2015 —2016 学年第 1 学期

硕士研究生 多媒体信息处理技术 课程设计

年级与专业　计算机应用技术　 学号 1120150620 姓名 蒲朝仪

**二分K均值聚类算法在Iris上的测试**

**目录**

[一、问题背景 3](#_Toc16425)

[二、解决思路 4](#_Toc14709)

[（1）K均值算法思想 4](#_Toc10142)

[（2）二分K均值算法 4](#_Toc11816)

[三、实验结果 5](#_Toc18921)

[（1）数据集 5](#_Toc16697)

[（2）实验结果 7](#_Toc12668)

[四、观察分析 9](#_Toc31820)

[参考文献 10](#_Toc11901)

[附录 11](#_Toc28603)

[附录1 实验数据汇总结果展示 11](#_Toc13697)

[附录2 二分K均值算法功能实现主要代码 13](#_Toc21722)

# 一、问题背景

目前，对于聚类问题的研究普遍存在于社会生活中的各个领域，如模式识别，图像处理、机器学习和统计学等。关于对生活中各种各样的数据的聚类分类问题己经成为众多学者的研究热题之一[1]。聚类和分类的区别在于，聚类没有任何先验知识可循，要通过数据自身的特点，将数据自动的划分到不同的类别中。聚类的基本形式定义为“在已给的数据集合中寻找数据点集的同类集合。每一个集合叫做一个类，并确定一个区域，在区域中对象的密度高于其他区域中的密度”[2]。

聚类方法有很多种，其中最简单的形式便是划分式聚类，划分式聚类试图将给定的数据集合分割成不相交的子集，使具体的聚类准则是最优的。实际中应用最广泛的准则是聚类误差平方和准则，即对于每一个点都计算它到相应的聚类中心点的平方距离，并对数据集合上的所有点的距离进行求和。一种最流行的基于最小聚类误差平法和的聚类方法是K-均值算法。K-均值算法是一种基于划分的聚类算法，它通过不断的迭代来进行聚类，当算法收敛到一个结束条件时就终止迭代过程，输出聚类结果。由于其算法思想简便，又容易实现对大规模数据的聚类，因此K-均值算法己成为一种最常用的聚类算法之一[3]。K-均值算法能找到关于聚类误差的局部的最优解，是一个能应用在许多聚类问题上的快速迭代算法。它是一种以点为基础的聚类算法，以随机选取的初始点为聚类中心，迭代地改变聚类中心来使聚类误差最小化。

K-均值算法由于其聚类过程简单，易于实现，因此已经成为当前最常用的聚类算法之一。但是K-均值的算法的聚类结果容易受到初始聚类中心点的选取的影响，不稳定，且容易受到数据中的噪声点、离群点的影响[4]。并且在K-均值方法的迭代过程中由于初值的选取就有随机性就会导致聚类容易陷入局部最优，而找不到全局最优。K-均值缺点详细介绍如下：

第一，K-均值算法中的K值必须由用户输入，在算法的流程图中我们可以看出，K-值是必须是一个用户最先确定的参数。K-均值方法必须在 K-值已知的前提下才能进行聚类。但是在一些实际问题的求解过程中，自然簇的个数K是没有事先给出的，通常是用户所不知道的。

第二，K-均值聚类算法对于噪声和离群点数据非常敏感，聚类结果很容易受到数据中所含有的噪声和离群点的影响。该算法中在簇的质心求解过程中，是通过对每个簇求均值得到的，当数据集中含有噪声和离群点数据时，在计算质心时将导致聚类中心偏离数据真正密集的区域，而得到的聚类中心将向噪声和离群点数据所在的区域偏移， 然后在此基础上进行数据点的重新分配，这必然会引起聚类结果的不准确[5,6]。

# 二、解决思路

本课程主要针对K均值的有点以及对K值的初始选择这一限制，设计一种改进的K-均值聚类方法，即二分K均值算法。通过查阅资料总结，二分K均值算法可以加速K-均值算法的执行速度，因为它的相似度计算少了[7]。另外二分K均值算法不受初始化问题的影响，因为这里不存在随机点的选取，且每一步都保证了误差最小。

## （1）K均值算法思想

K均值算法是一种经典的基于欧氏距离的硬划分算法，这种算法采用误差平方和函数作为优化的目标函数，误差平方和函数*SSE*的定义如所示[8]：

 （1）

 （2）

式中：*K*——聚类的数目；*Cj*(*j=1,2,...k*)——聚类的第*j*个簇；*X*——簇*Cj*中的任意一组数据对象；*Cj*——含有*mj*个数据对象的*Cj*质心。

可以看出SSE表示数据样本点和簇间中心间差异度平方之和，簇的中心会影响到*SSE*的值。很显然，如果*SSE*的值越小，那么就代表这种划分方法聚类的质量就越好。因此，K均值聚类的目标就是要设法找出能够使聚类准则函数*SSE*的值达到最小的聚类结果。

## （2）二分K均值算法

二分k均值（bisecting k-means）算法的主要思想是：首先将所有点作为一个簇，然后将该簇一分为二。之后选择能最大程度降低聚类代价函数（也就是误差平方和）的簇划分为两个簇。以此进行下去，直到簇的数目等于用户给定的数目k为止。二分K均值算法思想具体如下：

设*X=*{*x1，x2，...xi，...，xn*}为*n*个*Rd*空间的数据，在开始聚类前，指定*K*为聚类个数。为了得到K个簇，将所有点的集合分裂成两个类，放到簇表*S*中。从簇表中选取一个簇*Ci*，用基本的K均值聚类算法对选定的簇*Ci*进行二分聚类。从二分实验中选择具有最小总*SSE*的两个簇，将这两个簇添加到簇表*S*中，更新簇表。如此下去，知道产生*K*个簇。在二分K均值算法中，使用结果簇的质心作为基本K均值的初始质心。使用误差平方和*SSE*作为度量聚类质量的目标函数，也称*SSE*为散度。对于多次运行K均值产生的初级，选择误差平方和最小的那个，使得聚类的质心可以更好地代表簇中的点。其中*SSE*的定义如公式（3）。其中*ci*为簇*Ci*的聚类中心，*x*为该簇中的一个样本[9,10]。

 （3）

以上隐含着一个原则是：因为聚类的误差平方和能够衡量聚类性能，该值越小表示数据点月接近于它们的质心，聚类效果就越好。所以我们就需要对误差平方和最大的簇进行再一次的划分，因为误差平方和越大，表示该簇聚类越不好，越有可能是多个簇被当成一个簇了，所以我们首先需要对这个簇进行划分。

二分K均值算法受初始化问题的影响较小，因为它执行了多次二分试验并选择最小*SEE*的实验结果，且每步只有两个质心。但仍然受用户指定的聚类个数*K*的影响。

# 三、实验结果

## （1）数据集

第一个测试数据是人工设置的二维数据，这些数据值主要分布在-6到6之间，均为浮点型数据。本课程的模拟测试集数据共80个样本，有4个类。该数据集分布如图1所示：

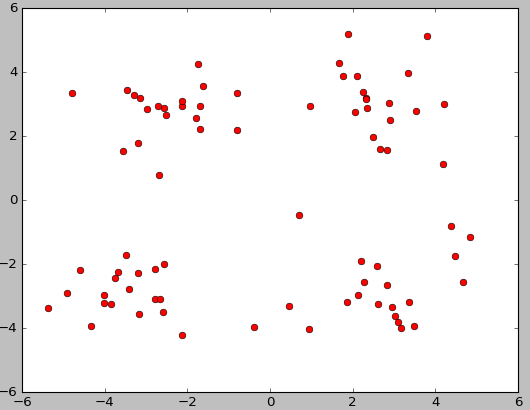


图1 随机二维数据分布

Iris数据集是常用的分类和聚类的实验数据集，也称鸢尾花卉数据，由Fisher, 1936收集整理，于1988年7月公开。Iris是一类多重变量分析的数据集，花萼长度，花萼宽度，花瓣长度，花瓣宽度4个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。通过iris以鸢尾花的特征作为数据来源，常用在分类操作中。该数据集由3种不同类型的鸢尾花的50个样本数据构成，即总共有150条鸢尾花数据。其中的一个种类与另外两个种类是线性可分离的，后两个种类是非线性可分离的。该数据集包含了5个属性，如下表1所示：

表1 Iris数据集

|  |  |  |
| --- | --- | --- |
| 属性 | 数据格式 | 备注 |
| Sepal.Length（花萼长度） | 浮点型（单位cm） | 种类有三种类型：Iris Setosa（山鸢尾）、Iris Versicolour（杂色鸢尾），以及Iris Virginica（维吉尼亚鸢尾） |
| Sepal.Width（花萼宽度） | 浮点型（单位cm） |
| Petal.Length（花瓣长度） | 浮点型（单位cm） |
| Petal.Width（花瓣宽度） | 浮点型（单位cm） |
| 种类 | 字符型 |

为方便进行数据聚类测试，对次数据做了一定的调整：将Iris数据集最后一个分类标签，及种类的这一列数据去除形成新的数据集作为二分K均值进行聚类的数据集；将Iris数据集种类中的数据改为整形数据作为评估二分K均值准确率的样本数据，其中Iris Setosa的值改为1，Iris Versicolour为2，Iris Virginica为3。该数据的分布如图2所示：

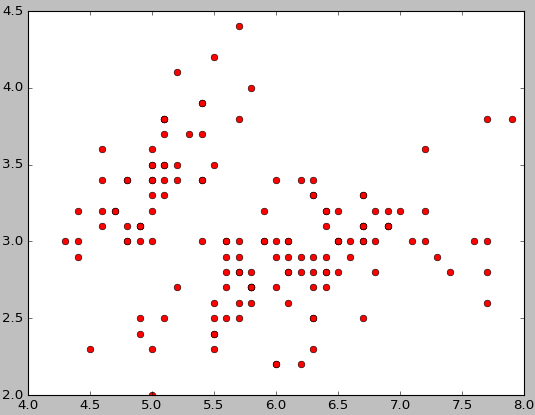


图2 鸢尾花数据分布

## （2）实验结果

本课程是在python环境下完成二分K均值聚类算法的实现的。上文中提到，为了验证二分K均值算法在聚类上的有效性，本课程设计了两组实验，包括模拟数据集上的测试和真实数据集上的测试。另外，预测准确率是由得到的聚类结果和真实数据集分类标签进行比较，按预测正确的结点个数占比得来

* **模拟数据集上的测试结果**

1. 聚类效果不太理想的情况

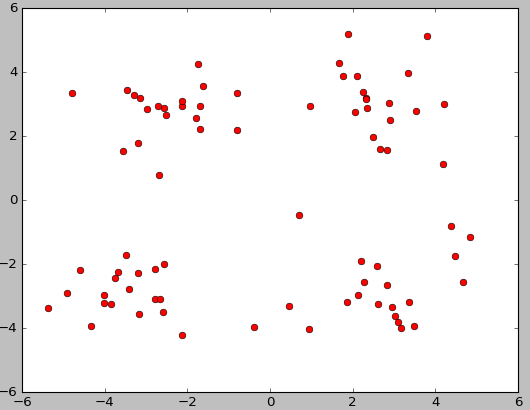
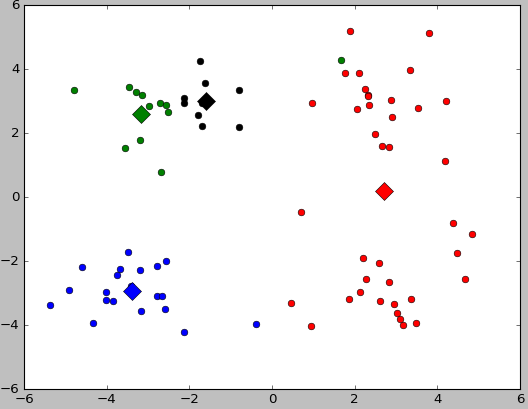
 

图3 效果较差的聚类结果

2. 聚类效果比较理想的情况

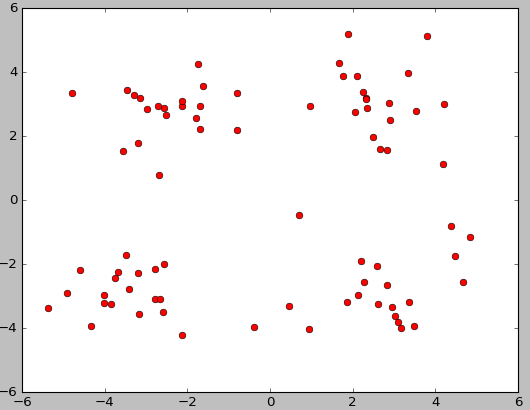
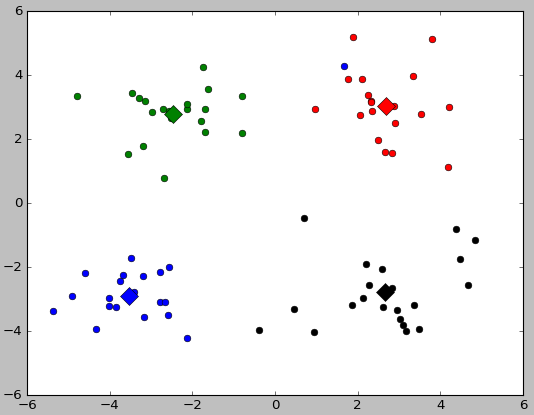
 

图4 效果较好的聚类结果

* **鸢尾花数据集上的测试结果**

1. 聚类效果不太理想的情况

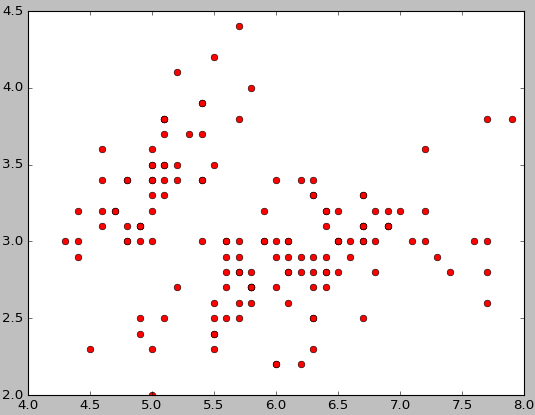
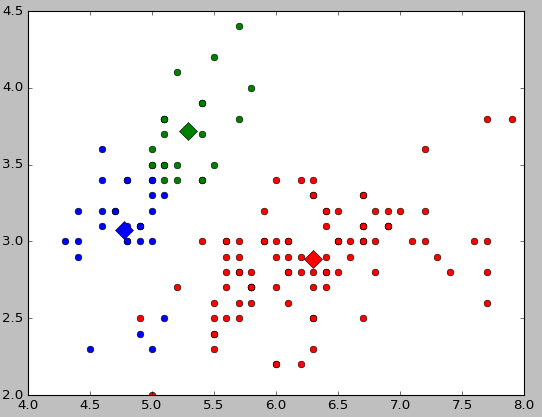
 

图5 聚类效果欠佳的情况

此时准确率如图6所示为66.7%：

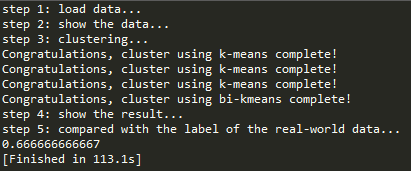


图6 聚类准确率展示

1. 聚类效果比较理想的情况

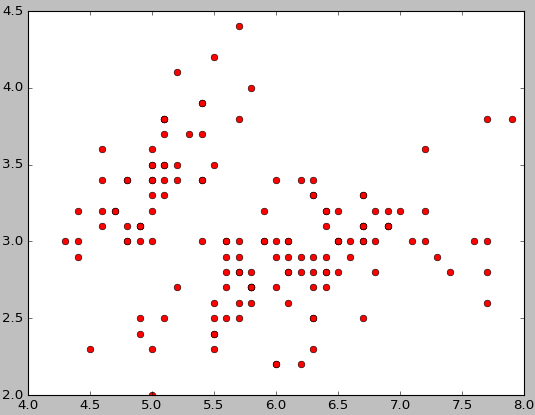
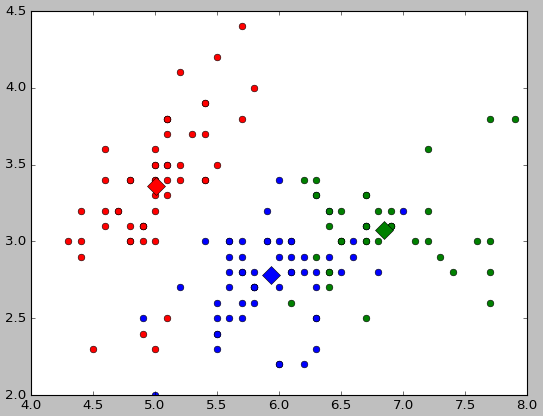
 

图7 聚类效果较好的情况

此时准确率如图8所示为86.7%：

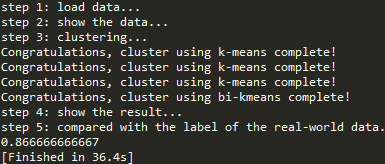


图8 聚类准确率展示

# 四、观察分析

由于在二分K均值算法中，使用误差平方和来度量聚类的质量的好坏，对各个样本点的误差采用欧式距离进行计算，然后计算误差的平方：

1.初始化全部点的质心，并建立所需要的数据存储结构

2.对每一个簇尝试二分（最开始就是一个簇），选出最好的

3.更新各个簇的元素个数

二分K均值算法没有初始化问题，其每一步操作实际上就是从*m*对子簇中找到误差平方和最小的一对子簇，然后再进行基本的K均值操作。从对模拟数据和Iris数据集的聚类实验可以看出二分K均值的聚类效果：

首先，二分K均值克服了K均值受*K*个初始点选择影响的缺点，不需要人工选定初始点。

其次，K均值算法是二分K均值建模的主要思想，它们的聚类原理是一致的，本课程在附录中随附了K均值和二分K均值在模拟数据集上的测试。二分K均值算法能够克服K均值收敛于局部最小的局限，在聚类效果上展示出比较稳定的性能，图4和图7为二分K均值算法在模拟数据集和Iris数据集上聚类效果比较好的情况，另外，在Iris数据集上能展示出86.7%的预测准确率。

另外，在图3和图5显示，实验运行代码结果会出现不太好的情况。这是由于虽然二分K均值能克服K均值收敛于局部最小的局限，但并不能保证收敛到全局最优值。

# 参考文献

[1]张娇. 基于二分K均值和SVM决策树的数据挖掘算法研究[D].陕西师范大学,2012.

[2]汪万紫,裘国永,张兵权. 基于线性判别分析和二分K均值的高维数据自适应聚类方法[J]. 郑州轻工业学院学报(自然科学版),2011,02:106-110.

[3]张军伟,王念滨,黄少滨,蔄世明. 二分K均值聚类算法优化及并行化研究[J]. 计算机工程,2011,17:23-25.

[4]蒋大宇. 快速有效的并行二分K均值算法[D].哈尔滨工程大学,2013.

[5]刘广聪,黄婷婷,陈海南. 改进的二分K均值聚类算法[J]. 计算机应用与软件,2015,02:261-263+277.

[6]王桐远. 基于二分K均值聚类的二部图网络推荐算法[J]. 经营管理者,2015,25:3.

[7]李梅. 改进的K均值算法在中文文本聚类中的研究[D].安徽大学,2010.

[8]张娇,裘国永,张奇. 基于二分K均值的SVM决策树的高维数据分类方法[J]. 赤峰学院学报(自然科学版),2012,07:13-15.

[9]裘国永,张娇. 基于二分K-均值的SVM决策树自适应分类方法[J]. 计算机应用研究,2012,10:3685-3687+3709.

[10]邹海,李梅. 一种用于文本聚类的改进二分K-均值算法[J]. 微型机与应用,2010,12:64-67.

# 附录

## 附录1 实验数据汇总结果展示

**1. K 模拟测试数据**

|  |  |  |
| --- | --- | --- |
| K 模拟测试数据 | | |
| 原图 | 聚类图 | 备注 |
| IMG_256 | IMG_256 | 由于二分K均值是由K均值算法改进而来，其聚类基本原理相同，本课程在做二分K均值聚类实验前测试了一遍K均值聚类算法在数据上的不同聚类效果 |
| IMG_256 | IMG_256 |

**2. 二分K均值模拟数据**

|  |  |  |
| --- | --- | --- |
| 二分K 模拟测试数据 | | |
| 原图 | 聚类图 | 备注 |
| IMG_256 | IMG_256 | 二分K均值算法没有初始化问题，其每一步操作实际上就是从m对子簇中找到误差平方和最小的一对子簇，然后再进行基本的K均值操作。 |
| IMG_256 | IMG_256 |
| IMG_256 | IMG_256 |

**3. 二分K均值鸢尾花测试**

|  |  |  |
| --- | --- | --- |
| 二分K 模拟测试数据 | | |
| 聚类图 | 预测准确率 | 备注 |
| IMG_256 | | 1.虽然二分K均值能克服K均值收敛于局部最小的局限，但并不能保证收敛到全局最优值  2.预测准确率是由得到的聚类结果和真实数据集分类标签进行比较，按预测正确的结点个数占比得来 |
| IMG_256 | IMG_256 |
| IMG_256 | IMG_256 |

## 附录2 二分K均值算法功能实现主要代码

**1.欧式距离函数**

def euclDistance(vector1, vector2):

return sqrt(sum(power(vector2 - vector1, 2)))

**2.K均值聚类函数**

def kmeans(dataSet, k):

numSamples = dataSet.shape[0]

# first column stores which cluster this sample belongs to,

# second column stores the error between this sample and its centroid

clusterAssment = mat(zeros((numSamples, 2)))

clusterChanged = True

# step 1: init centroids

centroids = initCentroids(dataSet, k)

while clusterChanged:

clusterChanged = False

# for each sample

for i in xrange(numSamples):

minDist = 100000.0

minIndex = 0

# for each centroid

# step 2: find the centroid who is closest

for j in range(k):

distance = euclDistance(centroids[j, :], dataSet[i, :])

if distance < minDist:

minDist = distance

minIndex = j

# step 3: update its cluster

if clusterAssment[i, 0] != minIndex:

clusterChanged = True

clusterAssment[i, :] = minIndex, minDist\*\*2

# step 4: update centroids

for j in range(k):

pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A == j)[0]]

centroids[j, :] = mean(pointsInCluster, axis=0)

print 'Congratulations, cluster using k-means complete!'

return centroids, clusterAssment

**3.二分K均值函数**

def biKmeans(dataSet, k):

numSamples = dataSet.shape[0]

# first column stores which cluster this sample belongs to,

# second column stores the error between this sample and its centroid

clusterAssment = mat(zeros((numSamples, 2)))

# step 1: the init cluster is the whole data set

centroid = mean(dataSet, axis=0).tolist()[0]

centList = [centroid]

for i in xrange(numSamples):

clusterAssment[i, 1] = euclDistance(mat(centroid), dataSet[i, :])\*\*2

while len(centList) < k:

# min sum of square error

minSSE = 100000.0

numCurrCluster = len(centList)

# for each cluster

for i in range(numCurrCluster):

# step 2: get samples in cluster i

pointsInCurrCluster = dataSet[

nonzero(clusterAssment[:, 0].A == i)[0], :]

# step 3: cluster it to 2 sub-clusters using k-means

centroids, splitClusterAssment = kmeans(pointsInCurrCluster, 2)

# step 4: calculate the sum of square error after split this

# cluster

splitSSE = sum(splitClusterAssment[:, 1])

notSplitSSE = sum(clusterAssment[nonzero(

clusterAssment[:, 0].A != i)[0], 1])

currSplitSSE = splitSSE + notSplitSSE

# step 5: find the best split cluster which has the min sum of

# square error

if currSplitSSE < minSSE:

minSSE = currSplitSSE

bestCentroidToSplit = i

bestNewCentroids = centroids.copy()

bestClusterAssment = splitClusterAssment.copy()

# step 6: modify the cluster index for adding new cluster

bestClusterAssment[nonzero(bestClusterAssment[:, 0].A == 1)[

0], 0] = numCurrCluster

bestClusterAssment[nonzero(bestClusterAssment[:, 0].A == 0)[

0], 0] = bestCentroidToSplit

# step 7: update and append the centroids of the new 2 sub-cluster

centList[bestCentroidToSplit] = bestNewCentroids[0, :]

centList.append(bestNewCentroids[1, :])

# step 8: update the index and error of the samples whose cluster have

# been changed

clusterAssment[nonzero(clusterAssment[:, 0].A ==

bestCentroidToSplit), :] = bestClusterAssment

print 'Congratulations, cluster using bi-kmeans complete!'

return mat(centList), clusterAssment,numSamples