**Проектная работа №2. Сжатие изображений, используя алгоритм кластеризации методом К-средних (Режим обучения: Без учителя)**

Техническая документация: <https://www.analyticsvidhya.com/blog/2020/10/a-simple-explanation-of-k-means-clustering/>

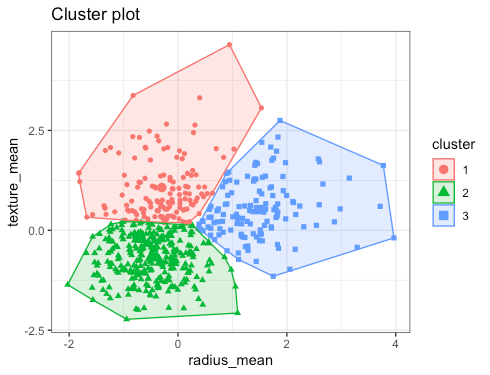
Кластеризация K-средних – один из наиболее широко используемых алгоритмов неконтролируемого машинного обучения, который формирует кластеры данных на основе сходства между экземплярами данных. Для работы этого конкретного алгоритма необходимо заранее определить количество кластеров. K в K-средних означает количество кластеров. Алгоритм K-средних начинается со случайного выбора значения центроида для каждого кластера.

После этого алгоритм итеративно выполняет три шага:

1. Найти евклидово расстояние между каждым экземпляром данных и центроидами всех кластеров.
2. (Назначьте экземпляры данных кластеру центроида с ближайшим расстоянием.
3. Вычислить новые значения центроидов на основе средних значений координат всех экземпляров данных из соответствующего кластера.

Метод k-средних (k-Means Clustering) – это очень известный и мощный алгоритм Обучения без учителя (Unsupervised Learning), который группирует похожие элементы в k кластеров. Он используется для решения многих сложных задач Машинного обучения (ML).

Пример. Предположим, мы пошли в магазин за овощами и увидели, что они будут расположены на полках по типу. Вся морковь хранится в одном месте, картошка – в другом.



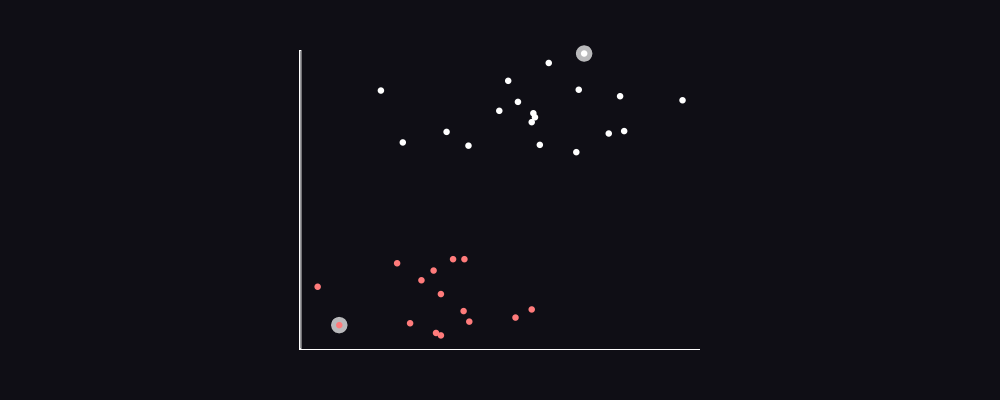
До применения кластеризации (появления окрашенных зон и обозначения записей разными иконками) перепутать категорию довольно легко. Неопытные мерчендайзеры до сих пор кладут арбузы в отдел ягод, хоть и правы с научной точки зрения.

Метод k-средних пытается сгруппировать похожие элементы в три этапа:

1. Выберем значение k
2. Инициализируем центроиды (разделительные линии)
3. Выберем группу и найдем среднее значение расстояния между точками.

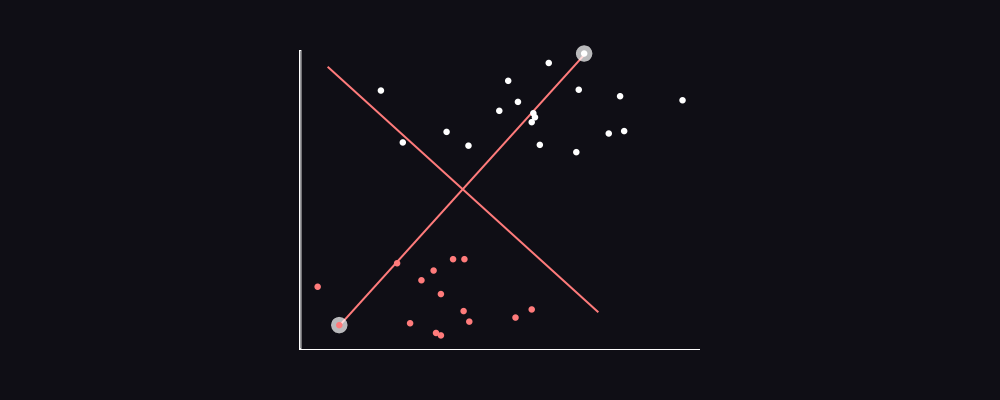
Давайте разберемся в вышеуказанных шагах с помощью иллюстраций. Допустим, мы на глаз кластеризовали наблюдения, причислив половину к белой категории, оставшуюся часть – к розовой.

Шаг 1. Мы случайным образом выбираем значение K, равное 2:



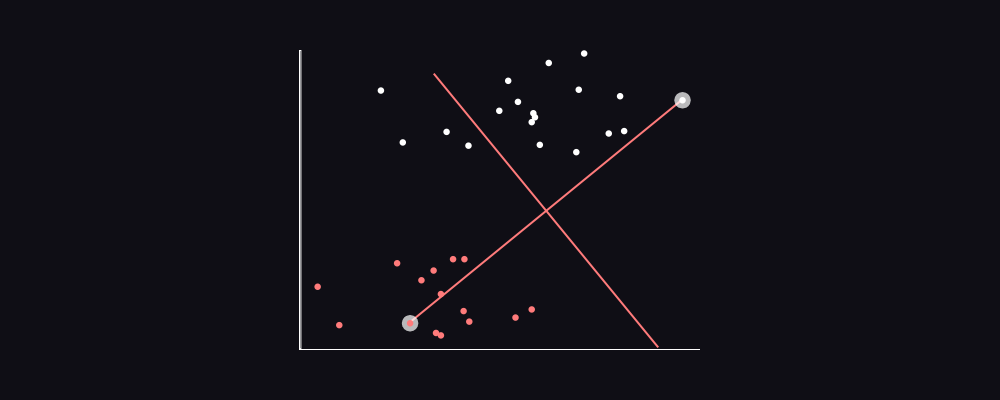
Существуют различные методы, с помощью которых мы можем выбрать правильные значения параметра k.

Шаг 2. Соединим две выбранные максимально удаленные точки, обозначенные белой полупрозрачной обводкой. Теперь, чтобы определить центроид, мы построим перпендикуляр к этой линии:

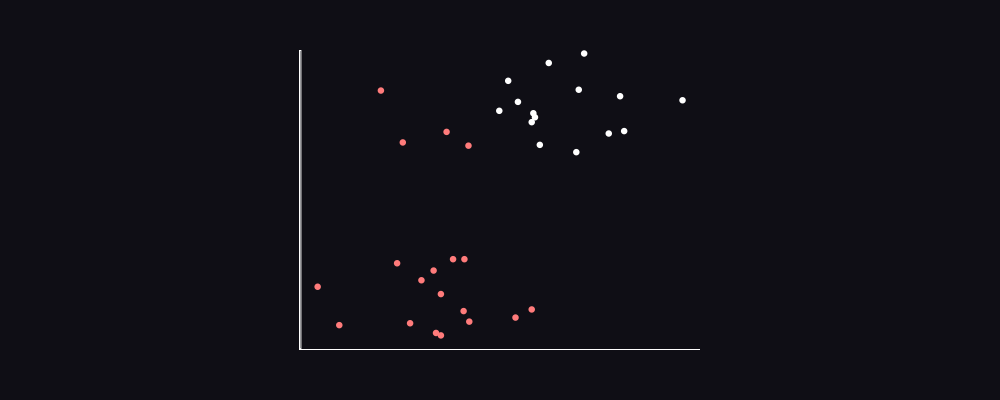


Если вы заметили, одна белая точка попала в группу розовых, и теперь относится к другой группе, чем предположено изначально.

Шаг 3. Мы соединим две другие удаленные точки, проведем к ним перпендикулярную линию и найдем центроид. Теперь некоторые белые точки преобразуются в розовые:



Этот процесс будет продолжаться до тех пор, пока мы не переберем все возможные сочетания пар дистанцированных точек и не уточним границы кластеров. Стабильность центроидов определяется путем сравнения абсолютного значения среднего Евклидова расстояния (Euclidian Distance) между наблюдениями и их соответствующими центроидами с пороговым значением.



***Как выбрать значение k?***

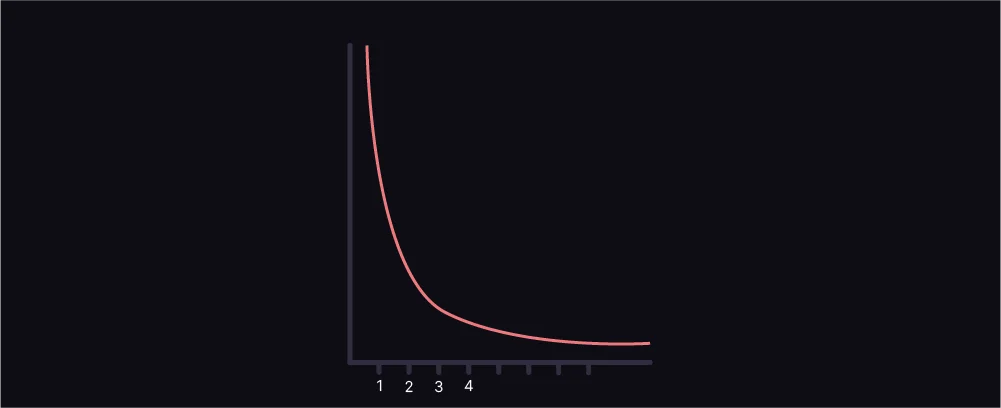
Одна из самых сложных задач в этом алгоритме кластеризации – выбрать правильные значения k. Существует два метода:

**Метод локтя**

Метод локтя (Elbow Rule) – один из самых известных методов, с помощью которого вы можете выбрать правильное значение k и повысить производительность Модели (Model). Этот эмпирический метод вычисляет сумму квадратов расстояний между точками и вычисляет Среднее значение (Mean).

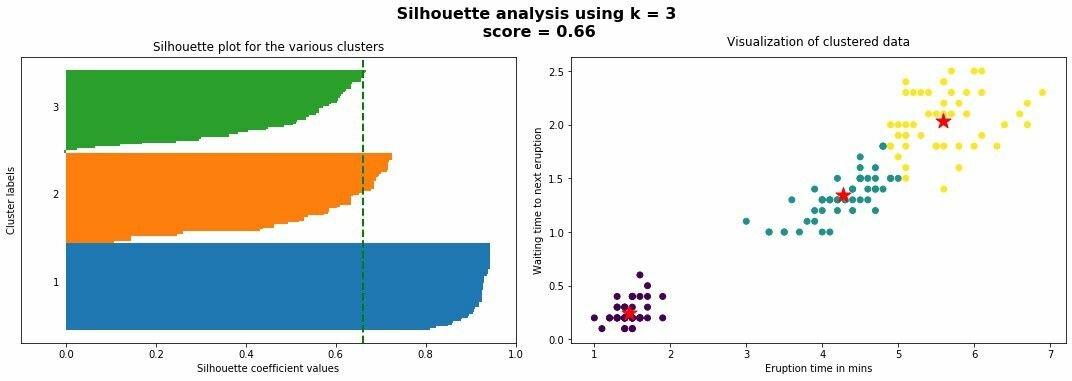
Когда значение k равно 1, сумма квадрата внутри кластера будет большой. По мере увеличения значения k сумма квадратов расстояний внутри кластера будет уменьшаться.

Наконец, мы построим график между значениями k и суммой квадрата внутри кластера, чтобы получить значение k. Мы внимательно рассмотрим график. В какой-то момент значение по оси x резко уменьшится. Эта точка будет считаться оптимальным значением k:



**Метод силуэта**

Метод силуэта (Silhouette Method) вычисляет среднее расстояние между точками в своем кластере a(i) и среднее расстояние от точек до следующего ближайшего кластера, называемого b(i).



Теперь мы можем вычислить коэффициент силуэта всех точек в кластерах и построить график. Последний также поможет в обнаружении [Выбросов (Outlier)](https://www.helenkapatsa.ru/vybros/). Значение метрики силуэта находится в диапазоне от -1 до 1. Обратите внимание, что коэффициент силуэта, равный –1 – это наихудший сценарий. Для картинки выше система вычислила расстояния между всеми точками при различных допущениях о числе кластеров и построила соответствующие горизонтальные гистограммы. Мы выбираем k, равный 3, потому что зеленая гистограмма меньше, хотя стоит, возможно, проверить и бо́льшие значения.

**Преимущества K-Means:**

* Простота реализации
* Масштабируемость до огромных наборов данных
* Метод очень быстро обучается на новых примерах
* Поддержка сложных форм и размеров.

**Недостатки K-means:**

* Не гарантируется достижение глобального минимума суммарного квадратичного отклонения *V*, а только одного из локальных минимумов.
* Результат зависит от выбора исходных центров кластеров, их оптимальный выбор неизвестен.
* Число кластеров надо знать заранее.

**Применение для задач глубокого обучения и машинного зрения**

В алгоритмах глубокого обучения метод k-средних иногда применяют не по прямому назначению (классификация разбивкой на кластеры), а для создания так называемых фильтров (ядер свёртки, словарей). Например, для распознавания изображений в алгоритм k-средних подают небольшие случайные кусочки изображений обучающей выборки, допустим, размером 16х16 в виде линейного вектора, каждый элемент которого кодирует яркость своей точки. Количество кластеров k задается большим, например 256. Обученный метод k-средних при определенных условиях вырабатывает при этом центры кластеров (центроиды), которые представляют собой удобные базисы, на которые можно разложить любое входное изображение. Такие "обученные" центроиды в дальнейшем используют в качестве фильтров, например для свёрточной нейронной сети в качестве ядер свёртки или других аналогичных систем машинного зрения. Таким образом осуществляется обучение без учителя при помощи метода k-средних.

**Использование алгоритма K-means для сжатия изображений**

Необходимые библиотеки:

**from** sklearn.cluster **import** KMeans  
**import** cv2  
**import** numpy **as** np  
**import** imutils

Ввод исходного изображения:

img = cv2.imread(**'111.jpg'**) *# Вводим исходное изображение*print(**"Img's shape: "**, img.shape) *# Выводим параметры начального изображения*

Параметры исходного изображения выводятся в формате X×Y×Ch, где X количество пикселей по горизонтали, Y – количество пикселей по вертикали, Ch – количество каналов (всегда 3 для цветных изображений RBG).

Нам нужно произвести нормализацию изображения в трехмерный вектор из исходного диапазона (0,255) к виду (0,1), а затем преобразовать его в X\*Y×3 так, чтобы каждая строка представляла пиксели, а три столбца - значения красного, зеленого и синего цветов.

img\_r = (img / 255.0).reshape(-1,3) *# Нормализуем изображение*print(**"New shape (img\_r): "**, img\_r.shape) *# Выводим параметры нормализованного изображения*

Установим необходимое количество цветов для сжатого изображения (чем меньше цветов, тем хуже качество). Количество цветов присваиваем параметру n\_clusters (кол-во кластеров).

k\_colors = KMeans(n\_clusters=4).fit(img\_r) *# Устанавливаем необходимое количество цветов и запускаем процесс обучения (кластеризации пикселей по группам)*

Каждая строка k\_colors.cluster\_centers\_представляет собой значение RGB центроида кластера. Параметр k\_colors.labels\_ содержит названия классов - по умолчанию в виде порядковых номеров соответствующих центров (k\_colors.cluster\_centers). Мы, в свою очередь, заменяем исходный цвет каждого пикселя новым цветом, определенным в параметре k\_colors.cluster\_centers\_.

imgC = k\_colors.cluster\_centers\_[k\_colors.labels\_]

Далее приводим сжатое нормализованное изображение в исходный вид:

imgC = (np.reshape(imgC, (img.shape))\*255).astype(np.uint8) *# Возвращаем изображению исходные параметры*print(**"Compressed image's shape: "**, imgC.shape)

Затем выводим исходное и сжатое изображения:

cv2.imshow(**"Original image"**, imutils.resize(img, width=800)) *# выводим исходное изображение*cv2.imshow(**"Compressed image"**, imutils.resize(imgC, width=800)) *# выводим сжатое изображение*cv2.waitKey(0)

После этого сохраняем сжатое изображение в папку с программой:

cv2.imwrite(**'Compressed.jpg'**, imgC) *# сохранение сжатого изображения*

Как следует из определения, процесс кластеризации в данной программе будет продолжаться до тех пор, пока мы не переберем все возможные сочетания пар дистанцированных точек, таким образом количество итераций при обработки изображения будет пропорционально увеличиваться в зависимости от его разрешения и количества кластеров.

Следовательно, программа будет выполняться довольно долго при указании большого количества кластеров и высоком разрешении исходного изображения.

Для ускорения процесса ниже представлена реализация алгоритма K-means в виде отдельной функции, с возможностью вручную указать количество итераций.

Реализация вышеуказанного алгоритма K-means c возможностью выбора количества итераций:

**import** numpy **as** np  
**import** matplotlib  
**from** PIL **import** Image  
  
**def** kmeans(Input,K,Max\_iters):  
   
 N,D = np.shape(Input)  
 R = np.random.permutation(N)  
 Kvec = Input[R[0:K],:]  
 Distance = np.zeros((N,K))  
   
 **for** nn **in** range(0,Max\_iters):  
 F = np.zeros((N,K))  
 **for** kk **in** range(0,K):  
 Distance[:,kk] = np.sum( np.square(Input - np.tile( Kvec[kk,:], (N,1)),dtype=np.float64),axis=1 )  
 Dmin = Distance.argmin(axis=1) % Distance.shape[1]  
 **for** mm **in** range(0,K):  
 **if** np.size(Dmin[mm==Dmin]) >0:  
 Kvec[mm,:] = np.mean(Input[mm==Dmin],axis=0)  
 **for** ii **in** range(0,N):  
 F[ii,Dmin[ii]] = 1  
 error = sum(sum((F\*Distance)/N))  
 print(**'Error = '** + str(error))  
 **return** Kvec,Dmin  
  
  
Datain = np.asarray(Image.open(**'input.jpeg'**),dtype=np.float64)  
  
ReshapedData = np.reshape(Datain, (np.size(Datain,0) \* np.size(Datain,1),np.size(Datain,2)))  
  
K = 10 # устанавливаем количество цветов  
Max\_iters = 15 # устанавливаем количество итераций, чем больше итераций, тем тщательнее будет осуществляться кластеризация и тем дольше будет работать программа  
  
Kvec,Dmin = kmeans(ReshapedData,K,Max\_iters)  
  
Dvec = np.zeros((len(Dmin),len(Kvec[0,:])))  
  
**for** jj **in** range(0,K):  
 Dvec[jj==Dmin,:] = Kvec[jj,:]  
  
imout = np.reshape(np.uint8(Dvec), (np.size(Datain,0),np.size(Datain,1),len(Kvec[0,:])))  
  
im = Image.fromarray(imout,**'RGB'**)  
im.show()  
im.save('output.jpeg', "JPEG", optimize = True)

**Задание**

1. Сжать исходное изображение используя метод K-means, встроенный в библиотеку Scikit-learn при следующих вариантах количества цветов: 2, 3, 4, 8, 16, 32, 64, 128, 256. Сохранить получившиеся изображения в отдельную папку (Scikit Compressor) и продемонстрировать их в отчёте.
2. Используя алгоритм, дающий возможность вручную выбрать количество итераций, подобрать оптимальное количество итераций по скорости выполнения и качеству сжатого изображения. После этого также следует произвести сжатие изображения с указанными выше количествами цветов. Сохранить получившиеся изображения в отдельную папку (K-means Compressor) и продемонстрировать их в отчёте.
3. Составить сравнительную таблицу со следующими данными: количество цветов, вес сжатого изображения по алгоритму из п.1, время выполнения программы для сжатия изображения по алгоритму из п.1, вес сжатого изображения по алгоритму из п.2., время выполнения программы для сжатия изображения по алгоритму из п.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество цветов (k) | *Scikit Compressor* | | *K-means Compressor* | |
| Информационный объем сжатого изображения (I) | Время выполнения программы | Информационный объем сжатого изображения (I) | Время выполнения программы |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 8 |  |  |  |  |
| 16 |  |  |  |  |
| 32 |  |  |  |  |
| 64 |  |  |  |  |
| 128 |  |  |  |  |
| 256 |  |  |  |  |

1. Подобрать оптимальное количество цветов и итераций для сжатия изображения без наглядного ухудшения качества изображения. Привести пример на пяти разных изображениях