K-均值聚类

数据分析——K-均值聚类(K-Means)算法

在数据挖掘中，聚类是一个很重要的概念。传统的聚类分析计算方法主要有如下几种：划分方法、层次方法、基于密度的方法、基于网格的方法、基于模型的方法等。其中 K-Means 算法是划分方法中的一个经典的算法。

一、K-均值聚类(K-Means)概述

1、聚类：

“类”指的是具有相似性的集合，聚类是指将数据集划分为若干类，使得各个类之内的数据最为相似，而各个类之间的数据相似度差别尽可能的大。聚类分析就是以相似性为基础，在一个聚类中的模式之间比不在同一个聚类中的模式之间具有更多的相似性。对数据集进行聚类划分，属于无监督学习。

2、K-Means：

K-Means 算法是一种简单的迭代型聚类算法，采用距离作为相似性指标，从而发现给定数据集中的 K 个类，且每个类的中心是根据类中所有数值的均值得到的，每个类的中心用聚类中心来描述。对于给定的一个（包含 n 个一维以及一维以上的数据点的）数据集 X 以及要得到的类别数量 K，选取欧式距离作为相似度指标，聚类目标实施的个类的聚类平反和最小，即最小化：

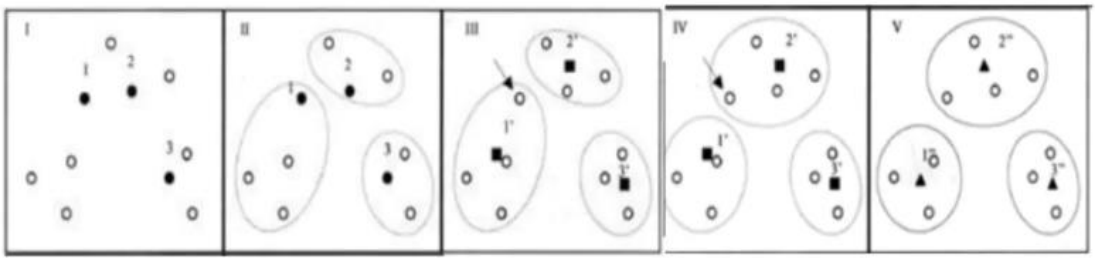
$$J = \sum_{k=1}^{k} \sum_{i=1}^{n} ||x_i - u_k||^2$$

公式.jpg

结合最小二乘法和拉格朗日原理，聚类中心为对应类别中各数据点的平均值，同时为了使算法收敛，在迭代的过程中，应使得最终的聚类中心尽可能的不变。

3、K-Means 算法流程：

-随机选取 K 个样本作为聚类中心；
-计算各样本与各个聚类中心的距离；
-将各样本回归于与之距离最近的聚类中心；
-求各个类的样本的均值，作为新的聚类中心；
-判定：若类中心不再发生变动或者达到迭代次数，算法结束，否则回到第二步。

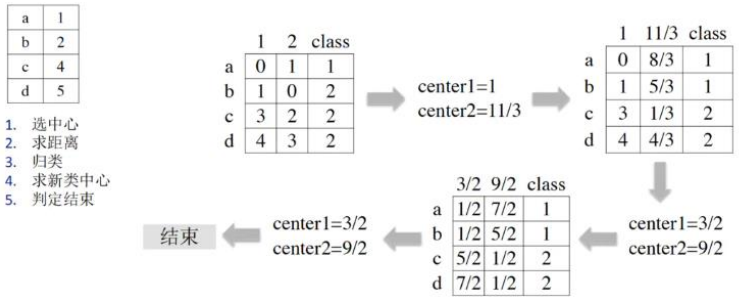


聚类演示-1.png

4、K-Means 演示举例

1.将 a~d 四个点聚为两类：

选定样本 a 和 b 为初始聚类中心，中心值分别为 1、2



聚类演示-2.png

2.将平面上的 100 个点进行聚类，要求聚为两类，其横坐标都为 0~99。

Python 代码演示：

```
import numpy as np


"""
    任务要求：对平面上的 100 个点进行聚类，要求聚类为两类，其横坐标都为 0 到 99。
"""
x = np.linspace(0, 99, 100)
```

```
y = np.linspace(0, 99, 100)
k = 2
n = len(x)
dis = np.zeros([n, k+1])

# 1.选择初始聚类中心
center1 = np.array([x[0], y[0]])
center2 = np.array([x[1], y[1]])
iter_ = 100

while iter_ > 0:
    # 2.求各个点到两个聚类中心距离
    for i in range(n):
        dis[i, 0] = np.sqrt((x[i] - center1[0])**2 + (y[i] - center1[1])**2)
        dis[i, 1] = np.sqrt((x[i] - center2[0])**2 + (y[i] - center2[1])**2)
        # 3.归类
        dis[i, 2] = np.argmin(dis[i,:2])  # 将值较小的下标值赋值给 dis[i, 2]

    # 4.求新的聚类中心
    index1 = dis[:, 2] == 0
    index2 = dis[:, 2] == 1
    center1_new = np.array([x[index1].mean(), y[index1].mean()])
    center2_new = np.array([x[index2].mean(), y[index2].mean()])

    # 5.判定聚类中心是否发生变换
    if all((center1 == center1_new) & (center2 == center2_new)):
       # 如果没发生变换则退出循环，表示已得到最终的聚类中心
       break

    center1 = center1_new
    center2 = center2_new

# 6.输出结果以验证
print(dis)
```

结果如下：
其中第 3 项代表聚类：

```
[[ 34.64823228 105.3589104    0.       ]
 [ 33.23401872 103.94469683   0.       ]
 [ 31.81980515 102.53048327   0.       ]
 [ 30.40559159 101.11626971   0.       ]
 [ 28.99137803  99.70205615   0.       ]
 [ 27.57716447  98.28784258   0.       ]
 [ 26.1629509   96.87362902   0.       ]
 [ 24.74873734  95.45941546   0.       ]
 [ 23.33452378  94.0452019    0.       ]
 [ 21.92031022  92.63098834   0.       ]
 [ 20.50609665  91.21677477   0.       ]
 [ 19.09188309  89.80256121   0.       ]
 [ 17.67766953  88.38834765   0.       ]
 [ 16.26345597  86.97413409   0.       ]
 [ 14.8492424   85.55992052   0.       ]
 [ 13.43502884  84.14570696   0.       ]
 [ 12.02081528  82.7314934    0.       ]
 [ 10.60660172  81.31727984   0.       ]
 [  9.19238816  79.90306627   0.       ]
 [  7.77817459  78.48885271   0.       ]
 [  6.36396103  77.07463915   0.       ]
 [  4.94974747  75.66042559   0.       ]
 [  3.53553391  74.24621202   0.       ]
 [  2.12132034  72.83199846   0.       ]
 [  0.70710678  71.4177849    0.      ]
 [  0.70710678  70.00357134   0.       ]
 [  2.12132034  68.58935778   0.       ]
 [  3.53553391  67.17514421   0.       ]
 [  4.94974747  65.76093065   0.       ]
 [  6.36396103  64.34671709   0.       ]
 [  7.77817459  62.93250353   0.       ]
 [  9.19238816  61.51828996   0.       ]
 [ 10.60660172  60.1040764    0.       ]
```

```
[ 12.02081528  58.68986284   0.       ]
[ 13.43502884  57.27564928   0.       ]
[ 14.8492424   55.86143571   0.       ]
[ 16.26345597  54.44722215   0.       ]
[ 17.67766953  53.03300859   0.       ]
[ 19.09188309  51.61879503   0.       ]
[ 20.50609665  50.20458146   0.       ]
[ 21.92031022  48.7903679    0.       ]
[ 23.33452378  47.37615434   0.       ]
[ 24.74873734  45.96194078   0.       ]
[ 26.1629509   44.54772721   0.       ]
[ 27.57716447  43.13351365   0.       ]
[ 28.99137803  41.71930009   0.       ]
[ 30.40559159  40.30508653   0.       ]
[ 31.81980515  38.89087297   0.       ]
[ 33.23401872  37.4766594    0.       ]
[ 34.64823228  36.06244584   0.       ]
[ 36.06244584  34.64823228   1.       ]
[ 37.4766594   33.23401872   1.       ]
[ 38.89087297  31.81980515   1.       ]
[ 40.30508653  30.40559159   1.       ]
[ 41.71930009  28.99137803   1.       ]
[ 43.13351365  27.57716447   1.       ]
[ 44.54772721  26.1629509    1.       ]
[ 45.96194078  24.74873734   1.       ]
[ 47.37615434  23.33452378   1.       ]
[ 48.7903679   21.92031022   1.       ]
[ 50.20458146  20.50609665   1.       ]
[ 51.61879503  19.09188309   1.       ]
[ 53.03300859  17.67766953   1.       ]
[ 54.44722215  16.26345597   1.       ]
[ 55.86143571  14.8492424    1.       ]
[ 57.27564928  13.43502884   1.       ]
[ 58.68986284  12.02081528   1.       ]
[ 60.1040764   10.60660172   1.       ]
```

```
[ 61.51828996   9.19238816   1.        ]
[ 62.93250353   7.77817459   1.        ]
[ 64.34671709   6.36396103   1.        ]
[ 65.76093065   4.94974747   1.        ]
[ 67.17514421   3.53553391   1.        ]
[ 68.58935778   2.12132034   1.        ]
[ 70.00357134   0.70710678   1.        ]
[ 71.4177849    0.70710678   1.        ]
[ 72.83199846   2.12132034   1.        ]
[ 74.24621202   3.53553391   1.        ]
[ 75.66042559   4.94974747   1.        ]
[ 77.07463915   6.36396103   1.        ]
[ 78.48885271   7.77817459   1.        ]
[ 79.90306627   9.19238816   1.        ]
[ 81.31727984  10.60660172   1.        ]
[ 82.7314934   12.02081528   1.        ]
[ 84.14570696  13.43502884   1.        ]
[ 85.55992052  14.8492424    1.        ]
[ 86.97413409  16.26345597   1.        ]
[ 88.38834765  17.67766953   1.        ]
[ 89.80256121  19.09188309   1.        ]
[ 91.21677477  20.50609665   1.        ]
[ 92.63098834  21.92031022   1.        ]
[ 94.0452019   23.33452378   1.        ]
[ 95.45941546  24.74873734   1.        ]
[ 96.87362902  26.1629509    1.        ]
[ 98.28784258  27.57716447   1.        ]
[ 99.70205615  28.99137803   1.        ]
[101.11626971  30.40559159   1.        ]
[102.53048327  31.81980515   1.        ]
[103.94469683  33.23401872   1.        ]
[105.3589104   34.64823228   1.        ]]


Process finished with exit code 0
```

以上代码结果中每个值的第一项和第二项分别代表到第一个聚类中心和第二个聚类中心的距离。

数据挖掘/机器学习

机器学习实战--k-均值聚类

1、聚类是一种无监督学习，他讲相似的对象放到同一簇下，有点像自动分类。聚类方法几乎可以用到任何对象上，簇内的对象越相似，聚类结果就越好。

2、K 均值聚类的优点

算法简单容易实现

缺点：

可能收敛到局部最小值，在大规模数据上收敛速度较慢

3、K-均值算法算法流程以及伪代码

首先随机选择 k 个初始点作为质心。然后将数据集中的每个点分配到一个簇中，具体来说，遍历数据集计算数据与质心之间的距离找到最小的距离将该数据划分到该簇内，完成后更新簇的质心。但是只要有数据点的簇发生变化就需要继续进行划分，直到没有点发生改变。

伪代码：

创建 K 个点作为起始质心（经常是随机选择，但是不能越界，下面的代码有具体的做法）

当任意一个点的簇分配结果发生变化时

对数据集中的每一个数据点

对每个质心

计算质心与数据点的距离

将数据点分配到距离最近的簇

对于每一个簇，计算簇中所有点的均值并将均值作为质心

```
from numpy import *

def loadDataSet(filename):
    dataMat=[]
    fr = open(filename)
    for line in fr.readlines():
        curLine = line.strip().split('\t')
        fltLine = list(map(float, curLine))
        #print(fltLine)
        dataMat.append(fltLine)
    return dataMat


def distEclud(vecA, vecB):
    return sqrt(sum(power(vecA - vecB, 2)))
```

```
def randCent(dataSet, k):
    n = shape(dataSet)[1]
    centroids = mat(zeros((k,n)))
    for j in range(n):
        minJ = min(dataSet[:, j])
        rangeJ = float(max(dataSet[:, j]) - minJ)
        centroids[:,j] = minJ + rangeJ * random.rand(k, 1)
    return centroids

def Kmeans(dataSet, k, distMeans = distEclud, createCent = randCent):
    m = shape(dataSet)[0]
    #建立一个 m*2 的矩阵位置 0 记录聚类的索引，位置 1 记录到聚类质心的距离
    clusterAssment = mat(zeros((m,2)))
    centroids = createCent(dataSet, k)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):
            minDist = inf; minIndex = -1
            for j in range(k):
                distJI = distMeans(centroids[j,:], dataSet[i,:])
                if distJI < minDist:
                    minDist = distJI; minIndex = j
            if clusterAssment[i, 0] != minIndex:
                clusterChanged = True
            clusterAssment[i, :] = minIndex, minDist**2
        #print(centroids)
        for cent in range(k):
            #通过数组过滤来获得给定簇的所有样本，nozero 返回一个 0/1 矩阵，然后取第一列的索引列
            ptsInClust = dataSet[nonzero(clusterAssment[:, 0].A == cent)[0]]
            centroids[cent, :] = mean(ptsInClust, axis = 0)
    return centroids, clusterAssment
#print(loadDataSet('testSet.txt'))
dataMat = mat(loadDataSet('testSet.txt'))
```

```
myCentroids, clustAssing = Kmeans(dataMat, 4)
print(myCentroids)
print(clustAssing)
```

4、使用后处理提高聚类的性能

K 均值算法中的 K 是用户给出的，如何衡量在给定 K 个聚类的情况下聚类的性能。对于 K 均值算法可能会收敛到局部最小，而不能收敛到全局最小，因此可能聚类效果不好。在上边的代码中 clusterAssment 保存了每个数据点的簇标号以及点到簇质心的误差（就是距离）。那么可以使用 SSE（sum of squared Erro，误差平方和）作为衡量聚类性能的指标，SSE 的数值越小就表示数据点越接近他们的质心聚类效果也就越好，同时由于对误差取了平方所以越远的点在 SSE 中就越受重视。

怎么降低 SSE？一种简单的方法是增加簇的数量，但是这样不好啊，你是去聚类了你弄了和数据集一样的簇，那还聚个啥。因此需要在不改变簇的数目的前提下降低 SSE。一种可行的办法是选择 SSE 最大的簇，然后对其继续进行 K 均值算法。同时为了保证簇的数目不变，可以将簇与簇进行合并。

合并的方法：质心之间距离最近的合并；合并两个簇然后计算总的 SSE，合并可以使 SSE 最小的簇。

5、二分 K 均值算法

为了克服 K 均值算法收敛于局部最小的问题，有人提出了一个二分 K 均值的算法。回来继续

分类: 机器学习实战

K 均值聚类

基本思想：通过迭代寻找 K 个簇的一种划分方法，使得聚类结果对应的代价函数最小。特别地，代价函数可以定义为各个样本距离所属聚类中心的误差平方和

$$J(c,\mu)=\sum_{i=1}^{M}||x_i-\mu_{c_i}||^2 J(c,\mu)=\sum_{i=1}^{M}||x_i-\mu_{c_i}||^2$$

具体步骤

- 数据预处理，如归一化、离群点处理等
- 随机选取 K 个簇中心，记为 $\mu_{(0)1},\cdots,u_{(0)k}\mu_1(0),\cdots,u_k(0)$
- 定义代价函数：$J(c,\mu)=min_{\mu}min_{c}\sum_{i=1}^{M}||x_i-\mu_{c_i}||^2 J(c,\mu)=min_{\mu}min_{c}\sum_{i=1}^{M}||x_i-\mu_{c_i}||^2$
- 令 $t=0,1,\cdots t=0,1,\cdots$为迭代步数，直到 JJ 收敛
  - 对于每一个样本 xixi，将其分配到最近的簇

$$c_{t_i}=argmin_k||x_i-\mu_{(t)k}||^2 c_{it}=arg min_k||x_i-\mu_{k(t)}||^2$$

  - 对于每一个类簇 k，重新计算该类簇的中心

$$\mu_{tk}=argmin_{\mu}\sum_{i:c_{(t)=k_i}}||x_i-\mu||^2 \mu_{kt}=arg min_{\mu}\sum_{i:c_i(t)=k}||x_i-\mu||^2$$

K 均值算法在迭代时，假设当前 JJ 没有达到最小值，那么首先固定类簇中心，调整每个样例所属类别来使 JJ 减小。然后固定样例所属类别，调整类簇中心使 JJ 减少。

优缺点

- 优点：可伸缩、高效，复杂度 $O(NKt)O(NKt)$
- 缺点：
  - 受初始值和离群点影响结果不稳定

- 局部最优解
- 数据簇分布差别特别大的情况（例如数量差距悬殊），需要手动设置 K 值
- 样本点只能划分到单一类中（可用模糊 C 均值或高斯混合模型）

算法调优

- 数据归一化和离群点处理：基于欧式距离度量的数据划分方法
- K 值的选择：手肘法、Gap Statistic 方法
- 核 K 均值算法：增加数据点线性可分的概率

改进模型

- K-means++算法：初始值选择改进
  - 假设已经选择了 n 个初始簇中心，则在选择第 n+1 个时，距离当前 n 个中心越远的点会有更高的概率被选中
- ISODATA 算法（迭代自组织数据分析法）
  - 当属于某个类别的样本数过少时，把该类别去掉
  - 当属于某个类别的样本数过多、分类程度较大时，分裂成两个子类别
  - 分裂操作、合并操作

收敛性证明

转换为概率模型，通过 EM 算法的收敛性得到证明

K-means 聚类算法

K-means 聚类算法（K-平均/K-均值算法）是最为经典也是使用最为广泛的一种基于**距离**的聚类算法。基于距离的聚类算法是指采用距离作为相似性量度的评价指标，也就是说当两个对象离得近时，两者之间的距离比较小，那么它们之间的相似性就比较大。

算法的主要思想是通过**迭代过程**把数据集划分为不同的类别，使得评价聚类性能的准则函数达到最优，从而使生成的**每个聚类（又称簇）紧凑**且独立。

K-means 聚类算法的缺点：对于**离群点**是敏感的，一个很大极端值的数据对象可能会显著地扭曲数据的分布。

常见的相似度/距离评价准则有：

- 欧几里得距离

其意义就是两个元素在欧氏空间中的集合距离，因为其直观易懂且可解释性强，被广泛用于标识两个标量元素的相异度。

$$d(X,Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2}$$

- 曼哈顿距离

$$d(X,Y) = |x_1 - y_1| + |x_2 - y_2| + ... + |x_n - y_n|$$

- 闵可夫斯基距离

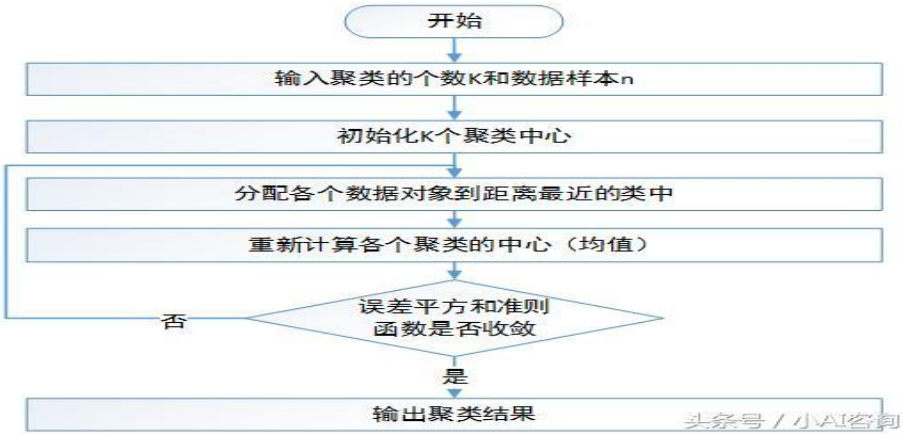
$$d(X,Y) = \sqrt[p]{|x_1 - y_1|^p + |x_2 - y_2|^p + \ldots + |x_n - y_n|^p}$$

聚类性能评价准则：

K-means 聚类算法通常使用误差平方和准则函数（等同于欧几里得距离）来评价聚类性能。给定数据集 X，其中只包含描述属性，不包含类别属性。假设 X 包含 K 个聚类子集 X1,X2,…XK；各个聚类子集中的样本数量分别为 n1，n2,…,nk;各个聚类子集的均值代表点（也称聚类中心）分别为 m1，m2,…,mk。

- 误差平方和准则函数公式

$$E = \sum_{i=1}^{k} \sum_{p \in X_i} \left\| p - m_i \right\|^2$$



K-means 聚类算法实例

初始数据集，共 5 条记录，每条数据记录包含两个属性 x 和 y。

O	x	y
1	0	2
2	0	0
3	1.5	0
4	5	0
5	5	2

作为一个聚类分析的二维样本，要求的簇的数量 K=2。

(1)选择 $O_1(0,2)$，$O_2(0,0)$ 为初始的簇中心，

即 $M_1=O_1=(0,2)$，$M_2=O_2=(0,0)$。

(2)对剩余的每个对象，根据其与各个簇中心的距离，将它赋给最近的簇。

对 O_3：$d(M_1,O_3)=\sqrt{(0-1.5)^2+(2-0)^2}=2.5$

$d(M_2,O_3)=\sqrt{(0-1.5)^2+(0-0)^2}=1.5$

显然 $d(M_2,O_3)\le d(M_1,O_3)$，故将 O_3 分配给 C_2

- 对于 O_4：$d(M_1,O_4)=\sqrt{(0-5)^2+(2-0)^2}=\sqrt{29}$

  $d(M_2,O_4)=\sqrt{(0-5)^2+(0-0)^2}=5$
- 因为 $d(M_2,O_4)\le d(M_1,O_4)$ 所以将 O_4分配给 C_2
- 对于 O_5：$d(M_1,O_5)=\sqrt{(0-5)^2+(2-2)^2}=5$

  $d(M_2,O_5)=\sqrt{(0-5)^2+(0-2)^2}=\sqrt{29}$
- 因为 $d(M_1,O_5)\le d(M_2,O_5)$ 所以将 O_5分配给 C_1
- 更新，得到新簇 $C_1=\{O_1,O_5\}$ 和 $C_2=\{O_2,O_3,O_4\}$
- 计算平方误差准则，单个方差为

$E_1=\left[(0-0)^2+(2-2)^2\right]+\left[(0-5)^2+(2-2)^2\right]=25$　$M_1=O_1=(0,2)$

$E_2=27.25$　$M_2=O_2=(0,0)$

总体平均方差是：$E=E_1+E_2=25+27.25=52.25$

（3）计算新的簇的中心。

$M_1=((0+5)/2,(2+2)/2)=(2.5,2)$

$M_2=((0+1.5+5)/3,(0+0+0)/3)=(2.17,0)$

重复（2）和（3），得到O_1分配给C_1；O_2分配给C_2，O_3分配给C_2，O_4分配给C_2，O_5分配给C_1。更新，得到新簇 $C_1=\{O_1,O_5\}$ 和 $C_2=\{O_2,O_3,O_4\}$。中心为 $M_1=(2.5,2)$，$M_2=(2.17,0)$。

单个方差分别为

$E_1=\left[(0-2.5)^2+(2-2)^2\right]+\left[(2.5-5)^2+(2-2)^2\right]=12.5$　$E_2=13.15$

总体平均误差是：$E=E_1+E_2=12.5+13.15=25.65$

由上可以看出，第一次迭代后，总体平均误差值**52.25~25.65**，显著减小。由于在两次迭代中，簇中心不变，所以停止迭代过程，算法停止。

学习参考：

https://www.cnblogs.com/leoo2sk/archive/2010/09/20/k-means.html#!comments 示例计算有误，思路没问题。

https://www.toutiao.com/i6451161136644489742/加了对数据[0,1]规格化处理的环节，中心思想还是一样的。

https://blog.csdn.net/leaf_zizi/article/details/82684921 文本聚类，可以做舆情信息汇总，分词软件 rost cm6。

https://www.toutiao.com/i6452271711302713870/K-medoids 聚类，K-means 基础上的改良。对小数据集有效。

分类: 机器学习算法

K-均值聚类算法

聚类是一种无监督的学习，它将相似的对象归到同一个簇中。
K-均值算法将数据点归为 K 个簇，每个簇的质心采用簇中所含数据点的均值构成。
K-均值算法的工作流程：首先随机确定 K 个初始点为质心，然后将数据集中的每个点非配到一个簇中，分配原则是分给距离最近的质心所在的簇。然后每个簇的质心更新为该簇所有数据点的平均值。
伪代码：

```
随机创建 K 个质心：
        当任意一个点的簇分配结果发生改变时：
                对数据集中的每个数据点：
                        对每个质心：
                                计算质心到数据点的距离
                        将数据点分配给距离最近的质心所在簇
                对每一个簇，计算簇中所有数据点的均值作为质心（更新质心）
```

K-均值聚类支持函数：加载文件、欧式距离计算、随机初始化质心

```
from  numpy  import  *
def  loadDataSet(filename):
        dataSet=[]
        f=open(filename)
        for  line  in  f.readlines():
                curLine=line.strip().split('\t')
                floatLine=list(map(float,curLine))
                dataSet.append(floatLine)
        return  dataSet
def  distEclud(vecA,vecB):
        return  sqrt(sum(power(vecA-vecB,2)))
```

```
def  randCent(dataSet,k):
        n=shape(dataSet)[1]
        centroids=mat(zeros((k,n)))
        for  i  in  range(n):
                minI=min(dataSet[:,i])
                rangeI=float(max(dataSet[:,i])-minI)
                centroids[:,i]=minI+rangeI*random.rand(k,1)
        return  centroids
```

K-均值聚类算法的结束条件是数据点的簇分配结果不再改变。即达到局部最优。

```
def  kMeans(dataSet, k, distMeans  =distEclud, createCent  =  randCent):
        m  =  shape(dataSet)[0]
        clusterAssment  =  mat(zeros((m,2)))
        centroids  =  createCent(dataSet, k)
        clusterChanged  =  True
        while  clusterChanged:
                clusterChanged  =  False
                for  i  in  range(m):
                        minDist  =  inf
                        minIndex  =  -1
                        for  j  in  range(k):
                                distJI  =  distMeans(centroids[j,:], dataSet[i,:])
                                if  distJI < minDist:
                                        minDist  =  distJI
                                        minIndex  =  j
                        if  clusterAssment[i,0] !=  minIndex:
                                clusterChanged  =  True
                        clusterAssment[i,:]  =  minIndex,minDist**2             #有这种用法，更
新数据点 i 的分类簇和误差
                # print(centroids)           #显示质心的迭代过程
                for  cent  in  range(k):
                        ptsInClust  =  dataSet[nonzero(clusterAssment[:,0].A  ==  cent)[0]]  #分
类为 cent 簇的数据点重新构成新的簇
```

```
                centroids[cent,:] = mean(ptsInClust, axis = 0)    #将新的簇的均值作为新的质点
        return centroids, clusterAssment
```

簇分配结果矩阵 clusterAssment 包含两列：第一列记录数据所属的簇索引值，第二列存储误差。误差即为欧式距离的平方。

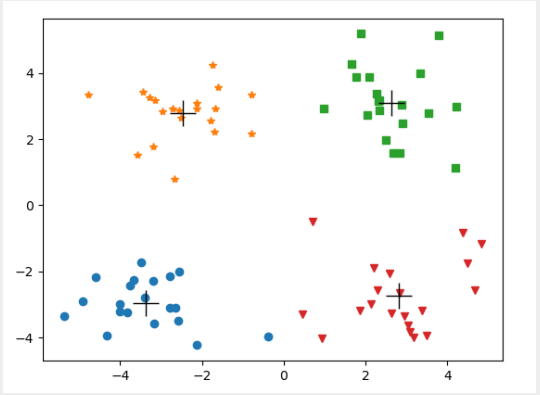
绘制数据点及聚类结果：

```
def plotPoint(dataMat,myCent,clustAssment):
        dataMat=array(dataMat)
        myCent=array(myCent)
        import matplotlib.pyplot as plt
        mark=['o','*','s','v','h']
        for cent in range(shape(myCent)[0]):
                typeCent=dataMat[nonzero(clustAssment[:,0].A==cent)[0]]
                plt.plot(typeCent[:,0],typeCent[:,1],mark[cent])
        plt.plot(myCent[:,0],myCent[:,1],'k+',markersize=20)    #标记散点形状和颜色的前后顺序任意
        plt.show()
```

聚类结果测试：

```
if __name__=='__main__':
        dataMat = mat(loadDataSet('testSet.txt'))
        myCent, clustAssing = kMeans(dataMat, 4)
        plotPoint(dataMat, myCent, clustAssing)
```

输出：



K-均值聚类的参数 K 是用户定义的，然而用户无法知道恰当 的 K 值，并且此算法只能达到局部最优，无法达到全局最优。
一种用于度量聚类效果的指标是 SSE(Sum of Squared Error，误差平方和)：对应于 clusterAssment 第二列的总和。SSE 值越小表示数据点越接近质心，聚类效果越好。聚类的目标是在保持簇数目不变的情况下提高簇的质量。
K-均值算法的改进：使用后处理来提高聚类性能。将具有最大 SSE 值得簇划分为两个簇。具体方法是将 SSE 最大的簇包含的点用数组过滤的方法过滤出来形成新的小数据集，在此小数据集上运行 K-均值算法将其一分为二。
二分 K-均值聚类算法：首先将所有数据点作为一个簇，然后将该簇一分为二。划分原则有两种：
1、选择对其划分可以最大程度降低 SEE 值的簇进行划分；
2、选择 SSE 最大的簇进行划分。
选择对其划分可以最大程度降低 SEE 值的簇进行划分：

```
def  bitKmeans(dataSet, k, distMeas=distEclud):
        m  =  shape(dataSet)[0]
        clusterAssment  =  mat(zeros((m,2)))
        centroid0  =  mean(dataSet, axis=0).tolist()[0]      #初始化为一簇
        centList  =[centroid0]      #簇列表，初始化只有一簇
        for  j  in  range(m):
                clusterAssment[j,1]  =  distMeas(mat(centroid0), dataSet[j,:])**2          #初始化分类簇的误差
        while  (len(centList) < k):
                lowestSSE  =  inf
                for  i  in  range(len(centList)):
                        ptsInCurrCluster  =  dataSet[nonzero(clusterAssment[:,0].A==i)[0],:]    #属于当前簇的所有数据点组成
的小数据集
                        centroidMat, splitClustAss  =  kMeans(ptsInCurrCluster,  2, distMeas)    #将小数据集二分
                        sseSplit  =  sum(splitClustAss[:,1])    #小数据集二分后的总误差
                        sseNotSplit  =  sum(clusterAssment[nonzero(clusterAssment[:,0].A!=i)[0],1])      #剩余数据点的总误
差
                        print  ("用于分割的数据集的误差：%s ,剩余数据集的总误差:%s "%(sseSplit,sseNotSplit))
                        if  (sseSplit  +  sseNotSplit) < lowestSSE:        #全部数据集的误差等于用于分割的数据集的误差加上
剩余数据集的误差
                                bestCentToSplit  =  i
                                bestNewCents  =  centroidMat    #更新质点
                                bestClustAss  =  splitClustAss.copy()          #更新分类簇信息
                                lowestSSE  =  sseSplit  +  sseNotSplit    #更新误差
```

```
                bestClustAss[nonzero(bestClustAss[:,0].A  ==  1)[0],0]  =  len(centList)         #如果将当前小数据集进行分割的话，划分出两个簇，标签分别是原有的 i 和质点的总数
                bestClustAss[nonzero(bestClustAss[:,0].A  ==  0)[0],0]  =  bestCentToSplit     #因为目前质点列表并没有及时更新，所以质点的长度为索引值加一
                print  ('最好的分割簇索引值: ',bestCentToSplit)
                print  ('被分割的数据点个数: ',  len(bestClustAss))
                centList[bestCentToSplit]  =  bestNewCents[0,:].tolist()[0]       #bestNewCents 2*n 的矩阵         被更新的那一行质点信息更新为 bestNewCents 的第 0 行
                centList.append(bestNewCents[1,:].tolist()[0])     #质心列表长度加一，新增的是 bestNewCents 的第 1 行信息
                clusterAssment[nonzero(clusterAssment[:,0].A  ==  bestCentToSplit)[0],:]=  bestClustAss         #更新簇信息，待更新的是原本分类簇为 i 的那些数据点
        return  mat(centList), clusterAssment
```

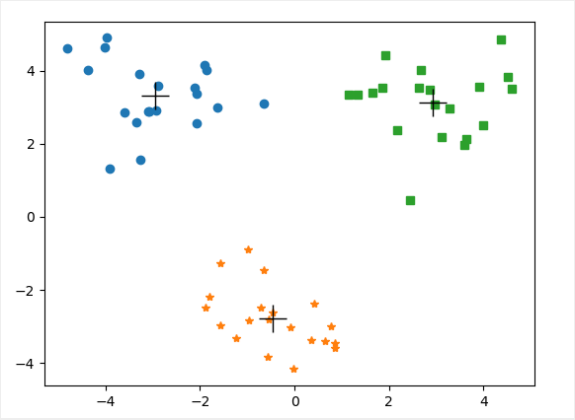
测试：

```
if  __name__=='__main__':
        dataMat  =  mat(loadDataSet('testSet2.txt'))
        myCent, clustAssing  =  bitKmeans(dataMat,  3)
        plotPoint(dataMat, myCent, clustAssing)
```

输出：

```
用于分割的数据集的误差：453.0334895807502  ,剩余数据集的总误差:0.0
最好的分割簇索引值:    0
被分割的数据点个数:    60
用于分割的数据集的误差：77.59224931775066  ,剩余数据集的总误差:29.15724944412535
用于分割的数据集的误差：12.753263136887313  ,剩余数据集的总误差:423.8762401366249
最好的分割簇索引值:    0
被分割的数据点个数:    40
```

因为只有 3 簇，所以划分两次就可以了。



对地图上的点进行聚类：没有使用 API，直接从 places.txt 中获取各俱乐部的经纬度信息。

```
def distSLC(vecA, vecB):    # 返回地球表面两点之间的距离
        a = sin(vecA[0, 1] * pi / 180) * sin(vecB[0, 1] * pi / 180)
        b = cos(vecA[0, 1] * pi / 180) * cos(vecB[0, 1] * pi / 180) * cos(pi * (vecB[0, 0] - vecA[0,
  0]) / 180)
        return arccos(a + b) * 6371.0

import matplotlib
import matplotlib.pyplot as plt

def clusterClubs(numClust=5):    # 将俱乐部进行聚类并画出结果
        datList = []
        for line in open('places.txt').readlines():
                lineArr = line.split('\t')
                datList.append([float(lineArr[4]), float(lineArr[3])])
        datMat = mat(datList)
        myCentroids, clustAssing = bitKmeans(datMat, numClust, distMeas=distSLC)
        fig = plt.figure()
        rect = [0.1, 0.1, 0.8, 0.8]
        scatterMarkers = ['s', 'o', '^', '8', 'p','d', 'v', 'h', '>', '<']
        axprops = dict(xticks=[], yticks=[])
```

```
        ax0 = fig.add_axes(rect, label='ax0', **axprops)
        imgP = plt.imread('Portland.png')    # imread 基于图像创建矩阵
        ax0.imshow(imgP)
        ax1 = fig.add_axes(rect, label='ax1', frameon=False)
        for i in range(numClust):
            ptsInCurrCluster = datMat[nonzero(clustAssing[:, 0].A == i)[0], :]
            markerStyle = scatterMarkers[i % len(scatterMarkers)]    # 使用索引来选择标记形状
            ax1.scatter(ptsInCurrCluster[:, 0].flatten().A[0],ptsInCurrCluster[:, 1].flatten().A[0],marker=markerSt
yle, s=90)
        ax1.scatter(myCentroids[:, 0].flatten().A[0],myCentroids[:, 1].flatten().A[0], marker='+',color='black',s=300)
        plt.show()
```

测试：

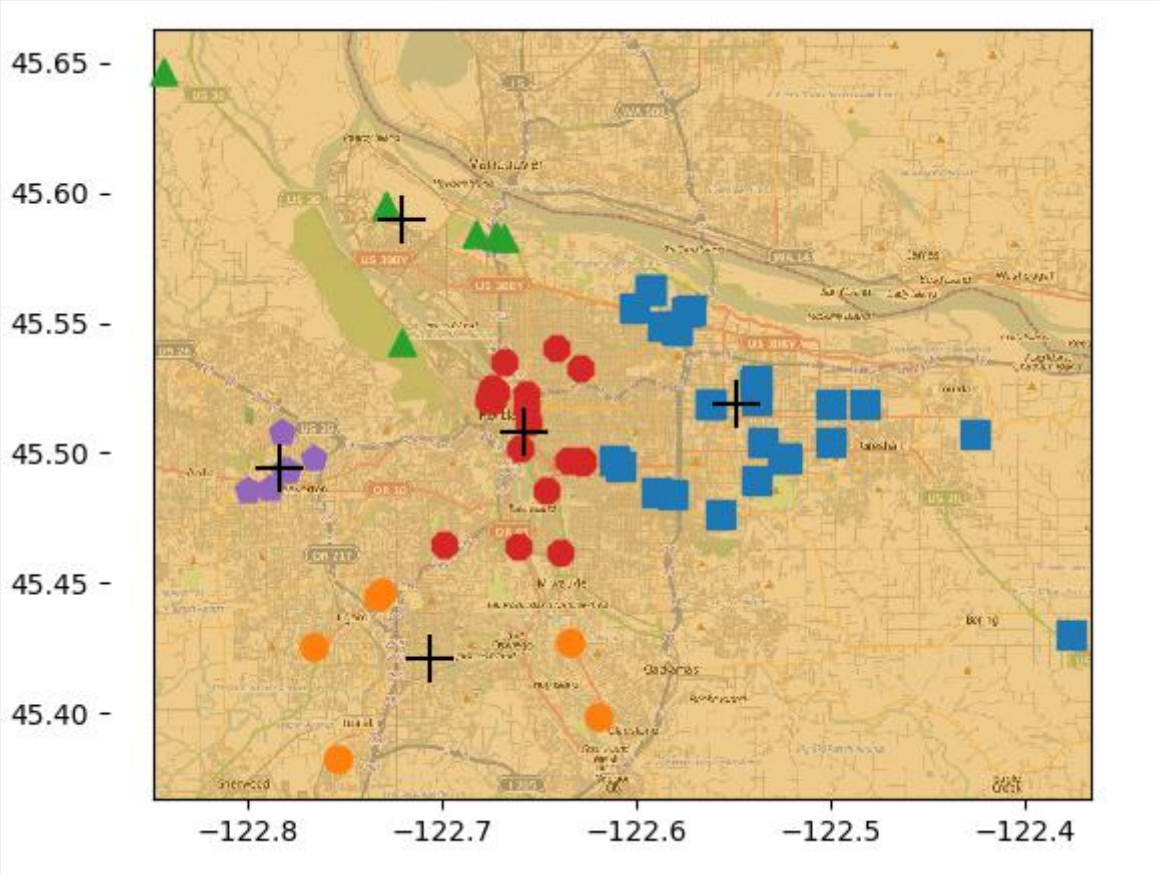
```
if __name__=='__main__':
    clusterClubs(5)
```

输出：

```
用于分割的数据集的误差：3436.254083558642 ,剩余数据集的总误差:0.0
最好的分割簇索引值:  0
被分割的数据点个数:  69
用于分割的数据集的误差：689.8706952996852 ,剩余数据集的总误差:2312.355075698593
用于分割的数据集的误差：1163.3644904835776 ,剩余数据集的总误差:1123.899007860049
最好的分割簇索引值:  1
被分割的数据点个数:  52
用于分割的数据集的误差：701.3190736813707 ,剩余数据集的总误差:1163.3644904835776
用于分割的数据集的误差：451.92459095764076 ,剩余数据集的总误差:1523.8240844582879
用于分割的数据集的误差：329.2991930704607 ,剩余数据集的总误差:1887.3384217453877
最好的分割簇索引值:  0
被分割的数据点个数:  17
用于分割的数据集的误差：18.792573676179693 ,剩余数据集的总误差:1690.9230046725252
用于分割的数据集的误差：463.6732476050007 ,剩余数据集的总误差:1101.2441502796096
用于分割的数据集的误差：197.3863640526132 ,剩余数据集的总误差:1464.7584875667094
```

```
16  用于分割的数据集的误差：253.98662218778765  ,剩余数据集的总误差:1337.1250499760008
17  最好的分割簇索引值:     1
18  被分割的数据点个数:     34
```

5 个簇，需要划分 4 次。



二分 K-均值聚类算法内部在将某个小数据集一分为二时调用了 K-均值算法，K-均值算法又调用了随机产生质心的 randCent 方法，造成了算法的不确定性，多次运行程序，发现聚类效果并不是每次都好。

重点在于如何在不引入随机的条件下对数据集初始化出两个簇。