山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名：黄瑞哲 | | 班级： 计科17.3 |
| 实验题目：K-Means | | | |
| 实验学时：4 | | 实验日期： 2019.12.6 | |
| 实验目的：  1. 掌握k均值中的KMeans算法  2. 应用KMeans算法实现图片的像素压缩 | | | |
| 硬件环境：  Intel Core i5-8300H @ 2.3GHz | | | |
| 软件环境：  Windows10 Pro 1903  Python 3.7  Visual Studio Code | | | |
| 实验步骤与内容：   1. 数据集   数据集中包含两张tiff格式的图片，一张像素是538x538的，另一张是128x128的。每个像素点是的颜色值是24位的，由3个8位的、范围在[0,255]之间的数构成，分别表示红R、绿G、蓝B。现在我们的任务是在图片中选取16个颜色来替代每个像素点中的颜色以实现图片压缩的效果。     1. KMeans算法   KMeans是实现聚类的一种算法，分为3步   1. 随机选取K个聚类中心 2. 使用欧几里得距离计算出每个样本数据与哪一个聚类中心更相似      1. 使用该类中所有的样本数据重新计算聚类中心     当中心点变化幅度较小或满足一定的迭代次数后算法停止   1. 图片压缩   算法作用在一个三维空间中，这个空间是以RGB三个颜色值为基底的三维空间。那么每个点的RGB值便是这个空间中的一个点，将所有像素点的RGB值放入该三维空间中然后做k=16的聚类，其中16个初始聚类中心从所有的颜色值中随机产生。完成聚类后可以知道每个点属于哪一个聚类中心，然后可以将这个点的RGB值替换为对应的聚类中心的RGB的值，最终再将图片还原即可。    可以看出颜色的对比度有了明显的减小，而且出现了粗糙的颜色轮廓。 | | | |
| 结论分析与体会：  KMeans算法的思路还是非常清晰的，使用样本数据更新聚类中心，然后根据新的聚类中心来分类，不断循环，最后能够得到一个较好的结果。而且算法实现起来也是非常简单，只需要按照算法一步步实现即可。但是自己手写的KMeans算法与现有的算法代码库中的KMeans在运行效率上还是有很大的区别：手写的算法迭代50多次可能需要将近半分钟的时间，而代码库如sklearn中实现的KMeans仅仅需要几秒就可以求得结果。  还有一点，python的绘图库中要求RGB的颜色要么是[0,1]的浮点数，要么是[0,255]的整型，因此在导入样本数据的时候需要将读入的数据/255将其转换为[0,1]之间的浮点数然后再进行运算，否则得到的结果因为会是浮点数，渲染图片的时候将会是错误的。 | | | |

附录：程序源代码

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image

# from sklearn.cluster import KMeans

class KMeans:

def \_\_init\_\_(self, n\_clusters, max\_iter=200):

self.n\_clusters = n\_clusters

self.max\_iter = max\_iter

def fit(self, X):

m, n = np.shape(X)

assert self.n\_clusters <= m

self.\_\_random\_cluster\_centers\_(X)

self.labels\_ = [0 for i in range(m)]

for iter in range(self.max\_iter):

for i in range(m):

k = 0

for j in range(1, self.n\_clusters):

if np.dot(X[i] - self.cluster\_centers\_[j], X[i] - self.cluster\_centers\_[j]) < np.dot(X[i] - self.cluster\_centers\_[k], X[i] - self.cluster\_centers\_[k]):

k = j

self.labels\_[i] = k

for j in range(self.n\_clusters):

sum\_x = np.zeros(n)

sum\_cnt = 0

for i in range(m):

if self.labels\_[i] == j:

sum\_x += X[i]

sum\_cnt += 1

if sum\_cnt > 0:

self.cluster\_centers\_[j] = sum\_x / sum\_cnt

def \_\_random\_cluster\_centers\_(self, X):

import random

ind = random.sample([i for i in range(len(X))], self.n\_clusters)

self.cluster\_centers\_ = []

for i in ind:

self.cluster\_centers\_.append(X[i])

self.cluster\_centers\_ = np.array(self.cluster\_centers\_)

if \_\_name\_\_ == "\_\_main\_\_":

def image\_reconstruction(cluster\_centers\_, labels\_, w, h):

channel = np.shape(cluster\_centers\_)[1]

image = np.zeros((w, h, channel))

ind = 0

for i in range(w):

for j in range(h):

image[i][j] = cluster\_centers\_[labels\_[ind]]

ind += 1

return image

bird = matplotlib.image.imread("exp6/data/bird\_small.tiff")

bird = np.array(bird, dtype=np.uint)

m, n, channel = np.shape(bird)

kmeans = KMeans(n\_clusters=16, max\_iter=50)

kmeans.fit(np.reshape(bird, (m \* n, channel)))

bird\_compressed = image\_reconstruction(kmeans.cluster\_centers\_, kmeans.labels\_, m, n)

matplotlib.image.imsave("exp6/data/bird\_compressed.tiff", bird\_compressed)

plt.figure(1)

plt.title("Origin image")

plt.axis("off")

plt.imshow(bird)

plt.figure(2)

plt.title("Compressed image (16 colors)")

plt.axis("off")

plt.imshow(bird\_compressed)

plt.show()