Задание № Б-3.1-1. Метод К-средних

Введение

В этом упражнении вы реализуете метод кластеризации под названием К-средних и примените его для сжатия изображения.

1. Кластеризация методом К-средних

Начнем работу с учебного двухмерного датасета, который позволит вам на интуитивном уровне понять, как работает алгоритм. После этого вы используете алгоритм для сжатия изображения путем уменьшения всей палитры цветов до наиболее часто используемых.

Скрипт `ex7.m` поможет вам с выполнением упражнения.

1.1 Реализация метода К-средних

Алгоритм К-средних позволяет автоматически группировать схожие данные в кластеры. Более конкретно, пусть нам дана обучающая выборка $\{x^{(1)}, \dots, x^{(m)}\}$, где $x^{(i)} \in \mathbb{R}^n$, и мы хотели бы сгруппировать элементы этой выборки в несколько «связных» кластеров. Идея метода К-средних заключается в следующем: сначала мы случайным образом выбираем позиции центроидов («центров тяжести» будущих кластеров), а затем итерационно уточняем эти позиции: сначала соотносим все примеры из выборки с ближайшими к ним центроидами, а затем переносим центроиды в центры тяжести множества соотнесенных с ними примеров.

Алгоритм К-средних выглядит следующим образом:

```
% Инициализируем центроиды
centroids = kMeansInitCentroids(X, K);
for iter = 1:iterations
      % Шаг назначения кластера: относим каждую точку к ближайшему
      % к ней центроиду. idx(i) соответствует c^(i) - индексу
      % центроида, к которому приписывается i-тый пример
      idx = findClosestCentroids(X, centroids);

      % Шаг перемещения центроидов: вычисляем центр тяжести (среднее значение)
      % по примерам, отнесенным к соответствующему центроиду
      centroids = computeMeans(X, idx, K);
end
```

Цикл алгоритма выполняет два шага: (1) относит каждый пример $x^{(i)}$ из обучающей выборки к ближайшему центроиду, и (2) перемещает центроиды в центры тяжести тех групп примеров, который соотнесены с ними. Алгоритм К-средних всегда сойдется к какому-то решению. Заметим, что полученное решение не всегда будет идеальным, так как оно зависит от начального положения центроидов. Поэтому, как правило, на практике этот алгоритм прогоняют несколько раз, каждый раз случайным образом инициализируя начальные позиции центроидов. Один из способов выбрать наилучший результат – это выбрать результат разбиения на кластеры с наименьшим значением функции стоимости (разброса).

В следующих секциях вы запрограммируете два шага данного алгоритма отдельно.

1.1.1 Нахождение ближайшего центроида

На шаге «назначения кластера» алгоритм для каждого примера из обучающей выборки находит ближайший к нему (с точки зрения евклидова расстояния) центроид. В частности, для i-го примера:

$$c^{(i)} := \underset{j}{\operatorname{argmin}} \left\| x^{(i)} - \mu_j \right\|^2,$$

где $c^{(i)}$ – это номер ближайшего к $x^{(i)}$ центроида, а μ_j – позиция (значение) j-го центроида. Заметим, что в коде упражнения $c^{(i)}$ соответствует `idx(i)`.

Ваша задача – завершить код в файле `findClosestCentroids.m`. Эта функция принимает на вход матрицу данных X и текущие позиции всех центроидов в массиве centroids, а должна выдать одномерный массив idx, содержащий номера (значения от 1 до K, где K – количество центроидов) ближайших центроидов для каждого примера из выборки X.

Вы можете реализовать этот код с помощью цикла, перебирающего все примеры.

После того как вы завершите, скрипт `ex7.m` выполнит ваш код, и вы должны увидеть результат [1 3 2], соответствующий назначениям центроидов для первых трех примеров из выборки.

1.1.2 Перемещение центроидов

Получив распределение всех примеров из выборки по центроидам, на втором шаге алгоритм вычисляет среднее значение по всем примерам своей группы и перемещает в него (присваивает ему это значение) соответствующий центроид. В частности, для k-го центроида:

$$\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)},$$

где C_k – множество примеров выборки, отнесенных к k-му центроиду. Более конкретно, если, например, образцы $x^{(3)}$ и $x^{(5)}$ были отнесены к центроиду с $k = 2$, то его позиция будет обновлена следующим образом: $\mu_2 = \frac{1}{2}\left(x^{(3)} + x^{(5)}\right)$.

Вам необходимо завершить код в файле `computeCentroids.m`. Вы можете реализовать этот код с помощью цикла по центроидам. Вы также можете использовать цикл по примерам из обучающей выборки, хотя если вам удастся векторизовать свой код, его выполнение станет намного быстрее.

1.2 Применение метода K-средних на примерном датасете

После того как вы допишете код двух функций (findClosestCentroids и computeCentroids), следующим шагом скрипт `ex7.m` выполнит данный алгоритм на примерном датасете и визуализирует результат, чтобы вы смогли лучше понять, как работает алгоритм. Ваши функции будут вызываться из файла runKmeans.m. Рекомендуем вам ознакомиться с его кодом.

После запуска скрипт `ex7.m` пошагово визуализирует процесс нахождения кластеров. Нажимайте enter и наблюдайте за тем, как алгоритм назначает примеры ближайшим центроидам, а затем перемещает эти центроиды в центры тяжести группы соотнесенных с ним примеров. В конце алгоритма график должен выглядеть так, как на Рисунке 1.

1.3 Рандомная инициализация

Начальная инициализация центроидов в предыдущем разделе была детерминированной специально, чтобы вы увидели такой же результат, как на Рисунке 1. На практике же обычно центроиды инициализируют значениями случайных примеров из обучающей выборки.

В этой части упражнения вам необходимо добавить в файл `kMeansInitCentroids.m` следующие строчки:

```
% Инициализируем центроиды случайными примерами из выборки
% Случайным образом перемешаем индексы примеров
randidx = randperm(size(X, 1));
% Выберем первые K примеров в качестве центроидов
centroids = X(randidx(1:K), :);
```

Вышеприведенный код сначала перемешивает список индексов примеров обучающей выборки (с помощью функции randperm). Затем он выбирает первые K примеров и присваивает их значения центроидам. Это гарантирует, что никакие два центроида не будут равны друг другу.

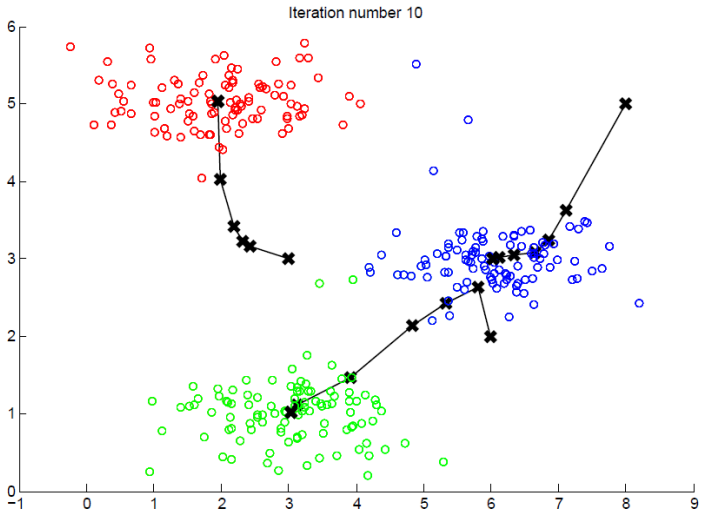

Рисунок 1: Результат кластеризации методом K-средних

1.4 Сжатие изображений с помощью алгоритма K-средних

В этой части упражнения вы примените метод K-средних для сжатия картинки, изображенной на Рисунке 2.

Рисунок 2: Исходное изображение, размер 128x128

В стандартной 24-битной цветовой схеме на каждый пиксель изображения приходится три целых 8-битных числа (со значениями от 0 до 255), обозначающих интенсивность красного, зеленого и синего цветов. Эта цветовая схема обычно называется RGB. Наше исходное изображение использует тысячи различных цветов, а вы сократите это число до 16. После этого мы сможем существенным образом сократить размер картинки. А именно, каждый пиксель теперь можно представить порядковым номером цвета в палитре (так как цветов 16, то для этого достаточно использовать 4 бита).

В этом упражнении вы с помощью алгоритма K-средних выберите 16 цветов, которые будут использоваться в сжатом изображении. Более конкретно, вы будете интерпретировать каждый пиксель как точку в трехмерном пространстве и с помощью метода K-средних найдете 16 цветов, которые наилучшим образом группируют (кластеризуют) все пиксели изображения. После этого вы замените все пиксели исходного изображения на значения ближайших к ним центроидам из 16 найденных.

1.4.1 Метод K-средних на пикселях

В Octave/Matlab изображение можно загрузить из файла следующим образом:

```
% Загружаем картинку из файла
A = imread('bird_small.png');
% У вас должен быть установлен пакет для работы с функцией imread
% Если по каким-то причинам этого пакета нет и установить его не представляется
% возможным, вы можете загрузить картинку с помощью следующей строки
%
% load('bird_small.mat'); % Загружаем изображение в переменную A
```

Этот код создает трехмерную матрицу A, в которой первые два измерения соответствуют позиции пикселя, а третье измерение соответствует цвету – красному, зеленому и синему. Например, A(50,33,3) дает интенсивность синего цвета пикселя, находящегося на пересечении 50-й строки и 33-го столбца.

Код скрипта `ex7.m` загружает изображение и трансформирует его в матрицу $m \times 3$, где $m = 16384 = 128 \times 128$, а дальше вызывает функцию алгоритма K-средних, которую вы написали.

После нахождения 16 цветов вы можете заменить каждый пиксель исходного изображения на ближайший к нему центроид (т.е. цвет). Заметим, что мы существенным образом сжали информацию. В исходном изображении каждый из 128×128 пикселей представлялся 24 битами информации о цвете, то есть вся картинка занимала $128 \times 128 \times 24 = 393216$ бит. Сжатое же изображение будет использовать всего 4 бита на каждый пиксель, что займет $128 \times 128 \times 4 = 65920$ бит. Это соответствует шестикратному сжатию. Результат можно посмотреть на Рисунке 3.

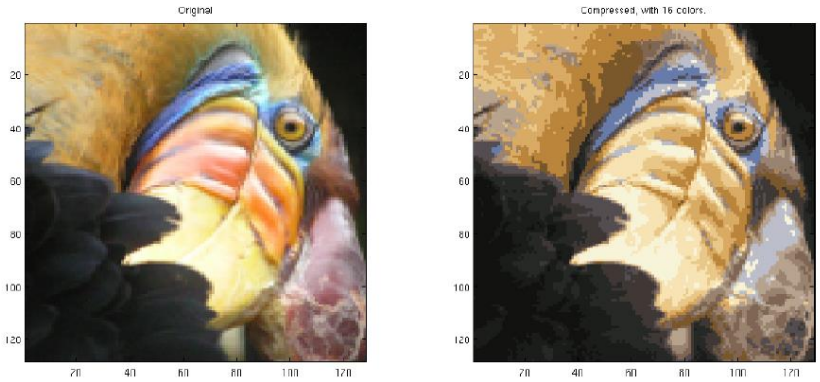

Рисунок 3: Исходное и сжатое с помощью метода K-средних изображения