Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет информатики и радиоэлектроники”

Факультет компьютерных систем и сетей

Специальность “Обработка больших объемов информации”

Лабораторная работа №6

**Кластеризация**

Выполнил:

магистрант гр. 858641 Кальман В.А.

Проверил:

Стержанов М. В.

Минск 2019

**Данные.**

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.**

**import** **heapq**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **scipy** **import** io

**from** **scipy.spatial** **import** distance\_matrix

**import** **cv2**

**from** **tqdm** **import** tqdm\_notebook

%matplotlib inline

ex6data1 = io.loadmat('Data/Lab 6/ex6data1.mat')

X = ex6data1['X']

**2. Реализуйте функцию случайной инициализации K центров кластеров.**

**def** random\_centers(X, k):

X\_uniq = np.unique(X, axis=0)

rand\_idx = np.random.permutation(len(X\_uniq))

**return** X\_uniq[rand\_idx[:k]]

**3. Реализуйте функцию определения принадлежности к кластерам.**

**def** l2\_dist(x, y, axis=1):

**return** np.sum((x - y) \*\* 2, axis=axis) \*\* 0.5

**def** assign\_centers(X, centers):

centr\_dist = np.array([l2\_dist(X, c) **for** c **in** centers])

**return** np.argmin(centr\_dist, axis=0)

**4. Реализуйте функцию пересчета центров кластеров.**

**def** update\_centers(X, labels):

k = labels.max() + 1

**return** np.array([np.mean(X[labels == i], axis=0) **for** i **in** range(k)])

**5. Реализуйте алгоритм K-средних.**

**def** normalize(X):

mean, std = X.mean(), X.std()

**return** (X - mean) / std, mean, std

**def** kmeans(X, k, max\_iters=1000):

history = []

centers = **None**

X\_norm, X\_mean, X\_std = normalize(X)

**for** i **in** range(max\_iters):

centers = update\_centers(X\_norm, labels) **if** i **else** random\_centers(X\_norm, k)

labels = assign\_centers(X\_norm, centers)

**if** history **and** np.all(history[-1][1] == labels):

**break**

orig\_centres = centers \* X\_std + X\_mean

history.append((orig\_centres, labels))

**return** history

**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.**

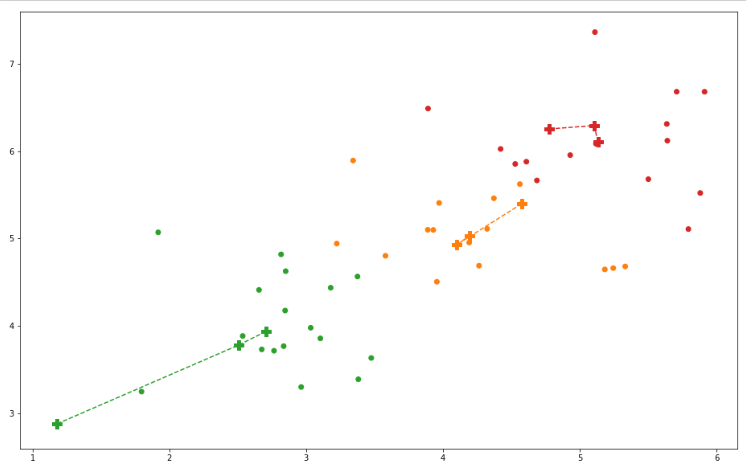


Рисунок 1 – трактория движения центров кластеров

K-means алгоритм чувствителен к выбору начальных координат кластеров. Алгоритм сходится в локальном минимуме, который может совпасть с глобальным, что и произошло в первых двух запусках. При третьем запуске выбор начальных координат оказался ненудачным.

**7. Загрузите данные bird\_small.matиз файла.**

bird\_small = io.loadmat('Data/Lab 6/bird\_small.mat')

A = bird\_small['A']

**8. С помощью алгоритма K-средних используйте 16 цветов для кодирования пикселей.**

**def** cluster\_image\_kmeans(img, k=16):

history = kmeans(img.reshape(-1, 3), k)

centers, labels = history[-1]

reduced\_colors = np.uint8(centers)

reduced\_img = np.reshape([reduced\_colors[x] **for** x **in** labels], img.shape)

**return** reduced\_img

bird\_kmeans = cluster\_image\_kmeans(A)

**9. Насколько уменьшился размер изображения? Как это сказалось на качестве?**

Размер изображения уменьшился более чем в два раза, при этом качество стало чуть хуже (видны пиксели, не такой плавный переход).

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)

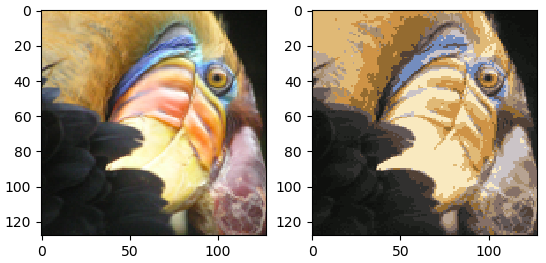
# Display the image

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

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

ax[1].imshow(X\_recovered)

plt.show()

Результат выполнения:

**10. Реализуйте алгоритм K-средних на другом изображении.**

car = cv2.imread('car\_100.png')

car = cv2.cvtColor(car, cv2.COLOR\_BGR2RGB)

car\_kmeans = cluster\_image\_kmeans(car)

fig, ax = plt.subplots(1,2, figsize=(16, 16))

ax[0].imshow(car)

ax[0].axis('off')

ax[0].set\_title('original', size=20)

ax[1].imshow(car\_kmeans)

ax[1].axis('off')

ax[1].set\_title('clustered', size=20)

plt.show()

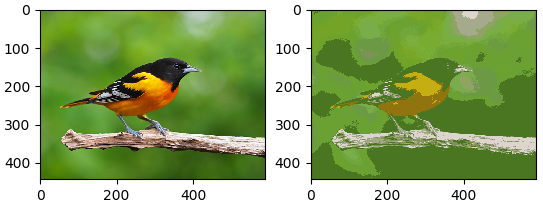


Рисунок 3 – собственное изображение до (слева) и после (справа) кодирования цвета пикселей с помощью к-средних с 16 кластерами с 1 итерацией

**11. Реализуйте алгоритм иерархической кластеризации на том же изображении. Сравните полученные результаты.**

file\_path = 'bird-small.png'

img = imageio.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

K-means определяет кластеры итеративно сходясь в локальном минимуме пресделую optimization objective. Этот алгоритм, даже при удачных начальных кластерах не всегда работает.

В отличии от K-means, hierarchical clustering алгоритм в данном примере выше не допустит ошибки. Hierarchical clustering разделяют на кластеры по top-down and bottom-up подходам. В bottom-up подходе происходит соединение 2 близжайших элементов.

В текущем примере оба алгоритма отработали схоже, успешно сжав изображение.

**Вывод**

Кластеризация — задача группировки множества объектов на подмножества (кластеры) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров по какому-либо критерию. Задача кластеризации относится к классу задач обучения без учителя. Часто в алгоритмах машинного обучения возникает необходимость классифицировать данные. Каждый объект данных представляется как вектор (точка) в p-мерном пространстве (упорядоченный набор p чисел). Каждая из этих точек принадлежит только одному из двух классов. Вопрос состоит в том, можно ли разделить точки гиперплоскостью размерности ( p−1). Это — типичный случай линейной разделимости. Искомых гиперплоскостей может быть много, поэтому полагают, что максимизация зазора между классами способствует более уверенной классификации. То есть, можно ли найти такую гиперплоскость, чтобы расстояние от неё до ближайшей точки было максимальным. Это эквивалентно[1] тому, что сумма расстояний до гиперплоскости от двух ближайших к ней точек, лежащих по разные стороны от неё, максимально. Если такая гиперплоскость существует, она называется оптимальной разделяющей гиперплоскостью, а соответствующий ей линейный классификатор называется оптимально разделяющим классификатором.