

**2020年春季学期  
计算学部《机器学习》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 陈啸 |
| 学号 | 1180300121 |
| 班号 | 1803104 |
| 电子邮件 | 291973093@qq.com |
| 手机号码 | 13685536606 |

**目录**

[1 2](#_Toc51916153)

[1.1 2](#_Toc51916154)

[2 2](#_Toc51916155)

[3 2](#_Toc51916156)

建议写出：问题的描述，解决问题的思路，实验的做法，实验结果的分析，结论，自拟标题

# 实验要求

目标：实现一个k-means算法和混合高斯模型，并且用EM算法估计模型中的参数。

测试：

用高斯分布产生k个高斯分布的数据（不同均值和方差）（其中参数自己设定）。

（1）用k-means聚类，测试效果；

（2）用混合高斯模型和你实现的EM算法估计参数，看看每次迭代后似然值变化情况，考察EM算法是否可以获得正确的结果（与你设定的结果比较）。

应用：可以UCI上找一个简单问题数据，用你实现的GMM进行聚类。

# 实验环境

系统:Win10

Gpu : 1060 630(不知道为啥 使用gpu加速以后依然使用的是集显)

外链库：cupy matplotlib

# 生成数据

## 二维正态分布

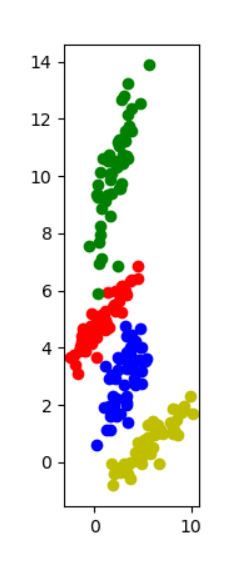
1. # 二维正态分布
2. mu = cp.array([[1, 5]])
3. Sigma = cp.array([[1, 0.5], [1.5, 3]])
4. R = cp.linalg.cholesky(Sigma)
5. print(R)
6. s = cp.dot(cp.random.randn(50, 2), R) + mu

生成一组二维高斯分布的数据，中心点在1 5

## 画图

画出数据在二维平面上的图像，每组数据使用一个单独的颜色

1. plt.subplot(144)
2. # 注意绘制的是散点图，而不是直方图
3. plt.plot(s[:,0],s[:,1],'ro')
4. plt.show()



# K-means算法

## 欧拉距离函数

1. # 欧拉距离函数
2. **def** EuclideanDistance(x , y , t = n):
3. cnt = 0
4. **for** \_t **in** range(t):
5. cnt = cnt + (x[\_t] - y[\_t]) \*\* 2
6. **return** cp.sqrt(cnt)

## 代码实现

1. # 创建c mju d lamda并初始化
2. mju = cp.ndarray(shape=(k , n) , dtype=float)
3. **for** i **in** range(k):
4. mju[i] = x[i]
5. d = cp.ndarray(shape = (m , k) , dtype = float)
6. c = cp.ndarray(shape = (k , m) , dtype = float)
7. lamda = cp.ndarray(shape = (m) , dtype = float)
8. **for** \_times **in** range(times):
9. **for** j **in** range(m):
10. **for** i **in** range(k):
11. c[i , j] = 0
12. **for** j **in** range(m):
13. lamda[j] = -1
14. \_min = 10000000000000000000000000.0
15. **for** i **in** range(k):
16. d[j , i] = EuclideanDistance(x[j] , mju[i])
17. **if** d[j , i] < \_min:
18. \_min = d[j , i]
19. lamda[j] = i
20. # print(lamda[j] , j)
21. c[int(lamda[j]) , j] = 1
22. **for** i **in** range(k):
23. **for** j **in** range(n):
24. mju[i , j] = 0
25. **for** j **in** range(m):
26. **if** c[i , j] == 1:
27. mju[i] = mju[i] + x[j]
28. mju[i] = mju[i] / sum(c[i])
29. cnt = 0
30. **for** i **in** range(m):
31. **for** j **in** range(m):
32. **if** ((lamda[i] == lamda[j]) **and** (y[i] == y[j])) **or** ((lamda[i] != lamda[j]) **and** (y[i] != y[j])) :
33. cnt = cnt + 1
34. **print**(cnt / m / m)

# GMM模型

## 高斯概率密度函数（分布函数）

1. # 高斯分布函数
2. def P(x , mju , sigma):
3. delta = cp.add(x , -1 \* mju)
4. R = -0.5 \* cp.dot(cp.dot(delta.T , cp.linalg.inv(sigma)) , delta)
5. S = cp.power((2 \* cp.pi) , n/2) \* cp.sqrt(cp.linalg.det(sigma))
6. **return** cp.exp(R) / S

## 数据读入以及初始化

1. # 迭代次数
2. times = 10
3. # 分类器数目
4. k=3
6. # 数据读入
7. f = open("data.txt","r")
8. str\_data = f.read()
9. ss = str\_data.split('\n')
10. m = len(ss)
11. str\_data = str\_data.split()
12. n = **int** (len(str\_data) / m - 1)
13. x = cp.ndarray(shape=(m , n) , dtype=**float**)
14. y = cp.ndarray(shape=(m , 1) , dtype=**float**)
15. **for** i in range(m):
16. **for** j in range(n):
17. x[i , j] = **float** (str\_data[i \* (n+1) + j])
18. y[i] = **float** (str\_data[i \* (n+1) + n])
19. # print(x)
21. # 创建alpha mju gamma sigma并初始化
22. alpha = cp.ndarray(shape = (k , 1) , dtype = **float**)
23. **for** i in range(k):
24. alpha[i , 0] = 1/k
25. gamma = cp.ndarray(shape=(m , k) , dtype=**float**)
26. mju = cp.ndarray(shape=(k , n) , dtype=**float**)
28. **for** i in range(k):
29. mju[i] = x[**int**((m/k\*i)+1)]
30. sigma = cp.ndarray(shape=(k , n , n) , dtype=**float**)
31. **for** i in range(k):
32. sigma[i] = cp.eye(n) \*0.1

## 分类

1. c = cp.ndarray(shape = (m) , dtype = **int**)
2. **for** j in range(m):
3. cnt = 0
4. ans = -1
5. **for** i in range(k):
6. **if** alpha[i , 0] \* P(x[j].T , mju[i].T , sigma[i]) > cnt:
7. cnt = alpha[i] \* P(x[j].T , mju[i].T , sigma[i])
8. ans = i
9. c[j] = ans

## 兰德系数

RI

兰德指数 (Rand index, RI), 将聚类看成是一系列的决策过程,即对文档集上所有N(N-1)/2 个【文档对】进行决策。当且仅当两篇文档相似时,我们将它们归入同一簇中。

正确决策:

TP 将两篇相似文档归入一个簇 (同 - 同)

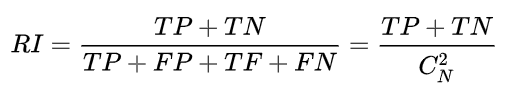
TN 将两篇不相似的文档归入不同的簇 (不同 - 不同)

错误决策:

FP 将两篇不相似的文档归入同一簇 (不同 - 同)

FN 将两篇相似的文档归入不同簇 (同- 不同) (worse)。

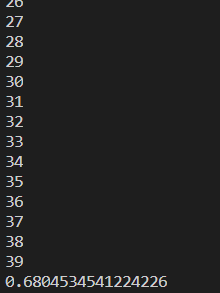
RI 则是计算「正确决策」的比率(精确率, accuracy).



1. cnt = 0
2. **for** i in range(m):
3. **for** j in range(m):
4. **if** ((c[i] == c[j]) and (y[i] == y[j])) or ((c[i] != c[j]) and (y[i] != y[j])) :
5. cnt = cnt + 1
6. print(cnt / m / m)

# 在自己生成的数据集上的结果

选取3.1中生成的数据集



在更大的迭代次数（42次）的时候，有部分系数已经变成了e-250,然后迭代后更小，直接变成了0，然后导致了不能计算逆元，因此迭代次数只能到40次

# 使用UCI数据集

## 使用iris数据集

Iris数据集是机器学习任务中常用的分类实验数据集，由Fisher在1936收集整理。Iris中文名是安德森鸢尾花卉数据集，英文全称是Anderson’s Iris data set，是一类多重变量分析的数据集。Iris一共包含150个样本，分为3类，每类50个数据，每个数据包含4个属性。可通过花萼长度，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。

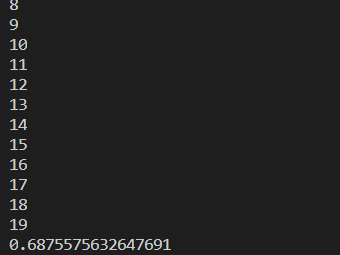
通俗地说，iris数据集是用来给莺尾花做分类的数据集，每个样本包含了花萼长度、花萼宽度、花瓣长度、花瓣宽度四个特征（下表中的前4列），我们需要建立一个分类器，该分类器可通过样本的四个特征来来判断样本属于山鸢尾（Setosa）、变色鸢尾（Versicolour）还是维吉尼亚鸢尾（Virginica）中的哪一个，即机器学习中的分类问题。  
iris的每个样本都包含了品种信息，即目标属性（第5列，也叫target或label）

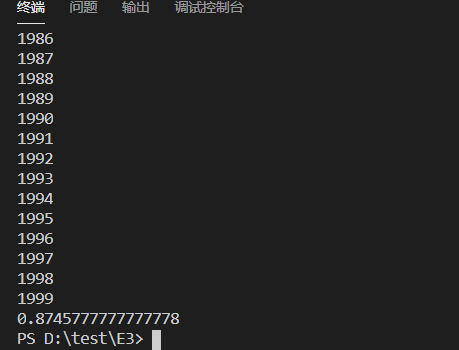
也就是说，我们可以使用前4个属性进行聚类分析，然后根据最后一个属性检验我们结果的正确性，此时考虑使用兰德系数来判断

## 数据集处理

1. f = open("iris.data","r")
2. ss=f.read().split()
3. f.close()
4. f = open("data.txt","w")
5. **for** j in range(150):
6. x=ss[j].split(',')
7. print(x)
8. **for** i in range(4):
9. f.write(x[i])
10. f.write(' ')
11. **if** x[4] == "Iris-setosa" :
12. f.write("0\n")
13. elif x[4] == "Iris-versicolor" :
14. f.write("1\n")
15. elif x[4] == "Iris-virginica" :
16. f.write("2\n")
17. **else** :
18. f.write("3\n")
20. f.close()

## 数据集实验结果





可见越大的迭代次数效果越好