K-means 小作业

211098325 毛丁

2023 年 10 月 12 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。

详细的实验步骤如下：

1. 首先定义 k-means 算法的具体类如下：其主要有两个函数 fit 和 pre-

dict。fit 函数是 kmeans 算法的主要流程的实现函数，其接收需要进行分类的 data，然后将聚类结果保存在类的变量中。predict 函数接收一个样本，然后返回这个样本应该属于的聚类中心的值。

```python
class K_Means(object):
    # k是分组数；tolerance‘中心点误差’；max_iter是迭代次数
    new *
    def __init__(self, k=2, tolerance=0.001, max_iter=300):
        self.centers_ = {}
        self.k_ = k
        self.tolerance_ = tolerance
        self.max_iter_ = max_iter

    1 usage   new *
    def fit(self, data):
        # 随机初始化聚类中心
        center_idx = np.random.randint(0, len(data), self.k_)
        for i in range(self.k_):
            self.centers_[i] = data[center_idx[i]]
        # 开始迭代
        for i in range(self.max_iter_):
            self.category = {}
            for i in range(self.k_):
                self.category[i] = []
            for dot in data:
                distances = []
                for center_idx in self.centers_:
                    # 使用欧氏距离
                    distances.append(np.linalg.norm(dot - self.centers_[center_idx]))
                classification = distances.index(min(distances))
                self.category[classification].append(dot)

            # 更新聚类中心
            prev_centers = dict(self.centers_)
            for c in self.category:
                self.centers_[c] = np.average(self.category[c], axis=0).astype(int)

            # 判断聚类结果是否满足要求
            optimized = True
            # 消除被除数为0的警告
            np.seterr(divide='ignore', invalid='ignore')
            for center_idx in self.centers_:
                org_centers = prev_centers[center_idx]
                cur_centers = self.centers_[center_idx]

                if np.sum((cur_centers - org_centers) / org_centers * 100.0) > self.tolerance_:
                    optimized = False
            if optimized:
                break

    # 给定一个像素点 返回它应该属于的聚类中心
    1 usage   new *
    def predict(self, p_data):
        distances = [np.linalg.norm(p_data - self.centers_[center]) for center in self.centers_]
        index = distances.index(min(distances))
        return self.centers_[index]
```

图 1: kmeans 类的实现

2. main 函数的实现如下图所示。对于一个图像的基本处理思想为：利用 OpenCV 库将图像的 RGB 值保存在数组中，数组的每一个元素都代表一个像素点，并用一个 1X3 的数组比如 [100,200,250] 表示，然后将这个数组传入到 kmeans 类的实例中，将聚类结果保存到类的成员变量里，然后遍历这个数组的每一个像素点，将其 RGB 值修改为聚类中心的 RGB 值，最后展示出来。

```
if __name__ == '__main__':
    # 读取原始图像
    img = cv2.imread('scenery1.png')
    # print(img.shape)
    # 图像二维像素转换为一维
    data = img.reshape((-1, 3))

    nets = []
    for i in range(2,7):
        nets.append(K_Means(i))
    arrs = [data]
    for net in nets:
        img_copy = data.copy()
        net.fit(img_copy)
        for idx in range(len(img_copy)):
            img_copy[idx] = net.predict(img_copy[idx])
        arrs.append(img_copy)

    images = []
    for result in arrs:
        img = result.reshape(img.shape)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        images.append(img)
    # 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(6):
        plt.subplot(2, 3, i + 1), plt.imshow(images[i], 'gray'),
        plt.title(titles[i])
        plt.xticks([]), plt.yticks([])
    plt.show()
```

图 2: main 函数

代码运行示例:

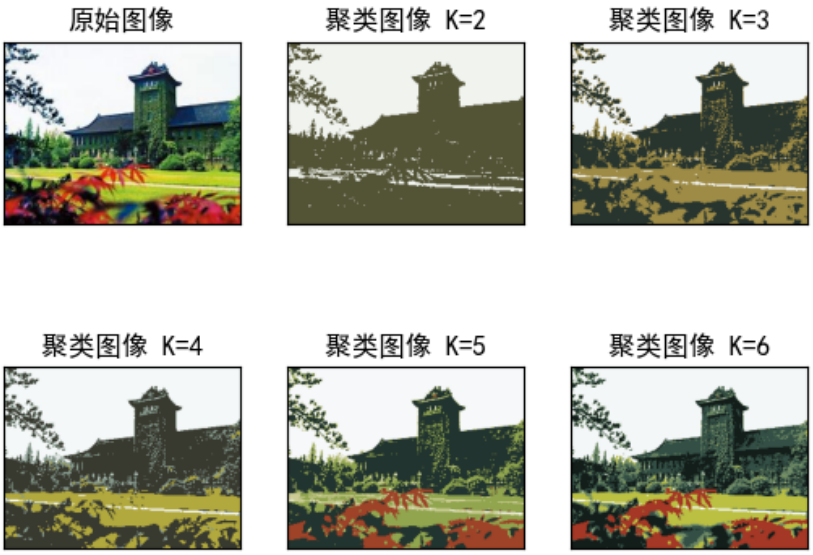

图 3: 聚类结果