数据挖掘期中大作业

阮正鑫 120032910012

1、k-means聚类算法

1.1算法简介

Kmeans算法是最常用的聚类算法，主要思想是:在给定K值和K个初始类簇中心点的情况下，把每个点(亦即数据记录)分到离其最近的类簇中心点所代表的类簇中，所有点分配完毕之后，根据一个类簇内的所有点重新计算该类簇的中心点(取平均值)，然后再迭代的进行分配点和更新类簇中心点的步骤，直至类簇中心点的变化很小，或者达到指定的迭代次数。

1.2算法流程

输入：样本集$D=\{x_1,x_2,\dots,x_m\}$；聚类簇数k

过程：

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

2: repeat

3: 令$C_i=\emptyset\,(1\leq i\leq k)$

4: for $j=1,2,\dots,m$ do

5: 计算样本x_j 与各均值向量$\mu_i\ (1\leq i\leq k)$ 的距离：$d_{ji}=\|x_j-\mu_i\|_2$；

6: 根据距离最近的均值向量确定x_j的簇标记：$\lambda_j=argmin_i d_{ji}$；

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 $i=1,2,\dots,k$ do

10: 计算新均值向量：$\mu_i'=\frac{1}{|C_i|}\sum_{x\in C_i}x$；

11: if $\mu_i'\neq\mu_i$ then

12: 将当前均值向量μ_i更新为μ_i'

13: else

14: 保持当前均值不变

15: end if

16: end for

17: until 当前均值向量均未更新

输出：簇划分$C=\{C_1,C_2,\dots,C_k\}$

算法流程引用自周志华的《机器学习》P203，按照这个算法流程我们在python中实现了k-means算法，并进行了实验结果分析

1.3 算法实现

1、计算每个中心点到各个样本的距离

```
def cal_dis(data, clu, k):
    dis = []
    for i in range(len(data)):
        dis.append([])
        for j in range(k):
            dis[i].append(m.sqrt((data[i, 0] - clu[j, 0])**2 + (data[i, 1]-
clu[j, 1])**2))
    return np.asarray(dis)
```

2、根据距离最近的均值向量确定的簇标记

```
def divide(data, dis):
    clusterRes = [0] * len(data)
    for i in range(len(data)):
        seq = np.argsort(dis[i])
        clusterRes[i] = seq[0]
    return np.asarray(clusterRes)
```

3、计算新均值向量

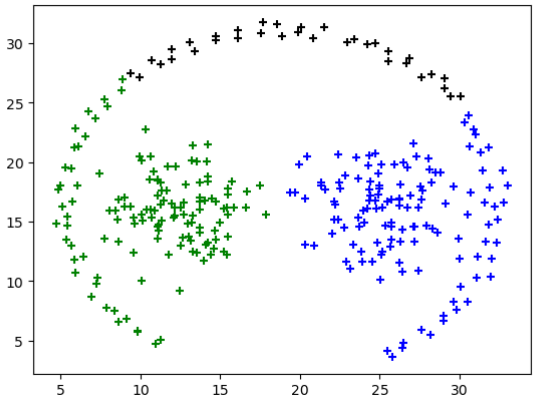
```
def center(data, clusterRes, k):
    clunew = []
    for i in range(k):
        # 计算每个组的新质心
        idx = np.where(clusterRes == i)
        sum = data[idx].sum(axis=0)
        avg_sum = sum/len(data[idx])
        clunew.append(avg_sum)
    clunew = np.asarray(clunew)
    return clunew[:, 0: 2]
```

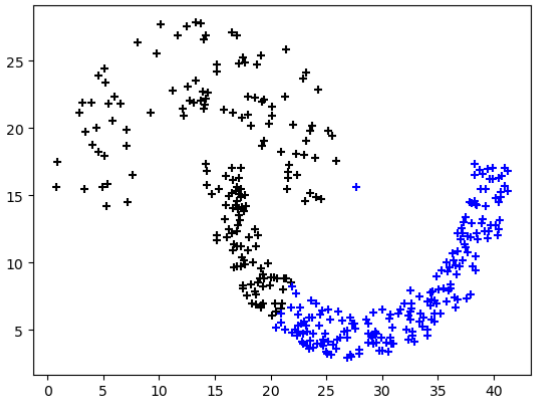
4、更新迭代，直至收敛

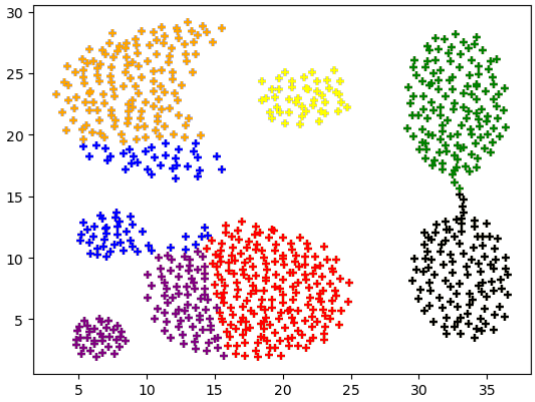
```
def classfy(data, clu, k):
    clulist = cal_dis(data, clu, k)
    clusterRes = divide(data, clulist)
    clunew = center(data, clusterRes, k)
    err = clunew - clu
    return err, clunew, k, clusterRes
```

完整代码见附件 Kmeans.py

1.4结果分析







根据K-means算法以上结果可以看出

在已知聚类簇个数的情况下，K-means算法能够在一定的数据情况下收敛，但存在明显的缺陷

1、K-means得到的簇更偏向于球形，这意味着K-means算法不能处理非球形簇的聚类问题，而现实中数据的分布情况是十分复杂的，所以K-means算法不太适用于现实大多数情况。 在前两个数据集分类中体现的十分明显，K-means算法无法实现对于环状分布的数据的聚类，使得聚类的结果出现错误。

2、K-means需要预知聚类结果的簇个数，在某些情况下是不允许的

3、K-means算法对初始选取的聚类中心点敏感。我们可以看到在最后一个数据聚类中，K-means算法会把不同簇聚合在一起来满足球形状簇的聚类，而这种情况下得到的中心点在两个簇中间。这说明此时K-means陷入了一个局部最优解，而陷入局部最优的一个原因是初始化中心点的位置不太好。

2、DBSCAN聚类算法

2.1算法简介

DBSCAN(Density-Based Spatial Clustering of Applications with Noise)是一个比较有代表性的基于密度的聚类算法。它将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。

2.2算法流程

输入：样本集$D=\{x_1,x_2,\ldots,x_m\}$,邻域参数$(\epsilon,MinPte)$;

过程：

1: 初始化核心对象集合：$\Omega=\emptyset$

2: for $j=1,2,\ldots,m$ do

3: 确定样本x_j的$\epsilon-$ 邻域$N_\epsilon(x_j)$;

4: if $|N_\epsilon(x_j)|\geq MinPts$ then

5: 将样本x_j 加入核心对象集合：$\Omega=\Omega\cup\{x_j\}$

6: end if

7: end for

8: 初始化聚类簇数：$k=0$

9: 初始化未访问样本集合： $\Gamma=D$

10: while $\Omega\neq\emptyset$ do

11: 记录当前未访问样本集合：$\Gamma_{old}=\Gamma$;

12: 随机选取一个核心对象$o\in\Omega$，初始化队列$Q=<o>$;

13: $\Gamma=\Gamma\backslash\{o\}$

14: while $Q\neq\emptyset$ do

15: 取出队列Q中的首个样本q;

16: if $|N_\epsilon(x_j)|\geq MinPts$ then

17: 令$\Delta=N_\epsilon(q)\cap\Gamma$;

18: 将Δ中的样本加入队列Q;

19: $\Gamma=\Gamma\backslash\Delta$

20: end if

21: end while

22: $k=k+1$,生成聚类簇$C_k=\Gamma_{old}\backslash\Gamma$;

23: $\Omega=\Omega\backslash C_k$

24: end while

输出：簇划分$\{C_1,C_2,\ldots,C_k\}$

算法流程引用自周志华的《机器学习》P213，按照这个算法流程我们在python中实现了dbscan算法，并进行了实验结果分析

2.3 算法实现

1、求出一个点的邻域内所有的点

```
def neighbor_points(data, pointId, radius):
    points = []
    for i in range(len(data)):
        if dist(data[i, 0: 2], data[pointId, 0: 2]) < radius:
            points.append(i)
    return np.asarray(points)
```

2、对于一个核心点，我们需要将它和它邻域内所有未分配的样本点分配给一个新类。若邻域内有其他核心点，重复上一个步骤，但只处理邻域内未分配的点，

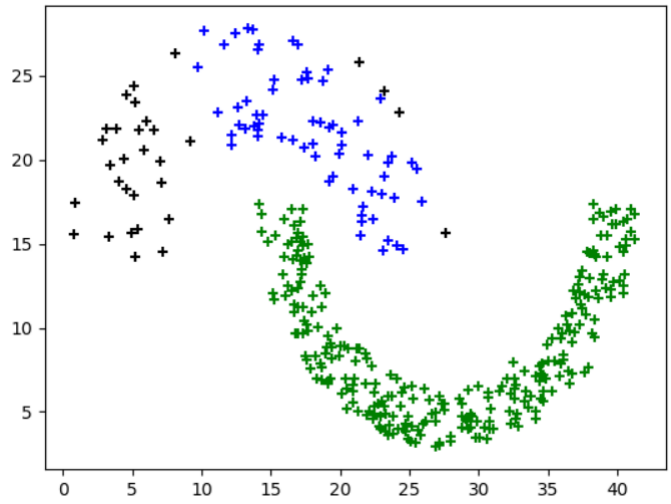
```
def to_cluster(data, clusterRes, pointId, clusterId, radius, minPts):
    points = neighbor_points(data, pointId, radius)
    points = points.tolist()

    q = queue.Queue()

    if len(points) < minPts:
        clusterRes[pointId] = NOISE
        return False
    else:
        clusterRes[pointId] = clusterId
    for point in points:
        if clusterRes[point] == UNASSIGNED:
            q.put(point)
            clusterRes[point] = clusterId

    while not q.empty():
        neighborRes = neighbor_points(data, q.get(), radius)
        if len(neighborRes) >= minPts:                    # 核心点
            for i in range(len(neighborRes)):
                resultPoint = neighborRes[i]
                if clusterRes[resultPoint] == UNASSIGNED:
                    q.put(resultPoint)
                    clusterRes[resultPoint] = clusterId
                elif clusterRes[clusterId] == NOISE:
                    clusterRes[resultPoint] = clusterId
    return True
```

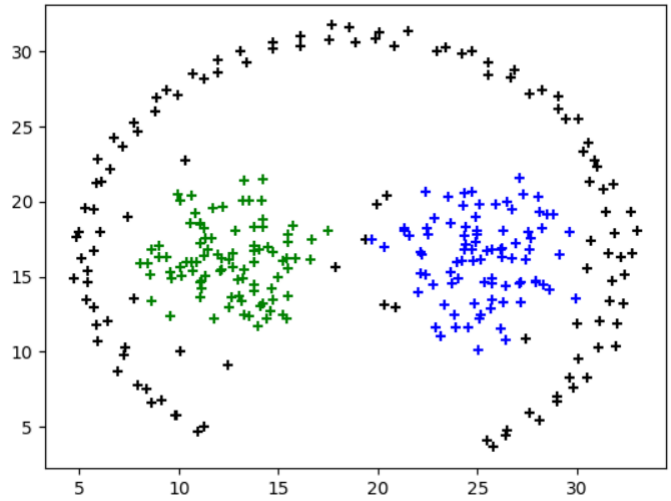
3、扫描整个数据集，为每个数据集打上核心点，边界点和噪声点标签的同时为样本集聚类

```
def dbscan(data, radius, minPts):
    clusterId = 1
    nPoints = len(data)
    clusterRes = [UNASSIGNED] * nPoints
    for pointId in range(nPoints):
        if clusterRes[pointId] == UNASSIGNED:
            if to_cluster(data, clusterRes, pointId, clusterId, radius, minPts):
                clusterId = clusterId + 1
    return np.asarray(clusterRes), clusterId
```

完整代码见附件 dbscan.py

2.4结果分析





从以上结果我们可以看出，与K-means聚类算法相比，DBSCAN可以实现非球形的聚类簇，一定程度上弥补了K-means算法的缺点。但同时存在各个簇之间密度分布不均匀，而且簇之间相距不大时，由于参数半径和局部密度阈值选取困难，导致聚类效果比较差等缺点。