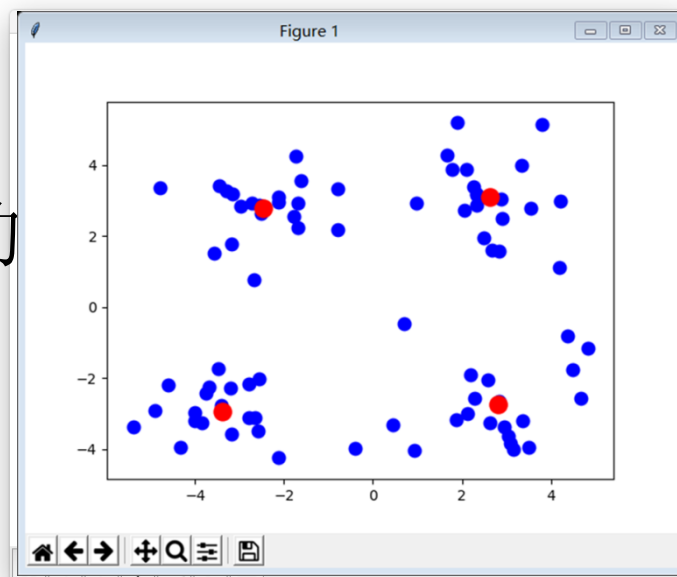


k-means算法

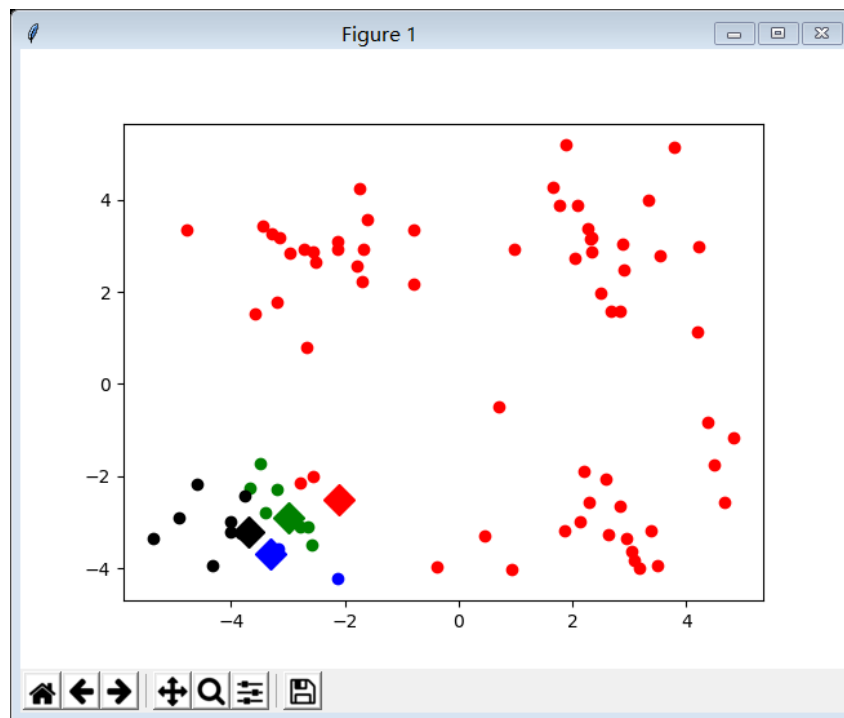
K-means(K-均值)算法是什么

- K-means(K-均值)算法是聚类的一种，在1967年由MacQueen提出。
- 为什么叫K-均值？
- K个簇(cluster)；
- 每个簇内数据点坐标的均值构成质心。

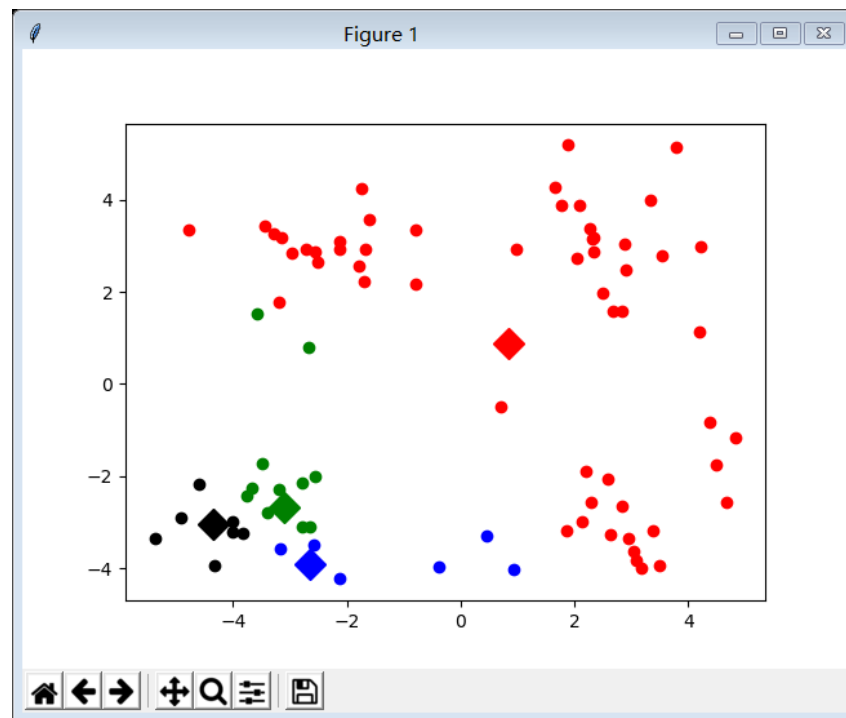


K-均值算法的工作流程

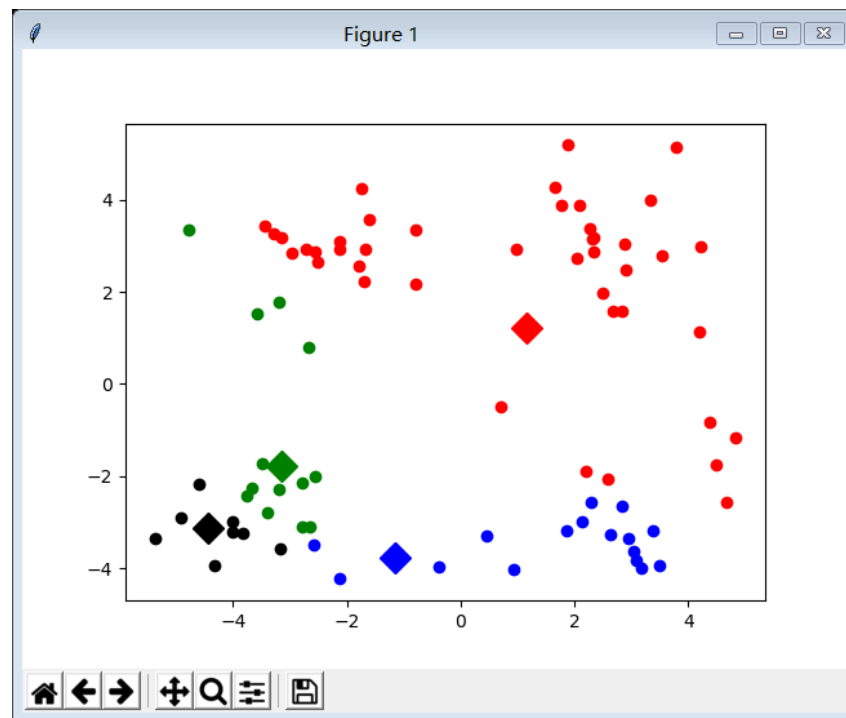
随机选择初始质心



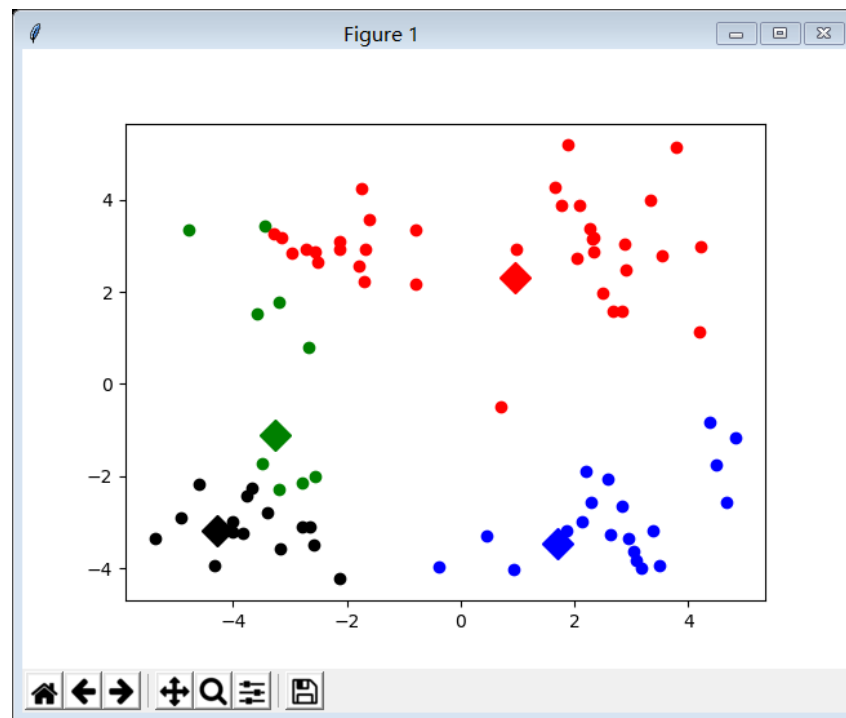
质心位置更新



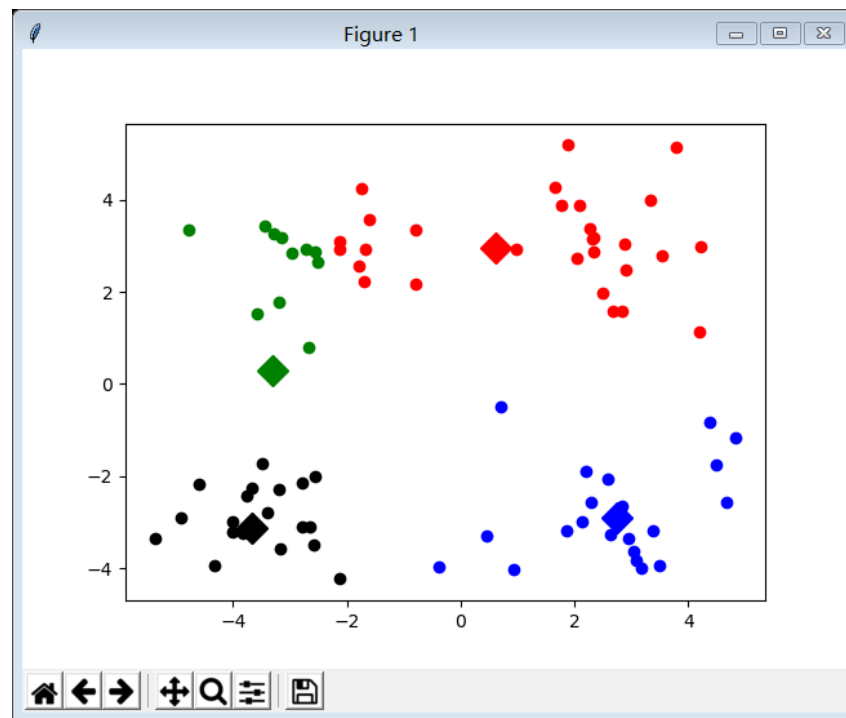
重新分簇



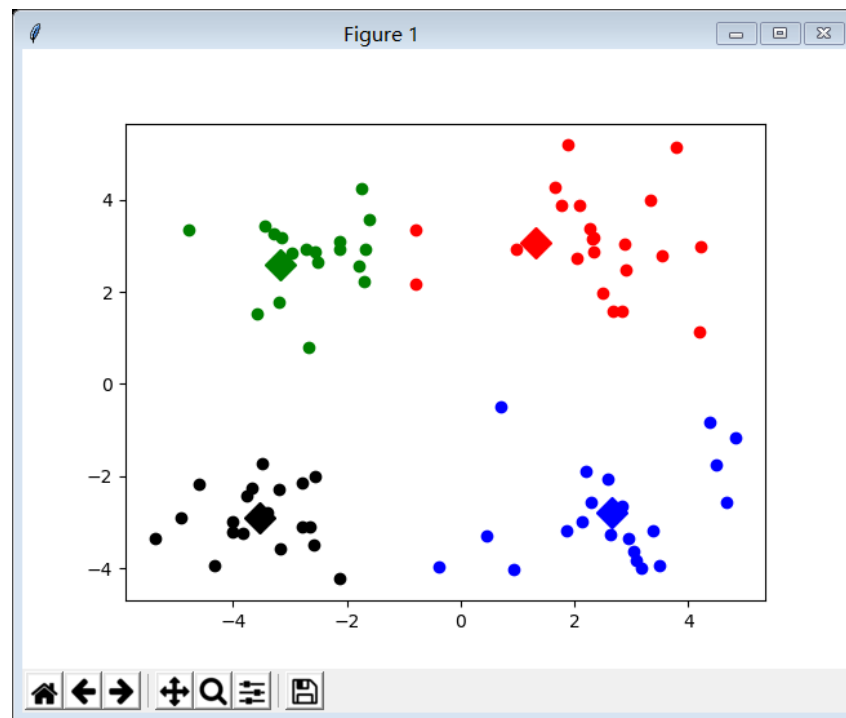
更新质心位置



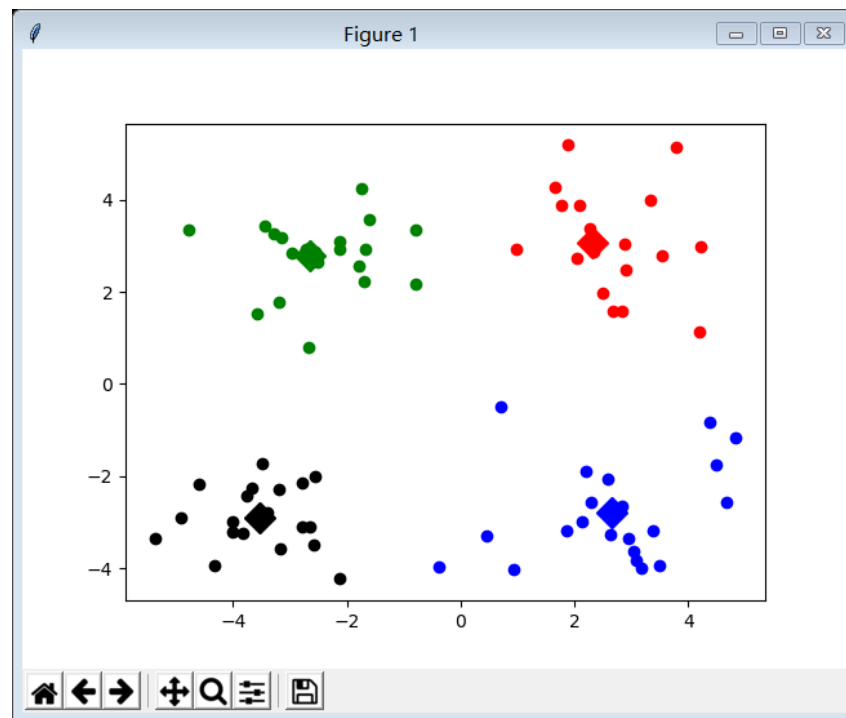
重新分簇



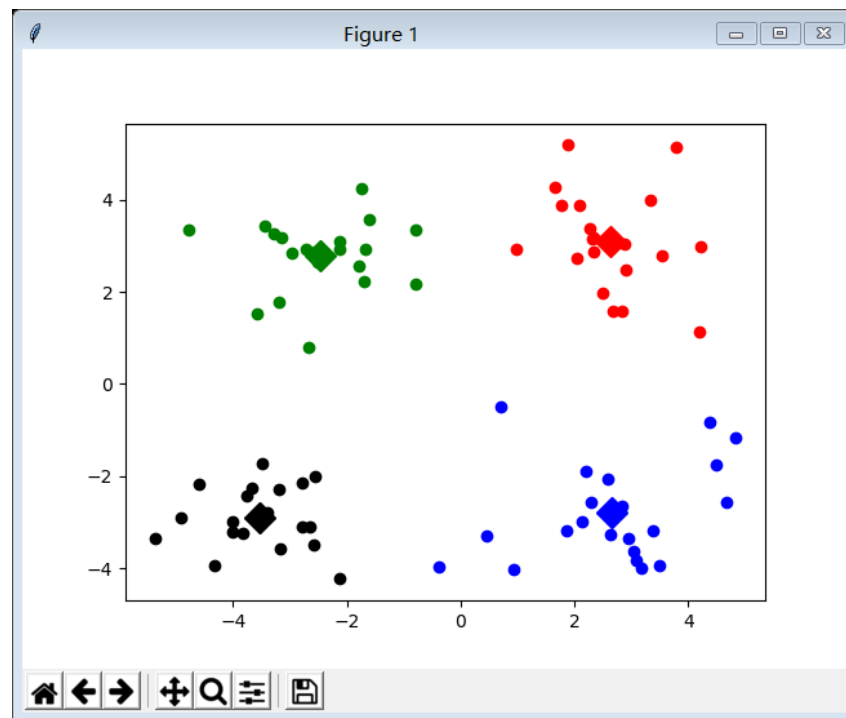
更新质心位置



重新分簇



聚类完成



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

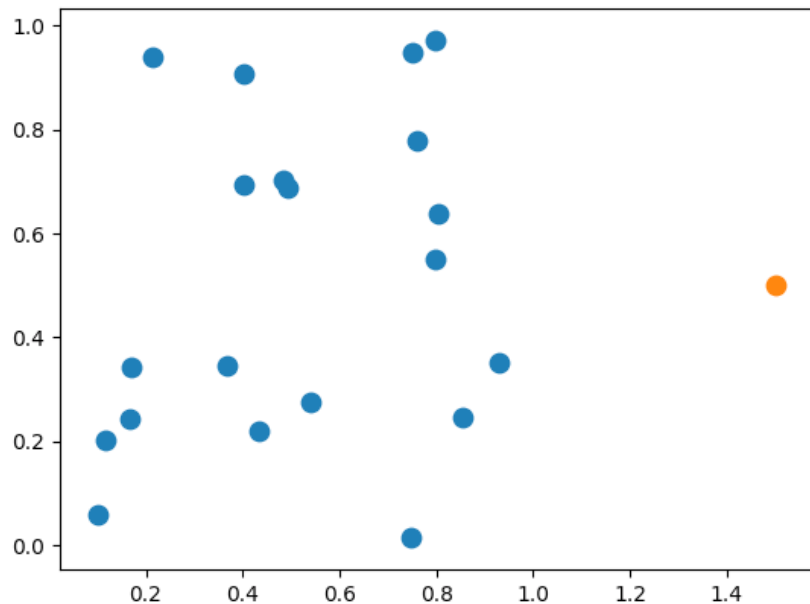
如何度量聚类效果及保证收敛？

- K-均值聚类一般用误差平方和（Sum of the Squared Error, SSE）作为聚类的目标函数来度量聚类效果。
- 对于k-means聚类来说，可以将每个簇的质心点看作这个簇的代表，一个样本点距离哪个质心点更近，它就属于哪个簇。既然如此，一个样本点与它所属的簇的质心点之间的距离就可以认为是这个样本点的误差，所有样本点的误差之和可以代表聚类的质量，这个和越小，说明每一个簇内的点聚的越紧密。

如何度量聚类效果及保证收敛？

- 但由于误差计算比较不方便(比如以欧式距离表示误差，计算欧式距离需要先求平方和再开方)，所以在k-means中让每个误差先平方再求所有误差的平方的和，以这个和来表示聚类效果，称为误差平方和。
- 可以证明在每一次的迭代过程中，SSE都是减小的，所以最后算法一定可以收敛。
- 但由于SSE是非凸函数，所以聚类可能收敛到局部最小值而非全局值。

每次迭代SSE均减小的证明



假设有 n 个数据点，坐标分别为 $(a_1, b_1), (a_2, b_2), (a_3, b_3) \dots (a_n, b_n)$ ，在二维平面上任取一点坐标为 (x, y) ，计算其到所有数据点的距离平方和可以得到：

$$SSE = (a_1 - x)^2 + (b_1 - y)^2 + (a_2 - x)^2 + (b_2 - y)^2 + \dots$$

每次迭代SSE均减小的证明

$$\begin{aligned} SSE &= \left[(a_1 - x)^2 + (a_2 - x)^2 + \cdots (a_n - x)^2 \right] + \left[(b_1 - y)^2 + (b_2 - y)^2 + \cdots (b_n - x)^2 \right] \\ &= \left[nx^2 - 2(a_1 + a_2 + \cdots + a_n)x + (a_1^2 + a_2^2 + \cdots + a_n^2) \right] \\ &\quad + \left[ny^2 - 2(b_1 + b_2 + \cdots + b_n)x + (b_1^2 + b_2^2 + \cdots + b_n^2) \right] \end{aligned}$$

- 当 $x = \frac{a_1 + a_2 + \cdots + a_n}{n}$ 时，第一个中括号取得最小值。
- 当 $y = \frac{b_1 + b_2 + \cdots + b_n}{n}$ 时，第二个中括号取得最小值

每次迭代SSE均减小的证明

- 所以当质心处于所有点的坐标均值处时，SSE最小，即k-means流程中，每次质心位置移动都会使得SSE减小。
- 在k-means的重新分簇步骤中，簇归属不变的点不会影响SSE，而簇归属改变的点会被划分到离自己更近的质心，即误差会减小，所以在这一步中SSE也是减小的。
- 所以在k-means的执行流程中，每一步SSE都是在减小的。

K-均值算法优点

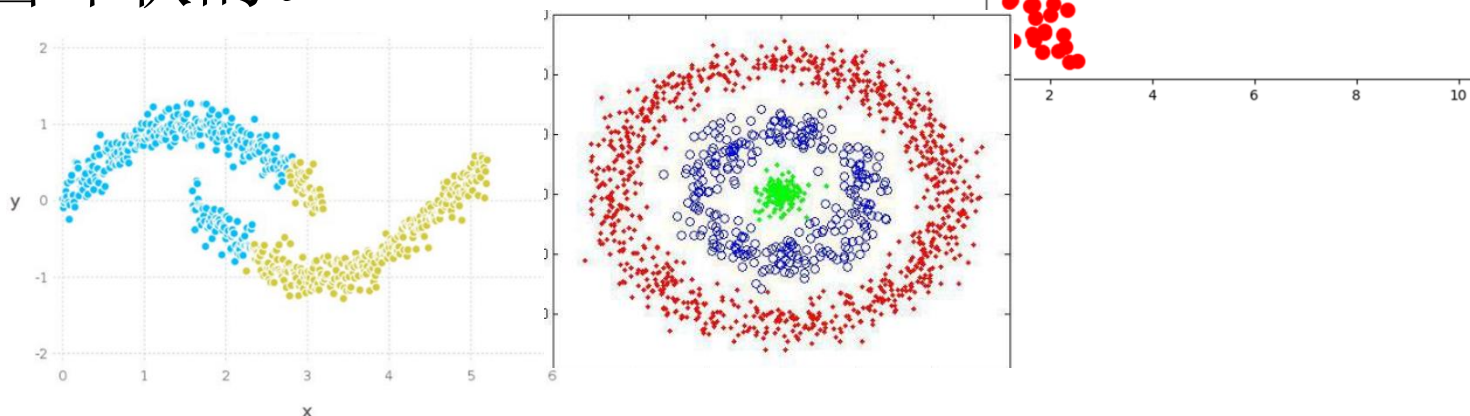
1. 算法简单，容易理解

2. 时间复杂度近于线性，而且适合挖掘大规模数据集。K-均值算法的时间复杂度是 $O(mktn)$, 其中 n 代表数据集中对象的数量， k 代表着簇的数目。 t 代表着算法迭代的次数， m 代表维度

3. 适合高维数据

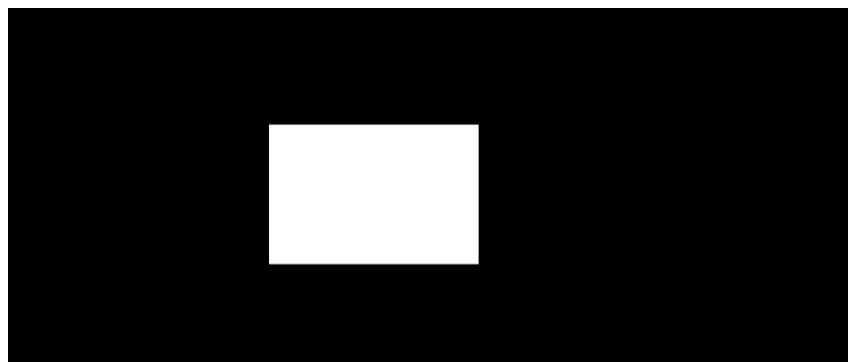
K-均值算法缺点

- 1.K值需要预先给定；
- 2.对初始质心位置敏感，选取不当易造成收敛于局部最小值；
- 3.对噪声点和离群点敏感；
- 4.不适合非球状簇，也不适合圆环状的。



图像压缩算法概述

- 对于一张非黑即白的图片来说，每个像素点需要1个bit进行存储。



- 对于一张灰度图来说，一般每个像素点需要8个bit，即1个字节进行存储(灰度值从0到255)。



图像压缩算法概述

- 红、绿、蓝称为光的三原色(红黄青是美术画图中的三原色，与计算机中用的三原色不同，计算机处理图像用的是光的三原色RGB)。
- 对于一张三通道的彩色图像来说，每个像素点需要存储红、绿、蓝三种颜色。



图像压缩算法概述

- 彩色图像中，每种原色的变化范围有多种形式，可以从0到255，也可以从0到63等，一般用0到255。
- 以三种原色都从0到255变化为例，则每个像素点可以有 $256 * 256 * 256 = 16777216$ 种颜色，通常也被简称为1600万色、千万色或1600万真彩色。 $256 * 256 * 256 = 2^{24}$ ，所以也称为24位色。
- 24位色每个像素点需要24个比特，即三个字节进行存储。一张1200万像素的图片需要大约36兆的空间。
- 大量的图片使得存储、传输和处理都极为困难。

思考

- 假设一张1200万像素的彩色三通道图像，每个像素点的颜色没有 2^{24} 种，只有256种，这张图片需要多少空间进行存储？

扩充阅读：把24位色图转化为灰度图

- 假设原来的图片每个像素点颜色为(R,G,B)，则可以按如下公式计算灰度值：

$$\text{Grey} = (R * 30 + G * 59 + B * 11) / 100$$

- 然后将每个像素点的像素值均改为对应的灰度值。

```
from PIL import Image
```

```
im = Image.open('.\\trump.png')
for i in range(im.size[0]):
    for j in range(im.size[1]):
        r,g,b,a = im.getpixel((i,j))
        grey = int((r * 30 + g * 59 + b * 11)/100)
        im.putpixel((i,j), (grey, grey, grey))
im.show()
```

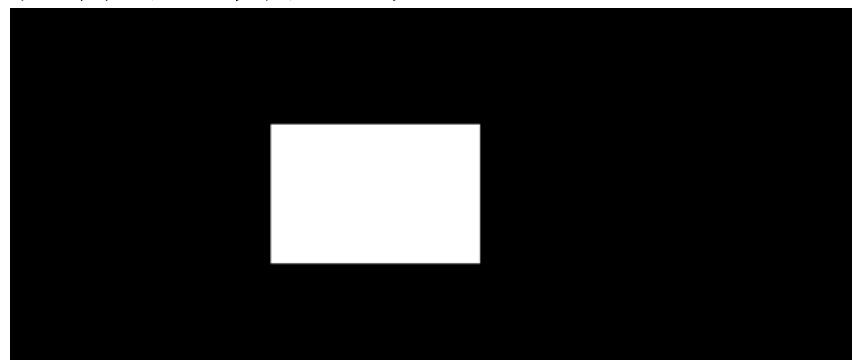
```
from PIL import Image
```

```
im = Image.open('.\\trump.png')
im = im.convert('L')
im.show()
```

图像压缩算法概述

- 图像数据之所以能被压缩，是因为用户可以容忍一定的图像失真以及数据中存在着冗余。
- 假设有一张纯黑的图片，由你手写记录其像素值，你会记成这种形式吗：

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



图像有损压缩算法

- 有损压缩是利用了人类对图像中的某些成分不敏感的特性，允许压缩过程中损失一定的信息；虽然不能完全恢复原始数据，但是所损失的部分对理解原始图像的影响较小，却换来了更大的压缩比(即压缩后得到的图像比无损压缩算法得到的图像所占用的空间更小)。
- 理论上人的眼睛大约能分辨**1000**万种颜色，但实际上一些相近的颜色很难区分。比如下图像素值分别为(100,150,120)和(100,145,120)，实际感觉差距并不大。



图像有损压缩算法

- 图像有损压缩算法主要有以下几种：

预测编码

变换编码

模型基编码

分形编码

预测编码

- 预测编码是根据离散信号之间存在着一定关联性的特点，利用前面一个或多个信号预测下一个信号进行，然后对实际值和预测值的差（预测误差）进行编码。如果预测比较准确，误差就会很小。在同等精度要求的条件下，就可以用比较少的比特进行编码，达到压缩数据的目的。
- 对于图像压缩来说就是利用一个像素点的像素值预测其周围像素点的像素值或者在连续的画面中(比如电影)，利用上一帧的像素值预测下一帧的像素值。

变换编码

- 变换编码不直接对图像进行编码，而是首先将空域图像信号映射变换到另一个正交矢量空间，产生一批变换系数，然后对这些变换系数进行编码处理。变换编码是一种间接编码方法，其中关键问题是在时域或空域描述时，数据之间相关性大，数据冗余度大，经过变换在变换域中描述，数据相关性大大减少，数据冗余量减少，参数独立，数据量少，这样再进行量化，编码就能得到较大的压缩比。典型的准最佳变换有DCT（离散余弦变换）、DFT(离散傅里叶变换)、WHT（Walsh Hadama 变换）、HrT(Haar 变换)等。其中，最常用的是离散余弦变换。

模型基编码

- 模型基编码的编码的原理为：首先，在编、解码两端分别建立相同的模型。在发送端，利用图像分析模块对输入图像提取景物的参数，如形状参数、运动参数等。在接收端，景物的这些参数被编码后通过信道传输到解码端，由解码器接收到的参数利用图像合成技术在重建图像，基本原理如图1-1所示。

分形编码





- 分形的方法是把一幅数字图像，通过一些图像处理技术将原始图像分成一些子图像，然后在分形集中查找这样的子图像。分形集存储许多迭代函数，通过迭代函数的反复迭代，可以恢复原来的子图像。

一个简单的图像有损压缩算法

- 图像有损压缩中最简单、最好理解的就是直接减少颜色数量。比如windows附件的画图中可以将图片保存为以下格式。

单色位图 (*.bmp;*.dib)
16色位图 (*.bmp;*.dib)
256 色位图 (*.bmp;*.dib)
24位位图 (*.bmp;*.dib)
JPEG (*.jpg;*.jpeg;*.jpe;*.jfif)
GIF (*.gif)
TIFF (*.tif;*.tiff)
PNG (*.png)

- 同一图片保存为不同的格式所占用的空间分别为

 trump16色位图.bmp	2018/9/17 15:02	BMP 图像	249 KB
 trump24位位图.bmp	2018/9/17 15:03	BMP 图像	1,488 KB
 trump256色位图.bmp	2018/9/17 15:02	BMP 图像	498 KB
 trump单色位图.bmp	2018/9/17 15:01	BMP 图像	63 KB

图像有损压缩算法

- 不同颜色数量的图片效果:

单色图



16色位图



256色位图



24位位图



图像有损压缩算法

- 原图像为:

图像	
分辨率	950 x 534
宽度	950 像素
高度	534 像素
位深度	24

- $950 \times 534 \times 3 = 1521900$, 存储图像需要1521900个字节的空间。

大小: 1.45 MB (1,523,022 字节)

占用空间: 1.45 MB (1,523,712 字节)

k-means将图压缩为256色



原图

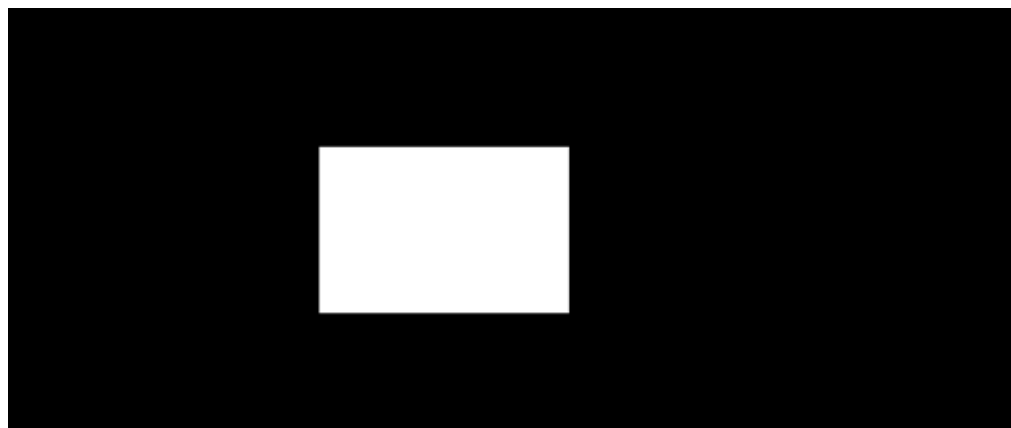


附件画图保存

k-means聚类结果

图像无损压缩算法

- 无损压缩是利用数据的统计冗余进行压缩，可完全恢复原始数据而不引起任何失真，但压缩率是受到数据统计冗余度的理论限制，一般为2:1到5:1。



图像无损压缩算法

- 常用的图像无损压缩算法有：

香农-范诺算法(Shannon-Fano coding)算法

哈夫曼编码(Huffman Coding)，也叫霍夫曼编码

游程长度编码(run-length code)

LZW(Lempel-Ziv-Welch)编码

算术编码

哈夫曼编码

- 哈夫曼编码(Huffman Coding)，又称霍夫曼编码，是一种编码方式，哈夫曼编码是可变字长编码(VLC)的一种。Huffman于1952年提出一种编码方法，该方法完全依据字符出现概率来构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫做Huffman编码（有时也称为霍夫曼编码）。

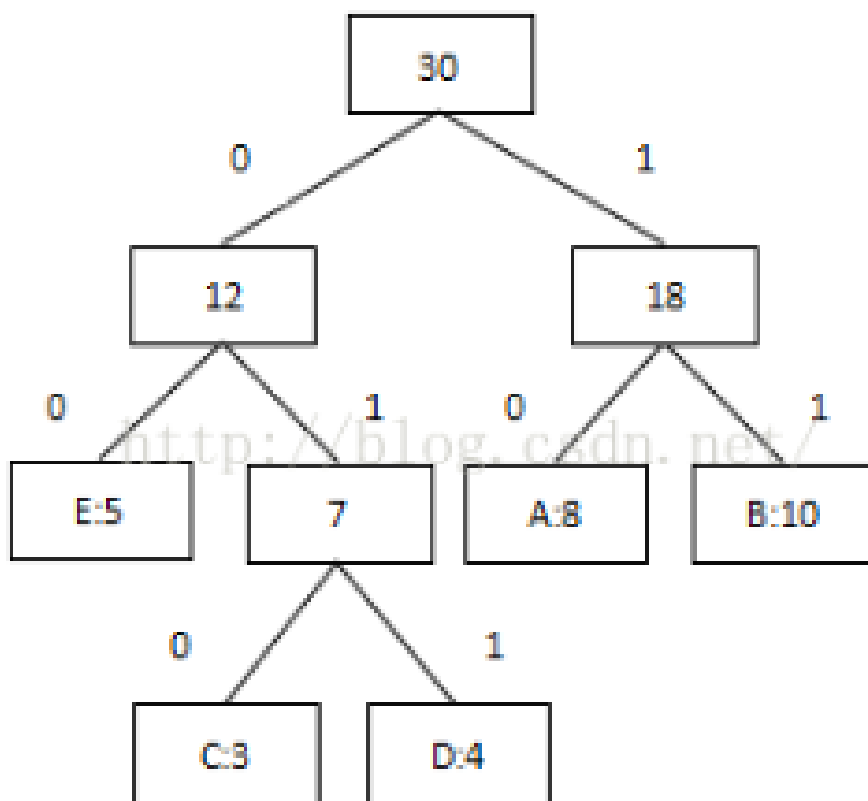
哈夫曼编码实例

- 现有一个由5个不同符号组成的30个符号的字符串：
BABACACADADABBCBABEBE DDABEEEEBB，首先计算各个字符出现的次数，得到

字符	次数
A	8
B	10
C	3
D	4
E	5

哈夫曼编码实例

- 把出现次数（概率）最小的两个相加，并作为左右子树，重复此过程，直到根节点



哈夫曼编码实例

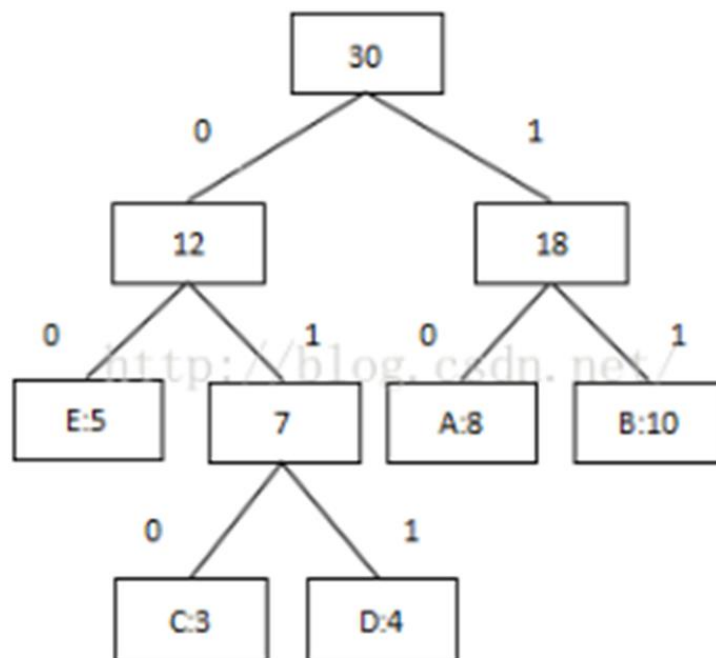
- 沿二叉树顶部到每个字符路径，获得每个符号的编码

字符	路径(编码)
A	10
B	11
C	010
D	011
E	00

- 根据编码原字符串可以化为1110111001010.....
- 在原来的编码中，每个符号均需要3个bit进行存储，而编码后A、B、D需要两个bit，C、D仍然需要3个。

思考

- 解码时会不会出现冲突，比如编码后得到100100101010，无法确定第一个字符是10还是100或者是1001。



游程长度编码

- 游程编码是一种比较简单的压缩算法，其基本思想是将重复且连续出现多次的字符使用（连续出现次数，某个字符）来描述。
- 比如对于字符串：“11111000000011100”就可以通过游程长度编码压缩为(5,1)(6,0)(3,1)(2,0)，原来16个数字压缩为8个数字表示，而且数据可完全恢复。

练习

- 有两个字符串，分别为
“111111100000011111000000” (7个1, 6个0, 5个1, 6个0)和“1010101010101010101010” (12个10), 以游程长度编码进行压缩, 看看能压缩多少。
- 如果有一个字符串为“8989898901010101”, 是否可以通过游程长度编码进行压缩, 如果不行, 有没有什么办法近似达到这一目的。

k-means与图像压缩

- 游程长度编码适合数据内拥有大量重复序列的数据，而对于不合适的数据，有可能越压缩后占用空间反而更大。
- 利用k-means可以将图像内像素值相似的点聚为一个簇，并取其均值，人为地造成图像内有大量重复的序列，然后再进行压缩就会得到较大的压缩比。