Лабораторная работа №11. Алгоритм *K*-средних.

Задачей лабораторной работы является реализация алгоритма *K*-средних и использование его для сжатия цветных изображений. Теоретический материал для лабораторной работы можно найти в разделе 8 учебного пособия.

Сначала вы будете использовать небольшой набор данных из 300 точек с двумя координатами для программирования и проверки работы алгоритма. Когда все процедуры будут готовы, алгоритм *K*-средних будет использован для сжатия изображения из файла. Рекомендуется стараться реализовать процедуры алгоритма *K*-средних в векторизованной форме, избегая циклов (хотя бы циклов по выборке). Так программа будет работать значительно быстрее.

В лабораторной работе используется библиотека PIL для загрузки и сохранения изображений. Если на компьютере библиотека еще не установлена, выполните команду:

pip install pillow

В самом начале функции main в программе загружается набор данных из файла ex7data2.npy. Набор содержит 300 точек $\{x^{(1)} \dots x^{(m)}\} \in \mathbb{R}^2$ ($m = 300, n = 2$).

Далее в программе последовательно реализуются все необходимые процедуры для работы алгоритма *K*-средних.

1 Шаг присвоения кластеров

Во внутреннем цикле алгоритма *K*-средних первым делом выполняется присвоение кластеров точкам выборки. На данном шаге вы должны определить, какой из центроидов кластеров ближе к точке, и сформировать массив индексов для всех точек выборки.

Обратите внимание, что в теоретическом материале кластеры нумеруются от 1 до *K*. В языке Python принята индексация массивов на базе нуля, поэтому в программе кластеры нумеруются от 0 до (*K*-1).

Найдите в программе функцию assign_clusters и дополните ее кодом для реализации первого шага алгоритма *K*-средних. На вход функция принимает параметры:

X – матрица размером $m \times n$ точек выборки;

mu – матрица размером $K \times n$ центроидов кластеров.

Функция должна вернуть вектор C размером m индексов кластеров соответствующих точек, $c^{(i)} \in \{0 \dots K - 1\}$.

Когда функция готова, запустите программу. Она запускает написанную функцию для трех предопределенных центроидов и выводит индексы кластеров нескольких первых точек загруженной выборки. Если функция написана правильно, они должны совпадать с ожидаемыми значениями.

Удалите оператор return, чтобы перейти к следующему шагу.

2 Вычисление центроидов кластеров

Вторым шагом внутреннего цикла алгоритма *K*-средних выполняется перемещение центроидов кластеров в геометрический центр всех точек своего кластера. Вам нужно выбрать все точки, принадлежащие одному кластеру, и вычислить среднюю точку по ним.

Найдите в программе функцию compute_mean_centroids и дополните кодом. Функция принимает параметры:

X – матрица точек выборки;

C – вектор индексов кластеров каждой точки;

K – количество кластеров.

Функция должна вернуть матрицу mu размером $K \times n$ новых центроидов всех кластеров.

Когда функция готова, запустите программу. Она запускает написанную функцию для найденного на прошлом шаге вектора C и выборки X и выводит найденные центроиды. Если обе функции (assign_clusters и compute_mean_centroids) написаны правильно, центроиды должны совпадать с ожидаемыми.

Удалите оператор return, чтобы перейти к следующему шагу.

3 Начальная инициализация центроидов

В самом начале алгоритма *K*-средних выполняется начальная инициализация центроидов кластеров. Рекомендуется в качестве центроидов выбирать K случайных точек из кластеризуемой выборки.

Найдите в программе функцию init_centroids и дополните рабочим кодом. Функция принимает параметры:

X – матрица точек выборки;

K – количество кластеров.

Функция должна вернуть матрицу init_mu размером $K \times n$ центроидов кластеров.

Когда функция готова, запустите программу. Она запускает функцию и проверяет, что начальная инициализация центроидов верна. В этом случае пишет «Инициализация верная». Если вы получили сообщение «Ошибка инициализации», перепроверьте код функции.

Удалите оператор return, чтобы перейти к следующему шагу.

4 Алгоритм К-средних

К данному моменту все готово, чтобы реализовать алгоритм *K*-средних. В алгоритме в цикле повторяется два действия: присвоение кластеров, вычисление центроидов кластеров. Цикл заканчивается, когда на двух последующих его итерациях не произошло изменения ни одного из центроидов кластеров.

Все эти присвоения кластеров и вычисления центроидов вами уже реализованы. Найдите в программе функцию kmeans и дополните ее кодом для верного вызова написанных ранее функций и реализации алгоритма *K*-средних.

Функция kmeans принимает параметры:

X – матрица точек выборки;

initial_mu – начальная инициализация центроидов.

Функция должна вернуть:

C – вектор индексов кластеров каждой точки после завершения кластеризации;

mu – матрица центроидов кластеров после завершения кластеризации.

Когда функция готова, запустите программу. Она запустит вашу функцию kmeans и выведен найденные центроиды кластеров. На текущий момент мы еще не проверим ее работу.

Удалите оператор return, чтобы перейти к следующему шагу.

5 Отображение результата кластеризации

На данном шаге вам необходимо реализовать отображение результата кластеризации. Найдите в программе функцию draw_clusters и дополните ее кодом. Функция принимает параметры:

X – матрица точек выборки;

C – вектор индексов кластеров точек;

mu – центроиды кластеров.

Функция не возвращает никаких значений, вместо этого она должна вывести на

экран точки выборки, каждый кластер различными цветами. Отдельно нужно вывести центроиды каждого кластера. Рекомендуется также вывести легенду графика, заголовок, подписать оси.

Когда функция готова, запустите программу. Если функции kmeans и draw_clusters реализованы верно, вы должны увидеть результат кластеризации. Примеры графиков показаны на рис. 1.

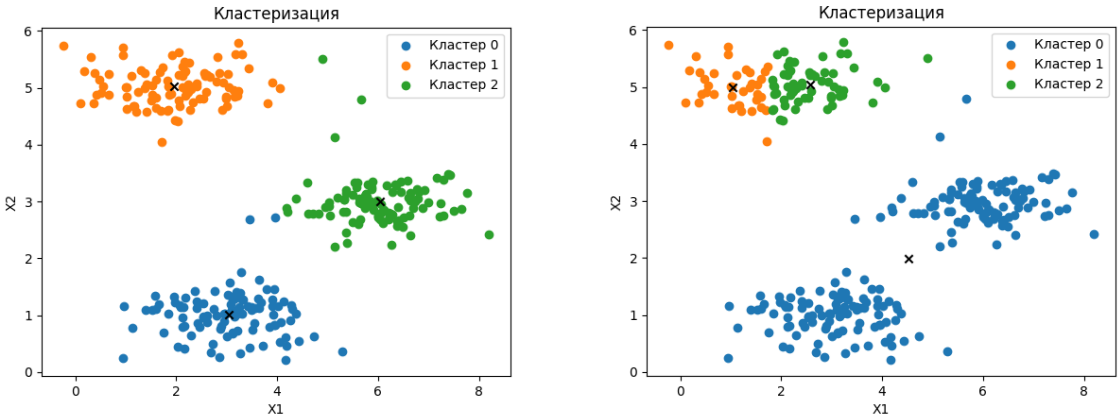

Рис. 1. Результаты кластеризации.

Удалите оператор return, чтобы перейти к следующему шагу.

6 Функция ошибки *K*-средних

На данном шаге вам необходимо реализовать вычисление функции ошибки кластеризации. Она вычисляется как средний квадрат расстояния от всех точек выборки до центроидов кластеров, которым они принадлежат.

Найдите в программе функцию distortion_function и дополните ее кодом для вычисления ошибки кластеризации.

Функция distortion_function принимает параметры:

X – матрица точек выборки;

C – вектор индексов кластеров точек;

mu – центроиды кластеров.

Функция должна вернуть J – значение функции ошибки от заданных параметров.

Когда функция готова, запустите программу. Она запустит вашу функцию для предопределенных центроидов и индексов кластеров и выводит вычисленное ею значение. Если все реализовано правильно, значение должно совпадать с ожидаемым.

Удалите оператор return, чтобы перейти к следующему шагу.

7 Оптимальная кластеризация

Известно, что результат алгоритма *K*-средних сильно зависит от начальной инициализации центроидов. Если она удачная, то можно получить хорошую кластеризацию с малым значением функции ошибки. При неудачной начальной инициализации кластеризация может быть плохой, и ошибка будет существенно больше.

Общепринятым простым приемом является запуск алгоритма *K*-средних множество раз и выбора из всех результатов того, который имеет наименьшее значение функции ошибки.

Найдите в программе функцию iterative_kmeans и дополните ее кодом для реализации данного подхода. Функция принимает параметры:

X – матрица точек выборки;

K – количество кластеров;

Q – количество повторений алгоритма.

Функция должна вернуть:

optimum_C – вектор индексов кластеров самого лучшего из результатов кластеризации;

optimum_mu – матрица центроидов кластеров самого лучшего результата.

Когда функция готова, запустите программу. Она выведет найденные итеративным алгоритмом центроиды. Если все было реализовано правильно, они должны совпадать с ожидаемыми значениями. Обратите внимание: порядок центроидов в массиве произволен и в каждом запуске процедуры может меняться, поэтому выведенный вашей функцией первый центроид может быть перечислен среди ожидаемых значений вторым или третьим. Еще важно, что итеративный алгоритм не дает гарантии нахождения оптимального решения. В очень редких случаях может получиться, что после всех итераций оптимальное решение не было найдено, и выведенные вашей функцией значения не совпадают с ожидаемыми. В таком случае перезапустите программу еще раз. Только если вы снова не получили желаемого результата, необходимо искать ошибку в программе.

После вывода центроидов программа отображает результат кластеризации на графике. Если все выполнено верно, вы должны получить хорошее решение с четко разделимыми тремя кластерами и минимумом ошибок.

Удалите оператор return, чтобы перейти к следующему шагу.

8 Использование *K*-средних для сжатия изображений

На данном шаге вам не нужно писать собственного кода. Программа использует написанную вами функцию iterative_kmeans для сжатия изображения.

Алгоритм *K*-средних может быть использован для сжатия изображений с потерями. Сжатие с потерями означает, что часть информации изображения теряется и качество изображения ухудшается. Но при этом файл изображения становится существенно меньше.

Принцип этого преобразования следующий. Цветная точка (пиксель) изображения на мониторе компьютера представляется комбинацией из трех базовых цветов: красного, зеленого и синего. Каждый из базовых цветов записывается в виде числа от 0 до 255 (8 бит) и получается, что один цвет может быть задан тремя 8-битными числами. Всего 24 бита, что образует 16 млн цветов (достаточно для изображения довольно высокого качества). Таким образом значение цвета может быть представлено в виде точки в трехмерном пространстве с осями R (красный), G (зеленых), B(синий) и интервалом значений от 0 до 255 по каждой оси.

Мы можем выбрать все пиксели одного изображения и сформировать выборку вида $\{x^{(1)} \dots x^{(m)}\} \in \mathbb{R}^3$, где m – количество пикселей изображения, $x^{(i)} = (r^{(i)}, g^{(i)}, b^{(i)})$, т. е. $n = 3$. Можно выполнить кластеризацию всех этих точек, и получить разбиение изображения на кластеры, в которые попадают близкие цвета (например, все оттенки красного могут попасть в один кластер).

Можно теперь заменить значения пикселей в изображении на значение центроида кластера. Качество изображения ухудшится (ведь в нем теперь некоторые различные оттенки заменены одним цветом), но размер его станет меньше. Большинство современных форматов хранения изображений делает файл тем меньшего размера, чем меньше в изображении различных цветов.

На данном шаге выполнения лабораторной работы загружается изображение из

файла формата PNG (файл bird_small.png) и представляется в виде выборки X размером $m \times 3$. Выборка передается написанной вами функции iterative_kmeans. Полученный результат кластеризации (C и mu) используется для генерации изображения, состоящего из цветов, равных центроидам найденных кластеров цветов исходного изображения.

Затем программа показывает на экране исходное и сжатое изображение (пример на рис. 2, кластеризация при K=16), а также сохраняет файл под названием bird_compressed.png. Полученный файл в несколько раз меньше исходного.

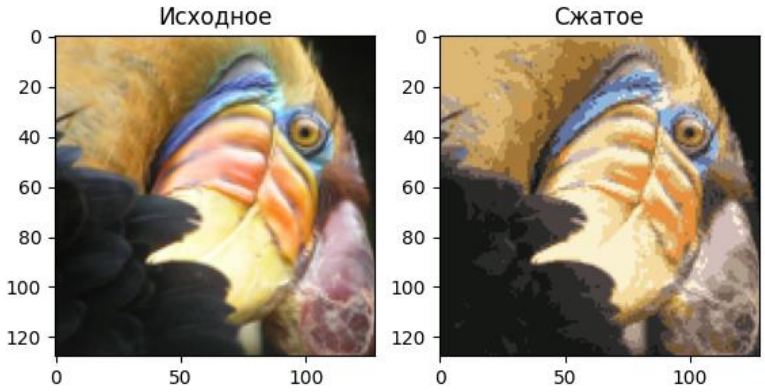

Рис. 2. Пример кластеризации изображения.

Если вы реализовали алгоритм *K*-средних в векторизованной форме, то вы быстро получите данный результат. Если же ваша программа содержит циклы, например, такого вида

```
for i in range(m):
```

то результат можно ждать долго. В таком случае можно порекомендовать уменьшить количество искомых кластеров и итераций (упростить задачу) или переписать алгоритм в векторизованной форме.

Самым простым вариантом будет упростить алгоритму задачу. Укажите в программе требуемое число кластеров K=2 .. 4 вместо 16 и Q=1 .. 2 вместо 10. Качество изображения будет существенно хуже, но результат будет получен гораздо быстрее.

Если вы решите добиться от вашей программы быстродействия, то учтите следующие рекомендации.

Оптимизируйте функции assign_clusters и compute_mean_centroids, именно на них тратится все время вычислений.

Избавьте от циклов по точкам выборки (циклов вида *for i in range*(*m*) и им подобных), так как в изображении точек более 16 тыс. Цикл по кластерам (до *K*) можно оставить, их в задаче мало.

Вместо вычисления расстояния от точки до центроида в цикле, продублируйте точку центроида *m* раз и отнимите всю выборку от полученной матрицы. Для дублирования используйте функции ***tile*** или ***repeat*** библиотеки NumPy.

Не вычисляйте квадрат и сумму для каждой точки по отдельности. Вы можете применить функции np.sum, np.argmin, np.power для всех точек выборки сразу и получить готовый вектор C без единого цикла.

Полностью векторизованная реализация выполняет 10 итераций разбиения тестового изображения на 16 кластеров за несколько секунд (зависит от производительности компьютера).

На этом выполнение лабораторной работы №11 завершается.