K-means 小作业

211870287 丁旭

2023 年11 月14 日

题目 1. 挑选一张你喜欢的图片，如有需要可以进行预处理（缩放、去噪、 归一化等）随机选取k 个中心点作为聚类中心，进行迭代，得到不同k 值下的聚类簇的可视化结果。请自行设置停止迭代的条件。需要提交的内容 包括但不限于：

1. 输入图像可视化、聚类的可视化；
2. 可运行代码，关键部分代码需要注释；
3. PDF 报告，其中阐述 k-means 算法的原理，包含详细的步骤说明、运行结果和截图。

解答. k-means 算法的原理：首先确定要分类的种数 k。然后在样本中随机选出k个样本作为初始的聚类中心，然后对于每个样本点计算样本点到聚类中心的聚类，并将其与最近的聚类中心分到一类。分完类后判断聚类的结果是否满足标准，若满足则直接结束，否则重新计算聚类中心，并开始新一轮的迭代，直到聚类结果满足标准或者达到最大循环次数为止。

步骤说明:

Kmeans函数的输入为一张图像self.img，其输出为图像中像素点所属的簇，保存在列表self.images中。

函数参数：

iter：迭代次数，即算法运行的最大轮数。

k：簇的个数，需要自己指定。

tol：停止条件，移动距离小于该值时算法终止。

具体实现过程如下：

数据预处理：将所有像素点的RGB三元组展开成一维向量，存储在矩阵data中。同时，在data的最后一列新增一列，用来记录每个像素点所属的簇心编号。

随机初始化：随机从data中选取k个像素点作为初始簇心。

迭代更新：对于每个迭代轮次，计算每个像素点到各个簇心的距离，并将其归为距离最近的簇心的簇。然后根据每个簇中的像素点计算该簇的新簇心。如果新旧簇心之间的距离小于tol，则停止迭代。最后将所有像素点所属的簇保存在列表self.images中。

关键代码解读：

img = plt.imread('enviromment.jpg')

读入图片，并以三维数组的形式存储（高，宽， 3）

3是特征数，图片即为RGB属性

由于K-means算法需要的数据应为样本数×特征数，因此进行以下变换

img = img.reshapt(-1, 3)

最后，我们新增一列，来标识每个像素点的聚类信息

img = np.column__stack((img, np.ones(row*col)))

质心选择开始时，随机选择[0, col*row-1]范围内的任意k个数，作为索引，从data（数据集）中根据索引，找到初始质心

cluster_center = data[np.random.choice(row*col, k)]

随后计算每个点到这k个质心的欧氏距离

distance = np.sqrt(np.sum((x - y)**2, axis=1))

将距离最近的质心的索引放入载入数据时添加的第三列中

```
data[:, 3] = np.argmin(distance, axis=0)
```

最后计算全部k个聚类的新质心（即该聚类中的所有点的均值）

```
for j in range(k):
    cluster.center[j] = np.mean(data[data[:, 3] == j], axis=0)
```

随后开始迭代这一过程

```python
# k是分组数；tol‘中心点误差’；iter是迭代次数
def kmeans(self, iter, k, tol):

    # 保存图片的行宽和列宽
    row = self.img.shape[0]
    col = self.img.shape[1]
    data = self.img.reshape(-1, 3)

    # 添加一列 之后用来存储离这个像素距离最近（也就是颜色最相近的簇心的下标j  簇心即为cluster_center[j]）
    data = np.column_stack((data, np.ones(row * col)))

    # 1.随机产生初始簇心
    cluster_center = data[np.random.choice(row * col, k)]

    # 2.分类
    distance = [[] for _ in range(k)]

    for i in range(iter):
        print("迭代次数：", i+1)
        # 2.1距离计算
        for j in range(k):
            # 这里采用了“广播”（broadcasting）的机制，让 cluster_center[j] 按照 data 的行数进行自动复制，使其变成一个与 data 形状相同的矩阵。
            distance[j] = np.sqrt(np.sum((data - cluster_center[j]) ** 2, axis=1))
        # 2.2归类
        data[:, 3] = np.argmin(distance, axis=0)
        # 3.计算新簇心
        pre_cluster_center = np.copy(cluster_center)
        for j in range(k):
            cluster_center[j] = np.mean(data[data[:, 3] == j], axis=0)

        # 4.停止条件
        move_distance = np.sqrt(np.sum((np.array(cluster_center) - np.array(pre_cluster_center)) ** 2))
        if move_distance < tol:
            print("移动距离小于阈值，算法终止。")
            break
    img_re = (data[:, 3]).reshape(row, col)
    self.images.append(img_re)
```

图 1: kmeans 方法的具体实现

调用kmeans函数 这里迭代次数均为100次，tol为0.001，并且簇的个数从2到6.

```
if __name__ == "__main__":
    unsupervisedLearning = UnsupervisedLearning('environment.jpg')
    for i in range(2, 7):
        unsupervisedLearning.kmeans(100, i, 0.001)
```

绘制分类结果：

```
def show(self):
    # 用来正常显示中文标签
    plt.rcParams['font.sans-serif'] = ['SimHei']
    # 显示图像
    titles = [u'原始图像', u'聚类图像 K=2', u'聚类图像 K=3',
              u'聚类图像 K=4', u'聚类图像 K=5', u'聚类图像 K=6']

    for i in range(len(self.images)):
        plt.subplot(2, 3, i + 1), plt.imshow(self.images[i]),
        plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
    plt.show()
```

原图：

结果图：