

山东大学机器学习课程

实验报告

——实验一：k-means聚类法研究与实现

姓名：刘梦源

学院：计算机科学与技术学院

班级：计算机14.4

学号：201400301007

**一、实验目的：**

（1）熟悉matlab实验软件及相关函数，

（2）学习以k-means为例的常见聚类方法的思想与算法

（3）根据已给数据集，用k-means方法实现聚类，并绘制图像

**二、实验环境：**

（1）硬件环境：

英特尔® 酷睿™ i7-7500U 处理器

512 GB PCIe® NVMe™ M.2 SSD

8 GB LPDDR3-1866 SDRAM

（2）软件环境：

Windows10家庭版64位操作系统

Matlab R2016a

**三、实验内容**

**3.1聚类概述**

聚类，简单地说就是把相似的东西分到一组，同Classification(分类)不同，对于一个classifier，通常需要你告诉它“这个东西被分为某某类”这样一些例子，理想情况下，一个classifier 会从它得到的训练集中进行“学习”，从而具备对未知数据进行分类的能力，这种提供训练数据的过程通常叫做 supervised learning (监督学习)，而在聚类的时候，我们并不关心某一类是什么，我们需要实现的目标只是把相似的东西聚到一起，因此，一个聚类算法通常只需要知道如何计算相似 度就可以开始工作了，因此clustering通常并不需要使用训练数据进行学习，这在机器学习中被称作unsupervised learning (无监督学习)。

我们经常接触到的聚类分析，一般都是数值聚类，一种常见的做法是同时提取 N 种特征，将它们放在一起组成一个 N 维向量，从而得到一个从原始数据集合到 N 维向量空间的映射——你总是需要显式地或者隐式地完成这样一个过程，然后基于某种规则进行分类，在该规则下，同组分类具有最大的相似性。

**3.2 k-means聚类算法**

k-means算法是一种很常见的聚类算法，它的基本思想是：通过迭代寻找k个聚类的一种划分方案，使得用这k个聚类的均值来代表相应各类样本时所得的总体误差最小。

k-means算法的基础是最小误差平方和准则。其代价函数是：

 （1）

式中，μc(i)表示第i个聚类的均值。我们希望代价函数最小，直观的来说，各类内的样本越相似，其与该类均值间的误差平方越小，对所有类所得到的误差平方求和，即可验证分为k类时，各聚类是否是最优的。

上式的代价函数无法用解析的方法最小化，只能有迭代的方法。k-means算法是将样本聚类成 k个簇（cluster），其中k是用户给定的，其求解过程非常直观简单，具体算法描述如下：

（1）随机选取 k个聚类质心点

（2）重复下面过程直到收敛  {

      对于每一个样例 i，计算其应该属于的类：

 （2）

对于每一个类 j，重新计算该类的质心：

 （3）

}

设计程序的伪代码，实现图1所示：

**图1. k-means算法伪代码**

function KMeans(输入数据data，中心点个数K)

获取输入数据的个数m和维数n

随机生成K个n维的点

while(算法未收敛)

对m个点：计算到各个中心点的距离，找到最近的一个

对于K个中心点：

找出所有属于自己这一类的所有数据点

把自己的坐标修改为这些数据点的平均值坐标

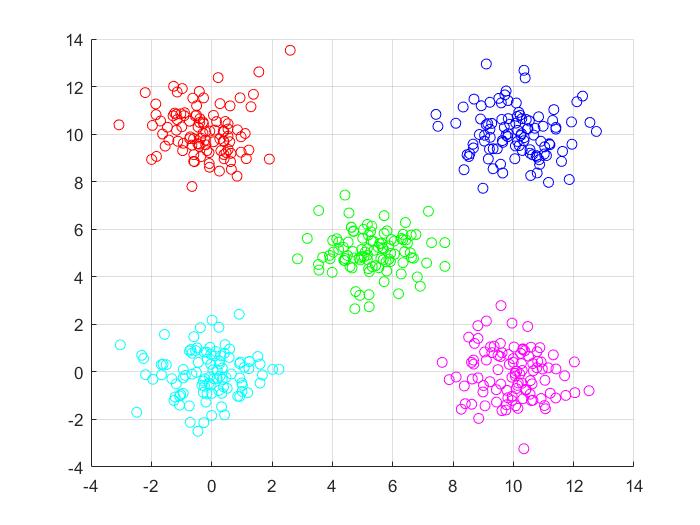
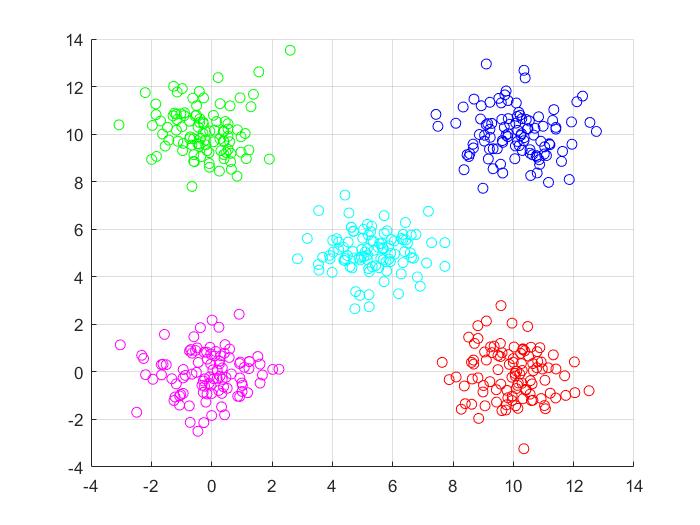
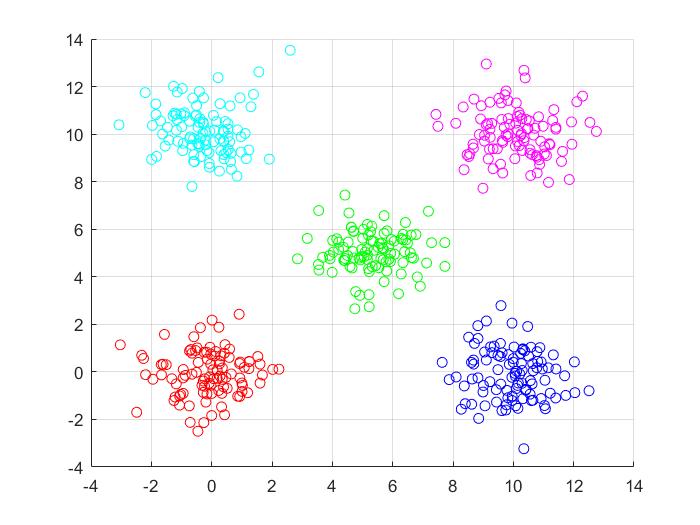
end

输出结果

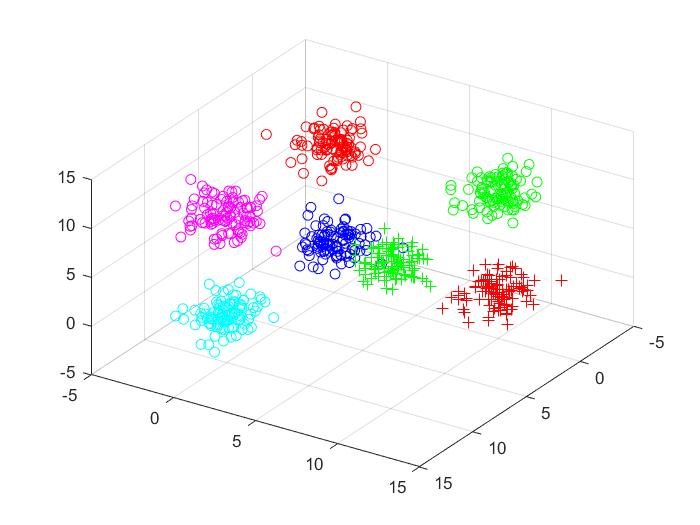
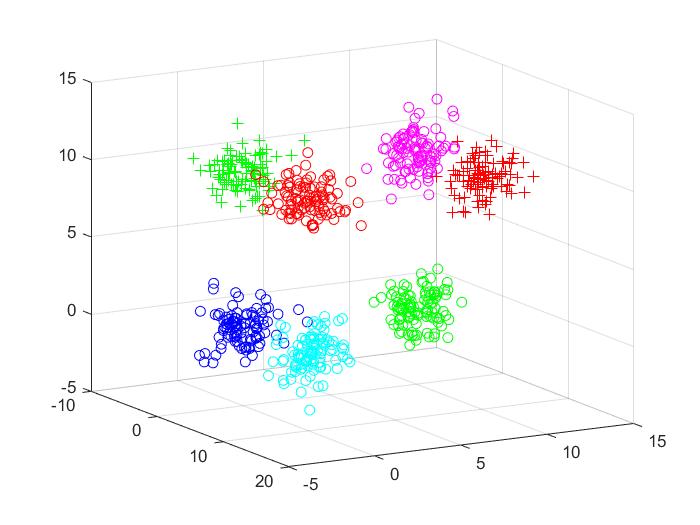
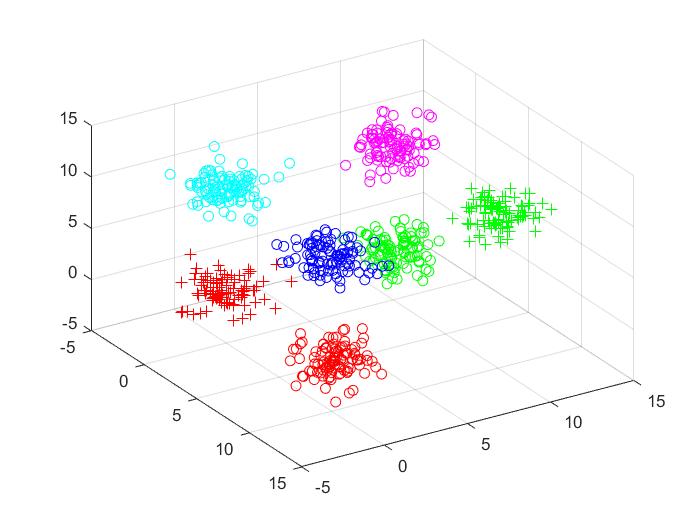
end

**四、实验结果**

用上述设计算法分别对给定k=5的二维度数据集2d-data和给定k=7的三维数据集3d-data进行多次实验，绘制散点图如图2，图3所示，聚类效果不随初始随机生成的中心点而改变，一直呈现较好的效果。至此用k-means方法完成聚类。



**图2. 2D-data的实验效果图**



**图3. 3D-data的实验效果图**

**五、k-means聚类算法的优化**

k-means算法有个比较大的缺点就是对初始k个质心点的选取比较敏感。通过查阅资料，有人提出了一个二分k均值（bisecting k-means）算法，它的出现就是为了一定情况下解决这个问题的。也就是说它对初始的k个质心的选择不太敏感。

优化后的设计算法伪代码如图4所示

function KMeans(输入数据data，中心点个数K)

对每一个簇

          计算总误差;

          在给定的簇上面进行k-均值聚类（k=2）;

          计算将该簇一分为二后的总误差;

        选择使得误差最小的那个簇进行划分操作;

end

end

**图4. 二分k-means算法伪代码**

**六、总结与归纳**

k-means算法比较简单，但也有几个比较大的缺点：

（1）k值的选择是用户指定的，不同的k得到的结果会有挺大的不同

（2）对k个初始质心的选择比较敏感，容易陷入局部最小值。

（3）[数据库](http://lib.csdn.net/base/mysql)比较大的时候，收敛会比较慢。

   k-means是较为经典的聚类算法。所以以上的这些不足也被世人的目光敏锐的捕捉到，并融入世人的智慧进行了某种程度上的改良。例如问题（1）对k的选择可以先用一些算法分析数据的分布，如重心和密度等，然后选择合适的k。而对问题（2），例如二分k均值（bisecting k-means）算法，它对初始的k个质心的选择就不太敏感，较好的优化了问题。

**七、附件**

测试观察与绘图：**testkmeans.m**

clc**;**

clear**;**

load**(**'2d-data.mat'**);**

%load('3d-data.mat');

data**=**r**;**

**[**centor**,** re\_data**]=**kmeans**(**data**,**5**);**

**[**m**,** n**]=**size**(**re\_data**);**

figure**;**

hold on**;**

**for** i**=**1**:**m

**if** re\_data**(**i**,**3**)==**1

plot3**(**re\_data**(**i**,**1**),**re\_data**(**i**,**2**),**re\_data**(**i**,**3**),**'ro'**);**

**elseif** re\_data**(**i**,**3**)==**2

plot3**(**re\_data**(**i**,**1**),**re\_data**(**i**,**2**),**re\_data**(**i**,**3**),**'go'**);**

**elseif** re\_data**(**i**,**3**)==**3

plot3**(**re\_data**(**i**,**1**),**re\_data**(**i**,**2**),**re\_data**(**i**,**3**),**'bo'**);**

**elseif** re\_data**(**i**,**3**)==**4

plot3**(**re\_data**(**i**,**1**),**re\_data**(**i**,**2**),**re\_data**(**i**,**3**),**'co'**);**

%elseif re(i,4)==5

%plot3(re(i,1),re(i,2),re(i,3),'r+');

%elseif re(i,4)==6

%plot3(re(i,1),re(i,2),re(i,3),'g+');

**else**

plot3**(**re\_data**(**i**,**1**),**re\_data**(**i**,**2**),**re\_data**(**i**,**3**),**'mo'**);**

**end**

**end**

grid on**;**

聚类算法：**kmeans.m**

**function** **[**centor**,** re\_data**]=**KMeans**(**data**,**k**)**

**[**num**,** dimension**]=**size**(**data**);**

centor**=**zeros**(**k**,**dimension**);**

random\_top**=**zeros**(**dimension**);**

random\_bottom**=**zeros**(**dimension**);**

**for** i**=**1**:**dimension

random\_top**(**i**)=**max**(**data**(:,**i**));**

random\_bottom**(**i**)=**min**(**data**(:,**i**));**

**for** j**=**1**:**k

centor**(**j**,**i**)=**random\_top**(**i**)+(**random\_bottom**(**i**)-**random\_top**(**i**))\***rand**();**

**end**

**end**

t**=**3**;**

**while** **(**t**>**0**)**

pre\_centor**=**centor**;**

**for** i**=**1**:**k

cell**{**i**}=[];**

**for** j**=**1**:**num

cell**{**i**}=[**cell**{**i**};**data**(**j**,:)-**centor**(**i**,:)];**

**end**

**end**

whole**=**zeros**(**num**,**k**);**

**for** i**=**1**:**num

abs\_cell**=[];**

**for** j**=**1**:**k

abs\_cell**=[**abs\_cell norm**(**cell**{**j**}(**i**,:))];**

**end**

**[~,** index**]=**min**(**abs\_cell**);**

whole**(**i**,**index**)=**norm**(**cell**{**index**}(**i**,:));**

**end**

**for** i**=**1**:**k

**for** j**=**1**:**dimension

centor**(**i**,**j**)=**sum**(**whole**(:,**i**).\***data**(:,**j**))/**sum**(**whole**(:,**i**));**

**end**

**end**

**if** norm**(**pre\_centor**-**centor**)<**0.1

**break;**

**end**

**end**

re\_data**=[];**

**for** i**=**1**:**num

cell**=[];**

**for** j**=**1**:**k

cell**=[**cell norm**(**data**(**i**,:)-**centor**(**j**,:))];**

**end**

**[~,** index**]=**min**(**cell**);**

re\_data**=[**re\_data**;**data**(**i**,:)** index**];**

**end**

**end**