Содержание

1 Основы.

- 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 любом 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 $< x$.

One-hot encoding. Берём категорию из k значений, преобразуем в k булевых категорий вида $= c_i$, 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 хреначим синус и косинус, чтобы периодично было:

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

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. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
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:

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

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 рандомно, можно каждый x-тый объект (систематически), можно стратифицированно (ровно в соотношение 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).

2 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 $\mathcal{L}_\mathcal{D}$ соотвественно).
В задачах оптимизации у нас есть некая функция потерь $\mathcal{L}$, для которой мы хотим найти минимум, то есть такой вектор параметров θ, что $\mathcal{L}(\theta) \to \min$. 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:

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

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 непростительно 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 сделает. 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

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

Где x — вход, y — выход, а $a, b, c, d, \ldots$ — параметры. И для обучения мы себе выбрали, что будем искать только полиномы размерности 2. Тогда вот это вот «2» — гиперпараметр, точки — набор данных, а $a, b, c, 0, 0, \ldots$ 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. $\mathcal{L}(\widehat{y}, y)$, $\widehat{y} = a_\theta(x)$ — 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 $\mathcal{L}(\widehat{y}, y)$.
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).

$a\colon X \to Y$	$Y = \{y_1, \ldots, y_k\}$	$Y = \Pr^k$	$Y = \mathbb{R}^k$
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:

- $Y = \{c_1, c_2, ..., c_n\}$: результатом является сопоставление класса каждому объекту из набора.
- $Y = \Pr^k$: 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:

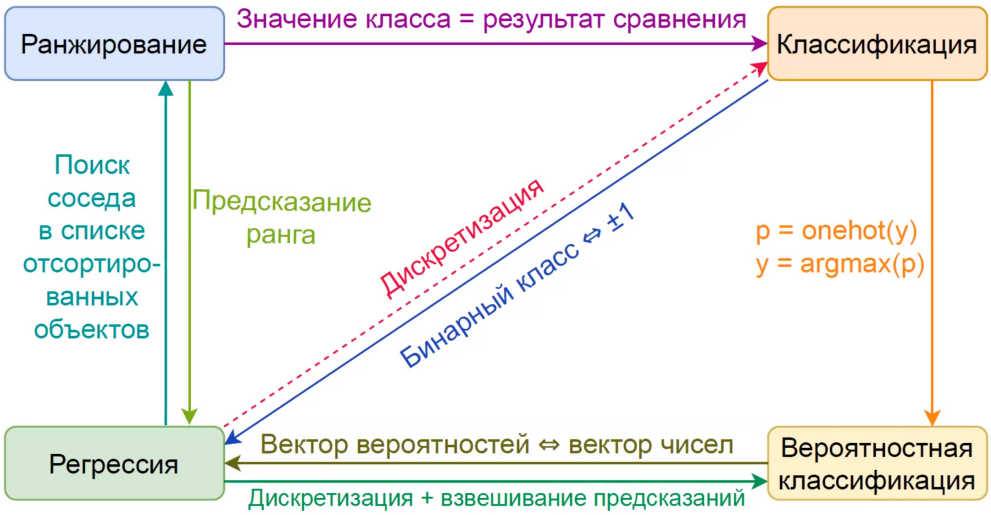
Обучение с учителем В качестве источника выступают размеченные данные, то есть выборка, содержащая верные ответы. Цель алгоритма — научится эти правильные ответы предсказывать.

- **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 и 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 пытается установить порядок на наших объектах. Используется это крайне редко.
Особняком стоят задачи прогнозирования временных рядов, где при имеющихся $y_{t-m}, \ldots, y_{t-2}, y_{t-1}$ 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

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

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 принадлежит 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 $\widehat{Y}$. И дальше уже распределение на задачи ведётся в зависимости от того, каким является наш $\widehat{Y}$ (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 рандомную 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 ML. 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. Трактуется 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 восстановив $f\colon X \to Y$. 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.

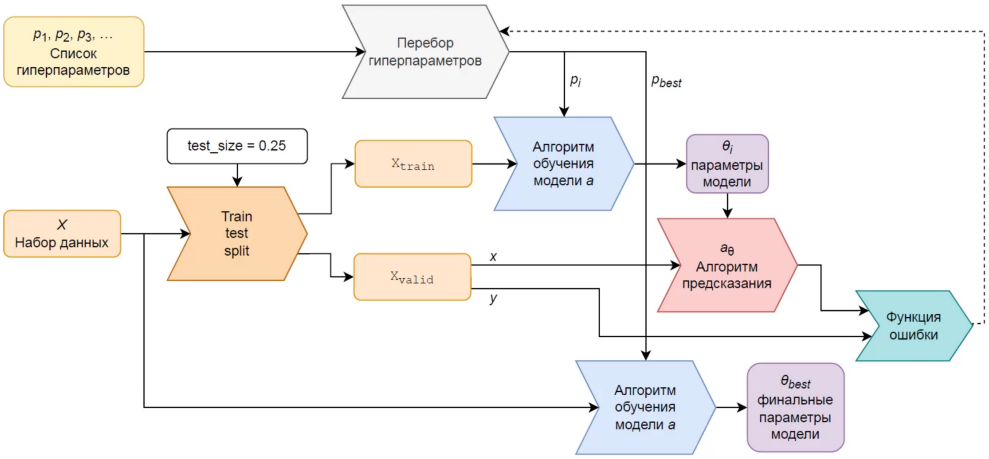
3 Обучение с учителем.

Если алгоритм работает хуже или сравнимо с наивным (наивный, напомню, берёт моду / мат. ожидание меток), значит или алгоритм плох (и не выявляет зависимость Y от X), либо данные плохи (и в них Y не зависит от X).
Сведение к бинарной классификации: пусть у нас есть алгоритм бинарной вероятностной классификации. Как из него многоклассовую сделать?

- «Один против всех». Заданный класс помечаем как единицу, остальные — как минус единицу. Возьмём k таких классификаторов, у кого вероятность больше, того и берём.
- «Один против одного»: обучим $\frac{k(k-1)}{2}$ классификаторов $a^b_{u,v}$: для каждой пары классов u, v выберем множество объектов $X_{u,v}$, принадлежащих одному из этих классов. Обучим на этом множестве классификатор $a^b_{u,v}$ предсказывать вероятность того, что объект принадлежит u, после чего сделаем себе
$$a(x) = \operatorname*{argmax}_c \prod_v a^b_{c,v}(x) \cdot \prod_u (1 - a^b_{u,c}(x))$$
Тут обучаем много классификаторов, но для каждого выборки поменьше.

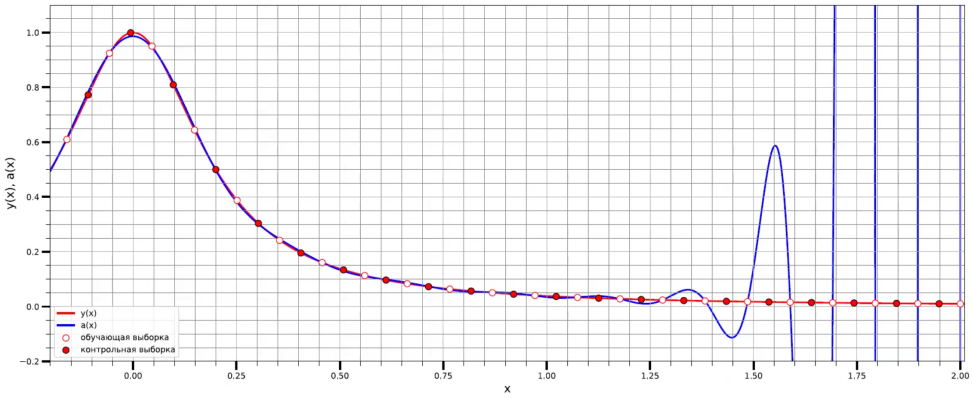
На мягкой классификации тоже работает.

Настройка гиперпараметров как часть обучения. Помните красивую идеалистичную картинку про гиперпараметры, параметры и данные? Так вот на самом деле ситуация иная:

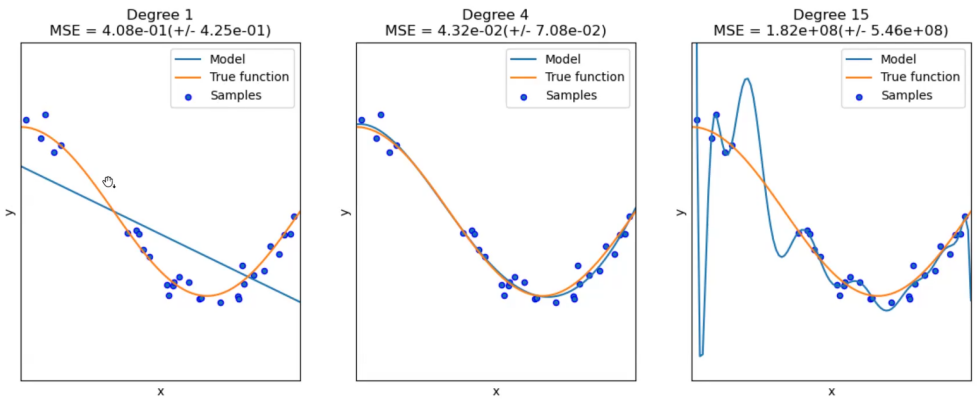


Естественно, гиперпараметры тоже надо настраивать, и не руками. Делим нашу выборку на две части: X_{train} и X_{valid}. И гиперпараметры у нас не одни, а несколько комбинаций. Мы кидаем этот набор в некоторый алгоритм перебора гиперпараметров, он нам даёт какие-то гиперпараметры, кидаем его в обучение, из них и X_{train} получаем параметры. Дальше из алгоритма предсказания и X_{valid} получаем функцию ошибки. Из неё меняем гиперпараметры. Повторим до тех пор, пока всё не будет хорошо. Когда будет — засунем уже весь набор в обучение и получим финальные параметры модели.
X_{valid} называется **валидационным множеством**. Иногда финальные параметры модели не берут заново, а берут ровно последний получившийся из обучения на X_{train} алгоритм. Ещё из интересного алгоритм обучения обычно неявно настраивает гиперпараметры (если может это сделать). Например, градиационный спуск имеет гиперпараметром количество шагов, но он итеративный алгоритм, и он сам примерно знает, когда его останавливать.
Начальное состояние генератора псевдослучайных чисел нельзя настраивать!
Как настраивать гиперпараметры? Поиск по сетке, случайный поиск эволюционные алгоритмы, или, из современного, байесовская оптимизация.

Переобучение и регуляризация. Начиная с определённого уровня сложности модели, чем лучше он работает на тренировочных данных, тем хуже на реальных. Пример: у нас есть $y(x) = \frac{1}{1+25x^2}$. Мы не знаем y. И мы аппроксимируем это чудо полиномом. Притом всё делаем на $[-2; 2]$. Тогда где-то до степени 15 ошибка на контрольной выборке падает, а потом резко начинает расти. Выглядит это примерно так:



Тут взята степень где-то 32, белые точки — обучающая выборка, красные — контрольная. Видно, что ошибка модели на контрольной модели огромна, а на обучающей она примерно ноль.
Аналогично есть недообучение, когда нам просто не хватает сложности модели:



Тут в качестве ошибки берётся MSE, и показаны значения ошибки для недообученной и переобученной модели.
В реальной жизни такие картиночки мы не построим, придётся бороться как-то иначе. Для этого есть регуляризация. Это добавление ограничений с целью предотвратить переобучение. ПО сути это штраф за сложность модели:

$$\mathcal{L}_{\text{reg}}(\theta, \mathcal{D}) = \mathcal{L}(\theta, \mathcal{D}) + \lambda \cdot \text{complexity}(\theta)$$

Тут λ — константа, и мы минимизируем $\mathcal{L}_{\text{reg}}(\theta, \mathcal{D})$.
Это мягкая регуляризация, а есть ещё жёсткая: минимизировать $\mathcal{L}(\theta, \mathcal{D})$ при условии, что $\text{complexity}(\theta) < C$ (C — новый гиперпараметр, являющийся ограничением на сложность).

Статистическое обоснование переобучения.

- **Смещение** (bias) — погрешность оценки, возникающая в результате ошибочного предположения в алгоритме. Высокое смещене — недообучение.
- **Дисперсия** (variance) — ошибка чувствительности к малым отклонениям в тренировочным наборе. Высокая дисперсия — переобучение.

Откуда это всё? Пусть f — реальная зависимость, $\widehat{y}$ — предсказание модели, $y = f(x) + \varepsilon$ — измерение с шумом ε, где $\mathsf{E}[\varepsilon] = 0$, $\mathsf{D}[\varepsilon] = \sigma^2$. Тогда

$$\mathsf{E}[y] = f \qquad \mathsf{D}[y] = \sigma^2$$

А отсюда ожидание квадрата ошибки:

$$\begin{aligned}
\mathsf{E}\left[(y - \widehat{y})^2\right] &= \mathsf{E}\left[y^2 + \widehat{y}^2 - 2y\widehat{y}\right] = \\
&= \mathsf{E}\left[y^2\right] + \mathsf{E}\left[\widehat{y}^2\right] - \mathsf{E}\left[2y\widehat{y}\right] = \\
&= \mathsf{D}[y] + f^2 + \mathsf{D}[\widehat{y}] + \mathsf{E}[\widehat{y}]^2 - 2f\,\mathsf{E}\left[\widehat{y}\right] = \\
&= \mathsf{D}[y] + \mathsf{D}[\widehat{y}] + \left(f - \mathsf{E}[\widehat{y}]\right)^2 = \\
&= \sigma^2 + \mathsf{D}[\widehat{y}] + \text{Bias}[\widehat{y}]^2
\end{aligned}$$

То есть тут неустранимая погрешность измерений, дисперсия модели и квадрат смещения модели.

Двойной спуск. Не так давно умные люди показали, что с определённой точки ошибка на тестовых данных снова начнёт падать. Нам это полезно, чтобы нейронные сети не переобучались (потому что в них параметров огромное количество, и возникает вопрос, почему они не переобучаются). Кому очень интересно, почитайте Belkin M., Shu, D.J., Ma, S., & Mandal, S. (2018). Reconciling modern machine learning and the bias-variance trade-off.
Но работает это только для супер-больших моделей, пока мы не пишем нейронки, у нас всё ещё есть классический bias-variance trade-off.

Перекрёстная проверка. Пусть $|\mathcal{D}_{\text{train}}| = r$, $|\mathcal{D}_{\text{test}}| = e$. Разобьём наш набор данных всеми возможными способами на $\mathcal{D}_{\text{train}}$ и $\mathcal{D}_{\text{test}}$. Тогда хотелось бы в качестве валидации использовать

$$\mathcal{L}(A, \mathcal{D}) = \frac{1}{\binom{e+r}{e}} \sum_{\mathcal{D}=\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}} \mathcal{L}(A(\mathcal{D}_{\text{train}}), \mathcal{D}_{\text{test}})$$

Тут мы точно не потеряем обобщающую способность. Но тут нам надо построить и обучить огромное количество моделей, поэтому такой подход (полная кросс-валидация) не используется.
Вместо этого используют кросс-валидацию по k блокам: распилим наш набор данных на k частей, каждая часть будет тестовой ровно один раз. Модель мы будем строить всего k раз таким образом. k бывает 5 либо 10.
Крайний случай — валидация по одному объекту Это полная кросс-валидация, где $e = 1$.
Ещё бывает стратифицированная кросс-валидация. Если объекты каких-то классов недостаточно часто встречаются, лучше разбивать на блоки так, чтобы в каждом блоке эта статистика была пропорциональна статистике выборки.
Если у нас задача временного ряда (т.е. ранние блоки дают более плохой результат), то тут среднее по всем разбиениям заменяется на среднее взвешенное среднее с экспоненциально растущими весами блоков.
Важно:

- Усреднять суммой можно только аддитивные функции ошибки/качества.
- Для усреднения можно использовать t кросс-валидацию по k блокам (повторить кросс-валидацию по k блокам t раз).
- Если кросс-валидация используется для настройки гиперпараметров, модель требуется заново обучить на всём наборе данных.

Другие причины переобучения. Перекрёстная проверка не поможет вам в следующих случаях:

- Нерепрезентативные (смещённые) данные.
- Плохо подобранная метрика качества измерения алгоритма.
- Систематические смещения в методах валидации.
- Непонимание скрытых гиперпараметров.

Короче, если вы продолбались сами или данные отстойные, то вам ничего не поможет.

3.1 Оценка задачи классификации.

Поскольку с категориями нельзя делать ничего, кроме проверки из на равенство, заведём себе вот такую точность:

$$\text{Accuracy}(y; \widehat{y}) = \frac{1}{n} \sum_{i=1}^{n} [y_i = \widehat{y}_i]$$

Где n — количество объектов в выборке, y — вектор реальных ответов, а $\widehat{y}$ — вектор предсказаний. Но тут есть проблема. Представьте, что у вас 90% объектов одного класса. И вы получили точность 70%. Это хорошо? Да ну хз, но, вероятно, нет, и вы переобучаетесь на мажоритарный класс.
Коэффициент Каппа Коэна:

$$\kappa(po, pe) = \frac{po - pe}{1 - pe}$$

Где po — полученная точность, а pe — точность случайного/наивного алгоритма.
Что использовать функцией ошибки? Например, вот это

$$\text{ErrorRate}(y; \widehat{y}) = \frac{1}{n} \sum_{i=1}^{n} [y_i \neq \widehat{y}_i] = 1 - \text{Accuracy}$$

Но это используют редко, чаще используют что-нибудь, что можно производной ковырять.
Из более интересного: Confusion matrix: в ячейке $CM_{t,c}$ хранится количество объектов класса t, которые были приняты нашим алгоритмом за объекты класса c.
Простой пример: бинарная классификация. Тогда у нас наша матрица состоит из четырёх ячеек, которые, как несложно догадаться, называются «True Positive», «True Negative», «False Positive» (мы посчитали, что positive, а оно нет, type 1 error) и «False Negative» (type 2 error). Тут следующие величины:

- Точность (не accuracy, а **precision**): $\dfrac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$.
- Полнота (**recall**): $\dfrac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$.
- Индекс Жаккара (**JaccardIndex**): $\dfrac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$.

Так вот, используют такую штуку, как F-мера:

$$F_\beta = (1+\beta^2)\frac{\mathrm{Precision}\cdot\mathrm{Recall}}{\beta^2\cdot\mathrm{Precision}+\mathrm{Recall}}$$

Например, могут использовать F_1.
Как делать F-меру для нескольких классов? Ну, можно делать «один против всех», мы уже учились. То есть у нс для каждого класса j сформировались свои TP_j, FP_j и FN_j. Тогда

- Можно усреднить TP, FP и FN, из них посчитать Precision и Recall. Полученная мера — micro-average F-score.
- Можно для каждого j посчитать $\mathrm{Precision}_j$ и Recall_j, после чего усреднить уже их. Результат будет другим, и называться будет macro-average F-score.
- А можно уже посчитать F-score для каждого класса и её усреднить. Это будет average F-score.

Лучшей среди них нет. Хороший заказчик сам знает, что ему надо из этого.
Ещё есть вот такая штука:

- **Чувствительность** (true positive rate, TPR) — тупо Recall (т.е. $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$).
- **Специфичность** (true negative rate, TNR) — отношение $\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$.
- ROC-кривая — зависимость TPR от TNR.
- Площадь под ROC-кривой (AUC ROC) — функция качества бинарной классификации.

Рандомный классификатор имеет тупо прямую линию из $(0;0)$ в $(1;1)$. Идеальный — точка $(0;1)$. В многоклассовой классификации используется редко, поэтому говорить об этом не будем.
Как ROC строить? Берём мягкую классификацию. Она для каждого объекта сообщает нам вероятность, что объект принадлежит положительному классу. Так вот что мы делаем? А делаем следующее. Установим карандаш в левый нижний угол, будем перебирать $(p_i; y_i)$ по возрастанию p_i. Если y_i положителен, сдвинем карандаш право, если отрицателен — то вверх. Получим ломаную, которую если аппроксимировать, получится нормальная ROC-кривая.

3.2 Оценка задачи регрессии.

Как сделать себе функцию ошибки? Есть сумма квадратов и его друзья:

- Простая и популярная функция:

$$\mathrm{SS}(y;\widehat{y}) = \|y-\widehat{y}\|_2^2 = \sum_{i=1}^{n}(y_i-\widehat{y}_i)^2$$

  Легко вычисляется, имеет приятную производную, но некорректно сравнивать результаты для массивов разного размера.

- $$\mathrm{MSE}(y;\widehat{y}) = \frac{1}{n}\,\mathrm{SS}(y;\widehat{y})$$

  Жаль, размерность не совпадает с размерностью целевого признака.

- Чтобы это починить, берём корень MSE (получаем RMSE). Но одна проблема всё ещё остаётся: нужен baseline.

- Чтобы по чинить и это, вводим:

$$\mathrm{NRMSE}(y;\widehat{y}) = \frac{\mathrm{RMSE}(y;\widehat{y})}{\sigma[Y]} = \sqrt{\frac{\mathrm{SS(y;\widehat{y})}}{\mathrm{SS}(y;\overline{y})}}$$

  Где $\overline{y}$ — среднее.
  Тут уже много всего хорошего: функция безразмерна, не требует baseline, минимум совпадает с MSE, на нормализованных данных вообще равна MSE. Разве что надо уточнять, какую мы берём нормализацию, в некоторых случаях делят на $\max[Y] - \min[Y]$ вместо $\sigma[Y]$.

- Ещё есть коэффициент детерминации, это уже функция качества, а не ошибки:

$$R^2 = 1 - (\mathrm{NRMSE}(y;\widehat{y}))^2$$

А ещё есть функции ошибки на основе модуля:

- $$\mathrm{MAE}(y;\widehat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i|$$

  Размерность совпадает с размерностью целевого признака.

- Средняя абсолютная процентная ошибка:

$$\mathrm{MAPE}(y;\widehat{y}) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y}_i}{y_i}\right|$$

  Хорошо, что безразмерна, плохо, что придётся доопределять при $y_i = 0$.

- Можно брать симметричную версию:

$$\mathrm{SMAPE}(y;\widehat{y}) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y}_i}{|y_i| + |\widehat{y}_i|}\right|$$

  Тоже безразмерна. Говорят что она симметрична относительно аргументов, но тут как посмотреть: если $y_i = 0$, то у $\widehat{y}_i = 110$ SMAPE будет меньше, чем у $\widehat{y}_i = 90$. Тут тоже иногда надо доопределять, а ещё больше разночтений: иногда 100% заменяют на 200%, и/или $|y_i| + |\widehat{y}_i|$ — на $y_i + \widehat{y}_i$.

Вообще для регрессий говорят чаще о функциях ошибки, а для классификаций — о функциях качестве. Но по сути это, конечно же, не важно.