

Конспект лекций по «Машинному обучению» в университете ИТМО за 2020 год. Лекции читались бакалаврам, а также магистрам, которые пропустили этот курс.

Ключевые места будут выделены жирным шрифтом. Разделы определял сам преподаватель на лекции.

Авторы курса: Андрей Фильченков, Сергей Муравьев. Использовались материалы одноименного курса К.В. Воронцова.

## 1 Введение

### 1.1 Организационные вопросы

Курс ведется сотрудниками Лаборатории машинного обучения. Лаборатория является частью Центра компьютерных технологий.

Если есть желание заниматься машинным обучением (МО) дополнительно, можно написать авторам курса, чтобы получить задачу. Этую задачу можно будет использовать при написании выпускной квалификационной работы.

План курса: обучение с учителем, глубокое обучение, обучение без учителя и обучение с подкреплением.

В конце курса оценка студента формируется из теории (экзамен и теоретический минимум), практики (лабораторные работы) и бонуса. Половина лабораторных работ проверяются автоматически, другую половину нужно защищать. За что можно получить бонусы определяет преподаватель по ходу курса. Посещение лекций не приносит баллов.

Курс разбит на несколько блоков. После каждого будет контрольная. Если все контрольные сданы хорошо, то теоретический минимум закрывается автоматически.

### 1.2 Как устроено МО

Том Митчелл, автор первого учебника по МО, дал следующее определение: программа обучается, если качество решения задачи растет вместе с опытом. Для определения качества используется специальная метрика.

Из данного определения следует, что опытом является количество взаимодействий, которые приводят к получению новых данных.

МО относится к информатике и занимается изучением алгоритмов, способных обучаться.

Некоторые подходы к решению задач в МО: классификация статистическая, на основе сходства, на основе разделимости, нейронные сети, индукция правил, кластеризация, регрессия, алгоритмические композиции, сокращение размерности и т.д. Каждый подход определяет идею работы и обучения алгоритма.

МО широко применяется: подстановка слов, рекомендательные системы, улучшение состава, общение с людьми, уточнение геолокации, улучшение качества изображения и т.д.

Области, в которых используется МО: pattern recognition, computer vision, natural

language processing, big data, information retrieval, data mining и т.д.

Data mining (DM) — это часть процесса извлечения знаний из базы данных. Включает сбор данных, выделение признаков, применение алгоритмов МО. Под DM часто подразумевают data analysis (DA).

DA состоит из следующих компонентов: exploratory DA (изучение данных без алгоритмов), confirmatory DA (подтверждение закономерностей), предсказательный анализ данных (применение МО), визуализация данных.

Data science — сбор данных, data integration (объединение, чистка), хранение данных (организация баз данных), анализ данных, высокопроизводительные вычисления.

**Если МО занимается изучением алгоритмов, то DA занимается решением задач с помощью этих алгоритмов.** В случае с DA изначально данные не типизированы. Это значит, что работу с изображением нельзя назвать анализом данных, но работу с множеством наблюдений за каким-либо явлением — можно.

Далее будут приведены 3 понятия, связанных с интеллектуальностью и знаниями. **Искусственный интеллект** (ИИ) является моделью разумного поведения. Такая модель может быть создана с использованием МО. Выделяют общий (general, сильный) и узкий (narrow, слабый) ИИ. Общий достигает или превосходит человеческий интеллект, узкий направлен на решение конкретных, хорошо формализуемых задач. Алгоритм, который распознает собак, не сможет определить эмоцию у человека.

Один из методов определения ИИ — тест Тьюринга. Выбирается жюри, человек и ИИ. Жюри может взаимодействовать с человеком и ИИ только при помощи общения. Если жюри не удаётся определить где ИИ, а где человек, то тест проходит.

В следствие развития ИИ некоторые задачи, например, связанные с творчеством, переходят из сферы общего ИИ в сферу узкого ИИ.

**Интеллектуальные системы** — это системы, которые обладают ИИ. Выделяют экспертные системы (ЭС) и системы на основе машинного обучения. Отличие заключается в том, что в ЭС знания извлекаются не из данных, а из экспертов, которые этими знаниями обладают. ЭС успешно применяются в сложных задачах, когда данных мало.

**Математическое моделирование** занимается описанием знаний в математической форме. Мат. модели могут быть предсказательными. Если имеется достаточно теории, то можно не обучать систему ИИ, а построить мат. модель. Если модель имеет параметры, которые требуют настройки, то мат. моделирование переходит в МО. Настройка параметров производится в ходе обучения модели.

Знания и данные — разные вещи. Знания могут извлекаться из данных. Данные появляются в результате наблюдений за объектом. Знания — закономерности в некоторой области, получаемые в ходе профессиональной деятельности, которые позволяют формулировать и решать проблемы в этой области.

Знания не обязательно должны извлекаться из данных, ими могут быть известные закономерности, физические явления. В таком случае для описания использу-

зуется мат. моделирование. Знания могут быть получены с помощью экспертов. Во всех случаях они извлекаются для решения проблемы. МО — один из способов извлекать знания, который сильно зависит от данных. Рост популярности МО связан с ростом накопленных данных.

Некоторые дисциплины, на которых основывается МО:

- теория вероятностей и мат. статистика (исследуются распределения данных),
- методы оптимизации (качество работы алгоритма должно быть максимальным),
- вычислительные (численные) методы (математическая задача должна решаться на компьютере),
- линейная алгебра (данные представляются в виде матрицы),
- дискретная математика (данные ограничены, широко используются графы),
- теория сложности алгоритмов (алгоритм должен работать быстро).

Некоторые типы задач МО:

- обучение с учителем (supervised learning),
- обучение без учителя (unsupervised learning),
- частичное обучение (ответы есть только для части данных, semi-supervised learning),
- обучение с подкреплением (поведение агента корректируется при помощи наград и штрафов, reinforcement learning),
- активное обучение (алгоритм выбирает объекты для обучения, active learning),
- динамическое обучение (объекты поступают потоком, online learning),
- структурные предсказания (вместо скалярных значений прогнозируются структуры, structured prediction),
- выбор и оценка модели (важно выбрать наилучший алгоритм, model selection and validation).

Агент — это то, что имеет логику действий и взаимодействует с окружением.

При обучении с учителем имеется множество объектов со своими ответами. Нужно понять, как объекты связаны с ответами. Для решения этого типа задач используют подходы: классификация, регрессия, ранжирование, прогнозирование.

При обучении без учителя есть множество примеров, но нет ответов к ним. Используют подходы: кластеризация, поиск ассоциативных правил, рекомендательные системы, уменьшение размерности.

### 1.3 Обучение с учителем

Наука занимается предсказанием. Любую задачу в МО можно сформулировать как получение предсказания. Хорошая научная теория дает хорошее предсказа-

ние. То же самое можно сказать об алгоритме.

Известных объектов всегда меньше, чем неизвестных.

Далее приводится формальное определение задачи обучения с учителем. **Неизвестная целевая функция** (зависимость):

$$y : X \rightarrow Y,$$

где  $X$  — множество объектов,  $Y$  — множество меток (ответов).

**Размеченный набор данных:**

$$D = \{(x_i, y_i)\},$$

где  $\{x_1, \dots, x_{|D|}\} \subset X$ , так как не для всех объектов имеются ответы,  $y_i = y(x_i)$ .

Нужно найти **решающую функцию**, которая приближает  $y$  на  $X$ :

$$a : X \rightarrow Y.$$

В качестве  $a$  будут рассматриваться алгоритмы. Разница между функцией и алгоритмом заключается в том, что алгоритм всегда вычислим.

Объект представляет из себя множество признаков. Определение  $n$  признаков у объектов:

$$f_j : X \rightarrow D_j, j = 1, \dots, n,$$

где  $j$  — номер признака.

Каждый признак имеет свое множество значений и соответствует одному из следующих типов:

- Бинарный:  $D_j = \{0, 1\}$ .
- Категориальный (номинальный):  $D_j = \{\text{черный, красный, зеленый}\}$ . Множество конечно.
- Порядковый (ординальный):  $D_j = \{\text{мальчик, подросток, юноша, мужчина}\}$ . Является категориальным и упорядоченным.
- Численный (количественный):  $D_j \in \mathbb{R}$ . Например, длина.

Один признак может принимать разные типы в зависимости от поставленной задачи. Признак возраста сам по себе может иметь любой тип.

Признаковое описание объекта  $x$ :

$$(f_1(x), \dots, f_n(x)).$$

Данные можно представить в виде матрицы:

$$F = \|f_j(x_i)\|_{|D| \times n} = \begin{pmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \cdots & \cdots & \cdots \\ f_1(x_{|D|}) & \cdots & f_n(x_{|D|}) \end{pmatrix}.$$

Разложим порядковый признак  $D_j = \{1, 2, 3\}$  на несколько бинарных:  $[f_j(x_i) \geq 1]$ ,  $[f_j(x_i) \geq 2]$ ,  $[f_j(x_i) \geq 3]$ . Такое разложение позволяет сохранить информацию о порядке. Если же раскладываемый признак представить в виде one-hot-вектора, то информация о порядке теряется. Такой вектор будет более разреженным, в нем будет меньше связей между компонентами, а значит, он будет менее информативным.

Распространенной ошибкой является использование идентификаторов в качестве значений категориального признака. Такой подход делает признак вещественным и позволяет вычислять расстояние между его значениями. Значения категориального признака не содержат информации о расстоянии по определению.

**Определение ответов** для классификации:

- $Y = \{-1, +1\}$  — бинарная классификация (родился ли человек в СССР),
- $Y = \{1, \dots, M\}$  —  $M$  непересекающихся классов (в какой стране человек родился?),
- $Y = \{0, 1\}^M$  —  $M$  пересекающихся классов (гражданином каких стран человек является).

Бывают задачи классификации с одним классом. Например, распознавание личной подписи. К первому классу относятся подлинные подписи. Границы второго класса невозможно определить, так как к нему может относиться все, что мы видим.

Ответы для ранжирования:  $Y$  — конечное упорядоченное множество (ранжирование стран по количеству туристов).

Ответы для регрессии:  $Y = \mathbb{R}$  или  $Y = \mathbb{R}^m$  (с какой вероятностью человек посетит страну или страны)

Метка одна, поэтому она может быть многомерной. Признаков несколько, поэтому если признак многомерный, мы просто делаем из него несколько признаков.

Параметрическое множество (**семейство алгоритмов**), из которых выбирается лучший:

$$A = \{M(x, \theta) \mid \theta \in \Theta\},$$

где  $M : X \times \Theta \rightarrow Y$  — некоторая функция,  $\theta$  — параметр этой функции.

Вектор параметров для линейной модели:  $\theta = (\theta_1, \dots, \theta_n)$ ,  $\Theta \in \mathbb{R}^n$ .

Саму линейную модель для задачи регрессии можно определить так:

$$M = F\theta + \epsilon,$$

где  $\epsilon$  — вектор-столбец случайных ошибок.

**Метод обучения** — это отображение из множества датасетов в множество алгоритмов:

$$\mu = \mu^{val} \circ \mu^A : (X \times Y)^{dim} \rightarrow A,$$

где  $(X \times Y)^{dim} = \bigcup_{i=1}^{+\infty} (X \times Y)^i$ ,  
 $D \in (X \times Y)^{dim}$

$\mu$  возвращает лучший  $a \in A$  для заданного  $D$ .

$\mu$  состоит из валидационного метода  $\mu^{val}$  и модели обучения  $\mu^A$  (является композицией этих функций).  $\mu^A$  состоит из предсказательной модели  $M$  и алгоритма обучения, который используется для поиска оптимального  $\theta$ .  $\mu^{val}$  позволяет оценить качество обученных моделей и выбрать лучшую.

Если решаем задачу регрессии, то пусть  $A$  состоит из двух  $M$ : прямой и полинома 9 степени. Если регрессия двумерная (т.е. признак один), то для прямой  $\Theta = \mathbb{R}^2$ , а для полинома  $\Theta = \mathbb{R}^9$ . Обучение для каждой  $M$  обозначается:  $\mu^A(D)$ . Степень полинома в данной задаче является гиперпараметром.

Если применить метод обучения к набору данных, то будет получен алгоритм:

$$a = \mu(D).$$

Полученный алгоритм можно применить к новому объекту, чтобы получить предсказание метки:

$$y = a(x).$$

Если мы говорим про классификацию, то и алгоритм обучения  $\mu$ , и алгоритм предсказания  $a$ , являются классификаторами. Поэтому важно понимать тип входящих данных.

Нужно понять, на сколько хорошо  $a$  приближает  $y$ . **Функция потерь** (loss function) позволяет определить ошибку  $a$  на  $x$ :

- для классификации  $L(a, x) = [a(x) \neq y(x)]$ ,
- для регрессии  $L(a, x) = d(a(x) - y(x))$ ,

Здесь  $[\dots]$  — функция-индикатор.

Расхождение предсказанного значения и истинного можно называть невязкой.  $d$  можно определить как квадратичную функцию потерь (квадрат невязки):

$$d(x) = x^2, \quad L(a, x) = (a(x) - y(x))^2$$

**Эмпирический риск** — оценка качества  $a$  на  $D$ . Простейшей оценкой может быть средняя ошибка:

$$L(a, D) = \frac{1}{|D|} \sum_{x \in D} L(a, x)$$

Риск называется эмпирическим потому что мы его наблюдаем на известных объектах.

Один из методов выбора алгоритма — **метод минимизации эмпирического риска**:

$$\mu_{EmpRM}(D) = \operatorname{argmin}_{a \in A} L(a, D)$$

Аргумент функции эмпирического риска, при котором она достигает минимума на множестве всех алгоритмов.

Уменьшение ошибки на тренировочном множестве может привести к снижению обобщающей способности алгоритма. В таком случае точность на всем множестве будет уменьшаться.

В каждой тренировочной выборке будут свои закономерности, которые не характерны для общей выборки. Не может быть уверенности, что тестовая выборка репрезентативна.

## 1.4 Проблема переобучения

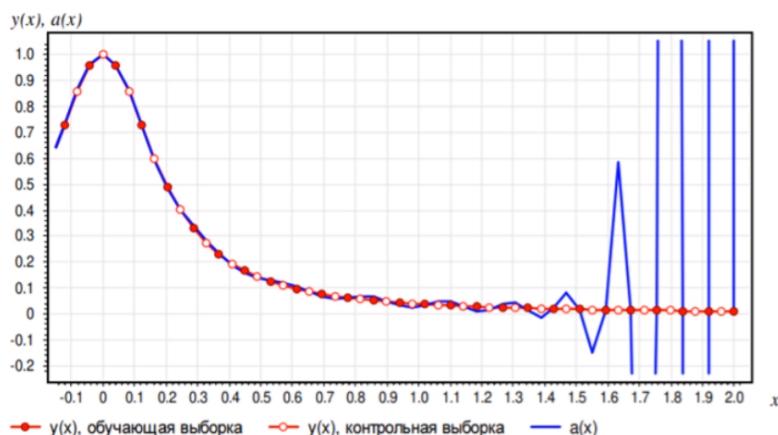
Начиная с определенного уровня сложности предсказательной модели, чем лучше алгоритм показывает себя на тренировочном наборе данных  $D$ , тем хуже он работает на реальных объектах.

В случае регрессии под сложностью понимается степень полинома, который используется в качестве алгоритма.

Точка в которой достигается максимум точности, при которой не началось переобучение называется **оптимумом сложности**.

Пример переобучения в задаче регрессии:

$$y(x) = \frac{1}{1 + 25x^2}; \quad a(x) — \text{многочлен степени } n = 38$$



## 2 Непараметрические методы

### 2.1 Валидация моделей

Человеческий мозг тоже склонен переобучаться, обобщать локальные закономерности и получать ложные сигналы. В качестве примера можно привести веру в сверхъестественное, построение теорий заговоров и т.д.

Аналитические методы борьбы с переобучением:

- регуляризация — добавление ограничений с целью упростить модель,
- выбор параметрического семейства алгоритмов,
- дизайн настройки алгоритмов.

Эмпирические методы:

- внешние меры валидации — оценка обобщающей способности  $\mu^A$  на имеющихся данных,
- внутренние меры валидации.

К внешним мерам относится **валидация на отложенной выборке** (hold-out validation, HO):

$$D = D_{train} \cup D_{test}.$$

Выбирается алгоритм, который обучен на тренировочных данных и дает наименьшую ошибку на тестовых данных:

$$\mu_{HO}^{val}(\mu_A, D_{train}, D_{test}) = \operatorname{argmin}_{\mu^A} L(\mu^A(D_{train}), D_{test}).$$

Проблема метода в том, что после разделения данных в обучающей выборке могут исчезнуть некоторые закономерности. Тогда сам факт обучения на такой выборке уже подразумевает переобучение. Если к переобучению приводит плохое разбиение данных, то говорят, что переобучение наступило на  $\mu^{val}$ .

Зафиксируем размер тренировочной и тестовой выборки:

$$|D_{train}| = r, |D_{test}| = e.$$

Затем разобьем  $D$  всеми возможными способами и определим **метод полной кросс-валидации** (complete cross-validation):

$$\mu_{CCV}^{val}(D, r, e) = \operatorname{argmin}_{\mu^A} \frac{1}{C_{r+e}^e} \sum_{D=D_{train} \cup D_{test}} L(\mu^A(D_{train}), D_{test})$$

Число сочетаний в формуле является количеством всех возможных разбиений  $D$ . Среди всех элементов выбирается первое множество и из оставшихся — второе:

$$D = \underbrace{\{\dots\}}_{r+e} = \underbrace{\{\dots\}}_r \cup \underbrace{\{\dots\}}_e$$

$$C_{r+e}^r \cdot C_e^e = C_{r+e}^e \cdot C_r^r$$

Если из определения  $\mu_{CCV}^{val}$  убрать  $C$ , то аргумент не изменится.

Метод CCV слишком ресурсозатратный. Чтобы это исправить, можно сократить количество разбиений в ущерб обобщающей способности алгоритма. **Метод кросс-валидации по  $k$  блокам** ( $k$ -fold cross-validation) заключается в разбиении  $D$  на  $k$  равных блоков, каждый из которых становится  $D_{test}$ :

$$D = D_1 \cup D_2 \cup \dots \cup D_k, |D_i| \approx |D_j|.$$

Определение метода:

$$\mu_{CV}^{val}(D, k) = \frac{1}{k} \operatorname{argmin}_{\mu^A} \sum_{i=1}^k L(\mu^A(D \setminus D_i), D_i)$$

**$t$  кросс-валидаций по  $k$  блокам** ( $t \times k$ -fold cross-validation) заключается в

проводении кросс-валидации по блокам  $t$  раз. В каждой итерации разбиение на блоки отличается.

Кросс-валидация по отдельным объектам (leave-one-out cross-validation, LOO). В тестовой выборке находится один объект, в тренировочной — все остальные.

$$\mu_{LOO}^{val}(D) = \frac{1}{|D|} \operatorname{argmin}_{\mu^A} \sum_{i=1}^{|D|} L(A(D \setminus p_i), p_i)$$

Когда алгоритм обучается не на всех объектах, теряется часть информации. Здесь не теряется. Но это долго.

**Стратифицированная кросс-валидация по  $k$  блокам** (stratified k-fold cross-validation). Если в выборке объекты какого-то класса или с каким-то значением признака встречаются редко, то при разбиении в каждом блоке эти объекты должны встречаться с той же частотой.

В конце кросс-валидации остаются модели, обученные на своих итерациях. Ни одна из этих моделей не должна в дальнейшем использоваться. Итоговая модель обучается на всех данных.

Смысл валидации в том, чтобы выбрать модель ( $\mu^A$ ), которая меньше всех переобучается на имеющихся данных (или лучше всех обобщает на обучающую выборку). Не в том, чтобы обучить модель без переобучения.

Кроме неподходящих моделей к переобучению также приводят:

- нерепрезентативный набор данных (набор не отражает все закономерности генеральной совокупности)
- метрика качества алгоритмов плохо подобрана
- имеются систематические смещения в методах валидации и обучения (модель выбиралась на одинх данных, а использовалась на других)
- непонимание скрытых гиперпараметров.

Бэнчмарк — хорошо размеченный датасет для тестирования алгоритма.

## 2.2 Классификация и регрессия на основе похожести

**Утиный тест** (duck test): если нечто выглядит как утка, плавает как утка и крякает как утка, то это, вероятно, и есть утка.

Этот тест можно рассматривать в качестве классификатора. Обучающие данные для него: много уток, много не уток (класс "неутка"). Чтобы построить классификатор: для уток выделить ключевые признаки (плавает как утка, крякает как утка, выглядит как утка), понятие похожести использовать для оценки близости признаков (определить насколько признак одного объекта похож на тот же признак другого объекта), логический сепаратор использовать для классификации (логическое И, если все три признака похожи, то это один класс).

Выводы из этого теста: похожие объекты принадлежат одному классу или имеют похожие значения целевой функции.

Этот вывод связан с рассуждением по аналогии, т.е выбор действий исходя из опыта в похожих ситуациях. Также связан с ленивым обучением: мы не обучаем модель, поиск похожих объектов выполняется каждый раз для новых объектов

### 2.3 Метод (одного) ближайшего соседа

Похожесть — это мера сходства между объектами. В данном случае это будет расстояние (метрика)

Метрическое пространство это множество с заданной на нем метрикой

$$p(x, y) : X \times X \rightarrow [0; +\infty)$$

симметрия неравенство треугольна различимость нетождественных объектов неразличимость тождественных объектов

Расстояние Минковского:

$$p(x, y) = \left( \sum_i |x_i - y_i|^p \right)^{\frac{1}{p}}$$

, при  $p = 2$  Евклидово, при  $p = 1$  Манхэттенское

Косинусное расстояние (косинус угла между объектами):

$$p(x, y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2} \sqrt{\sum_i y_i^2}}$$

Расстояние Махаланобиса:

$$p(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)},$$

где  $S$  - матрица ковариации между  $x$  и  $y$ .

Ближайший сосед объекта  $u$  имеет минимальную метрику до него:

$$x_{(u,1)} = \underset{x \in X_{train}}{\operatorname{argmin}} p(u, x)$$

Классификатор (одного) ближайшего соседа:

$$a_{1NN}(u, D_{train}) = y(x_{(u,1)})$$

Диаграмма Вороного визуализирует, как классификатор разбивает пространство объектов на классы.

Достоинства 1NN:

- простота реализации
- понятный
- интерпретируемость (можно легко объяснить решение классификатора)

Недостатки

- чувствительность к шуму (ошибочный класс одного объекта будет приводить к ошибкам классификации новых объектов)
- низкое качество работы
- нет обучаемых параметров (явно заданных)
- необходимость хранить все объекты

В случае с регрессией, мы строим зависимость по ближайшим точкам (определяем ближайшего соседа и возвращаем значение целевой функции на нем).

## 2.4 Метод k ближайших соседей (kNN)

Отсортируем объекты:

$$p(u, x_{(u,1)}) \leq p(u, x_{(u,2)}) \leq \dots \leq p(u, x_{u,|D_{train}|}).$$

Алгоритм (возвращаем тот класс, которому принадлежит большинство соседей):

$$a_{kNN}(u; D_{train}) = \operatorname{argmax}_{y \in Y} \sum_{i=1}^k [y(x_{(u,i)}) = y]$$

Лучше записать так, чтобы был понятней ОМК:

$$a_{kNN}(u; D_{train}) = \operatorname{argmax}_{y \in Y} \sum_{i=1}^{|D_{train}|} [y(x_{(u,i)}) = y][i \leq k]$$

Как можно улучшить 1NN:

- Более сложна модель (больше параметров) (добавить число соседей)
- Выбор расстояния, уменьшение размерности
- Использование эффективных структур для хранения данных (чтобы уменьшить потребление памяти)
- Прореживание объектов (брать только часть объектов, а не все), фильтрация шума, выбор прототипов (хранить только наиболее стереотипные объекты класса, можно один объект на каждый класс)

Если мы не знаем, как определить k и хотим извлекать информацию о классе нового объекта из каждого объекта. Обобщенный метрический классификатор, оценка близости объекта u к классу y:

$$a_{GenDistClassifier}(u, D_{train}) = \operatorname{argmax}_{y \in Y} \sum_{i=1}^{|D_{train}|} [y(x_{(u,i)}) = y] w_{(i,u)},$$

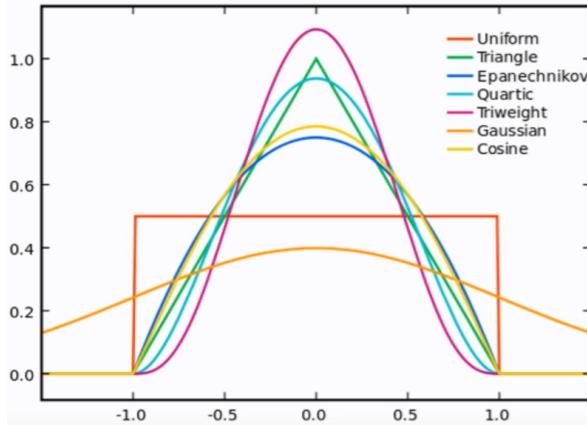
где  $w_{i,u}$  функция значимости i-го соседа u.

У нас нет k соседей, но есть веса.

$w(i,u)$ :

- линейно убывающая функция от расстояния,
- экспоненциально убывающая,
- ядерная функция.

Ядерная функция  $K(x)$  это симметричная неотрицательная функция  $\int_{-\infty}^{+\infty} K(x)dx = 1$ . Эта функция от расстояния. Отрицательной быть не может, так как веса положительны



Ядра бывают финитные (конечный интервал, где функция больше 0) и нефинитные. У финитных после единицы по модулю 0. Для финитных ядер расстояния нормируются (это будет видно дальше).

Окно Парзена-Розенблатта. Аргументы ядра нормализуются:

С фиксированной шириной окна

$$a_{GenDistClassh}(u, D_{train}, h, K) = \operatorname{argmax}_{y \in Y} \sum_{i=1}^{|D_{train}|} [y(x_{(u,i)}) = y] K\left(\frac{\rho(u, x_{(u,i)})}{h}\right)$$

где  $h$  — ширина окна. Если объект находится за окном, то аргумент ядра будет больше 1, а значит при использовании финитного ядра этот объект можно не учитывать.

Не фиксированной:

$$a_{GenDistClassk}(u, D_{train}, k, K) = \operatorname{argmax}_{y \in Y} \sum_{i=1}^{|D_{train}|} [y(x_{u,i}) = y] K\left(\frac{\rho(u, x_{(u,i)})}{\rho(u, x_{(u,k+1)})}\right)$$

Здесь вместо ширины окна используется расстояние до  $k$  соседа. Если объект находится дальше  $k$  соседа, то его можно не учитывать.

Расстояние можно обучать. Взвешенное расстояние Минковского позволяет выбрать наиболее значимые признаки. Обучение заключается в выборе оптимальных значений  $w_i$

$$(x, y) = \left( \sum_i w_i |x_i - y_i|^p \right)^{\frac{1}{p}}.$$

Выбирать ядро удобней, чем расстояние.

Использование структур для хранения данных повышает скорость поиска соседей. Чаще всего используются k-d-деревья. Существуют структуры, которые в ущерб точности увеличивают скорость поиска. k-d-дерево — бинарное дерево поиска, но для элементов в k мерном пространстве.

## 2.5 Непараметрическая регрессия

Так как предположений о том, как выглядит функция нет, Будем аппроксимировать функцию константой взвешанном виде. Наша регрессия  $(x) = \theta$ , где  $x \in X$ . Выбираем на регрессии  $x$ , ищем для него  $\theta$ , :

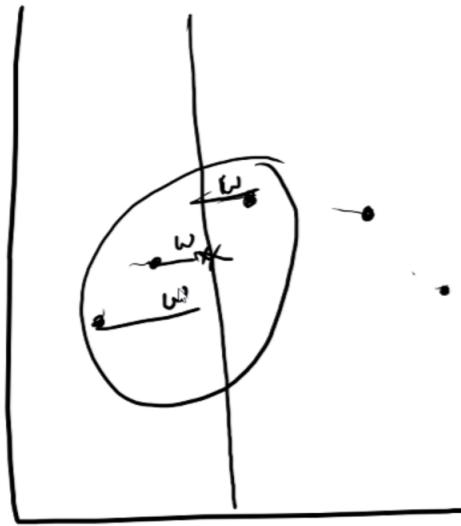
$$L(\theta, D_{train}) = \sum_{i=1}^{|D_{train}|} w_i(x)(\theta - y_i)^2 \rightarrow \min_{\theta \in \mathbb{R}}$$

w позволяет уменьшить значимость дальних объектов. Будем использовать ядерное сглаживание.

$$w_i(x) = K\left(\frac{\rho(x_i, x)}{h}\right)$$

Используем фиксированное окно, так как точки расположены очень плотно. Лучше усреднять по фиксированному интервалу.

Иллюстрация нахождения одной точки на регрессии:



Ядерное сглаживание Надара-Ватсона:

$$a_{NonParamRegh}(x, D_{train}) = \frac{\sum_{x_i \in D_{train}} y_i w_i(x)}{\sum_{x_i \in D_{train}} w_i(x)} = \frac{\sum_{x_i \in D_{train}} y_i K\left(\frac{\rho(x_i, x)}{h}\right)}{\sum_{x_i \in D_{train}} K\left(\frac{\rho(x_i, x)}{h}\right)}$$

В первой части формулы определяется средневзвешанное значение функции в заданной точке. Во второй — вес выражается через ядро.

При определенных условиях непараметрическая регрессия Надарай-Ватсона вероятно сходится к целевой функции.

Анализ методов:

- Выбор функции ядра влияет на гладкость функции потерь и не сильно значим для итогового качества
- Выбор  $h$  и  $k$  влияет на качество приближения, их можно настраивать
- Чувствительность к шуму сохраняется (при построении регрессии выборы сильно изменят ее при усреднении)

## 2.6 Оценка качества

Рассмотрим задачу бинарной классификации. Пусть один класс положительный (утрки), а второй — отрицательный (неутки).

Таблица решений классификатора:

	Положительный	Отрицательный
Отнесен к положительным	TP (True Positive)	FP (False Positive)
Отнесен к отрицательным	FN (False Negative)	TN (True Negative)

$FN$  в статистике — ошибка первого рода,  $FP$  в статистике — ошибка второго рода  $P = TP + FN$  — число Положительных примеров  $N = FP + TN$  — число отрицательных примеров

Некоторые оценки. Полнота (recall) или чувствительность (sensitivity) (доля положительных объектов, которую алгоритм находит) (True Positive Rate):

$$Recall = TPR = \frac{TP}{P}$$

Специфичность (specificity):

$$SPC = \frac{TN}{N}$$

Точность (precision) (Доля действительно положительных объектов среди тех, кого отнесли к положительным):

$$Precision = PPV = \frac{TP}{TP + FP}$$

Часто приходится балансировать между PPV и TPR

Точность (accuracy) (определяет на сколько хорошо работает классификатор):

$$Accuracy = ACC = \frac{TP + TN}{P + N}$$

Такая точность плохо работает для несбалансированных классов. Если один класс в несколько раз больше другого, то классификатор может относить все объекты к этому классу и иметь хорошую точность. Эта проблема решается с помощью

F-меры:

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

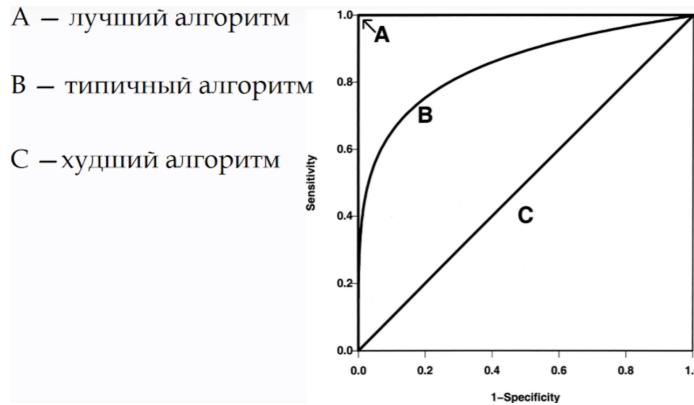
где  $\beta$  определяет важнее полнота или точность

Часто используется  $F_1$ -мера, которая является средней гармонической между полнотой и точностью:

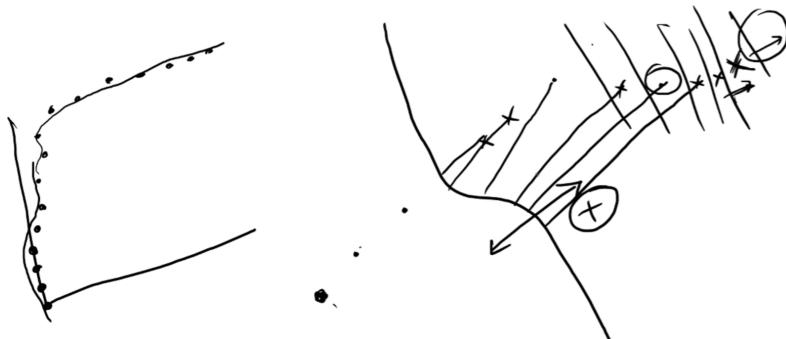
$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Средняя гармоническая:  $H(x_1, \dots, x_n) = \frac{1}{\frac{1}{n} \sum_{i=1}^n \frac{1}{x_i}}$

ROC-кривая (Receiver Operating Characteristic). Баланс между точностью и полнотой можно легко показать на примере радара. Если радар имеет высокую чувствительность, то он реагирует на все самолеты, но еще и на птиц. Если сделать радар очень точным, то он перестанет обнаруживать птиц, но может пропускать вражеские самолеты.



Здесь слева построение кривой. Справа приведена граница принятия решения. Объекты упорядочим: справа скорее относятся к положительному классу (крестик), чем объекты слева. Классификатор решает, что объект относится к положительному классу, если он справа от границы. Будем перемещать границу справа налево и строить ROC-кривую.



Площадь под кривой (Area under the curve, AUC). У худшего алгоритма площадь

0.5, лучшего — 1.

Что делать, когда классов больше 2:

- One vs one классификация — строим классификатор на каждую пару классов и оцениваем их по отдельности.
- One vs all (one vs rest) — строим классификатор на каждый класс, а в качестве второго брать инверсию.
- Иерархическая классификация. Решаем к какому из 2 классов относится объект, затем к какому из 2 классов внутри этого класса и т.д.
- Матрица ошибок для многих классов. По столбцам фактический класс объекта, по строкам, куда его занесли. Можно определить, какие пары классов плохо различаются.

Ошибки для регрессии: Среднеквадратичная ошибка (Mean squared error, MSE):

$$MSE = \sum_{(x_i, y_i) \in D} \frac{(a(x_i) - y_i)^2}{|D|}.$$

Среднеквадратичная ошибка (Root mean squared error, RMSE):

$$RMSE = \sqrt{\sum_{(x_i, y_i) \in D} \frac{(a(x_i) - y_i)^2}{|D|}}.$$

Абсолютная средняя ошибка (Mean absolute error, MAE):

$$MAE = \sum_{(x_i, y_i) \in D} \frac{|a(x_i) - y_i|}{|D|}.$$

Симметричная средняя абсолютная ошибка в процентах (Symmetric mean absolute percentage error, SMAPE): Чем мы дальше от нуля, тем менее важна ошибка. Разница между 2млн100 и 2млн не такая значительная, как разница между 200 и 300.

$$SMAPE = \frac{1}{|D|} \sum_{(x_i, y_i) \in D} \frac{2 \cdot |a(x_i) - y_i|}{|a(x_i)| + |y_i|}.$$

### 3 Линейные модели

#### 3.1 Линейная классификация

Формулировка задачи:

- $Y = \{-1, +1\}$ ,
- $D = \{(x_i, y_i)\}_{i=1}^{|D|}$ ,
- требуется найти  $a_w(x, D)$  в виде  $\text{sign}(f(x, w))$ ,  
где  $f(x, w)$  — функция распознавания,  $w$  — вектор параметров.

Ключевая гипотеза: объекты в пространстве хорошо разделимы.

Основная идея: поиск среди разделяющих поверхностей, описываемых уравнением  $f(x, w) = 0$ . Требуется построить разделяющую гиперплоскость.

**Функция отступа (margin)** для объекта  $x_i$ :

$$M_i(w) = y_i f(x_i, w),$$

$M_i(x) < 0$  — свидетельство того, что объект классифицирован некорректно.

Эмпирический риск отражает количество объектов, на которых классификатор допускает ошибку:

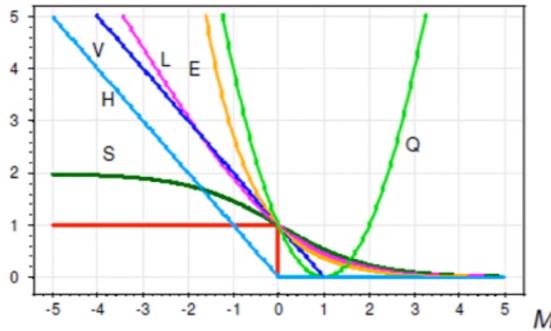
$$L(a_w, D) = L(w) = \sum_i^{|D|} [M_i(w) < 0]$$

Такая функция не является гладкой, а значит поиск экстремумов невозможен. Чтобы эту проблему решить, аппроксимируем ее:

$$\tilde{L}(w) = \sum_i^{|D|} L(M_i(w)),$$

где  $L(M_i(w)) = L(a_w(x_i, D), x_i)$  — функция потерь.

Желательно, чтобы  $L$  была неотрицательной, невозрастающей и гладкой. Примеры функций (функционалов):



$H(M) = (-M)_+$	— кусочно-линейная (правило Хебба)
$V(M) = (1 - M)_+$	— кусочно-линейная (SVM)
$L(M) = \log_2(1 + e^{-M})$	— логарифмическая
$Q(M) = (1 - M)^2$	— квадратичная
$S(M) = 2(1 + e^M)^{-1}$	— сигмоидная
$E(M) = e^{-M}$	— экспоненциальная

Если функция возрастает, то ее можно использовать, чтобы штрафовать модель и препятствовать переобучению.

Линейный классификатор:

$$a_w(x, D) = \text{sign} \left( \sum_{n=1}^n w_i f_i(x) - w_0 \right),$$

где  $f_j : X \rightarrow \mathbb{R}, j = 1, \dots, n$  — численные признаки,  
 $w_1, \dots, w_n \in \mathbb{R}$  — веса признаков.

То же самое можно записать в виде скалярного произведения, если добавить

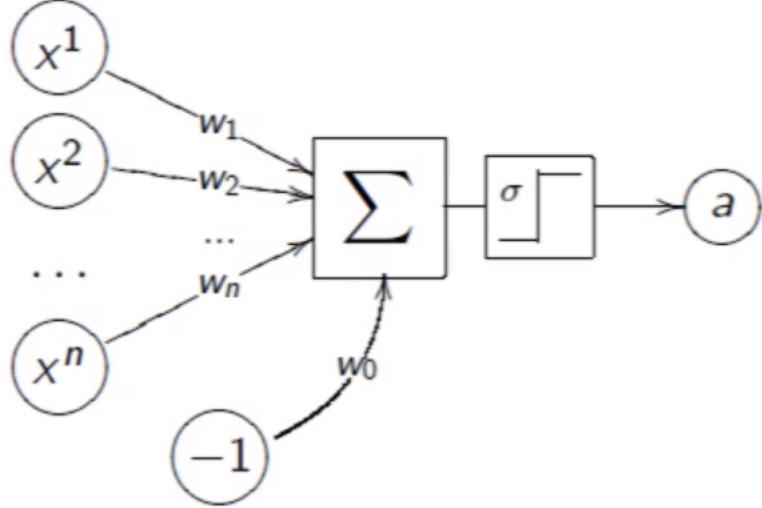
$f_0(x) = -1$  (чтобы включить  $w_0$ ):

$$a_w(x, D) = \text{sign}(\langle w, x \rangle).$$

Рассмотрим нейрон МакКаллока-Питтса:

$$a_w(x, D) = \sigma(\sum_{i=1}^n w_i f_i(x) - w_0),$$

где  $\sigma$  — функция активации.



Семейство алгоритмов, из которого требуется выбрать один:

$$A_{linear} = \{a_w(x) = \text{sign}(\langle w, x \rangle) \mid w \in \mathbb{R}^n\}$$

Таким образом, чтобы обучить классификатор, нужно найти  $w$ . Можно использовать почти любой алгоритм оптимизации, способный оптимизировать эмпирический риск в соответствующем пространстве.

### 3.2 Градиентный спуск

Задача оптимизации:

- $X \subset \mathbb{R}^n$  — допустимое множество (могут быть наложены условия),
- $f : X \rightarrow \mathbb{R}$  — целевая функция,
- минимум или максимум — критерий поиска.

Классификация методов оптимизации (методы поиска): детерминированные, случайные, комбинированные.

Классификация методов по критерию размерности  $X$ : многомерные, одномерные.

В задачах дискретного программирования  $X$  конечного или счетно.

В задачах целочисленного программирования  $X \subset \mathbb{Z}$ .

В задачах нелинейного программирования  $f$  — нелинейная функция,  $X \subset K^n$ ,  $|K^n| < \infty$ .

В задачах линейного программирования  $f$  — линейная функция.

Прямые методы — вычисление целевой функции в точках приближений (метод перебора, метод золотого сечения).

Методы первого порядка — вычисление первых частных производных (градиентный спуск, метод Коши, так же известный как наискорейший градиентный спуск).

Методы второго порядка — вычисление вторых частных производных, то есть гессиана целевой функции (метод Ньютона)

Задача минимизации эмпирического риска:

$$\tilde{L}(w) = \sum_i^{|D|} L(M_i(w)) = \sum_i^{|D|} L(\langle w, x_i \rangle y_i) \rightarrow \min_w .$$

Классический градиентный спуск:

$$w_{(k+1)} = w_{(k)} - \mu \nabla L(w_{(k)}) = w_{(k)} - \mu \sum_i^{|D|} L'(\langle w, x_i \rangle y_i) x_i y_i,$$

где  $w_{(0)}$  — некоторое начальное значение,

$\mu$  — шаг градиента или скорость сходимости,

имеем производную композиции, поэтому производная аргумента:  $w x_i y_i = x_i y_i$ .

Смотри простую реализацию градиентного спуска, чтобы понимать, что здесь написано: <https://github.com/vpunch/optimization-methods/blob/master/code/graddesc.m>

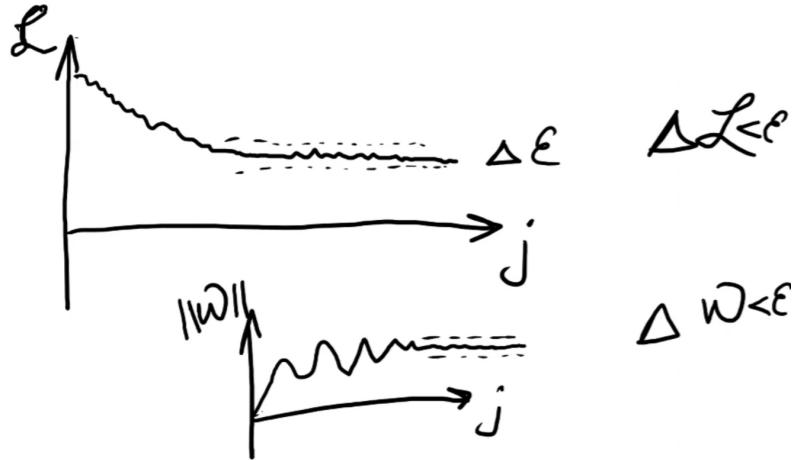
Проблема такого спуска в том, что приходится выполнять слишком много вычислений. Вместо того, чтобы считать производную по всем объектам, будем считать только по одному. Причем номер объекта зависит от номера итерации. Такой подход называется **стохастическим градиентным спуском** (СГС):

$$w_{(k+1)} = w_{(k)} - \mu L'(\langle w_{(k)}, x_{(k)} \rangle y_{(k)}),$$

где  $w_{(0)}$  — некоторое начальное значение,

$x_{(1)}, \dots, x_{(|D|)}$  — некоторый порядок объектов.

Критерий останова: когда значения  $L$  или  $w$  почти не меняются:



Также можно останавливать спуск по количеству итераций.

Чтобы значение функции не скакало, считаем экспоненциальное скользящее среднее эмпирического риска:

$$L_{(k+1)} = (1 - \alpha)L_{(k)} + \alpha L(\langle w_{(k)}, x_{(k)} \rangle y_{(k)}),$$

$\alpha$  обычно выбирают как 1, деленная на количество прошедших итераций.

Проблема СГС в том, что он слишком случайный, так как на каждой итерации зависит только от одного объекта. Будем использовать небольшое подмножество объектов и менять его элементы на каждой итерации. **Пакетный градиентный спуск**:

$$w_{(K+1)} = w_{(K)} - \mu \sum_{k=Kb}^{k=K+1} b L'(\langle w_{(K)}, x_{(k)} \rangle y_{(k)}),$$

$$L_{(K+1)} = (1 - \alpha)L_{(K)} + \alpha \sum_{k=Kb}^{k=(K+1)b} L(\langle w_{(K)}, x_{(k)} \rangle y_{(k)}),$$

где  $b$  — размер пакета

Критерий останова тот же.

Визуальное сравнение работы спусков (стрелочками показаны итерации):



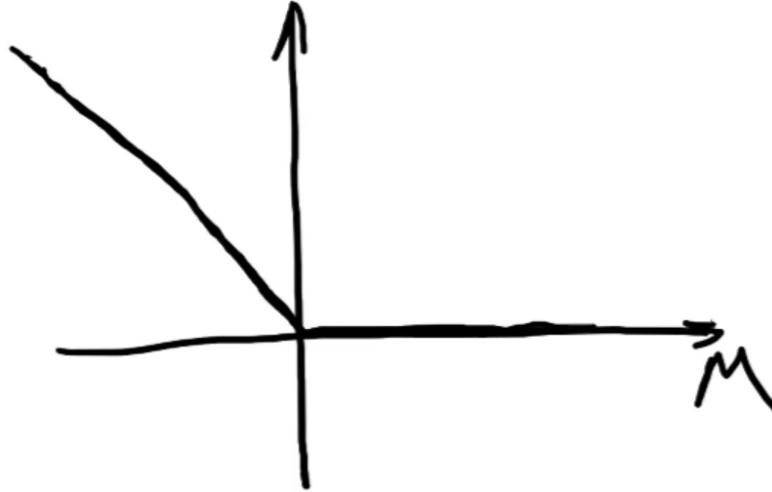
Во времена, когда градиентный спуск еще не придумали, использовали следующие алгоритмы.

**Правило Хебба** используется для настройки весов для классификации на множестве  $\{1, -1\}$ : если  $\langle w_{(k)} x_{(k)} \rangle y_{(k)} < 0$ , то

$$w_{(k+1)} := w_{(k)} + \eta x_{(k)} y_{(k)}$$

То есть, когда находится объект, на котором текущий вектор весов приводит к ошибке, мы изменяем вектор весов.

Функция ошибки выглядит так:



**Правило Розенблатта** следует из правила Хебба. Для классификации на множестве  $\{1; 0\}$ , будем для каждого  $x_{(k)}$  менять вектор весов:

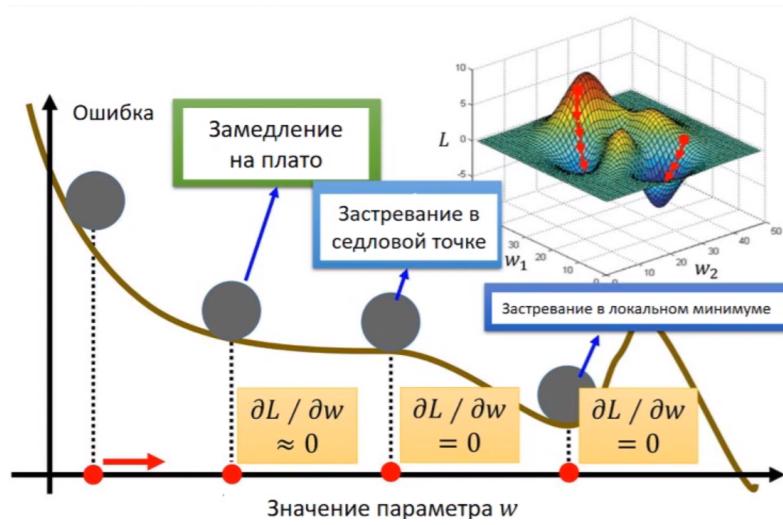
$$w_{(k+1)} := w_{(k)} - \eta(a_w(x_{(k)}) - y_{(k)}).$$

**Теорема Алекса Новикова** утверждает, что стохастический градиентный спуск приводит к оптимуму. А именно, если выборка  $D$  линейно разделима, то есть  $\exists w, \exists \delta > 0 : \langle w, x_i \rangle y_i > \delta$  для всех  $i = 1, \dots, |D|$ , тогда стохастический градиентный спуск с правилом Хебба находит вектор весов  $w$ , который разделяет выборку безошибочно:

- при любом начальном значении  $w_0$ ,
- при любом  $\mu > 0$ ,
- независимо от порядка объектов  $x_{(i)}$ ,
- за конечное число изменений вектора  $w$ .

Если  $w_{(0)} = 0$ , тогда количество изменений вектора  $w$ :  $t_{max} \leq \frac{1}{\delta^2} \max \|x_j\|$ .

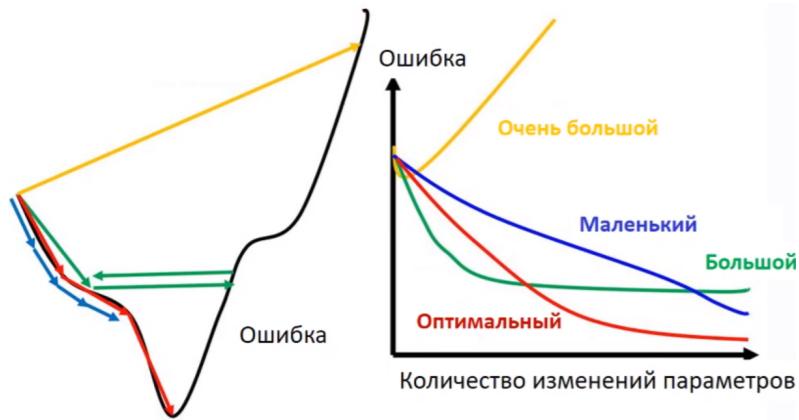
Проблемы сходимости спуска связаны с тем, что функция может содержать несколько локальных минимумов. Они могут привести к ранней остановке алгоритма:



Для обхода этих проблем применяют такие эвристики:

- $w_j = 0$  для всех  $j = 0, \dots, n$  (плохо для нейронных сетей),
- маленькие случайные значения  $w_j \in [-\frac{1}{2n}, \frac{1}{2n}]$ ,
- $w_j = \frac{\langle y, f_j \rangle}{\langle f_j, f_j \rangle}$  ( $f_j$  — признак, соответствующий компоненте вектора весов),
- обучение по небольшой случайной подвыборке объектов,
- многократные запуски из разных начальных приближений и выбор лучшего решения.

Проблема скорости сходимости:



Антиградиент задает направление перемещения аргумента. При малом масштабном коэффициенте антиградиента, алгоритм слишком медленный, при слишком большом — алгоритм может разойтись.

Способы решения проблемы сходимости:

- использовать **наискорейший спуск** (минимизировать функцию на каждой

итерации по шагу):

$$L(w_{(k)} - \mu_{(k)} \nabla L(w_{(k)})) \rightarrow \min_{\mu_{(k)}}$$

- увеличивать шаг на некоторых итерациях, чтобы перепрыгнуть локальный минимум;
- использовать методы второго порядка (если производная почти равна нулю);
- использовать средний вектор недавних шагов.

Эвристики на порядок предъявления объектов:

- на каждом шаге брать предметы из разных классов,
- чаще брать спорные объекты, у которых маленькие  $M_i$ ,
- реже брать хорошие объекты, у которых  $M_i > k_+$ ,
- реже брать объекты-шумы, у которых  $M_i < K_-$ .

Преимущества алгоритма градиентного спуска:

- простота реализации,
- легко обобщается для любых  $f$  (признаков) и  $L$ ,
- возможность динамического обучения,
- поддерживает сверхмалые выборки.

Недостатки:

- медленная сходимость, возможна расходимость,
- застревание в локальных минимумах и седловых точках,
- очень важен правильный подбор эвристик,
- переобучение.

### 3.3 Линейная регрессия и матричное разложение

Модель многомерной линейной регрессии:

$$f(x, \theta) = \sum_{j=1}^n \theta_j f_j(x), \quad \theta \in R^n$$

Матричные обозначения:

$$f = \begin{pmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \vdots & \cdots & \vdots \\ f_1(x_{|D|}) & \cdots & f_n(x_{|D|}) \end{pmatrix}, y = \begin{pmatrix} y_1 \\ \vdots \\ y_{|D|} \end{pmatrix}, \theta = (\theta_1, \dots, \theta_n)$$

Эмпирический риск в матричной записи:

$$L(\theta, D) = \sum_{i=1}^{|D|} (f(x_i, \theta) - y_i)^2 = \|F\theta - y\|^2 = (F\theta - y)^T(F\theta - y) \rightarrow \min_{\theta}$$

$\|F\theta - y\|^2$  — обозначение второй нормы в квадрате.

Для решения такой задачи существует множество способов. Один из наиболее популярных — матричное разложение (матричная факторизация).

Условие минимума:

$$\begin{aligned} \frac{\partial L(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} [(F\theta - y)^T(F\theta - y)] = \frac{\partial}{\partial \theta} [((F\theta)^T - y^T)(F\theta - y)] = \\ &= \frac{\partial}{\partial \theta} [\theta^T F^T - y^T](F\theta - y) = \frac{\partial}{\partial \theta} [\theta^T F^T(F\theta - y) - y^T(F\theta - y)] = \\ &= \frac{\partial}{\partial \theta} [\theta^T F^T F\theta - \cancel{\theta^T F^T y} - y^T F\theta + y^T y] = \frac{\partial}{\partial \theta} [\theta^T F^T F\theta - 2\theta^T F^T y + y^T y] = \\ &= 2F^T F\theta - 2F^T y = 2F^T(F\theta - y) = 0 \end{aligned}$$

Здесь  $(F\theta)^T y = y^T F\theta$ , но можно это не замечать и посчитать производные этих слагаемых по отдельности. Они будут одинаковыми.

Из условия выразим вектор параметров:

$$2F^T(F\theta - y) = F^T F\theta - F^T y = 0 \implies \theta^* = (F^T F)^{-1} F^T y$$

Если что-то не понятно, то стоит заглянуть сюда:

- дифференцирование матрицы: [http://nabatchikov.com/blog/view/matrix\\_der](http://nabatchikov.com/blog/view/matrix_der),
- заметки по матричным вычислениям: <http://www.machinelearning.ru/wiki/images/2/2a/Matrix-Gauss.pdf>
- матричные выражения: [http://mathprofi.ru/svoistva\\_operacij\\_nad\\_matricami-matrichnye\\_vyrazheniya.html](http://mathprofi.ru/svoistva_operacij_nad_matricami-matrichnye_vyrazheniya.html)

Псевдообратная матрица (обратное преобразование Мура-Пенроуза):

$$F^+ = (F^T F)^{-1} F^T$$

Проекционная матрица:

$$P_F = FF^+$$

Решение:

$$\theta^* = F^+ y.$$

Минимальное приближение:

$$L(\theta^*) = \|P_F y - y\|^2$$

**Теорема** Любая матрица  $F$  размера  $|D| \times n$  может быть представлена в виде сингулярного разложения:

$$F = VDU^T,$$

где

- $V$  — матрица  $|D| \times n$ , где столбцы  $v_j$  — собственные векторы матрицы  $FF^T$ :  $V = (v_1, \dots, v_n)$ , является ортогональной ( $V^T V = I_n$ );
- $U$  — матрица  $n \times n$ , где строки  $u_j$  — собственные векторы  $F^T F$ , является ортогональной ( $U^T U = I_n$ );
- $D = \text{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_n})$  размера  $n \times n$ ,  $\sqrt{\lambda_j}$  — **сингулярные числа** (т.е. квадраты собственных значений  $F^T F$ ).

Про собственные векторы и значения можно почитать здесь: [http://mathprofi.ru/sobstvennye\\_znachenija\\_i\\_sobstvennye\\_vektory.html](http://mathprofi.ru/sobstvennye_znachenija_i_sobstvennye_vektory.html)

Про ранг есть здесь: [http://mathprofi.ru/rang\\_matrix.html](http://mathprofi.ru/rang_matrix.html)

Представим какое-то скрытое пространство, в которое мы хотим проецировать данные (матрицу  $F$ ). Это пространство можно описать базисными векторами.  $D$  представляет важность каждого базисного вектора,  $V$  представляет, как объекты соответствуют базисным векторам,  $U$  показывает, как признаки соответствуют базисным векторам.

**Метод наименьших квадратов при помощи сингулярного разложения** (работает быстрей, чем классический):

$$\begin{aligned} F^+ &= (UDV^T VDU^T)^{-1} UDV^T = UD^{-1}V^T = \sum_{j=1}^n \frac{1}{\sqrt{\lambda_j}} u_j v_j^T; \\ \theta^* &= F^+ y = UD^{-1}V^T y = \sum_{j=1}^n \frac{1}{\sqrt{\lambda_j}} u_j (v_j^T y); \\ F\theta^* &= P_F y = (VDU^T)UD^{-1}V^T y = VV^T y = \sum_{j=1}^n v_j (v_j^T y); \\ ||\theta^*||^2 &= ||D^{-1}V^T y||^2 = \sum_{j=1}^n \frac{1}{\lambda_j} (v_j^T y)^2. \end{aligned}$$

Анализ метода:

- если можем вычислить сингулярное разложение, то можем легко решить задачу МНК,
- сингулярное разложение вычисляется за  $O(|D|^2 n + n^3)$ ,
- сингулярное разложение — важный инструмент во многих других задачах машинного обучения, в первую очередь, в снижении размерности.

### 3.4 Регуляризация

Переобучение могут вызывать скачки весов  $w$ . Чтобы это предотвратить, можно ограничить норму  $w$ .

Добавим штраф регуляризации для нормы весов:

$$L_\tau(a_w, D) = L(a_w, D) + \frac{\tau}{2} \|w\|^2 \rightarrow \min_w,$$

где  $\tau$  — коэффициент регуляризации, который отражает баланс между качеством и обобщаемостью.

**Гребневая регуляризация** основывается на предположении, что значения вектора  $\theta$  имеют распределение Гаусса с ковариационной матрицей  $\sigma I_n$ :

$$L_\tau(\theta) = \|F\theta - y\|^2 + \frac{1}{2\sigma} \|\theta\|^2 \rightarrow \min_\theta,$$

где  $\tau = \frac{1}{\sigma}$  — коэффициент регуляризации.

Внедрим такую регуляризацию в решение задачи МНК:

$$\begin{aligned} \theta_\tau^* &= (F^T F + \tau I_n)^{-1} F^T y = U(D^2 + \tau I_n)^{-1} D V^T y = \sum_{j=1}^n \frac{\sqrt{\lambda_j}}{\lambda_j + \tau} u_j (v_j^T y); \\ F\theta_\tau^* &= (V D U^T) \theta_\tau^* = V \text{diag} \left( \frac{\lambda_j}{\lambda_j + \tau} \right) V^T y = \sum_{j=1}^n \frac{\lambda_j}{\lambda_j + \tau} v_j (v_j^T y); \\ \|\theta^*\|^2 &= \|D^2 (D^2 + \tau I_n)^{-1} D^{-1} V^T y\|^2 = \sum_{j=1}^n \frac{1}{\lambda_j + \tau} (v_j^T y)^2. \end{aligned}$$

**Лассо Тибширани (least absolute shrinkage and selection operator, LASSO)** основывается на предположении, что значения вектора  $\theta$  имеют распределение Лапласа:

$$\begin{cases} L_\tau(\theta) = \|F\theta - y\|^2 \rightarrow \min_\theta; \\ \sum_{i=1}^n |\theta_i| \leq k. \end{cases}$$

Алгоритм используется в качестве инструмента выбора признаков, отсюда и название.

Результирующая задача оптимизации:

$$L_\tau(\theta) = \|F\theta - y\|^2 + \tau \|\theta\|_1 \rightarrow \min_\theta$$

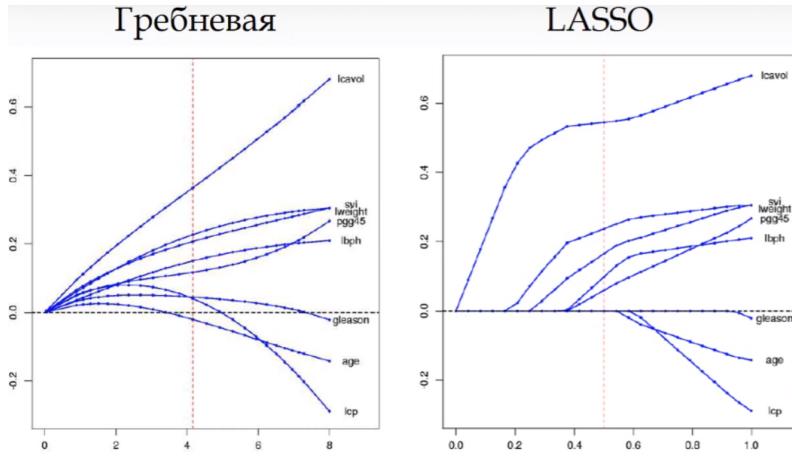
Хорошего аналитического решения не существует, но существует вычислительное решение.

Как применить регуляризацию для градиентного спуска:

$$\nabla L_\tau(w) = \nabla L(w) + \tau w,$$

$$w_{k+1} = w_k (1 - \mu \tau) - \mu \nabla L(w).$$

Сравнение методов регуляризации:



На оси абсцисс — значение параметра регуляризации  $\tau$ , ординат — значение признака. Видно, что в первом случае с уменьшением  $\tau$  уменьшаются все признаки, а во втором — признаки последовательно отсекаются.

Рассмотрим, что такое регуляризация в целом. Пусть мы решаем задачу:

$$\theta = \operatorname{argmin}_{\theta \in \Theta} L(\theta)$$

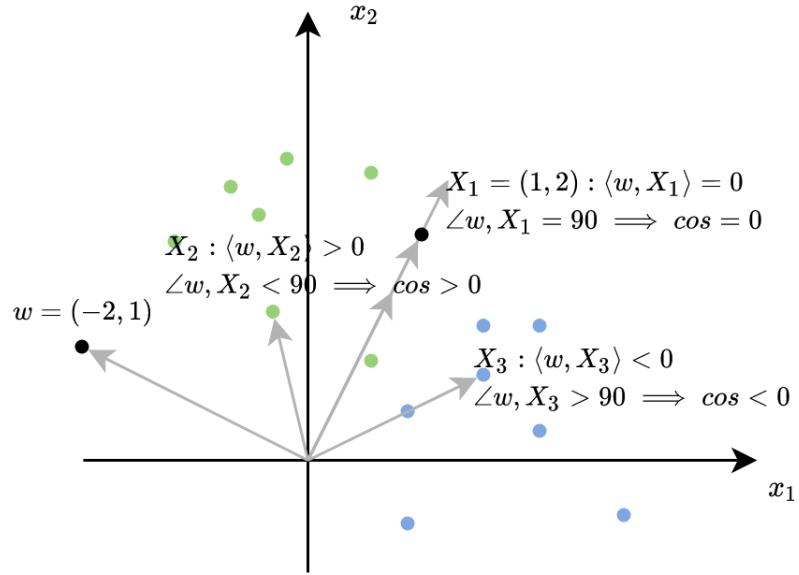
Регуляризация — это некоторые ограничения, которые мы накладываем на  $\theta$ , которые представляют сложность или вероятность модели и могут быть formalизованы с помощью некоторой функции  $c(\theta)$ :

$$\theta_{reg} = \operatorname{argmin}_{\theta \in \Theta} L(\theta) + ac(\theta)$$

- регуляризаторы, использующие  $l_1$ -норму или  $l_2$ -норму, наиболее популярные;
- **ElasticNet**, являющийся суммой двух предыдущих, также популярен;
- некоторые техники могут быть интерпретированы как регуляризация;
- регуляризация являются одним из двух подходов к борьбе с переобучением;
- регуляризация требуется для решения плохо поставленных задач;
- регуляризаторы могут быть добавлены для представления определенных специфических свойств модели;
- коэффициент регуляризации необходимо настраивать;
- не всегда понятно, какой регуляризатор выбирать.

## 4 Метод опорных векторов

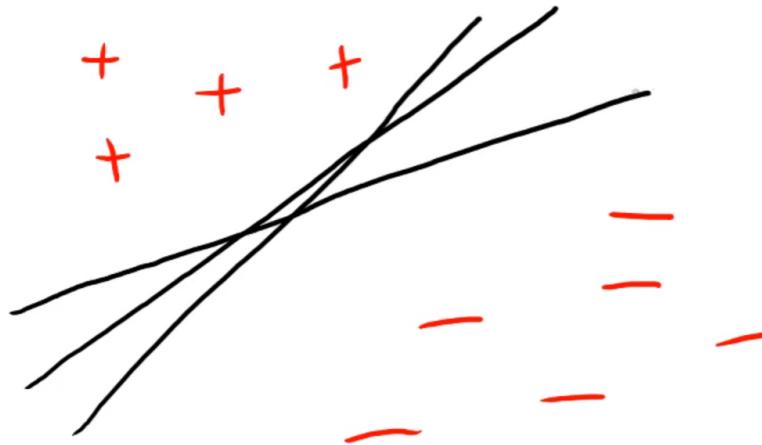
Относится к линейным классификаторам



#### 4.1 Линейно разделимый случай

Идея заключается в том, чтобы найти разделяющую гиперплоскость (прямую для двумерного пространства), которая наиболее удалена от классов (классификация с большим запасом).

Можно определить множество разделяющих прямых, но выбрать нужно одну.



Основная гипотеза заключается в том, что выборка является линейно разделимой:

$$\exists w, w_0 : M_i(w, w_0) = y_i(\langle w, x_i \rangle - w_0) > 0, i = 1, \dots, l.$$

Существуют такие веса, при которых отступ на каждом объекте положителен.

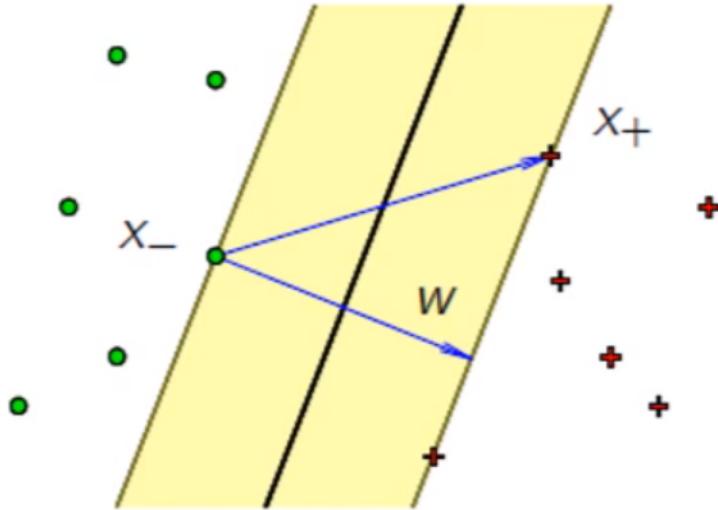
Определим формально искомую гиперплоскость. Нормализуем величину отступа (запас единица):

$$\min_i M_i(w, w_0) = 1$$

**Уравнение разделяющей полосы** (множество аргументов, при которых функция распознавания по абсолютному значению меньше 1):

$$\{x : -1 \leq \langle w, x \rangle - w_0 \leq 1\}$$

Полоса показана желтым цветом:



Ширина полосы:

$$\frac{\langle x_+ - x_-, w \rangle}{\|w\|} = \frac{(\langle x_+, w \rangle - w_0) - (\langle x_-, w \rangle - w_0)}{\|w\|} = \frac{2}{\|w\|}.$$

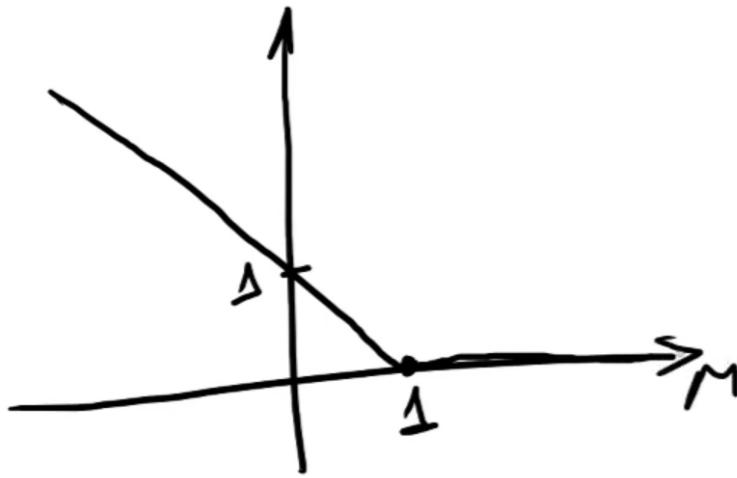
Здесь сначала находим вектор внутри полосы, оторый определяется опорными векторами разных классов, затем находим проекцию этого вектора на вектор весов, которая равна длине вектора, умноженного на косинус, либо скалярному произведению, деленному на вектор весов. Находить расстояние от точки до прямой через площадь параллелограмма здесь не нужно.

Т.е. полосы не зависит от объектов, лежащих на границе, а зависит только от вектора весов.

Формализуем задачу оптимизации. Пытаемся получить максимальную ширину полосы. Если минимизировать квадрат нормы вектора, то получим более сильное условие.

$$\begin{cases} \|w\|^2 \rightarrow \min_{w, w_0}; \\ M_i(w, w_0) \geq 1, i = 1, \dots, |D| \end{cases}$$

Функция ошибки в случае SVM:



## 4.2 Линейно неразделимый случай

На практике выборка обычно не является линейно разделимой:

$$\forall w, w_0 \exists x_d : M_d(w, w_0) = y_d(\langle w, x_d \rangle - w_0) < 0$$

Полностью разделительной гиперплоскости не существует, найдется объект, на котором будет допущена ошибка. Тогда будем пытаться найти гиперплоскость с наименьшими значениями отступов для каждого объекта.

В случае линейной неразделимости заданной выборки:

$$\begin{cases} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{|D|} \xi_i \rightarrow \min_{w, w_0, \xi}; \\ M_i(w, w_0) \geq 1 - \xi_i, i = 1, \dots, |D|, \xi_i \geq 0; \end{cases}$$

$\xi$  — релаксационная переменная, поправка для каждого объекта.

Мы ослабляем условие на отступ, вычитая из 1 поправку. Нужно минимизировать сумму поправок, чтобы находиться как можно ближе к линейно-разделимому случаю.

Так как мы не знаем, как соотносятся поправки с вектором весов, мы добавляем коэффициент  $C$ . Делим квадрат нормы  $w$ , чтобы было легче дифференцировать.

Эквивалентная задача безусловной оптимизации выражается через формулу аппроксимации эмпирического риска:

$$\sum_{i=1}^{|D|} (1 - M_i(w, w_0))_+ + \frac{1}{2C} \|w\|^2 \rightarrow \min_{w, w_0}$$

где  $(x)_+ = \frac{(x+|x|)}{2}$ .

Здесь уже нет поправок, так как:

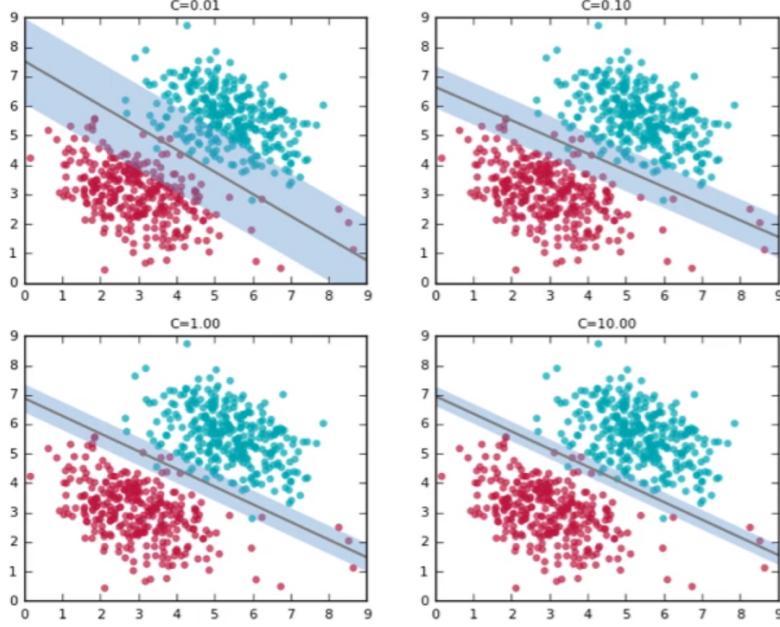
$$M_i(w, w_0) \geq 1 - \xi_i \rightarrow \xi_i \geq 1 - M_i(w, w_0)$$

Чем меньше  $1 - M_i(w, w_0)$ , тем меньше может быть  $\xi_i$ , поэтому теперь будем

минимизировать правую часть неравенства. Если подставить ее в первый кейс и разделить все на  $C$ , то придем к эквивалентной задаче.

Второе слагаемое в формуле отвечает за регуляризацию и является слагаемым гребневой регуляризации.

Гиперпараметр  $C$  регулирует баланс между шириной полосы и количеством ошибок. Его необходимо настраивать для конкретной задачи.



Рассмотрим как решаются системы, похожие на ту, что определена выше.

Задача математического программирования:

$$\begin{cases} f(x) \rightarrow \min_x, \\ g_i(x) \leq 0, & i = 1, \dots, m; j = 1, \dots, k. \\ h_j(x) = 0. \end{cases}$$

Имеется функционал, который нужно минимизировать. Также есть условия, которые выражаются функционалами  $g$  (ограничение-неравенство) и  $h$  (ограничение-равенство).

Лагранжиан:

$$L(x; \mu, \lambda) = f(x) + \sum_{i=1}^m \mu_i g_i(x) + \sum_{j=1}^k \lambda_j h_j(x)$$

где  $\mu, \lambda$  — двойственные переменные или множители Лагранжа.

Чтобы найти минимум требуется соблюсти условие Ка'руша-Ку'на-Та'ккера:

$$\frac{\delta L}{\delta x}(x^*; \mu, \lambda) = 0.$$

где  $x^*$  — точка минимума, в которой производная Лагранжиана равна нулю.

При этом должны соблюдаться следующие условия:

$$\begin{cases} g_i(x^*) \leq 0; \\ h_j(x^*) = 0; & i = 1, \dots, m; j = 1, \dots, k \\ \mu_i \geq 0; \\ \mu_i g_i(X^*) = 0. \end{cases}$$

Для нашей задачи:

$$\begin{cases} f \rightarrow \min_{w, w_0, \xi} \\ \xi_i \geq 0 \\ 1 - \xi_i - M_i \leq 0 \end{cases}$$

Лагранжиан:

$$L(w, w_0, \xi; \alpha, \beta) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{|D|} \alpha_i (M_i(w, w_0) - 1) - \sum_{j=1}^{|D|} \xi_j (\alpha_i + \beta_i - C)$$

где  $\alpha_i$  — переменные, двойственные для ограничений  $M_i \geq 1 - \xi$ ,  
 $\beta_i$  — переменные, двойственные для ограничений  $\xi_i \geq 0$ .

Условия минимума (имеем 3 переменные, поэтому находим частные производные по каждой):

$$\begin{cases} \frac{\partial L}{\partial w} = 0; \frac{\partial L}{\partial w_0} = 0; \frac{\partial L}{\partial \xi} = 0; \\ \xi_i \geq 0; \alpha_i \geq 0; \beta_i \geq 0; \\ \alpha_i = 0 \text{ или } M_i(w, w_0) = 1 - \xi_i; \\ \beta_i = 0 \text{ или } \xi_i = 0; \end{cases} \quad i = 1, \dots, |D|.$$

Производная квадрата нормы (производная по вектору):

$$\frac{d\|w\|^2}{dw} = \nabla\|w\|^2 = \nabla \sum_i w_i^2 = \nabla(w_1^2 + w_2^2 + \dots) = \begin{pmatrix} \frac{\partial(w_1^2 + w_2^2 + \dots)}{\partial w_1} = 2w_1 \\ \frac{\partial(w_1^2 + w_2^2 + \dots)}{\partial w_2} = 2w_2 \\ \dots \end{pmatrix} = 2w$$

Производная скалярного произведения векторов:

$$\frac{d\langle x, w \rangle}{dw} = \nabla(\langle x, w \rangle) = \nabla(x_1 w_1 + x_2 w_2 + \dots) = x$$

Таким образом:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{|D|} \alpha_i y_i x_i = 0 \implies w = \sum_{i=1}^{|D|} \alpha_i y_i x_i$$

$y_i$  — коэффициент из формулы отступа.

$$\frac{\partial L}{\partial w_0} = \sum_{i=1}^{|D|} \alpha_i y_i = 0 \implies \sum \alpha_i y_i = 0$$

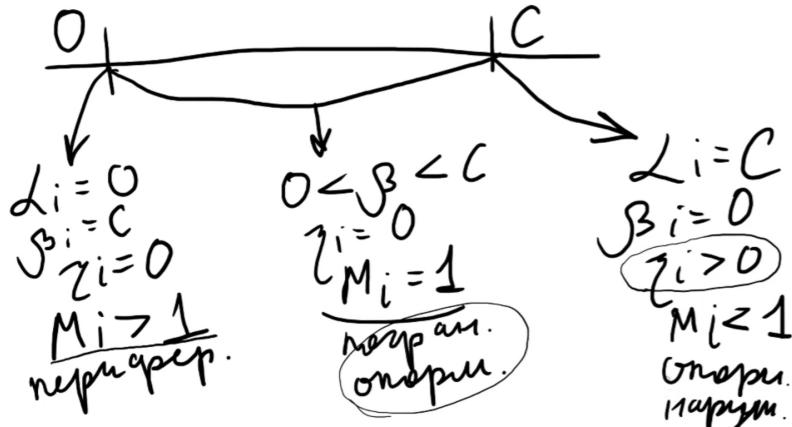
$$\frac{\partial L}{\partial \xi} = -\alpha_i - \beta_i + C = 0 \implies \alpha_i + \beta_i = C$$

Отсюда следует, что  $\alpha_i$  и  $\beta_i$  принадлежат отрезку от 0 до  $C$ .

Типы объектов:

1.  $\alpha_i; \beta_i = C; \xi_i = 0; M_i > 1$  — периферийные объекты (лежат за разделяющей полосой),
2.  $0 < \alpha_i < C; 0 < \beta_i < C; \xi_i = 0; M_i = 1$  — опорные пограничные объекты,
3.  $\alpha_i = C; \beta_i = 0; \xi_i > 0; M_i < 1$  — опорные нарушители.

Объект  $x_i$  — опорный, если  $a_i \neq 0$ . Через опорные объекты будем строить границы разделяющей полосы.



Упростим задачу оптимизации так, чтобы минимизировать только по одной переменной. Выразим  $w, w_0, \xi$  через  $\alpha$ . Получим:

$$-L(\alpha) = -\sum_{i=1}^{|D|} \alpha_i + \frac{1}{2} \sum_{i=1}^{|D|} \sum_{j=1}^{|D|} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \rightarrow \min_{\alpha}$$

$$\begin{cases} 0 \leq \alpha_i \leq C; \\ \sum_{j=1}^{|D|} \alpha_j y_j = 0. \end{cases}$$

Решение задачи может быть выражено следующим образом:

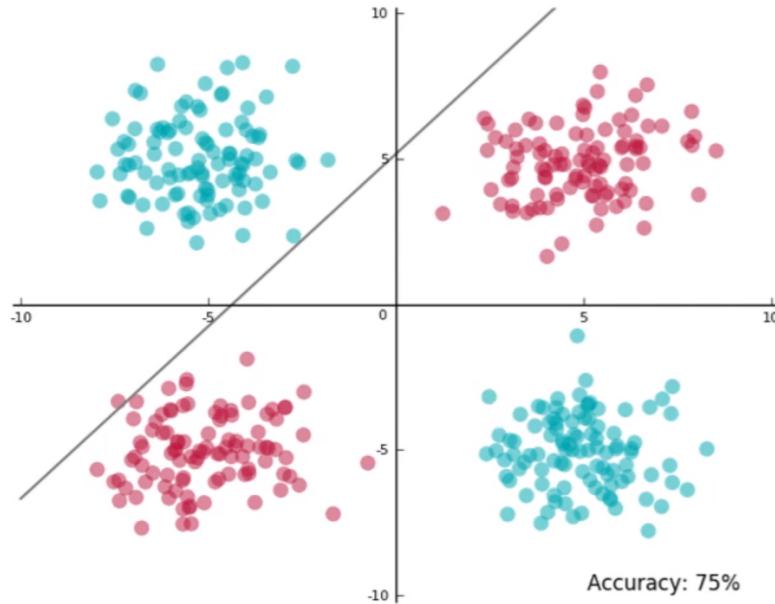
$$\begin{cases} w = \sum_{i=1}^{|D|} \alpha_i y_i x_i; \\ w_0 = \langle w, x_i \rangle - y_i \end{cases} \quad \forall i : \alpha_i > 0, M_i = 1.$$

Получаем линейный классификатор:

$$a(x) = \text{sign} \left( \sum_{i=1}^{|D|} \alpha_i y_i \langle x_i, x \rangle - w_0 \right)$$

### 4.3 Ядерный трюк

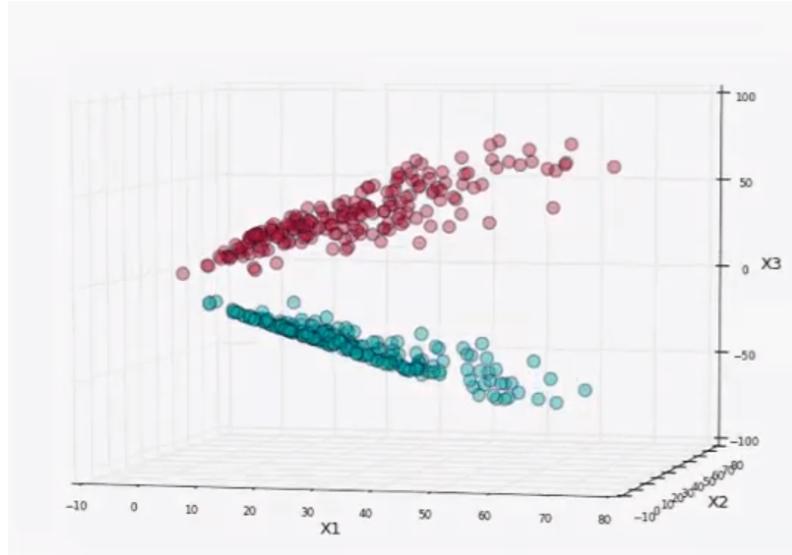
Плохой случай линейной неразделимости. Как бы мы не провели разделяющую гиперплоскость, точность будет плохая.



Основная идея: найти отображение в пространство большей размерности, где точки будут линейно разделимы.

Пусть разделяющая поверхность хорошо аппроксимируется суммой функций, зависящих от  $x_1, \dots, x_n : c_1 x_1 + \dots + c_n x_n + f_1(x_1, \dots, x_n) + \dots + f_k(x_1, \dots, x_n)$ . Если мы добавим признаки  $f_1(x_1, \dots, x_n), \dots, f_k(x_1, \dots, x_n)$ , тогда у нас будет новое пространство над переменными  $x_1, \dots, x_n, x_{n+1}, \dots, x_{n+k}$ , точки которых будут линейно разделимы.

Добавили третью размерность, теперь объекты хорошо разделимы:



Вместо скалярного произведения будем использовать функцию ядра. Проблема заключается в поиске ядра, которое переведет исходное пространство в линейно-разделимое.

#### 4.4 Выбор и синтез ядер

Функция  $K : X \times X \rightarrow R$  — функция ядра, если ее можно представить как  $K(x, x') = \langle \psi, \psi(x') \rangle$  с отображением  $\psi : X \rightarrow H$ , где  $H$  — пространство со скалярным произведением.

**Теорема Мерсера** Функция  $K(x, x')$  — ядерная функция, если она симметричная,  $K(x, x') = K(x', x)$  и неотрицательно определена на  $\mathbb{R}$ :

$$\int_X \int_X K(x, x') g(x) g(x') dx dx' \geq 0$$

для любой функции  $g : X \rightarrow \mathbb{R}$ .

Примеры функций ядра:

- $K(x, x') = \langle x, x' \rangle^2$  — квадратичное,
- $K(x, x') = \langle x, x' \rangle^d$  — многочлен с мономиальной степенью, равной  $d$ ,
- $K(x, x') = (\langle x, x' \rangle + 1)^d$  — многочлен с мономиальной степенью  $\leq d$ ,
- $K(x, x') = \sigma(\langle x, x' \rangle)$  — нейронные сети,
- $K(x, x') = \exp(-\beta \|x - x'\|^2)$  — радиально-базисный.

Ядра можно синтезировать. Правила для новых ядер:

- $K(x, x') = \langle x, x' \rangle$ ,
- $K(x, x') = K_1(x, x')K_2(x, x')$ ,
- $\forall \psi : X \rightarrow \mathbb{R} K(x, x') = \psi(x)\psi(x')$ ,
- $K(x, x') = \alpha_1 K_1(x, x') + \alpha_2 K_2(x, x')$  при  $\alpha_1, \alpha_2 > 0$

- $\forall \phi : X \rightarrow X$ , если  $K_0$  — функция ядра, тогда  $K(x, x') = K_0(\phi(x), \phi(x'))$ .
- Если  $s : X \times X \rightarrow R$  — симметричная и интегрируемая, тогда  $K(x, x') = \int_X s(x, z)(x', z)dz$

Анализ метода опорных векторов. Преимущества

- Задача выпуклого квадратичного программирования имеет единственное решение,
- Можно строить любую разделяющую поверхность (при помощи ядерного трюка),
- Небольшое количество опорных объектов, используемых для обучения.

Недостатки:

- Чувствителен к шуму
- Нет общих правил выбора функций ядра,
- Константу  $C$  требуется выбирать,
- Нет возможности выбора признаков.

## 4.5 Регуляризация для метода опорных векторов

Ключевая гипотеза заключается в том, что  $w$  скачет, что вызывает переобучение. Будем ограничивать норму  $w$ .

Добавим штраф регуляризации для нормы весов:

$$L_\tau(a_w, D) = L(a_w, D) + \frac{\tau}{2} \|w\|^2 \rightarrow \min_w.$$

Задача минимизации для метода опорных векторов:

$$\sum_{i=1}^l (1 - M_i(w, w_0)) + \frac{1}{2C} \|w\|^2 \rightarrow \min_{w, w_0}$$

Другие функции штрафов:

- Вектор релевантности:

$$\frac{1}{2} \sum_{i=1}^l \left( \ln \lambda_i + \frac{\alpha_i^2}{\lambda_i} \right)$$

- LASSO SVM (про  $\mu$  в следующем пункте):

$$\mu \sum_{i=1}^n |w_i|$$

- Машина опорных признаков (Support Feature Machine):

$$\sum_{i=1}^n R_\mu(w_i)$$

, где  $\mu$  — параметр селективности,  
 $R_\mu = \begin{cases} 2\mu|w_i|, & \text{if } w_i < \mu \\ \mu^2 + w_i^2, & \text{иначе.} \end{cases}$

## 5 Вероятностная классификация

### 5.1 Байесовская классификация

Теорема Байеса (или формула апостериорной вероятности):

$$p(x_i | y) = \frac{p(y | x_i)p(x_i)}{p(y)}$$

Допустим, сейчас в России ковидом болеет 300 000 человек или 0.5% населения. Допустим, у нас есть тест, который дает верный диагноз в 99% случаев. Безговидер сделал тест и получил положительный результат. Какая вероятность того, что он на самом деле болеет ковидом?

$$\frac{0.005 \cdot 0.99}{0.005 \cdot 0.99 + 0.995 \cdot 0.01} \approx 0.33$$

$y$  — положительный результат теста,  $x_1$  — относится к больным (тест не ошибся),  $x_2$  — не относится к больным (тест ошибся).

Если хочется посчитать вероятность для повторного теста с положительным результатом, то условную вероятность нужно просто возвести в квадрат.

Вместо неизвестной целевой функции  $y^*(x)$  будем думать о неизвестном распределении над  $X \times Y$  с плотностью  $p(x, y)$ .

Простой или независимой одинаково распределенной (independent identically distributed) называется выборка, содержащая независимые наблюдения из одного распределения.

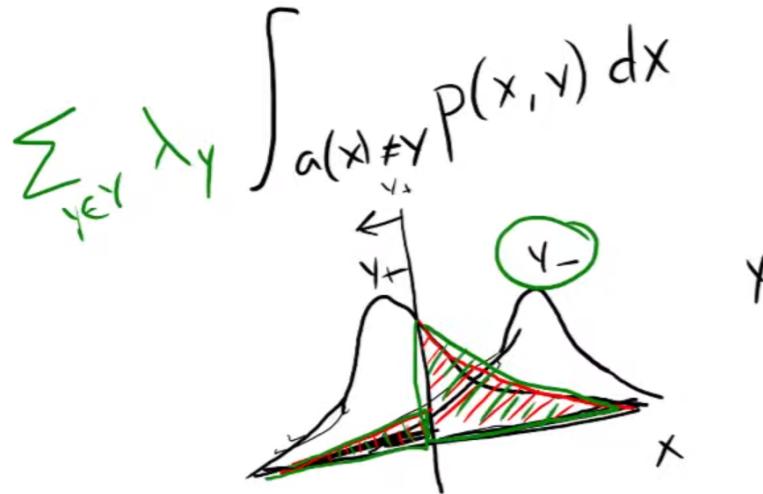
Задача: найти алгоритм классификации, который минимизирует вероятность ошибки.

Важно обеспечить независимость в выборке. Например, если делать выборку профилей во ВКонтакте и следующий профиль брать из друзей в предыдущем, то такая выборка не будет простой. Атрибуты друзей имеют связь, у них могут быть схожие политические взгляды и т.д.

По аналогии с эмпирическим риском введем **средний риск**:

$$R(a) = \sum_{y \in Y} \lambda_y \int_{a(x) \neq y} p(x, y) dx$$

, где  $\lambda_y$  — потеря от ошибки для объектов класса  $y$  (определяется экспертизой).



Иногда ошибаться в одних классах хуже, чем в других, поэтому нужны  $\lambda$ . Например, если мы классифицируем, болеет человек чумой или нет, то цена ошибки при отрицательной классификации должна быть выше.

$$p(x, y) = p(x)Pr(y | x) = Pr(y)p(x | y)$$

$Pr(y)$  — априорная вероятность класса  $y$ .

$p(x | y)$  — правдоподобие (likelihood) класса  $y$  (вероятность встретить в определенном классе определенный объект).

$Pr(y | x)$  — апостериорная вероятность класса  $y$  (то, что нужно для классификатора).

Рассматриваем задачу классификации, значит  $Y$  дискретная, поэтому используем вероятность ( $Pr(y)$ ). Признак непрерывный, поэтому для него используем плотность  $p(x)$ .

$$Pr(y | x) \sim Pr(y)p(x | y)$$

Можно убрать  $p(x)$  в знаменателе.

Проблема восстановления плотности распределения. Чтобы найти апостериорную вероятность класса, нужно найти эмпирические оценки  $\hat{Pr}(y)$  и  $\hat{p}(x | y)$ ,  $y \in Y$ .

Вторая проблема заключается в минимизации среднего риска. Если нам даны априорные вероятности  $Pr(y)$  и правдоподобие  $p(x | y)$ ,  $y \in Y$ , то нужно найти классификатор  $a$  с минимальным  $R(a)$ .

Первая проблема намного сложней второй. Ее решением занимается статистика.

Рассмотрим вторую проблему. Для нового объекта будем возвращать класс, к которому он принадлежит с наибольшей вероятностью. **Оценка апостериорного**

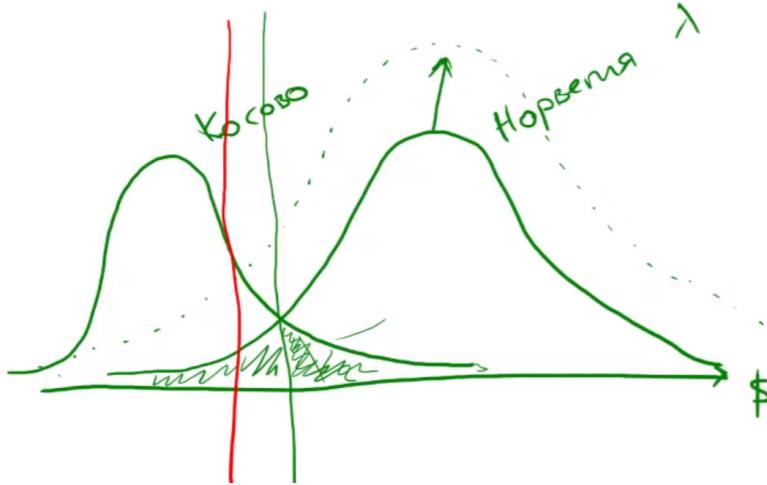
максимума (maximum a posteriori probability, MAP):

$$a(x) = \operatorname{argmax}_{y \in Y} Pr(y | x) = \operatorname{argmax}_{y \in Y} Pr(y)p(x | y)$$

**Теорема** Если известны  $Pr(y)$  и  $p(x | y)$ , то минимальный средний риск достигается байесовским классификатором (Optimal Bayesian Classification)  $a_{OB}$ :

$$a_{OB}(x) = \operatorname{argmax}_{y \in Y} \lambda_y Pr(y)p(x | y)$$

Классифицируем клиента по количеству оставленных чаевых:

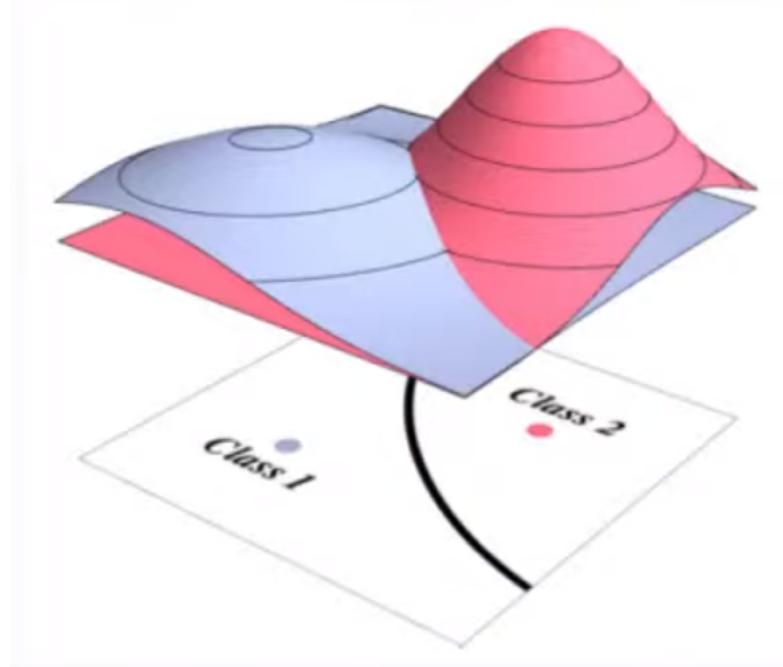


Классификатор  $a_{OB}(x)$  называется оптимальным байесовским классификатором, а достигаемое им значение  $R(a_{OB})$  — байесовским риском.

Чем лучше мы восстанавливаем распределения, тем лучше будет работать байесовский классификатор.

**Разделяющая поверхность** для двух классов  $y_+$  и  $y_-$  — это множество точек  $x \in X$ , на которых достигается равенство в байесовском разделяющем правиле:

$$\lambda_{y_+} Pr(y_+) p(x | y_+) = \lambda_{y_-} Pr(y_-) p(x | y_-)$$



## 5.2 Непараметрическое восстановление плотности распределения

Оценим априорную вероятность класса эмпирически:

$$\hat{Pr}(y) = \frac{|X_y|}{|D|}, \quad X_y = \{x_i, y_i \in D, y_i = y\}$$

В таком случае объекты в нашей выборке (датасете) должны быть распределены так же, как и объекты в генеральной совокупности. Такое бывает очень редко, поэтому вероятности можно брать из внешних источников, например, из статистических исследований.

Найдем  $\hat{p}(x \mid y)$  для каждого класса независимо. Поэтому будем писать  $\hat{p}(x)$ , которое нужно восстановить над над  $D_s = ((x_{(1)}, s), \dots, (x_{(m)}, s))$  для каждого  $s \in Y$ . Иными словами восстанавливаем  $p(x)$  по объектам из подвыборки для определенного класса. Так мы перешли к задаче восстановления плотности распределения по наблюдениям.

Если  $Pr([a, b])$  является метрикой вероятности на отрезке  $[a, b]$ , то

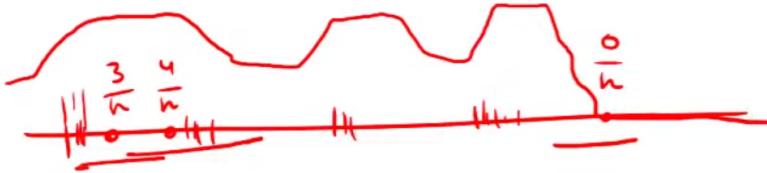
$$p(x) = \lim_{h \rightarrow 0} \frac{1}{2h} Pr([x - h, x + h])$$

Эмпирическая оценка вероятности с окном шириной  $h$ :

$$\hat{p}_h(x) = \frac{1}{2mh} \sum_{i=1}^m [|x - x_i| < h]$$

Проще говоря, вероятностью объекта будет  $1/n$ , где  $n$  — количество объектов. Так как у нас недостаточно данных, чтобы посчитать вероятность в каждой точ-

ке, будем считать вероятность на промежутке. Таким образом получим приближение к плотности распределения.



Сгладим окно:

$$\hat{p}_h(x) = \frac{1}{2hm} \sum_{i=1}^m \left[ \frac{x - x_i}{h} < 1 \right]$$

И перейдем к оценке Парзена-Розенблатта:

$$\hat{p}_h(x) = \frac{1}{hm} \sum_{i=1}^m K\left(\frac{x - x_i}{h}\right)$$

$\hat{p}_h(x)$  сходится к  $p(x)$ .

Если объекты описываются  $n$  вещественными признаками:

$$\hat{p}_h = \frac{1}{m} \sum_{i=1}^m \prod_{j=1}^n \frac{1}{h_j} K\left(\frac{f_j(x) - f_j(x_i)}{h_j}\right)$$

Если  $X$  — метрическое пространство с мерой расстояния  $\rho(x, x')$ :

$$\hat{p}_h(x) = \frac{1}{mV(h)} \sum_{i=1}^m K\left(\frac{\rho(x, x_i)}{h}\right)$$

, где  $V(h) = \int_X K\left(\frac{\rho(x, x_i)}{h}\right) dx$  — множитель нормализации (нужен, чтобы интеграл по плотности давал 1).

Вставим эту оценку в ОБК:

$$a(x; D, h) = \operatorname{argmax}_{y \in Y} \lambda_y Pr(y) \frac{1}{|D_y|} \sum_{i:y_i=y} K\left(\frac{\rho(x, x_i)}{h}\right)$$

Близость к классу:

$$\Gamma_y(x) = \lambda_y Pr(y) \frac{1}{|D_y|} \sum_{i:y_i=y} K\left(\frac{\rho(x, x_i)}{h}\right)$$

Чем больше у нас размерность, тем сложнее восстанавливать распределение.

Наивная гипотеза: все признаки соответствуют независимым случайным величи-

нам с плотностями вероятностей  $p_j(\xi_j | y)$ ,  $y \in Y, j = 1, \dots, n$ . В реальности этого быть не может, какая-то связь между признаками будет всегда.

Тогда правдоподобие классов можно расписать следующим образом:

$$p(x | y) = p_1(\xi_1 | y) \cdots p_n(\xi_n | y), \quad x = (\xi_1, \dots, \xi_n), \quad y \in Y.$$

Теперь считать легче, но мы потеряли часть информации. Подход хорошо применять при классификации текстов, где признаком является частота определенного слова в тексте. Если частота считается не по словам, а по n-граммам (последовательностям из n слов), то размерность вырастает еще сильнее. Между документами связь слов не нужна, поэтому потеря этой информации при использовании метода не страшна.

Исходя из нового правдоподобия классов построим классификатор (пусть лямбда равна 1):

$$a(x) = \operatorname{argmax}_{y \in Y} \left( \hat{Pr}(y) \cdot \prod_{j=1}^n \hat{p}_j(\xi_j | y) \right)$$

Имеется произведение от большого числа признаков. Это делает классификатор очень неустойчивым, так как один ноль обнуляет все. Перейдем от произведения к сумме.

Наивный байесовский классификатор:

$$a_{NB}(x) = \operatorname{argmax}_{y \in Y} \left( \ln \lambda_y \hat{Pr}(y) + \sum_{j=1}^n \ln \hat{p}_j(\xi_j | y) \right)$$

$p(\xi_i | y)$  считаем для каждого слова в каждом классе.

Чтобы не получить логарифм нуля, прибавляем к его числу маленькое значение.

### 5.3 Параметрическое восстановление плотности распределения

Будем выбирать распределение из параметрического семейства распределений. Будем искать  $p(x, y) \in \{\phi(x, y, \theta) | \theta \in \Theta\}$ , как раньше искали  $a(x, \theta)$ .

Плотность совместного распределения выборки:

$$p(D) = p((x_1, y_1), \dots, (x_{|D|}, y_{|D|})) = \prod_{(x,y) \in D} p(x, y)$$

То есть вероятность наблюдать данные — это вероятность наблюдать каждый объект в этих данных.

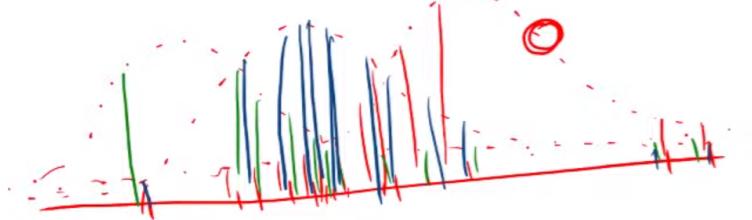
Правдоподобие данных (вероятность наблюдения данных):

$$L(\theta, D) = \prod_{(x,y) \in D} \phi(x, y, \theta)$$

MAP:

$$a_\theta(x) = \operatorname{argmax}_y \phi(x, y, \theta)$$

Чаще всего в качестве модели выбирают нормальное распределение. Параметрами будут средняя и дисперсия. В многомерном случае это вектор средних и матрица ковариаций. Простроим возможные распределения и выберем то, где вероятность наилучше налюдать данные будет наибольшей.



Возьмем логарифм и поменяем знак:

$$-\ln L(\theta, D) = -\sum_{(x,y) \in D} \ln \phi(x, y, \theta) \rightarrow \min_{\theta}$$

Выведем функцию потерь на объекте:

$$L(a_\theta, x) = -|D| \ln \phi(x, y, \theta)$$

Задача минимизации эмпирического риска:

$$\begin{aligned} L(a_\theta, D) &= \frac{1}{|D|} \sum_{x \in D} L(a_\theta, x) = \\ &= -\frac{1}{|D|} \sum_{(x,y) \in D} |D| \ln \phi(x, y, \theta) = -\sum_{(x,y) \in D} \ln \phi(x, y, \theta) \rightarrow \min_{\theta} \end{aligned}$$

**Принцип максимального правдоподобия** (выбираем параметры, которые дают оптимальные данные):

$$L(\theta; D_s) = \sum_{x \in D_s} \ln \phi(x; \theta) \rightarrow \max_{\theta}$$

Здесь  $L$  — likelihood.

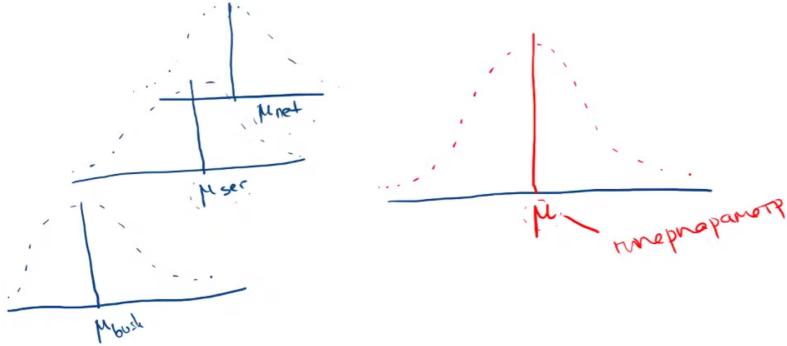
Оптимум  $\theta$  можно найти в точке, где производная

$$\frac{\partial L(\theta; D_s)}{\partial \theta} = 0$$

Чаще всего это можно искать только итеративно, например, с помощью градиентного спуска.

Рассмотрим распределения роста людей по странам. Вертикальной оборо-

значено среднее значение. Оно является параметром распределения. Можно построить распределение среднего значения роста. Среднее такого распределения будет называться гиперпараметром  $\gamma$ .



При помощи таких распределений можно оценивать правдоподобие параметров. Преобразуем  $\phi$ :

$$\phi(x_i, y_i, \theta) = p(x_i, y_i | w)p(w, \gamma),$$

где  $p(x_i, y_i | w)$  — вероятностная модель данных,  
 $p(w, \gamma)$  — априорное распределение параметров модели,  
 $\gamma$  — гиперпараметр (в статистическом смысле).

Пролагорифмируем и получим **принцип совместного максимального правдоподобия**:

$$\sum_{(x_i, y_i) \in D} \ln p(x_i, y_i | w) + \ln p(w, \gamma) \rightarrow \max_{w, \gamma}$$

Первое слагаемое выражает потерю на объекте, второе слагаемое выражает штраф за неправдоподобие модели. Если данные описываются редкими моделями, возможно наступило переобучение. По сути, второе слагаемое добавляет регуляризацию.

Часто в качестве распределения данных используют  $n$ -мерное нормальное распределение. Пусть  $w \in R^n$ , тогда

$$p(w; \sigma) = \frac{1}{(2\pi\sigma)^{n/2}} \exp\left(-\frac{\|w\|^2}{2\sigma}\right)$$

, где веса независимы,  
матожидания весов равны нулю,  
 $\sigma$  — дисперсия весов.

Из этого получаем квадратичную регуляризацию:

$$-\ln p(w; \sigma) = \frac{1}{2\sigma} \|w\|^2 + \text{const}(w)$$

## 5.4 Мягкая классификация

Вместо того, чтобы возвращать только класс, можно возвращать вектор вероятностей каждого класса:

$$\begin{aligned} a : X &\rightarrow Y, & a(x) &= y \\ b : X &\rightarrow \mathbb{R}^{|Y|}, & b(x) &= (q_1, \dots, q_{|Y|}), \sum_i q_i = 1 \end{aligned}$$

Мягкая классификация более информативна и позволяет использовать более чувствительные функции ошибки.

Перекрестная энтропия:

$$H(p, p') = - \sum_{x \in X} p(x) \log p'(x)$$

Функция ошибки перекрестной энтропии (cross-entropy loss):

$$L(b, x) = -\log q_{y(x)},$$

где  $q_{y(x)} = y(x)$  элемент вектора  $b(x)$ .

Можем пытаться предсказать вероятности. Рассмотрим бинарную классификацию:

$$q_+ + q_- = 1.$$

Можно предсказывать вероятность одного из классов, например,  $q_+$ . Вторую вероятность получим из первой.

Это не задача регрессии, потому что  $0 \leq q_+ \leq 1$ , а не просто  $q_- \in \mathbb{R}$ . Преобразуем  $q_+$ , чтобы получить интервал от 0 до  $+\infty$ , затем возьмем логарифм, чтобы расширить интервал до  $-\infty$ :

$$\log \frac{q_+}{1 - q_+} \in R$$

Такое преобразование называется логит-преобразованием. Преобразование монотонно.

Возьмем для предсказания линейную регрессию:

$$\log \frac{q_+}{1 - q_+} = \langle w, x \rangle,$$

отсюда

$$q_+ = \frac{1}{1 + e^{\langle w, x \rangle}}.$$

Логистическая регрессия:

$$a_{LogReg}(x) = \sigma(\langle w, x \rangle),$$

где  $\sigma(x) = \frac{1}{1 + e^x}$  — сигма-функция или сигмоида.

Сигмоида выглядит как буква S, ее задача перевести аргумент в интервал от 0 до 1. Если сигмоиду заменить функцией sign, то получим линейный классификатор.

Логарифмическая функция потерь:

$$L(a, D) = \sum_{(x,y) \in D} \ln(1 + \exp(-\langle w, x \rangle y)) \rightarrow \min_w$$

Красным показан штраф, который дает логарифмическая функция потерь.

Начисление штрафа за положительный аргумент заставляет алгоритм находить разделяющую поверхность, которая максимально делит объекты.

Несколько свойств сигмоиды. Производная:

$$\sigma'(x) = \sigma(x)\sigma(-x).$$

Градиент:

$$\nabla L(w_{(k)}) = - \sum_i^{|D|} y_i x_i \sigma(-M_i(w_{(k)})).$$

Шаг градиентного спуска:

$$w_{(k+1)} = w_{(k)} - \mu y_{(k)} x_{(k)} \sigma(-M_i(w_{(k)})).$$

## 6 Деревья решений и композиции

### 6.1 Логические правила

Предикат на множестве X:

$$\phi : X \rightarrow \{0, 1\}$$

Функция, которая возвращает либо 0, либо 1. Предикат **покрывает** объект  $x$ , если  $\phi(x) = 1$ . **Закономерность** — предикат, покрывающий подавляющее большинство объектов одного класса и малое число объектов других классов. **Правило** — легко интерпретируемая и высокинформативная закономерность, описываемая простыми логическими формулами.

Закономерность  $\phi$  называется интерпретируемой, если ее можно сформулировать на естественном языке и она зависит от небольшого числа параметров. Существует исследование, в котором утверждается, что правило должно иметь менее 7 параметров, чтобы человек его понял. Пример из русского языка: если слово является наречием и оканчивается на шипящую букву, в конце ставится мягкий знак. В качестве исключений слова: "уж" "замуж" "невтерпеж".

Закономерность является информативной для класса  $c$ , если

$$p(\phi) = |\{x_i \mid \phi(x_i) = 1, y_i = c\}| \rightarrow \max n(\phi) = |\{x_i \mid \phi(x_i) = 1, y_i \neq c\}| \rightarrow \min$$

, где  $p(\phi)$  — число истинно положительных значений,  
 $n(\phi)$  — должно положительных.

## 6.2 Индукция правил

Иначе правило можно определить как одноклассовый классификатор с отказами (может не классифицировать часть объектов).

Примеры построения. **Конъюнкция пороковых условий:**

$$R(x) = \bigwedge_{j \in J} [a_j \leq f(x_j) \leq b_j]$$

, где  $J$  — подмножество признаков, является гиперпараметром.

Накладываем на признаки граничные условия и выбираем объекты, которые им соответствуют.

**Синдром** — выполнение не менее  $d$  условий из  $J$ :

$$R(x) = \left[ \sum_{j \in J} [a_j \leq f(x_j) \leq b_j] \geq d \right]$$

Когда  $d = |J|$  получается конъюнкция, когда  $d = 1$  — дизъюнкция.

**Полуплоскость** (по подмножеству признаков строится гиперплоскость и берутся объекты на положительной стороне):

$$R(x) = \left[ \sum_{j \in J} w_j f_j(x) \geq w_0 \right]$$

**Шар** (выбираем объекты, которые лежат в определенном радиусе):

$$R(x) = [r(x, x_0) \leq r_0]$$

Правила можно создавать самостоятельно, заимствовать у экспертов, обучать.

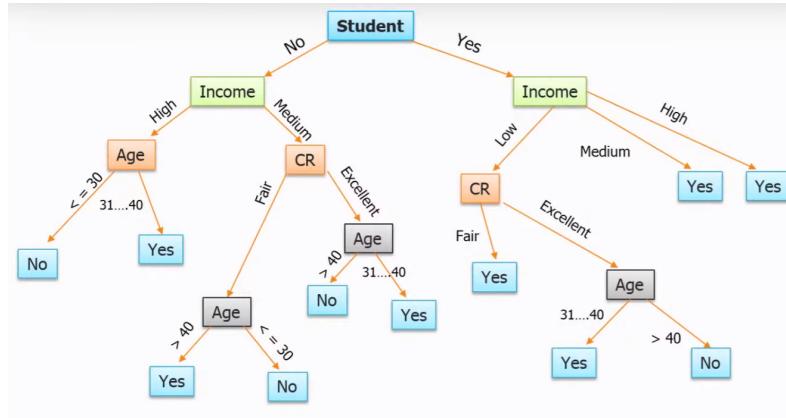
Правила можно обучать при помощи алгоритмов оптимизации, эвристических методов, специальными алгоритмами машинного обучения.

В основном это делается через экспертов.

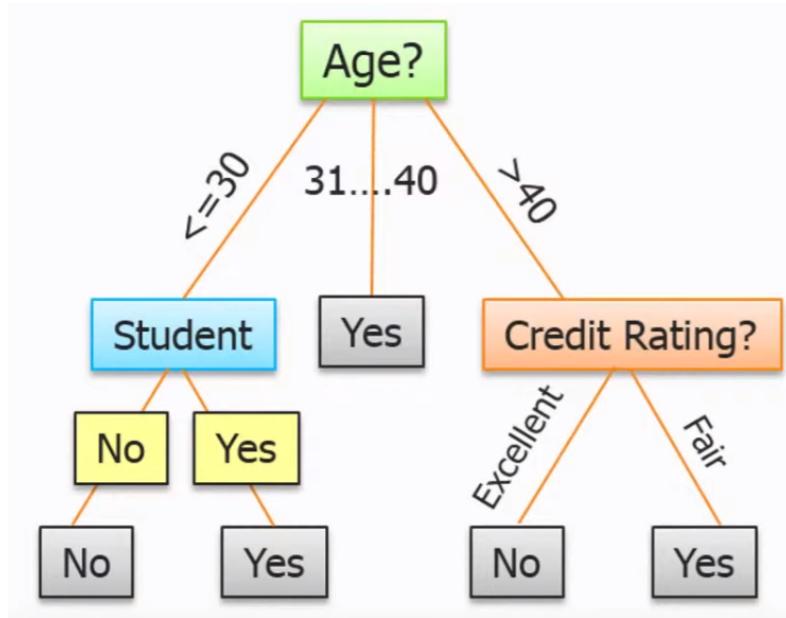
## 6.3 Деревья решений

Дерево решений — алгоритм классификации или регрессии. Вершины содержат разделяющие правила (вопросы), ребро является возможным ответом на вопрос в родительской вершине. Листья содержат решения (класс объекта для задачи классификации или число для задачи регрессии).

Решение для выдачи кредита:



То же самое, но проще:

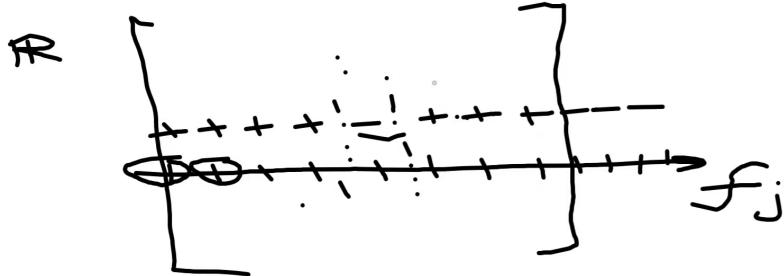


Рассмотрим построение дерева. Пусть имеется множество разделяющих правил  $B$  и критерий ветвления  $\Phi$ .

- Кладем всю выборку  $S$  в корневую вершину.
- На каждом шаге рекурсивно обрабатываем  $S$ .
- Если  $S$  содержит объекты только одного класса, то создаем лист с этим классом и останавливаемся.
- Если это не так, выбираем правило  $b \in B$ , наилучшее с точки зрения  $\Phi$ , и разделяем выборку на  $S_1, \dots, S_k$ .
- Если выполнился критерий остановки, то возвращаем наиболее популярный класс в текущей выборке  $S$ , в противном случае создаем  $k$  детей вершины, соответствующих  $S_i$ ,  $i = [1, \dots, k]$ .
- Подрезаем итоговое дерево.

Семейство разделяющих правил является семейством классификаторов. В большинстве случаев выбираются правила для одного признака:  $f_i(x) > m_i$  (для численных признаков),  $f_i(x) = d_i$  (для категориальных). Можно создавать комбинацию таких правил.

Значения признака с соответствующими метками объектов:



Разделили значения на несколько частей. Создадим правило, которое ошибается на одном объекте. Лучше ошибаться на небольшом количестве объектов, чем создавать для них дополнительную проверку.

Существует  $|D| - 1$  способов разбиения выборки. Рассмотрим каждое правило и выберем наиболее информативные, объединим диапазоны значений, пропустим маленькие диапазоны (один отрицательный объект в примере). Таким образом можно синтезировать правило для каждого признака.

Рассмотрим случаи разбиения выборки. Если выборка каждый раз делится на 2 части, то  $B$  — семейство бинарных правил, а дерево бинарное.

Если признак категориальный, то можно строить несколько ребер из вершины.

Если признак численный, то применяется бинаризация (дискретизация), т.е. перевод в категориальный.

На каждом шаге построения алгоритма количество ребер может быть различным, но обычно  $k$  — фиксированное.

Критерий ветвления  $\Phi$  задается следующей формулой:

$$\Phi(S) = \phi(S) - \sum_{i=1}^k \frac{|S_i|}{|S|} \phi(S_i)$$

Наиболее популярные функционалы  $\phi$ :

- $\phi_h(S)$  — энтропия,  $\Phi_h(S)$  называется IGain;
- $\phi_g(S) = 1 - \sum_{i=1}^m p_i^2$ , где  $p_i$  — вероятность (частота) присутствия элементов  $i$  класса в выборке  $S$  — индекс Джини.  $\Phi_h(S)$  называется GiniGain.

Используются также и другие критерии:

$$GainRatio = IGain(S)/Entropy(S)$$

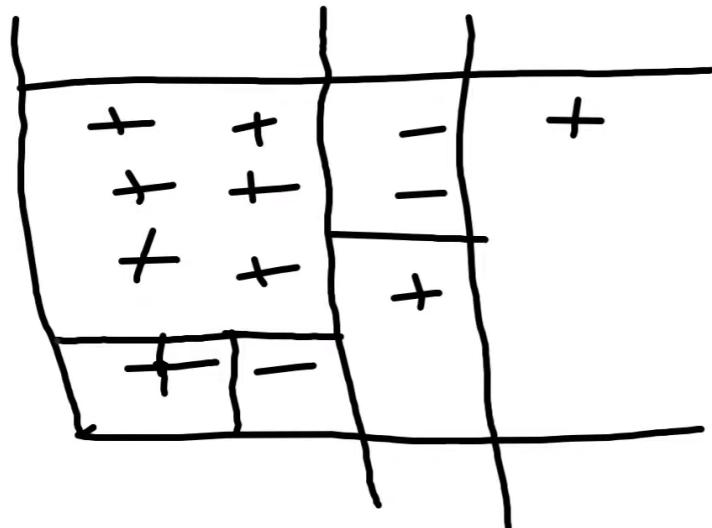
Выбор критерия ветвления обычно не сильно влияет на качество построения

дерева.

Популярные критерии остановки:

- один из классов остался пустым после разбиения,
- $\Phi(S)$  ниже определенного порога,
- $|S|$  ниже определенного порога,
- высота дерева выше определенного порога.

Рассмотрим двухклассовую классификацию с 2 признаками. Добавление вершины фактически означает добавление разделяющей плоскости. Слишком большое количество правил приводит к переобучению.



Другой тип критерия остановки — **предподрезка**. Останавливает рост дерева, когда нет статистически значимой связи между каким-либо признаком и классом в конкретном узле. Обычно используется тест  $\chi^2$ .

Только верхние вершины влияют на качество алгоритма, так как деревья решений склонны переобучаться. Будем отрезать нижние ветви. Обрезка — это обработка созданных деревьев, когда последовательно применяются операторы упрощения, если они улучшают качество по определенному критерию (например, уменьшение количества ошибок).

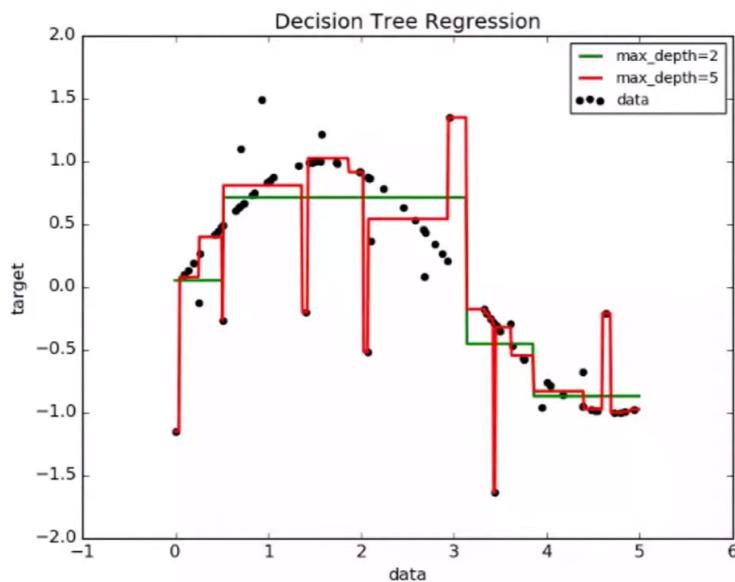
Рассмотрим, как работает алгоритм подрезки. Разбиваем выборку на тренировочную и тестовую в отношении 2:1. Для каждой вершины применяем определенный оператор упрощения, который является лучшим с точки зрения выбранного критерия качества:

- ничего не менять,
- заменить вершину ребенком (для каждого ребенка),
- заменить вершину листом (для каждого класса).

Известные реализации:

- ID3 (Quinlan, 1986), IGain,  $\Phi(S) < 0$ , без подрезки;
- C4.5 (Quinlan, 1993), GainRatio, с подрезкой;
- CART (Breinman, 1984), бинарное, GiniGain, подрезка, предназначено для регрессии.

Регрессионные деревья используются для решения задачи регрессии. Они работают так же, как и обычные деревья решений, но в листе возвращается среднее значение целевого признака вместо мажоритарного класса.



Красное дерево переобучилось. В листе находится среднее значение целевого признака.

**Деревья моделей** решают задачу регрессии. Они работают так же, как деревья решений и регрессии, но каждый лист возвращает некоторые гиперпараметры модели. Обычно используются для линейных моделей.

Преимущества алгоритма:

- легко понять и интерпретировать,
- быстро обучается,
- может работать с разными типами данных,
- выполняет выбор признаков внутри себя.

Недостатки:

- чувствителен к шумам,
- быстро переобучается.

6.4 Композиция алгоритмов

6.5 Бустинг

6.6 AdaBoost

6.7 а

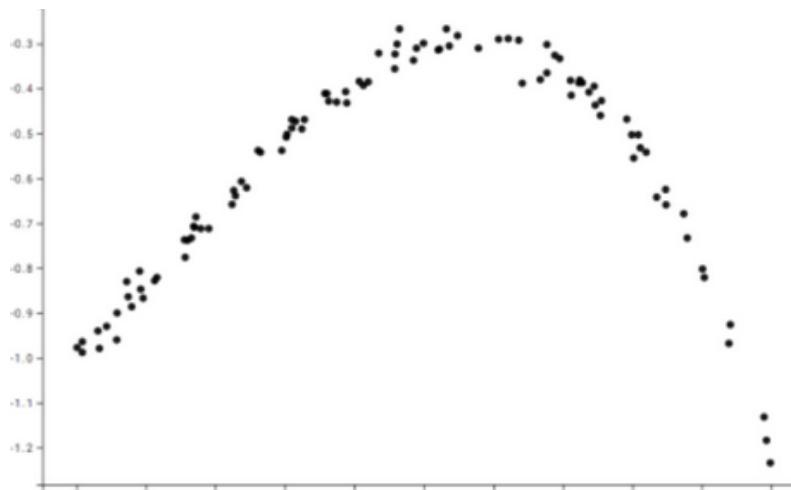
6.8 а

6.9 а

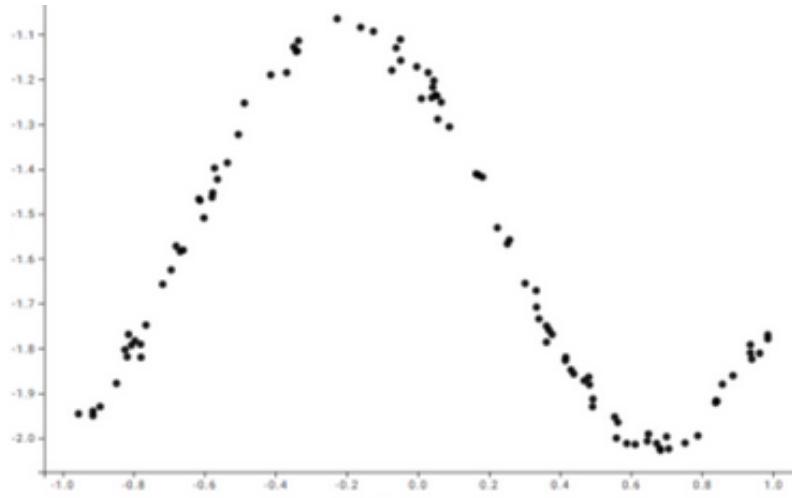
## 7 Автоматическое дифференцирование и нейронные сети

### 7.1 Сведение задачи обучения к задаче дифференцирования

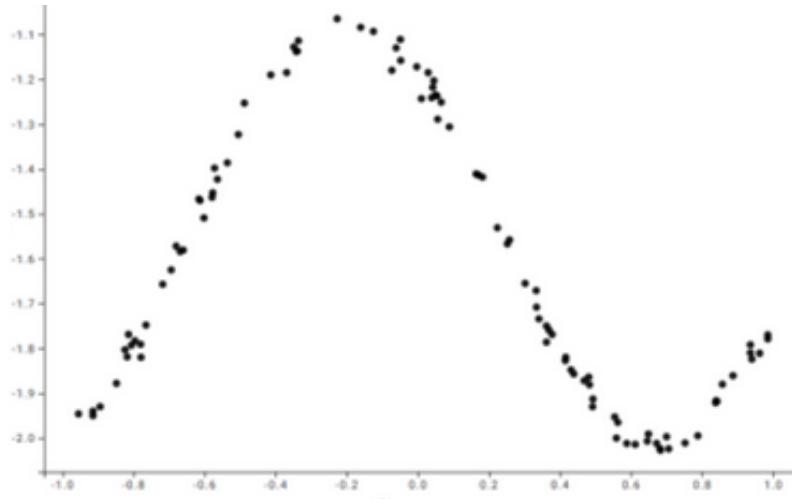
Пусть имеется набор данных со скрытой зависимостью, которую мы будем аппроксимировать. Нам нужно определиться, какую аппроксимирующую функцию использовать и какие ее параметры обучать.



Здесь можно взять полином:  $y(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \dots$  с обучаемыми параметрами  $a_0, a_1, a_2, \dots$ .

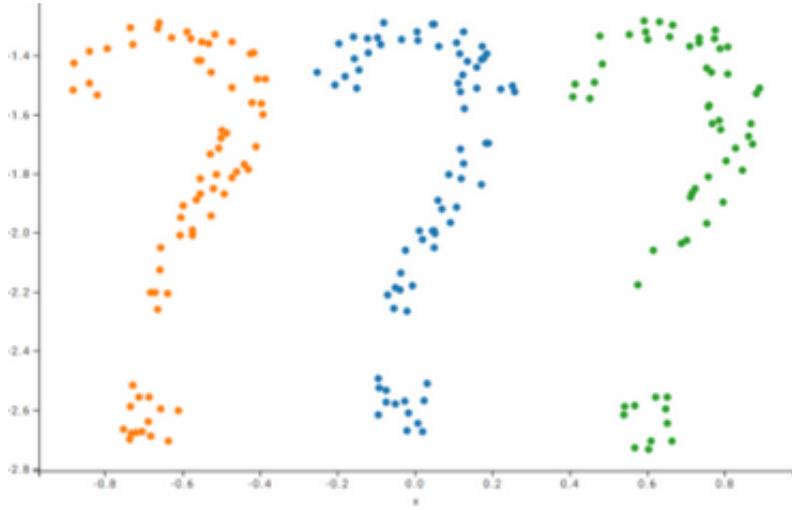


Здесь синусоида:  $y(x) = \sin(a_0 \cdot x + a_1) \cdot a_2 + a_3$  с обучаемыми периодом, фазой, амплитудой и сдвигом соответственно:  $a_0, a_1, a_2, a_3$ .



Здесь кривая второго порядка (сечение конусов), которая задается математической моделью  $a_0y^2 + a_1x^2 + a_2yx + a_3y + a_4x + a_5 > 0$  с обучаемыми параметрами  $a$ .

Нас интересует общий метод обучения, который можно применять для неизвестной модели  $f(x, y, a) = (p_1, p_2, p_3)$ , где  $a$  — вектор обучаемых параметров.



Обучаемая функция:  $f(x_1, x_2, \dots, x_n, a_1, a_2, \dots, a_m) : R^{n+m} \rightarrow R^k$ , принимает вектор входных параметров состоит из признаков объекта и обучаемых параметров. Выходом является вектор, который можно заменить множеством функций с одним выходом.

Набор данных:  $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ , состоит из множества объектов с известными целевыми значениями.

Функция ошибки:  $E(\hat{y}_1, \dots, \hat{y}_{|D|}, y_1, \dots, y_{|D|})$ , которая принимает предсказанный (наблюдаемый) вектор и целевой (ожидаемый) вектор. Для регрессии это среднеквадратичное отклонение  $MSE(\hat{y}_1, \dots, \hat{y}_{|D|}, y_1, \dots, y_{|D|}) = \frac{1}{|D| \cdot k} \sum_{i=1}^{|D|} \sum_{j=1}^k (y_{i,j} - \hat{y}_{i,j})^2$

Режим обучения определяет обучаемую функцию ошибки, где зафиксированы объекты, а обучаемые параметры изменяются для минимизации функции:  $E_{train}(a_1, \dots, a_m) = E(f(x_1, a), \dots, f(x_{|D|}, a), y_1, \dots, y_{|D|})$ .

Режим предсказания:  $f_{predict}(x_1, \dots, x_n) = f(x_1, \dots, x_n, a_1, \dots, a_m)$ . Здесь набор мы имеет обученные параметры, а объекты меняются, так как могут быть неизвестные.

Обучаемая функция и функция ошибки должны быть дифференцируемыми, так как минимизация выполняется градиентным спуском (выпуклость не обязательна, но если это не так, то можем попасть в локальный минимум). Иначе используется генетический алгоритм.

Можем модифицировать функцию ошибки, превращая градиентный спуск в пакетный  $E(\hat{y}_1, \dots, \hat{y}_r, y_1, \dots, y_r)$ , где  $1 \leq l \leq r \leq |D|$ , стохастический  $E(\hat{y}_i, y_i)$ , где  $1 \leq i \leq |D|$  добавлять регуляризацию  $E_{train}(a) = E_{old}(a) + E_{reg}(a)$ .

В задачах классификации функции ошибки не являются дифференцируемыми, поэтому задачу сводят к задаче мягкой классификации, когда обучаемая функция вместо класса предсказывает вектор вероятностей принадлежности объекта к каждому классу. Рассмотрим несколько подходов к решению такой задачи:

- Наивный подход заключается в использовании функции ошибки для задачи восстановления регрессии (MSE).

- Можно использовать  $F_1$ -меру, которая является функцией ошибки для задачи классификации:  $F_1 : CM \rightarrow R$ , где  $CM$  — матрица неточностей. Данная мера является дифференцируемой и нам не требуется, чтобы  $CM$  содержала только целые числа, поэтому будем прибавлять вектор вероятностей  $p$  к соответствующей реальному классу строке матрицы.
- Использовать перекрестную энтропию:  $H(p, q) = -\sum_{i=1}^k q_i \log p_i$ , где  $q$  — реальное распределение вероятностей, а  $p$  — предсказанное. Для классификации вырождается в метод максимального правдоподобия, так  $q$  состоит из 0 и одной 1, получаем сумму логарифмов вероятностей реального класса.

## 7.2 Дифференцирование составных функций по графу вычислений

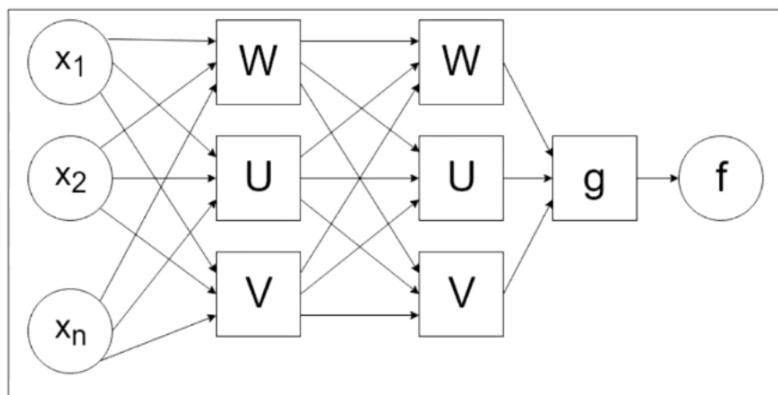
Существует правило дифференцирования сложных функций, где под сложной функцией понимается композиция двух других функций. Но термин «сложная функция» безмысленен, так как не доказано, что существуют простые функции, которые нельзя разложить в композицию других функций. Термин появился в результате неправильного перевода слова «составной» с польского, поэтому данные функции правильно называть составными. В английском языке правило называется «chain rule».

Будем называть функции сложными, когда они возвращают не скаляр, например, вектор или матрицу.

Если составную функцию представлять в виде композиции, то ее размер может экспоненциально расти. В примере слева потребуется меньше вычислений, чем справа:

1	<code>f(x1, x2 ... xn):</code>	1	<code>f=g(w(w(x1,x2 ... xn),</code>
2	<code>  a1 = w(x1, x2 ... xn)</code>	2	<code>  u(x1,x2 ... xn),</code>
3	<code>  a2 = u(x1, x2 ... xn)</code>	3	<code>  v(x1,x2 ... xn)),</code>
4	<code>  a3 = v(x1, x2 ... xn)</code>	4	<code>  u(w(x1,x2 ... xn),</code>
5		5	<code>  u(x1,x2 ... xn),</code>
6	<code>  b1 = w(a1, a2, a3)</code>	6	<code>  v(x1,x2 ... xn)),</code>
7	<code>  b2 = u(a1, a2, a3)</code>	7	<code>  v(w(x1,x2 ... xn),</code>
8	<code>  b3 = v(a1, a2, a3)</code>	8	<code>  u(x1,x2 ... xn),</code>
9		9	<code>  v(x1,x2 ... xn)))</code>
10	<code>  return g(b1, b2, b3)</code>		

Поэтому будем представлять составную функцию в виде графа вычислений и по нему находить производную



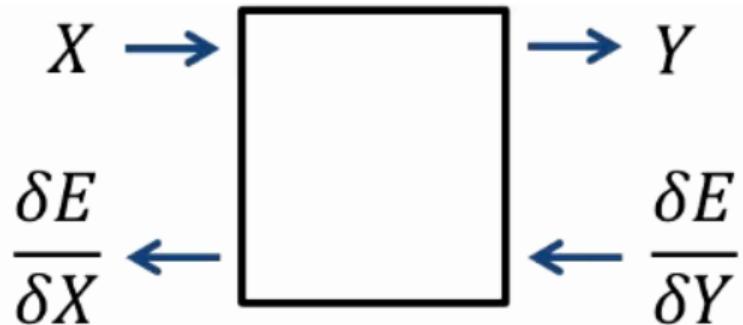
Статический граф — граф, который подстроен до вычисления функции (можем заранее оптимизировать вычисления по графу). Динамический — во время (более гибкий).

Автоматическое дифференцирование при помощи pytorch без знания что такое граф:

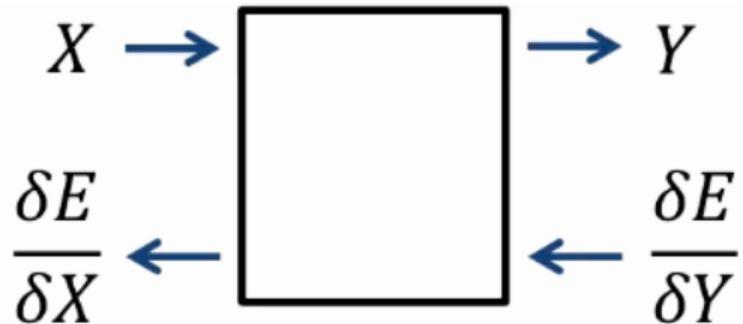
```
1 import torch
2 # Creating the graph
3 x = torch.tensor(1.0, requires_grad = True)
4 z = x ** 3
5 z.backward() #Computes the gradient
6 print(x.grad.data) #Prints '3' which is dz/dx
```

Вычисление функции по графу называется прямым проходом, а вычисление производной — обратным. Обратный проход следует после прямого. Обратный проход так называется, так как вычисление производной выполняется с конца функции.

Рассмотрим блок (обозначен прямоугольником) графа:



При прямом проходе по  $X$  получаем  $Y$ , при обратном — по производной целевой функции по  $Y$  получаем производную целевой функции по  $X$  (просто домножая производную уровнем выше на производную текущего уровня, согласно цепному правилу). Пример реализации:



Блок может запоминать  $X$  и  $Y$ . Для пересчета производной ему не нужно знать всю функцию.

Размерности всех входных X и dX, Y и dY совпадают

```

1  def mul(a,b):
2      def d_mul(dc):
3          return (b * dc, a * dc)
4      return a * b, d_mul

```

Блоки могут состоять из других блоков, например, два последовательных блока можно представить в виде одно блока или два параллельных — в виде одного блока, где входные и выходные данные являются кортежем соответствующих данных внутренних блоков.

$dX$  — сокращение для  $\frac{\delta E}{\delta X}$ , то есть производная функции ошибки  $E$  по вершине  $X$ .

$dX = df_{x \rightarrow Y}(dY)$  — пересчет производной из  $dY$  в  $dX$ .

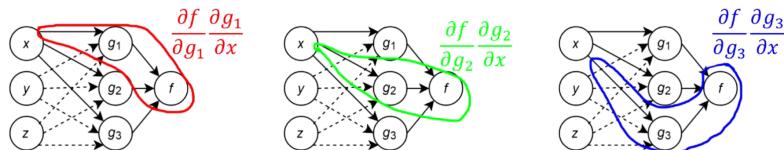
Если блок вычисляет скалярную функцию, то для пересчета производной используется цепное правило.

Если блок вычисляет сложную функцию, то мы можем представить, что блок вычисляет несколько скалярных составных функций.

Если вершина графа вычислений использовалась несколько раз, то ее производная будет суммой по всем использованием, что следует из обобщенного цепного правила:

$$\frac{\partial f(g_1(x), \dots, g_n(x))}{\partial x} = \sum_i \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial x}$$

Для сложной функции будут суммироваться не скаляры, а тензоры.



Алгоритм дифференцирования по графу вычислений:

- Вычисляем составную функцию Q, сохраняя граф вычислений  $G = (V, E)$ :  $(u, v) \in E$  (граф состоит из вершин и ребер, где каждое ребро представляется парой зависимых вершин).
- Для всех вершин  $v$  определим производную:  $\frac{\partial Q}{\partial v} = I[v = Q]$ .
- Для каждого ребра  $(u, v)$  в обратном порядке вычисления целевой функции: для простой функции  $\frac{\partial Q}{\partial u} += \frac{\partial Q}{\partial v} \cdot \frac{\partial u}{\partial v}$ , для сложной функции:  $\frac{\partial Q}{\partial u} += df_{u \rightarrow v}(\frac{\partial Q}{\partial v})$ . Оператор  $+=$  позволяет учиться случаю, когда вершина используется многократно.

Нахождение производной по графу вычислений является способом динамическо-

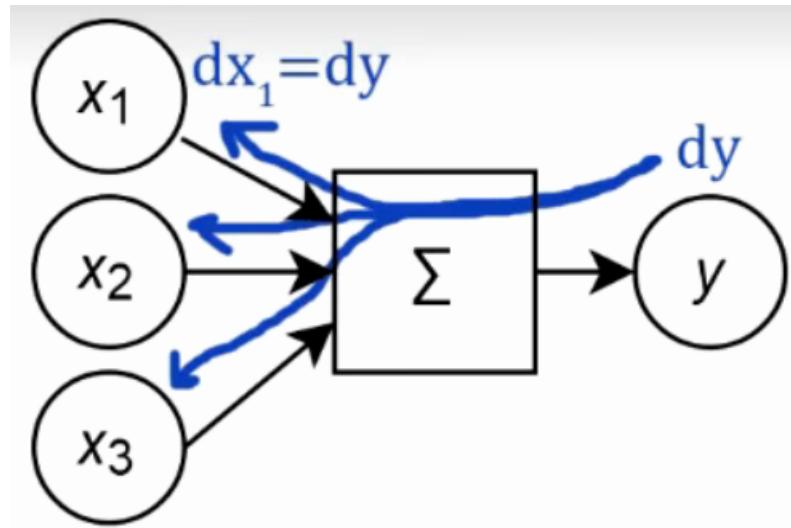
го программирования (решаются подзадачи исходной задачи, которые являются более простыми)

Пример дифференцирования простых функций:

Сумма:

$$y = \sum x_i$$

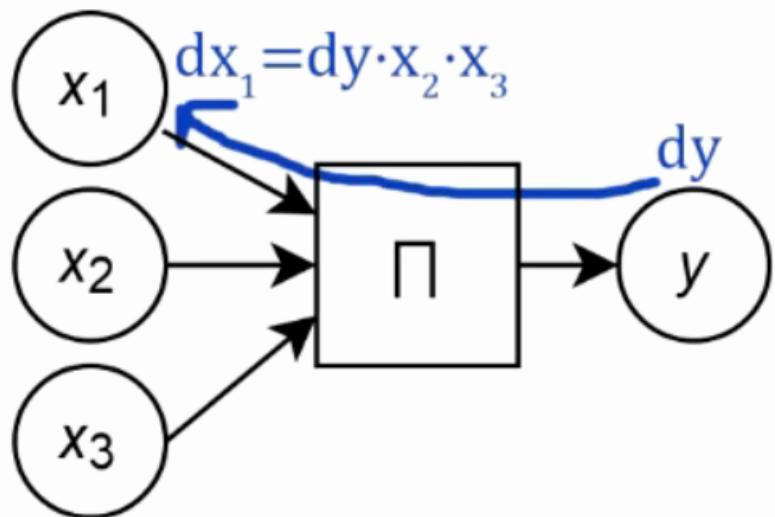
$$dx_i = dy$$



Произведение:

$$y = \prod x_i$$

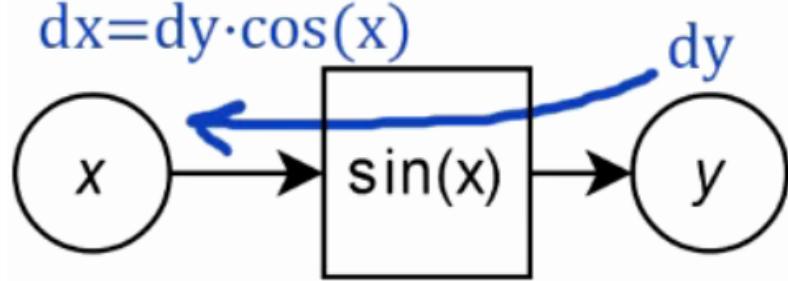
$$dx_i = dy \cdot \prod_{j \neq i} x_j$$



Применение функции:

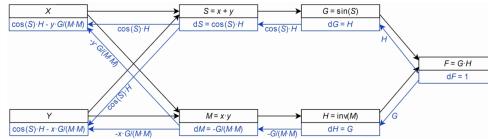
$$y = f(x)$$

$$dx = dy \cdot f'(x)$$

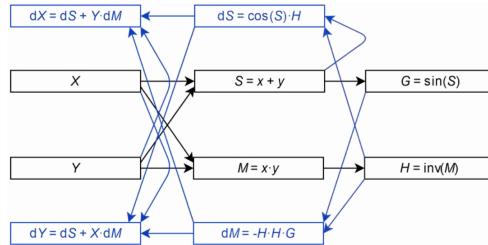


Рассмотрим пример для функции:  $F = \frac{\sin(x+y)}{x \cdot y}$ :

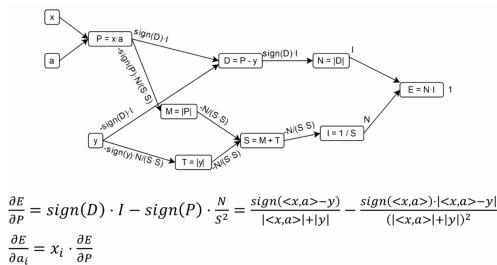
$\text{inv}(M)$  — инверсия функции  $M$  ( $\frac{1}{M}$ ).



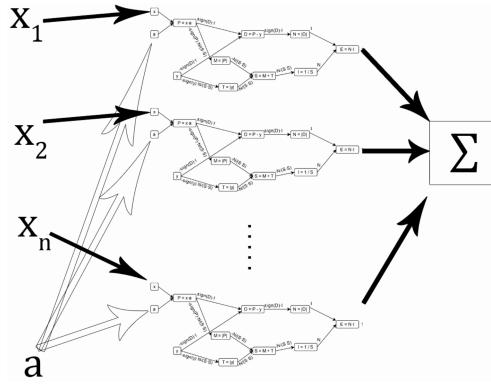
Для оптимизации можно отделить граф вычисления от графа производных и удалить лишние вершины:



Пример для функции SMAPE:  $E = \frac{|\langle x, a \rangle - y|}{|\langle x, a \rangle| + |y|}$



Если рассмотреть множество объектов, то график вычислений будет выглядеть так:



Блок сложной функции, которая вычисляет произведение матриц можно описать так:

$$\begin{array}{c}
 \text{Diagram showing the computation of } C_{1,i} = A_{1,i}B \\
 \text{Inputs: } A, dA \\
 \text{Outputs: } C_{1,i}, dC_{1,i} \\
 \text{Final Output: } C = AB, dC
 \end{array}$$

$$\frac{\partial}{\partial A_{i,:}} A_{i,:} B = \frac{\partial}{\partial A_{i,:}} (A_{i,:} B_{:,1}, \dots) = \frac{\partial}{\partial A_{i,:}} (\sum_k a_{k,:} b_{k,:}) = (B_{:,1}^T, \dots)$$

$$dA_{i,:} = (B_{:,1}^T, \dots) dC_{i,:} = B_{:,1}^T dC_1 + \dots = (b_{1,1} dC_1 + b_{1,2} dC_2 + \dots, \dots) = dC_{i,:} B^T$$

$$dA = dCB^T$$

Чтобы было легче искать производную, представим матричную функцию  $AB$  в виде нескольких вектор-функций  $A_{i,:}B$ . Для этого изобразим блок исходной функции в виде нескольких параллельных блоков. Умножение вектора-строки на матрицу  $B$  можно представить в виде вектора-стоки скалярных произведений строки  $A_i$  на столбец  $B_j$ . Скалярное произведение можно расписать в виде суммы. Производная по строке  $A_i$  будет строка  $(\frac{\partial A_i}{\partial A_{i,1}}, \dots)$ . В данном случае получаем строку, компоненты которой совпадают с компонентами столбца матрицы  $B$ . Чтобы найти производную функции ошибки по  $A_i$  воспользуемся цепным правилом и умножим найденную производную на производную уровнем выше  $dC_{i,:}$ . Получим скалярное произведение строки строк на строку. Покомпонентное сложение строк даст нам итоговую строку, где каждая компонента является скалярным произведением  $dC_{i,:}$  на строку  $B$ . В матричном виде строка умножается на столбец, поэтому транспортируем матрицу  $B$ , чтобы записать решение. Теперь обобщим функцию и получим итоговое решение для произведения матриц.

Если функция скалярная (результат один), то производная по вектору будет вектором такой же размерности частных производных одного и того же результата.

Если вектор-функция, производная по переменной, то мы по каждой компоненте ответа берем производную по переменной. В случае матричной функции подход тот же. Если имеется берем производную по вектору, для каждой компоненты (скаляра) берем производную по вектору.

Здесь вектора принимаются столбцами:

$$\frac{d}{dx} x^T A x = (A + A^T)x$$

$$\frac{d}{dx} x^T a = a$$

$$\frac{d}{dX} a^T X b = ab^T$$

Здесь дифференциал:

$$d(AXB) = A(dX)B$$

$$d(X^T) = (dX)^T$$

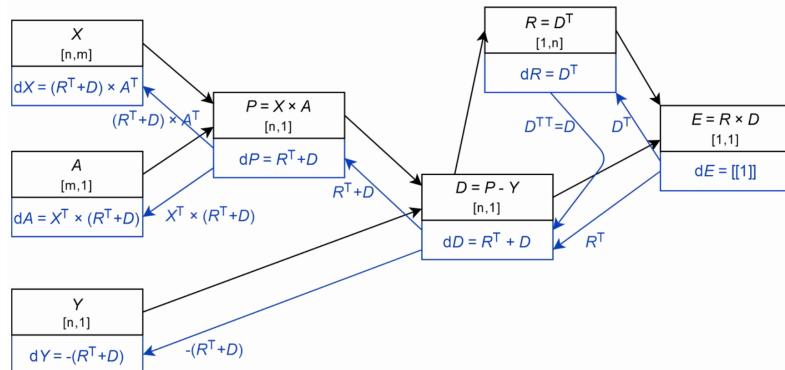
Произведение Адамара (Hadamard product) получается в результате покомпонентного умножения матриц (похоже на сложение):  $(A \circ B)_{i,j} = A_{i,j} \cdot B_{i,j}$ .

Скалярное произведение (dot product) двух строк можно записать в виде произведения матриц:  $\langle a, b \rangle = ab^T$ . Аналогично для столбцов:  $\langle a, b \rangle = a^T b$ . Скалярного произведения между строкой и столбцом быть не может, так как для проекции одного вектора на другой, они должны находиться в одном векторном пространстве.

В машинном обучении особое внимание уделяется матричным вычислениям, так как на аппаратном уровне имеются различные оптимизации, позволяющие ускорить вычисления, если они проводятся над матрицами. Программисту обычно не приходится думать, как задействовать эти оптимизации для своих вычислений, так как этим занимаются библиотеки, которые он использует.

Сумма квадратов отклонений (MSE) для линейной регрессии в матричном виде:

$$E = (X \times A - Y)^T \times (X \times A - Y)$$



### 7.3 Дифференцирование сложных функций

#### 7.4 Soft-Max и Soft-Arg-Max

#### 7.5 История нейронных сетей

#### 7.6 Функции активации

#### 7.7 Дропаут

#### 7.8 Дополнительные темы