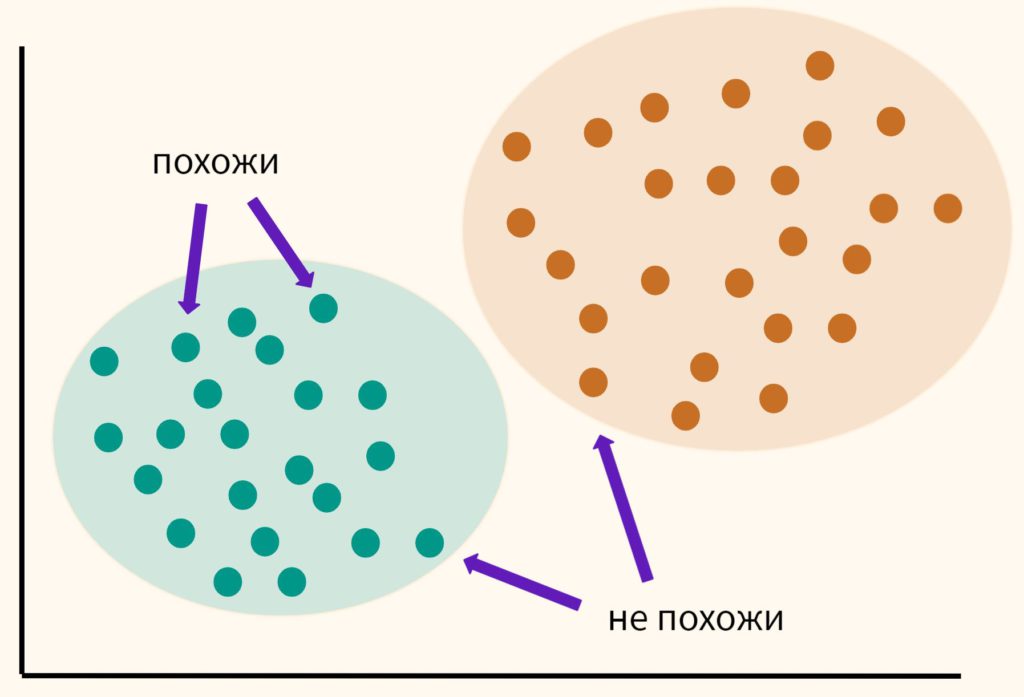
<https://www.dmitrymakarov.ru/intro/clustering-16/?ysclid=ls7h37e72u250903977#4-prakticheskiy-primer-tsvety-irisa>

# Кластерный анализ

[Все курсы](https://www.dmitrymakarov.ru/) > [Вводный курс](https://www.dmitrymakarov.ru/intro/) > Занятие 16

Основная идея кластерного анализа (clustering, cluster analysis) заключается в том, чтобы разбить объекты на группы или кластеры таким образом, чтобы внутри группы эти наблюдения были более похожи друг на друга, чем на объекты другого кластера.



При этом мы заранее не знаем на какие кластеры необходимо разбить наши данные. Это связано с тем, что мы обучаем модель на неразмеченных данных (unlabeled data), то есть без целевой переменной, компонента y. Именно поэтому в данном случае говорят по **машинное обучение без учителя** (Unsupervised Learning).

Кластерный анализ может применяться для сегментации потребителей, обнаружения аномальных наблюдений (например, при выявлении мошенничества) и в целом для структурирования данных, о содержании которых мало что известно заранее.

**Содержание занятия**

* [Как же разбить данные на кластеры?](https://www.dmitrymakarov.ru/intro/clustering-16/#0-kak-zhe-razbit-dannye-na-klastery)
  + [Метод k-средних](https://www.dmitrymakarov.ru/intro/clustering-16/#1-metod-k-srednikh)
  + [Сколько кластеров выбрать?](https://www.dmitrymakarov.ru/intro/clustering-16/#2-skolko-klasterov-vybrat)
  + [О важности нормализации данных](https://www.dmitrymakarov.ru/intro/clustering-16/#3-o-vazhnosti-normalizatsii-dannykh)
* [Практический пример — цветы ириса](https://www.dmitrymakarov.ru/intro/clustering-16/#4-prakticheskiy-primer-tsvety-irisa)
  + [Этап 1. Загрузка данных](https://www.dmitrymakarov.ru/intro/clustering-16/#5-etap-1-zagruzka-dannykh)
  + [Этап 2. Предварительная обработка данных](https://www.dmitrymakarov.ru/intro/clustering-16/#6-etap-2-predvaritelnaya-obrabotka-dannykh)
  + [Этап 3 и 4. EDA и отбор признаков](https://www.dmitrymakarov.ru/intro/clustering-16/#7-etap-3-i-4-eda-i-otbor-priznakov)
  + [Этап 5.1. Обучение модели](https://www.dmitrymakarov.ru/intro/clustering-16/#8-etap-51-obuchenie-modeli)
  + [Этап 5.2. Оценка качества модели](https://www.dmitrymakarov.ru/intro/clustering-16/#9-etap-52-otsenka-kachestva-modeli)
* [Подведем итог](https://www.dmitrymakarov.ru/intro/clustering-16/#10-podvedem-itog)
  + [Вопросы для закрепления](https://www.dmitrymakarov.ru/intro/clustering-16/#11-voprosy-dlya-zakrepleniya)
  + [Ответы на вопросы](https://www.dmitrymakarov.ru/intro/clustering-16/#12-otvety-na-voprosy)

### Как же разбить данные на кластеры?

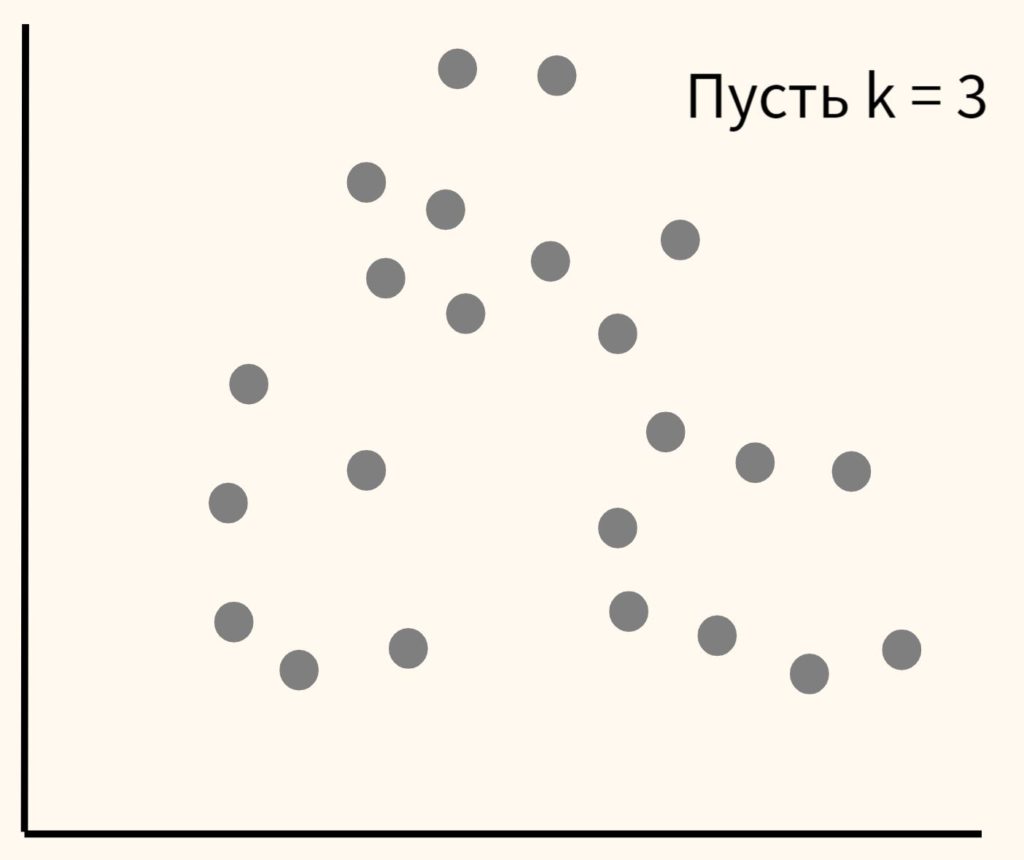
Изучая [векторы и матрицы](https://www.dmitrymakarov.ru/intro/vectors-12/), мы узнали, что векторы данных можно сравнивать между собой (оценивать их схожесть), измеряя расстояние между ними. Кластерный анализ использует именно этот подход. Мы измеряем расстояние между точками и на основе этого измерения принимаем решение к какому кластеру отнести то или иное наблюдение.

В рамках этого занятия мы поговорим про алгоритм, который называется **методом k-средних** (k-means clustering method).

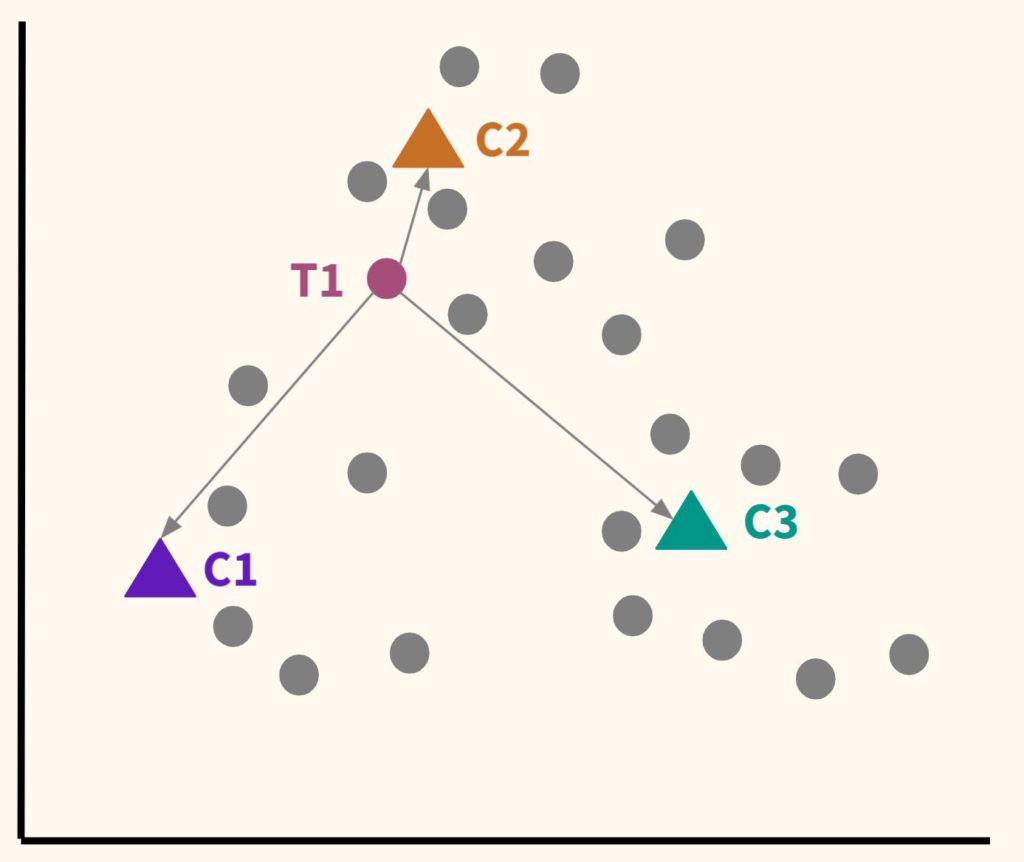
#### Метод k-средних

Давайте пошагово разберемся в том, как работает этот алгоритм.

**Шаг 1**. Вначале возьмем данные и самостоятельно выберем желаемое количество кластеров и обозначим их буквой k (отсюда название метода). Пусть в данном случае их будет три.

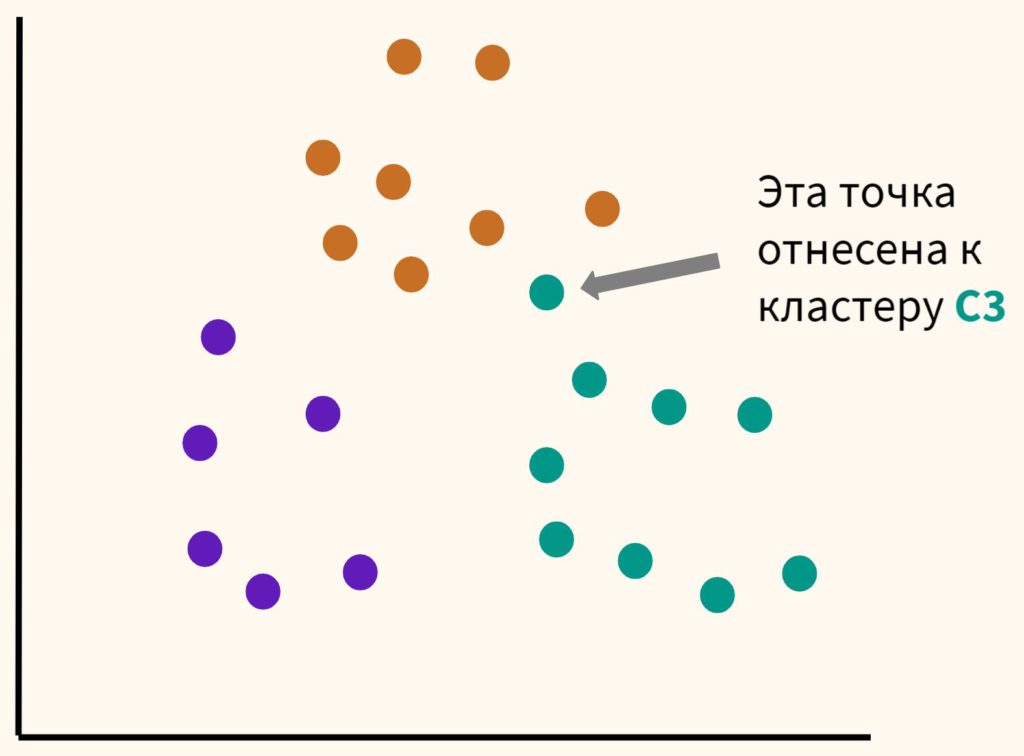


**Шаг 2**. Расположим несколько точек. Их количество будет равно количеству кластеров. Эти точки называются **центроидами**. Посчитаем расстояние от наших данных до каждого из центроидов. Логично отнести наблюдение к тому центроиду, который находится ближе.

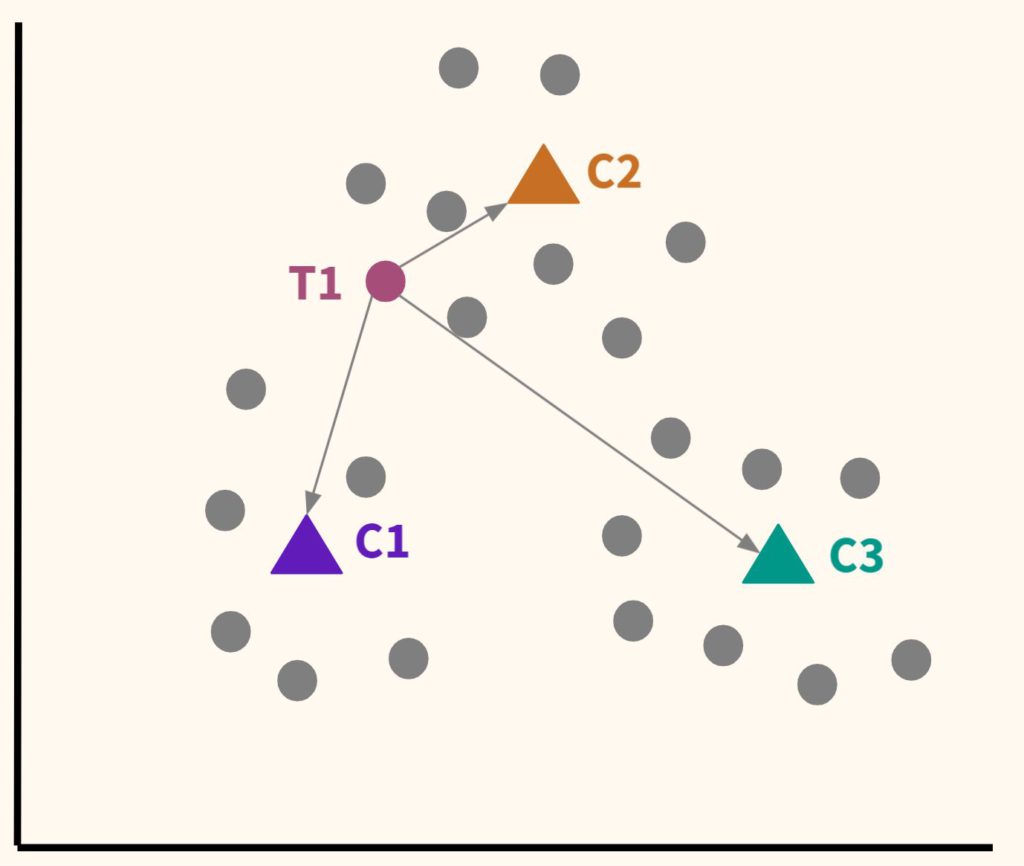


В частности, T1 будет отнесена к C2.

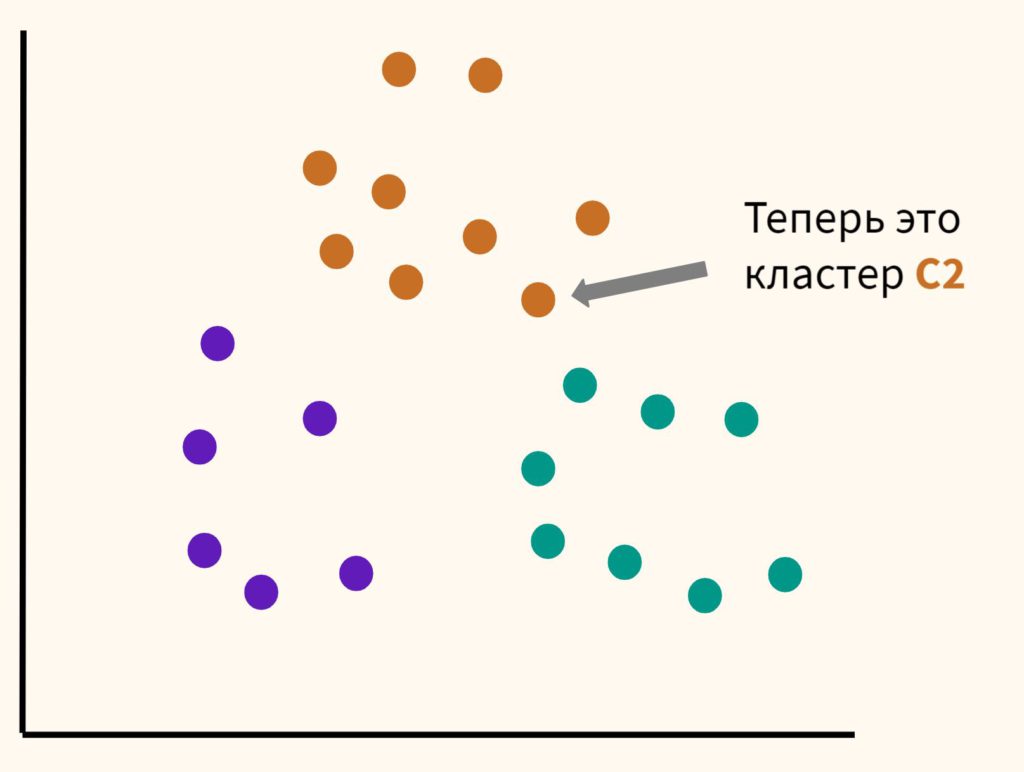
**Шаг 3**. Таким образом, каждая точка будет отнесена к определенному центроиду (кластеру).



**Шаг 4**. Сместим наши центроиды в центр получившихся кластеров.

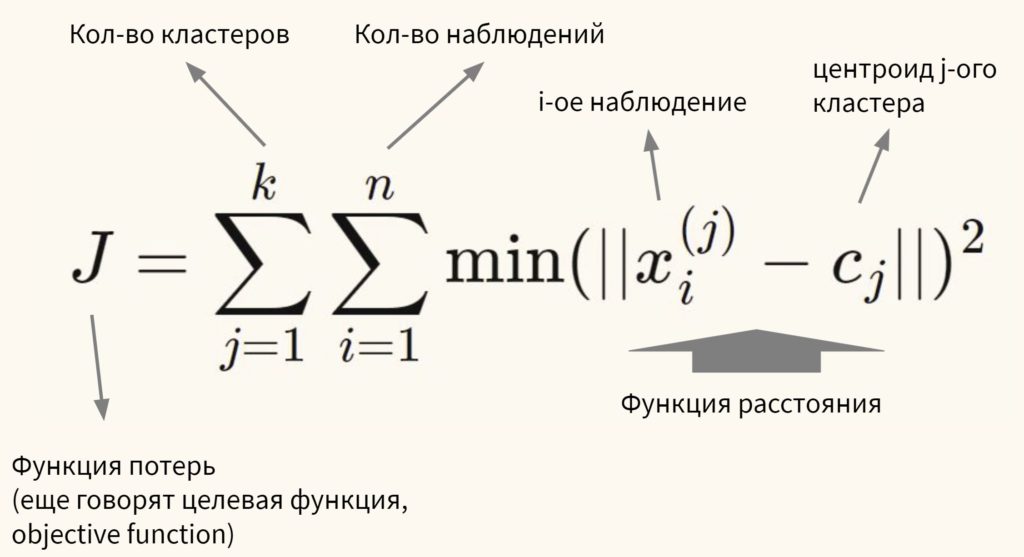


**Шаг 5**. Вновь отнесем точки к каждому из центроидов. Некоторые наблюдения «переметнутся» к другому центроиду.



Мы будем повторять шаги 4 и 5 до тех пор, пока алгоритм не стабилизируется, то есть до тех пор, пока наблюдения не перестанут переходить от одного центроида (кластера) к другому.

Говоря более формально, цель алгоритма — минимизировать **сумму квадратов внутрикластерных расстояний** до центра кластера (within-cluster sum of squares, WCSS, наша функция потерь):



Остается нерешенным важный вопрос.

#### Сколько кластеров выбрать?

Есть два способа выбора количества кластеров:

1. **Экспертный метод**. Выбор количества кластеров будет зависеть от знания о предметной области (domain knowledge)
2. **Метод локтя** (elbow method). Мы также можем (1) обучить модель используя несколько вариантов количества кластеров, (2) измерить сумму квадратов внутрикластерных расстояний и (3) выбрать тот вариант, при котором данное расстояние перестанет существенно уменьшаться.

На графике метод локтя выглядит следующим образом.

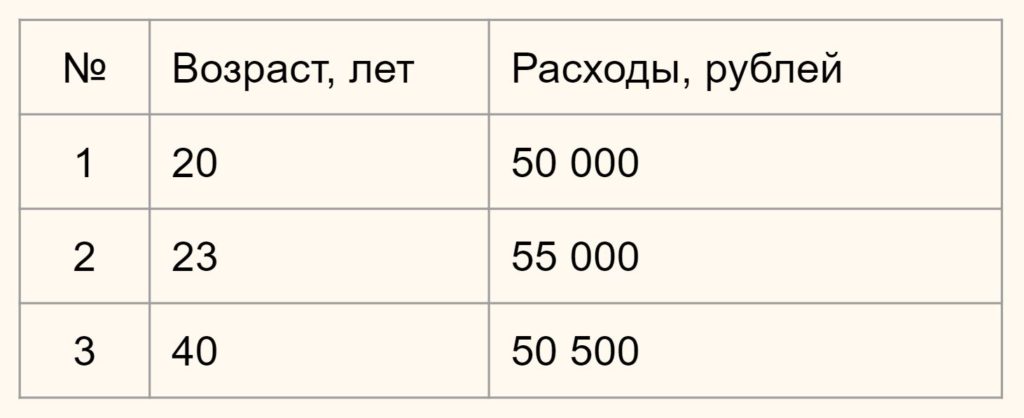


Как мы видим, после того как количество кластеров достигает трех, сумма квадратов внутрикластерных расстояний перестает существенно уменьшаться. Значит в данном случае три кластера и будет оптимальным значением.

#### О важности нормализации данных

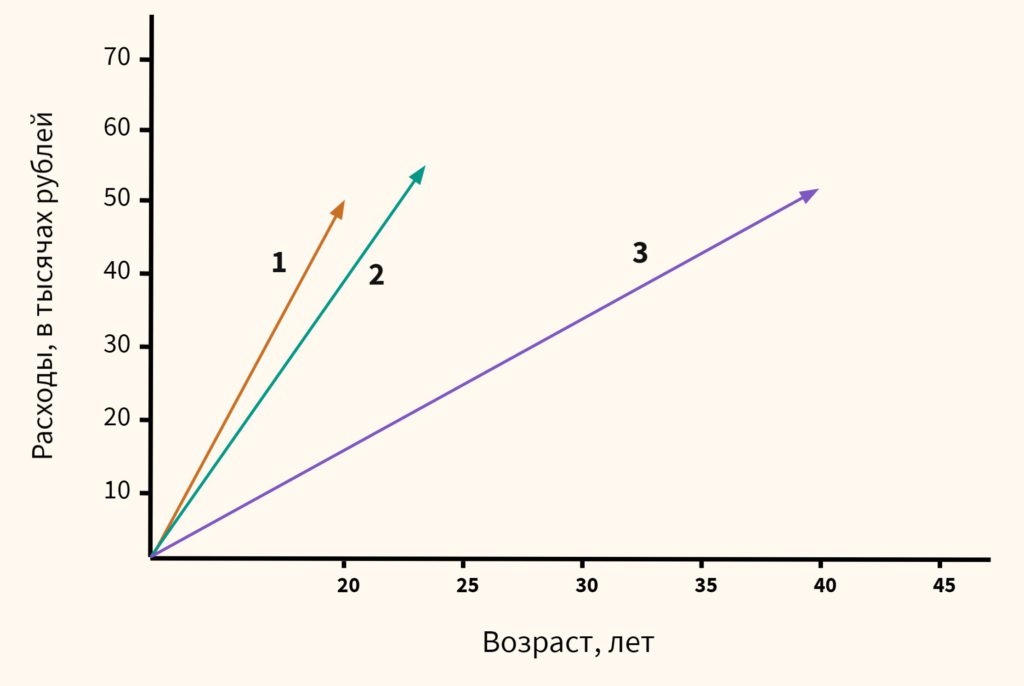
Алгоритм очень чувствителен к масштабу признаков. В связи с этим нормализация данных (feature scaling) приобретает особое значение. Так как при формировании кластеров мы измеряем расстояние (в частности, Евклидово расстояние), то признаки с большим масштабом будут иметь больший вес. Приведу пример.

Предположим у вас есть данные о возрасте и ежемесячных тратах людей по кредитным картам.



Нам нужно определить насколько человек 1 отличается (насколько велико расстояние) от человека 2 и 3. В зависимости от этого мы будем формировать наши кластеры.

Вначале давайте обратимся к **здравому смыслу**. Мы видим, что респонденты 1 и 2 схожи, потому что им обоим около 20-ти и расходы у них примерно одинаковы. Респондент 3, имея схожие расходы, сильно отличается из-за своего возраста. Он в два раза старше. Это особенно легко увидеть на графике ниже:



Теперь посмотрим, что скажет **математика**, если мы оставим масштаб признаков без изменений.

Напомню формулу Евклидова расстояния.

В данном случае x1 и x2 — это возраст двух сравниваемых нами людей, а y1 и y2 — их расходы. Подставим значения в формулу.

Как мы видим, расстояние от человека 1 до человека 2 целых 5000 единиц (лет и рублей), в то время как до человека 3 только 500. Результат обратный тому, на который мы рассчитывали исходя из здравого смысла.

Это связано с тем, что масштаб второго признака (расходов) намного больше масштаба первого (возраст). И даже небольшое изменение в расходах вызывает существенное изменение расстояния, в то время как значительное изменение возраста не оказывает на него практически никакого влияния.

### Практический пример — цветы ириса

Для иллюстрации работы алгоритма кластеризации мы возьмем еще один классический датасет из библиотеки sklearn, а именно данные о цветах ириса.

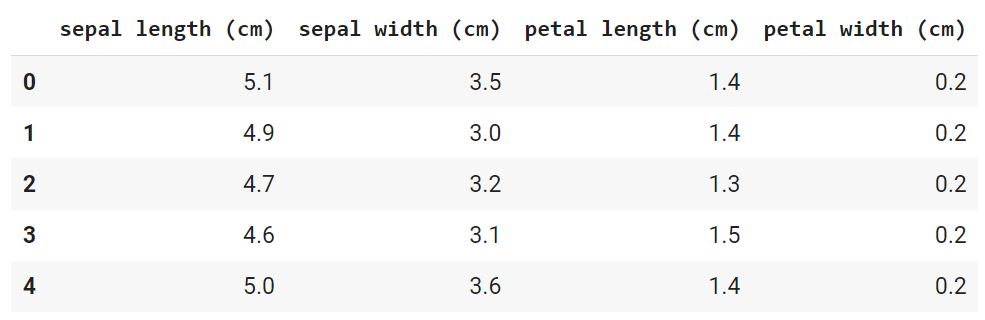


[По традиции вначале откроем ноутбук к этому занятию](https://colab.research.google.com/drive/1OxV-Hc7kQ_FGIt_0dXBXwQdYsJNyKU9X?usp=sharing)⧉

#### Этап 1. Загрузка данных

Давайте сразу загрузим данные и преобразуем их в формат датафрейма из библиотеки Pandas.

|  |
| --- |
|  |
| # импортируем набор данных из модуля datasets библиотеки sklearn  from sklearn.datasets import load\_iris  iris = load\_iris()    # создаем датафрейм, берем данные из iris.data и название столбцов из iris.feature\_names  iris\_df = pd.DataFrame(iris.data, columns = iris.feature\_names)    # смотрим первые пять значений  iris\_df.head() | |



|  |
| --- |
|  |
| # посмотрим на размерность  iris\_df.shape | |

|  |
| --- |
|  |
| (150, 4) | |

В данном случае речь идет о наборе данных, который состоит из 150 образцов цветов ириса, разделенных на три вида (Iris setosa, Iris virginica и Iris versicolor) по 50 растений в каждом. Каждый образец описан четырьмя атрибутами (длиной и шириной чашелистика и длиной и шириной лепестка).

Обратите внимание, мы сознательно не стали использовать целевую переменную, потому что решаем задачу кластеризации и предполагается, что мы не знаем заранее на какие группы или кластеры удастся разбить наши данные.

С другой стороны, тот факт, что нам заранее известно, что видов здесь три, поможет нам оценить качество кластерного анализа (об этом ниже).

#### Этап 2. Предварительная обработка данных

Для начала посмотрим, присутствуют ли **пропущенные значения**.

|  |
| --- |
|  |
| iris\_df.isnull().sum() | |

|  |
| --- |
|  |
| sepal length (cm)    0  sepal width (cm)     0  petal length (cm)    0  petal width (cm)     0  dtype: int64 | |

Мы видим, что пропущенных значений нет. Датасет был предварительно обработан.

**Категориальных переменных** у нас также нет. Это становится понятно из результата функции **head()**.

Остается разобраться с масштабом признаков. Как уже было сказано, для метода k-средних **нормализация данных** имеет особое значение. Даже небольшое различие в масштабе признаков может повлиять на конечный результат.

|  |
| --- |
|  |
| # импортируем необходимый класс из модуля preprocessing библиотеки sklearn  from sklearn.preprocessing import StandardScaler  # создадим объект этого класса  scaler = StandardScaler()  # приведем данные к единому масштабу  iris\_scaled = scaler.fit\_transform(iris\_df)    # заново создадим датафрейм с нормализованными значениями  iris\_df\_scaled = pd.DataFrame(iris\_scaled, columns = iris.feature\_names)    # посмотрим на первые пять строк  iris\_df\_scaled.round(2).head() | |



#### Этап 3 и 4. EDA и отбор признаков

Для целей кластерного анализа мы возьмем все имеющиеся у нас данные.

|  |
| --- |
|  |
| # оставим все признаки и для наглядности поместим наши данные в переменную Х  # целевой переменной у нас разумеется нет  X = iris\_df\_scaled | |

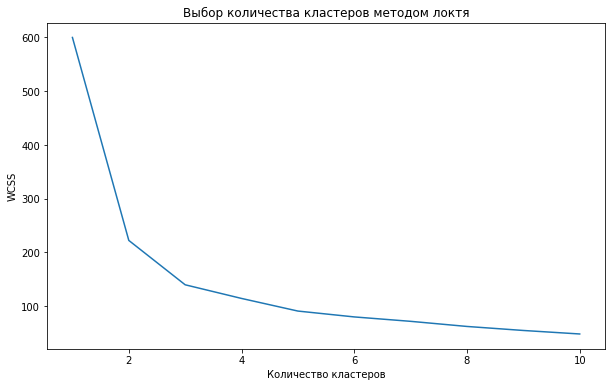
#### Этап 5.1. Обучение модели

Самый главный вопрос, который нам предстоит решить на этапе обучения модели заключается в **выборе количества кластеров**.

Количество кластеров в методе k-средних являтся так называемым **гиперпараметром**, то есть параметром, который нужно задать до обучения модели.

Мы усложним решаемую нами задачу и сделаем вид, что не обладаем экспертными знаниями о количестве видов ириса (на самом деле напомню, мы знаем, что их три). Значит нужно использовать метод локтя.

|  |
| --- |
|  |
| # импортируем нужный класс  from sklearn.cluster import KMeans    # создадим пустой список для записи показателя WCSS (нашей ошибки)  wcss = []    # воспользуемся новой для нас функцией range(), она возвращает последовательность чисел  # в данном случае числа от 1 до 10  for i in range(1, 11):        # настроим параметры модели (подробнее об этом расскажу чуть дальше)      kmeans = KMeans(n\_clusters = i, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 42)        # обучим модель на наших данных с разным количеством кластеров      kmeans.fit(X)        # для каждого кластера рассчитаем ошибку (атрибут inertia\_) и поместим в список      wcss.append(kmeans.inertia\_)    # зададим размер график  plt.figure(figsize = (10,6))  # передадим функции plot() последовательность кластеров и ошибку WCSS  plt.plot(range(1, 11), wcss)  # и добавим подписи  plt.title('Выбор количества кластеров методом локтя')  plt.xlabel('Количество кластеров')  plt.ylabel('WCSS') | |



Как видно на графике, когда мы перешли от трех до четырех кластеров, ошибка перестала существенно уменьшаться (это согласуется с тем, что видом действительно три).

Давайте создадим объект класса нашей модели, используя три кластера в качестве гипепараметра модели.

|  |
| --- |
|  |
| # заново обучим модель с тремя кластерами  kmeans = KMeans(n\_clusters = 3, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 42) | |

Подробнее рассмотрим параметры модели:

* **n\_clusters**: это количество кластеров, на которые мы хотим разбить наши наблюдения
* **init**: определяет, как мы выберем первоначальное расположение (инициализацию) центроидов; есть два варианта, (1) выбрать центроиды случайно init = 'random' или (2) выбрать их так, чтобы центроиды с самого начала располагались максимально далеко друг от друга init = 'k-means++'; второй вариант оптимальнее
* **n\_init**: сколько раз алгоритм будет инициализирован, т.е. сколько раз будут выбраны центроиды до начала оптимизации; на выходе будет выбран тот вариант, где ошибка была минимальна
* **max\_iter**: максимальное количество итераций алгоритма после первоначального выбора центроидов
* **random\_state**: воспроизводимость результата, с этим мы уже знакомы

Обучим модель и сделаем прогноз:

|  |
| --- |
|  |
| # обучение и прогноз в данном случае можно сделать одним методом .fit\_predict()  y\_pred = kmeans.fit\_predict(X) | |

#### Этап 5.2. Оценка качества модели

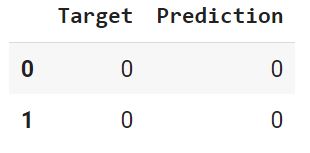
Остается проверить качество модели и здесь возникает сложность. Ведь если при обучении с учителем у нас был критерий (целевая переменная), то здесь такого критерия нет.

Впрочем, так как у нас учебный датасет, и мы заранее знаем, к какому виду относится каждый цветок, то можем сравнить результат нашей модели с целевой переменной.

На настоящих данных такое конечно невозможно.

Вначале преобразуем нашу целелевую переменную (iris.target) и наш прогноз (y\_pred) в датафрейм (предварительно слегка их изменив, подробности в [ноутбуке](https://colab.research.google.com/drive/1OxV-Hc7kQ_FGIt_0dXBXwQdYsJNyKU9X?usp=sharing)⧉).

|  |
| --- |
|  |
| # поместим переменные в словарь  d = {'Target' : iris.target, 'Prediction' : y\_pred}    # превратим словарь в датафрейм с соответствующими названиями столбцов  df = pd.DataFrame(d, columns = ['Target', 'Prediction']) | |



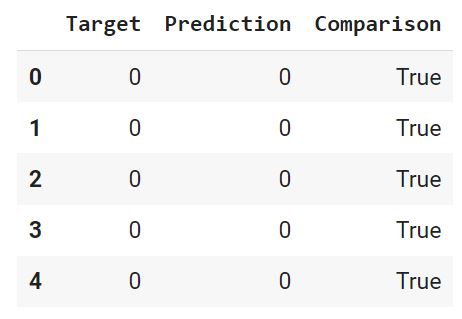
С помощью функции **where()** создадим массив Numpy, в котором сравним каждую строчку датафрейма, и если целевая переменная и прогноз совпадают, зададим значение True, в противном случае — False.

|  |
| --- |
|  |
| comparison = np.where(df['Target'] == df['Prediction'], True, False)  print(type(comparison))  print(comparison[:5]) | |

|  |
| --- |
|  |
| <class 'numpy.ndarray'>  [ True  True  True  True  True] | |

Добавим этот массив в качестве столбца в датафрейм.

|  |
| --- |
|  |
| result['Comparison'] = comparison  result.head() | |



Выведем долю совпавших (True) и не совпавших (False) значений. Для этого используем функцию **value\_counts()**, которая подсчитает, сколько раз встречается каждое значение. Параметр normalize = True вернет относительное значение или процент. Ровно это нам и нужно.

|  |
| --- |
|  |
| df['Comparison'].value\_counts(normalize = True).round(2) | |

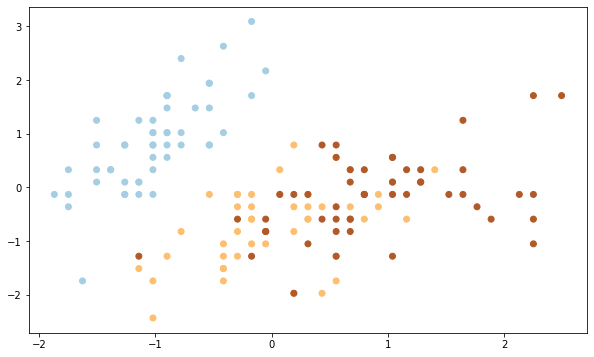
|  |
| --- |
|  |
| True     0.83  False    0.17  Name: Comparison, dtype: float64 | |

Как мы видим, модель была права в 83% случаев. Теперь давайте **визуально** оценим результат.

В исходном датафрейме четыре признака, а значит четыре измерения, столько мы представить графически не можем. Давайте возьмем первый (sepal length) и второй (sepal width) столбец исходного датафрейма.

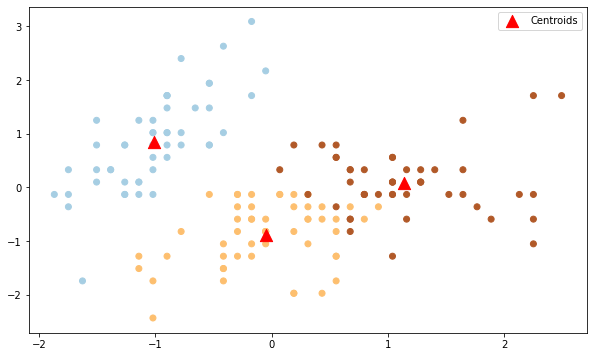
Вначале построим точечную диаграмму целевой переменной.

|  |
| --- |
|  |
| # зададим размер  plt.figure(figsize = (10,6))    # возьмем первый и второй столбцы с помощью функции iloc()  # параметр с задаст каждой точке определенный код (0, 1, 2), а  # camp определит цвет каждой из точек  plt.scatter(X.iloc[:,0], X.iloc[:,1], c = iris.target, cmap = 'Paired') | |



Теперь посмотим на результат алгоритма кластеризации.

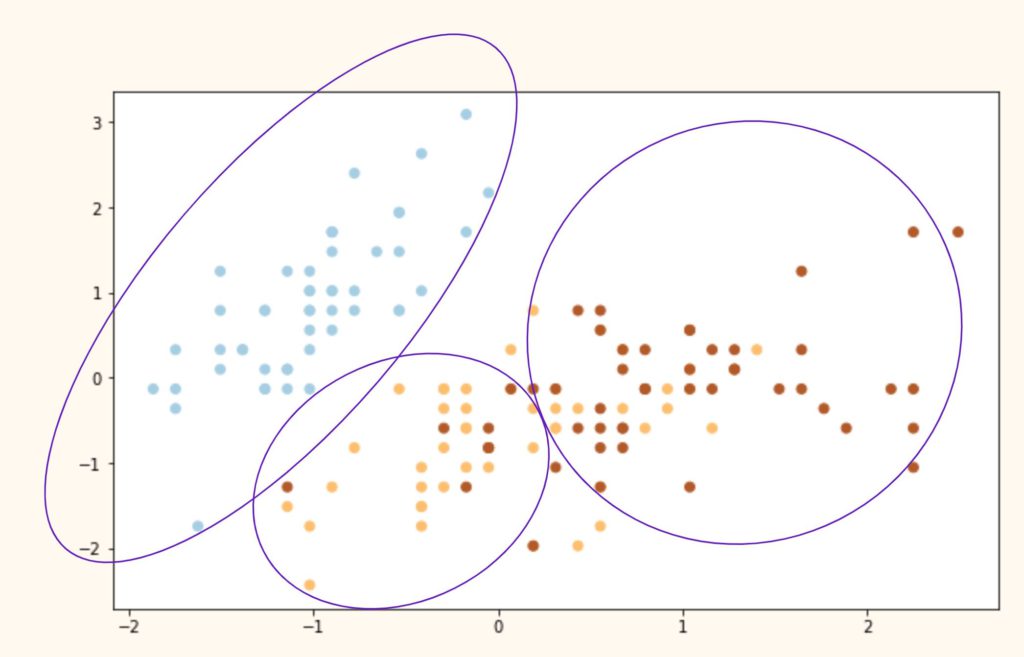
|  |
| --- |
|  |
| plt.figure(figsize = (10,6))    # снова построим точечную диаграмму, но коды точек возьмем из y\_pred  plt.scatter(X.iloc[:,0], X.iloc[:,1], c = y\_pred, cmap='Paired')    # кроме того выведем центроиды с помощью атрибута cluster\_centers\_  # зададим размер s = 150, красный цвет c = 'red', тип маркера - треугольник, и подпись  plt.scatter(kmeans.cluster\_centers\_[:,0], kmeans.cluster\_centers\_[:,1], s = 150, c = 'red', marker = '^', label = 'Centroids')    # укажем, где разместить легенду  plt.legend(loc = 'upper right') | |



**Выводы**. Как мы видим, алгоритм идеально справился с кластером 0 (светлоголубой), однако допустил ошибки при разделение кластеров 1 и 2 (желтый и коричневый цвета). Почему?

На самом деле все дело в самом алгоритме. Как мы сказали выше, алгоритм группирует данные вокруг центроидов, а это значит, что он хорошо работает с **выпуклыми** и **далеко расположенными** друг от друга кластерами. Как только данные «проникают друг в друга» и расположены слишком близко, алгоритм теряется.

Иллюстрация ниже примерно показывает, что сделал алгоритм с нашими исходными видами и почему он ошибочно группировал некоторые точки.



### Подведем итог

При решении задач кластеризации мы берем данные, обязательно их масштабируем и выбираем количество кластеров (с помощью экспертной оценки или метода локтя). К сожалению, дать точную оценку качества кластеризации бывает очень сложно из-за отсутствия разметки.

#### Вопросы для закрепления

Почему кластеризация называется машинным обучением без учителя?

Посмотреть правильный ответ

Как выбирается гиперпараметр модели (количество кластеров)?

Посмотреть правильный ответ

В чем заключаются основные ограничения модели k-средних?

Посмотреть правильный ответ

Дополнительные [упражнения](https://colab.research.google.com/drive/1OxV-Hc7kQ_FGIt_0dXBXwQdYsJNyKU9X#scrollTo=_AlUvfqkZnSl)⧉ вы найдете в конце ноутбука.

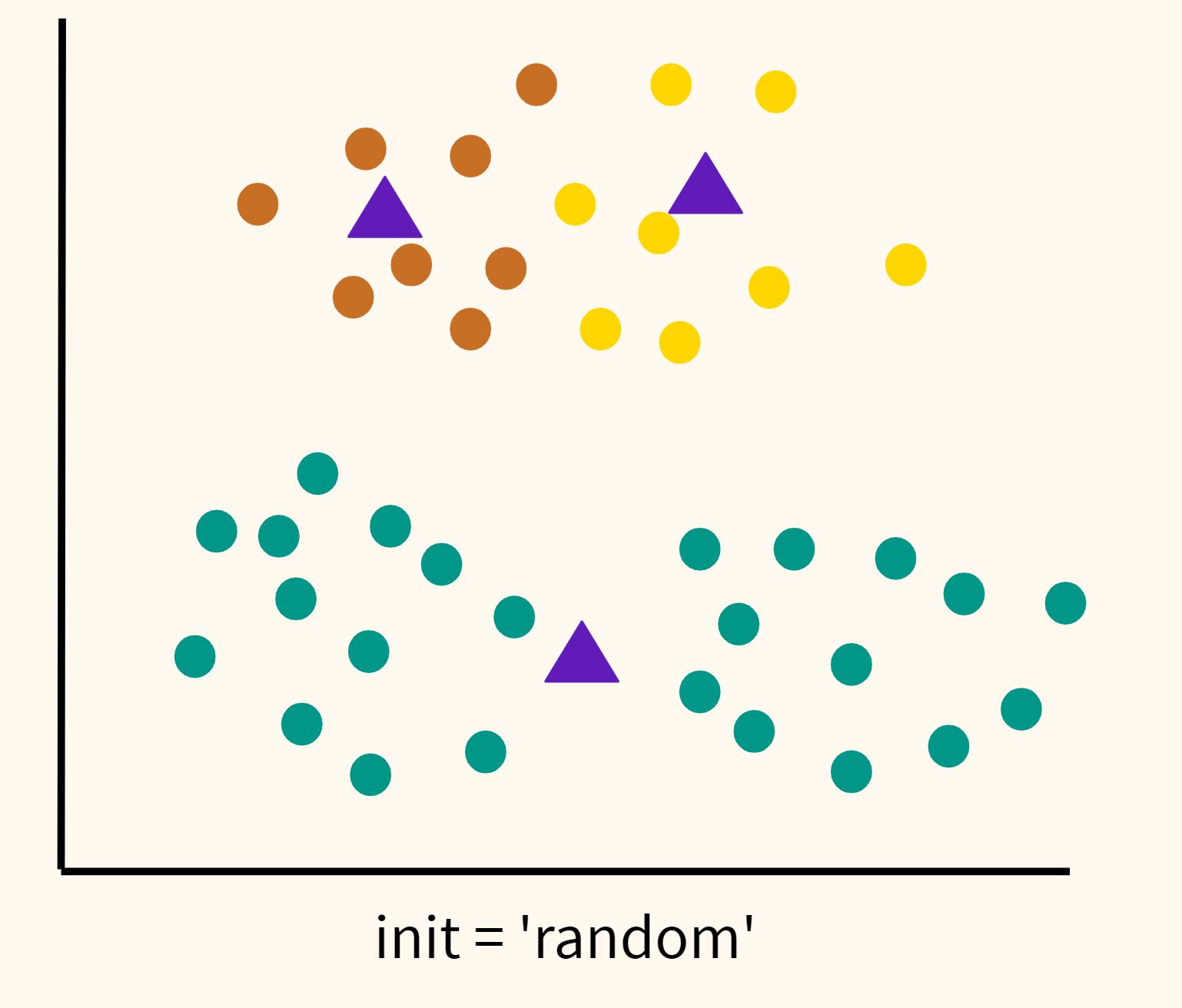
Мы закончили третий раздел классических алгоритмов машинного обучения. Пора переходить к более продвинутым задачам. Начнем с [рекомендательных систем](https://www.dmitrymakarov.ru/intro/recommender-17/).

#### Ответы на вопросы

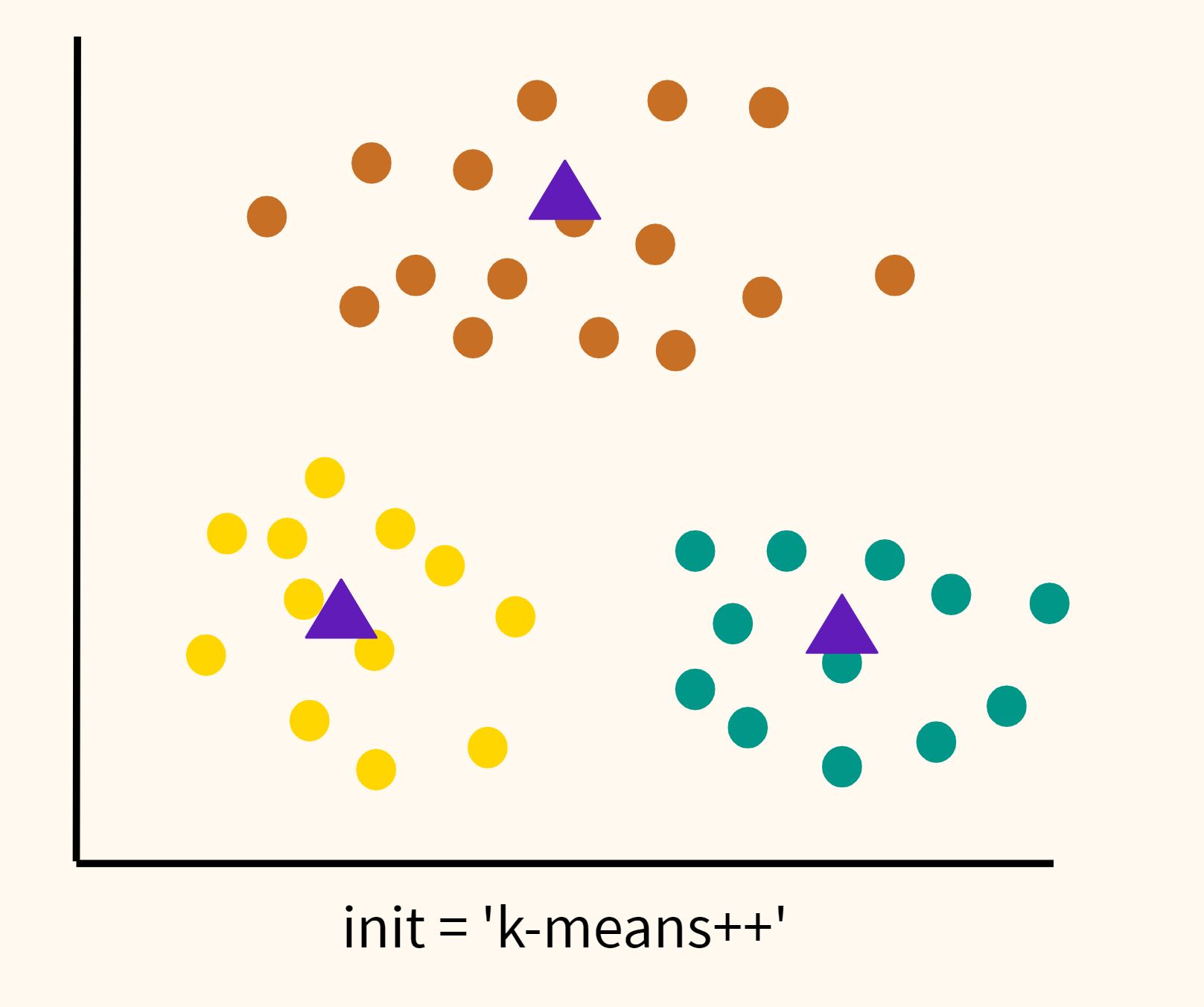
**Вопрос**. Скажите, а чем init = ‘k-means++’ лучше, чем init = ‘random’?

**Ответ**. Давайте разбираться. В первую очередь, посмотрим на **случайную инициализацию центроидов** ( init = 'random'). Как следует из самой формулировки, мы случайным образом выбираем центры кластеров, и затем алгоритм, как описано в лекции, старается минимизировать функцию потерь (WCSS в данном случае).

У такого подхода есть один недостаток. Если центры кластеров выбираются слишком близко друг к другу, то алгоритм может «разделить» то, что должно быть единым кластером, и «объединить» два разных. Пример ошибочной кластеризации на рисунке ниже.



Как мы видим, алгоритм действительно минимизировал WCSS, однако лишь в пределах того, что ему позволил изначальный выбор положения центроидов. Это так называемый локальный минимум функции потерь. Глобальный же найден не был. Схематически глобальный минимум мог бы выглядеться как на графике ниже.



В данном случае глобальный оптимум был бы достигнут за счет максимизации изначального расстояния между центроидами. На этом и основан **метод k-means++** ( init = 'k-means++').

Сам по себе алгоритм k-means++ очень прост:

1. Первый центроид выбирается случайно.
2. Далее рассчитывается Евклидово расстояние между этим центроидом и всеми остальными точками датасета. Наиболее удаленная точка станет следующим центроидом.
3. Каждая точка относится к ближайшему выбранному центроиду.
4. Точка, наиболее удаленная от «своего» центроида, становится следующим центром кластера.
5. Повторяем шаги 3 и 4 до тех пор, пока не выявим k центроидов.

© 2024 [Машинное обучение](https://www.dmitrymakarov.ru)