## Обучение без учителя

## Определение

Машинное обучение без учителя или неконтролируемое обучение (Unsupervised Learning) — метод машинного обучения (Machine Learning, ML), при котором модель обучается выявлять закономерности и скрытые взаимосвязи на наборах неразмеченных данных без контроля со стороны пользователя.

Обучение без учителя (Unsupervised Learning) было изобретено позже, аж в 90-е, и на практике используется реже. Но бывают задачи, где у нас просто нет выбора.

Размеченные данные, как я сказал, дорогая редкость. Но что делать если я хочу, например, написать классификатор автобусов — идти на улицу руками фотографировать миллионы автобусов и подписывать где какой? Так и жизнь вся пройдёт, а у меня еще игры в стиме не пройдены.

Когда нет разметки, есть надежда на биг.тех компании и миллион людей с [Яндекс.Толока](https://toloka.yandex.ru/" \t "_blank), которые готовы размечать для вас любые данные. Так обычно и поступают на практике. А вы думали где большие компании берут все свои крутые датасеты?

Либо, можно попробовать обучение без учителя. Хотя, честно говоря, из я не припомню чтобы где-то оно сработывало хорошо.

Обучение без учителя, всё же, чаще используют как метод анализа данных, а не как основной алгоритм.

Специализированный человек с хорошими математическими знаниями вбрасывает туда кучу информации и наблюдает. Кластеры есть? Зависимости появились? Нет? Ну штош, нужно продолжать, труд освобождает.

* Ассоциативные алгоритмы. Предназначены для нахождения данных или параметров, которые часто используются вместе. Например, ассоциативные алгоритмы помогают предлагать клиенту третий товар на основе двух выбранных.
* Снижение размерности. Подразумевает преобразование данных для уменьшения их числа и выделения основных переменных. Метод используется для удаления из выборки неинформативных и избыточных данных, усложняющих обработку.
* Кластеризация. Подразумевает разделение объектов (данных) из выборки на отдельные кластеры. То есть, при кластеризации алгоритмы изучают исходные данные, находят между ними взаимосвязи и создают на их основе группы.
* Обнаружение аномалий. Подразумевает нахождение в наборе данных, которые существенно отличаются от остальных данных набора.

## Уменьшение размерности данных

«Собирает конкретные признаки в абстракции более высокого уровня»

Изначально это были методы хардкорных Data Scientist'ов, которым сгружали две фуры цифр и говорили найти там что-нибудь интересное. Когда просто строить графики в экселе уже не помогало, они придумали напрячь машины искать закономерности вместо них. Так у них появились методы, которые назвали Dimension Reduction или Feature Learning.

Для нас практическая польза их методов в том, что мы можем объединить несколько признаков в один и получить абстракцию. Например, собаки с треугольными ушами, длинными носами и большими хвостами соединяются в полезную абстракцию «овчарки». Да, мы теряем информацию о конкретных овчарках, но новая абстракция всяко полезнее этих лишних деталей. Плюс, обучение на меньшем количестве размерностей идёт сильно быстрее.

Инструмент на удивление хорошо подошел для определения тематик текстов (Topic Modelling). Мы смогли абстрагироваться от конкретных слов до уровня смыслов даже без привлечения учителя со списком категорий. Алгоритм назвали [Латентно-семантический анализ](https://habr.com/post/110078/) (LSA), и его идея была в том, что частота появления слова в тексте зависит от его тематики: в научных статьях больше технических терминов, в новостях о политике — имён политиков. Да, мы могли бы просто взять все слова из статей и кластеризовать, но тогда мы бы потеряли все полезные связи между словами, например, что батарейка и аккумулятор, означают одно и то же в разных документах.

Точность такой системы — достаточно плоха, даже не пытайтесь.

Нужно как-то объединить слова и документы в один признак, чтобы не терять эти скрытые (латентные) связи. Отсюда и появилось название метода. Оказалось, что [Сингулярное разложение](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D0%B7%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) (SVD) легко справляется с этой задачей, выявляя для нас полезные тематические кластеры из слов, которые встречаются вместе.

Другое мега-популярное применение метода уменьшения размерности нашли в рекомендательных системах и коллаборативной фильтрации. Оказалось, если абстрагировать ими оценки пользователей фильмам, получается неплохая система рекомендаций кино, музыки, игр и чего угодно вообще.

Полученная абстракция будет с трудом понимаема мозгом, но когда исследователи начали пристально рассматривать новые признаки, они обнаружили, что какие-то из них явно коррелируют с возрастом пользователя (дети чаще играли в Майнкрафт и смотрели мультфильмы), другие с определёнными жанрами кино, а третьи вообще с синдромом поиска глубокого смысла.

Машина, не знавшая ничего кроме оценок пользователей, смогла добраться до таких высоких материй, даже не понимая их. Достойно. Дальше можно проводить соцопросы и писать дипломные работы о том, почему бородатые мужики любят романтические драмы и мультики.

В реальных задачах наборы данных, для которых мы используем алгоритмы машинного обучения, содержат миллионы образцов для каждой функции, что очень замедляет процесс обучения модели и найти хорошее решение будет очень трудно. В таких ситуациях мы должны использовать концепцию уменьшения размерности, что означает уменьшение размеров набора данных.

Уменьшение размерности набора данных приводит к потере информации, поэтому, хотя это и ускоряет процесс обучения модели машинного обучения, но оно может ухудшить производительность модели. Поэтому, если процесс обучения идет очень медленно, вы должны попытаться сначала обучить свою модель, не уменьшая размеры в исходном наборе данных.

Хотя в некоторых случаях он отфильтровывает шум и другие ненужные данные, и это дает повышение производительности вашей модели. Помимо повышения производительности и ускорения процесса, уменьшение размерности также очень полезно для визуализации данных.

Сегодня используют для:

* Рекомендательные Системы (★)
* Красивые визуализации
* Определение тематики и поиска похожих документов
* [Анализ фейковых изображений](https://vas3k.blog/blog/390/)
* Риск-менеджмент

## PCA

Метод главных компонентов

Зная зависимости и их силу, мы можем выразить несколько признаков через один, слить воедино, так сказать, и работать уже с более простой моделью. Конечно, избежать потерь информации, скорее всего не удастся, но минимизировать ее нам поможет как раз метод PCA.  
  
Выражаясь более строго, данный метод аппроксимирует n-размерное облако наблюдений до эллипсоида (тоже n-мерного), полуоси которого и будут являться будущими главными компонентами. И при проекции на такие оси (снижении размерности) сохраняется наибольшее количество информации.

представьте, что цветовые компоненты R, G и B мы взяли как оси координат — каждая от 0 до 255. И на этом трехмерном графике точками отметили все пиксели, которые есть на нашем изображении. Получится что-то похожее на картинку ниже.

Можно заметить, что наши пиксели не рассосались по графику равномерно, а вытянулись в элипсоид. Все реальные изображения примерно так устроены. Теперь мы можем построить новые оси — вдоль элипсоида (это самая главная) и две поперек — это и будут те самые «главные компоненты». Для каждого изображения набор цветов будет разным, элипсоид и главные компоненты будут направлены по-своему.

## Кластеризация

*«Разделяет объекты по неизвестному признаку. Машина сама решает как лучше»*

Сегодня используют для:

* Сегментация рынка (типов покупателей, лояльности)
* Объединение близких точек на карте
* Сжатие изображений
* Анализ и разметки новых данных
* Детекторы аномального поведения

Кластеризация — это классификация, но без заранее известных классов. Она сама ищет похожие объекты и объединяет их в кластеры. Количество кластеров можно задать заранее или доверить это машине. Похожесть объектов машина определяет по тем признакам, которые мы ей разметили — у кого много схожих характеристик, тех давай в один класс.

В задаче классификации мы имели дело с восстановлением отображения из множества объектов в конечный набор меток классов. При этом классы были зафиксированы заранее, то есть мы с самого начала примерно понимали, какого рода объекты должны относиться к каждому из них, и мы располагали обучающей выборкой с примерами объектов и классов, к которым они относятся. В задаче кластеризации мы тоже разбиваем объекты на конечное множество классов, но у нас нет ни обучающей выборки, ни понимания, какой будет природа этих классов. То, что модель кластеризации какие-то объекты сочла «похожими», отнеся к одному классу, будет новой информацией, «открытием», сделанным этой моделью. Обучающей выборки у нас также не будет: ведь мы не знаем заранее, что за классы получатся (а иногда и сколько их будет). Таким образом, **кластеризация** — это задача обучения без учителя. Из-за общего сходства постановок задач в литературе кластеризацию иногда называют **unsupervised classification**.

Отличный пример кластеризации — маркеры на картах в вебе. Когда вы ищете все рестораны в Москве, движку приходится группировать их в кружочки с циферкой, иначе браузер зависнет в потугах нарисовать миллион маркеров.

Более сложные примеры кластеризации можно вспомнить в приложениях iPhoto или Google Photos, которые находят лица людей на фотографиях и группируют их в альбомы. Приложение не знает как зовут ваших друзей, но может отличить их по характерным чертам лица. Типичная кластеризация.

Правда для начала им приходится найти эти самые «характерные черты», а это уже только с учителем.

Методы кластеризации часто применяют, когда фактически нужно решить задачу классификации, но обучающую выборку собрать затруднительно (дорого или долго). При этом валидационную выборку для оценки результатов кластеризации собрать значительно проще, так как для неё требуется меньше примеров. При этом стоит помнить, что точность работы supervised-методов значительно выше. Поэтому, если обучающую выборку всё-таки можно собрать, лучше решать задачу классификации, чем задачу кластеризации.

Сжатие изображений — еще одна популярная проблема. Сохраняя картинку в PNG, вы можете установить палитру, скажем, в 32 цвета. Тогда кластеризация найдёт все «примерно красные» пиксели изображения, высчитает из них «средний красный по больнице» и заменит все красные на него. Меньше цветов — меньше файл.

Проблема только, как быть с цветами типа Cyan — вот он ближе к зеленому или синему? Тут нам поможет популярный алгоритм кластеризации — [Метод К-средних (K-Means)](https://www.youtube.com/watch?v=_aWzGGNrcic). Мы случайным образом бросаем на палитру цветов наши 32 точки, обзывая их центроидами. Все остальные точки относим к ближайшему центроиду от них — получаются как бы созвездия из самых близких цветов. Затем двигаем центроид в центр своего созвездия и повторяем пока центроиды не перестанут двигаться. Кластеры обнаружены, стабильны и их ровно 32 как и надо было.

## Метод K средних

Пожалуй, один из наиболее популярных методов кластеризации — это метод K-средних (K-means). Основная идея метода — итеративное повторение двух шагов:

1. распределение объектов выборки по кластерам;
2. пересчёт центров кластеров.

В начале работы алгоритма выбираются К случайных центров в пространстве признаков. Каждый объект выборки относят к тому кластеру, к центру которого объект оказался ближе. Далее центры кластеров пересчитывают как среднее арифметическое векторов признаков всех вошедших в этот кластер объектов (то есть центр масс кластера). Как только мы обновили центры кластеров, объекты заново перераспределяются по ним, а затем можно снова уточнить положение центров. Процесс продолжается до тех пор, пока центры кластеров не перестанут меняться.

### Выбор начального приближения

Первый вопрос при выборе начального положения центров — как, выбирая центры из некоторого случайного распределения, не попасть в область пространства признаков, где нет точек выборки. Базовое решение — просто выбрать в качестве центров какие-то из объектов выборки.

Вторая потенциальная проблема — кучное размещение центров. В этом случае их начальное положение с большой вероятностью окажется далёким от итогового положения центров кластеров.

### Выбор метрик

Так как работа метода K-средних состоит из последовательного повторения до сходимости двух шагов, обоснованность применения различных метрик (расстояний между точками, а не метрик качества :) или функций близости связана с тем, «ломают» они какой-либо из этих шагов или нет.

Первый шаг с отнесением объектов к ближайшим центрам не зависит от вида метрики. Второй шаг предполагает пересчёт центров как среднего арифметического входящих в кластер точек, и вот здесь будет подвох: к оптимальности выбора центров в среднем арифметическом приводит именно евклидова.

Однако на практике никто не мешает использовать метод и без должного обоснования, поэтому можно экспериментировать с любыми расстояниями, с той лишь оговоркой, что не будет никаких теоретических гарантий, что метод сработает. Наиболее распространённая альтернатива евклидовой метрике — это косинусная мера близости векторов (она особенно популярна в задачах анализа текстов)

Искать центроиды удобно и просто, но в реальных задачах кластеры могут быть совсем не круглой формы. Вот вы геолог, которому нужно найти на карте схожие по структуре горные породы — ваши кластеры не только будут вложены друг в друга, но вы ещё и не знаете сколько их вообще получится.

Хитрым задачам — хитрые методы. [DBSCAN](https://habr.com/post/322034/), например. Он сам находит скопления точек и строит вокруг кластеры. Его легко понять, если представить, что точки — это люди на площади. Находим трёх любых близко стоящих человека и говорим им взяться за руки. Затем они начинают брать за руку тех, до кого могут дотянуться. Так по цепочке, пока никто больше не сможет взять кого-то за руку — это и будет первый кластер. Повторяем, пока не поделим всех. Те, кому вообще некого брать за руку — это выбросы, аномалии. В динамике выглядит довольно красиво:

### DBSCAN

Алгоритм [DBSCAN](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN) рассматривает кластеры , как участки высокой плотности , разделенных районах с низкой плотностью. Из-за этого довольно общего представления кластеры, обнаруженные с помощью DBSCAN, могут иметь любую форму, в отличие от k-средних, которое предполагает, что кластеры имеют выпуклую форму. Центральным компонентом DBSCAN является концепция *образцов керна* , то есть образцов, находящихся в областях с высокой плотностью. Таким образом, кластер представляет собой набор образцов керна, каждый из которых находится близко друг к другу (измеряется с помощью некоторой меры расстояния), и набор образцов, не относящихся к керну, которые близки к образцу керна (но сами не являются образцами керна). В алгоритме есть два параметра, min\_samples и eps, которые формально определяют, что мы имеем в виду, когда говорим «*плотный»* . Выше min\_samples или ниже eps указывают на более высокую плотность, необходимую для формирования кластера.

Более формально мы определяем образец керна как образец в наборе данных, так что существуют min\_samples другие образцы на расстоянии eps, которые определены как *соседи* образца керна. Это говорит нам о том, что основной образец находится в плотной области векторного пространства. Кластер — это набор образцов керна, который можно построить путем рекурсивного взятия образца керна, поиска всех его соседей, которые являются образцами керна, поиска всех *их* соседей, которые являются образцами керна, и т. Кластер также имеет набор неосновных выборок, которые представляют собой выборки, которые являются соседями керновой выборки в кластере, но сами не являются основными выборками. Интуитивно эти образцы находятся на периферии кластера.

Любой образец керна по определению является частью кластера. Любая выборка, не являющаяся образцом керна и находящаяся по крайней мере eps на расстоянии от любой выборки керна, считается алгоритмом выбросом.

Хотя параметр в min\_samples первую очередь контролирует устойчивость алгоритма к шуму (на зашумленных и больших наборах данных может быть желательно увеличить этот параметр), параметр epsимеет *решающее значение для правильного выбора* для набора данных и функции расстояния и обычно не может быть оставлен на значение по умолчанию. Он контролирует локальное окружение точек. Если выбран слишком маленький размер, большая часть данных вообще не будет кластеризована (и помечена как -1 для «шума»). Если выбран слишком большой, близкие кластеры будут объединены в один кластер, и в конечном итоге весь набор данных будет возвращен как единый кластер. Некоторые эвристики для выбора этого параметра обсуждались в литературе, например, на основе перегиба на графике расстояний до ближайших соседей (как обсуждается в ссылках ниже).

На рисунке ниже цвет указывает на принадлежность к кластеру, а большие кружки обозначают образцы керна, найденные алгоритмом. Меньшие кружки — это неосновные образцы, которые все еще являются частью кластера. Более того, выбросы обозначены ниже черными точками.

Как и классификация, кластеризация тоже может использоваться как детектор аномалий. Поведение пользователя после регистрации резко отличается от нормального? Заблокировать его и создать тикет саппорту, чтобы проверили бот это или нет. При этом нам даже не надо знать, что есть «нормальное поведение» — мы просто выгружаем все действия пользователей в модель, и пусть машина сама разбирается кто тут нормальный.

Работает такой подход, по сравнению с классификацией, не очень. Но можно попробовать, вдруг получится.

**ПРАКТИКА**

DBSCAN (Density-based spatial clustering of applications with noise, плотностной алгоритм пространственной кластеризации с присутствием шума), как следует из названия, оперирует плотностью данных.

Его преимуществом перед алгоритмом K-means (Ллойда) в том, что он позволяет выделять кластеры произвольной формы и, кроме того, сам определяет число кластеров в данных.

Все начинается с понятия эпсилон-окрестности объекта. Для любого вектора X  в метрическом признаковом пространстве эпсилон-окрестность определяется как множество точек, отстоящих от x не более чем на эпсилон

Разумеется, параметр эпсилон мы задаем сами, как параметр работы алгоритма DBSCAN. Далее, в алгоритме на основе эпсилон-окрестности объекты делятся на три типа:

* **корневой**: содержащий не менее m объектов в своей эпсилон-окрестности;
* **граничный**: не корневой, но в окрестности корневого;
* **шумовой** (выброс): не корневой и не граничный.

Предположим, у нас имеется некоторый набор данных в двумерном признаковом пространстве. Вначале мы случайным образом выбираем объект Xi из этого набора. Если в ε-окрестности этого объекта менее m других объектов, то он помечается как возможный шумовой. Затем, мы снова выбираем случайным образом объект (исключая ранее рассмотренные) и опять проверяем полноту его ε-окрестности. Если в ней находится не менее m других образов, то вектор помечается как корневой и для всех образов (точек), входящих в эту окрестность, процедура рекуррентно повторяется. Причем, если объект не содержит достаточного числа соседей в своей окрестности, то он помечается граничным, иначе – корневым. В результате, мы перебираем все объекты, которые захватываются заданной ε-окрестностью. Таким образом, формируется кластер.

Далее, процесс повторяется с самого начала, исключая ранее обработанные образы. Также случайно отбирается объект и формируется еще один кластер, либо шумовые образы. В итоге, после прохождения по всем объектам выборки, на выходе мы получаем разбиение данных на кластеры и шумовые образы, не вошедшие ни в один из кластеров. Причем, число кластеров было определено автоматически, исходя из заданных параметров ε и m.

Чтобы не изобретать велосипед. Конечно, если вам понадобиться использовать алгоритм DBSCAN на языке Python, то лучше воспользоваться библиотекой Scikit-Learn, в которой уже реализован этот алгоритм на уровне языка С++, а потому работает гораздо быстрее:

**import** **numpy** **as** **np**

**from** **sklearn.cluster** **import** DBSCAN

**from** **sklearn** **import** metrics

**from** **sklearn.datasets** **import** make\_blobs

**from** **sklearn.preprocessing** **import** StandardScaler

*# #############################################################################*

*# Generate sample data*

centers = [[1, 1], [-1, -1], [1, -1]]

X, labels\_true = make\_blobs(n\_samples=750, centers=centers, cluster\_std=0.4,

random\_state=0)

X = StandardScaler().fit\_transform(X)

*# #############################################################################*

*# Compute DBSCAN*

db = DBSCAN(eps=0.3, min\_samples=10).fit(X)

core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = **True**

labels = db.labels\_

*# Number of clusters in labels, ignoring noise if present.*

n\_clusters\_ = len(set(labels)) - (1 **if** -1 **in** labels **else** 0)

n\_noise\_ = list(labels).count(-1)

print('Estimated number of clusters: ***%d***' % n\_clusters\_)

print('Estimated number of noise points: ***%d***' % n\_noise\_)

print("Homogeneity: ***%0.3f***" % metrics.homogeneity\_score(labels\_true, labels))

print("Completeness: ***%0.3f***" % metrics.completeness\_score(labels\_true, labels))

print("V-measure: ***%0.3f***" % metrics.v\_measure\_score(labels\_true, labels))

print("Adjusted Rand Index: ***%0.3f***"

% metrics.adjusted\_rand\_score(labels\_true, labels))

print("Adjusted Mutual Information: ***%0.3f***"

% metrics.adjusted\_mutual\_info\_score(labels\_true, labels))

print("Silhouette Coefficient: ***%0.3f***"

% metrics.silhouette\_score(X, labels))

*# #############################################################################*

*# Plot result*

**import** **matplotlib.pyplot** **as** **plt**

*# Black removed and is used for noise instead.*

unique\_labels = set(labels)

colors = [plt.cm.Spectral(each)

**for** each **in** np.linspace(0, 1, len(unique\_labels))]

**for** k, col **in** zip(unique\_labels, colors):

**if** k == -1:

*# Black used for noise.*

col = [0, 0, 0, 1]

class\_member\_mask = (labels == k)

xy = X[class\_member\_mask & core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),

markeredgecolor='k', markersize=14)

xy = X[class\_member\_mask & ~core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),

markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: ***%d***' % n\_clusters\_)

plt.show()

### **Интуитивное объяснение**

В гигантском зале толпа людей справляет чей-то день рождения. Кто-то слоняется один, но большинство — с товарищами. Некоторые компании просто толпятся гурьбой, некоторые — водят хороводы или танцуют ламбаду.  
  
Мы хотим разбить людей в зале на группы.  
  
Но как выделить группы столь разной формы, да ещё и не забыть про одиночек? Попробуем оценить плотность толпы вокруг каждого человека. Наверное, если плотность толпы между двумя людьми выше определённого порога, то они принадлежат одной компании. В самом деле, будет странно, если люди, водящие «паровозик», будут относиться к разным группам, даже если плотность цепочки между ними меняется в некоторых пределах.  
  
Будем говорить, что рядом с некоторым человеком собралась толпа, если близко к нему стоят несколько других человек. Ага, сразу видно, что нужно задать два параметра. Что значит «близко»? Возьмём какое-нибудь интуитивно понятное расстояние. Скажем, если люди могут дотронуться до голов друг друга, то они находятся близко. Около метра. Теперь, сколько именно «несколько других человек»? Допустим, три человека. Двое могут гулять и просто так, но третий — определённо лишний.  
  
Пусть каждый подсчитает, сколько человек стоят в радиусе метра от него. Все, у кого есть хотя бы три соседа, берут в руки зелёные флажки. Теперь они коренные элементы, именно они формируют группы.  
  
Обратимся к людям, у которых меньше трёх соседей. Выберем тех, у которых по крайней мере один соседей держит зелёный флаг, и вручим им жёлтые флаги. Скажем, что они находятся на границе групп.  
  
Остались одиночки, у которых мало того что нет трёх соседей, так ещё и ни один из них не держит зелёный флаг. Раздадим им красные флаги. Будем считать, что они не принадлежат ни одной группе.  
  
Таким образом, если от одного человека до другого можно создать цепочку «зелёных» людей, то эти два человека принадлежат одной группе. Очевидно, что все подобные скопища разделены либо пустым пространством либо людьми с жёлтыми флагами. Можно их пронумеровать: каждый в группе №1 может достичь по цепочке рук каждого другого в группе №1, но никого в №2, №3 и так далее. То же для остальных групп.  
  
Если рядом с человеком с жёлтым флажком есть только один «зелёный» сосед, то он будет принадлежать той группе, к которой принадлежит его сосед. Если таких соседей несколько, и у них разные группы, то придётся выбирать. Тут можно воспользоваться разными методами — посмотреть, кто из соседей ближайший, например. Придётся как-то обходить краевые случаи, но ничего страшного.