

-
- Exploratory DA — анализ (чаще всего глазками) данных, чтобы найти в нём пропуски, выбросы и прочий мусор, который вы можете глазами найти. Тут же вы думаете о том, чем эти данные являются, и как их хранить. А ещё тут вы можете формировать базовые гипотезы.
 - Confirmatory DA — не рассказали, что такое.
 - Показательный анализ данных — машинное обучение с целью что-нибудь предсказывать.
 - Визуализация данных.

Знание — закономерность в некоторой области, которые позволяют решать проблемы.

Data Mining — один из этапов в извлечении знаний из данных. Тут три этапа:

1. Сбор данных.
2. Выделение признаков.
3. Применение ML.

Фактически это как DA, но с акцентом на немного другое.

Data Science:

- Сбор данных.
- Интеграция данных (data integration).
- Хранение данных (data warehousing).
- Анализ данных.
- Высокопроизводительные вычисления (high-performance computing).

Как это с точки зрения бизнеса выглядит? Есть CRISP-DM:

1. Понимание бизнеса (перевод слов заказчика на что-то понятное).
2. Понимание данных (требование правильных данных от заказчика).
3. Подготовка данных (понять, как хранить данные).
4. Моделирование (собственно, решать задачу).
5. Оценка (понять, что мы получили).
6. Если всё хорошо, внедряем, заворачиваем в красивую обёртку и т.п.

Данные. Данные бывают структурированные, неструктурированные и частично структурированные. Чаще всего мы встречаем неструктурированные. Это видео, посты в соцсетях и любая другая хрень, которую мы встречаем. Частично структурированные — XML, JSON и прочий мусор, который до полностью структурированных не дотягивает. А полностью структурированные данные — это матрицы (из n объектов по m характеристик у каждого).

Имея неструктурированные или частично структурированные данные, мы почти всегда хотим его структурировать, чтобы было удобнее.

В DA строку матрицы называют object, instance, sample, example и как угодно ещё, а столбец матрицы — feature, attribute, characteristic или factor.

Что важно про наши матрицы? Чтобы порядок столбцов и строк был не важен. Если ваш алгоритм не такой, это странно. Исключение: временной ряд.

Признаки. Глобально их два: категория и число (качество и количество). Первое дискретно, второе непрерывно, категории мы можем только на равенство проверять, а с числами можем арифметику делать. Категориям обычно целое число сопоставляют, чтобы удобно было. Но есть и другие типы, например, порядковый: дискретный, но есть порядок. Впрочем, очень часто он сводится к либо к категории, либо к числу.

С числами работать гораздо удобнее, но если надо, легко можно дискретизировать наше число любым образом (например, баллы в оценку — дискретизация). Как преобразовать порядковый тип во что-то? Можно преобразовать в число через порядковый номер. А можно в категорию, создав k бинарных значений вида x_i .

One-hot encoding. Берём категорию из k значений, преобразуем в k булевых категорий вида $x_i = c_i$, а потом из истины сделать 1, а из лжи — 0. Отличный план, как из категории сделать число.

Ещё можно преобразовывать иначе, например категории сопоставлять случайный вектор или число, бинарное представление которого используется как новые признаки.

Важное примечание: числа — не всегда числа. Например, если мы цифры распознаём, то плохо считать числа числами, а не категориями. Потому что изображение тройки не находится между изображением двойки и четвёрки. Поэтому аккуратно с преобразованием категорий в числа.

Ещё интересность: время. Если у нас есть периоды, то храним синус и косинус, чтобы периодически было:

$$f_{2e-1} = \sin \frac{2\pi t}{T_e} \quad f_{2e} = \cos \frac{2\pi t}{T_e}$$

Или ещё интересность, цвет. Цвет кодируем в RGB, а не в HSV/HSB, потому что там тон цвета неоднозначный. Красный — это и 0, и 360.

Как хранить данные? Самый простой формат: CSV. Он плохо стандартизирован, правда. А ещё его недостаток в том, что там шапки нет. Нет ни информации о типах, ничего. Если хочется более человеческого — ARFF: примерно как CSV, но более стандартизирован и имеет шапку с типами данных. Типы: Numeric (число), Nominal (категория), String и Date.

```
1  % 1. Title: Iris Plants Database
2  %
3  % 2. Sources:
4  %      (a) Creator: R.A. Fisher
5  %      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
6  %      (c) Date: July, 1988
7  %
8  @RELATION iris
9
10 @ATTRIBUTE sepallength NUMERIC
11 @ATTRIBUTE sepalwidth NUMERIC
12 @ATTRIBUTE petallength NUMERIC
13 @ATTRIBUTE petalwidth NUMERIC
14 @ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
15
16 @DATA
17 5.1,3.5,1.4,0.2,Iris-setosa
18 4.9,3.0,1.4,0.2,Iris-setosa
19 4.7,3.2,1.3,0.2,Iris-setosa
20 4.6,3.1,1.5,0.2,Iris-setosa
21 5.0,3.6,1.4,0.2,Iris-setosa
22 5.4,3.9,1.7,0.4,Iris-setosa
23 4.6,3.4,1.4,0.3,Iris-setosa
24 5.0,3.4,1.5,0.2,Iris-setosa
25 4.4,2.9,1.4,0.2,Iris-setosa
26 4.9,3.1,1.5,0.1,Iris-setosa
```

Прочие типы данных.

- Картинка. Самое простое — двух- (или трёх-)мерная матрица. И есть 1000 алгоритмов, как привести её в один вектор (притом довольно маленький).
- Текст. Тут сложнее. Самое тупое — посчитать частоту слов. Но не надо, тут сразу двигайте в сторону LLM.
- Звук. Это сигнал во времени, и самый простой вариант его представить — вектор значений. Для него тоже есть всякие штуки типа спектрограммы Мэла, чтобы преобразовывать звук во что-то более адекватное.
- Видео. Это мультимодальный объект, то есть объект, в котором разные типы внутри. Их анализировать тоже своими методами.

Нормализация. Зачем? Чтобы признаки с большей дисперсией не влияли на результат сильнее остальных.

Как? Сначала отбросим единицы измерения. Чтобы это было корректно, алгоритм должен быть инвариантен относительно линейных преобразований над признаками. А дальше надо как-то нормализовать дисперсию. Можно, например, вычесть из каждого признака минимум и поделить на разность между максимумом и минимумом. Это $[0; 1]$ -масштабирование. А ещё можно вычесть (выборное) мат. ожидание из признака и поделить на (выборное) стандартное отклонение это Z-масштабирование. Способов ещё куча, но это два простых.

Как жить с ситуацией, когда у нас признаки зависимы? Никак не жить, если так, вы плохо собрали данные.

Веса и сэмплирование. Отлично, мы всё нормализовали, а теперь у нас из специфики задачи какие-то признаки более важны. Или могут быть объекты более важны.

Говорят, что объект/признак обладает весом w , если он в w раз больше влияет на результат. Если w целое, то это эквивалентно тому, что объект/признак встречается w раз.

Примеры: учёт веса признака

$$\text{dist}(a; b) = \sqrt{\sum_{j=1}^m w_j (a_j - b_j)^2}$$

учёт веса объекта:

$$\mathcal{L}_D(\theta) = \sum_{x \in D} w(x) \mathcal{L}(x; \theta)$$

А есть сэмплирование: вместо вхождения кучу раз в датасет мы можем сэмплировать объекты с вероятностью $\frac{w_j}{\sum w_j}$.

Ещё это может использоваться для балансировки. Если вы знаете, что реальное распределение не очень сочетается с распределением данных, то мы можем пореже выбирать объекты мажоритарного класса или почаще объекты миноритарного класса (например, в данных одна группа большая, а другая — маленькая, а в жизни они равны, и мы даём большие веса малой группе). Да и в принципе, если нас не интересует распределение признаков по классам, распределение будет чаще подсовывать вам штуки из мажоритарного класса, что вам не всегда хочется.

Как сэмплировать? Можно случайно, можно каждый x -тый объект (систематически), можно стратифицированно (ровно в соотношении с какой-то категорией), можно кластерное (если кто-то уже за нас разбил объекты на множества, их множеств можно выбрать несколько).

Шум. Аномалии — плохие объекты для модели. Ошибки — плохие объекты для реальности. Пример: пусть у нас есть расход топлива автомобилями. И у нас у одного указан 30л / 100км. Если это грузовой автомобиль, а остальные — легковые, то это аномалия. А если объект всем хорош, и данные такие, то это ошибка (а возникнуть такое могло из-за того, что кто-то подразумевал мили на галлон). Пропуски в наборе данных. Откуда? Из склейки разных наборов данных, из изначально херовых данных или откуда угодно ещё. В CSV нет стандартного способа это обозначить, в ARFF это вопросик, а ещё можно специальное значения: None/Null для строк, категория с несуществующим номером для

категорий или NaN для числа. Что делать с этим? Вырезать (признак или объект), заменить (заполнить пропуск средним арифметическим/модой/алгоритмом предсказания, который умеет в пропуски) или добавить (добавить новый признак, который говорит, были ли пропуск или нет; если там была категория, можно отдельное значение категории туда добавить, а не новое свойство делать).

1 Терминология.

Виды искусственного интеллекта.

- **Слабый (узкий) ИИ.** Нацелен на решение специфичной задачи, причем способ ее решения не подходит для решения других задач.
- **Сильный (общий) ИИ.** Способен решать множество задач, достигает или превосходит человеческие способности.
- **AI-полная задача** — задача, решение которой предполагает создание сильного AI.
- **Проблема ускользающей цели** — по мере решения задач ИИ их выписывают из списка задач сильного ИИ и относят к задачам слабого ИИ.
- **Интеллектуальная система** — система, решающая одну или несколько задач ИИ.
- Помимо машинного обучения выделяют **экспертную систему**, построенную на основе фактов и правил, извлекаемых из экспертов.

Отличия экспертной системы от ML:

- Экспертная система строится не от частного к общему, как ML, а от общего к частному.
- Экспертная система получает решение не с помощью обучения на большом количестве данных, а с помощью привлечения экспертов (лингвистов) для формализации правил и выстраивания паттернов.

Связь ML и других дисциплин.. Статистика. Цель статистики — на основе большого числа данных выявить паттерны, по которым строятся эти данные (т.е. распределения и другое). ML не всегда основано на статистике, но часто является «*более сильной версией*» статистики.

Анализ данных. Пока ML — разработка алгоритмов, DA — это их применение.

Устройство машинного обучения глобально. Определение 1: машинное обучение — процесс, дающий компьютерам способность обучаться новому, не будучи непосредственно запрограммированными делать это.

Определение 2: программа обучается с опытом E решению некоторой задачи T , по метрике качества P , если качество ее решения задачи T , измеренное согласно P , растет вместе с увеличением E .

Задача машинного обучения состоит из:

- Набора данных \mathcal{D} .
- Целевой функции — либо функция качества (выигрыша, правдоподобия) \mathcal{Q} , либо функция потерь (ошибок, риска) \mathcal{L} .

Алгоритм учится решать задачу, если он максимизирует функцию качества (или минимизирует функцию потерь) для заданного набора данных ($\mathcal{Q}_{\mathcal{D}}$ или $\mathcal{L}_{\mathcal{D}}$ соответственно).

В задачах оптимизации у нас есть некая функция потерь \mathcal{L} , для которой мы хотим найти минимум, то есть такой вектор параметров θ , что $\mathcal{L}(\theta) \rightarrow \min$. В задачах машинного обучения помимо параметров

есть еще данные. Если мы поменяем датасет, минимум функции тоже поменяется. Чтобы их учесть, давайте на каждом объекте набора данных измерим функцию потерь для этого объекта с конкретными параметрами и будем минимизировать сумму этих значений по всем объектам:

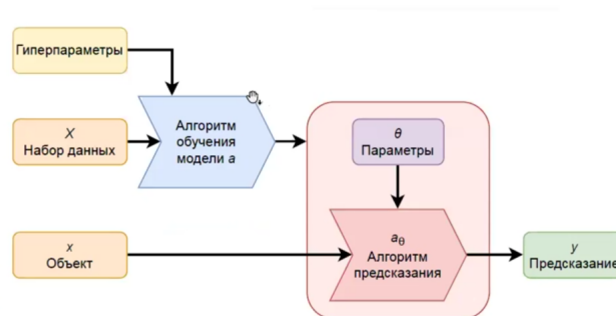
$$\mathcal{L}_{\mathcal{D}}(\theta) = \sum_{x \in \mathcal{D}} \mathcal{L}(x, \theta) \rightarrow \min$$

Эту сумму можно было бы поделить на размер набора данных, но поскольку он всегда константный вне зависимости от θ , можно минимизировать без него.

Аппроксимация функции ошибки. Иногда функция качества или функция потерь, которую сообщает заказчик, не формализуется или слишком сложно вычислима. В таком случае выбирается другая функция, которая определяется конкретной задачей машинного обучения. Почему так норм? Ну, например, потому, что настоящая функция потерь уже наверняка была упрощена при формализации потери. Или потому, что при подсчёте истинной функции потерь мы будем работать непростоительно долго.

Аппроксимация применяется не только по отношению к функциям: например, в реальной жизни данных может быть потенциально бесконечное количество, но в задаче мы ограничиваемся конечным набором (выборкой).

Алгоритм машинного обучения в деталях. Обычно у алгоритмов в жизни есть параметры и ввод. И исходя из них алгоритм выдаёт какой-то вывод. Например, утилите `ffmpeg` вы скармливаете видеофайл(ы) и опции вида `-b:v`, `-b:a` и любые другие. А она вам выдаёт какие-то другие видеофайл(ы). Тут ситуация несколько другая:



Мы хотим сделать алгоритм предсказания, который по объекту делает предсказания. Но тут у нас параметры нам даёт не пользователь, а мы делаем их сами. А точнее, пишем специальный алгоритм (алгоритм обучения), который их нам и делает. И это уже обычный алгоритм, с обычным входом и выходом. Его вход — набор данных, его выход — параметры алгоритма предсказания, а его параметры называются специальным словом **«гиперпараметры»**. То есть параметры — это параметры модели, которые меняются в процессе обучения, а гиперпараметры — фиксированные параметры алгоритма обучения.

Пример: аппроксимация полиномом. Мы решили, что наш алгоритм предсказания будет выглядеть как

$$y = a + bx + cx^2 + dx^3 + \dots$$

Где x — вход, y — выход, а a, b, c, d, \dots — параметры. И для обучения мы себе выбрали, что будем искать только полиномы размерности 2. Тогда вот это вот «2» — гиперпараметр, точки — набор данных, а $a, b, c, 0, 0, \dots$ будут нашими параметрами.

Уровни модельности алгоритмов.

1. **Безмодельные алгоритмы** («in-place») решают задачу без явного построения модели. Самый тупой пример — нахождение матожидания и использование его как результат всех предсказаний. То есть in-place алгоритм не имеет обучения и параметров, он просто запоминает данные и гиперпараметры, и сразу на основе них делает предсказание.
2. **Моделирующие алгоритмы** обучаются один раз и строят модель, которую потом можно переиспользовать без повторного обучения, но нельзя обновить при поступлении новых данных. Пример — алгоритмы кластеризации.
3. **Алгоритмы онлайн-обучения** имеют возможность переобучаться при добавлении новых данных.
4. Возможность объединить две модели.

Определение понятий машинного обучения. Непонятно, что называть решением задачи машинного обучения: алгоритм обучения модели, обученную модель или результат ее применения? Также непонятно, от чего вычисляется функция оценки качества: от алгоритма, от модели или от ее применения. Ответ: зависит от ситуации. Обозначим за A алгоритм обучения, за θ — полученные в результате обучения параметры, за a_θ — полученную в результате обучения модель с параметрами θ . Введем следующие функции оценки качества:

1. $\mathcal{L}(A, \mathcal{D})$ — функция от алгоритма обучения
2. $\mathcal{L}_{\mathcal{D}}(a_\theta)$ — функция от обученной модели
3. $\mathcal{L}(\hat{y}, y)$, $\hat{y} = a_\theta(x)$ — функция для результата применения, то есть отклонение истинного результата от предсказанного обученной моделью

Очевидно, нам хочется, имея одну из этих функций, вычислять остальные. Поэтому введем три понятия:

1. **Валидация** — вычисление $\mathcal{L}(A, \mathcal{D})$ из $\mathcal{L}_{\mathcal{D}}(a_\theta)$.
2. **Тестирование модели** a_θ на наборе данных \mathcal{D} — вычисление $\mathcal{L}_{\mathcal{D}}(a_\theta)$ из $\mathcal{L}(\hat{y}, y)$.
3. **Применение** — вычисление $a_\theta(x)$.

Валидация на отложенных данных. Пусть мы хотим понять обобщающую способность нашей модели, но у нас есть ограниченный набор данных. Мы хотим убедиться, что наша модель будет выдавать хорошие результаты за пределами датасета, на котором ее обучали (для этого мы ее и обучаем). Обычно в таких случаях исходный датасет разбивают на training и test часть случайным образом. (Вообще, необязательно случайным; например, если объекты могут составить временной ряд, то разбивать объекты лучше по времени.) Тогда:

$$\mathcal{L}(A, \mathcal{D}) = \mathcal{L}(A(\mathcal{D}_{\text{train}}), \mathcal{D}_{\text{test}})$$

Сравнение алгоритмов. Чтобы сравнить в машинном обучении два алгоритма, один из которых не является частным случаем другого (тогда очевидно, что общий алгоритм лучше), нужно, во-первых, ввести метрику их сравнения, а во-вторых сравнивать на одинаковых наборах данных. Под метрикой подразумевается уже знакомая нам функция качества от алгоритма $\mathcal{L}(A, \mathcal{D})$, а также способ ее вычисления. Часто при решении задачи берут какой-то наивный алгоритм (baseline) и сравнивают остальные алгоритмы с ним. Теорема «No free lunch theorem» гласит, что алгоритм может работать хорошо только на определенном наборе данных, а на остальных данных проседает, т.е. нельзя покрыть одним алгоритмом все, нужно выбирать исходя набора данных в данной задаче. Лучший алгоритм для конкретной задачи с конкретным набором данных и конкретной метрикой называется State-of-the-art (SOTA).

$a: X \rightarrow Y$	$Y = \{y_1, \dots, y_k\}$	$Y = \mathbb{R}^k$	$Y = \mathbb{R}^k$
Обучение с учителем	Классификация	Мягкая классификация	Восстановление регрессии
One-class classification	Поиск аномалий	Восстановление плотности	Генерация объектов
Обучение без учителя	Кластеризация	Мягкая кластеризация	Выделение признаков

Стандартные задачи машинного обучения. Что здесь происходит? Задачу машинного обучения можно представить как отображение из X в Y . В качестве столбцов у нас то, что может быть результатом:

- $Y = \{c_1, c_2, \dots, c_n\}$: результатом является сопоставление класса каждому объекту из набора.
- $Y = \mathbb{R}^k$: результатом является сопоставление вектора вероятностей размера k каждому объекту из набора.
- $Y = \mathbb{R}^k$: результатом является одно число или вектор чисел размера k .

В качестве строк выступает то, что служит источником для построения модели:

Обучение с учителем В качестве источника выступают размеченные данные, то есть выборка, содержащая верные ответы. Цель алгоритма — научиться эти правильные ответы предсказывать.

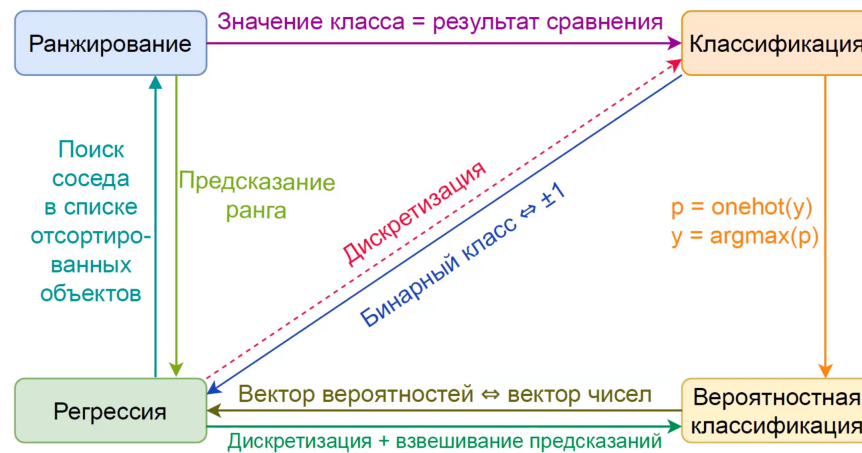
- **Задача классификации** — задача сопоставления каждому объекту категории на основе распределения по категориям в обучающем датасете. Наивное решение — всегда брать моду (самый популярный класс). Примеры: определить, является ли письмо спамом или определить, какая цифра на картинке.
- **Задача мягкой классификации** — задача сопоставления объекту вектора, где каждое число обозначает уверенность в попадании объекта в конкретный класс. Выделяют вероятностную классификацию, где вектор должен быть корректным вероятностным вектором.
- **Задача восстановления регрессии** — задача предсказания ответа для произвольного объекта на основе сопоставления ответов объектам в обучающем датасете. Наивное решение — брать мат. ожидание.

Ещё в случае обучения с учителем Y может быть перестановкой или рангом. В таком случае мы пытаемся установить порядок на наших объектах. Используется это крайне редко.

Особняком стоят задачи прогнозирования временных рядов, где при имеющихся $y_{t-m}, \dots, y_{t-2}, y_{t-1}$ надо предсказать y_t . Самый яркий пример — цены акций. Особняком они потому, что у них нет признаков (кроме, собственно, y_t). Где такие взять? Ну, например, считать, что

$$x_t = (y_{t-m}, \dots, y_{t-2}, y_{t-1})$$

Это называется авторегрессия. Или можно брать какую-то функцию от t , это конструирование признаков. Наивное решение — арифметическое скользящее среднее или экспоненциально скользящее среднее.



One-class classification В качестве источника выступают данные, почти все из которых принадлежат одному классу (известно для каждого объекта, принадлежит он или нет).

- **Задача поиска аномалий** — среди существующих объектов нужно найти объекты другого класса.
- **Задача новизны** — среди каких-то новых объектов нужно найти объекты другого класса.
- **Задача восстановления плотности** — требуется восстановить плотность распределения и для новых объектов определить, кто из них выбивается из распределения. Обычно используется метод максимального правдоподобия, где мы оцениваем параметры распределения на основе объектов из выборки. Подробнее — на матстате.
- **Задача генерации новых объектов** — по данным объектам нужно сгенерировать новые.

Не путайте генерацию с сэмплированием (выбрать объект из существующих) и аугментацией (примитивными аналитическими операциями сгенерировать новые объекты, например, повернуть картинку котика, чтобы получить новую картинку котика). Это другие вещи, но их можно использовать как наивное решение.

Из интересного, чтобы оценить качество генерации используются алгоритмы классификации (на два класса: реальный объект или сгенерированный). А ещё можно сохранять некоторую статистику между реальными объектами, тогда классификация разделяется на два вида моделей:

- **Генеративные** — обучают совместное распределение вероятностей, и задача классификации сводится к задаче восстановления плотности.
- **Дискриминативные** — обучают условное распределение, и модель пытается найти разделяющее правило.

Обычно используются в паре, но об этом подробнее во втором семестре.

Обучение без учителя. В качестве источника выступают неразмеченные данные, для которых требуется самостоятельно придумать целевой признак \hat{Y} . И дальше уже распределение на задачи ведётся в зависимости от того, каким является наш \hat{Y} (а не Y , как было в таблице).

- **Задача кластеризации** — требуется разделить объекты на подмножества (кластеры), чтобы в каждом кластере объекты были максимально похожи друг на друга. Пример: вы пишете Твиттер, и хотите разделить пользователей по информационным пузырям.
- **Задача мягкой кластеризации** — аналогично мягкой классификации.

- **Задача выделения признаков** — алгоритм должен отображать объект из X в пространство (чаще всего меньшей размерности), которое он сам и придумает. Самое наивное — умножим на случайную матрицу. Используется для уменьшения размерности, чтобы, например, нарисовать наши данные.
- **Задача конструирования признаков** — по сути, более общая задача извлечения признаков: дана какая-то абстрактная штука (картинка, текст, etc), хочется сделать себе вектор признаков. Решается явно, а не алгоритмами МЛ. Пример: пытаемся наложить шаблон на изображения, суммируем по всем возможным наложениям. Получаем новый признак для каждого шаблона. А вот шаблоны уже можно искать машинным обучением.

Другие задачи машинного обучения.

- Можно не только принимать на вход пропуски, но и возвращать. Трактуются как отказ от классификации/кластеризации. Первое используется в ансамблях, второе — в поиске аномалий.
- **Предсказание и заполнение пропусков.** Выберем признак с пропусками целевым, остальные заполним как-нибудь, натренируем модель на данных без пропусков, заполним пропуски результатом тестового прогона.
- **Коллаборативная фильтрация.** Дано множество оценок пользователями. И оценок, блин, мало (существенно меньше произведения числа пользователей и вещей). Требуется предсказать оценку данной вещи данным пользователем. Решается трудно, этим занимается специальный раздел (рекомендательные системы), очевидно, тем же алгоритмом, что заполнение пропусков, оно не решается, зато наоборот (заполнить пропуски совместной фильтрацией) — норм план.
- **Обучение на привилегированных данных.** Тренировочные данные содержат дополнительную информацию (X'), недоступную при тестировании. Базовое решение: либо не использовать X' , либо взять другую модель, которая предсказывает X' , и пользоваться ей, либо честно самому предсказывать и X' , и Y . Пример: предсказываем результат футбольного матча, и наши привилегированные данные — число красных и жёлтых карточек, например. На тестовых данных такое неизвестно.
- **Обучение на частично размеченных данных** (оно же **обучение с частичным привлечением учителя**). Лишь малая часть тренировочных данных (и никакая часть тестовые данные) содержит правильный ответ. Наивное решение: не использовать разметку (т.е. обучаться полностью без учителя) или не использовать неразмеченные объекты.
- **Активное обучение** — вариация на тему предыдущего. Можно задавать Оракулу вопросы о значении меток, и надо за минимум обращений к Оракулу восстановить $f: X \rightarrow Y$. То есть по сути надо не функцию найти даже, а стратегию выработать, как обращаться к Оракулу.
- **Обучение с подкреплением (Reinforcement Learning).** Есть агент и есть среда. Агент взаимодействует со средой, среда меняет состояние и возвращает награду. Задача: максимизировать награду. Пример: задача об одноруком бандите, он что-то знает об одноруком бандите, и должен выработать стратегию, как максимизировать выигрыш.