# Деревья принятия решений

Сергей Николенко

Machine Learning — CS Club, весна 2008

# Outline

# Пример

Задача: выиграет ли «Зенит» свой следующий матч?

Параметры:

- выше ли находится соперник по турнирной таблице;
- дома ли играется матч;
- пропускает ли матч кто-либо из лидеров команды;
- идёт ли дождь.

Мы знаем об исходах нескольких матчей и хотим предсказать исход следующего матча, параметры которого нам ещё не встречались.

# Постановка задачи

Главная задача:

- Классификация данных
- Аппроксимация заданной булевской функции

То есть имеется *частично* заданная функция $f$, и мы хотим понять, как она работает на ещё не известных примерах.

# Постановка задачи

Дано:

- Атрибуты (параметры функции)
- Тестовые примеры ($f(0,0,1)$, $f(0,1,1)$, $f(1,1,0)$, $f(1,1,1)$)

Нужно:

- Продолжить функцию на другие значения атрибутов (найти $f(0,0,0)$)
- Сделать это красиво и экономично

# Дерево принятия решений

Дерево принятия решений — это дерево. На нём есть метки:

- В узлах, не являющиеся листьями: атрибуты, по которым различаются случаи
- В листьях: значения целевой функции
- На рёбрах: значения атрибута, из которого исходит ребро

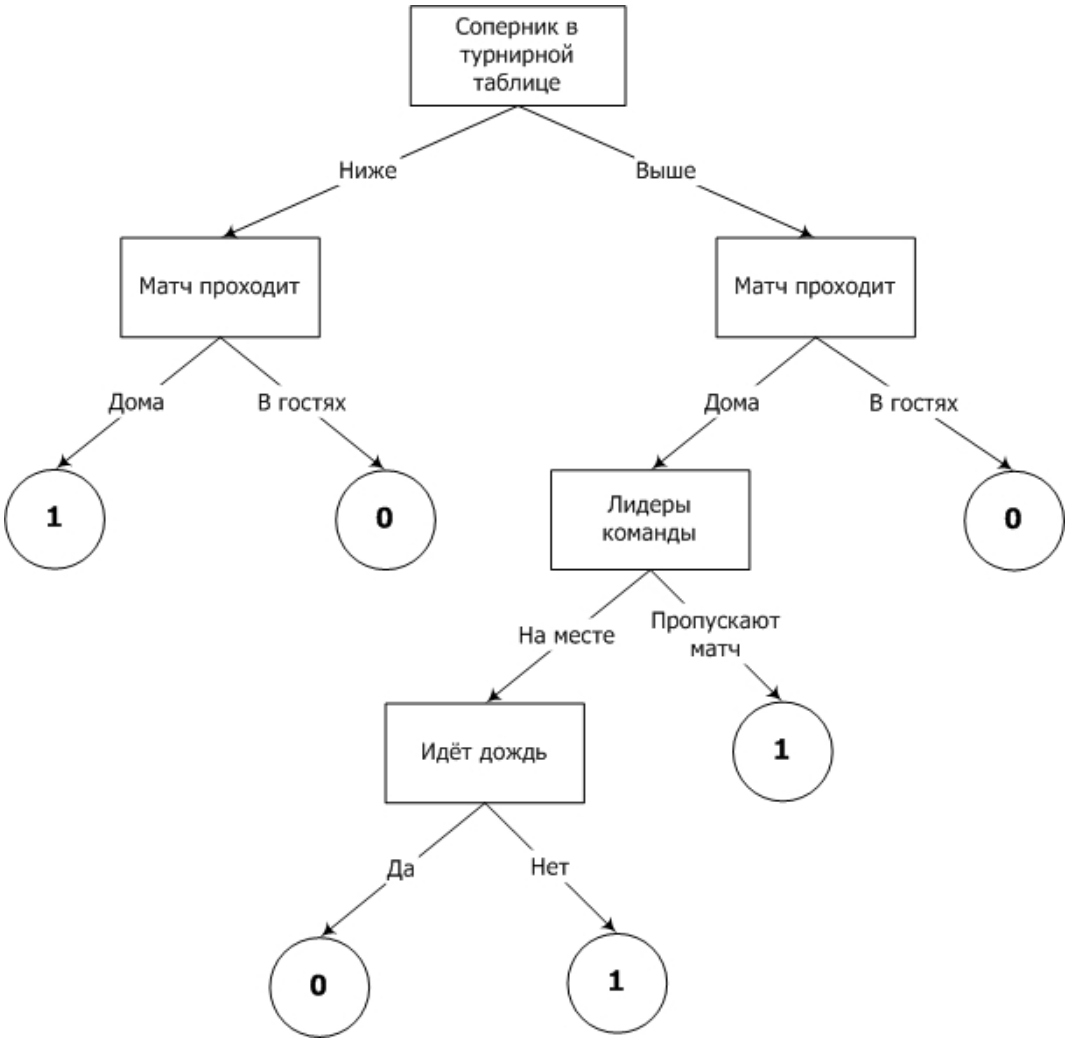
Чтобы классифицировать новый случай, нужно спуститься по дереву до листа и выдать соответствующее значение.

# Начальные данные

Таблица: Как играет «Зенит».

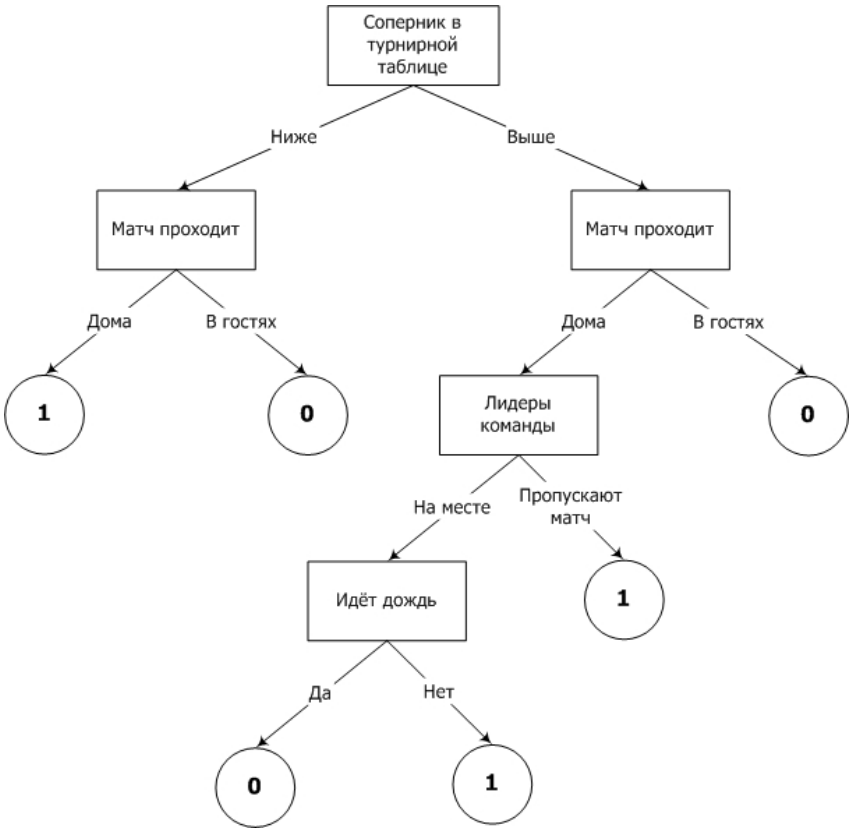
| Соперник | Играем | Лидеры | Дождь | Победа |
|---|---|---|---|---|
| Выше | Дома | На месте | Да | Нет |
| Выше | Дома | На месте | Нет | Да |
| Выше | Дома | Пропускают | Нет | Да |
| Ниже | Дома | Пропускают | Нет | Да |
| Ниже | В гостях | Пропускают | Нет | Нет |
| Ниже | Дома | Пропускают | Да | Да |
| Выше | В гостях | На месте | Да | Нет |
| Ниже | В гостях | На месте | Нет | *???* |

# Само дерево


Соперник в турнирной таблице
Ниже
Выше
Матч проходит
Матч проходит
Дома
В гостях
Дома
В гостях
1
0
Лидеры команды
0
На месте
Пропускают матч
Идёт дождь
1
Да
Нет
0
1

# Его использование

Как использовать:

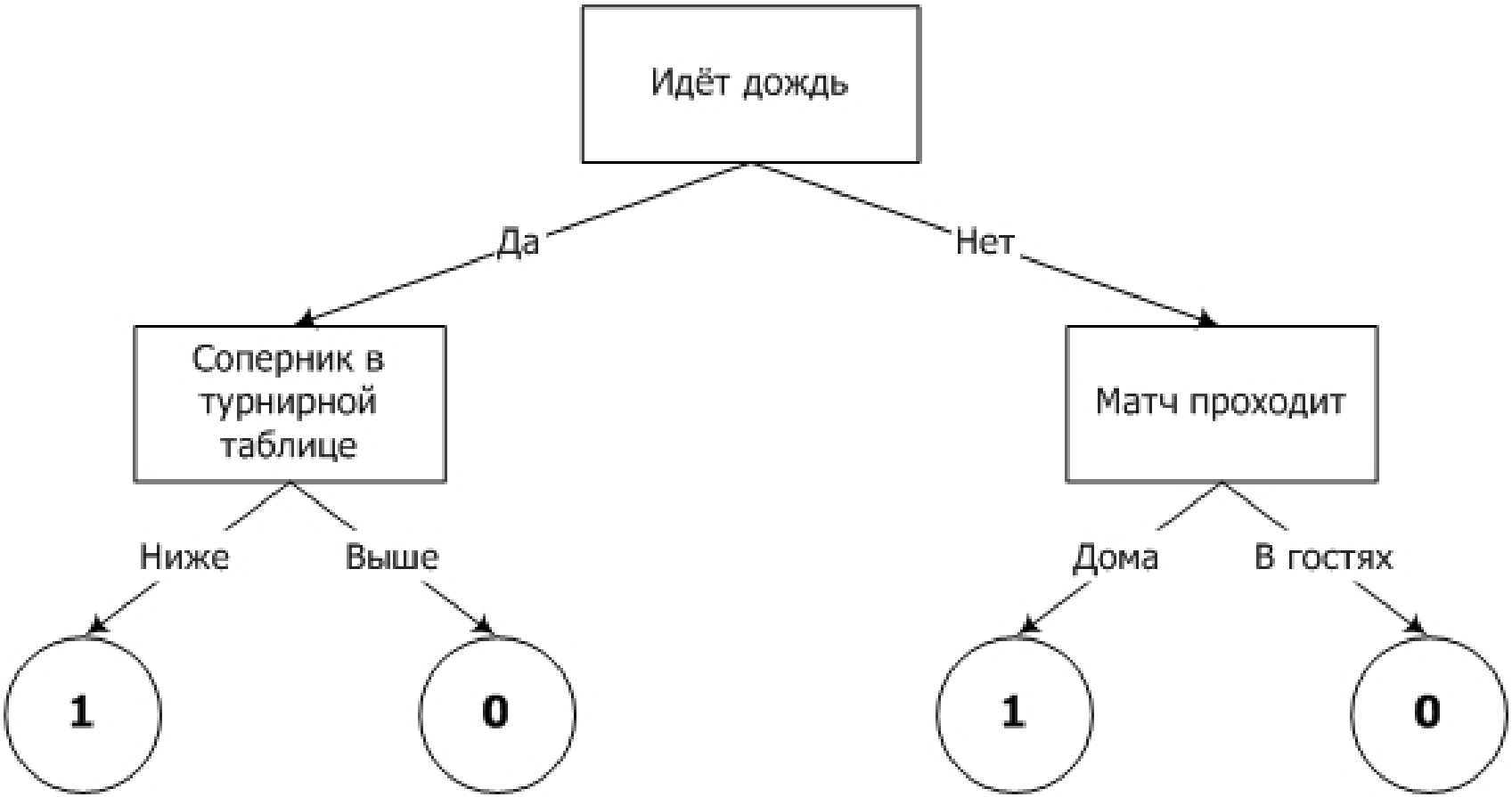



Соперник = Ниже
Играем = В гостях
Лидеры = На месте
Дождь = Нет
Победа = ???

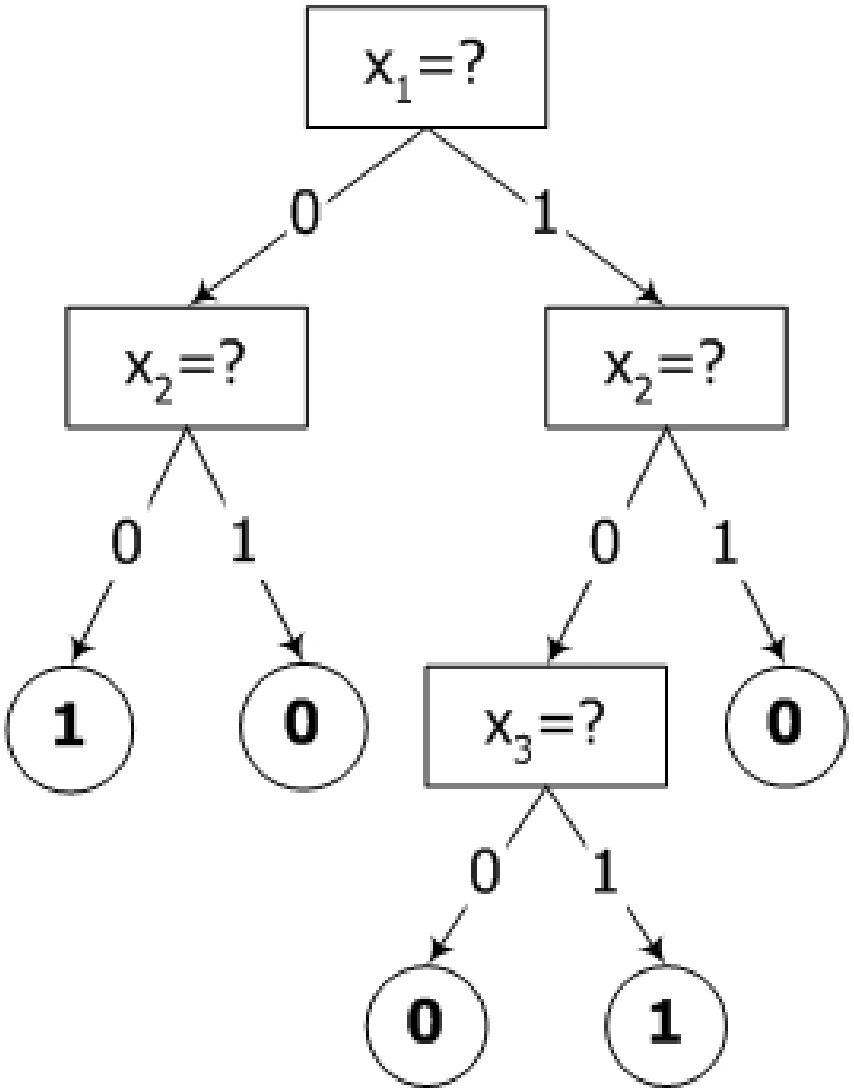
Спускаемся по дереву, выбирая нужные атрибуты, и получаем ответ: судя по нашему дереву, «Зенит» этот матч должен проиграть.

# Оптимальное дерево

Это большое дерево. А вот дерево для тех же самых данных, но куда меньше:



# Деревья и булевские функции




Из дерева принятия решений легко добыть булевскую функцию в ДНФ.
Например, дерево на рисунке соответствует функции:

$$f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2x_3.$$

# Упражнения

**Упражнение.** Нарисовать деревья принятия решений, соответствующие функциям:

1. $x \vee (y \wedge \bar{z})$;
2. $(x \wedge \bar{y}) \vee (y \wedge \bar{z} \wedge t)$;
3. $(x \vee y) \wedge (\bar{y} \vee z)$.

# Алгоритм построения

Как строить дерево:

- Выбираем очередной атрибут $Q$, помещаем его в корень
- Для всех его значений $i$:
  - Оставляем из тестовых примеров только те, у которых значение атрибута $Q$ равно $i$
  - Рекурсивно строим дерево в этом потомке
- Выдаём полученное дерево

# Алгоритм построения

Как строить дерево:

- Выбираем очередной атрибут $Q$, помещаем его в корень
- Для всех его значений $i$:
  - Оставляем из тестовых примеров только те, у которых значение атрибута $Q$ равно $i$
  - Рекурсивно строим дерево в этом потомке
- Выдаём полученное дерево

Главная проблема:

- Как выбирать новый атрибут?

# Энтропия

## Определение

*Предположим, что имеется множество $A$ из $n$ элементов, $m$ из которых обладают некоторым свойством $S$. Тогда энтропия множества $A$ по отношению к свойству $S$ — это*

$$H(A,S) = -\frac{m}{n}\log_2\frac{m}{n} - \frac{n-m}{n}\log_2\frac{n-m}{n}.$$

Энтропия зависит от пропорции, в которой разделяется множество. Чем «ровнее» поделили, тем больше энтропия.

# Энтропия

Если свойство $S$ не бинарное, а может принимать $s$ различных значений, каждое из которых реализуется в $m_i$ случаях, то

$$H(A, S) = -\sum_{i=1}^{s} \frac{m_i}{n} \log \frac{m_i}{n}.$$

Энтропия — это среднее количество битов, которые требуются, чтобы закодировать атрибут $S$ у элемента множества $A$. Если вероятность появления $S$ равна $1/2$, то энтропия равна 1, и нужен полноценный бит; а если $S$ появляется не равновероятно, то можно закодировать последовательность элементов $A$ более эффективно.

# Энтропия: пример

В нашем примере из 7 матчей «Зенит» три проиграл и четыре выиграл. Поэтому исходная энтропия

$$H(A, \text{Победа}) = -\frac{4}{7}\log_2\frac{4}{7} - \frac{3}{7}\log_2\frac{3}{7} \approx 0.9852.$$

# Прирост информации

Атрибут для классификации нужно выбирать так, чтобы после классификации энтропия (относительно целевой функции) стала как можно меньше.

## Определение

*Предположим, что множество $A$ элементов, характеризующихся свойством $S$, классифицировано посредством атрибута $Q$, имеющего $q$ возможных значений. Тогда прирост информации (information gain) определяется как*

$$\mathrm{Gain}(A, Q) = H(A, S) - \sum_{i=1}^{q} \frac{|A_i|}{|A|} H(A_i, S),$$

*где $A_i$ — множество элементов $A$, на которых атрибут $Q$ имеет значение $i$.*

# Прирост информации: пример

Теперь вычислим приросты информации для различных атрибутов:

$$\begin{aligned}\text{Gain}(A, \text{Соперник}) = H(A, \text{Победа}) - \frac{4}{7} H(A_{\text{выше}}, \text{Победа}) - \\ - \frac{3}{7} H(A_{\text{ниже}}, \text{Победа}) \approx \\ \approx 0.9852 - \frac{4}{7}\left(-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}\right) - \\ - \frac{3}{7}\left(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}\right) \approx 0.0202.\end{aligned}$$

Мы явно выбрали не слишком удачный атрибут для корня дерева...

# Прирост информации: пример

$$\mathrm{Gain}(A, \text{Играем}) \approx 0.4696.$$
$$\mathrm{Gain}(A, \text{Лидеры}) \approx 0.1281.$$
$$\mathrm{Gain}(A, \text{Дождь}) \approx 0.1281.$$

Прирост информации советует сначала классифицировать по тому, домашний ли матч или гостевой.

**Упражнение.** Дерево (проверьте) получится глубины 3. Как нужно модифицировать выбор атрибутов, чтобы получить дерево глубины 2, причём с меньшим количеством узлов, чем в приведённом выше?

# Outline

# Алгоритм ID3

$ID3(A, S, \mathcal{Q})$

- Создать корень дерева.
- Если $S$ выполняется на всех элементах $A$, поставить в корень метку 1 и выйти.
- Если $S$ не выполняется ни на одном элементе $A$, поставить в корень метку 0 и выйти.
- Если $Q = \emptyset$, то:
  - если $S$ выполняется на половине или большей части $A$, поставить в корень метку 1 и выйти;
  - если $S$ не выполняется на большей части $A$, поставить в корень метку 0 и выйти.

- Выбрать $Q \in \mathcal{Q}$, для которого $\mathrm{Gain}(A, Q)$ максимален.
- Поставить в корень метку $Q$.
- Для каждого значения $q$ атрибута $Q$:
  - добавить нового потомка корня и пометить соответствующее исходящее ребро меткой $q$;
  - если в $A$ нет случаев, для которых $Q$ принимает значение $q$ (т.е. $|A_q| = 0$), то пометить этого потомка в зависимости от того, на какой части $A$ выполняется $S$ (аналогично пункту 1);
  - иначе запустить $ID3(A_q, S, \mathcal{Q} \setminus \{Q\})$ и добавить его результат как поддерево с корнем в этом потомке.

# Проблема критерия прироста информации

Проблема: прирост информации выбирает атрибуты, у которых больше всего значений. Например, пусть в таблице игр были записаны ещё и даты матчей. Прирост информации:

$$\text{Gain}(A, \text{Дата}) = H(A, \text{Победа}) - \\ - \sum_{i=1}^{n} \frac{1}{n} H(A_{\text{Дата}=i}, \text{Победа}) = H(A, \text{Победа}),$$

потому что в каждой из веток только один случай, и энтропия в каждой ветке равна нулю.
Прирост информации — максимальный из возможных, но полученное дерево абсолютно бесполезно.

# Gain Ratio

Gain Ratio учитывает не только количество информации, требуемое для записи результата, но и количество информации, требуемое для разделения по текущему атрибуту. Поправка:

$$\mathrm{SplitInfo}(A, Q) = -\sum_{i=1}^{q} \frac{|A_q|}{|A|} \log_2 \frac{|A_q|}{|A|},$$

Сам критерий — максимизация величины

$$\mathrm{GainRatio}(A, Q) = \frac{\mathrm{Gain}(A, Q)}{\mathrm{SplitInfo}(A, Q)}.$$

# Gain Ratio: пример

У атрибута «Дата»

$$\mathrm{SplitInfo}(A, \text{Дата}) = -\sum_{i=1}^{7} \frac{1}{7} \log_2 \frac{1}{7} \approx 2.80735\ldots,$$

и Gain Ratio получается равным

$$\mathrm{GainRatio}(A, \text{Дата}) = \frac{\mathrm{Gain}(A, \text{Дата})}{\mathrm{SplitInfo}(A, \text{Дата})} \approx 0.350935\ldots$$

А для атрибута, показывающего, где проходит матч,

$$\mathrm{SplitInfo}(A, \text{Играем}) = -\frac{5}{7} \log_2 \frac{5}{7} - \frac{2}{7} \log_2 \frac{2}{7} \approx 0.86312\ldots,$$

и итоговый Gain Ratio получается

$$\mathrm{GainRatio}(A, \text{Играем}) = \frac{\mathrm{Gain}(A, \text{Играем})}{\mathrm{SplitInfo}(A, \text{Играем})} \approx 0.5452\ldots$$

# Индекс Гини

Для набора тестов $A$ и свойства $S$, имеющего $s$ значений, этот индекс вычисляется как

$$\mathrm{Gini}(A, S) = 1 - \sum_{i=1}^{s} \left( \frac{|A_i|}{|A|} \right)^2.$$

Соответственно, для набора тестов $A$, атрибута $Q$, имеющего $q$ значений, и целевого свойства $S$, имеющего $s$ значений, индекс вычисляется следующим образом:

$$\mathrm{Gini}(A, Q, S) = \mathrm{Gini}(A, S) - \sum_{j=1}^{q} \frac{|A_j|}{|A|} \mathrm{Gini}(A_j, S).$$

# Индекс Гини — экономика

- Кстати, индекс Гини пришёл из экономики.
- Коррадо Гини (Corrado Gini) в 1912 году предложил его как меру неравенства людей в экономике.
- Если построить кривую распределения дохода, то её индекс Гини будет тем больше, чем бóльшая часть дохода сосредоточена в руках меньшего количества людей.
- По данным ЦРУ, сейчас коэффициент Гини самый низкий в Швеции, самый высокий — в Намибии; Россия между Арменией и Сенегалом, проигрывает всей Европе, но значительно опережает США.

# Оверфиттинг

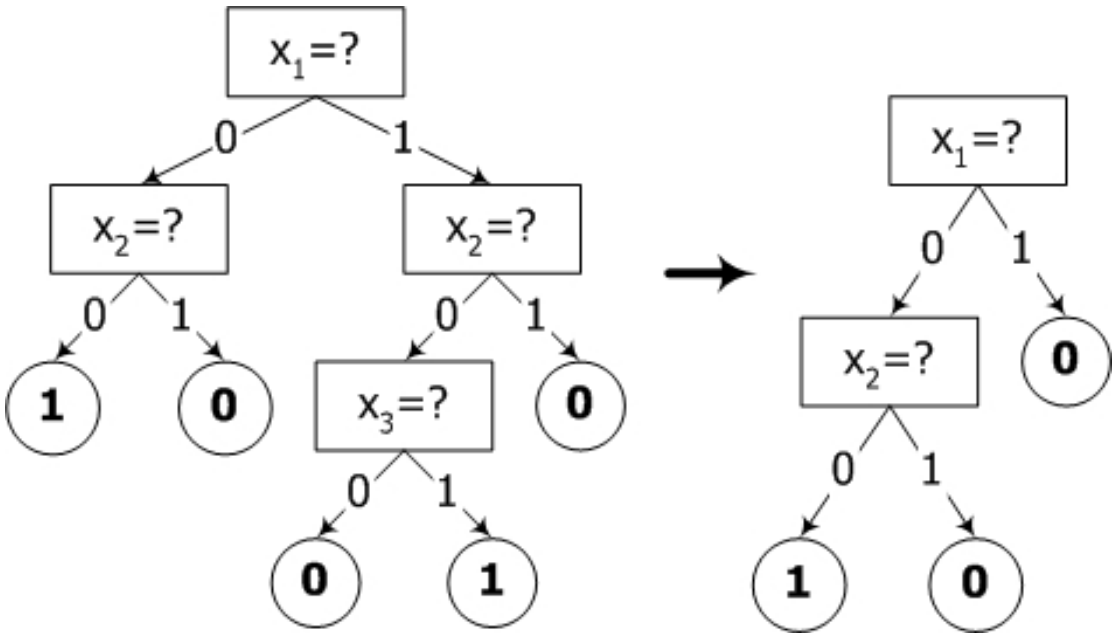
- ID3 удовлетворяет *всем* данным
- Но часть данных могут быть «шумом» или содержать ошибки
- Из-за этого дерево сильно растёт и хуже работает

# Оверфиттинг: пример

- Пусть «Зенит» дома выигрывает в 90% случаев, и ни от чего это больше не зависит.
- И среди исходных данных имеется одно домашнее поражение
- ID3 учтёт все «причины» и будет в дальнейшем предсказывать, что «Зенит» проиграет в аналогичных ситуациях
- Но на самом деле он будет выигрывать с вероятностью 90%

# Обрезание

Надо научиться обрезать лишние ветки. Обычно это делают так: ветку заменяют на значение, которое принимает большинство тестовых примеров в этой ветке.




Как выяснить, какие ветки обрезать?

# Обрезание: общий алгоритм

- Построим дерево по части исходных данных
- Тестировать будем на оставшейся части
- Для каждой вершины:
  - Обрежем ветку с корнем в этой вершине
  - Если обрезанное дерево будет лучше справляться с тестами, так и оставим обрезанную ветку, иначе вернём как было

# Outline

# Применяем теорему Байеса

- Вспомним прошлую лекцию.
- Нам нужно найти наиболее вероятную гипотезу $h \in H$ при условии данных $D$.
- Иными словами, нужно максимизировать $p(h|D)$.
- Что нам скажет теорема Байеса?

# Применяем теорему Байеса

- Вспомним прошлую лекцию.
- Нам нужно найти наиболее вероятную гипотезу $h \in H$ при условии данных $D$.
- Иными словами, нужно максимизировать $p(h|D)$.
- Что нам скажет теорема Байеса?
- $$p(h|D) = \frac{p(D|h)p(h)}{p(D)}.$$

# Применяем теорему Байеса

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}.$$

# Применяем теорему Байеса

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}.$$

- Итого нам нужно найти гипотезу

$$h = \operatorname{argmax}_{h \in H} p(h|D).$$

- Такая гипотеза называется *максимальной апостериорной гипотезой* (maximum a posteriori hypothesis, MAP).

# Применяем теорему Байеса

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}.$$

$$h = \operatorname{argmax}_{h \in H} p(h|D) =$$

$$= \operatorname{argmax}_{h \in H} \frac{p(D|h)p(h)}{p(D)} = \operatorname{argmax}_{h \in H} p(D|h)p(h),$$

потому что $p(D)$ от $h$ не зависит.

# Применяем теорему Байеса

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}.$$

Часто предполагают, что гипотезы изначально равновероятны: $p(h_i) = p(h_j)$. Тогда ещё проще:

$$h = \operatorname{argmax}_{h \in H} p(D|h).$$

# Алгоритм

- Для каждой гипотезы $h \in H$ вычислить апостериорную вероятность

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}.$$

- Выбрать гипотезу с максимальной апостериорной вероятностью:

$$h = \operatorname{argmax}_{h \in H} p(h|D).$$

# Как его применять: пример

- Нужно задать $p(h)$ и $p(D|h)$.
- Пусть выполняются следующие условия:
  - В $D$ нет шума (т.е. все тестовые примеры с правильными ответами).
  - Целевая функция $c$ лежит в $H$.
  - Нет априорных причин верить, что одна из гипотез более вероятна, чем другая.
- Именно эти условия мы сначала предполагали в нашей задаче классификации.

# Как его применять: пример

- Из третьего условия следует:

$$p(h) = \frac{1}{|H|} \text{ для всех } h \in H.$$

- $p(D|h)$ — вероятность наблюдать значения целевых функций $D = \langle d_1, \ldots, d_m \rangle$ для фиксированного набора входных данных $\langle x_1, \ldots, x_m \rangle$ при условии гипотезы $h$. Поскольку шума нет, $p(d_i|h) = 1$, если $d_i = h(x_i)$, и 0 в противном случае. Итого:

$$p(D|h) = \begin{cases} 1, & \text{если } d_i = h(x_i) \text{ для всех } d_i \in D, \\ 0, & \text{в противном случае.} \end{cases}$$

# Как его применять: пример

- Давайте подсчитаем вероятность $p(D)$. $\mathrm{Cons}(D)$ — множество гипотез $h \in H$, совместимых с $D$. Тогда:

$$p(D) = \sum_{h \in H} p(D|h)p(h) = \sum_{h \in \mathrm{Cons}(D)} \frac{1}{|H|} = \frac{|\mathrm{Cons}(d)|}{|H|}.$$

- Итого получается:

$$p(h|D) = \begin{cases} \frac{1}{|\mathrm{Cons}(d)|}, & \text{если } d_i = h(x_i) \text{ для всех } d_i \in D, \\ 0, & \text{в противном случае.} \end{cases}$$

- То есть каждая гипотеза, совместимая со всеми данными — максимальная апостериорная гипотеза.

# Задачи классификации

- Мы получили, что результат алгоритма ID3, например, является максимальной апостериорной гипотезой.
- Да и вообще любое дерево принятия решений, совместное со всеми данными, будет представлять собой MAP.

# Зачем нужен MAP для анализа алгоритмов

- Казалось бы, мы ничего нового не узнали: алгоритм выдаёт MAP, ну и что?
- Важно другое — важны *предположения*, в которых мы смогли это доказать.

# Зачем нужен MAP для анализа алгоритмов

- Казалось бы, мы ничего нового не узнали: алгоритм выдаёт MAP, ну и что?
- Важно другое — важны *предположения*, в которых мы смогли это доказать.
- Пусть выполняются следующие условия:
  - В $D$ нет шума (т.е. все тестовые примеры с правильными ответами).
  - Целевая функция $c$ лежит в $H$.
  - Нет априорных причин верить, что одна из гипотез более вероятна, чем другая.

# Зачем нужен MAP для анализа алгоритмов

- Казалось бы, мы ничего нового не узнали: алгоритм выдаёт MAP, ну и что?
- Важно другое — важны *предположения*, в которых мы смогли это доказать.
- Иначе говоря, мы поняли, что алгоритм обучения концептам Find-S работает оптимальным образом, если гипотезы априори равновероятны, и среди тестовых примеров нет шума. То же верно для ID3, например. А если гипотезы неравновероятны, можно сделать лучше.

# Зачем нужен MAP для анализа алгоритмов

- Казалось бы, мы ничего нового не узнали: алгоритм выдаёт MAP, ну и что?
- Важно другое — важны *предположения*, в которых мы смогли это доказать.
- Байесовский метод позволил установить *границы применимости* алгоритмов. Теперь мы знаем, когда их можно применять смело, а когда можно искать более хорошие алгоритмы. Это *очень* важно для AI.

# Зачем нужен MAP для анализа алгоритмов

- Казалось бы, мы ничего нового не узнали: алгоритм выдаёт MAP, ну и что?
- Важно другое — важны *предположения*, в которых мы смогли это доказать.
- Мы рассматривали «обрезания» и пытались найти дерево минимальной глубины. Тем самым мы изменяли *априорные вероятности*: предполагали, что дерево меньшей глубины будет более правдоподобно, чем дерево большей глубины.

# Зачем нужен MAP для анализа алгоритмов

- Казалось бы, мы ничего нового не узнали: алгоритм выдаёт MAP, ну и что?
- Важно другое — важны *предположения*, в которых мы смогли это доказать.
- Это, кстати, тоже можно обосновать математически...

# Бритва Оккама

- Обычно пишут так: «Entia non sunt multiplicanda praeter necessitatem» («Не следует умножать сущности без необходимости»).
- Сам Оккам так не писал, самое близкое — «Numquam ponenda est pluralitas sine necessitate» («Не следует утверждать многое без необходимости»)
- Выдвигалась и Джоном Дунсом Скотом, и Фомой Аквинским, и ещё Аристотелем; Оккам просто активно применял её.
- Базовый философский принцип — неужели его можно доказать математически?

# MAP и бритва Оккама

$$h_{MAP} = \operatorname{argmax}_{h \in H} p(D|h)p(h) =$$
$$= \operatorname{argmax}_{h \in H} \{\log_2 p(D|h) + \log_2 p(h)\} =$$
$$= \operatorname{argmin}_{h \in H} \{-\log_2 p(D|h) - \log_2 p(h)\}.$$

# MAP и бритва Оккама

- $h_{MAP} = \operatorname{argmin}_{h \in H} \{-\log_2 p(D|h) - \log_2 p(h)\}$.
- Но $(-\log_2 p(D|h))$ — это длина описания $D$ при условии использования гипотезы $h$ в оптимальном кодировании (по Шеннону), а $(-\log_2 p(h))$ — длина описания самой гипотезы $h$.

# MAP и бритва Оккама

- $h_{MAP} = \operatorname{argmin}_{h \in H} \{-\log_2 p(D|h) - \log_2 p(h)\}$.
- Иначе говоря, поиск MAP рекомендует не умножать сущности — использовать кратчайшую из возможных записей описываемой ситуации! Это ещё называется MDL — Minimum Description Length principle.

# Outline

# Лирическое отступление

- Поскольку мы с вами всё-таки в Computer Science Club, а не в Artificial Intelligence Club, было бы интересно увидеть связь между тем, чем мы занимаемся, и теоретической информатикой.
- Сейчас мы немножко отвлечёмся от задач искусственного интеллекта — но не от деревьев принятия решений!
- Мы увидим, как они используются в теории сложности алгоритмов.

# Деревья и функции

- Мы как-то уже отмечали, что каждое дерево принятия решений задаёт булевскую функцию.
- Можно пойти и обратно: каждую функцию можно описать деревом.
- Размер (глубина) *минимального* такого дерева — это хорошая мера сложности для функции.
- Сейчас мы её и рассмотрим.

# Определение

- Рассмотрим $f : \{0,1\}^n \to \{0,1\}$. Когда функция спускается по своему дереву, она проверяет биты входа $x = x_1 x_2 \dots x_n$ и выбирает направление дальнейшего спуска.
- Обозначим через $\mathrm{cost}(t, x)$ количество битов, за которые дерево $t$ на входе $x$ придёт к листу.

**Определение**

*Сложность дерева принятия решений функции $f$, $D(f)$, это*

$$\min_{t \in \mathcal{T}} \max_{x \in \{0,1\}^n} \mathrm{cost}(t, x),$$

*где $\mathcal{T}$ — множество деревьев принятия решений, задающих функцию $f$.*

# Определение

## Определение

*Сложность дерева принятия решений функции $f$, $D(f)$, это*

$$\min_{t \in \mathcal{T}} \max_{x \in \{0,1\}^n} \text{cost}(t, x),$$

*где $\mathcal{T}$ — множество деревьев принятия решений, задающих функцию $f$.*

- $D(f)$ — это максимальная глубина самого эффективного дерева принятия решений функции $f$.

# Пример

- Рассмотрим функцию *связности графа*: по данному графу $G$ определить, связный он или нет.
- Как доказать, что у неё большая $D(f)$?
- На самом деле $D(f) = \binom{n}{2}$, где $n$ — количество вершин в графе. То есть любое дерево для какого-то графа должно исследовать *все* рёбра.

# Пример

- Рассмотрим это как игру.
- Мы построили какое-нибудь дерево, а противник строит граф, для которого в этом дереве обязательно будет длинный путь вниз.
- Если он сможет построить такой граф, что путь будет длины $\binom{n}{2}$, это даст нам нужную оценку на $D(f)$.

# Пример

- Стратегия противника проста: когда мы спрашиваем о ребре $e_i$, противник отвечает «нет» всегда, когда это не делает граф автоматически несвязным (т.е. жадный алгоритм — не добавлять рёбра, пока это возможно).
- Обозначим через $Y_i$ рёбра, про которые противник ответил «да», и через $E_i$ — ещё не исследованные рёбра.
- Тогда получается, что противник поддерживает такой инвариант: на каждом шаге $i \leq \binom{n}{2}$ $Y_i$ — несвязный лес, а $Y_i \cup E_i$ связен.
- Отсюда и следует, что нужно будет спросить про каждое ребро.

# Пример II

- Другой пример, попроще: функция OR. $f(x_1, \ldots, x_n) = \bigvee_{i=1}^{n} x_i$.
- Здесь противник будет на первые $(n-1)$ запросов отвечать 0, и мы до последнего не узнаем значение $\vee$.
- Отсюда следует, что у функции OR сложность $D(f) = n$.

# От P к NP и coNP

- $D(f)$ — это, грубо говоря, P мира деревьев принятия решений.
- Сейчас мы рассмотрим, так сказать, NP и coNP.
- И докажем, что в контексте decision trees $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$.

# Сложность с сертификатом

## Определение

*Для функции* $f : \{0,1\}^n \to \{0,1\}$ *и входа* $x$ *такого, что* $f(x) = 0$, *0-сертификатом для* $x$ *является последовательность битов* $x$, *которой достаточно для того, чтобы доказать, что* $f(x) = 0$. *Аналогично, 1-сертификат для такого* $x$, *что* $f(x) = 1$, — *это последовательность битов* $x$, *доказывающая, что* $f(x) = 1$.

## Определение

*Сложность с сертификатом* $C(f)$ — *это*

$$C(f) = \max_x \{\text{длина минимального 0- или 1-сертификата для } x\}.$$

# Пример

- Если $f$ определяет связность заданного графа, то 0-сертификат должен содержать все возможные рёбра некоторого сечения графа (чтобы доказать, что их там нет).
- А 1-сертификат — это рёбра некоторого остовного дерева.
- Т.е. размер 1-сертификата не превышает $n-1$, а размер 0-сертификата не превышает (и иногда равен) $(n/2)^2$.
- Значит, $C(f) = n^2/4$.

# О P, NP и coNP

- Грубо говоря, задачи, у которых есть короткий 1-сертификат — это аналог NP.
- А те, у которых есть короткий 0-сертификат — аналог coNP.
- А вот их пересечение (множество задач с небольшой $C(f)$) в точности равно аналогу P, т.е. задачам с небольшой $D(f)$.

# Связь $D(f)$ и $C(f)$

**Теорема**

$$D(f) \leq C(f)^2.$$

- Докажем это. Рассмотрим множества $S_0$ и $S_1$ минимальных 0- и 1-сертификатов для функции $f$.
- Обозначим $k = C(f)$, т.е. в каждом $s \in S_0 \cup S_1$ не больше $k$ битов.

# Связь $D(f)$ и $C(f)$

**Теорема**

$$D(f) \leq C(f)^2.$$

- Заметим, что каждый 0-сертификат обязан пересекаться с некоторым 1-сертификатом, причём в пересечении должен быть хоть один различающийся бит.
- Иначе можно было бы построить вход, у которого есть и 0-, и 1-сертификат.

# Связь $D(f)$ и $C(f)$

**Теорема**

$$D(f) \leq C(f)^2.$$

- Мы построим дерево принятия решений, которое вычислит $f$ за $\leq k^2$ запросов.
- На каждом шаге выберем некоторый $c_0 \in S_0$.
- Запросим из него все биты.

# Связь $D(f)$ и $C(f)$

**Теорема**

$$D(f) \leq C(f)^2.$$

- Если все биты подходят под 0-сертификат, выдаём 0.
- В противном случае обрежем множество 1-сертификатов. Каждый из них должен пересекать $c_0$, т.е. у каждого $c_1 \in S_1$ мы уже проверили по одному биту.
- Если бит не подходит, выбросим этот $c_1$; если подходит, выбросим этот бит из $c_1$.

# Связь $D(f)$ и $C(f)$

**Теорема**

$$D(f) \leq C(f)^2.$$

- Таким образом, на каждом шаге мы запрашиваем $k$ битов и обрезаем все 1-сертификаты на 1 бит.
- Но длина 1-сертификатов не превышает $k$.
- Значит, за $k^2$ запросов процесс остановится.

# Обозначения

- Начнём с того, что вспомним (или изучим) лемму Яо (Yao's Lemma).
- Лемма Яо — один из ключевых инструментов в вероятностном анализе алгоритмов.
- Она немедленно следует из теоремы о минимаксе из теории игр, но мы даже это доказательство рассматривать не будем.

# Обозначения

- Рассмотрим набор входов $\mathcal{X}$ и набор алгоритмов $\mathcal{A}$ (оба конечные), которые решают некоторую вычислительную задачу на этих входах.
- Будем, как и раньше, обозначать $\mathrm{cost}(A, x)$ «цену» алгоритма $A \in \mathcal{A}$ на входе $x \in \mathcal{X}$.
- Вероятностный алгоритм можно рассмотреть либо как алгоритм со случайным входом, либо как распределение на множестве алгоритмов.
- Мы выберем второй подход: вероятностный алгоритм $\mathcal{R}$ — это распределение $\mathcal{R}$ на $\mathcal{A}$.
- Его «цена» — это, конечно, $\mathbf{E}_{A \in \mathcal{R}}[\mathrm{cost}(A, x)]$.

# Randomized vs. distributional complexity

## Определение

*Randomized complexity вычислительной задачи — это*

$$\min_{\mathcal{R}} \max_{x \in \mathcal{X}} \text{cost}(\mathcal{R}, x).$$

## Определение

*Distributional complexity вычислительной задачи — это*

$$\max_{\mathcal{D}} \min_{A \in \mathcal{A}} \text{cost}(A, D),$$

*где $\mathcal{D}$ — некоторое распределение на входах, а* $\text{cost}(A, D) = \mathbf{E}_{x \in \mathcal{D}}[cost(A, x)]$.

# Лемма Яо

**Теорема**

*Randomize complexity задачи равна distributional complexity.*

# Вероятностные деревья принятия решений

- Давайте введём вероятностные деревья принятия решений.
- Мы рассмотрим $\mathcal{P}$ — распределение на множестве $\mathcal{T}$ деревьев, вычисляющих ту или иную функцию.
- Тогда для входа $x$ можно определить

$$c(\mathcal{P}, x) = \sum_{t \in \mathcal{T}} \mathcal{P}(t)\mathrm{cost}(t, x),$$

ожидаемое количество запросов дерева из $\mathcal{T}$, взятого по $\mathcal{P}$, на входе $x$.

# Randomized DT complexity

## Определение

*Randomized decision tree complexity* $\mathcal{R}(f)$ *функции* $f$ *— это*

$$\mathcal{R}(f) = \min_{\mathcal{P}} \max_{x} c(\mathcal{P}, x).$$

- Очевидно, $\mathcal{R}(f) \geq C(f)$, потому что $C(f)$ — минимум $\text{cost}(t, x)$ из всех $t$, а $\mathcal{R}(f)$ — среднее.

# Distributional DT complexity

- Для распределения $\mathcal{D}$ на входах можно определить

$$d(A, \mathcal{D}) = \sum_x \mathcal{D}(x)\text{cost}(A, x) = \mathbf{E}_{x \in \mathcal{D}} [\text{cost}(A, x)] .$$

**Определение**

*Distributional decision tree complexity $\Delta(f)$ функции $f$ — это*

$$\Delta(f) = \max_{\mathcal{D}} \min_{A} d(A, \mathcal{D}).$$

# Лемма Яо для DT

**Теорема**

$$\mathcal{R}(f) = \Delta(f).$$

**Доказательство.**

Лемма Яо. □

## Спасибо за внимание!

- Lecture notes и слайды будут появляться на моей homepage:

  `http://logic.pdmi.ras.ru/~sergey/index.php?page=teaching`
- Присылайте любые замечания, решения упражнений, новые численные примеры и прочее по адресам: `sergey@logic.pdmi.ras.ru`, `snikolenko@gmail.com`
- Заходите в ЖЖ  **smartnik**.