【华子】机器学习021-使用K-means进行图片的矢量量化操作 -

(本文所使用的Python库和版本号: Python 3.5, Numpy 1.14, scikit-learn 0.19, matplotlib 2.2)

前一篇文章我们讲解了K-means算法的定义方法，并用K-means对数据集进行了简单的聚类分析。此处我们讲解使用k-means对图片进行矢量量化操作。

1. 矢量量化简介

矢量量化（Vector Quantization, VQ）是一种非常重要的信号压缩方法，在图片处理，语音信号处理等领域占据十分重要的地位。

矢量既有大小又有方向,比如说力 、速度、位移。
标量只有大小没有方向,比如说质量、体积、温度、路程。

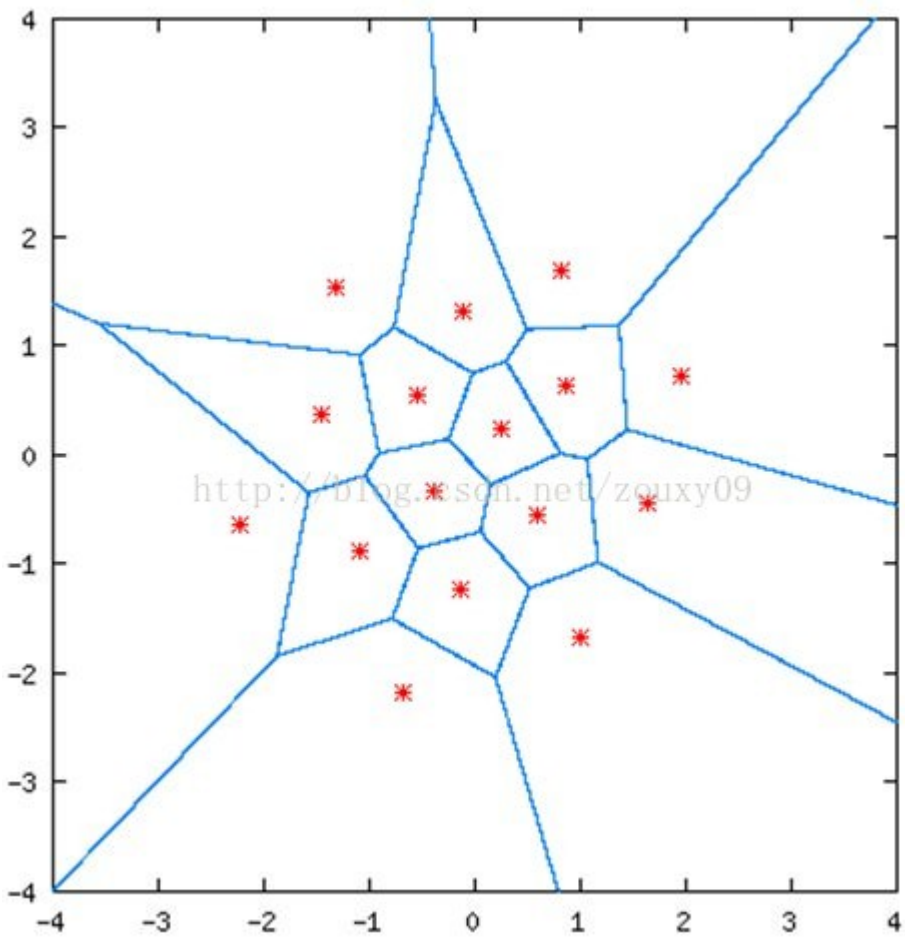
矢量量化是一种基于块编码规则的有损数据压缩方法，在图片压缩格式JPEG和视频压缩格式MPEG-4中都有矢量量化这一步，其基本思想是：将若干个标量数据组构成一个矢量，然后在矢量空间给以整体量化，从而达到压缩数据的同时但不损失多少信息。

矢量量化实际上是一种逼近，其核心思想和“四舍五入”基本一样，就是用一个数来代替其他一个数或者一组数据，比如有很多数据（6.235,6.241,6.238,6.238954,6.24205.。。），这些数据如果用四舍五入的方式，都可以得到一个数据6.24，即用一个数据（6.24）来就可以代表很多个数据。

了解了这个基本思想，我们可以看下面的一维矢量量化的例子：

在这个数轴上，有很多数据，我们可以用-3来代表所有小于-2的数据，用-1代表-2到0之间的数据，用1代表0到2之间的数据，用3代表大于2的数据，故而整个数轴上的无线多个数据，都可以用这四个数据（-3，-1,1,3）来表示，我们可以对这四个数进行编码，只需要两个bit就可以，如（-3=00，-1=01,1=10,3=11），所以这就是1-dimensional(维度), 2-bit VQ，其量化率rate=2bits/dimension.

下面看看稍微复杂一点的二维矢量量化的例子：

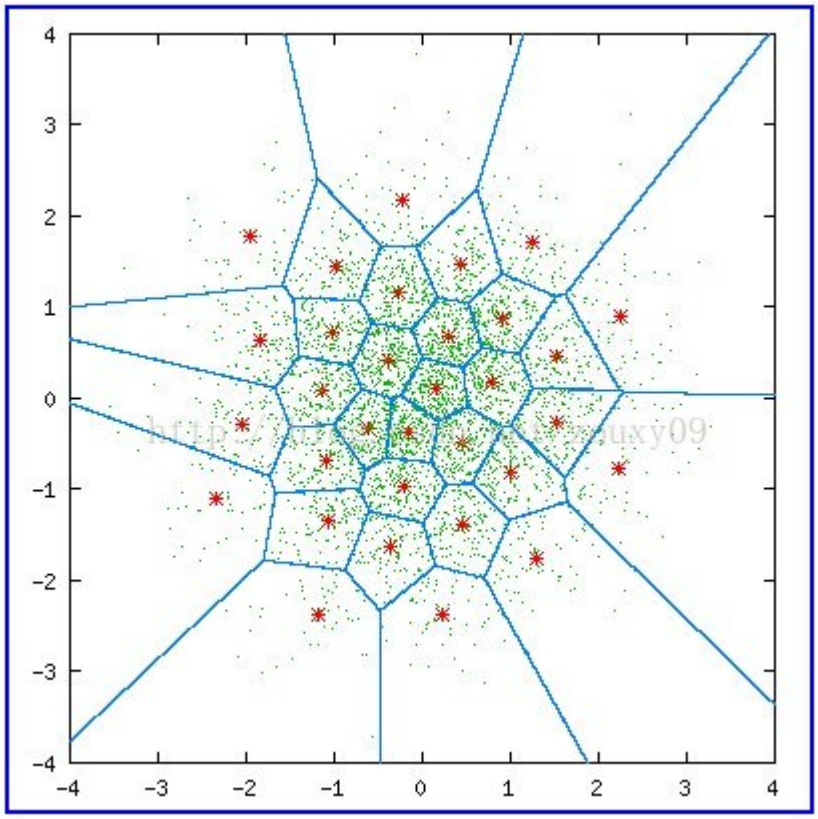


由于是二维，故而平面上的任意一个点都可以表示为（x,y）这种坐标形式，图中，我们用蓝色实线将整个二维平面划分为16个区域，故而任意一个数据点都会落到这16个区域的某一个。我们可以用平面上的某些点来**代表**这个平面区域，故而得到16个红点，这16个红点的坐标就代表了某一个区域内的所有二维点。

更进一步，我们就用4bit二进制码来编码表示这16个数，故而这个问题是2-dimensional, 4-bit VQ， 其量化率也是rate=2bits/dimension.

此处图中显示的红星，也就是16个代表，被称为**编码矢量**（code vectors），而蓝色边界定的区域叫做**编码区域**（encoding regions)，所有这些编码矢量的集合被称为**码书**（code book), 所有编码区域的集合称为**空间的划分**（partition of the space).

对于图像而言，可以认为图像中的每个像素点就是一个数据，用k-means对这些数据进行聚类分析，比如将整幅图像聚为K类，那么会得到K个不同的质心（关于质心的理解和直观感受，可以参考**[机器学习020-使用K-means算法对数据进行聚类分析]**），或者说通俗一点，可以得到K个不同的数据代表，这些数据代表就可以代表整幅图像中的所有点的像素值，故而我们只需要知道这K个数据代表就可以了（想想人大代表就明白这个道理了），从而可以极大的减少图片的存储空间（比如一张bmp的图像可能有2-3M，而压缩成jpg后只有几百K的大小，当然压缩成jpg的过程还有其他压缩方式，不仅仅是矢量量化，但大体意思相同），当然，这个代表的过程会造成一定的图像像素失真，失真的程度就是K的个数了。用图片可以表示为：（小的绿色的点就是训练样本。）



（以上内容部分来源于博客矢量量化（Vector Quantization））

2. 使用K-means对图像进行矢量量化操作

根据上面第一部分对矢量量化的介绍，我们可以对某一张图片进行矢量量化压缩，可以从图片中提取K个像素代表，然后用这些代表来表示一张图片。具体的代码为：

```
from sklearn.cluster import KMeans
# 构建一个函数来完成图像的矢量量化操作
def image_VQ(image,K_nums): # 貌似很花时间。。
    # 构建一个KMeans对象
    kmeans=KMeans(n_clusters=K_nums,n_init=4)
    # 用这个KMeans对象来训练数据集，此处的数据集就是图像
    img_data=image.reshape((-1,1))
    kmeans.fit(img_data)
    centroids=kmeans.cluster_centers_.squeeze() # 每一个类别的质心
    labels=kmeans.labels_ # 每一个类别的标记
    return np.choose(labels,centroids).reshape(image.shape)
```

上面我们先建立一个函数来完成图像的矢量量化压缩操作，这个操作首先建立一个Kmeans对象，然后用这个KMeans对象来训练图像数据，然后提起分类之后的每个类别的质心和标记，并使用这些质心来直接替换原始图像像素，即可得到压缩之后的图像。

为了查看原始图像和压缩后图像，我们将这两幅图都绘制到一行，绘制的函数为：

```
# 将原图和压缩图都绘制出来，方便对比查看效果
def plot_imgs(raw_img,VQ_img,compress_rate):
    assert raw_img.ndim==2 and VQ_img.ndim==2, "only plot gray scale images"
    plt.figure(12,figsize=(25,50))
    plt.subplot(121)
    plt.imshow(raw_img,cmap='gray')
    plt.title('raw_img')
    plt.subplot(122)
    plt.imshow(VQ_img,cmap='gray')
    plt.title('VQ_img compress_rate={:.2f}%'.format(compress_rate))
    plt.show()
```

为了使用方便，我们可以直接将压缩图像函数和显示图像函数封装到一个更高级的函数中，方便我们直接调用和运行，如下所示：

```
import cv2
def compress_plot_img(img_path,num_bits):
    assert 1<=num_bits<=8, 'num_bits must be between 1 and 8'
    K_nums=np.power(2,num_bits)

    # 计算压缩率
    compression_rate=round(100*(8-num_bits)/8,2)
#     print('compression rate is {:.2f}%'.format(compression_rate))

    image=cv2.imread(img_path,cv2.IMREAD_GRAYSCALE) # 读取为灰度图
    VQ_img=image_VQ(image,K_nums)
    plot_imgs(image,VQ_img,compression_rate)
```

准备好了各种操作函数之后，我们就可以直接调用compress_plot_img()函数来压缩和显示图像，下面是采用三种不同的比特位来压缩得到的图像，可以对比看看效果。

```
img_path = './021-kmeanstest.jpg'
compress_plot_img(img_path,4)
compress_plot_img(img_path,2)
compress_plot_img(img_path,1)
```

##########################**小**********结#################################

1, 对图像进行矢量量化压缩，其本质就是将图像数据划分为K个不同类比，这种思想和K-means的思想一致，故而对图像进行矢量量化是Ｋ-means算法的一个重要应用。

2, 通过K-means算法得到图像的K个类别的质心后，就可以用着Ｋ个不同质心来代替图像像素，进而得到压缩之后的，有少许失真的图像。

3, 从上述三幅图的比较可以看出，图像压缩率越大，图像失真的越厉害，最后的比特位为1时的图像可以说就是二值化图，其像素值非0即1，非1即0。

##

参考资料:

1, Python机器学习经典实例，Prateek Joshi著，陶俊杰，陈小莉译