k-means 聚类分析实验报告

刘群*

地球系统科学研究中心

2015 年 4 月 24 日

1 k-means 算法简介

k-means 是一种聚类分析的算法，属于聚类分析中的划分方法, 主要思想是把数据划分成 k 个区域，通过基于距离的启发式的迭代，找到最优的聚类方案。其主要过程是:

1. 从问题空间中随意选取 k 个点，作为初始簇的中心;
2. 重复以下过程:
   (a) 对数据集中的每个数据点，计算它到不同簇中心的距离，根据距离的比较，将该数据点分配到与其距离最近的中心点;
   (b) 针对每一个簇，计算簇中所有点的均值，并以均值作为新的中心点;
3. 直到各簇的中心点不再发生变化.

但是 k-means 也有一些问题，即在初始条件下我们并不知道最好要分几类，而且在 k 值确定的基础上，计算结果倾向于收敛到局部最小值，而不是全局最小值, 并且对于数据中的离群点非常敏感。

2 k-means.py 程序简介

本次作业的主要任务是利用 k-means 方法对 Iris 数据进行分类，由于这组数据里本身就含有三个类别，因此在进行分类时，我也采取了 k = 3 来进行分类。主要过程如下:

1. 确定 k 值，由于这里的测试数据本身就是来自 3 个类别，因此这里取 $k = 3$. 当然，我们应该取不同的 k 值进行测试，但是对于不同的 k 值所分出的不同的类别，我们应该如何去判断优劣呢? 由于这个标准不能简单的选择到中心点的距离来判断，因此这里我就暂时不对 k 值进行讨论。
2. 随机选择 k 个" 中心点", 这里我采用的是最简单的选取方法，即采用 python 中自带的 random 模块产生 k 组随机数作为初始迭代的中心点。

```
center = np.array(random.sample(x_train, k))
```

3. 计算所有点到所有中心点的距离，这里我们采用的是欧式距离，
4. 求每个样本聚类的结果，在这里，我们是根据所有样本到中心点的距离来判断的。对于一个样本来说，其归属于距离自己最近的中心点，我们在这里是基于对距离的排序确定的。对于每一个样本来说，都要计算其到各个中心点的值，这样就可以生成 $150 \times k$ 的矩阵，在进行计算时，我们可以利用 tile 函数将数组进行扩充形成一个大矩阵，然后利用矩阵的相关操作进行运算，这样可以提高计算效率。然后我们利用 python 中的 argmin 函数，对每一行进行排序即可得到相应的类别分类。

```
for i in range(k):
    C = np.tile(center[i,:], (150,1))
    dist[:,i]=np.sqrt(np.sum((x_train-C)**2,axis=1))

labels = np.argmin(dist, axis=1)
```

*电子邮件: liu-q14@mails.tsinghua.edu.cn，学号: 2014211591

5. 计算新的中心点，这里我们采用的方法是将某一类别的数据提取出来，分别计算器几何中心（即每个维度求各自的平均）。

```
 newcenter = np.zeros((k,4))
for i in range(k):
    newcenter[i,:] =  np.average(x_train[labels==i,:], axis=0)
```

6. 判断中心点是否发生变化，若没有发生变则训练结束，若有变化，重复步骤 3。在进行判断时，我们引入了一个新的变量 changed 来判断。由于 $newcenter == center$ 之后，我们得到的是一个 bool 型矩阵，因此我们又利用了 $.all()$ 函数来判断矩阵中是否全为 True，如果全为 True，即新旧中心点是相同的，则停止迭代，否则就继续迭代。

```
changed = (newcenter == center).all() == False
```

下面是完整 kmeans 函数。

```
def kmeans(x_train, k):
    changed = True
    center = np.array(random.sample(x_train, k))
    dist = np.zeros((150, k))
    # Loop until the center won't change
    while changed:
        for i in range(k):
            C = np.tile(center[i,:], (150,1))
            dist[:,i]=np.sqrt(np.sum((x_train-C)**2,axis=1))

        labels = np.argmin(dist, axis=1)
        newcenter = np.zeros((k,4))
        for i in range(k):
            newcenter[i,:] =  np.average(x_train[labels==i,:], axis=0)
        changed = (newcenter == center).all() == False
        center = newcenter
    # Calculate the number of each group
    num_of_each_type = np.array([x_train[labels == i,:].shape[0] for i in range(k)])
    # Calcultate the total distance of the classification
    totalDistance = 0
    for i in range(k):
        #print sum(labels==i)
        newCenter = np.tile(center[i,:], (sum(labels==i),1))
        #print "dist"+str(np.sum((x_train[labels==i,:]-newCenter)**2,axis=1))
        totalDistance += np.sum((np.sum((x_train[labels==i,:]-newCenter)**2,axis=1)))
    # return type labels and total distance and number of each type
    return labels, np.sqrt(totalDistance), num_of_each_type
```

7. 评价聚类准确性，在这里调用 kmeans 函数时，我调用了 10 次，取出这 10 次中到中心点距离最小的情形作为最终聚类的结果。

```
 # Loading data
iris = datasets.load_iris()
x_train = iris.data
y_train = iris.target

k = 3
n = 10 # num of loops
labels = {}
num = {}
totalDist = np.zeros(n)
minDist = 10**6
minIndex = 0
for j in range(n):
    labels[j], totalDist[j], num[j] = kmeans(x_train, k)
    if totalDist[j] < minDist:
        minIndex = j
        minDist = totalDist[j]
```

然后对聚类的结果和原先分好的类的结果进行对比，统计聚类结果中实际上含有的各个类别的比例。然后将计算的结果输出，输出时按照列表的形式打印出来。由于中心点刚开始是随机选择的，也不能确定其到底是属于正确分类中的哪一组，因此真实的类别名用 $type0, type1, type2$ 来表示，而不是用真实的类别名。

```
 # Print the classification results
print "Number of samples in each claster is " + str(num[minIndex]) + "."
#print "\t\t %-15s %-15s %-15s" %("setosa", "versicolor", "virginica")
print "\t\t %-15s %-15s %-15s" %("type0", "type1", "type2")
for i in range(k):
    a = sum((y_train[labels[minIndex]==i])==0)*1.0/(x_train[labels[minIndex]==i,:]).shape
    [0]
    b = sum((y_train[labels[minIndex]==i])==1)*1.0/(x_train[labels[minIndex]==i,:]).shape
    [0]
```

```
    c = sum((y_train[labels[minIndex]==i])==2)*1.0/(x_train[labels[minIndex]==i,:]).shape
    [0]
    print "%8s\t %-15.4f %-15.4f %-15.4f" %("cluster"+str(i), a, b, c)
```

下面是某次程序运行后输出的结果：

```
 Number of samples in each claster is [50 38 62].
            type0           type1           type2
cluster0     1.0000          0.0000          0.0000
cluster1     0.0000          0.0526          0.9474
cluster2     0.0000          0.7742          0.2258
```

3 总结

聚类分析可以将大量数据分成几类，挖掘出数据中隐藏的我们还不确定的规律和特征，在数据分析中非常有用。但是对于初值的选择十分敏感，因此在选择初值时我们应该格外注意。但是在这里我们主要是采用了随机的方式来选择初值，这样的效果可能不是很好，但是通过 10 次或者更多次迭代，或许可以得到比较好的结果。