

Конспект по теме «Кластеризация»

Задача кластеризации

Кластеризация — объединение похожих объектов в группы, или кластеры. Кластеризацию также называют кластерным анализом или сегментацией.

В большинстве методов кластеризации схожесть или различие объектов определяется расстоянием между ними. Чем дальше объекты друг от друга, тем меньше сходства и наоборот. Кластеризацию не стоит путать с задачей классификации, в которой заранее известны классы и принадлежащие им объекты. В кластеризации они не заданы и определяются разными способами. Они зависят от того, что считать похожими объектами.

В бизнесе кластеризация помогает:

- сегментировать пользователей или товары. Такая задача тесно связана с рекомендательными системами.
- выявлять мошенников по нетипичному поведению: например, накрутка кликов или лайков в соцсетях.

Алгоритм k-средних

Самый популярный алгоритм кластеризации — метод **k-средних** (англ. *k-means*). Он основан на понятии **центроида** (англ. *centroid*), или центра кластера. От степени близости к конкретному центру зависит, в какой кластер попадёт объект. У каждого кластера центроид свой, а вычисляется он как среднее арифметическое объектов кластера.

K-средних сегментирует объекты пошагово, поэтому это итеративный алгоритм. Разберём, как он работает для заданного числа кластеров k :

1. Каждому объекту алгоритм случайным образом присваивает номер кластера — от 1 до k .

2. Пока кластеры объектов не перестанут меняться, алгоритм повторяет итерацию из двух шагов:

- вычисляет центроид каждого кластера;
- каждому объекту присваивает номер нового кластера, центроид которого расположен ближе всего к объекту.

Другим условием остановки алгоритма может быть выполнение максимального количества итераций (**max_iter**).

Значение центроида (μ) вычисляется как среднее значение векторов x^j — произведение суммы всех векторов на число $1/N$, где N — количество векторов:

$$\mu = \frac{1}{N} \sum_{j=1}^N x^j = \frac{1}{N} \left(\sum_{j=1}^N x_1^j, \sum_{j=1}^N x_2^j, \dots, \sum_{j=1}^N x_n^j \right)$$

где первая координата полученного вектора — это сумма первых координат всех векторов, вторая — сумма вторых координат и далее до n (размерности векторов).

Для решения задачи кластеризации пригодятся приближённые признаки объекта в каждом сегменте. Они нужны, чтобы подсказать алгоритму, где искать кластеры. Значения этих признаков передадим на вход k -средних, чтобы задать **начальные центроиды** — это optionalный параметр.

Если на вход $k\text{-means}$ передать начальные центроиды, Python во время работы выдаст предупреждение *RuntimeWarning*. Чтобы этого избежать, добавим блокировку этого вывода **filterwarnings()**. Вернёмся к предупреждению позже, а пока в код задачи добавим такие строки:

```
import warnings  
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

Обучим алгоритм $k\text{-means}$:

```

from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)

# n_clusters - число кластеров
model = KMeans(n_clusters=3, random_state=12345)
model.fit(data)

# Получение центроидов кластеров
print("Центроиды кластеров:")
print(model.cluster_centers_)

# init - начальные центроиды
centers = np.array([[20, 80, 8], [50, 20, 5], [20, 30, 10]])
model = KMeans(n_clusters=3, init=centers, random_state=12345)
model.fit(data)

print("Центроиды кластеров для модели с начальными центроидами:")
print(model.cluster_centers_)

```

Целевая функция

В задачах обучения с учителем функция потерь показывает расхождение между предсказанием и правильным ответом. Когда разметки нет, о потерях говорить некорректно: неизвестно, какие ответы верные. Чтобы оценить качество модели в задачах обучения без учителя, нужна **целевая функция** (*objective function*).

Прежде чем перейти к определению целевой функции, повторим задачу обучения с учителем. Она формулируется так: найти параметры модели, при которых значение функции потерь на обучающей выборке минимально. Функция потерь — это частный случай целевой функции. Нас также интересует поиск её минимального значения, обучающая выборка есть.

Рассмотрим задачу алгоритма k -средних. Нужно распределить объекты таким образом, чтобы расстояние между ними внутри одного кластера (внутрикластерное расстояние) было минимальным. Целевая функция алгоритма определяется как сумма внутрикластерных расстояний, а задача обучения состоит в минимизации этой суммы.

Вычислим целевую функцию метода k -средних, но сперва заменим центроид для N объектов на центроид кластера в этой формуле:

$$\mu = \frac{1}{N} \sum_{j=1}^N x^j = \frac{1}{N} \left(\sum_{j=1}^N x_1^j, \sum_{j=1}^N x_2^j, \dots, \sum_{j=1}^N x_n^j \right)$$

То есть центроид каждого кластера (μ_k) вычисляется как среднее по всем объектам кластера:

$$\mu_k = \frac{1}{|C_k|} \sum_{x^j \in C_k} x^j = \frac{1}{|C_k|} \left(\sum_{x^j \in C_k} x_1^j, \sum_{x^j \in C_k} x_2^j, \dots, \sum_{x^j \in C_k} x_n^j \right)$$

- где $|C_k|$ — число точек в кластере;
- x — объекты, то есть векторы размером n ;
- \in — означает принадлежность объекта к кластеру.

Чтобы для каждого объекта найти ближайший центроид и присвоить ему номер нужного кластера, от объекта до центроида каждого кластера вычисляется евклидово расстояние d_2 . Затем из таких расстояний выбирается минимальное.

Формула записывается так:

$$\min_k d_2(\mu_k, x) = \min_k \sqrt{\sum_{i=1}^n (\mu_i^k - x_i)^2}$$

Значение k , при котором достигается минимум, и будет номером ближайшего кластера.

Когда объекту присвоен номер ближайшего кластера C_K , можно переходить к расчёту внутрикластерного расстояния. Оно вычисляется

как сумма квадратов расстояний от каждого объекта кластера C_k до центроида μ_k :

$$D_k = \sum_{x^j \in C_k} (d_2(\mu_k, x^j))^2$$

В начале работы алгоритма номера кластеров объектам присвоены случайно, а внутрикластерное отклонение большое: то есть все точки перемешаны, фактически кластеров нет. В конце работы алгоритма все объекты с одинаковым номером кластера объединены, и уже заметны границы кластера.

Итоговая целевая функция алгоритма вычисляется как сумма внутрикластерных отклонений:

$$SD = \sum_k D_k = \sum_k \sum_{x^j \in C_k} (d_2(\mu_k, x^j))^2$$

Для обученной модели кластеризации, значение целевой функции содержится в атрибуте `model.inertia_`.

Локальный минимум

Разберём *RuntimeWarning* подробнее. Если убрать блокировку, обучение модели выдаст такое предупреждение:

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3, init=centers, random_state=12345)
model.fit(data)
```

```
print("Целевая функция модели с начальными центроидами:")
print(model.inertia_)
```

```
Целевая функция модели с начальными центроидами:
74253.20363562103
/usr/local/lib/python3.6/site-packages/sklearn/cluster/k_means_.py:969:
RuntimeWarning: Explicit initial center position passed: performing only one init
in k-means instead of n_init=10
return_n_iter=True)
```

Блокировка вывода предупреждений `filterwarnings()` скрывает сообщение о том, что количество запусков алгоритма (`n_init`) с начальными центроидами стало равно единице.

Повторим: каждый запуск начинается с того, что алгоритм случайным образом присваивает объектам номер кластера. То есть получаем стартовый набор объектов определённого кластера. При следующем запуске этим же объектам присваиваются новые номера, а значение целевой функции по итогу работы алгоритма меняется.

То есть при каждом запуске алгоритма целевая функция получается минимальной для конкретного стартового набора объектов кластера. Такой минимум называется **локальным**. При запуске на другом стартовом наборе значение функции может стать иным. Это будет новым локальным минимумом.

Параметр `n_init` по умолчанию равен 10. Алгоритм запускается 10 раз с разными начальными кластерами. Из всех локальных минимумов выбирается наименьший.

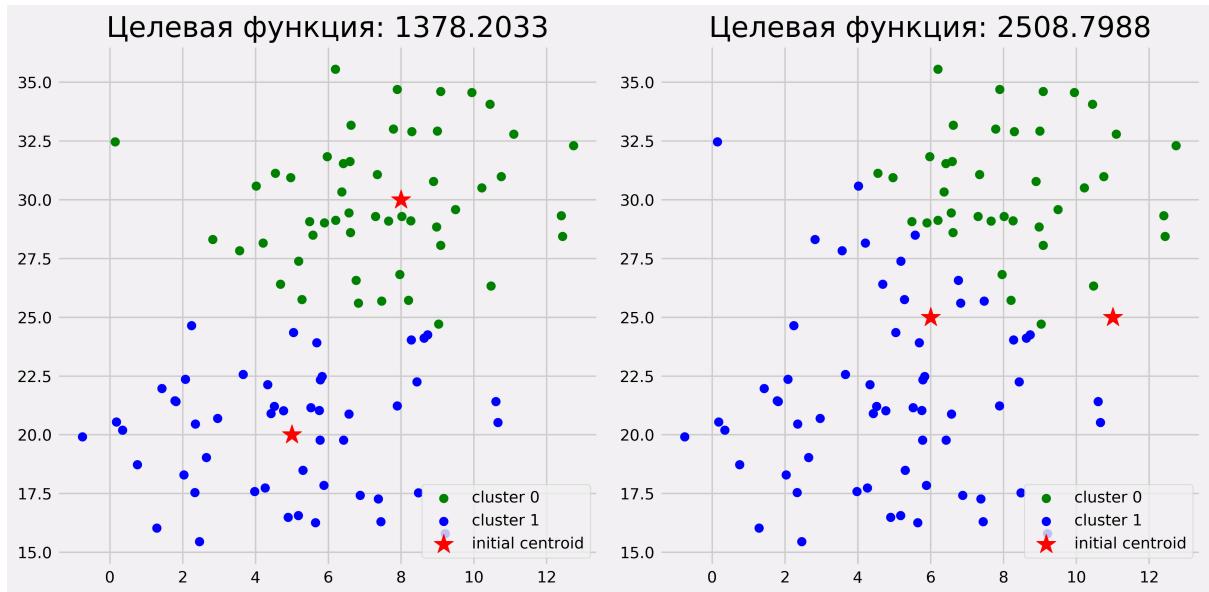
Подытожим. Целевая функция потеря с начальными центроидами получилась меньше, потому что алгоритм запускался всего один раз. При 10-кратном запуске случайно удалось найти начальные центроиды лучше.

Разберём на примере ещё один параметр — `max_iter`, определяющий количество итераций алгоритма. Чем их больше, тем ближе к локальному минимуму мы подойдём.

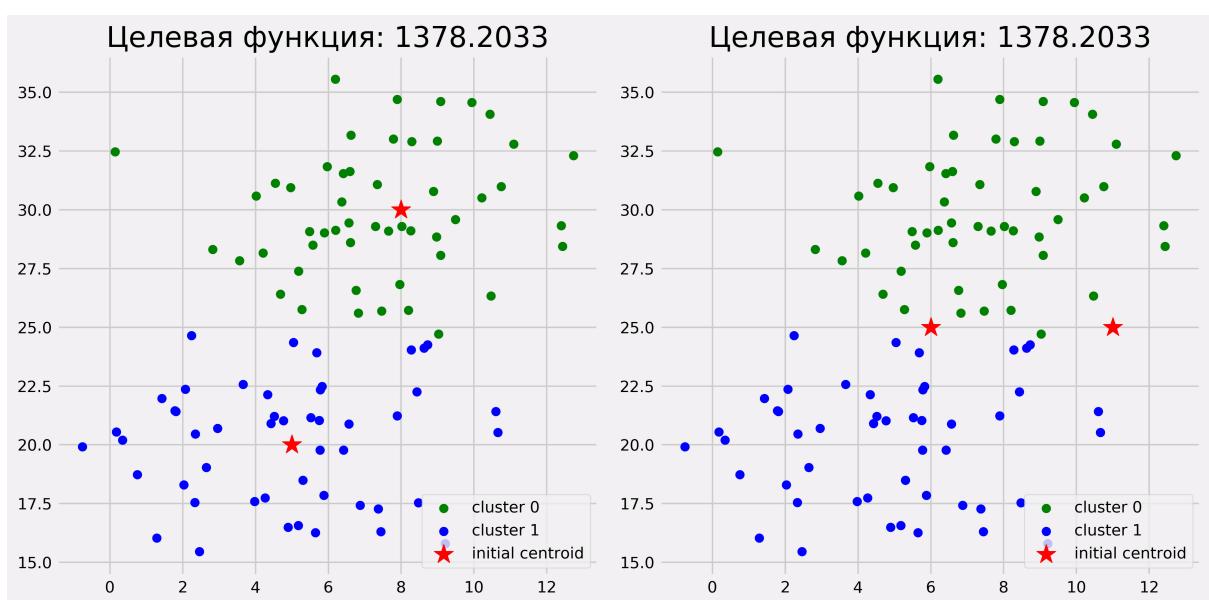
Перед вами результаты двух запусков алгоритма на синтетических данных с параметром `max_iter=1`. Данные сгенерированы как два облака

точек. (Мы заранее знаем, что это два кластера.) На графиках отмечены начальные центроиды: они-то и повлияли на распределение точек по кластерам.

Слева начальные центроиды попали примерно в центры кластеров. Справа начальные центроиды уже смешены от кластеров, сами кластеры также сдвинулись. Целевая функция больше значения слева.



С увеличением числа итераций до четырёх значения целевой функции становятся одинаковыми:

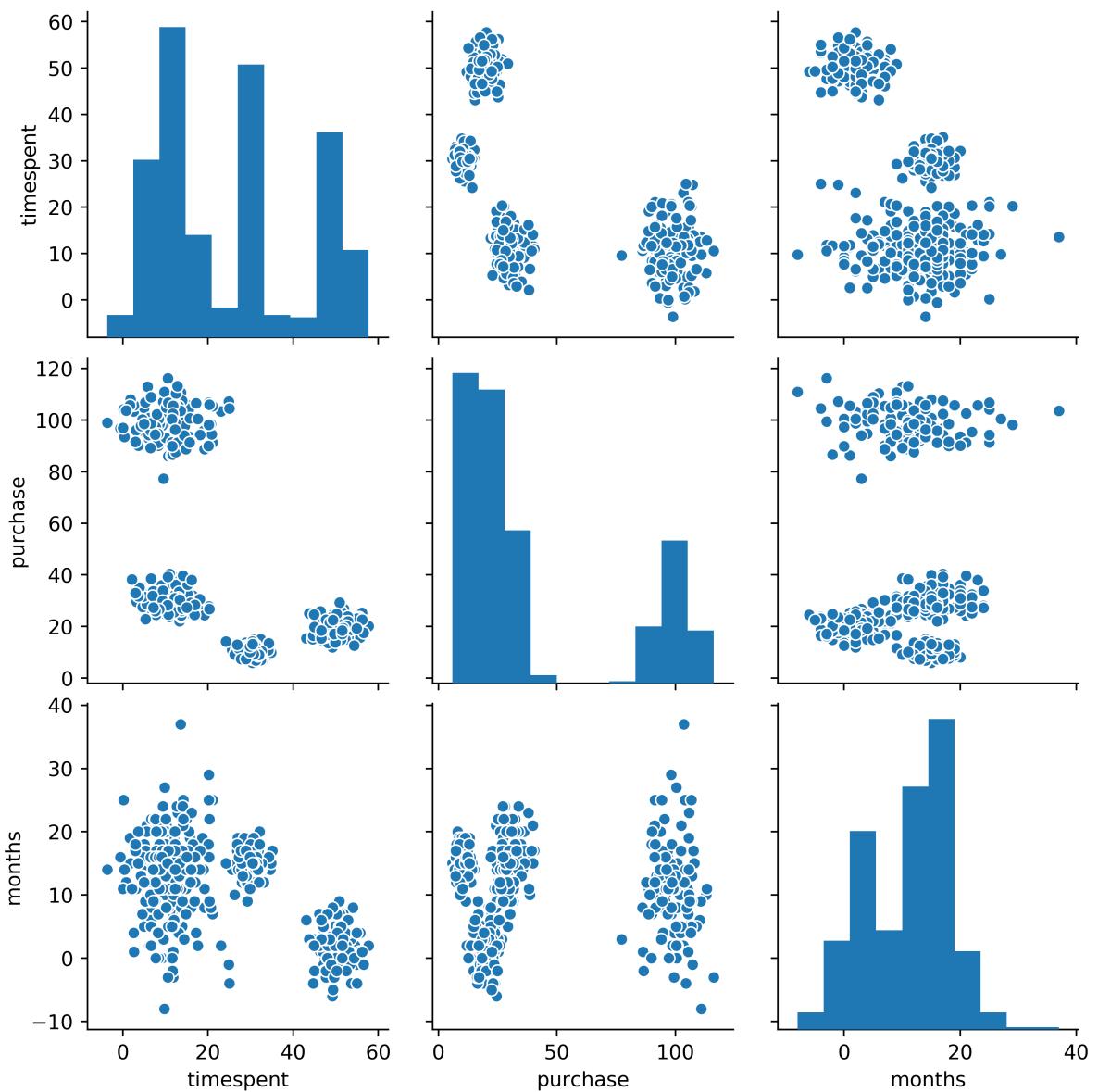


Число итераций по умолчанию равно 300 и в большинстве задач этого достаточно.

Визуализация

Для визуализации данных о пользователях сервиса построим график методом **pairplot** (англ. «парный график») из библиотеки *seaborn*. На диагонали находится распределение признака, в других ячейках — диаграммы рассеяния между всеми парами признаков. Тип графика по диагонали определяется параметром *diag_kind*:

```
import seaborn as sns
sns.pairplot(data, diag_kind='hist')
```



На графике изображены скопления точек — это будущие кластеры. Что интересного заметили?

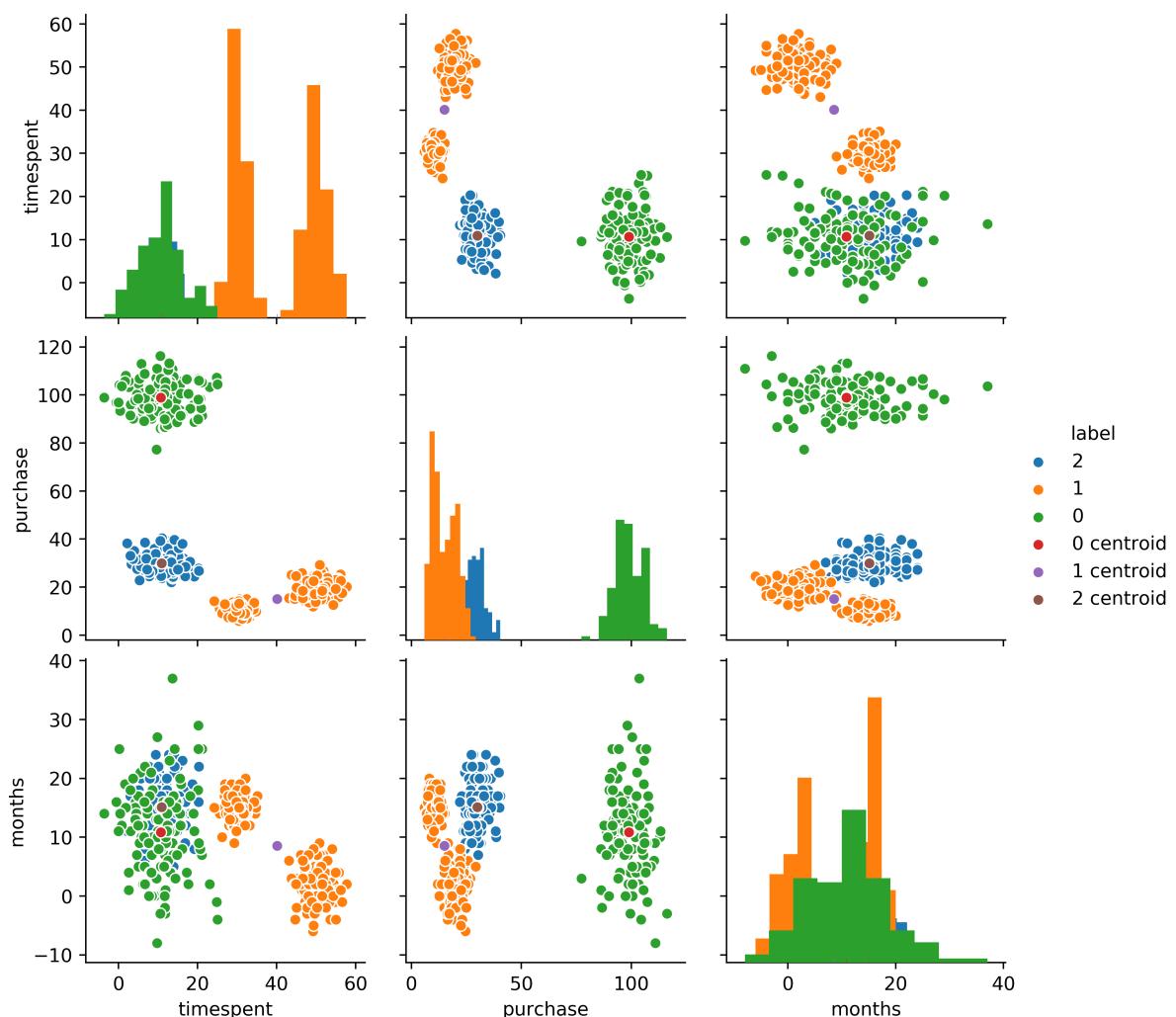
- Чётко выделяются четыре группы объектов на проекции пары признаков: *purchase* (средний чек) и *timespent* (среднее время сессии).
- На других проекциях видно по две-три группы точек.
- Скопление объектов с высокими значениями *purchase* заметно отделено от остальных точек.

Добавим заливку кластеров в график модели, обученной без начальных центроидов. Правила заливки задаёт параметр **hue**. Он принимает на вход массив из строковых переменных. Поэтому в массив строк переведём номера кластеров. А чтобы добавить на график центроиды, дадим им названия. Затем все данные объединим.

```
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sns

centroids = pd.DataFrame(model.cluster_centers_, columns=data.columns)
# Добавление столбца с номером кластера
data['label'] = model.labels_.astype(str)
centroids['label'] = ['0 centroid', '1 centroid', '2 centroid']
# Сброс индекса понадобится дальше
data_all = pd.concat([data, centroids], ignore_index=True)

# Построение графика
sns.pairplot(data_all, hue='label', diag_kind='hist')
```



Наши наблюдения подтвердились: группа точек с высокими значениями признака *purchase* попала в отдельный кластер. Мы определили премиальный сегмент.

С базовым и продвинутым сегментами сложнее. Чтобы определить подходящие им кластеры, на график добавим начальные центроиды дополнительным слоем. Для этого сохраним соответствующий графику объект — **PairGrid**:

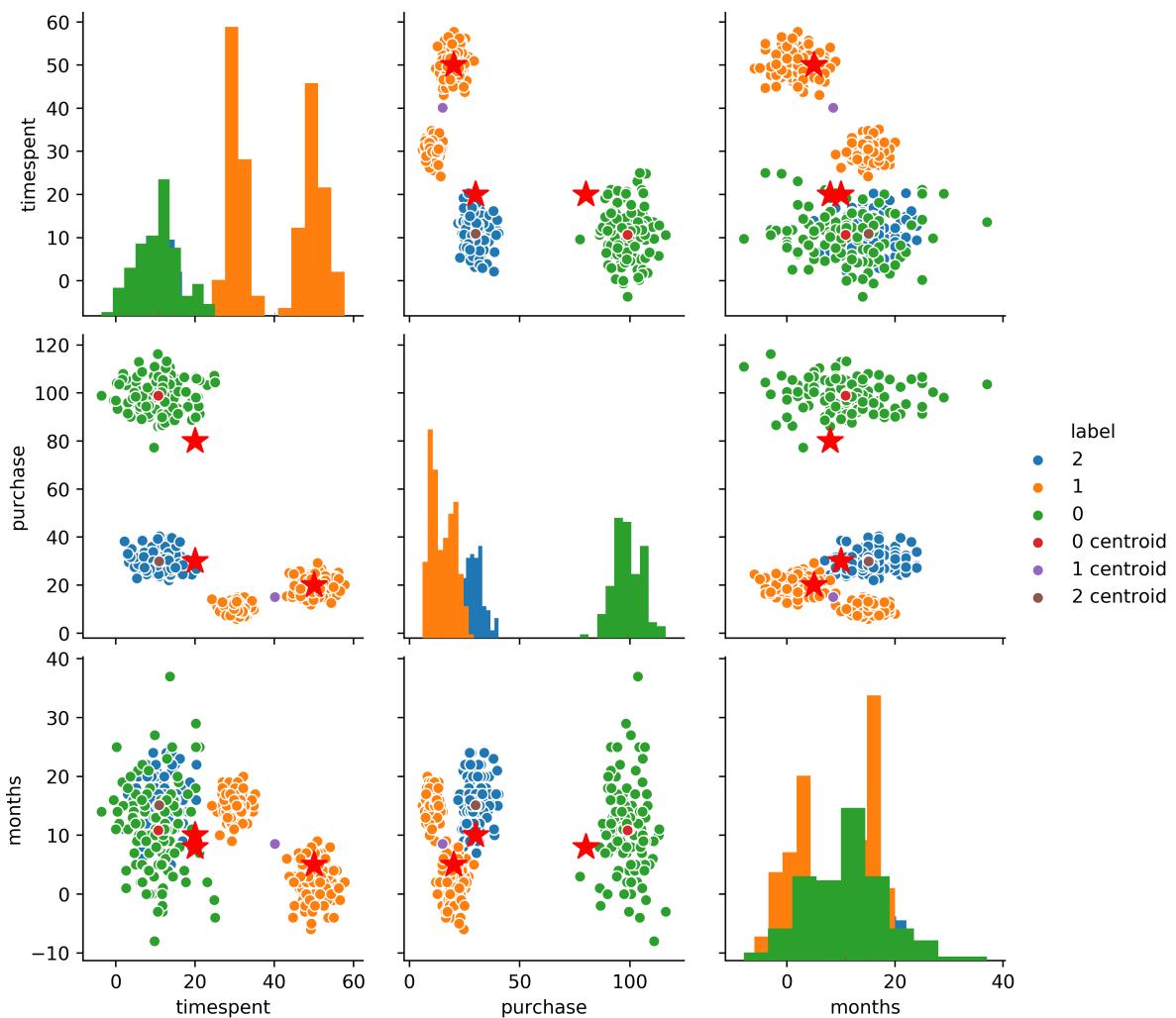
```
pairgrid = sns.pairplot(data_all, hue='label', diag_kind='hist')
```

Дополнительные значения для построения графика передадим через атрибут `data`:

```
pairgrid.data = pd.DataFrame([[20, 80, 8], [50, 20, 5], [20, 30, 10]], \
    columns=data.drop(columns=['label']).columns)
```

Вызовем ещё один метод — `map_offdiag`. Он строит данные из `pairgrid.data` на проекциях вне диагоналей. Параметр *func* определяет тип графика, *s* — размер (от англ. *size*), *marker* — форму точек, а *color* — цвет:

```
pairgrid.map_offdiag(func=sns.scatterplot, s=200, marker='*', color='red')
```



Оптимальное число кластеров

Целевая функция метода k -средних уменьшается с увеличением количества кластеров. Если у каждого объекта кластер отдельный, то внутрикластерное расстояние равно нулю. Так мы минимизируем целевую функцию, но смысла в такой кластеризации нет: кластеров по сути не будет.

Если признаков десятки и сотни, то графиком *pairplot* их охватить уже сложно. Также в наших данных изначально была структура, то есть видны скопления точек. В результате работы алгоритма они-то и стали кластерами. Но данные так чётко разделены не всегда. Поэтому познакомимся с другим способом поиска числа кластеров — **методом локтя** (*elbow method*). Своё название получил неслучайно: по форме его график напоминает согнутую в локте руку. Оптимальное количество

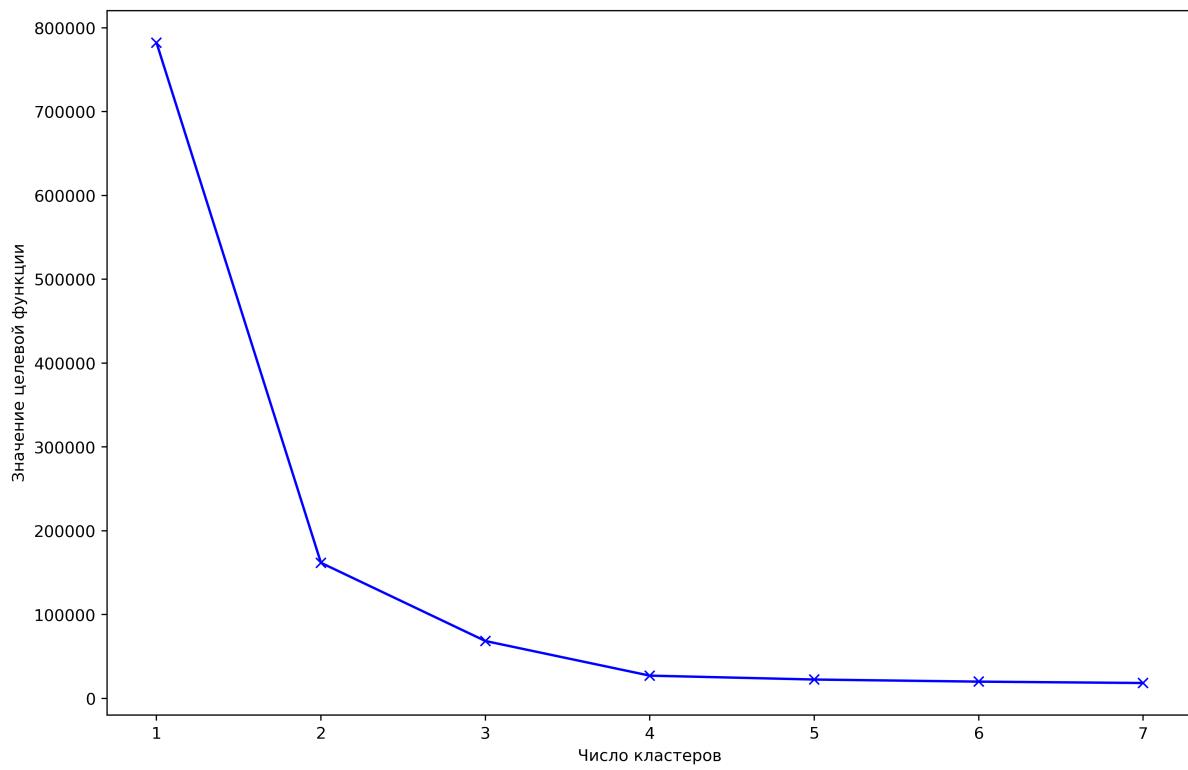
кластеров определяется условным «локтем». Чтобы построить график метода, нужно составить список из значений целевой функции для разного количества кластеров: от 1 до 10 (реже 20). Для этого обучим модель несколько раз и сохраним значения целевой функции каждой модели в список **distortion**:

```
distortion = []
K = range(1, 8)
for k in K:
    model = KMeans(n_clusters=k, random_state=12345)
    model.fit(data)
    distortion.append(model.inertia_)
```

Отобразим полученный список на графике:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

plt.figure(figsize=(12, 8))
plt.plot(K, distortion, 'bx-')
plt.xlabel('Число кластеров')
plt.ylabel('Значение целевой функции')
plt.show()
```



Это и есть график метода локтя: значение целевой функции сначала резко уменьшается, а затем выходит на плато.

Этот момент перехода как раз отражает *оптимальное количество кластеров*.

На этом графике плато начинается после четвёртого кластера. Причём неплохие результаты также показывают второй и третий кластеры: после них скачок вниз целевой функции не очень большой.

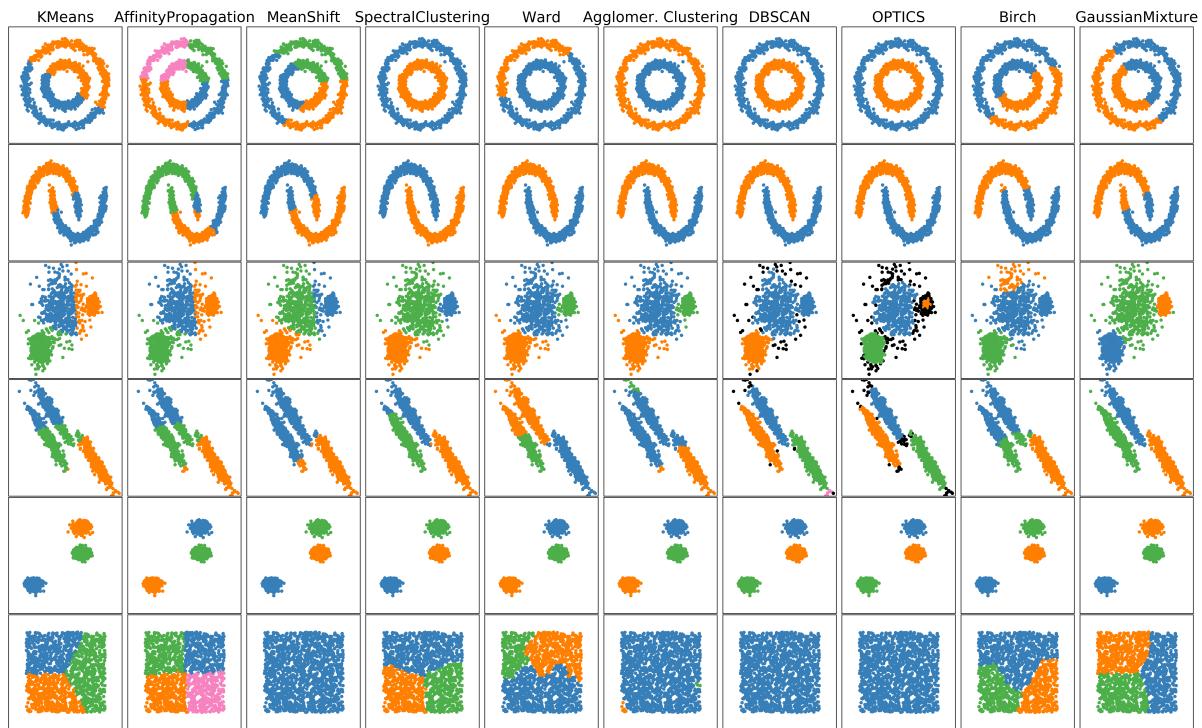
Перейдём к нашей задаче и посчитаем значения целевой функции для моделей, обученных на разном количестве кластеров.

Интерпретация результата

На примере с хорошо разделимыми данными вы узнали, как количество кластеров связано с целевой функцией. Пора разобрать, как k -средних работает на данных разного характера.

Перед вами синтетические данные и алгоритмы кластеризации из [документации sklearn](#).

k -средних находится в крайнем левом столбце. Лучше всего этот метод работает на чётко выделенных группах данных похожего размера, как в пятой строке. k -средних находит кластеры даже там, где их нет:



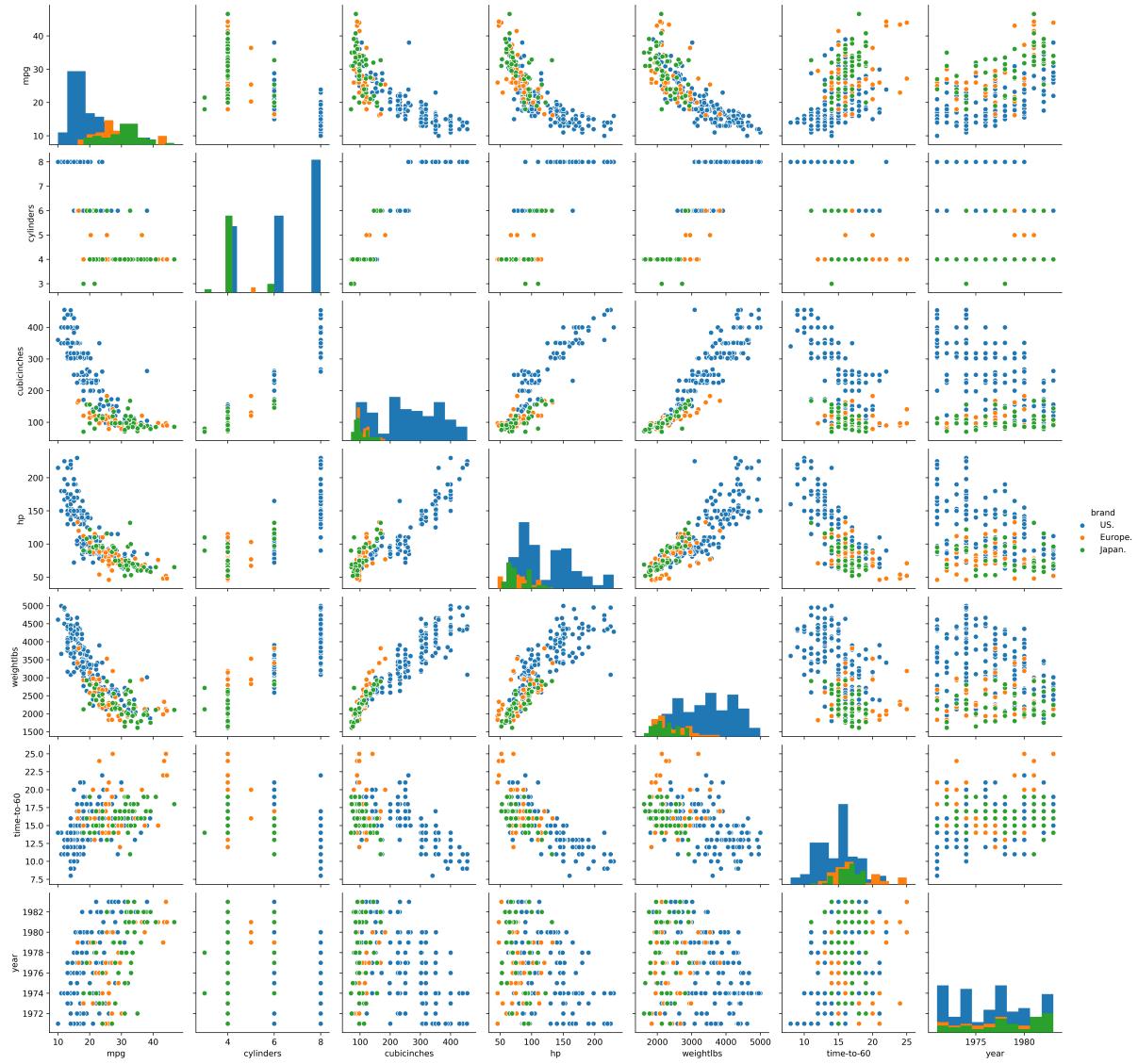
В третьей строке заметно, как часть объектов из большого кластера переходит в маленький сегмент поблизости, то есть выполняется расщепление кластера. Остальные строки показывают, что метод плохо разделяет близко расположенные вытянутые группы.

В библиотеке `sklearn` учтены все особенности работы алгоритма k -средних. Если он плохо подходит данным, например, с большой размерностью, можно изменить параметр `max_iter`. Число итераций алгоритма выясняется вызовом атрибута `n_iter`.

Поиск структуры в данных

Кластеризацию можно применить и к задачам с разметкой. Она позволит увидеть в данных структуру и понять, какие признаки важнее.

Пусть у нас есть датасет, на котором мы хотим обучить классификатор. В датасете известен целевой признак, по которому объекты делятся по группам. Распределение объектов по группам выглядит так:



Уже анализируя график, можно увидеть некоторые инсайты. Чтобы найти новые инсайты, можно обучить алгоритм k -средних без столбца с целевым признаком.