机器学习（三）：K均值聚类

机器学习中有两类的大问题，一个是分类，一个是聚类。

分类是根据一些给定的已知类别标号的样本，训练某种学习机器，使它能够对未知类别的样本进行分类。这属于supervised learning（监督学习）。

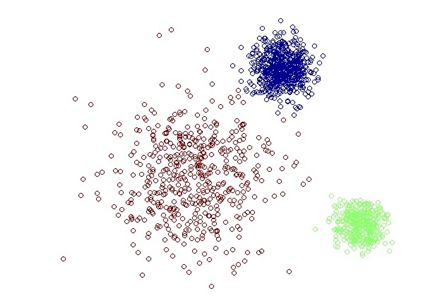
而聚类指事先并不知道任何样本的类别标号，希望通过某种算法来把一组未知类别的样本划分成若干类别，这在机器学习中被称作 unsupervised learning （无监督学习）。

k均值（k-means）算法就是一种比较简单的聚类算法。

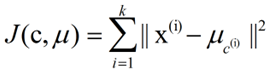
#一、k-means基本思想

K-means算法是聚类分析中使用最广泛的算法之一。它把n个对象根据他们的属性分为k个聚类以便使得所获得的聚类满足：同一聚类中的对象相似度较高；而不同聚类中的对象相似度较小。

比如下图中的n个点，就可以分为3个聚类，用不同的颜色表示。



k-means算法的基础是最小误差平方和准则。其代价函数是：



式中，μc(i)表示第i个聚类的均值。我们希望代价函数最小，直观的来说，各类内的样本越相似，其与该类均值间的误差平方越小，对所有类所得到的误差平方求和，即可验证分为k类时，各聚类是否是最优的。

上式的代价函数无法用解析的方法最小化，只能有迭代的方法。k-means算法是将样本聚类成 k个簇（cluster），其中k是用户给定的，其求解过程非常直观简单，具体算法描述如下：

（1）随机选取 k个聚类质心点

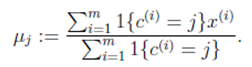
（2）重复下面过程直到收敛

{

对于每一个样例 i，计算其应该属于的类：

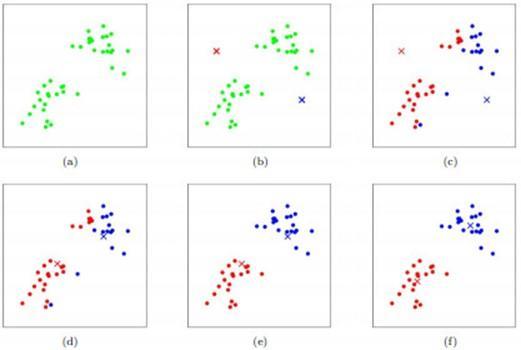
http://img.blog.csdn.net/20131226191250687?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvem91eHkwOQ==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

对于每一个类 j，重新计算该类的质心：



}

下图从(a)到(f)演示了对n个样本点进行K-means聚类的过程和效果，这里k取2。



#二、伪代码

```

创建k个点作为初始的质心点（随机选择）

当任意一个点的簇分配结果发生改变时

对数据集中的每一个数据点

对每一个质心

计算质心与数据点的距离

将数据点分配到距离最近的簇

对每一个簇，计算簇中所有点的均值，并将均值作为质心

```

#三、程序

编写此程序使用的是python 3，并且需要安装Numpy和matplotlib库。

安装方法可参考 https://www.jianshu.com/p/717521015940

##（一）

为了方便理解，咱们第一次只准备了四组数据，放在testSet.txt里

```

1.658985 4.285136

-3.453687 3.424321

4.838138 -1.151539

-5.379713 -3.362104

```

在程序里，可以逐步打印出执行结果

```

from numpy import \*

import matplotlib.pyplot as plt

# 计算欧氏距离的平方

def euclDistance(vector1, vector2):

return sum(power(vector2 - vector1, 2))

# 用随机样本初始化中心centroids

def initCentroids(dataSet, k):

numSamples, dim = dataSet.shape

centroids = zeros((k + 1, dim))

print("\n")

s = set()

for i in range(1, k + 1):

while True:

index = int(random.uniform(0, numSamples))

#这里为了方便查看结果，下面四行代码强制将两次随机数分别置为1和0

#若能理解此程序，则下面四行代码要删掉，才是真正的随机数

if 1 == i:

index = 1

elif 2 == i:

index = 0

#去重操作

if index not in s:

s.add(index)

break

print ("random index: %d" % index)

centroids[i, :] = dataSet[index, :]

print("\ncentroids:")

print(centroids)

return centroids

# 与中心的距离平方和，即最小误差平方和

def getTotalDistance(clusterAssment):

len = clusterAssment.shape[0]

Sum = 0.0

for i in range(len):

Sum = Sum + clusterAssment[i, 1]

return Sum

# k-means主算法

def kmeans(dataSet, k):

numSamples = dataSet.shape[0]

# 第一列存这个样本点属于哪个簇

# 第二列存这个样本点和样本中心的误差

clusterAssment = mat(zeros((numSamples, 2)))

for i in range(numSamples):

clusterAssment[i, 0] = -1

print("\nInitial clusterAssment:")

print(clusterAssment)

clusterChanged = True

# step 1: 初始化中心centroids

centroids = initCentroids(dataSet, k)

# 如果收敛完毕，则clusterChanged为False

while clusterChanged:

clusterChanged = False

# 对于每个样本点

print("\n")

for i in range(numSamples):

minDist = 100000.0

minIndex = 0

# 对于每个样本中心

# step 2: 找到最近的样本中心

for j in range(1, k + 1):

distance = euclDistance(centroids[j, :], dataSet[i, :])

print("i = %d, j = %d, distance = %s" % (i, j, distance))

if distance < minDist:

minDist = distance

minIndex = j

print("minIndex = %d, minDist = %f" % (minIndex, minDist))

# step 3: 更新样本点与中心点的分配关系

if clusterAssment[i, 0] != minIndex:

clusterChanged = True

clusterAssment[i, :] = minIndex, minDist

else:

clusterAssment[i, 1] = minDist

print ("clusterAssment:\n %s \n" % clusterAssment)

# step 4: 更新样本中心

for j in range(1, k + 1):

# 改变中心心位置

pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A == j)[0]]

print("\nPointsInCluster:\n%s\n" % pointsInCluster)

centroids[j, :] = mean(pointsInCluster, axis=0)

print("\ncentroids:\n%s\n" % centroids)

print ('Congratulations, cluster complete!')

return centroids, clusterAssment

# 以2D形式可视化数据

def showCluster(dataSet, k, centroids, clusterAssment):

numSamples, dim = dataSet.shape

if dim != 2:

print ("Sorry! I can not draw because the dimension of your data is not 2!")

return 1

mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']

if k > len(mark):

print ("Sorry! Your k is too large!")

return 1

# 绘制所有非中心样本点

for i in range(numSamples):

markIndex = int(clusterAssment[i, 0])

plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex - 1])

mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']

# 绘制中心点

for i in range(1, k + 1):

plt.plot(centroids[i, 0], centroids[i, 1], mark[i - 1], markersize=12)

plt.show()

# step 1: 载入数据

print ("step 1: load data...")

dataSet = []

fileIn = open('./testSet.txt')

for line in fileIn.readlines():

lineArr = line.strip().split('\t')

dataSet.append([float(lineArr[0]), float(lineArr[1])])

# step 2: 开始聚合...

print ("step 2: clustering...")

dataSet = mat(dataSet)

print ("dataSet:")

print (dataSet)

k = 2

centroids, clusterAssment = kmeans(dataSet, k)

#

print ("Final centroids:")

print (centroids)

print ("Final clusterAssment:")

print (clusterAssment)

print ("Total distance:")

print (getTotalDistance(clusterAssment))

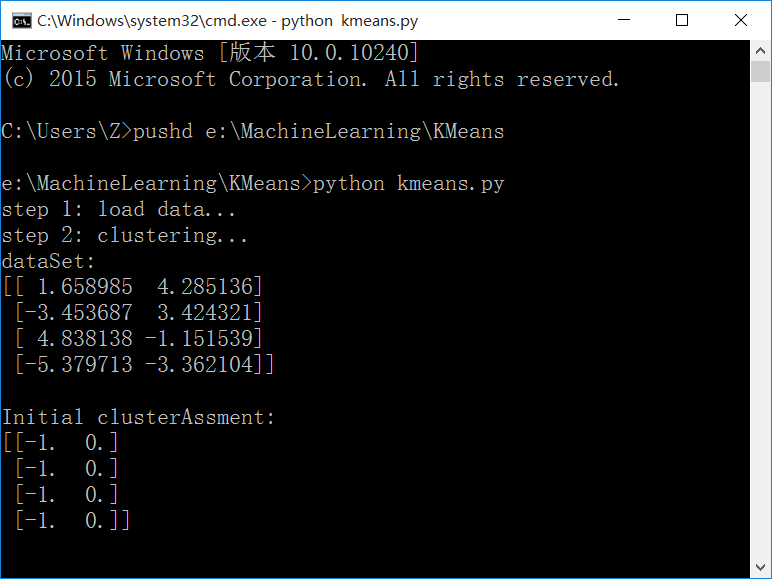
# step 3: 显示结果

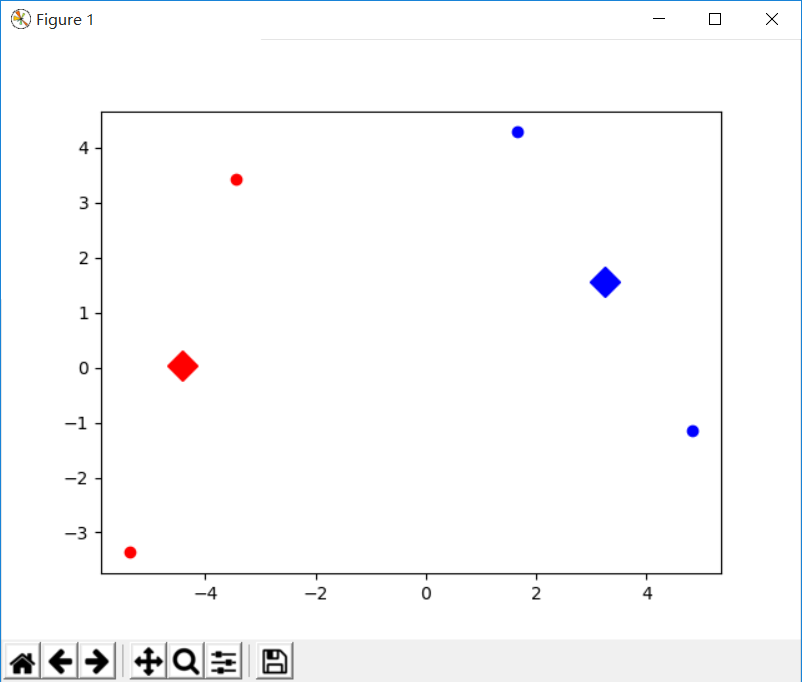
print ("step 3: show the result...")

showCluster(dataSet, k, centroids, clusterAssment)

```

运行结果





##（二）

为了观察到更好的效果，这次准备了多一点（80行）数据，放在testSet2.txt里

```

1.658985 4.285136

-3.453687 3.424321

4.838138 -1.151539

-5.379713 -3.362104

0.972564 2.924086

-3.567919 1.531611

0.450614 -3.302219

-3.487105 -1.724432

2.668759 1.594842

-3.156485 3.191137

3.165506 -3.999838

-2.786837 -3.099354

4.208187 2.984927

-2.123337 2.943366

0.704199 -0.479481

-0.392370 -3.963704

2.831667 1.574018

-0.790153 3.343144

2.943496 -3.357075

-3.195883 -2.283926

2.336445 2.875106

-1.786345 2.554248

2.190101 -1.906020

-3.403367 -2.778288

1.778124 3.880832

-1.688346 2.230267

2.592976 -2.054368

-4.007257 -3.207066

2.257734 3.387564

-2.679011 0.785119

0.939512 -4.023563

-3.674424 -2.261084

2.046259 2.735279

-3.189470 1.780269

4.372646 -0.822248

-2.579316 -3.497576

1.889034 5.190400

-0.798747 2.185588

2.836520 -2.658556

-3.837877 -3.253815

2.096701 3.886007

-2.709034 2.923887

3.367037 -3.184789

-2.121479 -4.232586

2.329546 3.179764

-3.284816 3.273099

3.091414 -3.815232

-3.762093 -2.432191

3.542056 2.778832

-1.736822 4.241041

2.127073 -2.983680

-4.323818 -3.938116

3.792121 5.135768

-4.786473 3.358547

2.624081 -3.260715

-4.009299 -2.978115

2.493525 1.963710

-2.513661 2.642162

1.864375 -3.176309

-3.171184 -3.572452

2.894220 2.489128

-2.562539 2.884438

3.491078 -3.947487

-2.565729 -2.012114

3.332948 3.983102

-1.616805 3.573188

2.280615 -2.559444

-2.651229 -3.103198

2.321395 3.154987

-1.685703 2.939697

3.031012 -3.620252

-4.599622 -2.185829

4.196223 1.126677

-2.133863 3.093686

4.668892 -2.562705

-2.793241 -2.149706

2.884105 3.043438

-2.967647 2.848696

4.479332 -1.764772

-4.905566 -2.911070

```

代码与上面的几乎一样，只是做了少量修改

```

from numpy import \*

import matplotlib.pyplot as plt

# 计算欧氏距离的平方

def euclDistance(vector1, vector2):

return sum(power(vector2 - vector1, 2))

# 用随机样本初始化中心centroids

def initCentroids(dataSet, k):

numSamples, dim = dataSet.shape

centroids = zeros((k + 1, dim))

print("\n")

s = set()

for i in range(1, k + 1):

while True:

index = int(random.uniform(0, numSamples))

#去重操作

if index not in s:

s.add(index)

break

centroids[i, :] = dataSet[index, :]

return centroids

# 与中心的距离平方和，即最小误差平方和

def getTotalDistance(clusterAssment):

len = clusterAssment.shape[0]

Sum = 0.0

for i in range(len):

Sum = Sum + clusterAssment[i, 1]

return Sum

# k-means主算法

def kmeans(dataSet, k):

numSamples = dataSet.shape[0]

# 第一列存这个样本点属于哪个簇

# 第二列存这个样本点和样本中心的误差

clusterAssment = mat(zeros((numSamples, 2)))

for i in range(numSamples):

clusterAssment[i, 0] = -1

clusterChanged = True

# step 1: 初始化中心centroids

centroids = initCentroids(dataSet, k)

# 如果收敛完毕，则clusterChanged为False

while clusterChanged:

clusterChanged = False

# 对于每个样本点

for i in range(numSamples):

minDist = 100000.0

minIndex = 0

# 对于每个样本中心

# step 2: 找到最近的样本中心

for j in range(1, k + 1):

distance = euclDistance(centroids[j, :], dataSet[i, :])

if distance < minDist:

minDist = distance

minIndex = j

# step 3: 更新样本点与中心点的分配关系

if clusterAssment[i, 0] != minIndex:

clusterChanged = True

clusterAssment[i, :] = minIndex, minDist

else:

clusterAssment[i, 1] = minDist

# step 4: 更新样本中心

for j in range(1, k + 1):

# 改变中心心位置

pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A == j)[0]]

centroids[j, :] = mean(pointsInCluster, axis=0)

print ('Congratulations, cluster complete!')

return centroids, clusterAssment

# 以2D形式可视化数据

def showCluster(dataSet, k, centroids, clusterAssment):

numSamples, dim = dataSet.shape

if dim != 2:

print ("Sorry! I can not draw because the dimension of your data is not 2!")

return 1

mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']

if k > len(mark):

print ("Sorry! Your k is too large!")

return 1

# 绘制所有非中心样本点

for i in range(numSamples):

markIndex = int(clusterAssment[i, 0])

plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex - 1])

mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']

# 绘制中心点

for i in range(1, k + 1):

plt.plot(centroids[i, 0], centroids[i, 1], mark[i - 1], markersize=12)

plt.show()

# step 1: 载入数据

print ("step 1: load data...")

dataSet = []

fileIn = open('./testSet2.txt')

for line in fileIn.readlines():

lineArr = line.strip().split('\t')

dataSet.append([float(lineArr[0]), float(lineArr[1])])

# step 2: 开始聚合...

print ("step 2: clustering...")

dataSet = mat(dataSet)

print ("dataSet:")

print (dataSet)

k = 4

centroids, clusterAssment = kmeans(dataSet, k)

#

print ("Final centroids:")

print (centroids)

print ("Final clusterAssment:")

print (clusterAssment)

print ("Total distance:")

print (getTotalDistance(clusterAssment))

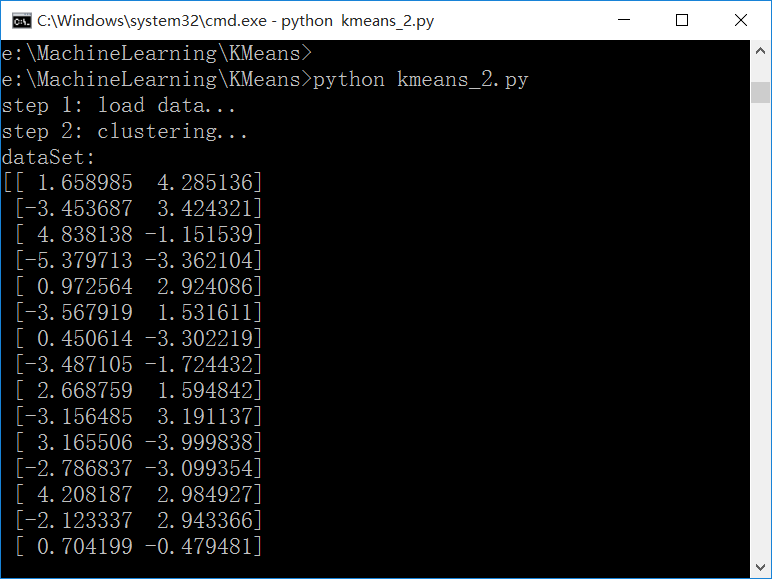
# step 3: 显示结果

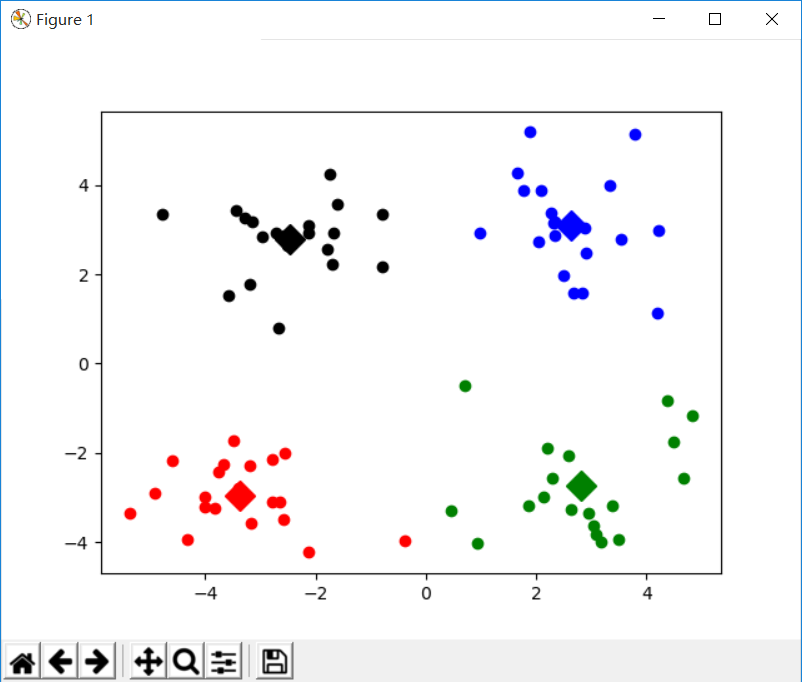
print ("step 3: show the result...")

showCluster(dataSet, k, centroids, clusterAssment)

```

运行结果：





#四、Github源码下载

<https://github.com/zhenghaishu/MachineLearning/tree/master/KMeans>

#五、参考

（1）<http://blog.csdn.net/zouxy09/article/details/17589329>

（2）<http://blog.csdn.net/eventqueue/article/details/73133617>