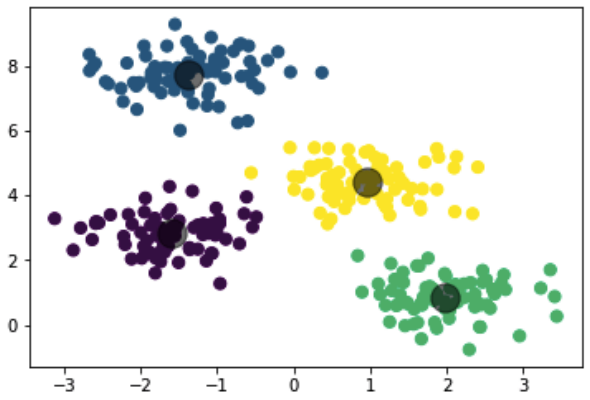
Кластеризация — это задача группировки объектов на подмножества (кластеры) таким образом, чтобы объекты из одного кластера были более похожи друг на друга, чем на объекты из других кластеров, по какому-либо критерию.

K-Means

Идея алгоритма k-means состоит в том, что он итеративно разбивает множество объектов на заранее заданное пользователем число кластеров и стремится минимизировать расстояние от объектов внутри кластера до центроида кластера.

В результате мы получаем сгруппированные по кластерам **точки** и **центроид** (его ещё иногда называют **центром тяжести**) для каждого кластера. Визуально это может выглядеть, например, так (на графике показано разделение на четыре кластера):



Алгоритм Ллойда:

Дана выборка $x_1, \ldots, x_N$.
Параметр, который мы задаём для инициализации алгоритма, — число кластеров K: $K = |Y|$.

В качестве результата мы хотим получить центры кластеров μ_k и метки $k \in Y$.

1. Задаём начальные приближения центров μ_k для всех $k \in Y$ (например, случайным образом).

2. Повторяем по очереди:

- Относим каждый объект к ближайшему центру:

$$k_i = \arg\min_{k \in Y} \|x_i - \mu_k\|, i = 1, \ldots, N$$

Примечание. Ближайший центр — это центр, до которого евклидово расстояние минимально.

- Перемещаем центр каждого кластера в центр тяжести:

$$\mu_k = \frac{\sum_{i=1}^{N} [k_i = k] x_i}{\sum_{i=1}^{N} [k_i = k]}, k \in Y$$

Пояснение. В числителе суммируются все точки, которые принадлежат кластеру, для которого мы ищем центроид ($[k_i = k]$ проверяет принадлежность кластеру k), а в знаменателе суммируется их количество. В итоге получается отношение суммы и количества — это и есть среднее арифметическое.

Работа алгоритма закончится тогда, когда выполнится число заданных заранее итераций или когда кластеры и центроиды перестанут изменяться.

Для того чтобы запустить алгоритм кластеризации k-means, нам нужна библиотека sklearn и модуль KMeans.

Что необходимо для запуска?

- ➔ Задать количество кластеров, на которые необходимо разделить данные.
- ➔ Данные, т. е. параметры объектов `()`, которые мы будем передавать в виде матрицы наблюдений X.

У алгоритма k-means есть ряд недостатков:

- ⛔ число кластеров надо знать заранее;

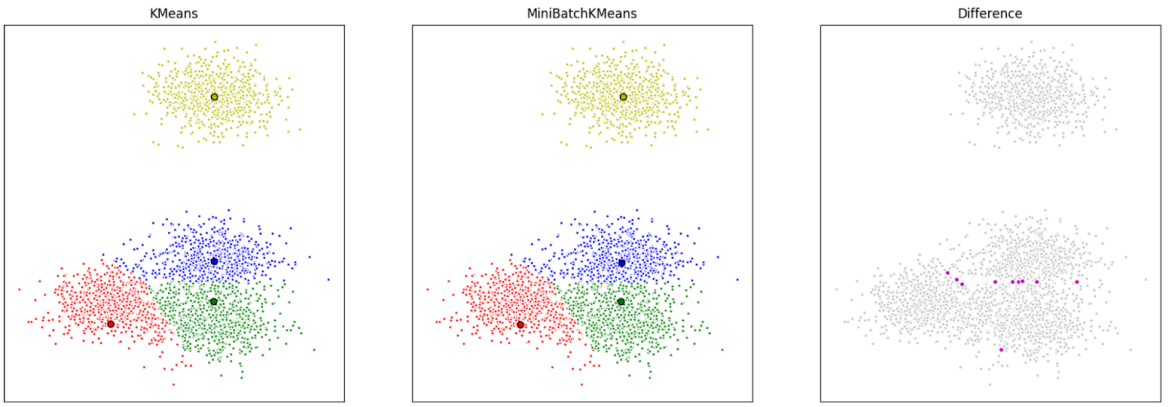
- ⛔ очень чувствителен к первичному выбору центроидов;
- ⛔ не гарантирует достижения глобального минимума суммы квадратов расстояний, периодически «застревает» в локальном минимуме.

Также у k-means есть вариации, которые применяются в некоторых специфических случаях.

Mini-Batch K-Means

Данная вариация k-means используется, когда данных очень много. Из-за их объёма вычисление центров по всей выборке занимает много времени.

Для решения этой проблемы k-means на каждом шаге работает с небольшой подвыборкой данных. В общем случае упрощённый алгоритм должен сходиться к тому же результату, что и на полной выборке. Однако исследования показывают, что качество кластеров может ухудшаться по сравнению с классическим k-means. Обычно разница в кластеризации методом Mini-Batch K-means и классическим k-means заключается в пограничных точках близко расположенных кластеров:



Для того чтобы запустить алгоритм кластеризации Mini-Batch K-means, нам нужна библиотека sklearn и модуль MiniBatchKMeans. Его применение и параметры аналогичны классическому Kmeans. Отличие лишь в том, что можно дополнительно настроить объём подвыборки:

```
# два кластера и подвыборки объёма 6
kmeans = MiniBatchKMeans(n_clusters=2,random_state=42,batch_size=6)
```

K-Means++

Данную вариацию k-means используют, если признаков очень много.

Результат и время работы алгоритма зависят от изначального выбора центроидов. Чтобы минимизировать затраты, будем действовать следующим образом:

1. Первый центроид выбираем случайным образом.
2. Для каждой точки вычисляем квадрат расстояния до ближайшего центроида из тех, что уже поставлены.
3. Далее из этих точек выбираем следующий центроид так, чтобы вероятность выбора точки была пропорциональна вычисленному для неё квадрату расстояния.
4. Когда все точки выбраны, реализуем k-means.

По умолчанию при запуске k-means в sklearn используется именно алгоритм k-means++. Выбор алгоритма задаётся через параметр `init`:

- `init='random'` — для классической версии k-means;
- `init='k-means++'` — для вариации k-means++.

Ниже можно увидеть формулу суммы квадратов всех расстояний от точек x_i до центров кластеров C_k, к которым принадлежат данные точки. Это значение ещё называют **инерцией**.

$$J(C) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2 \to \min_C$$

При реализации алгоритма k-means для получения значения инерции используется атрибут `inertia_`.

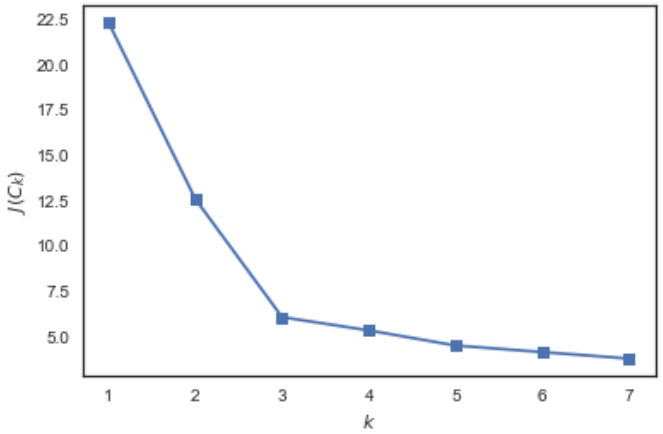
Разумеется, чем меньше эта величина, тем лучше. Однако здесь есть большая проблема: минимальное значение этой функции будет достигаться тогда, когда количество кластеров будет равняться количеству объектов (т. е. каждый кластер будет состоять из одной точки и расстояния будут нулевыми). Это уже будет ситуация переобучения, так как алгоритм чересчур сильно подстроится под данные.

Для решения этой проблемы была выведена следующая эвристика: берётся такое число кластеров, начиная с которого значение функционала

уменьшается уже не так быстро. Формально это можно записать следующим образом:

$$D(k) = \frac{|J(C_k) - J(C_{k+1})|}{|J(C_{k-1}) - J(C_k)|} \rightarrow \min_k$$

Вероятно, вы помните, что визуальное отображение данных значений и формулировка вывода по ним называется **методом локтя**:



Если в ходе решения задачи вы встречаете график, на котором невозможно найти «локоть», на помощь придёт **коэффициент силуэта**. Для того чтобы его вычислить, используется следующая формула:

$$s_i = \frac{(b_i - a_i)}{max\,(a_i, b_i)}$$

где:

- a_i — среднее расстояние от данного объекта x_i до объектов из того же кластера;
- b_i — среднее расстояние от данного объекта x_i до объектов из другого ближайшего кластера.

Для вычисления коэффициента силуэта используется silhouette_score.

Коэффициент силуэта обладает следующими свойствами:

- Значение коэффициента находится в диапазоне от -1 до +1, где высокое значение указывает, что объект хорошо согласуется с кластером, которому он принадлежит, и плохо согласуется с «чужими» кластерами.
- Если у подавляющего большинства объектов этот коэффициент высокий, то можно считать кластеризацию достаточно качественной.
- Если же у большого числа объектов низкий или отрицательный коэффициент силуэта, то, возможно, кластеров слишком много/мало или данные просто плохо поддаются разделению на кластеры.

Для получения итогового значения рассчитывается среднее значение силуэта для всего датасета.

EM-алгоритм

K-means не справляется с кластеризацией. В таком случае в качестве альтернативы можно взять один из алгоритмов кластеризации **EM (Expectation-maximization) — модель гауссовой смеси (Gaussian Mixture Model, GMM)**, в которой данные описываются нормальным законом распределения (его ещё называют распределением Гаусса).

Можно заметить аналогию EM-алгоритма с k-means, только в k-means мы однозначно относили каждую точку к тому или иному кластеру, а здесь для каждой точки вычисляем вероятности принадлежности к определённым кластерам. K-means можно считать упрощённой версией EM-алгоритма.

В более общем виде **последовательность действий в EM-алгоритме** можно сформулировать следующим образом:

1. Выбрать количество кластеров, которое кажется нам оптимальным для наших данных.
2. Случайным образом выбрать параметры распределений в пространстве данных.
3. Для каждой точки набора данных рассчитать вероятность принадлежности к каждому кластеру.
4. Обновить параметры распределений таким образом, чтобы максимизировать вероятность принадлежности точек, отнесённых к кластеру.
5. Повторять шаги 3-4 фиксированное число раз либо до тех пор, пока центроиды не сойдутся.

Сформулируем **последовательность реализации EM-алгоритма математически:**

1 **Инициализируем алгоритм:**
- ➔ k кластеров.

2
- ➔ Априорные вероятности кластеров: $w_1, \ldots, w_k$.
- ➔ Плотности распределения кластеров: $p_1(x), \ldots\ldots, p_k(x)$.

Необходимо определить:

Плотность распределения признаков объекта: $x : p(x) = \sum_{j=1}^{k} w_j p_j(x)$.

Результат:

Получим вероятность принадлежности объекта к кластеру.

$$p(x) = \sum_{j=1}^{k} w_j p_j(x)$$

,

где:

- ➔ w_j — априорная вероятность кластера $\sum_{j=1}^{k} w_j = 1$;
- ➔ $p_j(x) = \phi(x; \theta_j)$ — функция правдоподобия для j-го кластера, которая содержит некоторые параметры, определяющие вероятность.

3 **E-шаг (expectation)** — вычисляем ожидаемый кластер для каждого объекта. На этом этапе мы определяем, к какому распределению

$$g_{ji} = p(j \mid x_i) = \frac{w_j p_j(x_i)}{p(x_i)}$$

какая точка относится: — вероятность того, что x_i принадлежит j-му распределению.

4 **M-шаг (maximization)** — оцениваем вес (априорную вероятность) и параметры распределения для каждого кластера. На этом этапе мы

оцениваем новые параметры распределений по распределённым точкам:

$$w_j = \frac{1}{N}\sum_{i-1}^{N} g_{ji}$$

$$\theta_j = \arg\max_{\theta} \sum_{i=1}^{N} g_{ji} \ln \phi(\theta; x)$$

Алгоритм EM очень похож на k-means и, по сути, является его упрощённым вариантом. Однако **у этих алгоритмов есть различия:**

- кластеры в EM эллиптические, а в k-means — сферические;
- в EM кластеризация мягкая (определяем вероятность принадлежности объекта к кластеру), а в k-means — жёсткая (определяем конкретный кластер для объекта).

Для реализации алгоритма в sklearn мы используем GaussianMixture. Для запуска алгоритма GaussianMixture необходимо задать следующие основные параметры:

- `n_components` — количество кластеров;
- `random_state` — так как в алгоритме есть случайность при инициализации, то для воспроизводимости результатов от запуска к запуску следует передать какое-то число.

Плюсы и минусы EM-алгоритма:

✅ Устойчив к шумам и пропущенным значениям в данных. ✅ Кластеры, которые находит этот алгоритм, могут принимать форму эллипса, а не ограничиваться окружностями. K-Means является частным случаем GMM.	⛔ При неудачной инициализации сходимость может оказаться медленной. ⛔ Нужно задавать количество кластеров. ⛔ Иногда останавливается в локальном минимуме и не достигает глобального.

✅ Если объект находится в середине двух перекрывающихся кластеров, то, в отличие от k-means, не нужно решать, к какому кластеру он принадлежит — объект может принадлежать к двум разным кластерам с разной вероятностью.	

DBSCAN

Этот алгоритм хорошо подходит для ситуаций, когда необходимо выделить кластеры различной, порой довольно сложной формы.

Идея алгоритма DBSCAN заключается в том, что объект принадлежит кластеру, если находится достаточно близко ко многим объектам из этого кластера.

Алгоритм DBSCAN инициализируется двумя параметрами:

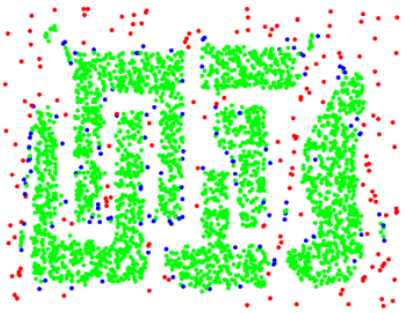
- `eps` — расстояние, определяющее окрестности. Две точки считаются достаточно близкими, чтобы находиться в одном кластере, если расстояние между ними меньше или равно eps. В библиотеке sklearn оно по умолчанию равно 0.5.
- `min_samples` — минимальное количество точек в данных, которое может быть в кластере. В библиотеке sklearn оно по умолчанию равно 5.

Эти параметры очень сильно влияют на результат кластеризации, и следует перебирать разные варианты, т. к. их подходящие значения (особенно `eps`) может быть сложно подобрать. Не стоит опираться только на результаты алгоритма, обученного со значениями по умолчанию.

На основании этих двух параметров точки классифицируются как **центральные** (их ещё часто называют **корневыми** или **основными**), **граничные** и **шумовые**:

Точка является **центральной**, если в её окрестностях имеется не менее `min_samples` точек (включая саму точку). Окрестность определяется как область внутри окружности радиуса eps, центр которой находится в центральной точке.	Точка является **граничной**, если она достижима из центральной точки (то есть находится в окрестности центральной точки) и количество точек в её окрестностях меньше `min_samples`.	Точка называется **шумовой**, если она не является центральной и не достижима ни из одной из центральных точек (то есть не находится в окрестностях центральных точек).

Изначальные данные



Центральные, граничные и шумовые точки

- Если точка q находится от центральной точки p на расстоянии, не превышающем `eps`, то точку q называют **прямо достижимой по плотности** из p.
- Если мы можем составить от точки p до точки q путь $p_1, p_2 \ldots\ldots, p_n$, где $p_1 = p$, $p_n = q$ и где каждая точка p_{i+1} прямо достижима из p_i и является центральной, то точка q называется достижимой по плотности из p.

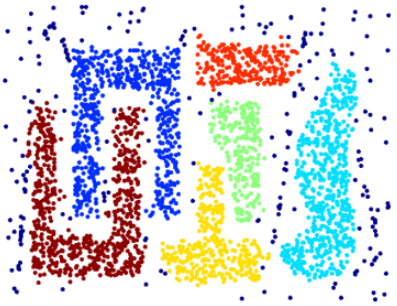
Все точки, не достижимые из центральных точек, считаются шумовыми (выбросами).

Формирование кластеров происходит следующим образом:

- Центральная точка формирует кластер со всеми точками, достижимыми из неё.

➔ Шумовые точки не включаются в кластеры.

Изначальные данные



Кластеры

Плюсы и минусы DBSCAN:

✅ Не нужно задавать количество кластеров. ✅ Устойчив к шуму и выбросам, может находить выбросы в данных. ✅ Способен найти кластеры любой геометрической формы.	⛔ Не может качественно кластеризовать наборы данных с большой разницей в плотности. ⛔ Равноудалённые от двух кластеров объекты могут быть отнесены к тому кластеру, с которого алгоритм начал работать, что не всегда оказывается верным. ⛔ Бывает сложно подобрать значение `eps`.

Агломеративная иерархическая кластеризация

Иерархическая кластеризация — очень популярный метод кластеризации в силу простоты реализации и интерпретации. По ходу её работы создаются кластеры, организованные в иерархическую структуру. Эту структуру можно визуализировать с помощью древовидной диаграммы, называемой **дендрограммой**.

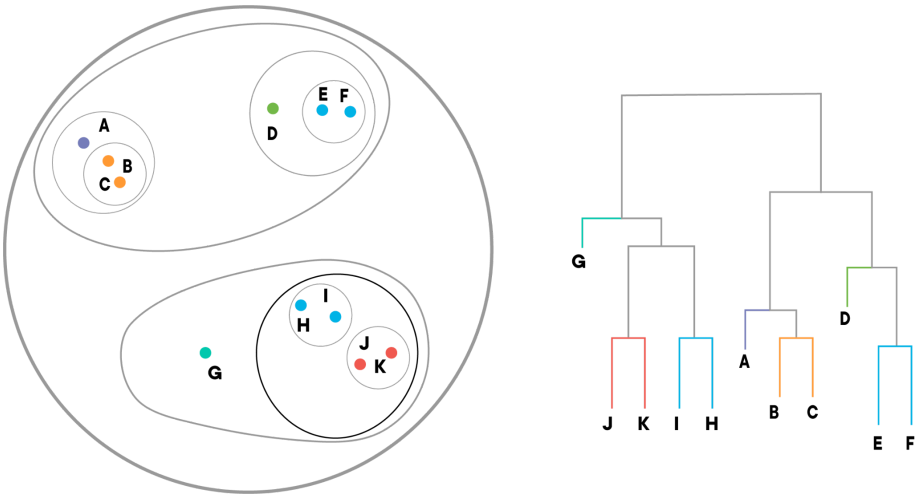
При **агломеративном** подходе кластеризации дендрограмма записывает последовательность слияний, при **дивизионном** — последовательность расщеплений.

Агломеративный алгоритм

- изначально каждый объект является отдельным кластером;
- на каждом шаге соединяем два самых близких кластера;
- останавливаемся, когда получаем требуемое количество кластеров или единственный кластер.

Дивизионный (или дивизивный) алгоритм

- начинаем с ситуации, когда все объекты входят в один кластер;
- на каждом шаге разделяем один из кластеров пополам;
- останавливаемся, когда достигается необходимое количество кластеров или количество кластеров, равное числу объектов.



Например, слева мы видим, что изначально в самые маленькие внутренние кластеры объединились E и F, I и H, J и K, B и C. Справа мы также видим объединение этих кластеров на самом первом (нижнем) уровне дендрограммы. Далее слева видно, что к кластеру с элементами E и F присоединился элемент D. На дендрограмме справа мы также видим это присоединение в крайней правой ветви. Так можно провести полную аналогию между этими двумя схемами.

Расстояние между точками можно вычислять различными способами. В библиотеке sklearn для вычисления расстояний предлагается несколько вариаций — вот наиболее популярные из них:

1 **Евклидово расстояние:**

$$d(a,b) = \sqrt{\sum_i^N (a_i - b_i)^2}$$

2 **Расстояние городских кварталов (манхэттенское расстояние):**

$$d(a,b) = \sum_i^N |a_i - b_i|$$

3 **Косинусная мера близости**

Это метрика, которая показывает не расстояние, а близость векторов (насколько маленький угол между ними). Чем больше её значение, тем ближе векторы друг к другу (максимально близки они в том случае, если совпадают). В числителе находится скалярное произведение, а в знаменателе — произведение длин векторов:

$$d(a,b) = \frac{\langle a,b \rangle}{\|a\| \cdot \|b\|}$$

? Как вычислить расстояния между двумя кластерами, если хотя бы в одном из них более одного объекта? То есть нам нужна метрика, как-то обобщающая попарные расстояния между точками кластеров. Таких метрик несколько.

Расстояния между условными кластерами A и B можно вычислить с помощью одной из следующих формул:

1 **Single linkage** — метод одиночной связи (минимум попарных расстояний между точками из двух кластеров):

$$\min\{d(a,b) : a \in A, b \in B\}$$

2 **Complete linkage** — метод полной связи (максимум попарных расстояний между точками из двух кластеров):

$$\max\{d(a,b) : a \in A, b \in B\}$$

3 **Average linkage** — метод средней связи (среднее арифметическое попарных расстояний между точками из двух кластеров):

$$\frac{1}{|A| * |B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$$

4 **Centroid linkage** — центроидный метод (расстояние между центроидами двух кластеров):

$$d = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

Первые три подхода более экономны с точки зрения вычислительной сложности, так как для них не надо заново рассчитывать расстояния после склеивания.

Стоит помнить, что использование методов средней и полной связи приносит хорошие результаты, только если данные имеют высокую плотность и не пересекаются или однородно распределены. В остальных случаях данные методы отрабатывают плохо и стоит использовать метод одиночной связи.

Резюмируем последовательность действий:

1. Сначала каждый элемент определяем как отдельный кластер.
2. Далее итеративно, пока не останется один кластер,
    a. находим два ближайших кластера,
    b. объединяем их.

Для реализации алгоритма в sklearn вам понадобится AgglomerativeClustering. Обычно ему передают следующие параметры:

- `n_clusters` — количество кластеров; по умолчанию — 2.
- `linkage` — метод определения расстояния между кластерами, которое мы рассматривали выше.
- `affinity` — метод определения расстояний между точками (например, евклидово или манхэттенское).

Плюсы и минусы агломеративной иерархической кластеризации:

✅ Можно построить	⛔ Необходимо выбирать

дендрограмму и понять, как устроены данные. ✅ Работает даже на небольшом датасете.	оптимальное количество кластеров либо оптимальное расстояние в момент инициализации алгоритма. ⛔ Если данных много, дендрограмма становится большой и сложной для понимания. ⛔ Может неравномерно разделять данные на кластеры.