

计算机与信息技术学院

研究生《机器学习》课程大作业

实验报告

**实验名称 编程实现不同的聚类算法**

**小组成员 贺荣伟**

**李金富**

**王 涛**

**日 期 2017-12-27**

1. **实验内容**

本实验的内容是实现K-Means、GMM聚类算法，并在Chap\_7\_dataSet数据集上进行测试，聚类结果并用NMI进行评价。实验的目的是熟练掌握K-Means、GMM算法，提高对机器学习聚类的理解。

K-Means算法基本原理：

1. 从中数据集中随机取 k 个点，作为 k 个簇的各自的中心。
2. 分别计算剩下的点到 k 个簇中心的相异度，将这些元素分别划归到相异度最低的簇。两个点之间的相异度大小采用欧氏距离公式衡量，对于两个点 T0(x1,y2)和 T1(x2,y2)，T0 和 T1 之间的欧氏距离为：d=sqrt((x1-x2)^2+(y1-y2)^2)欧氏距离越小，说明相异度越小。
3. 根据聚类结果，重新计算 k 个簇各自的中心，计算方法是取簇中所有点各自维度的算术平均数。

（4）将数据集中全部点按照新的中心重新聚类。

（5）重复第 4 步，直到聚类结果不再变化。

GMM算法基本原理：

1. 初始化混合高斯分布的模型参数，包括各个高斯分布组合系数、均值、协方差；
2. 计算每个样本由各个高斯分布生成的后验概率
3. 更新混合高斯分布的模型参数
4. 重复2、3两步直到满足条件

本实验小组分工情况：

贺荣伟：实验报告+PPT

李金富：聚类实现

**二、实验设计**

在本实验中，我首先处理数据集，并设计合适的数据结构读取数据；其次实现K-Means算法和GMM算法，在数据集做实验，并把实验结果可视化。再其次实现NMI评价指标程序，评价实验结果。最后进行实验结果分析。实验设计流程如下图所示：



图 1

**三、实验环境及实验数据集**

实验环境：

操作系统：Windows10

编程语言：Python3.6

IDLE： PyCharm

内存： 4G

CPU： Intel7 4核

数据：

数据集：chap\_7\_dataSet

格式：Excel表格

大小：500

样式：

数据集分类表如表1所示

表1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 样例 | |  |  | | --- | --- | | x |  | | y | 类别 |
| 样例1 | 0.840611 | 2.864587 | 1 |
| 样例2 | 4.145202 | 3.761701 | 2 |
| 样例3 | 1.244201 | 2.019797 | 3 |

**四、详细实验过程**

K-均值是发现给定数据集的K个簇的算法。簇个数K是用户给定的，每一个簇通过其质心（centroid），即簇中所有点的中心来描述。

首先，随机确定k个初始点的质心；然后将数据集中的每一个点分配到一个簇中，即为每一个点找到距其最近的质心，并将其分配给该质心所对应的簇；该步完成后，每一个簇的质心更新为该簇所有点的平均值。

程序清单附录中的代码包含几个K-均值算法中要用到的辅助函数。第一个函数loadDataSet()函数，它将文本文件导入到一个列表中。文本文件每一行为tab分隔的浮点数。

每一个列表会被添加到dataMat中，最后返回dataMat。该返回值是一个包含许多其他列表的列  表。这种格式可以很容易将很多值封装到矩阵中。

下一个函数distEclud()计算两个向量的欧式距离。这是本章最先使用的距离函数，也可  以使用其他距离函数。

最后一个函数是randCent()，该函数为给定数据集构建一个包含k个随机质心的集合。随机 质心必须要在整个数据集的边界之内，这可以通过找到数据集每一维的最小和最大值来完成。然 后生成0到1之间的随机数并通过取值范围和最小值，以便确保随机点在数据的边界之内。

所有支持函数正常运行之后，就可以准备实现完整的K-均值算法了。该算法会创建k个质心，  然后将每个点分配到最近的质心，再重新计算质心。这个过程重复数次，直到数据点的簇分配结 果不再改变为止。

kMeans()函数接受4个输入参数。只有数据集及簇的数目是必 选参数，而用来计算距离和创建初始质心的函数都是可选的。kMeans()函数一开始确定数据集中  数据点的总数，然后创建一个矩阵来存储每个点的簇分配结果簇分配结果矩阵clusterAssment  包含两列:一列记录簇索引值，第二列存储误差。这里的误差是指当前点到簇质心的距离，后边会使用该误差来评价聚类的效果。

  直到所有数据点的簇分配结果不再改变为止。程序中可以创建一个标志变量clusterChanged，如果该值为True，则继续迭代。

 上述迭代使用while循环来实现。接下来遍历所有数据找到距离每个点最近的质心，这可以通过  对每个点遍历所有质心并计算点到每个质心的距离来完成0。计算距离是使用distMeas参数给  出的距离函数，默认距离函数是distEclud()，该函数的实现已经在程序清单附录中给出。如果  任一点的簇分配结果发生改变，则更新  clusterChanged标志。  最后，遍历所有质心并更新它们的取值.  具体实现步骤如下:首先通过数组过滤来获得给  定簇的所有点;  然后计算所有点的均值，选项axis  =0表示沿矩阵的列方向进行均值计算;最后，程序返回所有的类质心与点分配结果。

K-Means算法伪代码如图2所示

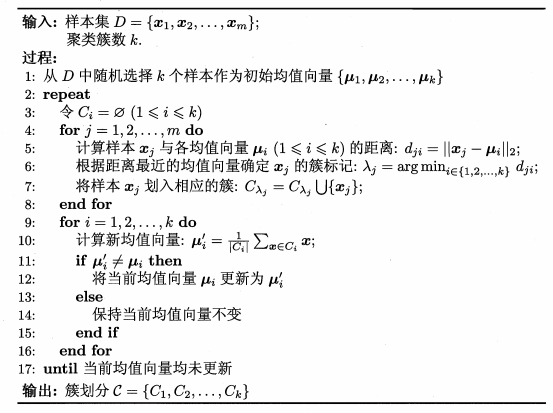


图2

高斯混合模型（Gaussian Mixture Model），是一种业界广泛使用的聚类算法，该方法使用了高斯分布作为参数模型，并使用了期望最大（Expectation Maximization，简称EM）算法进行训练。简单地讲，就是将多个高斯模型混合起来，作为一个新的模型，这样就可以综合运用多模型的表达能力。

高斯混合模型（Gaussian Mixture Model）如图3所示

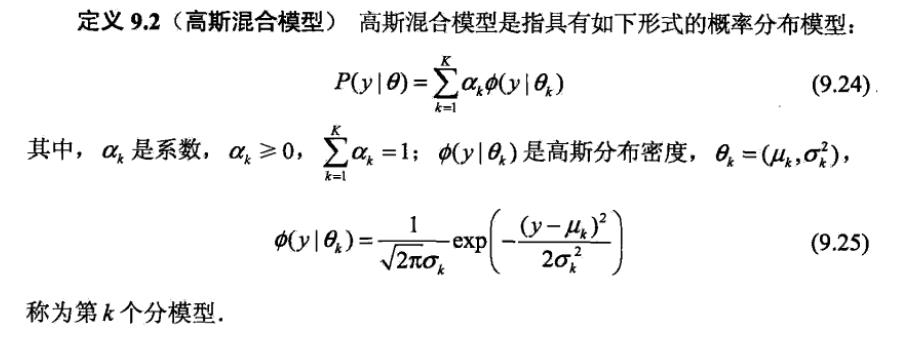


图3

1.组合模型原理

GMM 可以认为是 K-means 算法的升级版。在 K-means 中，我们会先计算出几个聚类中心，然后根据数据点与聚类中心的距离，直接将数据点归类到最近的聚类中心。这种做法被称作“硬划分”，因为有很多边缘点属于两个聚类中心的概率可能相差不大，如果直接将它归到某一个中心，实在是太粗暴了。而 GMM 不同于 K-means 的地方就在于，它除了给出聚类中心外，还能告诉你每个点归属于某个聚类中心的概率，因此，GMM 又被称作“软划分” (soft assignment)。

首先，GMM 模型的公式如下：

公式（1）

其中，。可以看出，GMM 将几个高斯模型线性组合起来，习惯上把这里面的各个高斯模型称为 Component。其中， 表示每个模型的占比，或者说数据属于模型 k 的概率，这个值越大，说明聚集在这个模型内的数据越多。

这种模型组合的方式可以更好地拟合数据。理论上，高斯模型一般成椭圆状（二维）或椭球状（三维），可以把这个椭圆或椭球认为是一种聚类的形状，而圆心或球心则是聚类中心（对应高斯函数的参数）。但真实世界中，数据的分布并不一定都是按这样的形状分布的。因此，一个高斯模型可能没法很好的拟合这些数据，而如果能综合考虑几个高斯模型的表达能力，让这些模型发挥所长，不同的模型拟合不同的数据，这样一来，所有数据就可以很好地被这个「组合模型」拟合起来。

这种组合模型的思路可以应用到很多模型上，比如：泊松模型。而由于高斯模型本身一些良好的性质，因此 GMM 这种模型被用的比较多。

2.聚类原理

GMM 本质上是一种聚类算法，介绍GMM 模型如何聚类，如何计算聚类概率。

如果已知一个 GMM 模型，现在给定一个点，我们要怎么知道这个点属于哪个聚类中心呢？更具体一点说，怎么知道这个点属于每个聚类中心的概率是多少？用数学的语言表就是，已知一个 GMM 模型：它的 K 个聚类中心为，现在要求概率值。求解的方法很简单，根据贝叶斯公式：，我们可以得出：

因此，对于每个聚类中心，由于分母都是相同的，我们只需要计算即可。得到的值就是数据点属于 的概率，至于具体要将 归类到哪个中心，可以根据具体情况决定，比如将概率最大的作为归属的聚类中心。这一点也是 GMM 优于 K-means 的地方，前者是通过概率的方式来决定归属，因此提供了更加丰富的信息。

3.参数估计

GMM 模型需要根据一堆数据点估计出模型的参数，需要确定的参数有如下三类：

高斯模型的个数，这个参数跟 K-means 的 一样，需要人工事先设定， 越大，聚类的粒度也越细；

， 每个 Component 的概率分量，或者说在总样本中的占比；

、，各个 Component 的参数。

如果样本所属分类已知（即已知 属于哪个 ），那 GMM 的参数就很容易确定了。首先，参数 人工设置。然后，假设样本容量为，归属于聚类中心的样本数量为，归属每个的样本集合为，可以用以下公式求出其他参数：

公式（2）

对于样本的分类未知的情况，要预测高斯模型的概率分量以及每个模型各自的参数和，最简单的方法是使用极大似然估计。假设有 m 个样本，首先，写出 的似然函数：

公式（3）

直接求导数的方法很难求出 和 ，因此，可以使用下节介绍的EM 算法求解。

4.均值最大化算法 EM

K-means求出聚类中心分三步进行：

随机初始化 K 个聚类中心的位置；

将所有样本点，按照跟各个聚类中心的距离进行归类；

根据样本重新归类的结果，更新聚类中心的位置，重复步骤 2 直到收敛（即聚类中心重新调整的幅度小于某个阈值）。

GMM 采用K-means 的思路来求出 GMM 的参数。GMM 的几个参数（，，）计算方式如下。假设我们已经知道了后验概率，则可以根据以下公式计算参数（其中，m 表示样本数量）：

公式（4）

这个公式把所有样本属于的概率求平均后，作为这个聚类中心（或者说这个高斯模型）的出现概率。

公式（5）

这个求均值的公式，跟单个高斯模型不同的地方在于，用的是加权平均。因为每个样本点都有一定的概率属于聚类中心 ，所以，每个样本对 对应的高斯模型的均值也会产生一定的作用，只是由于的值不同，因此这种作用也会有显著差别。

公式（6）

类似地，协方差也是用加权平均求出来的。上述三个参数的求法是从极大似然函数推出来的。

以上公式都是基于 计算出来的，而这个公式本身又需要知道（即）等参数。借助 K-means 的思路，事先随机初始化这些参数，然后计算出，再用它更新 GMM 参数，然后再用更新的模型计算，如此迭代下去，直到一个收敛阈值，就可以像 K-means 一样计算出参数。

迭代计算 GMM 参数的步骤如下：

随机初始化各个高斯模型的参数；

根据参数，计算，这一步其实是计算出每一个样本归属于每一个聚类中心的概率；

根据第 2 步计算得到的，按照公式上述公式重新计算 GMM 参数，并重复步骤 2 直到收敛。

上述计算步骤就是EM 算法的核心部分，EM 算法主要分为 E 和 M 两步：

E 指的是 Expectation，即计算均值的过程，对应的是上面的步骤 2，这一步主要是计算每个样本归属的聚类中心；

M 指的是 Maximum，即对参数的最大似然估计，对应的是上面的步骤 3。计算参数的公式是用最大似然函数推出来的，所以，这一步其实是在根据步骤 2 的分类结果，重新用最大似然函数来估计参数。

EM 算法求解 GMM 参数的完整过程如图4所示

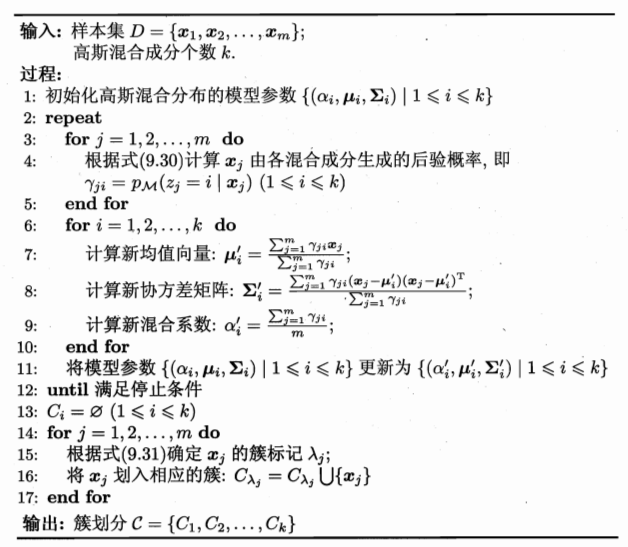


图4

**五、实验结果**

原始数据集划分与可视化结果如表2所示

表2

|  |  |
| --- | --- |
| 类/数 | 样本数/个 |
| 类1 | 250 |
| 类2 | 150 |
| 类3 | 100 |

原始数据集划分与可视化如图5所示

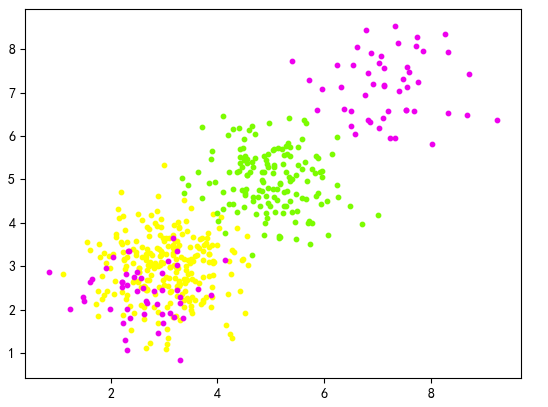


图5

K-Means算法实验结果

求得质点坐标如表3所示

表3

|  |  |  |
| --- | --- | --- |
| 类/质点坐标 | x | y |
| 类1 | 2.94805141 | 2.84547461 |
| 类2 | 4.92859254 | 4.93144926 |
| 类3 | 7.16504475 | 7.12121176 |

求得聚类后样本数如表4所示

表4

|  |  |
| --- | --- |
| 类/聚类后样本数 | 聚类后样本数/个 |
| 类1 | 292 |
| 类2 | 157 |
| 类3 | 51 |

求得NMI指标为0.710414311017。

K-Means算法实验可视化结果如图6所示

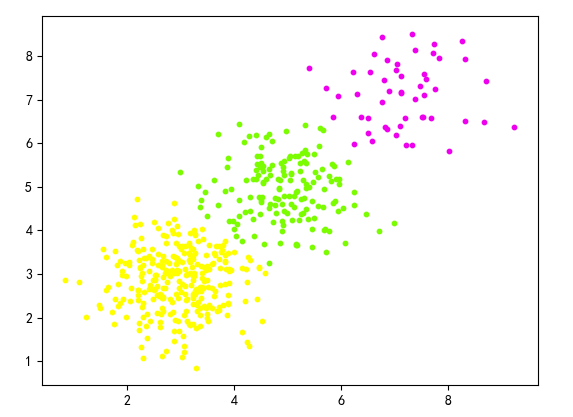


图6

GMM算法实验结果如表5所示

表5

|  |  |
| --- | --- |
| 类/数 | 聚类后样本数/个 |
| 类1 | 145 |
| 类2 | 304 |
| 类3 | 51 |

高斯分布组合系数结果如表6所示

表6

|  |  |  |  |
| --- | --- | --- | --- |
| 高斯分布组合系数 | 分布1 | 分布2 | 分布3 |
|  | 0.2888169 | 0.60819362 | 0.10298948 |

GMM算法计算质点坐标如表7所示

表7

|  |  |  |
| --- | --- | --- |
| 类/数 | x | y |
| 类1 | 5.01155579 | 4.98114383 |
| 类2 | 2.98814222 | 2.90373602 |
| 类3 | 7.13940504 | 7.10678027 |

高斯分布的协方差矩阵如表8所示

表8

|  |  |  |  |
| --- | --- | --- | --- |
| 高斯分布 | 分布1 | 分布2 | 分布3 |
| 协方差矩阵 | 0.440 -0.076  -0.076 0.469 | 0.475 0.049  0.049 0.557 | 0.644 0.049  0.049 0.555 |

求得NMI指标为0.696580528943。

GMM算法实验可视化结果如图7所示

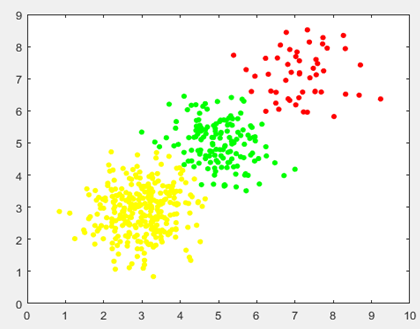


图7

实验结果分析

本实验的数据集有500个样本，每个样本有两个特征，共分为三类。在对数据集进行可视化后，发现数据相对清晰的分为三堆，其中有一堆的样本包含了两类数据。初步猜想聚类结果也应为三类，K-Means算法和GMM算法应该把每堆样本分一类。

通过把实验数据集通过K-Means算法和GMM算法聚类后，从数据集聚类实验结果和数据集可视化结果上看，数据集被聚为三类，每堆分为一类。另外从数据集聚类实验结果上看，K-Means算法和GMM算法聚类效果相差不大，但K-Means效果稍好。同时，从聚类的NMI指标上看，K-Means算法也比GMM算法高，也说明了K-Means算法比GMM算法稍好。

1. **实验心得体会**

K-Means算法和GMM算法的比较

相同点：

都是迭代执行的算法，且迭代的策略也相同：算法开始执行时先对需要计算的参数赋初值，然后交替执行两个步骤，一个步骤是对数据的估计（K-Means是估计每个点所属簇；GMM是计算隐含变量的期望；）;第二步是用上一步算出的估计值重新计算参数值，更新目标参数（K-Means是计算簇心位置；GMM是计算各个高斯分布的中心位置和协方差矩阵）。  
不同点：  
 1）需要计算的参数不同：K-Means是簇心位置；GMM是各个高斯分布的参数  
 2）计算目标参数的方法不同：K-Means是计算当前簇中所有元素的位置的均值；GMM是基于概率的算法，是通过计算似然函数的最大值实现分布参数的求解的。

通过这次实验最大的收获是对K-Means算法和GMM算法的理解程度有了相应的加深。起初只是理解K-Means算法的一点思想，但是到真正实现的时候才发现理解并不是很到位。对于GMM算法，我起初是完全不理解，根本没法实现，所以我又下了很大的功夫去理解GMM算法，并且也在实现GMM算法的同时对算法加深了一定的理解。

最后，我的另外一个比较大的收获就是对Python机器学习框架的认识。做大作业以前对机器学习的认识还只是停留在书本上，不知道如何实现、运用等。经过此次实验我对Python机器学习的框架有了一定的了解，可以逐步实现其他有意思的算法，希望之后能把相应的理论算法和实践掌握得更加熟练，越做越好吧。

**七、参考文献**

[1]周志华. 机器学习 : = Machine learning[M]. 清华大学出版社, 2016.

[2]于剑. 机器学习 从公理到算法[M]. 清华大学出版社，2017.

**八、附录**

读取数据

# -\*- coding: utf-8 -\*-

import xdrlib

import sys

import xlrd

#打开文件

def open\_excel(file):

try:

data = xlrd.open\_workbook(file)

return data

except Exception as e:

print(e)

#从目标文件中读取数据，并以列表的形式返回

def excel\_table\_byname(file, colnameindex=0, byname=u'Sheet1'):

data = open\_excel(file)

table = data.sheet\_by\_name(byname)

nrows = table.nrows

colnames = table.ncols

datalist = []

for rownum in range(0, nrows):

row = table.row\_values(rownum)

rowlist = []

for colnum in range(0, colnames):

rowlist.append(row[colnum])

datalist.append(rowlist)

return datalist

K-Means算法

# -\*- coding: utf-8 -\*-

from readDataByExcel import excel\_table\_byname

from NMI import nmi

from random import randint

import numpy as np

#距离函数

def distEclude(vectA, vectB):

dist = np.sum(np.power(vectA-vectB, 2))

return np.sqrt(dist)

#随机初始化类中心

def randCent(dataSet, k):

row = np.shape(dataSet)[0]

col = np.shape(dataSet)[1]

centroids = np.mat(np.zeros((k, col)))

init = []

for i in range(0, k):

id = randint(0, row)

init.append(id+1)

centroids[i,:] = dataSet[id,:]

print('初始化类中心：', init)

return centroids

#K-Means算法

def kMeans(dataSet, k, distMeans=distEclude, createCent=randCent):

row = np.shape(dataSet)[0]

cluster\_assment = np.mat(np.zeros((row, 2)))

centroids = createCent(dataSet, k)

clusterChanged = True

count = 1

while clusterChanged:

count += 1

clusterChanged = False

#每个样本加入其最近的簇

for i in range(0,row):

minDist = float('inf')

minIndex = -1

for j in range(0,k):

dist = distEclude(dataSet[i,:], centroids[j,:])

if minDist > dist:

minDist = dist; minIndex = j

if cluster\_assment[i,0] != minIndex:

clusterChanged = True

cluster\_assment[i,:] = minIndex,minDist\*\*2

#更新簇

for cent in range(0, k):

ptsInClust = dataSet[np.nonzero(cluster\_assment[:,0].A==cent)[0]]

centroids[cent,:] = np.mean(ptsInClust, axis=0)

if count > 300:

break

print('迭代次数：', count)

return centroids,cluster\_assment

def main():

print('开始读入数据集！！！')

data\_mat = np.mat(excel\_table\_byname(u'C:/Users/one\_two/Desktop/Machine Learning/The Seventh Chapter/chap\_7\_dataSet.xlsx'))

col = np.shape(data\_mat)[1] - 1

data\_set = data\_mat[:,:col]

print('数据集读入成功！！！')

print('K-Means开始！！！')

my\_centroids, cluster\_assment = kMeans(data\_set, 3)

print('K-Means结束！！！')

#print(cluster\_assment)

nmi\_index = nmi(data\_mat, cluster\_assment, 3)

print('NMI指标：', nmi\_index)

if \_\_name\_\_ == '\_\_main\_\_':

main()

GMM算法

# -\*- coding: utf-8 -\*-

from readDataByExcel import excel\_table\_byname

from ReadAndWriteCSV import write\_list\_to\_csv

from NMI import nmi

from random import randint

import numpy as np

#随机初始化类中心

def randCent(data, k):

N = np.shape(data)[0]

dimension = np.shape(data)[1]

centroids = np.mat(np.zeros((k, dimension)))

init = []

for i in range(k):

id = randint(0, N)

init.append(id+1)

centroids[i,:] = data[id,:]

print('初始化类中心：', init)

return centroids

#初始化协方差矩阵

def initDet(data, k):

dimension = np.shape(data)[1]

init = np.zeros((dimension, dimension))

for i in range(dimension):

init[i,i] = 0.1

L = [init for i in range(k)]

return np.array(L)

#高斯函数

def gaussian(x, mean, cov):

dimension = np.shape(cov)[0]

#print(dimension)

cov\_dev = np.linalg.det(cov+np.eye(dimension)\*0.01)

cov\_inv = np.linalg.inv(cov+np.eye(dimension)\*0.01)

xdiff = x - mean

#print(np.shape(xdiff))

#print(np.shape(cov\_inv))

prob = 1.0/(np.power(2\*np.pi, 1.0\*dimension/2)\*np.sqrt(np.abs(cov\_dev)))\*np.exp(-1.0/2\*np.dot(np.dot(xdiff, cov\_inv), xdiff.reshape((dimension, 1))))

return prob

#GMM算法

def GMM(data, k):

N = np.shape(data)[0]

dimension = np.shape(data)[1]

pis = np.ones(k) \* (1/k)

means = randCent(data, k)

cov = initDet(data, k)

gammas = [np.zeros(k) for i in range(N)]

loglikelyhood = 0

oldloglikelyhood = 1

count = 1

while np.abs(loglikelyhood-oldloglikelyhood) > 0.0001:

print('第 %d 次迭代' % count)

count += 1

oldloglikelyhood = loglikelyhood

#E\_step

for n in range(N):

respons = [pis[i]\*gaussian(data[n], means[i], cov[i]) for i in range(k)]

sum\_respons = np.sum(respons)

for i in range(k):

gammas[n][i] = respons[i]/sum\_respons

#M\_step

for i in range(k):

nk = np.sum([gammas[n][i] for n in range(N)])

means[i] = 1.0/nk \* np.sum([gammas[n][i]\*data[n] for n in range(N)], axis=0)

xdiffs = data - means[i]

cov[i] = 1.0/nk \* np.sum([gammas[n][i]\*xdiffs[n].reshape(dimension,1)\*xdiffs[n] for n in range(N)], axis=0)

pis[i] = nk/N

loglikelyhood = np.sum([np.log(np.sum([pis[i]\*gaussian(data[n], means[i], cov[i]) for i in range(k)])) for n in range(N)])

print('似然函数的最值：', loglikelyhood)

return pis, means, cov, gammas

#聚类标定

def clusterData(gammas):

cluster\_set = []

N,K = np.shape(gammas)

for n in range(N):

cluster=[np.where(gammas[n]==np.max(gammas[n]))[0][0], np.max(gammas[n])]

cluster\_set.append(cluster)

return np.array(cluster\_set)

def main():

print('开始读入数据集！！！')

data\_array = np.array(excel\_table\_byname(u'C:/Users/one\_two/Desktop/Machine Learning/The Seventh Chapter/chap\_7\_dataSet.xlsx'))

dimension = np.shape(data\_array)[1] -1

data\_set = data\_array[:,:dimension]

print('数据集读入成功！！！')

print('GMM开始！！！')

pis,means,cov,gammas = GMM(data\_set, 3)

print('GMN结束！！！')

print('高斯分布组合系数：')

print(pis)

print('高斯分布的均值：')

print(means)

print('高斯分布的协方差矩阵：')

print(cov)

cluster\_set = clusterData(gammas)

nmi\_index = nmi(data\_array, cluster\_set, 3)

print('NMI指标：', nmi\_index)

r,c = data\_array.shape

result = np.array(np.zeros((r, c+1)))

for i in range(r):

result[i,:c] = data\_array[i,:]

result[i,c] = cluster\_set[i,0]+1

write\_list\_to\_csv(result,['x','y','data\_kind','cluster\_kind'], u'C:/Users/one\_two/Desktop/MachineLearning/TheSeventh Chapter/chap\_7\_dataSet.csv')

if \_\_name\_\_ == '\_\_main\_\_':

main()

NMI指标：

# -\*- coding: utf-8 -\*-

import numpy as np

#读实验结果

def loadData(data\_set, cluster\_set, k):

n = np.shape(data\_set)[0]

matrix = np.mat(np.zeros((k, k)), dtype=int)

for i in range(0, n):

data\_kind = int(data\_set[i,2]-1+0.5)

cluster\_kind = int(cluster\_set[i,0]+0.5)

matrix[data\_kind,cluster\_kind] = matrix[data\_kind,cluster\_kind] + 1

print(matrix)

return matrix

#原始数据的entropy

def dataEntropy(matrix, k):

#print('data\_set\_kind')

mcol = np.sum(matrix, axis=1)

entropy = 0

for i in range(k):

v = mcol[i,0]

#print(v)

entropy += 1.0\*v\*np.log2(v/np.sum(matrix))

return –entropy

#聚类结果的entropy

def clusterEntropy(matrix, k):

#print('cluster\_set\_kind')

mrow = np.sum(matrix, axis=0)

entropy = 0

for i in range(k):

v = mrow[0,i]

entropy += 1.0\*v\*np.log2(v/np.sum(matrix))

return –entropy

#混合的entropy

def totalEntropy(matrix, k):

mrow = np.sum(matrix, axis=0)

#print(mrow)

mcol = np.sum(matrix, axis=1)

#print(mcol)

entropy = 0

account = np.sum(matrix)

for i in range(k):

vi = mcol[i, 0]

#print(vi)

for j in range(k):

vj = mrow[0, j]

#print(vj)

if matrix[i, j] != 0:

entropy += matrix[i, j] \* np.log2((matrix[i, j]\*account)/(vi \* vj))

return entropy

#nmi指标

def nmi(data\_set, cluster\_set, k):

matrix = loadData(data\_set, cluster\_set, k)

data\_set\_entropy = dataEntropy(matrix, k)

#print(data\_set\_entropy)

cluster\_set\_entropy = clusterEntropy(matrix, k)

#print(cluster\_set\_entropy)

total\_entropy = totalEntropy(matrix, k)

return 2\*total\_entropy / (data\_set\_entropy+cluster\_set\_entropy)

实验报告编写要求：

1. 正文要求小四号宋体，行间距1.5倍；
2. 英文要求小四号Times New Roman；
3. 实验报告配图用visio画图，每幅图应有编号和标题，编号和标题应位于图下方处，居中，中文用五号宋体；
4. 表格应为三线表，每个表格应有编号和标题，编号和标题应写在表格上方正中，距正文段前0.5倍行距。表中量与单位之间用“/”分隔，编号与标题中的中文用五号宋体；
5. 图、表、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。其标注形式应便于互相区别，可分别为：图1、表2、公式(5)等；