Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №6**

«Кластеризация»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2019

ХОД РАБОТЫ

**Задание.**

Набор данных ex6data1.mat представляет собой файл формата \*.mat (т.е. сохраненного из Matlab). Набор содержит две переменные X1 и X2 - координаты точек, которые необходимо кластеризовать.

Набор данных bird\_small.mat представляет собой файл формата \*.mat (т.е. сохраненного из Matlab). Набор содержит массив размером (16384, 3) - изображение 128x128 в формате RGB.

1. Загрузите данные ex6data1.mat из файла.
2. Реализуйте функцию случайной инициализации K центров кластеров.
3. Реализуйте функцию определения принадлежности к кластерам.
4. Реализуйте функцию пересчета центров кластеров.
5. Реализуйте алгоритм K-средних.
6. Постройте график, на котором данные разделены на K=3 кластеров (при помощи различных маркеров или цветов), а также траекторию движения центров кластеров в процессе работы алгоритма
7. Загрузите данные bird\_small.mat из файла.
8. С помощью алгоритма K-средних используйте 16 цветов для кодирования пикселей.
9. Насколько уменьшился размер изображения? Как это сказалось на качестве?
10. Реализуйте алгоритм K-средних на другом изображении.
11. Реализуйте алгоритм иерархической кластеризации на том же изображении. Сравните полученные результаты.

**Результат выполнения:**

1. Код выгрузки данных из файла представлен ниже:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex6data1.mat')

dataset = sio.loadmat(file\_path)

X = dataset["X"]

Но здесь стоит отметить, что при визуализации начальных данных я заметил, что те данные, которые даны в этой лабораторной, используются Andrew Ng для PCA, а те данные, которые Andrew Ng использует в K-Means – даны нам в PCA. Т. е. по сути данные с двух работ перемешаны между собой. И если данные, которые Andrew Ng использует для PCA ещё можно использовать в задаче кластеризации, то данные которые были в кластеризации использовать в РСА кажется немного неразумным, поскольку там будет довольно-таки большая ошибка, и данный метод будет работать немного некорректно, и как следствие, графики будут не такими, какими мы их ожидаем увидеть. Поэтому данные, которые были даны к этим лабораторным я опять поменял местами. Т. е. я использую такие же данные, который использует Andrew Ng. Но в любом случае эти данные можно поменять местами и посмотреть на новые графики. Код менять в этом случае не придётся.

1. Код реализации:

def generate\_k\_rand\_centroids(k, n, min=0, max=8):

centroids = []

for i in range(k):

centroid = []

for j in range(n):

centroid.append(random.randint(min, max))

centroids.append(centroid)

return np.array(centroids)

K = 3

initial\_centroids = generate\_k\_rand\_centroids(K, 2)

print(initial\_centroids)

initial\_centroids = np.array([[3, 3], [6, 2], [8, 5]]) # use mock

3. Код реализации:

print(find\_closest\_centroid(np.array([[3.38156267, 3.38911268]]), initial\_centroids))

centroids = find\_closest\_centroid(X, initial\_centroids)

Результат выполнения:

[[1.]]

4. Код реализации:

def find\_closest\_centroid(X, centroids):

K = centroids.shape[0]

idx = np.zeros((X.shape[0], 1))

temp = np.zeros((centroids.shape[0], 1))

for i in range(X.shape[0]):

for j in range(K):

dist = X[i] - centroids[j]

temp[j] = np.sum(dist\*\*2) # a^2 + b^2

idx[i] = np.argmin(temp) + 1

return idx

compute\_centroids(X, centroids, K)

5. Код реализации:

def k\_means(X, idx, K, num\_iters):

for i in range(num\_iters):

# Compute the centroids mean

centroids = compute\_centroids(X, idx, K)

# assign each training example to the nearest centroid

idx = find\_closest\_centroid(X, centroids)

return [centroids, idx]

k\_means(X, centroids, K, 10)

6. Код реализации:

initial\_centroids = generate\_k\_rand\_centroids\_from\_dataset(X, K)

plot\_k\_means(X, initial\_centroids, K, 10)

Результат выполнения:

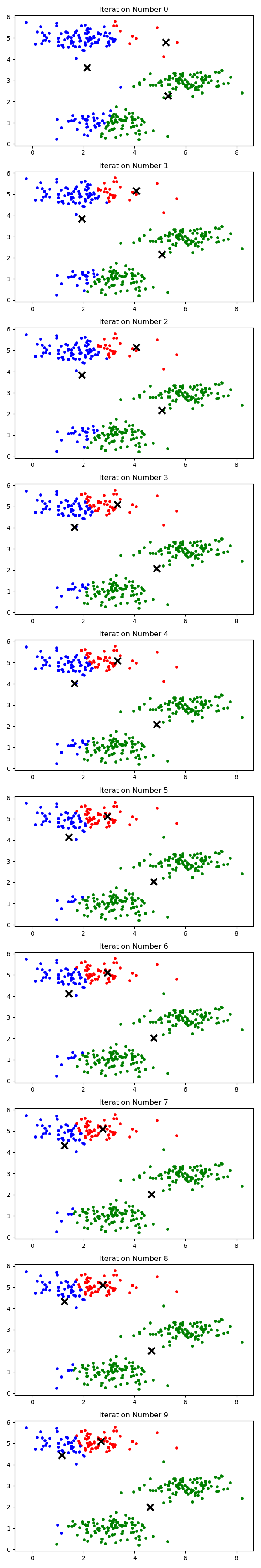
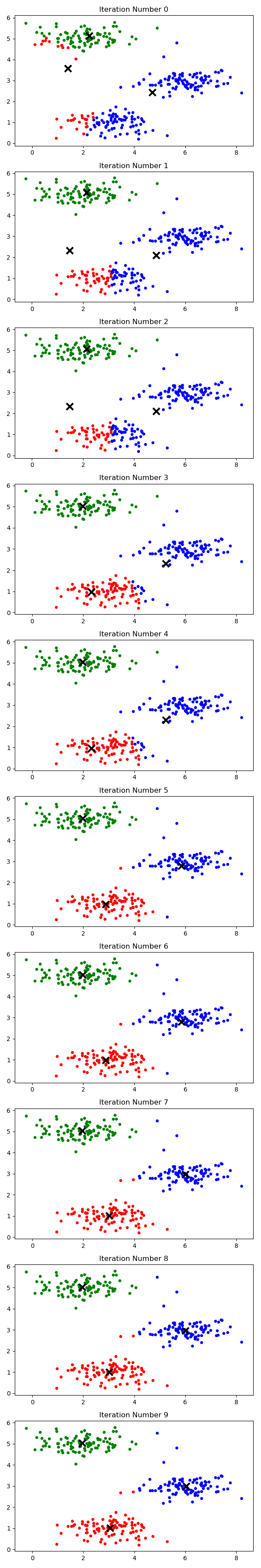
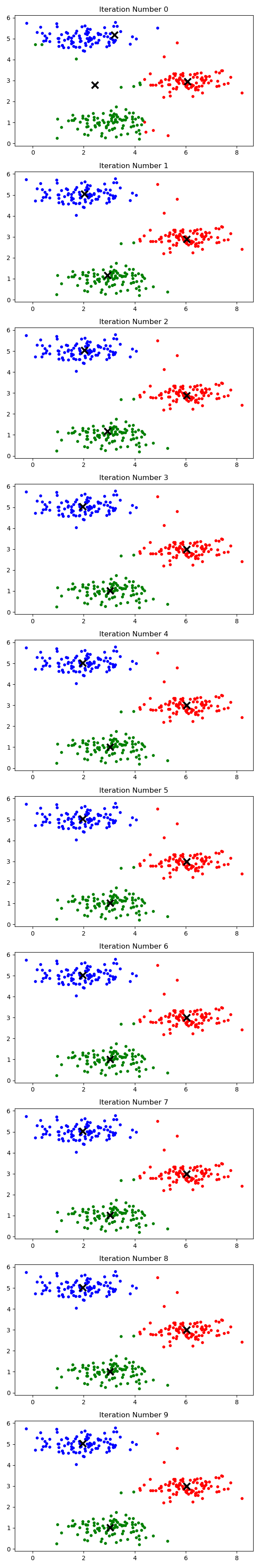


Рисунок 1 – трактория движения центров кластеров

Здесь важен тот факт, что работа алгоритма к-средних очень сильно зависит от выбора начальных кластеров. Изначально реализация была сделана с помощью выбора случайных значений из диапазона, однако наиболее предпочтительным является выбор случайных К начальных данных из самого набора обучения. Как видно на рисунке, из-за того, что точки были выбраны каждый раз по-разному – итоговое расположение классов может сильно отличаться. Для того, чтобы лучше понимать, какие данные лучше взять для обучения рекомендуется запускать алгоритм несколько раз и смотреть, на каких данных ошибка будет минимальной.

7. Код выгрузки данных из файла, нормализация и преобразование данных:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'bird\_small.mat')

dataset = sio.loadmat(file\_path)

A = dataset["A"]

# preprocess and reshape the image

X = (A/255).reshape(128\*128, 3)

8. Код реализации:

K = 16

num\_iters = 10

initial\_centroids = generate\_k\_rand\_centroids\_from\_dataset(X, K, 0, 16384)

idx = find\_closest\_centroid(X, initial\_centroids)

[centroids, idx] = k\_means(X, idx, K, num\_iters)

9. Код реализации:

X\_recovered = X.copy()

for i in range(1, K+1):

X\_recovered[(idx == i).ravel(), :] = centroids[i-1]

# Reshape the recovered image into proper dimensions

X\_recovered = X\_recovered.reshape(128, 128, 3)

# Display the image

fig, ax = plt.subplots(1, 2)

ax[0].imshow(X.reshape(128, 128, 3))

ax[1].imshow(X\_recovered)

plt.show()

Результат выполнения:

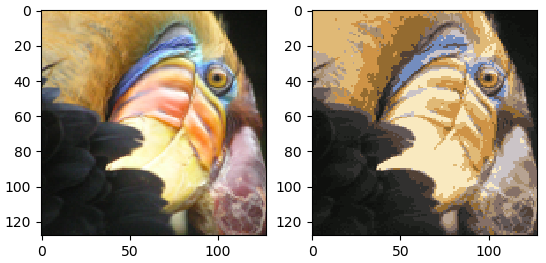


Рисунок 2 – изображение до (слева) и после (справа) кодирования цвета пикселей с помощью к-средних с 16 кластерами

Изображение уменьшилось примерно в 7-8 раз. На приведенных выше рисунках можно аметить, что количество цветов в изображении стало значительно меньше, что привело к эффекту «пикселизации» и «потери» качества.

10. Код реализации:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'bird.png')

matrix = misc.imread(file\_path)

X = (matrix/255).reshape(443\*590, 3)

K = 16

num\_iters = 10

initial\_centroids = generate\_k\_rand\_centroids\_from\_dataset(X, K, 0, 16384)

idx = find\_closest\_centroid(X, initial\_centroids)

[centroids, idx] = k\_means(X, idx, K, num\_iters)

X\_recovered = X.copy()

for i in range(1, K+1):

X\_recovered[(idx == i).ravel(), :] = centroids[i-1]

# Reshape the recovered image into proper dimensions

X\_recovered = X\_recovered.reshape(443, 590, 3)

# Display the image

fig, ax = plt.subplots(1, 2)

ax[0].imshow(X.reshape(443, 590, 3))

ax[1].imshow(X\_recovered)

# np.savetxt('output.txt', idx) # for next assignment

# misc.imsave('output.jpg', X\_recovered)

plt.show()

Закомментированный код необходим для выполнения следующей лабороторной, поскольку там нужно работать с тем же изображением, с которым я работал в этой лабораторной. Поэтому в качестве референса я сохранял данные изображения (матричное представление пикселей и их цветовую схему) и итоговый результат (саму картинку).

Результат выполнения:

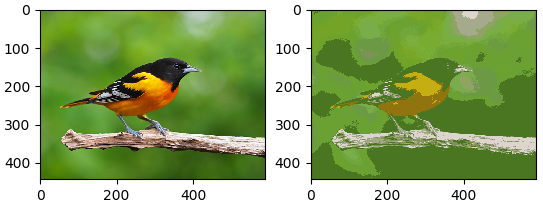


Рисунок 3 – собственное изображение до (слева) и после (справа) кодирования цвета пикселей с помощью к-средних с 16 кластерами с 1 итерацией

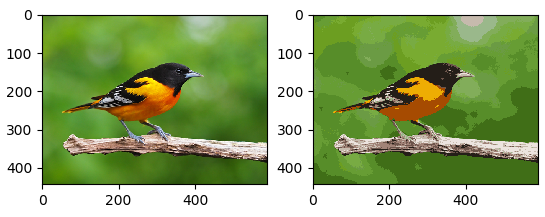


Рисунок 4 – собственное изображение до (слева) и после (справа) кодирования цвета пикселей с помощью к-средних с 16 кластерами с 10 итерациями

Изображение сжалось с 344 килобайт до 29 килобайт. Качество сжатия можно оценить по приведенным изображениям выше. Если описать это простыми словами, то можно сказать, что справа изображение выглядит так, как будто его рисовал художник кистью. Именно так и работает алгоритм ближайших соседей, т. е. берётся каждый пиксель изображения и относится к какому-нибудь одному из 16 классов. Поэтому совокупность пикселей (участки изображения) становятся одним цветом. Из-за этого и возникает этот «эффект».

11. Код реализации:

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'bird-small.png')

img = misc.imread(file\_path)

plt.imshow(img)

plt.show()

img = img / 255 # feature scaling

points = np.reshape(img, (img.shape[0] \* img.shape[1], img.shape[2]))

distance\_mat = pdist(points)

Z = hierarchy.linkage(distance\_mat, 'single')

max\_d = .3

while max\_d > 0.005:

max\_d \*= .5

print(max\_d)

clusters = fcluster(Z, max\_d, criterion='distance')

meshx, meshy = np.meshgrid(np.arange(128), np.arange(128))

plt.axis('equal')

plt.axis('off')

plt.scatter(meshx, -(meshy - 128), c=clusters.reshape(128, 128), cmap='inferno', marker=',')

plt.show()

Результат выполнения:

0.15

0.075

0.0375

0.01875

0.009375

0.0046875

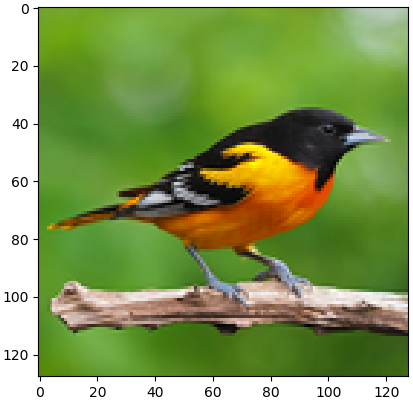
****

Рисунок 5 – исходное изображение

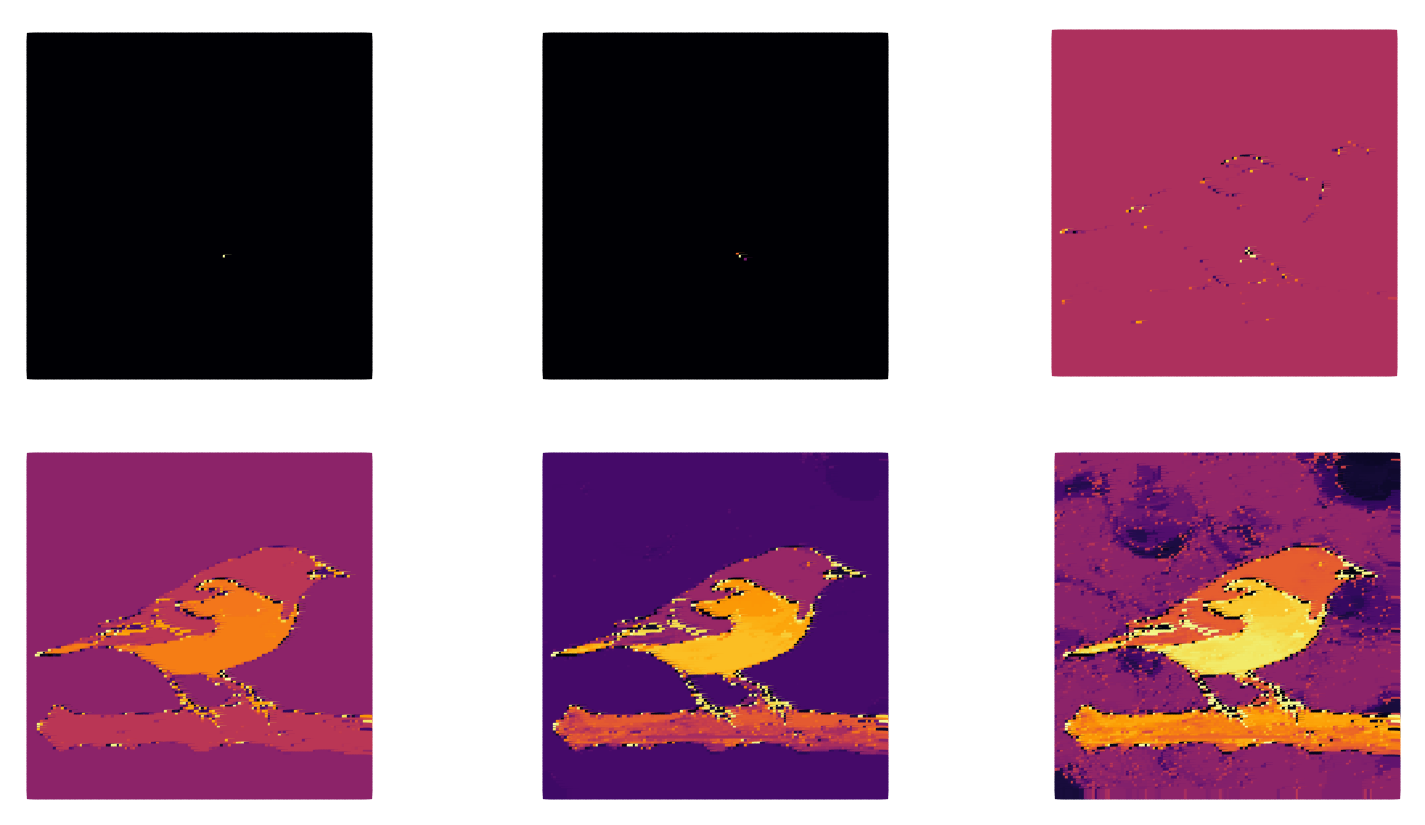
****

Рисунок 6 – результат работы алгоритма иерархической кластеризации

Если сравнивать два алгоритма, то можно увидеть, что они работают по-разному. Оба алгоритма сжимают изображение, на основе кластеризации, однако результаты получаются разными из-за различной концепции и реализации алгоритмов. В случае с иерархической кластеризации код для сжатия был реализован с помощью библиотечной реализации из sklearn. Можно заметить, что цвета «не сохраняются» при втором алгоритме. Но и его используют для других целей, а именно – обнаружение границ изображения.

**Программный код:**

from \_\_future\_\_ import division

from goto import with\_goto

import scipy.io as sio

import scipy.misc as misc

import matplotlib.pyplot as plt

import os

import random

import numpy as np

from scipy.cluster.hierarchy import fcluster

from scipy.cluster import hierarchy

from scipy.spatial.distance import pdist

def generate\_k\_rand\_centroids(k, n, min=0, max=8):

centroids = []

for i in range(k):

centroid = []

for j in range(n):

centroid.append(random.randint(min, max))

centroids.append(centroid)

return np.array(centroids)

def generate\_k\_rand\_centroids\_from\_dataset(X, K, min=0, max=8):

m, n = X.shape[0], X.shape[1]

centroids = np.zeros((K, n))

for i in range(K):

centroids[i] = X[np.random.randint(min, max),:]

return centroids

def find\_closest\_centroid(X, centroids):

K = centroids.shape[0]

idx = np.zeros((X.shape[0], 1))

temp = np.zeros((centroids.shape[0], 1))

for i in range(X.shape[0]):

for j in range(K):

dist = X[i] - centroids[j]

temp[j] = np.sum(dist\*\*2) # a^2 + b^2

idx[i] = np.argmin(temp) + 1

return idx

def compute\_centroids(X, idx, K):

m, n = X.shape[0], X.shape[1]

centroids = np.zeros((K, n))

count = np.zeros((K, 1))

for i in range(m):

index = int((idx[i]-1)[0])

centroids[index] += X[i]

count[index] += 1

return centroids/count

def plot\_k\_means(X, initial\_centroids, K, num\_iters):

"""

plots the data points with colors assigned to each centroid

"""

m, n = X.shape[0], X.shape[1]

idx = find\_closest\_centroid(X, initial\_centroids)

fig, ax = plt.subplots(nrows=num\_iters, ncols=1, figsize=(6, 36))

history = k\_means\_with\_history(X, idx, K, num\_iters)

for i in range(num\_iters):

[centroids, idx] = history[i]

# Visualisation of data

color = "rgb"

for k in range(1, K+1):

grp = (idx == k).reshape(m, 1)

ax[i].scatter(X[grp[:, 0], 0], X[grp[:, 0], 1], c=color[k-1], s=15)

# visualize the new centroids

ax[i].scatter(centroids[:, 0], centroids[:, 1], s=120, marker="x", c="black", linewidth=3)

title = "Iteration Number " + str(i)

ax[i].set\_title(title)

plt.tight\_layout()

plt.show()

def k\_means(X, idx, K, num\_iters):

for i in range(num\_iters):

# Compute the centroids mean

centroids = compute\_centroids(X, idx, K)

# assign each training example to the nearest centroid

idx = find\_closest\_centroid(X, centroids)

return [centroids, idx]

def k\_means\_with\_history(X, idx, K, num\_iters):

history = []

for i in range(num\_iters):

[centroids, idx] = k\_means(X, idx, K, 1)

history.append([centroids, idx])

history.append(k\_means(X, idx, K, 1))

return history

@with\_goto

def main():

goto .task

label .task

# 1

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'ex6data1.mat')

dataset = sio.loadmat(file\_path)

X = dataset["X"]

# 2

K = 3

initial\_centroids = generate\_k\_rand\_centroids(K, 2)

print(initial\_centroids)

initial\_centroids = np.array([[3, 3], [6, 2], [8, 5]]) # use mock

# 3

print(find\_closest\_centroid(np.array([[3.38156267, 3.38911268]]), initial\_centroids))

centroids = find\_closest\_centroid(X, initial\_centroids)

# 4

compute\_centroids(X, centroids, K)

# 5

k\_means(X, centroids, K, 10)

# 6

plot\_k\_means(X, initial\_centroids, K, 10)

initial\_centroids = generate\_k\_rand\_centroids\_from\_dataset(X, K)

plot\_k\_means(X, initial\_centroids, K, 10)

# 7

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'bird\_small.mat')

dataset = sio.loadmat(file\_path)

A = dataset["A"]

# preprocess and reshape the image

X = (A/255).reshape(128\*128, 3)

# 8

K = 16

num\_iters = 10

initial\_centroids = generate\_k\_rand\_centroids\_from\_dataset(X, K, 0, 16384)

idx = find\_closest\_centroid(X, initial\_centroids)

[centroids, idx] = k\_means(X, idx, K, num\_iters)

# 9

X\_recovered = X.copy()

for i in range(1, K+1):

X\_recovered[(idx == i).ravel(), :] = centroids[i-1]

# Reshape the recovered image into proper dimensions

X\_recovered = X\_recovered.reshape(128, 128, 3)

# Display the image

fig, ax = plt.subplots(1, 2)

ax[0].imshow(X.reshape(128, 128, 3))

ax[1].imshow(X\_recovered)

plt.show()

# 10

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'bird.png')

matrix = misc.imread(file\_path)

X = (matrix/255).reshape(443\*590, 3)

K = 16

num\_iters = 10

initial\_centroids = generate\_k\_rand\_centroids\_from\_dataset(X, K, 0, 16384)

idx = find\_closest\_centroid(X, initial\_centroids)

[centroids, idx] = k\_means(X, idx, K, num\_iters)

X\_recovered = X.copy()

for i in range(1, K+1):

X\_recovered[(idx == i).ravel(), :] = centroids[i-1]

# Reshape the recovered image into proper dimensions

X\_recovered = X\_recovered.reshape(443, 590, 3)

# Display the image

fig, ax = plt.subplots(1, 2)

ax[0].imshow(X.reshape(443, 590, 3))

ax[1].imshow(X\_recovered)

# np.savetxt('output.txt', idx)

# misc.imsave('output.jpg', X\_recovered)

plt.show()

# 11

file\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'data', 'bird-small.png')

img = misc.imread(file\_path)

plt.imshow(img)

plt.show()

img = img / 255 # feature scaling

points = np.reshape(img, (img.shape[0] \* img.shape[1], img.shape[2]))

distance\_mat = pdist(points)

Z = hierarchy.linkage(distance\_mat, 'single')

max\_d = .3

while max\_d > 0.005:

max\_d \*= .5

print(max\_d)

clusters = fcluster(Z, max\_d, criterion='distance')

meshx, meshy = np.meshgrid(np.arange(128), np.arange(128))

plt.axis('equal')

plt.axis('off')

plt.scatter(meshx, -(meshy - 128), c=clusters.reshape(128, 128), cmap='inferno', marker=',')

plt.show()

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

main()