哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型：主修

实验题目： Kmeans、GMM

学号：1161000309

姓名： 高靖龙

1. **实验目的**

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。**二、实验要求及实验环境**

**实验要求：**

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。

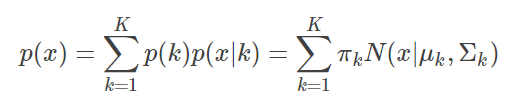
**实验环境：**

Manjaro linux

**三、设计思想（本程序中的用到的主要算法及数据结构）**

1. **Kmeans**
   1. **算法原理**Kmeans是一种典型的无监督聚类算法，实现简单，效果合格，应用广泛。算法的主要思想是将样本集划分为K个簇。让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。即最小化，对于该目标，采用启发式算法迭代，实质上可以证明是EM算法。
   2. **EM过程**Initial：针对k个聚点进行初始化，要求聚点间不能完全相同。  
      E-step：由k个聚点对样本点分类，样本点标记为距离其最近聚点的聚类类。  
      M-step：由带标记的样本重新计算k个聚点为各聚类内点的均值。
   3. **算法的实现**Kmeans实现较为简单，这里简单描述实现重点。  
      means的初始化：各聚点初始化为其序号填充。  
      类的存储：使用dic{k：X}存储，k为聚类序号，X为属于该类的样本。  
      在E步中，特别的，可能存在类为空集。  
      停止条件：当E步不在改变样本标记则说明收敛，停止算法输出结果。  
      需要注意的是means的初始化要差异化，不能存在两个mean是相同的。

|  |
| --- |
| **def** kmeans(XX,clusternum):  *"""  使用kmeans方法将一组向量XX聚为k类。  :param XX:样本  :param k:聚类数量  :return:labels={类序号:[属于该类的样本],0:[X1,X2,,,Xn]}  """* KK=[]  **for** i **in** range(clusternum):  KK.append([i]\*len(XX[0]))  labels = mark(XX, KK)  KK=recalClusterPoint(XX, labels, clusternum)  count=0  **while True**:  *#对XX打标签，标签0，1,2,，，k-1* tmp = labels  labels=mark(XX,KK)  flag=**True  for** l **in** labels:  **if** labels[l]!=tmp[l]:  flag=**False** count+=1  **if** flag **and** count>10:  **return** labels,np.array(KK)  *#重新计算聚点* KK =recalClusterPoint(XX,labels,clusternum) |

1. **GMM**
   1. **算法原理**对k聚类问题，混合高斯模型是基于齐次隐马尔科夫链的产生式模型。隐状态为k个n-GaussianDistribution，n为样本空间维数。隐状态间一步转移概率与条件无关。因此也可以当做转移概率作为混合比的k个高斯模型的混合，每个高斯模型代表一个分类。  
        
      使用MLE进行求解，具体使用EM算法给出结果。
   2. **EM过程**Initial：对于k个n-GaussianDistribution均值、协方差矩阵进行初始化，要求均值不相同。  
      E-step：根据估计的高斯函数模型计算类别后验概率。  
      M-step：根据类别后验概率重新估计各高斯模型均值方差
   3. **算法实现**实验中GMM实现了两个版本，初版没有合理使用矩阵便捷运算，使用了大量for语句，高斯函数发射概率的计算也是自己编写的。第二版主要区别在于使用矩阵对代码进行优化，有效减少代码长度，提高代码整洁程度以及可读性，高斯函数发射概率换用scipy.stats.multivariate\_normal计算。下面主要介绍一下实现过程中的难点以及使用log对类后验概率的计算溢出改进。  
        
      **初始化模型**这部分中，经过多次试验发现初值的选择对聚类结果有较大影响，符合EM算法的一般特性。

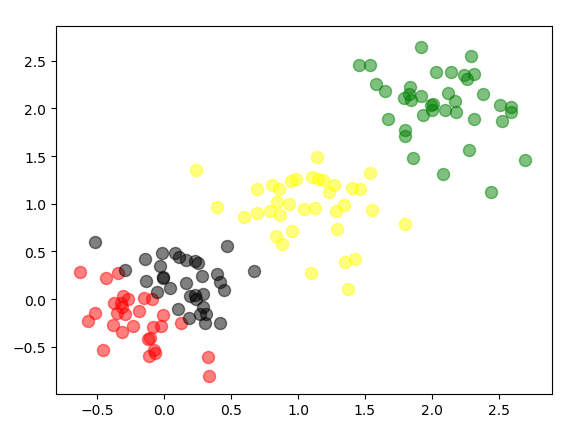
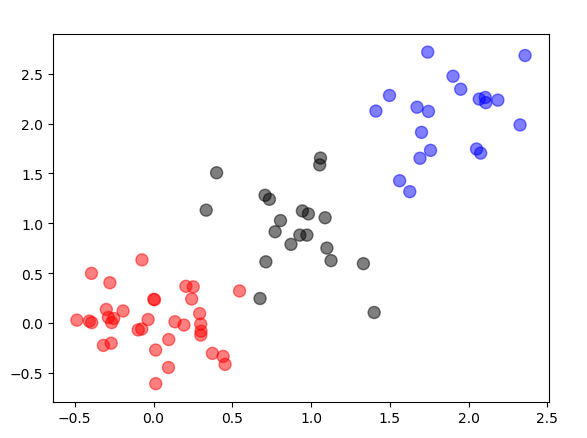
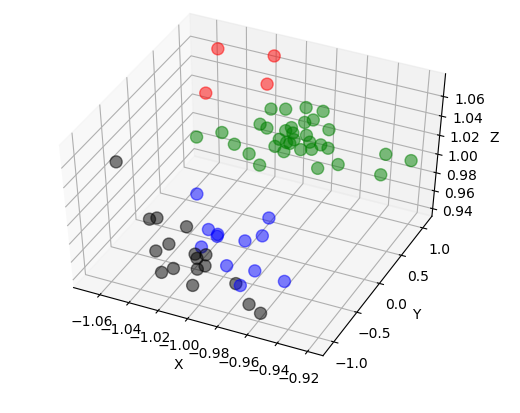
**E-step**  
这部分逻辑在def recalPKX(pk, means, vars, XX)中，重新计算类别后验概率。为了解决溢出问题使用log函数进行改进。  
对溢出有一定的减缓。  
  
**M-step**

|  |
| --- |
| *# M: max likehood with respect model:P(K) means vars* NK\_array = np.sum(pkx\_array, axis=0) **for** j **in** range(k):  means\_array[j] = np.sum(np.tile(pkx\_array[:, j], (len\_X, 1)).T \* XX, axis=0) / NK\_array[j] **for** j **in** range(k):  tmp = np.array(  [pkx\_array[i, j] \* (np.mat(XX[i] - means\_array[j]).T \* np.mat(XX[i] - means\_array[j])).A **for** i **in** range(len(XX))])  **assert** tmp.shape == (len(XX), len\_X, len\_X)  vars\_array[j] = np.sum(tmp, axis=0) / NK\_array[j]  **assert** vars\_array[j].shape == (len\_X, len\_X) |

这部分使用numpy矩阵整体运算优化代码执行速度。以下是第一版，作为对比。

|  |
| --- |
| *# M: max likehood with respect model:P(K) means vars* tmpmeans = [] tmpvars = [] tmppk = [] **for** j **in** range(k):  Nk = 0  mean = mat(zeros(D)).T  var = mat(zeros((D, D)))  **for** i **in** range(len(XX)):  v = pkx[i][j][0, 0]  mean += v \* XX[i]  Nk += v  **for** i **in** range(len(XX)):  delt = XX[i] - mean / Nk  var += pkx[i][j][0, 0] \* delt \* delt.T  tmpmeans.append(mean / Nk)  tmpvars.append(var / Nk)  tmppk.append(Nk / len(XX)) means = tmpmeans vars = tmpvars pk = tmppk |

**四、实验结果与分析**

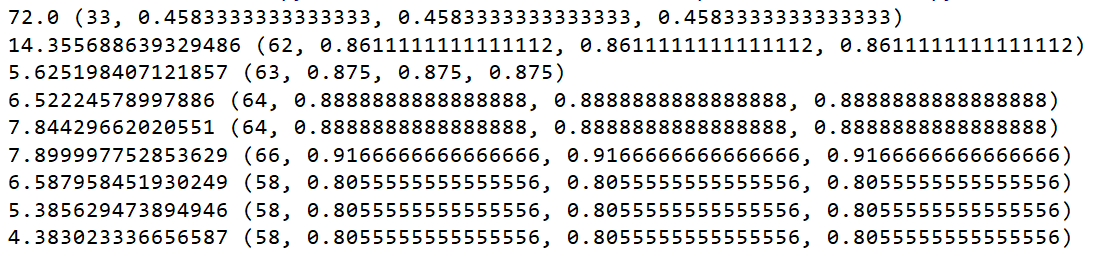
1. **K-means**K-means给出的结果较为生硬，聚类效果一般，可以接受。下面仅展示分类结果。  
   
2. **GMM**
   1. GMM分类结果是一个概率，相比较K-means该结果能继续作为概率参与计算而不是简单的分类。在本次实验中为了方便展示模型效果，我们仍然选取概率最大作为分类以统计性能。下面分别对二维3分类和3维4分类进行展示。如果要测试可以进入GMM.py取消main函数中相应函数的注释符号”#”进行测试。  
        
      
   2. **下面对GMM的初值敏感进行讨论,下面是在UCI.iris数据集上的测试结果。**  
      **初值函数**：means\_array=np.tile(np.random.random\_sample(k), (len\_X, 1)).T

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | TPtrain | TPtest | Len(X\_train) | Len(X\_test) |
| 1 | 80 | 35 | 112 | 38 |
| 2 | 90 | 29 | 112 | 38 |
| 3 | 101 | 35 | 112 | 38 |
| 4 | 80 | 29 | 112 | 38 |

可见仅仅是1的范围内随机，性能差异很大。这和EM算法的特性有关，因为初值设定的问题出现收敛偏差，导致收敛到局部最优而达不到全局最优，它是初值敏感的。

* 1. **下面分析在每次迭代中EM算法效果**

**输出格式**

类后验矩阵增量 (TP , precision , recall , F )  
  
可以看到在此次迭代中，EM算法没有落入最优点。这可能和我们的停止准则选取有关。在本次实验选取的停止条件是当类后验概率矩阵增量很小的时候停止算法。  
在开始阶段，EM过程确实是朝向似然函数增大方向迭代，但是越过了峰值。  
  
也可能是因为在计算高斯分布产生某X的概率时出现的奇异矩阵影响了最终结果。

**五、结论**

1. Kmeans能够给出一个较好的聚类结果，但是相比GMM其结果是一个类，不能参与概率运算，而GMM算出来的混合比更有价值，特别是对一些在分界上的点，能更好的代表他们的特性。
2. EM算法是初值敏感的，不同的初值会展现不同的效果。
3. GMM算法在迭代过程中并没有一直朝向似然函数增大方向上升，给出以下三种可能原因：
   1. 初值问题
   2. 在计算高斯模型产生某个X的概率时，奇异矩阵没有被正确处理。
   3. 代码实现中由于概率值太小造成了下溢出。

## 六、参考文献

[1]NumPy for Matlab users — NumPy v1.15 Manual[EB/OL]. Docs.scipy.org, 2018. (2018)[2018 -11 -15]. https://docs.scipy.org/doc/numpy-1.15.1/user/numpy-for-matlab-users.html.

[2]ArrogantL/AiLearning[EB/OL]. GitHub, 2018. (2018)[2018 -11 -15]. https://github.com/ArrogantL/AiLearning.

[3]BISHOP C. PATTERN RECOGNITION AND MACHINE LEARNING[M]. [S.l.]: SPRINGER-VERLAG NEW YORK, 2016.

[4]BOYD S, VANDENBERGHE L. Convex optimization[M]. Cambridge: Cambridge University Press, 2015.

**七、附录：源代码（带注释）**

|  |
| --- |
| from numpy.random import \* |
|  |  |
|  |  |
|  | def generateTwoDimensionalData\ |
|  | (num,featurenum=3, means=[[0,0], [2,2],[1,1],[2,2]], |
|  | vars=[[0.1, 0], [0, 0.1]]): |
|  | """ |
|  | 生成二维数据 |
|  | :param vars: |
|  | :param means: |
|  | :param num:数据量 |
|  | :return: |
|  | """ |
|  | XX = [] |
|  | Y = [] |
|  | for i in range(num): |
|  | if randint(0, featurenum) == 0: |
|  | XX.append(multivariate\_normal(means[0], vars)) |
|  | elif randint(0, featurenum) == 1: |
|  | XX.append(multivariate\_normal(means[1], vars)) |
|  | elif randint(0, featurenum) == 2: |
|  | XX.append(multivariate\_normal(means[2], vars)) |
|  | elif randint(0, featurenum) == 3: |
|  | XX.append(multivariate\_normal(means[3], vars)) |
|  |  |
|  |  |
|  | return XX, Y |
|  | v=0.001 |
|  | def generateThiDimensionalData\ |
|  | (num,featurenum=4, means=[[-1,-1,1], [-1,1,1], [-1,1,-1], [1,-1,-1]], |
|  | vars=[[v, 0,0], [0,v,0],[0, 0,v]]): |
|  | """ |
|  | 生成三维数据 |
|  | :param vars: |
|  | :param means: |
|  | :param num:数据量 |
|  | :return: |
|  | """ |
|  | XX = [] |
|  | Y = [] |
|  | for i in range(num): |
|  | if randint(0, featurenum) == 0: |
|  | XX.append(multivariate\_normal(means[0], vars)) |
|  | elif randint(0, featurenum) == 1: |
|  | XX.append(multivariate\_normal(means[1], vars)) |
|  | elif randint(0, featurenum) == 2: |
|  | XX.append(multivariate\_normal(means[1], vars)) |
|  | elif randint(0, featurenum) == 3: |
|  | XX.append(multivariate\_normal(means[1], vars)) |
|  |  |
|  |  |
|  |  |
|  | return XX, Y |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | XX, Y=generateThiDimensionalData(30) |
|  | print(XX) |
|  | print(Y) |

|  |
| --- |
| import matplotlib.pyplot as plt |
|  | import numpy as np |
|  | import pandas as pd |
|  | from scipy.stats import multivariate\_normal |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from mpl\_toolkits.mplot3d import Axes3D |
|  |  |
|  | from DataGenerator import generateTwoDimensionalData,generateThiDimensionalData |
|  |  |
|  | def gaussianMixtureModel(XX, k, minincrement=10 \*\* -5,tag=True): |
|  | """ |
|  | 使用kmeans方法将一组向量XX聚为k类。 |
|  | :param XX:样本 |
|  | :param k:聚类数量 |
|  | :param minincrement:E步最小增量，小于则停止迭代 |
|  | :return: pk, means, vars 即 混合概率，各聚类高斯均值，各聚类高斯协方差阵 |
|  | """ |
|  | # convert list X to column vector X |
|  | XX = np.array(XX) |
|  | len\_X = len(XX[0]) |
|  | # start: random model:P(K) means vars,Attention：mean[i]!=mean[j] for any i!=j, |
|  | # otherwise they will get the same parameters during the iteration. |
|  | # labels, means\_array = kmeans(XX.tolist(), k) |
|  |  |
|  | means\_array=np.tile(np.random.random\_sample(k), (len\_X, 1)).T |
|  |  |
|  | vars\_array = np.array([np.eye(len\_X)] \* k) |
|  | assert vars\_array.shape == (k, len\_X, len\_X) |
|  | pk\_array = np.ones(k) \* (1.0 / k) |
|  | # P(K|X) rowX columnK |
|  | pkx\_array = np.zeros(len(XX) \* k).reshape(len(XX), k) |
|  | # start EM process |
|  | while True: |
|  | # E: estimate P(K|X) |
|  | tmp = pkx\_array.copy() |
|  | pkx\_array = recalPKX(pk\_array, means\_array, vars\_array, XX) |
|  | # increment is less than the threshold,return result |
|  | increment = np.sum(np.abs(pkx\_array - tmp)) |
|  | print(increment) |
|  | if increment < minincrement: |
|  | break |
|  | # M: max likehood with respect model:P(K) means vars |
|  | NK\_array = np.sum(pkx\_array, axis=0) |
|  | for j in range(k): |
|  | means\_array[j] = np.sum(np.tile(pkx\_array[:, j], (len\_X, 1)).T \* XX, axis=0) / NK\_array[j] |
|  | for j in range(k): |
|  | tmp = np.array([pkx\_array[i, j] \* (np.mat(XX[i] - means\_array[j]).T \* np.mat(XX[i] - means\_array[j])).A for i in |
|  | range(len(XX))]) |
|  | assert tmp.shape == (len(XX), len\_X, len\_X) |
|  | vars\_array[j] = np.sum(tmp, axis=0) / NK\_array[j] |
|  | assert vars\_array[j].shape == (len\_X, len\_X) |
|  | return pk\_array, means\_array, vars\_array |
|  |  |
|  |  |
|  | def recalPKX(pk, means, vars, XX): |
|  | """ |
|  | E步重新计算类后验概率矩阵P（Y|X） |
|  | :param pk: |
|  | :param means: |
|  | :param vars: |
|  | :param XX: |
|  | :return: pkx\_array |
|  | """ |
|  | logpxi\_array = np.zeros(len(XX) \* len(pk)).reshape(len(XX), len(pk)) |
|  | pkx\_array = np.zeros(len(XX) \* len(pk)).reshape(len(XX), len(pk)) |
|  | for j in range(len(XX)): |
|  | for i in range(len(pk)): |
|  | try: |
|  | logpxi\_array[j, i] = multivariate\_normal.logpdf(XX[j], mean=means[i], cov=vars[i]) + np.log(pk[i]) |
|  | except: |
|  | print("Singular Matrix!!") |
|  | for k in range(len(pk)): |
|  | pkx\_array[:, k] = np.sum(np.exp(logpxi\_array - np.tile(logpxi\_array[:, k], (len(pk), 1)).T), axis=1) |
|  | pkx\_array = 1.0 / pkx\_array |
|  | return pkx\_array |
|  |  |
|  |  |
|  | def twoDimensionalClusterDisplay(): |
|  | clusternum = 3 |
|  | # 二维划分展示 |
|  | XX, Y = generateTwoDimensionalData(num=200) |
|  | XX = np.array(XX) |
|  | XX\_train\_array = XX[len(XX) // 2:] |
|  | XX\_test\_array = XX[len(XX) // 2:] |
|  | # XX\_test\_array = XX[:len(XX) // 2] |
|  | # 使用一半样本来训练 |
|  | pk, means, vars = gaussianMixtureModel(XX\_train\_array, clusternum) |
|  | # 绘制散点图 |
|  | colors = ['blue', 'red', 'black', 'green'] |
|  | pkx\_array = recalPKX(pk, means, vars, XX\_test\_array) |
|  | labels = np.argmax(pkx\_array, axis=1) |
|  | assert len(labels) == len(XX\_test\_array) |
|  | plt.scatter(XX\_test\_array[:, 0], XX\_test\_array[:, 1], s=75, c=[colors[i] for i in labels], alpha=0.5) |
|  | plt.show() |
|  | def thiDimensionalClusterDisplay(): |
|  | clusternum = 4 |
|  | # 二维划分展示 |
|  | XX, Y = generateThiDimensionalData(num=200,featurenum=clusternum) |
|  | XX = np.array(XX) |
|  | XX\_train\_array = XX[len(XX) // 2:] |
|  | XX\_test\_array = XX[len(XX) // 2:] |
|  | # XX\_test\_array = XX[:len(XX) // 2] |
|  | # 使用一半样本来训练 |
|  | pk, means, vars = gaussianMixtureModel(XX\_train\_array, clusternum) |
|  | # 绘制散点图 |
|  | colors = ['blue', 'red', 'black', 'green'] |
|  | pkx\_array = recalPKX(pk, means, vars, XX\_test\_array) |
|  | labels = np.argmax(pkx\_array, axis=1) |
|  | assert len(labels) == len(XX\_test\_array) |
|  |  |
|  | ax = plt.subplot(111, projection='3d') # 创建一个三维的绘图工程 |
|  | ax.scatter(XX\_test\_array[:, 0], XX\_test\_array[:, 1],XX\_test\_array[:, 2], s=75, c=[colors[i] for i in labels], alpha=0.5 ) # 绘制数据点 |
|  | ax.set\_zlabel('Z') # 坐标轴 |
|  | ax.set\_ylabel('Y') |
|  | ax.set\_xlabel('X') |
|  | plt.show() |
|  |  |
|  |  |
|  |  |
|  |  |
|  | def GMMandAnalyze(X\_train, X\_test, y\_test, k): |
|  | """ |
|  | train model with X\_train, test model with X\_test:y\_test |
|  | :param X\_train: |
|  | :param X\_test: |
|  | :param y\_test: |
|  | :param k: |
|  | :return: pk, means, vars, TP(correct clustering in X\_test:y\_test) |
|  | """ |
|  | X\_train=np.array(X\_train) |
|  | X\_test=np.array(X\_test) |
|  | y\_test = np.array(y\_test) |
|  | pk, means, vars = gaussianMixtureModel(X\_train, k) |
|  | pkx\_array = recalPKX(pk, means, vars, X\_test) |
|  | labels = np.argmax(pkx\_array, axis=1) |
|  | TP = 0 |
|  | cluster={} |
|  | for i in range(len(labels)): |
|  | cluster.setdefault(labels[i],[]) |
|  | cluster[labels[i]].append(i) |
|  | for v in cluster.values(): |
|  | vy=y\_test[v] |
|  | right=sorted(vy)[0] |
|  | for i in vy: |
|  | if i ==right: |
|  | TP+=1 |
|  |  |
|  |  |
|  | return pk, means, vars, TP |
|  |  |
|  |  |
|  | def processUCIheart(): |
|  | """ |
|  | process UCIiris data with GMM, give analyze result. |
|  | """ |
|  | data = pd.read\_csv("data/motified\_data.csv") |
|  | Y = data["5"].values.tolist() |
|  | data = data.drop("5", axis=1) |
|  | XX = data.iloc[:, 1:].values.tolist() |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(XX, Y, test\_size=0.25, random\_state=0) # 随机选择25%作为测试集，剩余作为训练集 |
|  |  |
|  | pk, means, vars, TPtest = GMMandAnalyze(X\_train, X\_test, y\_test, 4) |
|  | pk, means, vars, TPtrain = GMMandAnalyze(X\_train, X\_train, y\_train, 4) |
|  | print("TPtrain,TPtest,len(X\_train),len(X\_test)=", TPtrain, TPtest, len(X\_train), len(X\_test)) |
|  | # TPtrain,TPtest,len(X\_train),len(X\_test)= 80 35 112 38 |
|  |  |
|  | def preprocessData(): |
|  | """ |
|  | data-preprocess. |
|  | """ |
|  | data = pd.read\_csv("data/iris.csv") |
|  | data.to\_csv("data/motified\_data.csv") |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | # 测试说明：下面三条分别展示UCI数据、二维、三维数据的GMM效果 |
|  | # processUCIheart() |
|  | # twoDimensionalClusterDisplay() |
|  | thiDimensionalClusterDisplay() |

|  |
| --- |
| from numpy import \* |
|  | import numpy as np |
|  | from DataGenerator import generateTwoDimensionalData |
|  | import matplotlib.pyplot as plt |
|  |  |
|  | def kmeans(XX,clusternum): |
|  | """ |
|  | 使用kmeans方法讲一组向量XX聚为k类。 |
|  | :param XX:样本 |
|  | :param k:聚类数量 |
|  | :return:labels={类序号:[属于该类的样本],0:[X1,X2,,,Xn]} |
|  | """ |
|  | #随机生成k个聚点,聚点的维数要和X相同,clusterPoint。注意初始化k个点不能重叠 |
|  | KK=[] |
|  | for i in range(clusternum): |
|  | KK.append([i]\*len(XX[0])) |
|  | labels = mark(XX, KK) |
|  | KK=recalClusterPoint(XX, labels, clusternum) |
|  | count=0 |
|  | while True: |
|  | #对XX打标签，标签0，1,2,，，k-1 |
|  | tmp = labels |
|  | labels=mark(XX,KK) |
|  | flag=True |
|  | for l in labels: |
|  | if labels[l]!=tmp[l]: |
|  | flag=False |
|  | count+=1 |
|  | if flag and count>10: |
|  | return labels,np.array(KK) |
|  | #重新计算聚点 |
|  | KK =recalClusterPoint(XX,labels,clusternum) |
|  |  |
|  |  |
|  | def mark(XX,KK): |
|  | Labels = {} |
|  | for k in range(len(KK)): |
|  | Labels.setdefault(k,[]) |
|  | for x in range(len(XX)): |
|  | X=XX[x] |
|  | d,l=min((distanceOfKmeans(X,KK[i]),i) for i in range(len(KK))) |
|  | Labels[l].append(x) |
|  | return Labels |
|  |  |
|  | def distanceOfKmeans(X,K): |
|  | sum=0 |
|  | for i in range(len(X)): |
|  | sum+=power(X[i]-K[i],2) |
|  | return power(sum,0.5) |
|  | def recalClusterPoint(XX,Labels,clusternum): |
|  | KK = [] |
|  | for i in range(clusternum): |
|  | KK.append([0] \* len(XX[0])) |
|  | for k in Labels: |
|  | for x in Labels[k]: |
|  | X=XX[x] |
|  | KK[k]=[KK[k][m]+X[m] for m in range(len(KK[0]))] |
|  | if len(Labels[k])!=0: |
|  | KK[k]=[KK[k][m]/len(Labels[k]) for m in range(len(KK[0]))] |
|  | return KK |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | clusternum=4 |
|  | XX, Y = generateTwoDimensionalData(200) |
|  | labels,means=kmeans(XX, clusternum) |
|  | # 计算颜色值 |
|  | #color = np.arctan2(y, x) |
|  | # 绘制散点图 |
|  | colors=['red','yellow','green','black'] |
|  | for k in labels: |
|  | for i in labels[k]: |
|  | plt.scatter(XX[i][0],XX[i][1], s=75, c=colors[k], alpha=0.5) |
|  | plt.show() |