Алгоритм кластеризации k-means: часть 2

Задача на практику

Крупный ученый, профессор Буковски, занимается изучением обезьян. Он направил своего ассистента в Африку, чтобы он собрал информацию о некоторых особях. Но нерадивый ассистент измерил только рост и вес некоторых обезьян, не собрав больше никаких данных. Профессору Буковски требуется определить, сколько видов обезьян удалось исследовать ассистенту.



Вот экспериментальные данные, собранные ассистентом:

№	Рост, см	Вес, кг	№	Рост, см	Вес, кг
1	30	10	11	160	63
2	145	40	12	48	25
3	197	140	13	15	29
4	175	130	14	205	152
5	81	139	15	67	139
6	157	47	16	75	115
7	132	55	17	12	18
8	28	31	18	55	123
9	87	122	19	156	153
10	141	127	20	135	32

Эта задача решается в 2 этапа.

На первом этапе необходимо дополнить функцию **`k_means`** так, чтобы она возвращала ошибку – средний квадрат расстояния от каждой точки до центра ближайшего кластера. Чем меньше эта ошибка, тем точнее выполнена кластеризация. В предельном случае ошибка будет минимальной, когда каждая точка будет представлять отдельный кластер.

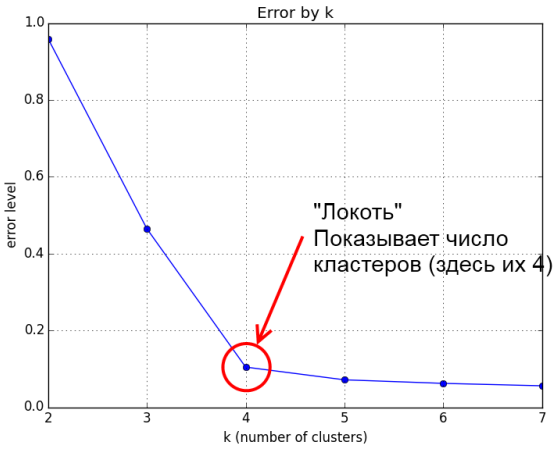
На втором этапе необходимо установить количество кластеров в имеющихся данных. Так как при различных запусках алгоритм кластеризации **`k-means`** может находить разные центры кластеров (из-за случайной инициализации центров в начале алгоритма), получающаяся ошибка кластеризации может быть различной. Поэтому алгоритм запускают многократно, и из множества запусков выбирают минимальный уровень ошибки, то есть самый точный вариант разбиения на кластеры. Эту процедуру

повторяют для разного количества кластеров. Ниже приведен псевдокод алгоритма определения числа кластеров и примерный вид графика зависимости ошибки от числа кластеров:

```
цикл по числу кластеров (напр., от 2 до 7)
  многократный запуск алгоритма k_means
  выбор минимальной ошибки среди запусков
конец цикла

построение графика: ошибка/число кластеров
определение числа кластеров по «локтю»
```

Псевдокод алгоритма определения числа кластеров



Зависимость ошибки от числа кластеров

На этом графике хорошо различима точка, называемая также «локоть», после которой уровень ошибки в зависимости от числа кластеров уменьшается незначительно. Следовательно, за число кластеров принимается то значение **k**, которое соответствует «локтю».

Решать задачу надо в такой последовательности:

1. в файле **run.py** сформировать матрицу **X**, количество столбцов которой равно 2. Первый столбец обозначает рост особи, второй – вес. Количество строк в матрице **X** соответствует количеству исследованных обезьян.

2. Для работы алгоритма кластеризации необходимо, чтобы значения признаков были одного порядка. У нас же ситуация иная – рост и вес обезьян лежат в разных диапазонах, поэтому необходимо их нормировать. Для нормировки используют как правило одну из двух формул:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

$$x_{new} = \frac{x - \mu}{\sigma}$$

Пример нормализации массива по второй формуле:

```
m = np.mean(X, axis=0)
s = np.std(X, axis=0)
X = (X - m) / s
```

3. В файле **k_means.py** требуется модифицировать функцию **k_means** так, чтобы она возвращала список центров кластеров на каждой итерации работы алгоритма. Этот список будет представлять собой трехмерную матрицу, в которой первый индекс будет обозначать номер итерации алгоритма, второй – номер кластера (от **1** до **k**), третий – номер признака (от **1** – рост до **2** – вес. Для этого может потребоваться использовать функцию **reshape** из библиотеки **numpy**. Необходимо познакомиться с тем, как она работает (команда **help('numpy.reshape')** и поэкспериментировать в консоли).

Возврат функцией нескольких значений может выглядеть так:

```
def k_means(k, X):
    # код функции
    return centers, all_centers, errors
```

Пример вызова функции:

```
centers, all_centers, errors = k_means(k, X)
```

4. Модифицировать функцию **k_means** так, чтобы она вычисляла ошибку на каждой итерации работы алгоритма. Ошибка вычисляется как средний квадрат расстояния от каждой точки до ближайшего центра кластера. Функция должна возвращать список ошибок на каждой итерации.

5. Запустить файл **run.py**, перейдя для этого в консоли **Python** в директорию с файлом и выполнив команду **run**. Для смены директории используется команда **os.chdir(<путь>)**, а для проверки текущей директории **os.getcwd()**. Выполнение этих команд требует предварительного подключения пакета **os** так: **import os**. Убедиться по первому графику, что с каждой итерацией работы алгоритма ошибка уменьшается. По второму графику убедиться, что центры кластеров с каждой итерацией сходятся к фактическим центрам кластеров.

6. Дополнить код вычисления минимальной ошибки для каждого значения **k** (количества кластеров). Для этого необходимо разобраться с псевдокодом, приведенным выше.

7. Отобразить график зависимости минимальной ошибки от количества кластеров. Найти «локоть», сделать вывод о том, сколько видов обезьян было исследовано вашим ассистентом.