### 模式识别第三次上机大作业

**人工智能32班 20009201208 张佳凡**

###### 问题重述

Kmeans和FCM算法性能比较

要求：

1. 查阅无监督聚类的评价标准有哪些，选择其中一个标准作为后续试验的验证指标。

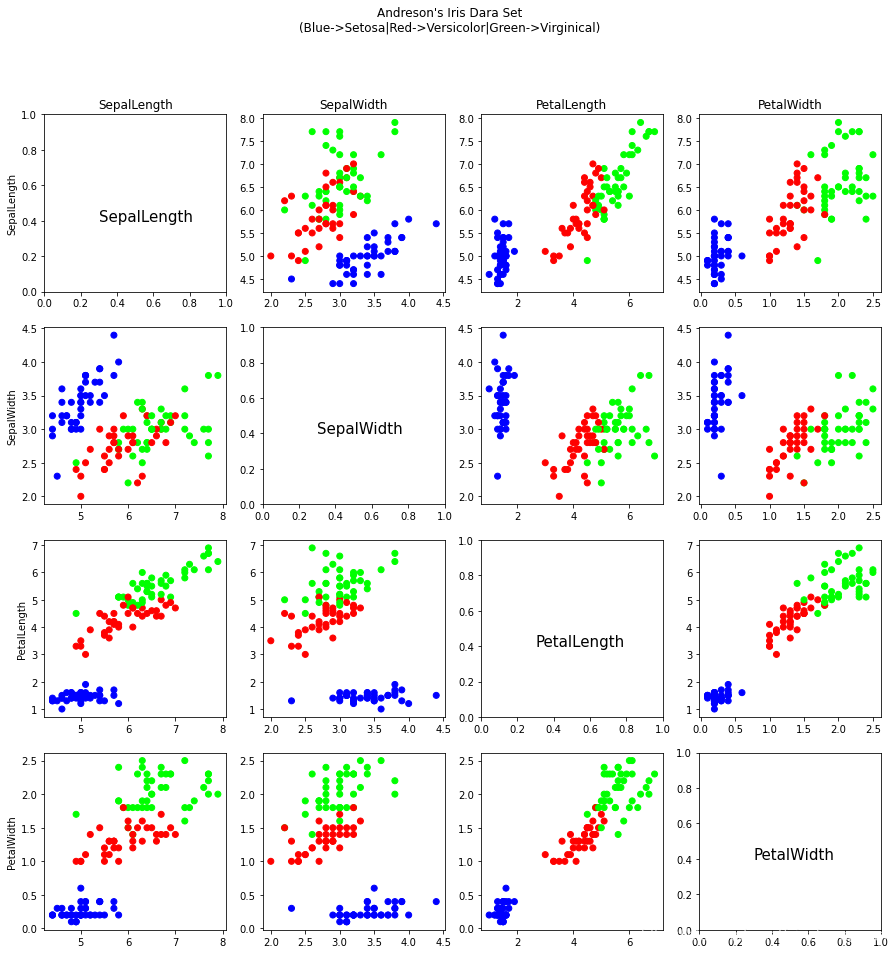
2. 在已有数据上分别验证两种聚类算法。同时，也可以先利用kmeans算法选择初始聚类中心，然后再使用FCM聚类，观察其结果。

选做：利用图像分割任务验证算法

###### 二、数据集介绍

**Iris鸢尾花数据集**

Iris数据集以鸢尾花的特征作为数据来源，数据集包含150个数据集，有4维，分为3 类（setosa、versicolour、virginica），每类50个数据，每个数据包含4个属性，花萼长度、宽度和花瓣长度、宽度。对数据进行统计学处理及数据可视化，分别研究二维数据可视化散点图如下：



###### 三、实验原理

3.1 KMeans算法

利用相似性度量方法来衡量数据集中所有数据之间的关系，将关系比较密切的数据划分到一个集合中。

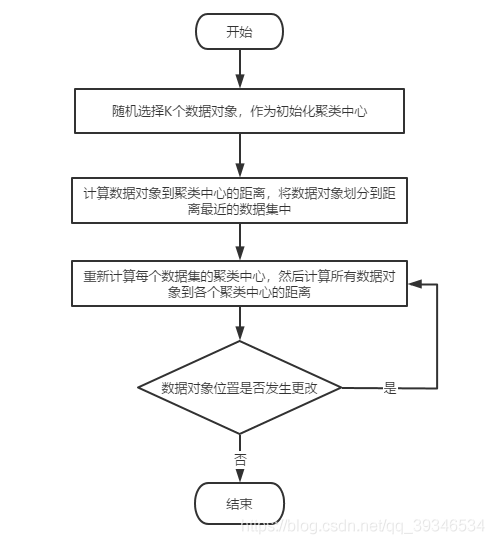
（1） K-means算法首先需要选择K个初始化聚类中心

（2） 计算每个数据对象到K个初始化聚类中心的距离，将数据对象分到距离聚类中心最近的那个数据集中，当所有数据对象都划分以后，就形成了K个数据集（即K个簇）

（3）接下来重新计算每个簇的数据对象的均值，将均值作为新的聚类中心

（4）最后计算每个数据对象到新的K个初始化聚类中心的距离，重新划分

（5）每次划分以后，都需要重新计算初始化聚类中心，一直重复这个过程，直到所有的数据对象无法更新到其他的数据集中。



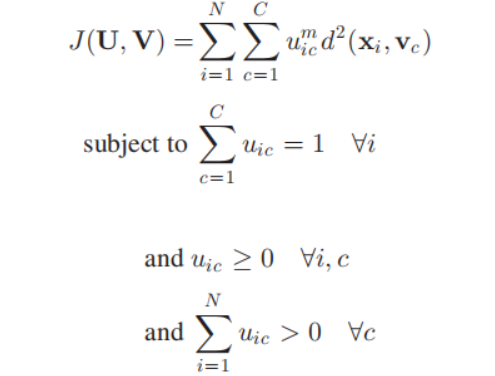
3.2 FCM算法

FCM算法是基于对目标函数的优化基础上的一种数据聚类方法。聚类结果是每一个数据点对聚类中心的隶属程度，该隶属程度用一个数值来表示。该算法允许同一数据属于多个不同的类。

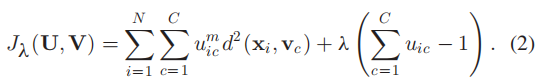
FCM算法是一种无监督的模糊聚类方法，在算法实现过程中不需要人为的干预。这种算法的不足之处:首先，算法中需要设定一些参数，若参数的初始化选取的不合适，可能影响聚类结果的正确性;其次，当数据样本集合较大并且特征数目较多时，算法的实时性不太好。

1. means也叫硬C均值聚类（HCM），而FCM是模糊C均值聚类，它是HCM的延伸与拓展，HCM与FCM最大的区别在于隶属函数（划分矩阵）的取值不同，HCM的隶属函数只取两个值：0和1，而FCM的隶属函数可以取[0,1]之间的任何数。K-means和FCM都需要事先给定聚类的类别数，而FCM还需要选取恰当的加权指数α，α的选取对结果有一定的影响，α属于[0,+∞)。

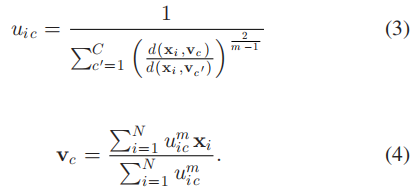
FCM的目标函数：



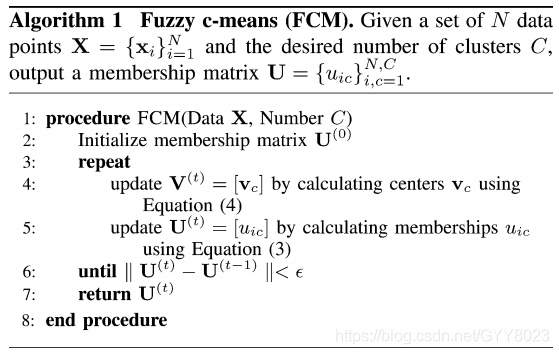
对于FCM算法采用拉格朗日乘子法进行优化



迭代公式



算法流程



###### 聚类算法性能度量

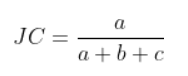
聚类性能度量指标分为外部指标和内容指标。

外部指标：

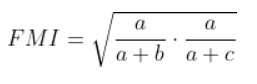
外部指标，也就是有参考标准的指标，通常也可以称为有监督情况下的一种度量聚类算法和各参数的指标。具体就是聚类算法的聚类结果和已知的（有标签的、人工标准或基于一种理想的聚类的结果）相比较，从而衡量设计的聚类算法的性能、优劣。

外部指标主要有：

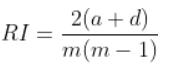
Jaccard系数（Jaccard Coefficient, JC）



FM指数（Fowlkes and Mallows Index, FMI）



Rand指数（Rand Index, RI）



上述性能度量的结果值均在[0,1]区间，值越大越好，值越大表明聚类结果和参考模型（有标签的、人工标准或基于一种理想的聚类的结果）直接的聚类结果越吻合，聚类结果就相对越好。

兰德系数（Rand index，RI）需要给定实际类别信息C，假设K是聚类结果

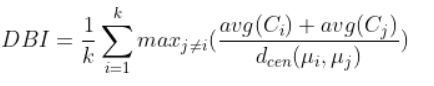
RI取值范围为[0,1]，值越大意味着聚类结果与真实情况越吻合。

对于随机结果，RI并不能保证分数接近零。为了实现“在聚类结果随机产生的情况下，指标应该接近零”，调整兰德系数（Adjusted rand index）被提出，它具有更高的区分度：

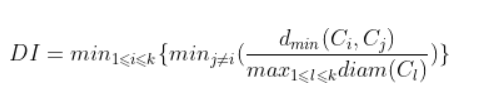
调整兰德系数（Adjusted rand index） ARI取值范围为[-1,1]，值越大意味着聚类结果与真实情况越吻合。从广义的角度来讲，ARI衡量的是两个数据分布的吻合程度。

常用的聚类性能度量内部指标：

DB指数（简称DBI）



Dunn指数（简称DI）



显然DBI的值越小越好， DI的值越大越好。

###### 实验结果

**4.1 KMeans算法运行过程及结果**

初始聚类中心: [array([6.4, 3.2, 5.3, 2.3]), array([5.7, 2.6, 3.5, 1. ]), array([4.7, 3.2, 1.3, 0.2])]

1 次迭代后聚类中心:

[6.57727273 2.99090909 5.34848485 1.89545455]

[5.65 2.64117647 4.04705882 1.25 ]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.88

2 次迭代后聚类中心:

[6.6015873 2.98571429 5.38412698 1.91587302]

[5.68378378 2.67837838 4.09189189 1.26756757]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.9

3 次迭代后聚类中心:

[6.63220339 2.99830508 5.43050847 1.93728814]

[5.72926829 2.6902439 4.15121951 1.3 ]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.9

4 次迭代后聚类中心:

[6.66481481 3.00740741 5.5 1.96851852]

[5.78913043 2.71304348 4.20869565 1.3326087 ]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.92

5 次迭代后聚类中心:

[6.702 3.016 5.556 1.992]

[5.822 2.728 4.256 1.36 ]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.92

6 次迭代后聚类中心:

[6.76956522 3.03695652 5.6 2.00869565]

[5.82962963 2.73148148 4.31481481 1.39259259]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.9066666666666666

7 次迭代后聚类中心:

[6.80232558 3.04418605 5.64883721 2.03023256]

[5.85438596 2.74210526 4.34561404 1.40877193]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.9

8 次迭代后聚类中心:

[6.8275 3.07 5.7 2.0625]

[5.885 2.74 4.37666667 1.41833333]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.8933333333333333

9 次迭代后聚类中心:

[6.85384615 3.07692308 5.71538462 2.05384615]

[5.88360656 2.74098361 4.38852459 1.43442623]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.8866666666666667

10 次迭代后聚类中心:

[6.85384615 3.07692308 5.71538462 2.05384615]

[5.88360656 2.74098361 4.38852459 1.43442623]

[5.006 3.418 1.464 0.244]

2 1 0 分类准确率为: 0.8866666666666667

已找到聚类结果

DI=0.0021

**4.2 FCM算法运行过程及结果**

1 次迭代后聚类中心:

[5.91164195 3.06220868 3.93009519 1.28738611]

[5.87288085 3.00649929 3.95499047 1.26521718]

[5.77250852 3.0626559 3.46005088 1.08502286]

2 次迭代后聚类中心:

[6.00934981 3.01512483 4.16684199 1.37015594]

[5.97661807 2.98179788 4.17105485 1.36275731]

[5.45864215 3.18368029 2.73751378 0.77729036]

3 次迭代后聚类中心:

[6.21444116 2.92794861 4.71666847 1.5974438 ]

[6.1664438 2.90831572 4.6746827 1.57491646]

[5.08231756 3.35567111 1.71365647 0.34836889]

4 次迭代后聚类中心:

[6.3327112 2.90225813 4.99471164 1.7221245 ]

[6.24201518 2.87516421 4.88073884 1.65874595]

[5.01132947 3.39803096 1.5070508 0.26036538]

5 次迭代后聚类中心:

[6.40244002 2.92298013 5.10637891 1.78014425]

[6.20596016 2.86366278 4.82300496 1.62775687]

[5.00692315 3.39633128 1.50193409 0.25849923]

6 次迭代后聚类中心:

[6.5165444 2.96226146 5.28385843 1.877979 ]

[6.11091041 2.83598949 4.6759615 1.5490192 ]

[5.00642339 3.39595368 1.50160406 0.25837917]

7 次迭代后聚类中心:

[6.64702469 3.0109922 5.48379858 1.98674387]

[6.00010218 2.80298424 4.50903979 1.46154461]

[5.00539212 3.39742862 1.49725483 0.2566096 ]

8 次迭代后聚类中心:

[6.72635597 3.03936924 5.59366087 2.0386194 ]

[5.9325263 2.77917937 4.41512727 1.41755991]

[5.00413431 3.40012597 1.49060112 0.25387409]

9 次迭代后聚类中心:

[6.75941387 3.0490057 5.63214448 2.05143003]

[5.90471812 2.76777904 4.38095434 1.40360039]

[5.00365454 3.40191212 1.48691594 0.25234162]

10 次迭代后聚类中心:

[6.77074525 3.05166593 5.64345332 2.05367166]

[5.8947837 2.76353164 4.37006409 1.39958459]

[5.00356884 3.40263965 1.48563018 0.25180504]

11 次迭代后聚类中心:

[6.77420092 3.05234484 5.64647839 2.05391838]

[5.8913365 2.76208248 4.36654456 1.39834577]

[5.00356075 3.40289361 1.48522188 0.25163444]

12 次迭代后聚类中心:

[6.77513005 3.05249344 5.64716348 2.05385849]

[5.89010856 2.76158336 4.36530485 1.39789833]

[5.00356128 3.4029802 1.48508826 0.25157834]

13 次迭代后聚类中心:

[6.77531379 3.05250631 5.6472326 2.05377946]

[5.88963823 2.76139869 4.36480609 1.39770265]

[5.00356172 3.4030111 1.48504084 0.25155822]

第1蔟的实际标签为:2 第2蔟的实际标签为:1 第3蔟的实际标签为:0

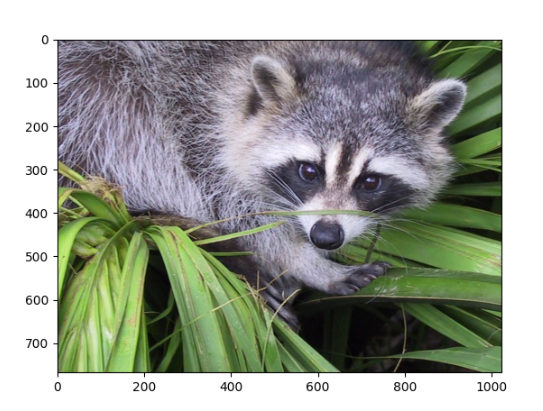
归1分类准确率为: 0.8933333333333333

模糊分类准确率为: 0.8247925342700775

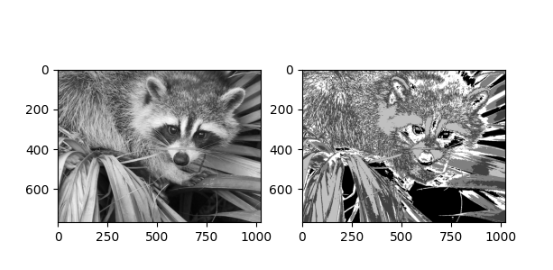
DI = 0.047

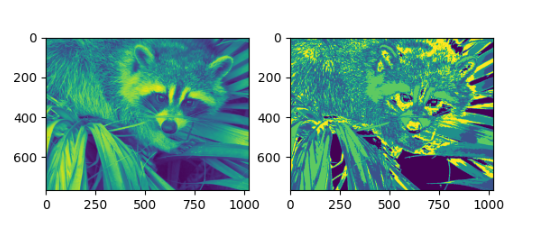
###### KMeans图像分割

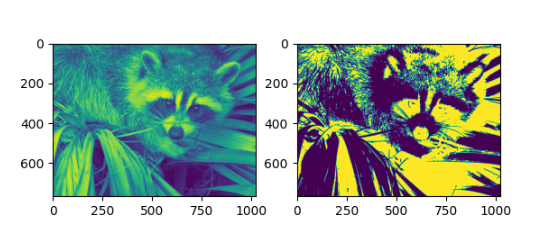
原始图像：



该图像不同通道用KMeans算法聚类分割结果如下：







###### 附录

1. **KMeans.py**

import numpy as np

import numpy.linalg as lg

import math

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import random

import copy

# 读入文件

path = r'iris.data'

f = open(path, 'r')

data = f.read()

data = data.split()

# 从str获取信息的函数 分为数据与标签

def get\_data(datastr):

species = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}

datastr = datastr.split(',')

data\_part = list(map(float, datastr[:4]))

label\_part = species[datastr[4]]

data\_part = np.array(data\_part)

return data\_part, label\_part

# 创建样本空间向量集合

Iris\_data\_list = []

Iris\_label\_list = []

for iris in data:

iris\_data, iris\_label = get\_data(iris)

Iris\_data\_list.append(iris\_data)

Iris\_label\_list.append(iris\_label)

# 初始化K个聚类中心

K = 3 # 聚类中心个数

z = [] # 存放K个聚类中心

while True: #随机选择聚类中心

newz = random.choice(Iris\_data\_list)

same\_flag = False

for i in z:

if (i == newz).all():

same\_flag = True

if not same\_flag:

z.append(newz)

if len(z) == K:

break

assert len(z) == K

print("初始聚类中心:", z)

# 迭代

iterations\_num = 0 # 迭代次数计数器

while True:

z\_old = copy.copy(z)

clusters = [[] for i in range(K)] # K个聚类蔟中包含的点

for i in range(len(Iris\_data\_list)):

distance = [] # 该点到K个中心的距离表

for j in range(K):

distance.append(lg.norm(Iris\_data\_list[i] - z[j]))

nearest\_zpoint = distance.index(min(distance)) # 找出距离其最近的聚类中心

clusters[nearest\_zpoint].append(i)

same\_flag = True

for i in range(K):

cluster\_mean = np.zeros(len(z[i]))

for j in range(len(clusters[i])):

cluster\_mean += Iris\_data\_list[clusters[i][j]]/len(clusters[i])

z[i] = cluster\_mean

if (z[i] != z\_old[i]).all():

same\_flag = False

iterations\_num += 1

print(iterations\_num, "次迭代后聚类中心:", '\n', z[0], '\n', z[1], '\n', z[2])

# 计算分类准确率

clusters\_labels = []

accuracy = 0

for i in range(K):

label\_list = []

for j in range(len(clusters[i])):

label\_list.append(Iris\_label\_list[clusters[i][j]])

true\_label = []

for j in range(K):

true\_label.append(label\_list.count(j))

accuracy += max(true\_label) # 选取数量最大的标签作为其标签

print(true\_label.index(max(true\_label)), end=' ')

clusters\_labels.append(true\_label.index(max(true\_label)))

accuracy = accuracy/len(Iris\_label\_list)

print("分类准确率为:", accuracy)

if same\_flag:

if len(set(clusters\_labels)) == K:

print('已找到聚类结果')

else:

print('聚类结果错误！')

break

1. **FCM.py**

import numpy as np

import numpy.linalg as lg

import math

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import random

import copy

# 读入文件

path = r'iris.data'

f = open(path, 'r')

data = f.read()

data = data.split()

# 从str获取信息的函数

def get\_data(datastr):

species = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}

datastr = datastr.split(',')

data\_part = list(map(float, datastr[:4]))

label\_part = species[datastr[4]]

data\_part = np.array(data\_part)

return data\_part, label\_part

# 创建样本空间向量集合

Iris\_data\_list = []

Iris\_label\_list = []

for iris in data:

iris\_data, iris\_label = get\_data(iris)

Iris\_data\_list.append(iris\_data)

Iris\_label\_list.append(iris\_label)

# 初始化K个聚类中心

K = 3 # 聚类中心个数

z = [0, 0, 0] # 存放K个聚类中心

# 初始化隶属矩阵U

U = np.random.rand(len(Iris\_data\_list), K)

for i in range(len(Iris\_data\_list)):

U[i] = U[i]/sum(U[i])

# 迭代

J = 0

a = 2 # 柔性参数

iterations\_num = 0 # 迭代次数计数器

while True:

z\_old = copy.copy(z)

U\_old = copy.copy(U)

J\_old = J

# 计算新聚类中心

for j in range(K):

sum\_ux = 0

sum\_u = 0

for i in range(len(Iris\_data\_list)):

sum\_ux += (U[i][j]\*\*a) \* Iris\_data\_list[i]

sum\_u += U[i][j]\*\*a

z[j] = sum\_ux/sum\_u

iterations\_num += 1

print(iterations\_num, "次迭代后聚类中心:", '\n', z[0], '\n', z[1], '\n', z[2])

# 计算代价函数

J = 0

for j in range(K):

for i in range(len(Iris\_data\_list)):

J += (U[i][j]\*\*a) \* (lg.norm(z[j]-Iris\_data\_list[i])\*\*2)

# 终止条件

if abs(J - J\_old) < 0.0001:

break

# 计算新矩阵U

for i in range(len(Iris\_data\_list)):

for j in range(K):

sum\_ud = 0

for k in range(K):

sum\_ud += ((lg.norm(z[j]-Iris\_data\_list[i])) / (lg.norm(z[k]-Iris\_data\_list[i]))) \*\* (2/(a-1))

U[i][j] = 1/sum\_ud

# 计算第几蔟的实际标签是什么

label\_order = []

for i in range(K):

K\_list = [0]\*K

for j in range(len(Iris\_data\_list)):

if np.argmax(U[j]) == i:

K\_list[Iris\_label\_list[j]] += 1

label\_order.append(K\_list.index(max(K\_list)))

assert len(set(label\_order)) == K, '出现了两类相同蔟！'

# 输出判断结果

for i in range(K):

print('第{}蔟的实际标签为:{}'.format(i+1, label\_order[i]), end=' ')

print()

# 计算实际标签对应的是第几蔟

un\_label\_order = [0]\*K

for i in range(K):

un\_label\_order[label\_order[i]] = i

accuracy1 = 0

for i in range(len(Iris\_data\_list)):

if U[i][un\_label\_order[Iris\_label\_list[i]]] == max(U[i]):

accuracy1 += 1

accuracy1 /= len(Iris\_data\_list)

print("归1分类准确率为:", accuracy1)

accuracy2 = 0

for i in range(len(Iris\_data\_list)):

accuracy2 += U[i][un\_label\_order[Iris\_label\_list[i]]]

accuracy2 /= len(Iris\_data\_list)

print("模糊分类准确率为:", accuracy2)