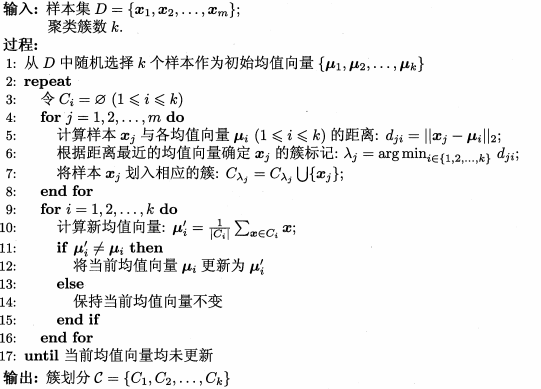
# k-means算法实现

## 概念

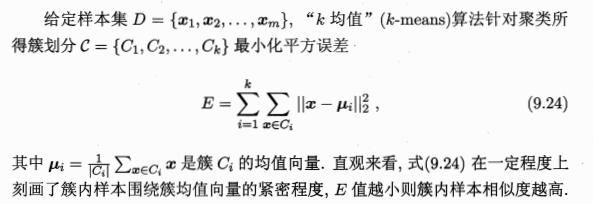
K-means算法是硬[聚类算法](https://baike.baidu.com/item/%E8%81%9A%E7%B1%BB%E7%AE%97%E6%B3%95)，是基于原型的[目标函数](https://baike.baidu.com/item/%E7%9B%AE%E6%A0%87%E5%87%BD%E6%95%B0)聚类方法的代表，它是数据点到原型的某种距离作为优化的目标函数，利用函数求极值的方法得到迭代运算的调整规则。K-means算法以[欧式距离](https://baike.baidu.com/item/%E6%AC%A7%E5%BC%8F%E8%B7%9D%E7%A6%BB)作为相似度测度，它是求对应某一初始聚类中心向量最优分类，使得评价指标最小。用[误差平方和](https://baike.baidu.com/item/%E8%AF%AF%E5%B7%AE%E5%B9%B3%E6%96%B9%E5%92%8C)准则函数作为聚类准则函数。

## 算法原理

随机选取k个初始均值向量，然后对数据集进行遍历，每次找到一个距离某个均值向量最近的点，将该数据集加入该均值向量所在的cluster，重复迭代多次，直到k个均值向量均不会更新为止（均值向量都相等），算法步骤如下所示：

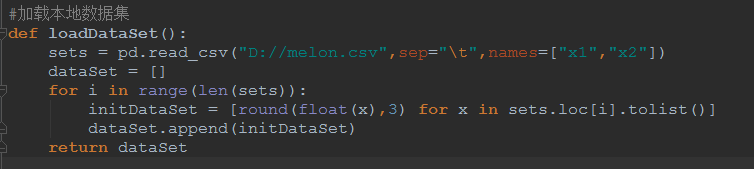


算法的评价函数(误差平方和 )公式如下所示

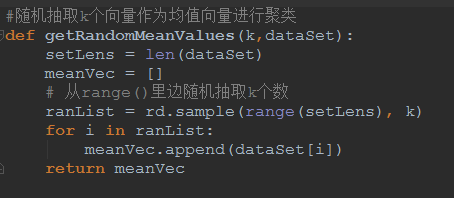


## 算法源码

1.首先就是对本地文件系统数据集的加载



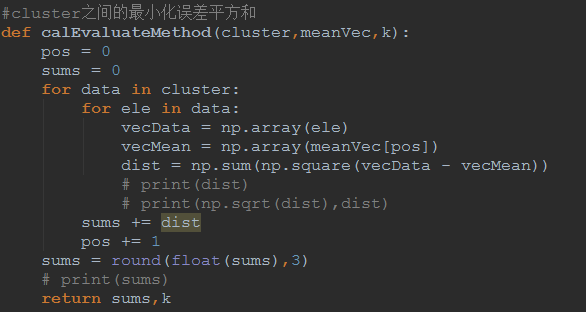
2.从数据集里边随机选取k个均值向量作为初始均指向量



3.遍历数据集，迭代划分类

#k均值算法,迭代更新均值向量，直至均值向量都不更新为止  
**def calDistanaceAndIteration**(dataSet,k):  
 #先找出均值向量  
 meansVec = getRandomMeanValues(k,dataSet)  
  
 cluster = []  
 **for** e **in** range(k):  
 cluster.append([])  
  
 nums = 0  
 count = 0  
 sortArray = []  
 **while** count == 0:  
 nums += 1  
 # cluster = [[], [], []] #每次都需要清空  
 cluster = []  
 **for** e **in** range(k):  
 cluster.append([])  
  
 **for** data **in** dataSet:  
 min = 0  
 #当前均值向量的位置  
 p = 0  
 **for** pos **in** range(len(meansVec)):  
 vecData = np.array(data)  
 vecMean = np.array(meansVec[pos])  
 dist = np.sqrt(np.sum(np.square(vecData - vecMean)))  
 sortArray.append(dist)  
  
 min = sortArray[0]  
 **for** m **in** range(len(sortArray)):  
 **if** sortArray[m] < min:  
 min = sortArray[m]  
 p = m  
 cluster[p].append(data)  
 sortArray = []  
 # 计算新的均值向量,如果每次三个均值都更新的话，count每次更新都加1，否则就是没更新，继续循环，划分新的均指向量  
 pos = 0  
 # print(meansVec)  
 tempList = []  
 **for** data **in** cluster:  
 sumx1 = 0  
 sumx2 = 0  
 lens = len(data)  
 # 计算初始的一个cluster的新的均值  
 **for** d **in** data:  
 sumx1 += d[0]  
 sumx2 += d[1]  
 meanX1, meanX2 = round(float(sumx1 / lens), 3), round(float(sumx2 / lens), 3)  
 tempList.append(meanX1)  
 tempList.append(meanX2)  
 # print(meanX1, meanX2)  
 # print(tempList)  
 #不等于，表示还得继续循环  
 **if** meansVec[pos] != tempList:  
 meansVec[pos] = tempList  
 count += 1 #要是变，说明有一个更新了，还需要继续循环  
 **else**:  
 **pass** pos += 1  
 tempList = []  
 **if** count != 0:  
 #还需要继续循环  
 count = 0  
 **else**: #不需要循环,给定一个较大的数值,让他直接退出  
 count = 100  
 # print(meansVec)  
 # print(count)  
 **return** cluster,meansVec,nums,k

1. k-means算法评价函数(cluster之间的最小化误差平方和), 绘制E-K图,找到最优k值(E-K第一个拐点处的位置) 如下所示

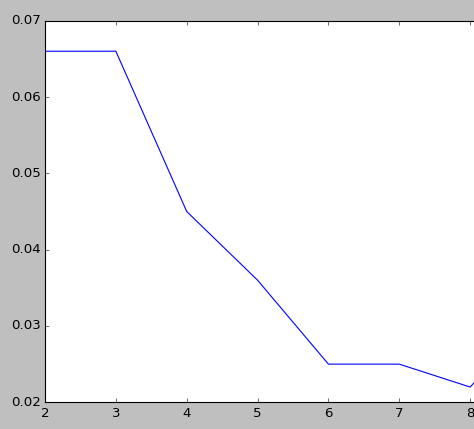


**for** i **in** range(2,10):  
 cluster, meanVec, nums, k = calDistanaceAndIteration(dataSet, i)  
 sums,k = calEvaluateMethod(cluster,meanVec,i)  
 eArray.append(sums)  
 kArray.append(i)  
drawEkCurve(eArray,kArray)  
  
#根据多次执行算法，绘制出E-K得知，k = 4的时候聚类效果是最好的

5.随机选取多个k值，迭代多次，计算误差平方和E，划出 E-K图

#聚类的个数  
# 从range()里边随机抽取1个数  
k = rd.sample(range(2,10), 1)  
k = k[0]  
# print(k)  
bestK = 0  
#E,k保存的位置  
eArray = []  
kArray = []  
dataSet = loadDataSet()  
  
# i表示随机产生的k值  
**for** i **in** range(2,10):  
 cluster, meanVec, nums, k = calDistanaceAndIteration(dataSet, i)  
 sums,k = calEvaluateMethod(cluster,meanVec,i)  
 eArray.append(sums)  
 kArray.append(i)  
drawEkCurve(eArray,kArray)  
  
#根据多次执行E-K得知，k = 4的时候聚类效果是最好的  
#所以令 k = 4的时候,迭代聚类  
bestK = 4  
cluster, meanVec, nums, k = calDistanaceAndIteration(dataSet, bestK)  
drawImage(cluster,bestK)

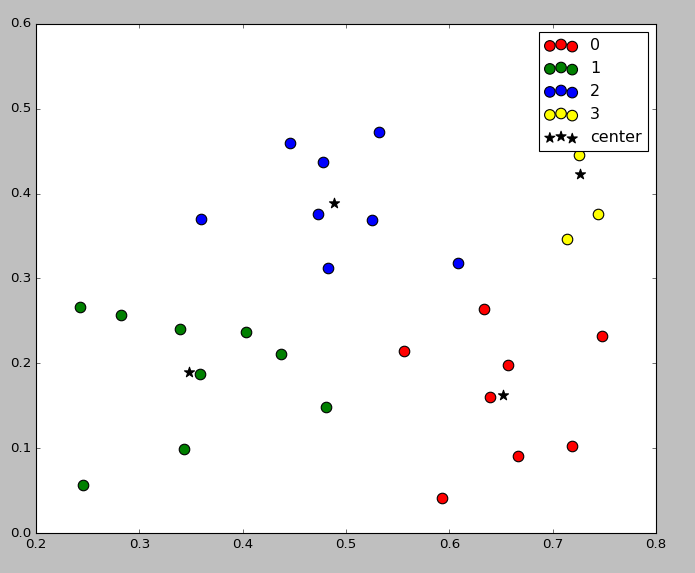
迭代多次求得的E-K图如下图所示



可以看出当k = 4的时候聚类效果是最好

## 4．算法执行结果

当k = 4的时候聚类效果如下图所示：



## 总结

聚类是无监督学习的一种，而 K-means 是聚类最常见的一种算法。 K-means 只能处理连续型属性的样本，若有标称型或离散型属性要先将其转化为连续型数值（用 one-hot编码）。基本简单的实现了k-means算法，由于k-means算法需要重复迭代多次，所以算法的效率不是太高，在本次算法中，未对噪声点进行处理，还有待处理，算法效率还需要进一步优化。

# EM算法实现

## 概念

当有部分数据缺失或者无法观察到时，EM[算法](http://lib.csdn.net/base/datastructure)提供了一个高效的迭代程序用来计算这些数据的最大似然估计。在每一步迭代分为两个步骤：期望（Expectation）步骤和最大化（Maximization）步骤，因此称为EM算法, EM算法解决这个的思路是使用启发式的迭代方法，既然我们无法直接求出模型分布参数，那么我们可以先猜想隐含数据（EM算法的E步），接着基于观察数据和猜测的隐含数据一起来极大化对数似然，求解我们的模型参数（EM算法的M步)。由于我们之前的隐藏数据是猜测的，所以此时得到的模型参数一般还不是我们想要的结果。不过没关系，我们基于当前得到的模型参数，继续猜测隐含数据（EM算法的E步），然后继续极大化对数似然，求解我们的模型参数（EM算法的M步)。以此类推，不断的迭代下去，直到模型分布参数基本无变化，算法收敛，找到合适的模型参数。

## 算法原理

  假设全部数据Z是由可观测到的样本X={X1, X2，……, Xn}和不可观测到的样本Z={Z1, Z2，……, Zn}组成的，则Y = X∪Z。EM算法通过搜寻使全部数据的似然函数Log(L(Z; h))的期望值最大来寻找极大似然估计，注意此处的h不是一个变量，而是多个变量组成的参数集合。此期望值是在Z所遵循的概率分布上计算，此分布由未知参数h确定。然而Z所遵循的分布是未知的。EM算法使用其当前的假设h`代替实际参数h，以估计Z的分布。



EM算法重复以下两个步骤直至收敛。

步骤1：估计（E）步骤：使用当前假设h和观察到的数据X来估计Y上的概率分布以计算Q( h` | h )。

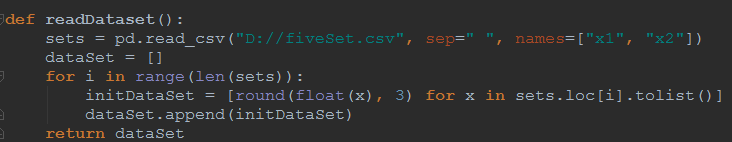


       步骤2：最大化（M）步骤：将假设h替换为使Q函数最大化的假设h:



## 算法源码

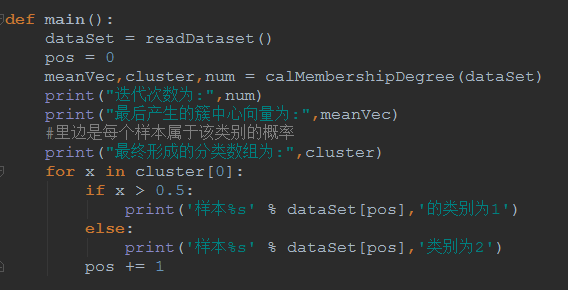
1. 数据集的加载



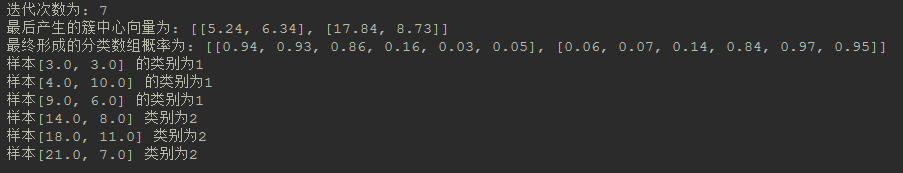
2.迭代计算数据集合和初始簇中心的的隶属度，就是计算概率

#迭代计算数据集合和初始簇中心的的隶属度，就是计算概率  
**def calMembershipDegree**(dataSet):  
  
 xAndYArray = []  
 xArray,yArray = getXandY(dataSet)  
 xAndYArray.append(xArray)  
 xAndYArray.append(yArray)  
  
 dataLen = len(dataSet)  
 #放置最终结果概率的数组，  
 cluster = [[],[]]  
  
 meanVec = [[],[]]  
  
 #迭代次数  
 num = 0  
  
 #初始类中心  
 clusterCenter = [[3.0,3.0],[4.0,10.0]]  
  
 #放置距离的数组  
 distance = []  
 outDistance1 = []  
 outDistance2 = []  
 outDistance = [[],[]]  
  
 posOut = 0  
 #循环条件  
 count = 0  
 **while** count == 0:  
 cluster = [[], []]  
 #迭代产生的均值向量  
 meanVec = [[], []]  
 num += 1  
 **for** data **in** dataSet:  
 posIn = 0  
  
 **for** ele **in** clusterCenter:  
 # distance = []  
 **if** data **in** clusterCenter:  
 **if** data == ele:  
 cluster[posIn].append(1)  
 **else**:  
 cluster[posIn].append(0)  
 posIn += 1  
 **else**:  
 vecData = np.array(data)  
 centors = np.array(ele)  
 dist = np.sum(np.square(vecData - centors))  
 distance.append(dist)  
 # cluster[posIn].append(dist)  
 posIn += 1  
 flag = data **in** clusterCenter  
 #E步 计算向量之间的距离，以便于后边方面进行计算  
 **if** flag **is False**:  
 sums = distance[0]+distance[1]  
 outDistance1.append(round(float(distance[1]/sums),2))  
 outDistance2.append(round(float(distance[0] / sums),2))  
 distance = []  
 posOut += 1  
 # print(outDistance1,outDistance2)  
 outDistance[0].append(outDistance1)  
  
 outDistance[1].append(outDistance2)  
 outDistance1 = []  
 outDistance2 = []  
 **for** i **in** range(2):  
 **for** x **in** outDistance[i]:  
 cluster[i].extend(x)  
 # print(cluster)  
 outDistance = [[],[]]  
 #M步  
 **for** ins **in** range(2):  
 #计算概率  
 **for** out **in** range(2):  
 aa = [(cluster[ins][n]\*\*2)\*xAndYArray[out][n] **for** n **in** range(dataLen)]  
 bb = [cluster[ins][m]\*\*2 **for** m **in** range(dataLen)]  
 # print(aa)  
 # print(bb)  
 aa = ft.reduce(add,aa)  
 bb = ft.reduce(add,bb)  
 result = retainDecimal(aa / bb)  
 meanVec[ins].append(result)  
  
 **for** i **in** range(2):  
 vecData = np.array(meanVec[i])  
 cent = np.array(clusterCenter[i])  
 #计算新的均值向量和旧的初始向量的差别，判断是否需要迭代  
 oDist = np.sum(np.square(vecData - cent))  
 **if** oDist < 0.0001: #说明均值接近，可不用在迭代  
 count += 1  
 **else**:  
 **pass  
 if** count >= 2: #两个新的均值向量和两个旧的均值向量都接近，无需迭代，直接退出迭代  
 count = 100  
 **else**: #还需要进行迭代，迭代完成之后更新均指向量  
 count = 0  
 clusterCenter[0] = meanVec[0]  
 clusterCenter[1] = meanVec[1]  
  
 **return** meanVec,cluster,num

3.主函数如下，k的个数为2



## 4．算法执行结果



## 5．总结

　EM算法里已知的是观察数据，未知的是隐含数据和模型参数，在E步，就是把模型参数的值固定，优化隐含数据的分布，而在M步，我们所做的事情是固定隐含数据分布，优化模型参数的值。