# 参考1

[opencv K均值聚类(python)\_python opencv 聚类-CSDN博客](https://blog.csdn.net/first_bug/article/details/123514692)

# opencv K均值聚类(python)

#### K均值聚类

* 预测的是一个离散值时，做的工作就是“分类”。
* 预测的是一个连续值时，做的工作就是“回归”。

机器学习模型还可以将训练集中的数据划分为若干个组，每个组被称为一个“簇（cluster）”。这种学习方式被称为“聚类（clusting）”，它的重要特点是在学习过程中不需要用标签对训练样本进行标注。也就是说，学习过程能够根据现有训练集自动完成分类（聚类）。

根据训练数据是否有标签，可以将学习划分为监督学习和无监督学习。

*K近邻、支持向量机都是监督学习，提供有标签的数据给算法学习，然后对数据分类*

*聚类是无监督学习，事先并不知道分类标签是什么，直接对数据分类。*

聚类能够将具有相似属性的对象划分到同一个集合（簇）中。

聚类方法能够应用于所有对象，簇内的对象越相似，聚类算法的效果越好。

##### K均值聚类的基本步骤

K均值聚类是一种将输入数据划分为k个簇的简单的聚类算法，该算法不断提取当前分类的中心点（也称为质心或重心），并最终在分类稳定时完成聚类。

从本质上说，K均值聚类是一种迭代算法。

在实际处理过程中需要进行多轮的迭代，直到分组稳定不再发生变化，即可认为分组完成。

**K均值聚类算法的基本步骤如下：**

1. 随机选取k个点作为分类的中心点。
2. 将每个数据点放到距离它最近的中心点所在的类中。
3. 重新计算各个分类的数据点的平均值，将该平均值作为新的分类中心点。
4. 重复步骤2和步骤3，直到分类稳定。

*可以是随机选取k个点作为分类的中心点，也可以是随机生成k个并不存在于原始数据中的数据点作为分类中心点。*

距离最近: 要进行某种形式的距离计算。（在具体实现时，可以根据需要采用不同形式的距离度量方法。）

##### K均值聚类模块

OpenCV提供了函数cv2.kmeans()来实现K均值聚类。

该函数的语法格式为：

retval, bestLabels, centers=cv2.kmeans(data, K, bestLabels, criteria, attempts,

flags)

* data：输入的待处理数据集合，应该是np.float32类型，每个特征放在单独的一列中。
* K：要分出的簇的个数，即分类的数目，最常见的是K=2，表示二分类。
* bestLabels：表示计算之后各个数据点的最终分类标签（索引）。实际调用时，参数bestLabels的值设置为None。
* criteria：算法迭代的终止条件。当达到最大循环数目或者指定的精度阈值时，算法停止继续分类迭代计算。该参数由3个子参数构成，分别为type、max\_iter和eps。
  + type表示终止的类型，可以是三种情况
    - cv2.TERM\_CRITERIA\_EPS：精度满足eps时，停止迭代。
    - cv2.TERM\_CRITERIA\_MAX\_ITER：迭代次数超过阈值max\_iter时，停止迭代。
    - cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER：上述两个条件中的任意一个满足时，停止迭代。
  + max\_iter：最大迭代次数。
  + eps：精确度的阈值。
* attempts：在具体实现时，为了获得最佳分类效果，可能需要使用不同的初始分类值进行多次尝试。指定attempts的值，可以让算法使用不同的初始值进行多次（attempts次）尝试。
* flags：表示选择初始中心点的方法，主要有以下3种。
  + cv2.KMEANS\_RANDOM\_CENTERS：随机选取中心点。
  + cv2.KMEANS\_PP\_CENTERS：基于中心化算法选取中心点。
  + cv2.KMEANS\_USE\_INITIAL\_LABELS：使用用户输入的数据作为第一次分类中心点；如果算法需要尝试多次（attempts 值大于1时），后续尝试都是使用随机值或者半随机值作为第一次分类中心点。
* retval：距离值（也称密度值或紧密度），返回 每个点到相应中心点距离的平方和(是一个数)。
* bestLabels：各个数据点的最终分类标签（索引）。
* centers：每个分类的中心点数据。

##### 简单例子

**例1：**

随机生成一组数据，使用函数cv2.kmeans()对其分类。

* 一组数据在[0,50]区间
* 另一组数据在[200,250]区间
* 使用函数cv2.kmeans()对它们分类。

主要步骤如下：

1. 数据预处理

使用随机函数随机生成两组数据，并将它们转换为函数cv2.kmeans()可以处理的格式。

1. 设置参数

设置函数cv2.kmeans()的参数形式。将参数criteria的值设置为“(cv2.TERM\_CRITERIA\_EPS+ cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)”，在达到一定次数或者满足一定精度时终止迭代。

1. 调用函数cv2.kmeans()

调用函数cv2.kmeans()，获取返回值，用于后续步骤的操作。

1. 确定分类

根据函数cv2.kmeans()返回的标签（“0”和“1”），将原始数据分为两组

1. 显示结果

绘制经过分类的数据及中心点，观察分类结果。

完整程序：

import numpy as np

import cv2

from matplotlib import pyplot as plt

# 随机生成两组数组

# 生成60个值在[0,50]内的数据

num1 = np.random.randint(0,50,60)

# 生成60个值在[200,250]内的数据

num2 = np.random.randint(200,250,60)

# 组合数据为num

num = np.hstack((num1, num2))

# 使用reshape函数将其转换为(120,1)

num = num.reshape((120,1)) #每个数据为1列

# 转换为float32类型

num = np.float32(num)

# 调用kmeans模块

# 设置参数criteria的值

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

# 设置参数flags的值

flags = cv2.KMEANS\_RANDOM\_CENTERS

# 调用函数kmeans

retval, bestLabels, centers = cv2.kmeans(num,2, None, criteria,10, flags)

# 打印返回值

print(retval)

print(bestLabels)

print(centers)

# 获取分类结果

n1 = num[bestLabels==0]

n2 = num[bestLabels==1]

# 绘制分类结果

# 绘制原始数据

plt.plot(np.ones(len(n1)),n1,'ro')

plt.plot(np.ones(len(n2)),n2,'bo')

# 绘制中心点

#plt.plot([1],centers[0],'rx')

#plt.plot([1],centers[1],'bx')

plt.show()

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**例2：**

有两种物体：

* 物体1的长和宽都在 [0,20] 内
* 物体2的长和宽都在[40,60] 内

使用随机数模拟两种物体的长度和宽度，并使用函数cv2.kmeans()对它们分类。

根据题目要求，主要步骤如下：

1. 随机生成数据，并将它们转换为函数cv2.kmeans()可以处理的形式。
2. 设置函数cv2.kmeans()的参数形式。
3. 调用函数cv2.kmeans()。
4. 根据函数cv2.kmeans()的返回值，确定分类结果。
5. 绘制经过分类的数据及中心点，观察分类结果。

import numpy as np

import cv2

from matplotlib import pyplot as plt

# 随机生成两组数值

#长和宽都在[0,20]内

m1 = np.random.randint(0,20, (30,2))

#长和宽的大小都在[40,60]

m2 = np.random.randint(40,60, (30,2))

# 组合数据

m = np.vstack((m1, m2))

# 转换为float32类型

m = np.float32(m)

# 调用kmeans模块

# 设置参数criteria值

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

# 调用kmeans函数

ret, label, center=cv2.kmeans(m,2, None, criteria,10, cv2.KMEANS\_RANDOM\_CENTERS)

'''

#打印返回值

print(ret)

print(label)

print(center)

'''

# 根据kmeans的处理结果，将数据分类，两大类

res1 = m[label.ravel()==0]

res2 = m[label.ravel()==1]

# 绘制分类结果数据及中心点

plt.scatter(res1[:,0], res1[:,1], c = 'g', marker = 's')

plt.scatter(res2[:,0], res2[:,1], c = 'r', marker = 'o')

plt.scatter(center[0,0], center[0,1], s = 200, c = 'b', marker = 'o')

plt.scatter(center[1,0], center[1,1], s = 200, c = 'b', marker = 's')

plt.xlabel('Height'), plt.ylabel('Width')

plt.show()

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**例3：**

使用函数cv2.kmeans()将灰度图像处理为只有两个灰度级的二值图像。

需要对灰度图像内的色彩进行分类，将所有的像素点划分为两类。然后，用这两类的中心点像素值替代原有像素值，满足题目的要求。

主要步骤如下：

1. 图像预处理

读取图像，并将图像转换为函数cv2.kmeans()可以处理的形式。

*在读取图像时，如果是3个通道的RGB图像，需要将图像的RGB值处理为一个单独的特征值。具体实现时，用函数cv2.reshape()完成对图像特征值的调整。*

为了满足函数cv2.kmeans()的要求，需要将图像的数据类型转换为numpy.float32类型。

1. 设置函数cv2.kmeans()的参数形式

设置参数criteria的值为“(cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)”，让函数cv2.kmeans()在达到一定精度或者达到一定迭代次数时，即停止迭代。

设置参数K的值为2，将所有像素划分为两类。

1. 调用函数cv2.kmeans()

调用函数cv2.kmeans()，得到距离值、分类中心点和分类标签，用于后续操作。

1. 值替换

将像素点的值替换为当前分类的中心点的像素值。

1. 显示变换前后的图像

分别显示原始图像和二值化图像。

import numpy as np

import cv2

import matplotlib.pyplot as plt

# 读取待处理图像

img = cv2.imread('./hand2.jpg')

# 使用reshape将一个像素点的RGB值作为一个单元处理

data = img.reshape((-1,3)) # n行 3列

# 转换为kmeans可以处理的类型

data = np.float32(data)

# 调用kmeans模块

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

K =2

ret, label, center=cv2.kmeans(data, K, None, criteria,10, cv2.KMEANS\_RANDOM\_CENTERS)

# 转换为uint8数据类型，将每个像素点都赋值为当前分类的中心点像素值

# 将center的值转换为uint8

center = np.uint8(center)

# 使用center内的值替换原像素点的值

res1 = center[label.flatten()] # 根据索引来取值，最后结果的大小同索引的大小

# 使用reshape调整替换后的图像

res2 = res1.reshape((img.shape))

# 显示处理结果

plt.subplot(121)

plt.imshow(img[:,:,::-1])

plt.axis('off')

plt.subplot(122)

plt.imshow(res2[:,:,::-1])

plt.axis('off')

plt.show()

*调整程序中的K值，就能改变图像的显示结果。例如，K=8，则可以让图像显示8个灰度级。*

# 参考2

[OpenCV学习笔记(十七)——K均值聚类\_cv2.kmeans-CSDN博客](https://blog.csdn.net/cyj972628089/article/details/122033850)

# OpenCV学习笔记(十七)——K均值聚类

*当我们要预测的是一个离散值时，做的工作就是“分类”。机器学习模型还可以将训练集中的数据划分为若干个组，每个组被称为一个“簇（cluster）”。它的重要特点是在学习过程中不需要用标签对训练样本进行标注。也就是说，学习过程能够根据现有训练集自动完成分类（聚类）。*

根据训练数据是否有标签，我们可以将学习划分为监督学习和无监督学习。前面介绍的K近邻、支持向量机都是监督学习，提供有标签的数据给算法学习，然后对数据分类。而聚类是无监督学习，事先并不知道分类标签是什么，直接对数据分类。

#### 1. 理论基础

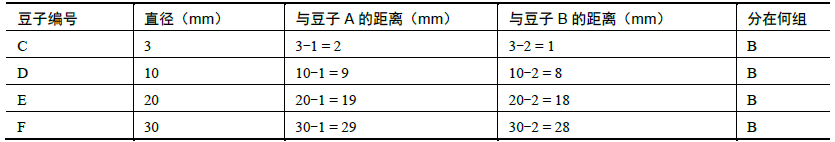
本节首先用一个实例来介绍K 均值聚类的基本原理，在此基础上介绍K 均值聚类的基本步骤，最后介绍一个二维空间下的K 均值聚类示例。

##### 1.1 分豆子

假设有 6 粒豆子混在一起，我们可以在不知道这些豆子类别的情况下，将它们按照直径大小划分为两类。

**第一步**：经过测量，以mm（毫米）为单位，这些豆子的直径大小分别为1、2、3、10、20、30。下面将它们标记为A、B、C、D、E、F，并进行分类操作。

**第二步**：计算每粒豆子的直径距离豆子A 和豆子B 的距离。距离哪个豆子更近，就将新豆子划分在哪个豆子所在的组。



在本步骤结束时，6 粒豆子被划分为以下两组。

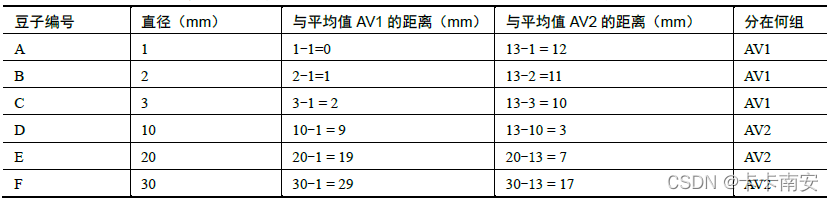
* 第 1 组：只有豆子A。
* 第 2 组：豆子B、C、D、E、F，共5 粒豆子。

**第三步**：分别计算第1 组豆子和第2 组豆子的直径平均值。然后，将各个豆子按照与直径平均值的距离大小分组。

* 计算第 1 组豆子的平均值AV1 = 1mm。
* 计算第 2 组豆子的平均值AV2 = (2+3+10+20+30)/5 = 13mm。

得到上述平均值以后，对所有的豆子再次分组：

* 将平均值AV1 所在的组，标记为AV1 组。
* 将平均值AV2 所在的组，标记为AV2 组。计算各粒豆子距离平均值AV1 和AV2 的距离，并确定分组，

  
距离平均值AV1 更近的豆子，就被划分为AV1 组；距离平均值AV2 更近的豆子，就被划分为AV2 组。现在，6 粒豆子的分组情况为：

* AV1 组：豆子A、豆子B、豆子C。
* AV2 组：豆子D、豆子E、豆子F。

**第四步**：重复第3 步，直到分组稳定不再发生变化，即可认为分组完成。

在本例中，重新计算AV1 组的平均值AV41、AV2 组的平均值AV42，依次计算每个豆子与平均值AV41 和AV42 的距离，并根据该距离重新划分分组。按照与第3 步相同的方法，重新计算平均值并分组后，6 粒豆子的分组情况为：

* AV41 组：豆子A、豆子B、豆子C。
* AV42 组：豆子D、豆子E、豆子F。

与上一次的分组相比，并未发生变化，我们就认为分组完成了。我们将直径较小的那一组称为“小豆子”，直径较大的那一组称为“大豆子”。

当然，本例是比较极端的例子，数据很快就实现了收敛，在实际处理中可能需要进行多轮的迭代才能实现数据的收敛，分类不再发生变化。

##### 1.2 K均值聚类的基本步骤

K 均值聚类是一种将输入数据划分为k 个簇的简单的聚类算法，该算法不断提取当前分类的中心点（也称为质心或重心），并最终在分类稳定时完成聚类。从本质上说，K 均值聚类是一种迭代算法。

**K 均值聚类算法的基本步骤如下**：

1. 随机选取k 个点作为分类的中心点。
2. 将每个数据点放到距离它最近的中心点所在的类中。
3. 重新计算各个分类的数据点的平均值，将该平均值作为新的分类中心点。
4. 重复步骤2 和步骤3，直到分类稳定。

在第 1 步中，可以是随机选取k 个点作为分类的中心点，也可以是随机生成k 个并不存在于原始数据中的数据点作为分类中心点。

在第 3 步中，提到的“距离最近”，说明要进行某种形式的距离计算。在具体实现时，可以根据需要采用不同形式的距离度量方法。当然，不同的计算方法会对算法的性能产生影响。

#### 2. K均值聚类模块

retval, bestLabels, centers=cv2.kmeans(data, K, bestLabels, criteria, attempts, flags)

式中各个参数的含义为：

* data：输入的待处理数据集合，应该是np.float32 类型，每个特征放在单独的一列中。
* K：要分出的簇的个数，即分类的数目，最常见的是K=2，表示二分类。
* bestLabels：表示计算之后各个数据点的最终分类标签（索引）。实际调用时，参数bestLabels 的值设置为None。
* criteria：算法迭代的终止条件。当达到最大循环数目或者指定的精度阈值时，算法停止继续分类迭代计算。该参数由3 个子参数构成，分别为type、max\_iter 和eps。  
  type 表示终止的类型，可以是三种情况，分别为：  
  · cv2.TERM\_CRITERIA\_EPS：精度满足eps 时，停止迭代。  
  · cv2.TERM\_CRITERIA\_MAX\_ITER：迭代次数超过阈值max\_iter 时，停止迭代。  
  · cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER：上述两个条件中的  
  任意一个满足时，停止迭代。
* max\_iter：最大迭代次数。
* eps：精确度的阈值。
* attempts：在具体实现时，为了获得最佳分类效果，可能需要使用不同的初始分类值进行多次尝试。指定attempts 的值，可以让算法使用不同的初始值进行多次（attempts 次）尝试。
* flags：表示选择初始中心点的方法，主要有以下3 种。  
  · cv2.KMEANS\_RANDOM\_CENTERS：随机选取中心点。  
  · cv2.KMEANS\_PP\_CENTERS：基于中心化算法选取中心点。  
  · cv2.KMEANS\_USE\_INITIAL\_LABELS：使用用户输入的数据作为第一次分类中心点；  
  如果算法需要尝试多次（attempts 值大于1 时），后续尝试都是使用随机值或者半随机值作为第一次分类中心点。

返回值的含义为：

* retval：距离值（也称密度值或紧密度），返回每个点到相应中心点距离的平方和。
* bestLabels：各个数据点的最终分类标签（索引）。
* centers：每个分类的中心点数据。

#### 3. 简单示例

##### 3.1 随机生成一组数据，使用函数cv2.kmeans()对其分类

为了方便理解，假设有两种豆子，其中一种是“xiaoMI”，另外一种是“daMI”。它们的直径不一样，xiaoMI 的直径在[0, 50]区间；daMI 的直径在[200, 250]区间。用随机数模拟两种豆子的直径，并使用函数cv2.kmeans()对它们分类。

主要步骤如下：

1. **数据预处理**

使用随机函数随机生成两组豆子的直径数据，并将它们转换为函数cv2.kmeans()可以处理的格式。

1. **设置参数**

设置函数 cv2.kmeans()的参数形式。将参数criteria 的值设置为“(cv2.TERM\_CRITERIA\_EPS+ cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)”，在达到一定次数或者满足一定精度时终止迭代。

1. **调用函数cv2.kmeans()**

调用函数cv2.kmeans()，获取返回值，用于后续步骤的操作。

1. **确定分类**

根据函数cv2.kmeans()返回的标签（“0”和“1”），将原始数据分为两组。

1. **显示结果**

绘制经过分类的数据及中心点，观察分类结果。

根据上述分析，编写代码如下：

import numpy as np

import cv2

from matplotlib import pyplot as plt

# 随机生成两组数组

# 生成60 个值在[0,50]内的xiaoMI 直径数据

xiaoMI = np.random.randint(0,50,60)

# 生成60 个值在[200,250]内的daMI 直径数据

daMI = np.random.randint(200,250,60)

# 将xiaoMI 和daMI 组合为MI

MI = np.hstack((xiaoMI,daMI))

# 使用reshape 函数将其转换为(120,1)

MI = MI.reshape((120,1))

# 将MI 转换为float32 类型

MI = np.float32(MI)

# 调用kmeans 模块

# 设置参数criteria 的值

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

# 设置参数flags 的值

flags = cv2.KMEANS\_RANDOM\_CENTERS

# 调用函数kmeans

retval,bestLabels,centers = cv2.kmeans(MI,2,None,criteria,10,flags)

'''

# 打印返回值

print(retval)

print(bestLabels)

print(centers)

'''

# 获取分类结果

XM = MI[bestLabels==0]

DM = MI[bestLabels==1]

# 绘制分类结果

# 绘制原始数据

plt.plot(XM,'ro')

plt.plot(DM,'bo')

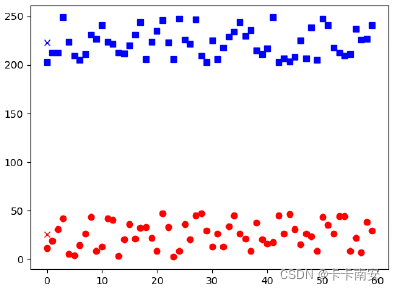
# 绘制中心点

plt.plot(centers[0],'rx')

plt.plot(centers[1],'bx')

plt.show()

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

  
在图中，上面的小方块是标签为“0”的数据点，下方的圆点是标签为“1”的数据点。上方的“x”标记是标签为“0”的数据组的中心点，其值大概在225 左右；下方的“x”是标签为“1”的数据组的中心点，其值大概在25 左右。

##### 3.2 有一堆米粒，按照长度和宽度对它们分类

为了方便理解，假设米粒有两种，其中一种是XM，另外一种是DM。它们的直径不一样，XM 的长和宽都在[0, 20]内，DM 的长和宽都在[40, 60]内。使用随机数模拟两种米粒的长度和宽度，并使用函数cv2.kmeans()对它们分类。

根据题目要求，主要步骤如下：  
（1）随机生成两组米粒的数据，并将它们转换为函数cv2.kmeans()可以处理的形式。  
（2）设置函数cv2.kmeans()的参数形式。  
（3）调用函数cv2.kmeans()。  
（4）根据函数cv2.kmeans()的返回值，确定分类结果。  
（5）绘制经过分类的数据及中心点，观察分类结果。

import numpy as np

import cv2

from matplotlib import pyplot as plt

#随机生成两组数值

#xiaomi组,长和宽的大小都在[0,20]

xiaomi = np.random.randint(0,20,(30,2))

#dami组,长和宽的大小都在[40,60]

dami = np.random.randint(40,60,(30,2))

#组合数据

MI = np.vstack((xiaomi,dami))

np.random.shuffle(MI)

# 转换为float32类型

MI = np.float32(MI)

# 调用kmeans模块

#设置参数criteria值

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

#调用kmeans函数

ret,label,center=cv2.kmeans(MI,2,None,criteria,10,cv2.KMEANS\_RANDOM\_CENTERS)

'''

#打印返回值

print(ret)

print(label)

print(center)

'''

# 根据kmeans处理结果，将数据分类，分为XM和DM两大类

XM = MI[label.ravel()==0]

DM = MI[label.ravel()==1]

# 绘制分类结果数据及中心点

plt.scatter(XM[:,0],XM[:,1],c = 'g', marker = 's')

plt.scatter(DM[:,0],DM[:,1],c = 'r', marker = 'o')

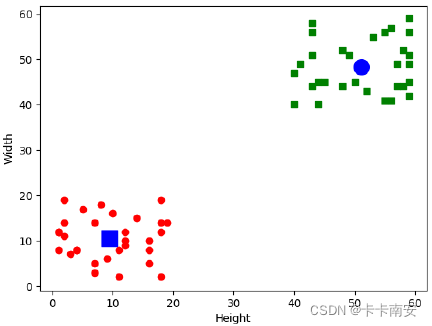
plt.scatter(center[0,0],center[0,1],s = 200,c = 'b', marker = 'o')

plt.scatter(center[1,0],center[1,1],s = 200,c = 'b', marker = 's')

plt.xlabel('Height'),plt.ylabel('Width')

plt.show()

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

  
在图中，右上方的小方块是标签为“0”的数据点，左下方的圆点是标签为“1”的数据点。右上方稍大的圆点是标签“0”的数据组的中心点；左下方稍大的方块是标签为“1”的数据组的中心点。

##### 3.3 使用函数cv2.kmeans()将灰度图像处理为只有两个灰度级的二值图像

根据题目要求，需要对灰度图像内的色彩进行分类，将所有的像素点划分为两类。然后，用这两类的中心点像素值替代原有像素值，满足题目的要求。

根据分析，主要步骤如下：

1. **数据预处理**

读取图像，并将图像转换为函数cv2.kmeans()可以处理的形式。

在读取图像时，如果是3 个通道的RGB 图像，需要将图像的RGB 值处理为一个单独的特征值。具体实现时，用函数cv2.reshape()完成对图像特征值的调整。

为了满足函数cv2.kmeans()的要求，需要将图像的数据类型转换为numpy.float32 类型。

1. **设置参数**

设 置参数criteria 的值为“ (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)”，让函数cv2.kmeans()在达到一定精度或者达到一定迭代次数时，即停止迭代。

设置参数K 的值为2，将所有像素划分为两类。

1. **调用函数cv2.kmeans()**

调用函数cv2.kmeans()，获取返回值，用于后续步骤的操作。

1. **值替换**

将像素点的值替换为当前分类的中心点的像素值。

1. **显示结果**

分别显示原始图像和二值化图像。

import numpy as np

import cv2

import matplotlib.pyplot as plt

#读取待处理图像

img = cv2.imread('lena.bmp')

#使用reshape将一个RGB像素点值的三个值作为一个单元

data = img.reshape((-1,3))

#转换为kmeans可以处理的类型

data = np.float32(data)

#调用kmeans模块

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

K =2

ret,label,center=cv2.kmeans(data,K,None,criteria,10,cv2.KMEANS\_RANDOM\_CENTERS)

# 转换为uint8数据类型，将每个像素点值都赋值为当前组的中心点值

#将center的值转换为uint8

center = np.uint8(center)

#使用center内的值替换原有像素点值

res1 = center[label.flatten()]

#使用reshape调整替换后图像

res2 = res1.reshape((img.shape))

#显示处理结果

plt.subplot(121)

plt.imshow(img)

plt.axis('off')

plt.subplot(122)

plt.imshow(res2)

plt.axis('off')

  
调整程序中的K 值，就能改变图像的显示结果。例如，K=8，则可以让图像显示8 个灰度级。

# 参考3

[Python，OpenCV中的K均值聚类 - 掘金 (juejin.cn)](https://juejin.cn/post/7023696890843627557)

# Python，OpenCV中的K均值聚类

# Python，OpenCV中的K均值聚类

这篇博客将介绍什么是 K-Means 聚类以及 如何使用 cv2.kmeans() 函数进行数据聚类。

* K-Means Cluster K均值聚类
* cv2.kmeans() 进行数据聚类

## 1. 效果图

**抽样生成5堆点后聚类，分别以不同的颜色绘制每一种分类，效果图1如下：同样生成5堆点聚类5，得到效果图如下：抽样生成10堆点后聚类，效果图如下:单特征聚类图：** T恤size只根据身高进行分类为3种**2个特征聚类图：** T恤size根据身高和体重共同来进行聚类，

**颜色量化以减少色彩数量，并且得到新的图像，原图 VS 聚类15种色彩效果图如下：**

聚类色彩数越多，算法要相对慢一些，但看起来跟原图越接近。

## 2. 原理

### 2.1 什么是K均值聚类？

考虑一家公司将向市场推出一种新型 T 恤。显然，他们将不得不制造不同尺寸的模型以满足各种尺寸的人。于是公司做了一个人的身高体重数据，绘制成图表，如下图：公司无法生产所有尺寸的 T 恤。相反，他们将人分为小、中、大，只生产这三种适合所有人的型号。**这种将人分成三组可以通过 k-means 聚类来完成，算法为我们提供了最好的 3 种尺寸，这将满足所有人的需求。如果没有，公司可以将人分成更多组，可能是五个**，如下图：

### 2.2 K均值聚类过程

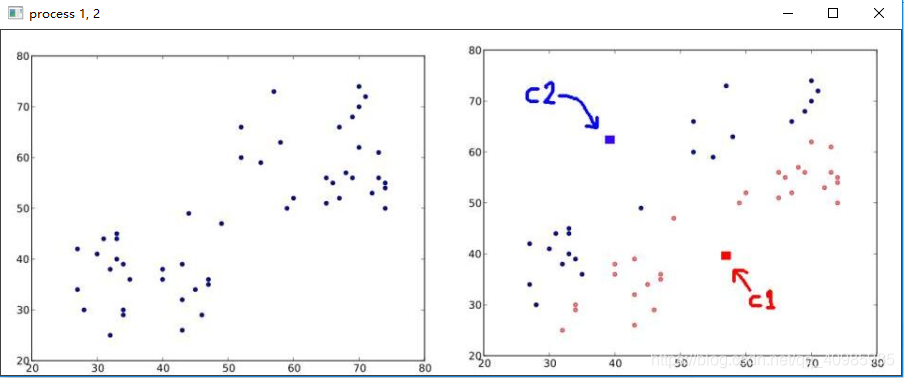
K均值聚类是一个迭代过程，最基本的K均值聚类有以下4个步骤。

1. 假设有一堆点，要分成2类；
2. 随机选择俩个点C1，C2作为质心，计算周围点到C1，C2的距离，距离C1近则归入C1类，距离C2近则归入C2类。
3. 分割完后，计算C1类所有点的平均值作为C1的新质心，计算C2类所有点的平均值作为C2的新质心；
4. 迭代2，3过程，根据设置的条件终止（迭代过程达到多少次，或达到设置的精度）终止。
5. 最终的收敛是C1到其所有点的距离 + C2到其所有点的距离 = 最小值。

这些点使得测试数据与其对应的质心之间的距离之和最小。

**以上4步只是 K-Means 聚类的顶层。该算法有很多优化，例如：如何选择初始质心，如何加快迭代过程。**

该迭代过程可以用以下图片近似表示，分别代表以上4个过程。

**下图迭代1，2步骤，随机选中点作为质心点，计算距离进行分类；****下图重新计算质心后迭代3，4步骤然后得到结果图，可以看到多次迭代后随机点被分成2堆点，红色，蓝色；质心点也相对居中，比较规整。**

### 2.3 cv2.kmeans(z, 2, None, criteria, 10, flags)

**compactness, labels, centers = cv2.kmeans(z, 2, None, criteria, 10, flags)**

**入参:**

* *samples: np.float32 数据类型，每个特征放在一个列中。*
* *nclusters(K) : 最后需要的簇数*
* *creteria: 迭代终止标准。当满足此条件时，算法迭代停止。实际上，它应该是一个包含 3 个参数的元组。****（type、max\_iter、epsilon）:****a - 终止条件的类型：它有 3 个标志，cv2.TERM\_CRITERIA\_EPS - 如果达到指定的准确度 epsilon，则停止算法迭代。cv2.TERM\_CRITERIA\_MAX\_ITER - 在指定的迭代次数 max\_iter 后停止算法。cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER - 当满足上述任何条件时停止迭代。 b - 最大迭代次数 - 指定最大迭代次数的整数。 c - 精度 - 要求的准确性*
* *attempts ：标记以指定使用不同的初始标签执行算法的次数。该算法返回产生最佳紧凑性的标签。这种紧凑性作为输出返回。*
* *flags ：此标志用于指定如何采用初始中心。通常为此使用两个标志：cv2.KMEANS\_PP\_CENTERS 和 cv2.KMEANS\_RANDOM\_CENTERS。*

**返回值：**

* *compactness: 紧凑度，指每个点到其相应中心的距离平方和*
* *labels: 标签数组，其中每个元素都标记为“0”、“1”.....*
* *centers: 聚簇中心的数组*

## 3. 源码

### 3.1 抽样生成点后聚类

# K均值聚类随机生成一堆点并聚类demo

from \_\_future\_\_ import print\_function

import cv2 as cv

import numpy as np

from numpy import random

def make\_gaussians(cluster\_n, img\_size):

points = []

ref\_distrs = []

for \_i in range(cluster\_n):

mean = (0.1 + 0.8 \* random.rand(2)) \* img\_size

a = (random.rand(2, 2) - 0.5) \* img\_size \* 0.1

cov = np.dot(a.T, a) + img\_size \* 0.05 \* np.eye(2)

n = 100 + random.randint(900)

pts = random.multivariate\_normal(mean, cov, n)

points.append(pts)

ref\_distrs.append((mean, cov))

points = np.float32(np.vstack(points))

return points, ref\_distrs

def main():

cluster\_n = 5

img\_size = 300

# 生成明亮的调色板

colors = np.zeros((1, cluster\_n, 3), np.uint8)

colors[0, :] = 255

colors[0, :, 0] = np.arange(0, 180, 180.0 / cluster\_n)

colors = cv.cvtColor(colors, cv.COLOR\_HSV2BGR)[0]

while True:

print('sampling distributions...')

points, \_ = make\_gaussians(cluster\_n, img\_size)

term\_crit = (cv.TERM\_CRITERIA\_EPS, 30, 0.1)

\_ret, labels, \_centers = cv.kmeans(points, cluster\_n, None, term\_crit, 10, 0)

img = np.zeros((img\_size, img\_size, 3), np.uint8)

for (x, y), label in zip(np.int32(points), labels.ravel()):

c = list(map(int, colors[label]))

cv.circle(img, (x, y), 1, c, -1)

cv.imshow('kmeans', img)

ch = cv.waitKey(0)

if ch == 27:

break

print('Done')

if \_\_name\_\_ == '\_\_main\_\_':

print(\_\_doc\_\_)

main()

cv.destroyAllWindows()

### 3.2 单特征聚类、多特征聚类、颜色量化

# OpenCV实现K-Means Cluster K均值聚类

# 输入参数

# - samples: np.float32 数据类型，每个特征放在一个列中。

# - nclusters(K) : 最后需要的簇数

# - creteria: 迭代终止标准。当满足此条件时，算法迭代停止。实际上，它应该是一个包含 3 个参数的元组。它们是（类型、max\_iter、epsilon）：

# a - 终止条件的类型：它有 3 个标志，如下所示：cv2.TERM\_CRITERIA\_EPS - 如果达到指定的准确度 epsilon，则停止算法迭代。

# cv2.TERM\_CRITERIA\_MAX\_ITER - 在指定的迭代次数 max\_iter 后停止算法。

# cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER - 当满足上述任何条件时停止迭代。

# b - 最大迭代次数 - 指定最大迭代次数的整数。

# c - 精度 - 要求的准确性

# - attempts ：标记以指定使用不同的初始标签执行算法的次数。该算法返回产生最佳紧凑性的标签。这种紧凑性作为输出返回。

# - flags ：此标志用于指定如何采用初始中心。通常为此使用两个标志：cv2.KMEANS\_PP\_CENTERS 和 cv2.KMEANS\_RANDOM\_CENTERS。

#

# 输出参数

#

# - compactness: 紧凑度，指每个点到其相应中心的距离平方和

# - labels: 标签数组，其中每个元素都标记为“0”、“1”.....

# - centers: 聚簇中心的数组

# 1. 只有一个特征的数据（T恤问题，只根据身高）

import numpy as np

import cv2

from matplotlib import pyplot as plt

x = np.random.randint(25, 100, 25)

y = np.random.randint(175, 255, 25)

z = np.hstack((x, y))

z = z.reshape((50, 1))

z = np.float32(z)

plt.hist(z, 256, [0, 256]), plt.show()

# 每当运行 10 次算法迭代或达到 epsilon = 1.0 的准确度时，停止算法并返回

# 定义终止准则 criteria = ( type, max\_iter = 10 , epsilon = 1.0 )

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

# 设置标志（只是为了避免代码中的换行符）

flags = cv2.KMEANS\_RANDOM\_CENTERS

# 应用K均值聚类算法

compactness, labels, centers = cv2.kmeans(z, 2, None, criteria, 10, flags)

A = z[labels == 0]

B = z[labels == 1]

# 以红色绘制 A，以蓝色绘制 B，以黄色绘制它们的质心。

plt.hist(A, 256, [0, 256], color='r')

plt.hist(B, 256, [0, 256], color='b')

plt.hist(centers, 32, [0, 256], color='y')

plt.title("one feature res")

plt.show()

# 2. 具有多重数据的特征（T恤问题，根据身高、体重）

import numpy as np

import cv2

from matplotlib import pyplot as plt

X = np.random.randint(25, 50, (25, 2)) # 身高

Y = np.random.randint(60, 85, (25, 2)) # 体重

Z = np.vstack((X, Y))

# 转换为 np.float32

Z = np.float32(Z)

# 定义终止准则以及应用KMeans聚类

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

ret, label, center = cv2.kmeans(Z, 2, None, criteria, 10, cv2.KMEANS\_RANDOM\_CENTERS)

# 分离数据，并拉平flatten

A = Z[label.ravel() == 0]

B = Z[label.ravel() == 1]

# 绘制数据

plt.scatter(A[:, 0], A[:, 1])

plt.scatter(B[:, 0], B[:, 1], c='r')

plt.scatter(center[:, 0], center[:, 1], s=80, c='y', marker='s')

plt.xlabel('Height'), plt.ylabel('Weight')

plt.title("two features res")

plt.show()

## 3. 颜色量化

# 颜色量化是减少图像中颜色数量的过程。这样做的原因之一是减少内存。有时某些设备可能有限制，以至于它只能产生有限数量的颜色。

# 在这些情况下，也会执行颜色量化。这里我们使用 k-means 聚类进行颜色量化。有3个特征，例如 R、G、B。需要将图像重塑为 Mx3 大小的数组（M 是图像中的像素数）。

# 在聚类之后将质心值（也是 R、G、B）应用于所有像素，这样生成的图像将具有指定数量的颜色，最后需要将其重塑回原始图像的形状。

import numpy as np

import cv2

img = cv2.imread('./ml.png')

Z = img.reshape((-1, 3))

# 转换 np.float32

Z = np.float32(Z)

# 定义终止准则以及应用KMeans聚类

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)

K = 15

ret, label, center = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS\_RANDOM\_CENTERS)

# 转换回uint8, 制作原始图像

center = np.uint8(center)

res = center[label.flatten()]

res2 = res.reshape((img.shape))

cv2.imshow("origin",img)

cv2.imshow('res2', res2)

cv2.waitKey(0)

cv2.destroyAllWindows()