程1班 |

2020-10-24

# K-Means与DBSCAN聚类算法实验

**摘要**

聚类分析又称群分析，它是研究（样品或指标）分类问题的一种统计分析方法，同时也是数据挖掘的一个重要算法，能够将数据分成有意义的、有用的群。如果有意义的群是目标，那么聚类应该捕捉到数据的本身结构信息。然而，在某些情况下，聚类分析被用来进行数据总结，以减少数据的大小。无论是为了理解还是实用，聚类分析长期以来在各种领域都发挥着重要作用：心理学和其他社会科学、生物学、统计学、模式识别、信息检索、机器学习和数据挖掘。本次实验，我们将进行两种K-Means变体（二分K-Means算法、K-Means++算法）以及DBSCAN算法的代码实现工作，进一步理解这三种聚类算法的原理，并通过可视化工具直观感受两种方法的聚类效果。

## **介绍**

1. 进一步理解二分K-means、K-Means++算法和DBSCAN算法的原理。
2. 体会三种不同的聚类算法的聚类效果。
3. 熟悉anaconda环境配置。

## **方法和理论**

1 K-Means算法模型

1.1基本介绍

K-MEANS算法是输入[聚类](https://baike.so.com/doc/5926838-6139761.html" \t "_blank)个数k，以及包含 n个[数据对象](https://baike.so.com/doc/9696829-10043128.html" \t "_blank)的数据库，输出满足方差最小标准k个聚类的一种算法。k-means 算法接受输入量 k ;然后将n个[数据对象](https://baike.so.com/doc/9696829-10043128.html" \t "_blank)划分为 k个[聚类](https://baike.so.com/doc/5926838-6139761.html" \t "_blank)以便使得所获得的聚类满足:同一聚类中的对象相似度较高;而不同聚类中的对象相似度较小。

1.2算法伪代码



2 二分K-Means算法模型

2.1基本介绍

二分K-Means算法首先将所有点作为一个簇，然后将该簇一分为二。之后选择其中一个簇继续进行划分，选择哪一个簇进行划分取决于对其划分是否可以最大程度降低SSE的值。上述基于SSE的划分过程不断重复，直到得到用户指定的簇数目为止。

2.2算法伪代码



3 K-means++算法模型

3.1基本介绍

K-Means++算法在聚类中心的初始化过程中的基本原则是使得初始的聚类中心之间的相互距离尽可能远。其思想是：假设已经选取了n个初始聚类中心(0<n<K)，则在选取第n+1个聚类中心时：距离当前n个聚类中心越远的点会有更高的概率被选为第n+1个聚类中心。在选取第一个聚类中心(n=1)时同样通过随机的方法（聚类中心互相离得越远越好）

3.2算法伪代码

4 DBSCAN算法模型

4.1算法介绍

DBSCAN（Density—Based Spatial Clustering of Application with Noise）算法是一种典型的基于密度的聚类方法。它将簇定义为密度相连的点的最大集合，能够把具有足够密度的区域划分为簇，并可以在有噪音的空间数据集中发现任意形状的簇。

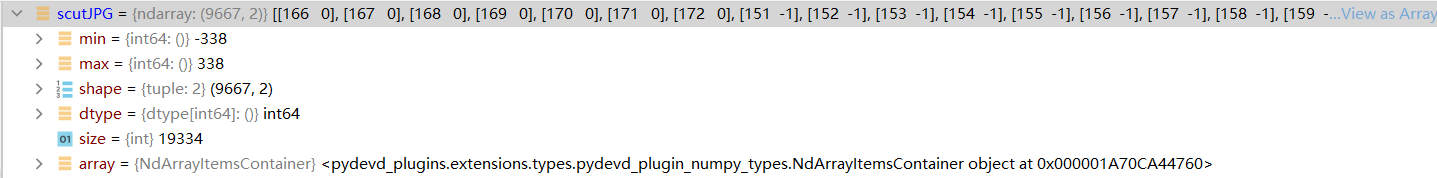
4.2 DBSCAN算法模型伪代码



## **实验**

### 数据集

1数据内容

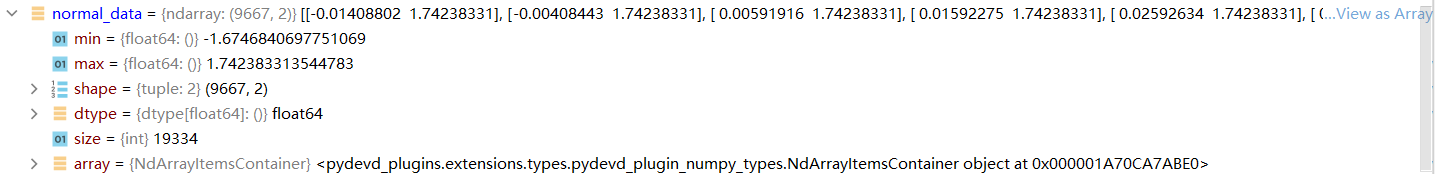


（Figure. 1. 华工校徽的数据集内容）

2数据数量

由图可知，一共有9667个样本，每个样本有2个特征，因此一共有19334个数据

3训练数据集



（Figure. 2.训练数据集的数据信息）

4验证数据集

由于本次实验为聚类实验，因此无验证数据集信息。

### 实现

0环境配置

import numpy as np  
from sklearn.preprocessing import StandardScaler, MinMaxScaler  
import matplotlib.pyplot as plt  
import sklearn.datasets as datasets  
import DBSCAN  
import KMeansRelevant

import numpy as np  
from scipy.spatial.distance import pdist  
from scipy.spatial.distance import squareform  
import torch

import numpy as np  
import sys

将代码导入Pycharm后，通过使用Anaconda3的Python环境，通过Alt+Shift+Enter便能自动安装以上所有包的环境

1初始化过程

1.1数据导入以及预处理

scutJPG = np.load(**"scutVec.npy"**) *# 华工校徽*

dataSet = scutJPG *# ['noisy\_circles', 'noisy\_moons', 'blobs', 'scutJPG']  
# 2, 对数据进行处理*scaler = StandardScaler()  
normal\_data = scaler.fit\_transform(dataSet)

1.2定义绘图函数

def show\_cluster\_result(cluster, normal\_data, cluster\_method):  
 *# 第一幅图展示原始图* plt.figure(figsize=(24, 10), dpi=80)  
 plt.subplot(1, 2, 1)  
 plt.scatter(normal\_data[:, 0], normal\_data[:, 1], color=**'black'**)*#列号0为X轴，列号1位Y轴* plt.title(**'raw graph'**)  
  
 *# 第二幅图聚类结果* plt.subplot(1, 2, 2)  
 plt.scatter(normal\_data[:, 0], normal\_data[:, 1], c=cluster, marker=**'o'**)*#根据聚类结果为不同的簇分配不同的颜色* plt.title(cluster\_method)  
 plt.legend(labels=[**'x'**, **'y'**])  
 plt.show()

1.3选择相应的聚类算法并运行

cluster\_method = **"biKMeans"** *# ['kMeans', 'biKMeans', 'kMeans++', 'DBSCAN']*if cluster\_method == **"DBSCAN"**:  
 *# DBSCAN的参数设置与运行, 请根据K-dis图像来获得大致的eps值* min\_pts = 8*#最大点是8个  
 # 查看k-dist* show\_K\_dis(normal\_data, min\_pts)  
 eps = 0.08 *# 0.09 距离范围为0.09* cluster = DBSCAN.dbscan(normal\_data, eps, min\_pts)  
elif cluster\_method == **"biKMeans"**: *# 运行二分K-means算法，请指定k值* cluster = KMeansRelevant.biKMeans(normal\_data, 10)  
elif cluster\_method == **"kMeans++"**: *# 运行K-means++算法，请指定k值* cluster = KMeansRelevant.kMeansPP(normal\_data, 10)  
else: *# 默认其他跑KMeans* cluster = KMeansRelevant.origin\_kMeans(normal\_data, 10)

1.5定义随机获取质心函数

def randCent(dataSet, k):  
 *"""  
 随机生成k个点作为质心，其中质心均在整个数据数据的边界之内  
 """* n = dataSet.shape[1] *# 获取数据的维度* centroids = np.mat(np.zeros((k, n)))  
 for j in range(n):  
 minJ = np.min(dataSet[:, j])  
 rangeJ = np.float(np.max(dataSet[:, j]) - minJ)  
 centroids[:, j] = minJ + rangeJ \* np.random.rand(k, 1)  
 return centroids

1.6定义K-Means++生成质心法

def kpp\_initialize(dataSet, k):  
 *"""  
 用于K-Means++的初始化质心  
 """* centroids=[]*#质心矩阵  
 #随机选择一个质心* centroids.append(dataSet[np.random.randint(dataSet.shape[0]),:])*#随机选取一个样本作为质心* for c\_id in range(k-1):*#随机生成k个质心* dist=[]  
 for i in range(dataSet.shape[0]):  
 point=dataSet[i,:]*#选取第i个样本，这里要对每一个样本都要遍历一次* d=sys.maxsize  
 for j in range(len(centroids)):  
 temp\_dist=distEclud(point,centroids[j])*#第j个质心与该样本的距离* d=min(d,temp\_dist)*#选取该样本与所有质心之中的最小距离的一个* dist.append(d)*#将该距离加入距离矩阵中* dist=np.array(dist)  
 next\_centroid=dataSet[np.argmax(dist),:]*#选取距离矩阵中距离最大的一个，通过argmax获取该最大值的下标* centroids.append(next\_centroid)*#更新质心集合* centroids=np.array(centroids)*#将最终的所有质心转换成nparray并返回* return centroids

1.7定义欧氏距离函数

def distEclud(vecA, vecB):  
 *"""  
 计算两个向量的欧式距离  
 """* return np.sqrt(np.sum(np.power(vecA - vecB, 2)))

2模型实现

2.1基础K-Means算法模型

*#createCent为生成质心的方法，此处通过随机生成的方式生成质心  
#distMeas为计算距离的方法*def origin\_kMeans(dataSet, k, disMeas=distEclud):  
 \_, clusterAssment = kMeans(dataSet, k, distMeas=disMeas, createCent=randCent)  
 return clusterAssment[:, 0].A[:, 0] *#这里是将二维数组clusterAssment[:, 0]转换为一维数组*

def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):  
 *"""  
 k-Means聚类算法,返回最终的k各质心和点的分配结果  
 """* m = dataSet.shape[0] *# 获取样本数量  
 # 构建一个簇分配结果矩阵，共两列，第一列为样本所属的簇类值，第二列为样本到簇质心的误差* clusterAssment = np.mat(np.zeros((m, 2)))*#m行2列  
 # 1. 初始化k个质心* centroids = createCent(dataSet, k)  
 clusterChanged = True*#判断簇分配是否继续分配* while clusterChanged:  
 clusterChanged = False  
 for i in range(m):  
 minDist = np.inf*#无穷大，计算其到k个质心的距离* minIndex = -1*#设置该簇为未分配状态  
 # 2. 找出最近的质心* for j in range(k):  
 distJI = distMeas(centroids[j, :], dataSet[i, :])*#获取此样本与每一个质心之间的距离* if distJI < minDist:  
 minDist = distJI*#更新距离* minIndex = j*#更新分配编号  
 # 3. 更新每一行样本所属的簇* if clusterAssment[i, 0] != minIndex:*#判断是否有簇分配发生变化* clusterChanged = True  
 clusterAssment[i, :] = minIndex, minDist \*\* 2*#质心，到质心的距离* print(centroids) *# 打印质心  
 # 4. 更新质心* for cent in range(k):  
 ptsClust = dataSet[np.nonzero(clusterAssment[:, 0].A == cent)[0]]  
 *# 获取给定簇的所有点，A应该是该下标下点的值，这里获取的是行号，np.nonzero(clusterAssment[:, 0].A == cent)返回的是矩阵对象* centroids[cent, :] = np.mean(ptsClust, axis=0) *# 沿矩阵列的方向求均值* return centroids, clusterAssment*#centroids为所有质心的坐标，clusterAssment为每一个点所分配的质心以及距离*

2.2二分K-Means算法模型

def biKMeans(dataSet, k, distMeas=distEclud):  
 *"""  
 二分k-Means聚类算法,返回最终的k各质心和点的分配结果  
 """* m = dataSet.shape[0] *# 获取样本数量  
 # 构建一个簇分配结果矩阵，共两列，第一列为样本所属的簇类值，第二列为样本到簇质心的误差* clusterAssment = np.mat(np.zeros((m, 2))) *# m行2列  
   
 #获取数据集中所有数据坐标的均值，生成一个初始质心，所有样本点都分配给它* centroid0=np.mean(dataSet,axis=0).tolist()[0]  
 centList=[centroid0]*#可变数组长度  
  
 #计算每一个点到质心的距离* for j in range(m):  
 clusterAssment[j,1]=distMeas(np.mat(centroid0),dataSet[j,:])\*\*2  
  
 *#选出k个质心* while len(centList)<k:  
 lowestSSE=np.inf;*#初始化误差总和为无穷大* for i in range(len(centList)):  
 pstInCurrCluster=dataSet[np.nonzero(clusterAssment[:,0].A==i)[0],:]*#对于第i个簇中的所有点，全部赋值，然后二分* centroidMat,splitClustAss=kMeans(pstInCurrCluster,2,distMeas)*#对第i个簇进行K-Means算法* sseSplit=sum(splitClustAss[:,1])*#计算需要分裂的簇与不需要分裂的簇之间的误差之和* sseNotSplit=sum(clusterAssment[np.nonzero(clusterAssment[:,0].A!=i)[0],1])  
 if(sseSplit+sseNotSplit<lowestSSE):*#选出效果最好的，即误差最小的* bestCentToSplit=i *#选出最适合分裂的簇号* bestNewCents=centroidMat *#两个新簇的质心* bestClustAss=splitClustAss.copy() *#复制新簇的点集归属* lowestSSE=sseSplit+sseNotSplit *#更新lowestSSE  
 #更新簇编号 0更新为划分簇，1更新为新簇编号，确实要先赋值len(centList)，否则先如果先赋值bestCentToSplit，  
 #当bestCentToSplit=1时，会导致可能分配的簇是1的簇又分配给len(centList)* bestClustAss[np.nonzero(bestClustAss[:, 0].A == 1)[0], 0] = len(centList)  
 bestClustAss[np.nonzero(bestClustAss[:,0].A==0)[0],0]=bestCentToSplit  
  
 print(**"the bestCentToSplit is:"**,bestCentToSplit)  
 print(**"the len of bestClustAss is "**,len(centList))  
  
 *#增加质心* centList[bestCentToSplit]=bestNewCents[0,:]  
 centList.append(bestNewCents[1,:])  
  
 *#更新簇分配结果(更新原来分配编号是i的簇)* clusterAssment[np.nonzero(clusterAssment[:,0].A==bestCentToSplit)[0],:]=bestClustAss  
 *#bestClustAss里面已经包含了簇的分配编号以及误差，分配编号以及误差同时更新  
  
 #print(clusterAssment[:, 0])  
 #print(clusterAssment[:, 0].A[:, 0])* return clusterAssment[:,0].A[:,0]*#将二维数组转换成一维数组*

2.3 K-Means++算法模型

*#createCent为生成质心的方法，K-Means++的方式选取质心  
#distMeas为计算距离的方法*def kMeansPP(dataSet, k, disMeas=distEclud):  
 \_, clusterAssment = kMeans(dataSet, k, distMeas=disMeas, createCent=kpp\_initialize)  
 return clusterAssment[:, 0].A[:, 0]*#这里是将二维数组clusterAssment[:, 0]转换为一维数组*

2.4 DBSCAN算法模型

*#展示每一个样本距离相差第k的距离*def k\_nearest\_neighbour\_distance(data\_set, k):  
 *# 获取距离矩阵* distance\_mat = get\_distance\_mat(data\_set)  
  
 *# 对各个点，获得其第K个最近邻的距离值(可使用torch的topk方法)* value, \_ = torch.topk(torch.from\_numpy(distance\_mat), k+1 , largest=False)  
 distance = value[:, k]  
 distance = distance.numpy()  
  
 *# 排序* distance = np.sort(distance)  
  
 return distance

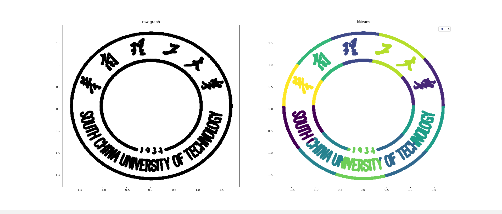
*#图形化展示每一个样本距离相差第k的距离*def show\_K\_dis(data, K):  
 distance = DBSCAN.k\_nearest\_neighbour\_distance(dataSet, K)*#k个近邻点* x = [i for i in range(distance.shape[0])]  
 plt.scatter(x, distance)  
 plt.title(**"Points Sorted by Distance to %ith Nearest Neighbor"** % K)  
 plt.legend()  
 plt.show()

*# DBSCAN算法模型*

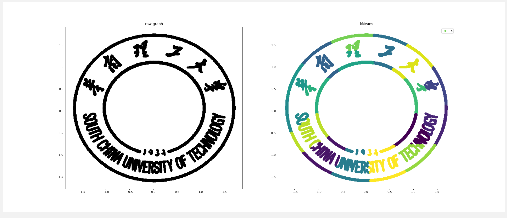
def dbscan(data\_set, eps, min\_pts):  
 *"""  
 DBSCAN聚类算法,返回每个样本的cluster类别  
 """  
 #获得距离矩阵* distance\_mat=get\_distance\_mat(data\_set)  
 *#样本数量* example\_nums=np.shape(data\_set)[0]  
 *#获得核心点坐标索引  
 #print(distance\_mat)，寻找周围有min\_pts个点以上的核心点能在范围内的，即为核心点,最外层的np.where返回的就是行号，即点的编号  
 #核心点选取法则，在所有样本点当中，假如某一个点再周围eps范围内，有八个以上近邻点，则该点就是核心店* core\_points=np.where(np.sum(np.where(distance\_mat<=eps,1,0),axis=1)>=min\_pts)[0] *#axis是指对列操作  
 #类别标签，如果为-1表示没有被分类* cluster=np.full(example\_nums,-1)*#初始将所有点都记为未分类* k=0*#初始化一个类别为0* for core\_point in core\_points:  
 *#如果核心点没有被分类，就先将其作为种子点，并入种子集合* if(cluster[core\_point]==-1):  
 cluster[core\_point]=k  
 *#将该核心点周围eps范围内的所有点全都标为种子点，进行BFS算法* core\_point\_neighbour=np.where(distance\_mat[:,core\_point]<=eps)[0]  
 seeds=set(core\_point\_neighbour)*#先将核心点附近的点分类* while(len(seeds)>0): *#如果存在种子点* point\_in\_seeds=seeds.pop()*#BFS算法开始* cluster[point\_in\_seeds]=k*#先将种子点分为原先的核心点的簇中* point\_in\_seeds\_neighbour=np.where(distance\_mat[:,point\_in\_seeds]<=eps)[0]*#搜寻该种子点周围eps范围内的所有店* if(len(point\_in\_seeds\_neighbour)>=min\_pts):*#如果种子附近的点的数量有min\_pts个以上，就说明这些点都属于这个种子点的簇中，否则，则不属于这个簇* for point in point\_in\_seeds\_neighbour:  
 if(cluster[point]==-1):*#对于该范围内的点，如果没有被分类，就加入种子点序列中，最终也会分配给这个簇* seeds.add(point)  
  
 *#寻找下一个类别* k=k+1  
 return cluster

3实验结果绘图

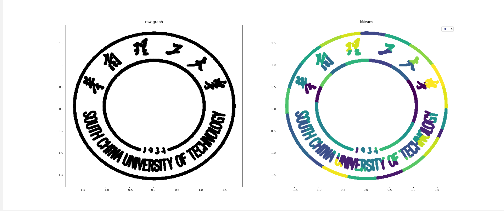
3.1 K-Means算法



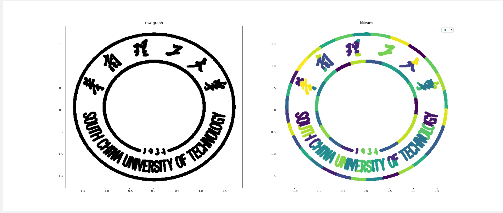
（Figure. 3.1. K=10聚类结果图像）



（Figure. 3.2. K=20聚类结果图像）

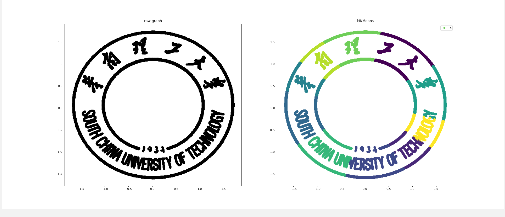


（Figure. 3.3. K=50聚类结果图像）

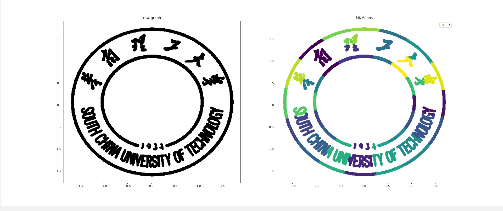


（Figure. 3.4. K=100聚类结果图像）

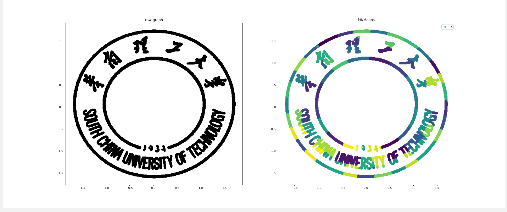
3.2 BIK-K-Means算法



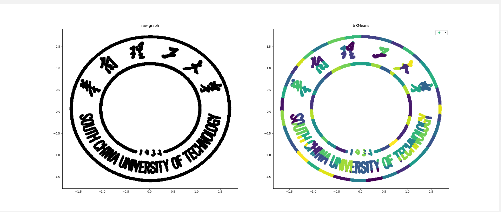
（Figure. 3.5. K=10聚类结果图像）



（Figure. 3.6. K=20聚类结果图像）

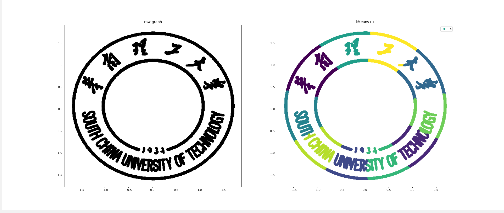


（Figure. 3.7. K=50聚类结果图像）

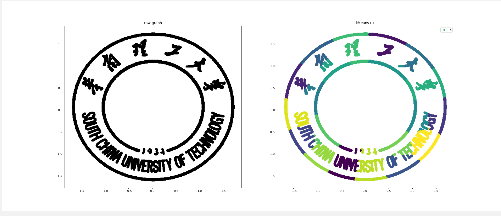


（Figure. 3.8. K=100聚类结果图像）

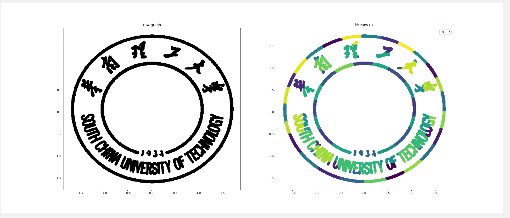
3.3 K-Means++算法



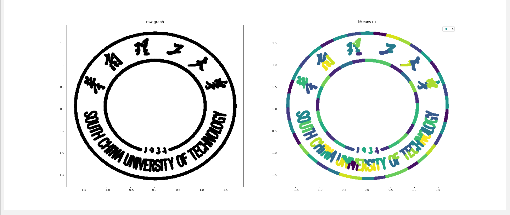
（Figure. 3.9. K=10聚类结果图像）



（Figure. 3.10. K=20聚类结果图像）

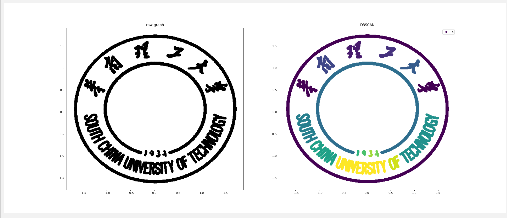


（Figure. 3.11. K=50聚类结果图像）

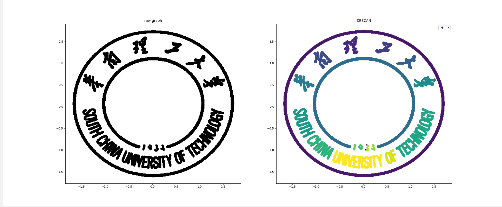


（Figure. 3.12. K=100聚类结果图像）

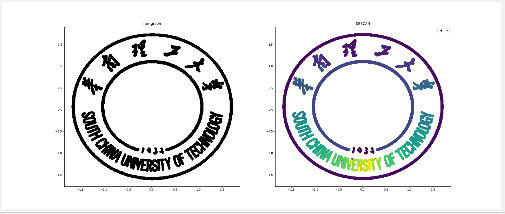
3.4 DBSCAN算法



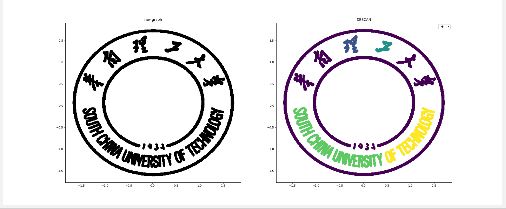
（Figure. 3.13. eps=0.08聚类结果图像，cluster num=15）



（Figure. 3.14. eps=0. 05聚类结果图像，cluster num=17）



（Figure. 3.15. eps=0.03聚类结果图像，cluster num=44）



（Figure. 3.16. eps=0.1聚类结果图像，cluster num=5）

4实验问题回答

4.1 K-Means与其变体的对比

一种广泛使用的聚类算法是K-均值算法，其中K是用户指定的要创建的簇的数目。K-均值聚类算法以K个随机质心开始。算法会计算每个点到质心的距离。每个点会被分配到距其最近的簇质心，然后紧接着基于新分配到簇的点更新簇质心。以上过程重复数次，直到簇质心不再改变。这个简单的算法非常有效但是也容易受到初始簇质心的影响。为了获得更好的聚类效果。

而二分K-均值算法首先将所有点作为一个簇，然后使用K-均值算法(K= 2)对其划分。下一次迭代时，选择有最大误差的簇进行划分。该过程重复直到K个簇创建成功为止。二分K-均值的聚类效果要好于K-均值算法。

k-means++算法选择初始seeds的基本思想就是：初始的聚类中心之间的相互距离要尽可能的远。从输入的数据点集合中随机选择一个点作为第一个聚类中心，对于数据集中的每一个点x，计算它与最近聚类中心(指已选择的聚类中心)的距离D(x)，选择一个新的数据点作为新的聚类中心，选择的原则是：D(x)较大的点，被选取作为聚类中心的概率较大重复2和3直到k个聚类中心被选出来。而k-means算法的质心只是随机选出的，其存在较大的质心敏感，如果初始质心选得不好，会导致聚类效果极为不佳。

4.2 DBSCAN相较于K-Means及其变体方法的区别

1. k-means需要指定聚类簇数k，并且且初始聚类中心对聚类影响很大。k-means把任何点都归到了某一个类，对异常点比较敏感。DBSCAN能剔除噪声，需要指定邻域距离阈值eps和样本个数阈值MinPts，可以自动确定簇个数。
2. K均值和DBSCAN都是将每个对象指派到单个簇的划分聚类算法，但是K均值一般聚类所有对象，而DBSCAN丢弃被它识别为噪声的对象。
3. K均值很难处理非球形的簇和不同大小的簇。DBSCAN可以处理不同大小或形状的簇，并且不太受噪声和离群点的影响。当簇具有很不相同的密度时，两种算法的性能都很差。
4. K均值只能用于具有明确定义的质心（比如均值或中位数）的数据。DBSCAN要求密度定义（基于传统的欧几里得密度概念）对于数据是有意义的。
5. K均值算法的时间复杂度是O(m)，而DBSCAN的时间复杂度是O(m^2)。
6. DBSCAN多次运行产生相同的结果，而K均值通常使用随机初始化质心，不会产生相同的结果。
7. K均值DBSCAN和都寻找使用所有属性的簇，即它们都不寻找可能只涉及某个属性子集的簇。
8. K均值可以发现不是明显分离的簇，即便簇有重叠也可以发现，但是DBSCAN会合并有重叠的簇。
9. K均值可以用于稀疏的高维数据，如文档数据。DBSCAN通常在这类数据上的性能很差，因为对于高维数据，传统的欧几里得密度定义不能很好处理它们。

## **结论**

在本次实验中，主要是实现了K-Means算法的两种变体以及DBSCAN算法的实现，他们都是基本的聚类算法。

在K-Means算法中，其主要需要确定簇的数量K，然后通过训练不断更新质心以及簇分配，最终达到最优的效果。但单靠K-Means算法是不足的，因为其是对质心敏感的，如果初始质心选得不好十分影响性能，因此要对其进行优化，此时便引进二分K-Means与K-Means++算法，由实验结果可知，二分K-Means与K-Means++的聚类效果均比普通K-Means好，虽然二分K-Means与K-Means++的算法实现不同，但本质的思想都是为了降低K-Means算法的质心敏感。

此外还有DBSCAN算法，它是一种典型的基于密度的聚类方法。它与K-Means的实现方式不同，它需要定义聚类范围eps以及密度阈值minPts，在eps范围内的样本数量要高于minPts才允许聚集成簇，由实验结果可知，当eps越大，聚类得到的簇的数量越多。

对于本次实验而言，主要有代码参考，加上聚类算法结构比较简单，其算法模型比较容易理解，但更多是了解如何去进行代码实现，这与在课本上仅学习到的公式是相差很大的。此外还学习到了许多机器学习的库，让自己对这些库中的函数有了更进一步的了解与运用。总体而言本次实验的收获是挺大的。

## **拓展实验**

1目前实验均是对二维数据进行实验，若对于（相对）高维的数据集，情况会如何变化？请下载[wine](http://archive.ics.uci.edu/ml/datasets/Wine)（13维）与[iris](http://archive.ics.uci.edu/ml/datasets/Iris)（4维）数据集，并编写自己的数据加载函数与聚类评估函数，完成聚类实验并对聚类结果PCA降维后进行可视化。

2数据加载实现

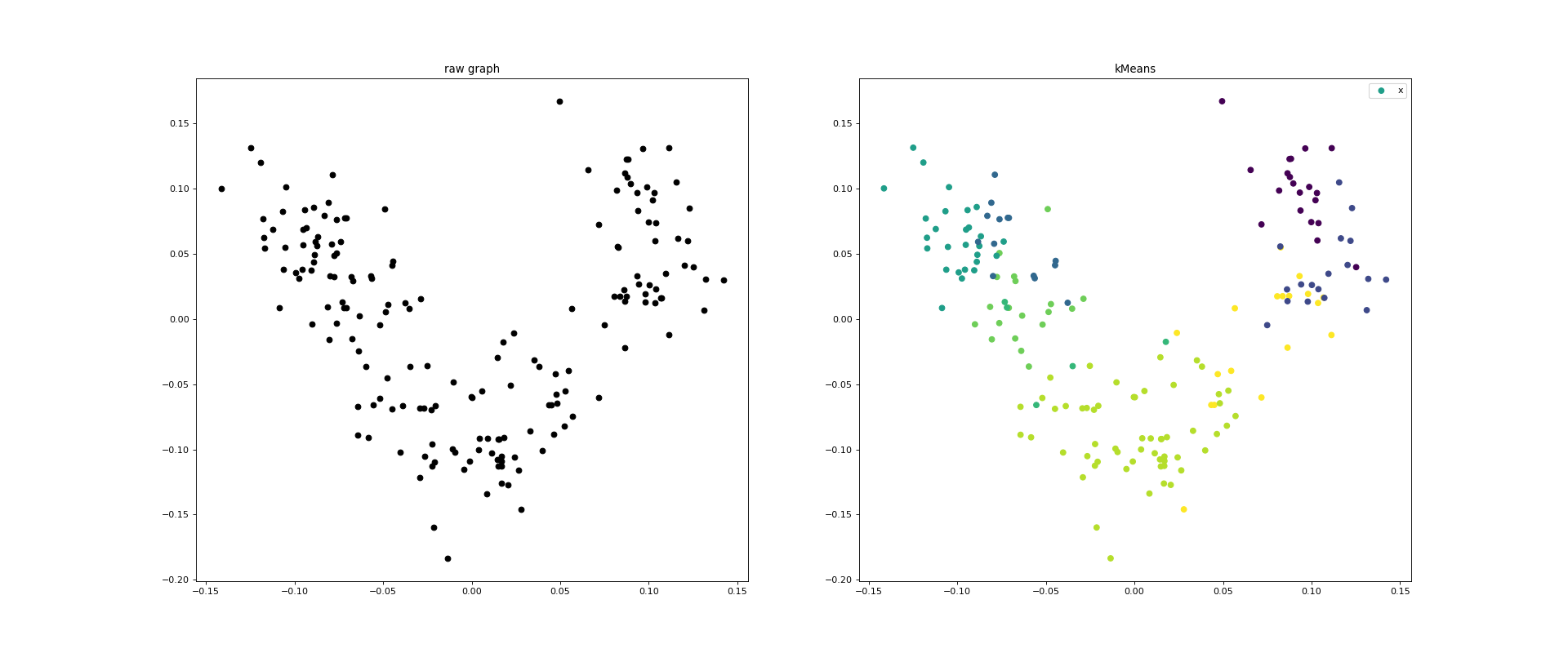
1. wineDATA = pd.read\_csv("wine.data", header=None,encoding='utf-8', sep=',')
2. wineDATA=np.array(wineDATA)
3. irisDATA=pd.read\_csv("iris.data", header=None,encoding='utf-8', sep=',')
4. irisDATA=np.array(irisDATA)
5. irisDATA=irisDATA[:,0:3]

3PCA降维实现

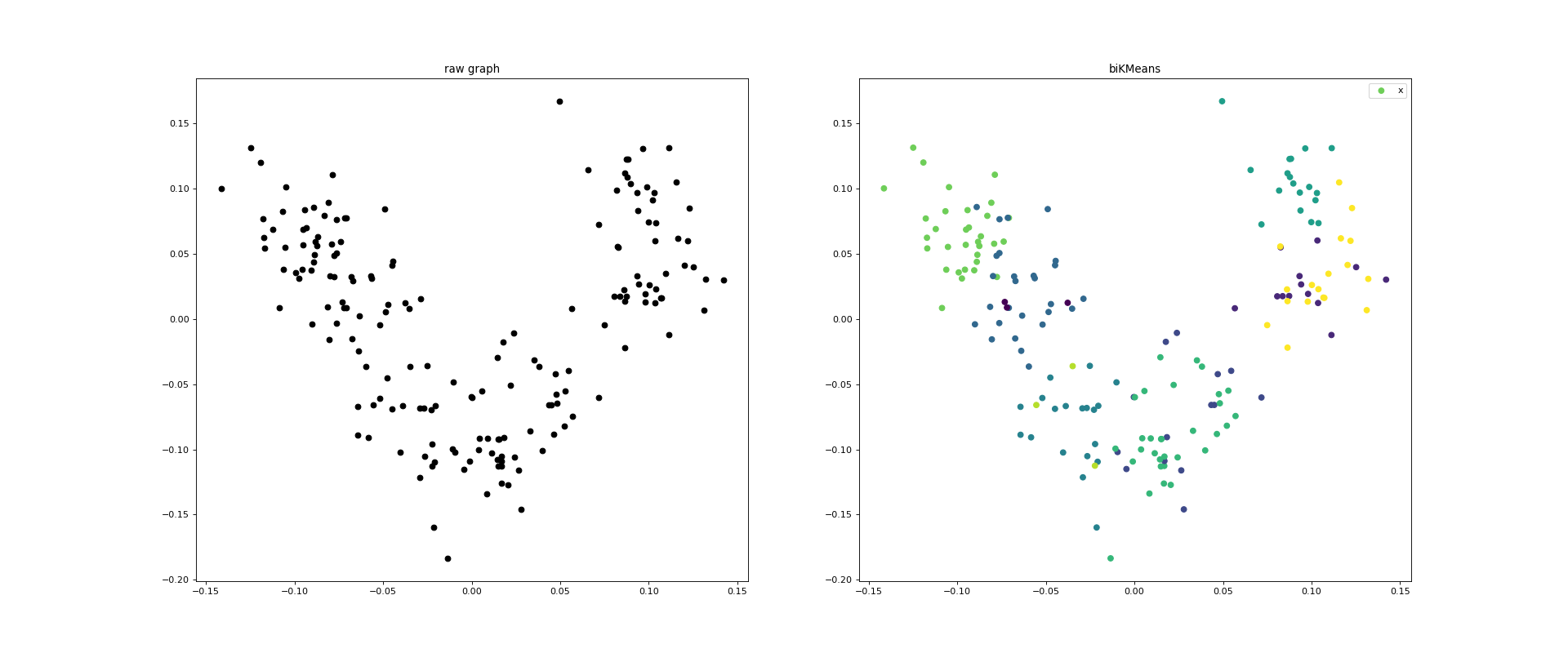
1. normal\_data\_tensor=torch.from\_numpy(normal\_data)
2. normal\_data\_pca=torch.pca\_lowrank(normal\_data\_tensor,2,True,10000)
3. normal\_data\_pca=normal\_data\_pca[0].numpy()

4实验结果

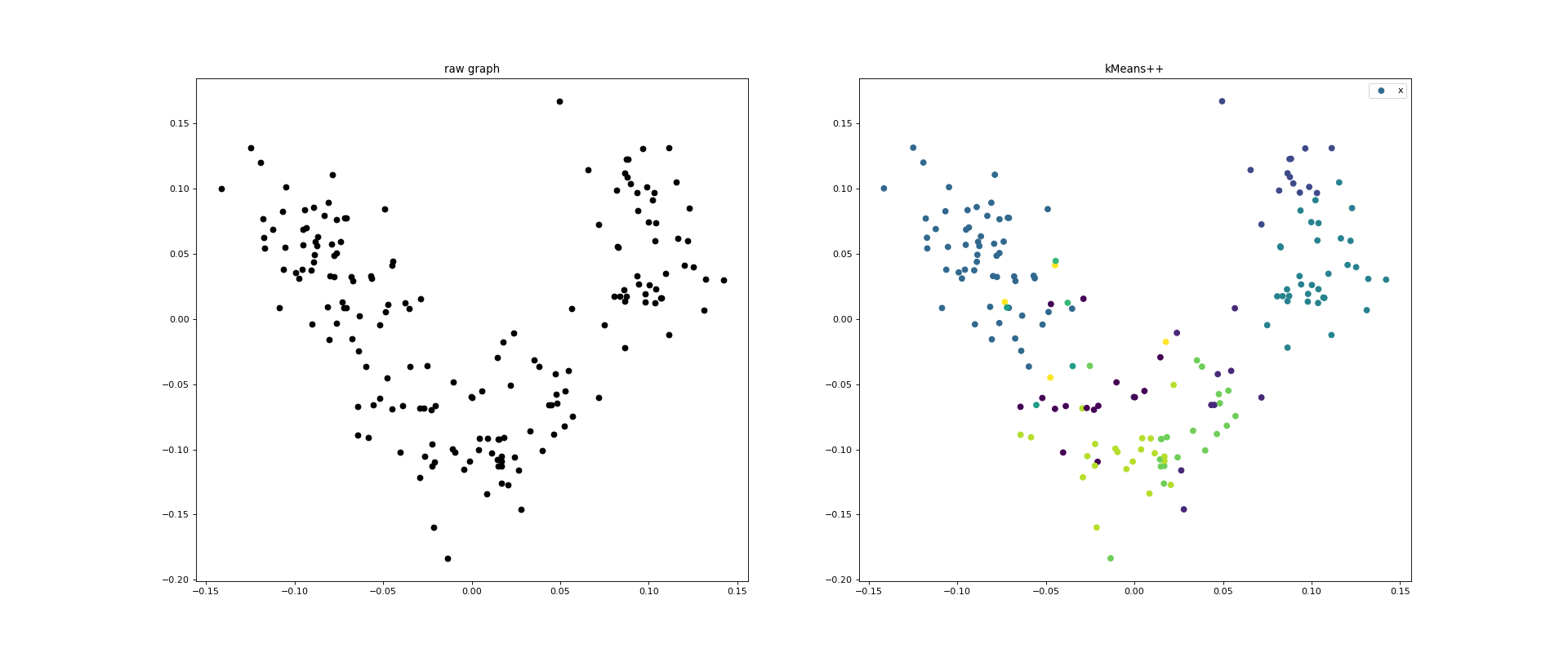
4.1 wine.data（具体实现详见main2.py）



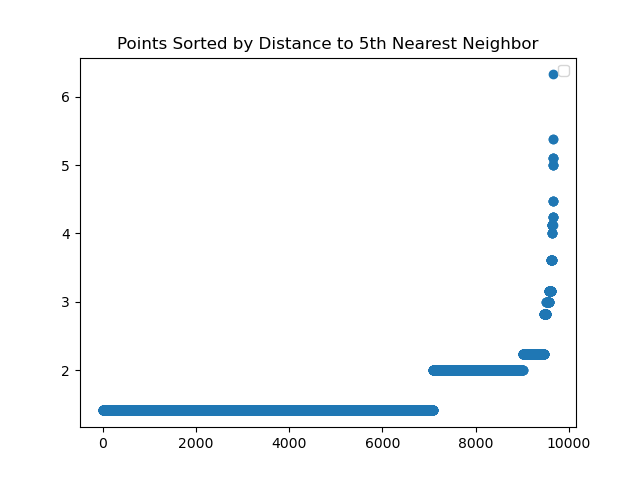
（Figure. 4.1. K=10时kMeans算法聚类结果图像）



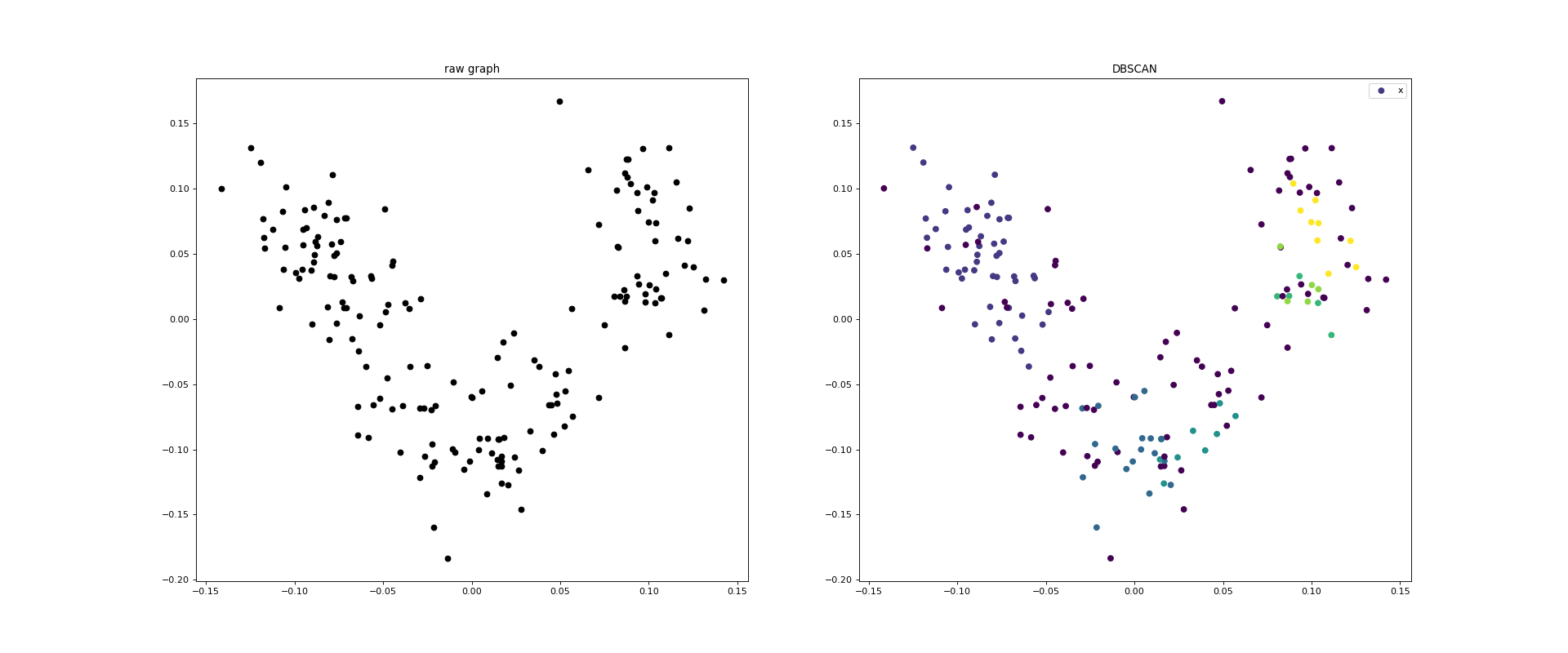
（Figure. 4.2. K=10时bikMeans算法聚类结果图像）



（Figure. 4.3. K=10时kMeans++算法聚类结果图像）

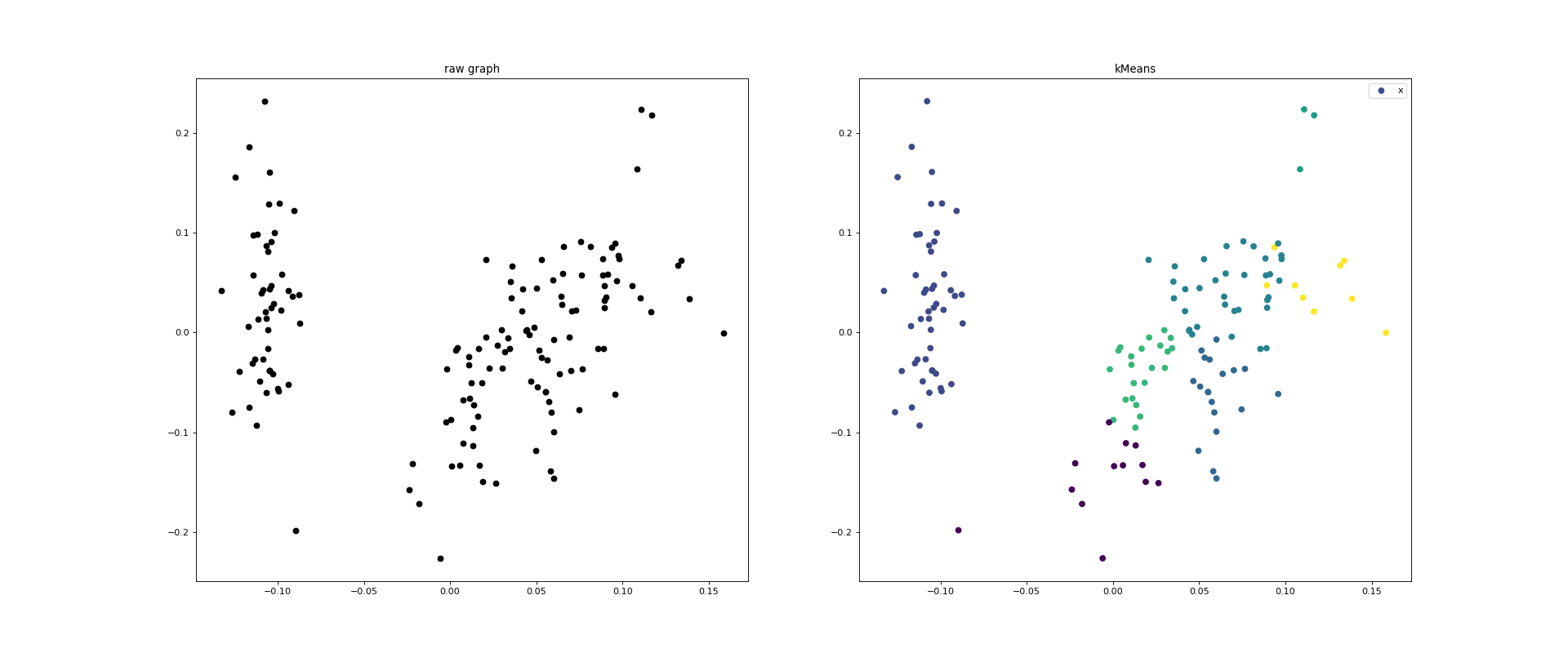


（Figure. 4.4. min\_pts=5时各点之间距离分布图）

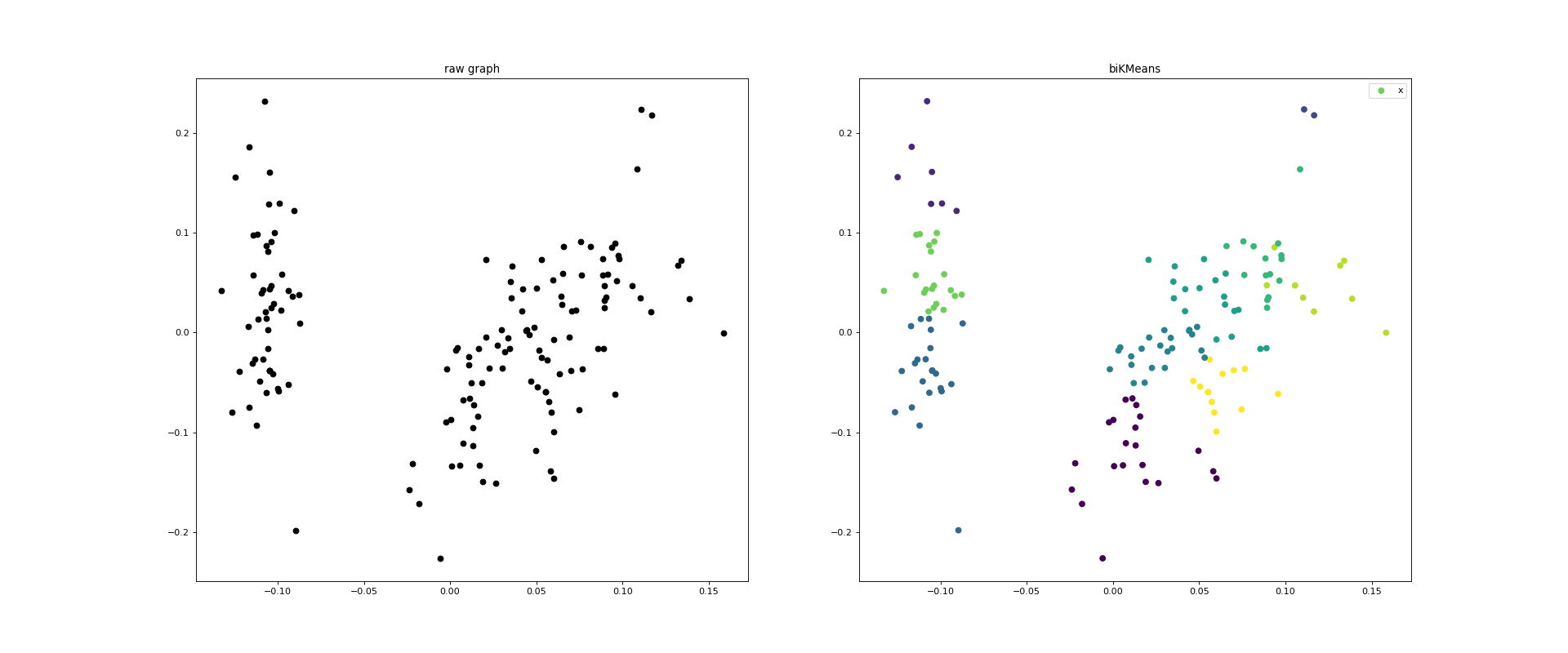


（Figure. 4.5. eps=2时DBSCAN算法聚类结果图像，K=6）

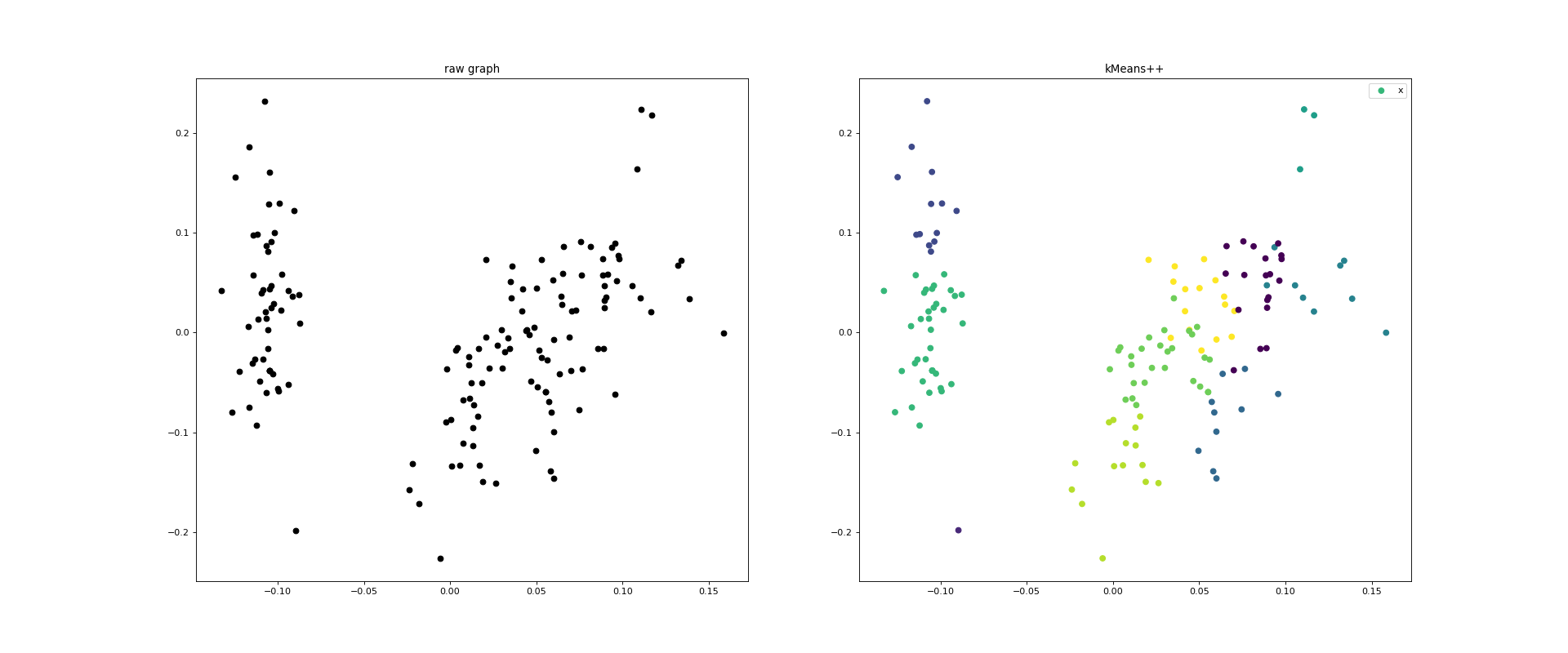
4.2 iris.data（具体实现详见main3.py）



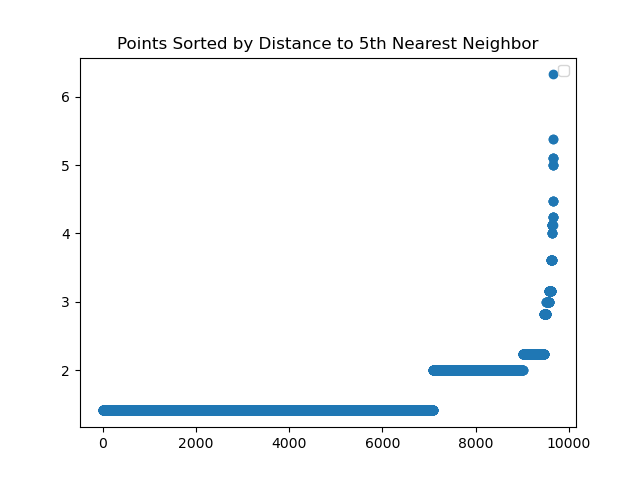
（Figure. 5.1. K=10时kMeans算法聚类结果图像）



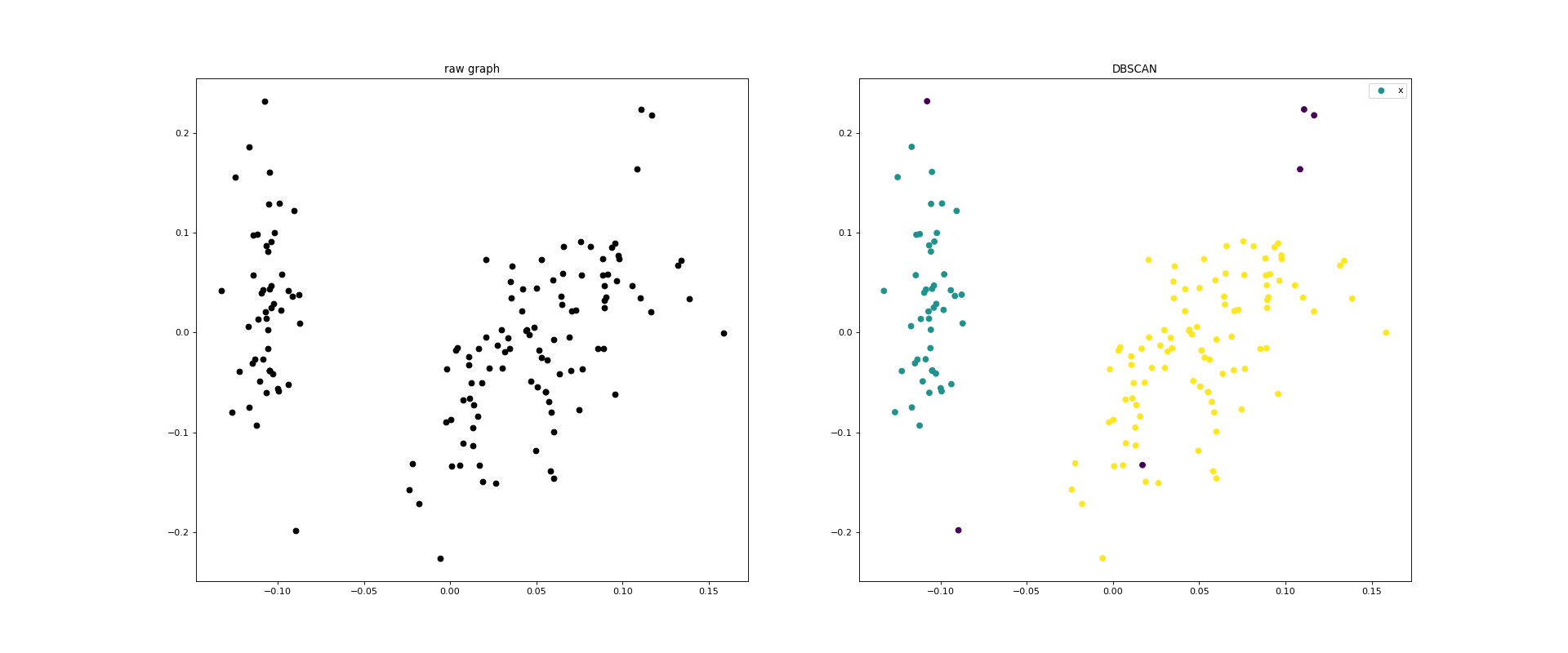
（Figure. 5.2. K=10时bikMeans算法聚类结果图像）



（Figure. 5.3. K=10时kMeans++算法聚类结果图像）



（Figure. 5.4. min\_pts=5时各点之间距离分布图）



（Figure. 5.5. eps=0.75时DBSCAN算法聚类结果图像，K=2）

5实验结论

从实验结果看可知，kMeans、bikMeans、kMeans++、DBSCAN对wine.data以及iris.data的聚类效果还是不错的，但是图中出现离群点以及噪声点，其主要原因是因为PCA算法对于高维向量的降维是有着一定的误差，但总体而言，该聚类结果还是可观的。