

АККУЛЯ 3



Deep Learning School

Решающие деревья и композиции алгоритмов

План лекции



- Решающие деревья
 - устройство дерева
 - анализ
- Композиции алгоритмов
 - бэггинг
 - стекинг
 - бустинг
- Градиентный бустинг
 - алгоритм обучения
 - преимущества и недостатки

Решающие деревья



Алгоритмы машинного обучения

- Алгоритм ближайших соседей
- Линейные алгоритмы

В жизни мы принимаем решения не так!

Примеры решающих правил

- Если в анкете указан домашний телефон и зарплата клиента $> \$1000$ и размер кредита $< \$3000$, то кредит выдать
- Если возраст пациента > 60 и пациент ранее перенёс инфаркт, то операцию не делать

Логические алгоритмы

Логический алгоритм — алгоритм, использующий логические закономерности в данных.

Решающие деревья

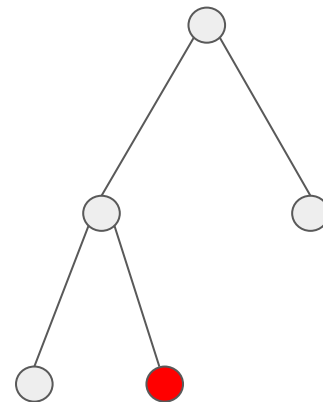
- В каждой вершине дерева находится вопрос
- В зависимости от ответа на вопрос, алгоритм направляется в нужную ветвь дерева
- Листы дерева соответствуют решению алгоритма

Решающие деревья



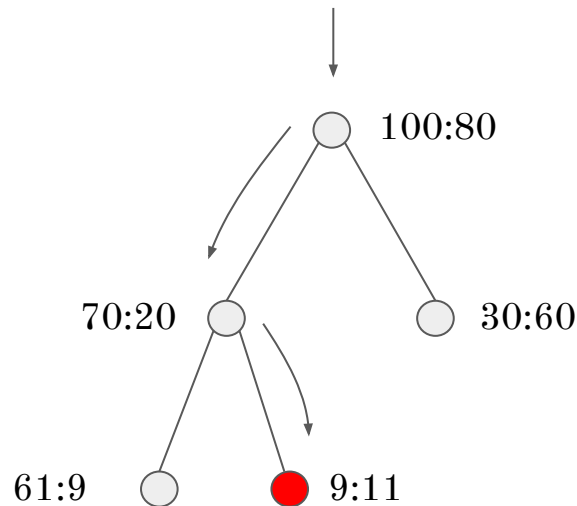
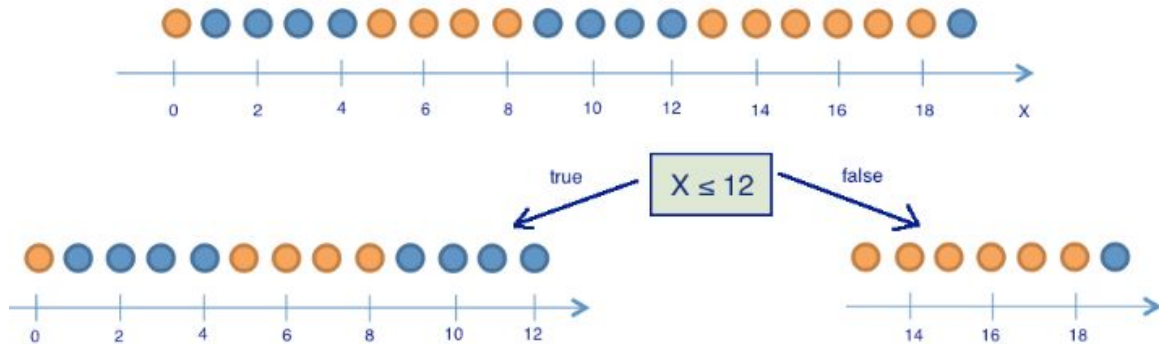
Обучение решающего дерева

- Находимся в красной вершине



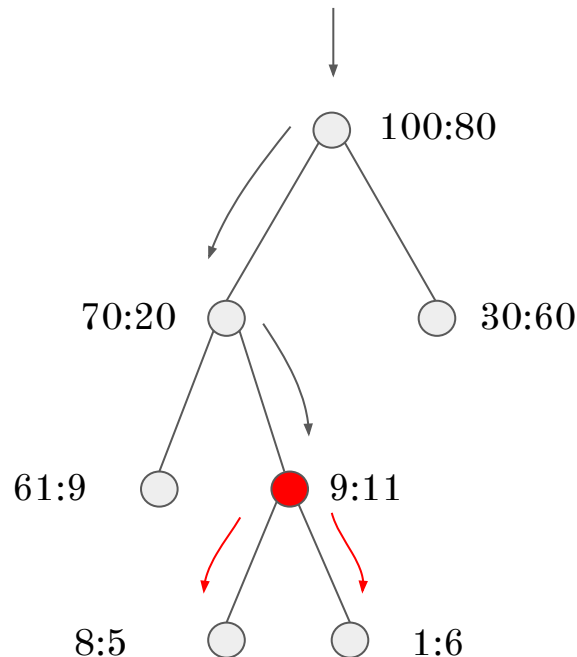
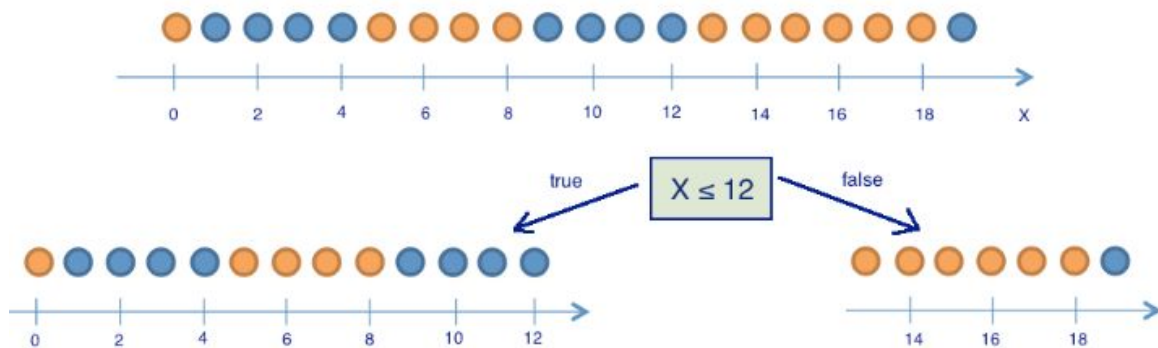
Обучение решающего дерева

- Находимся в красной вершине
- До красной вершины дошла часть объектов обучающей выборки



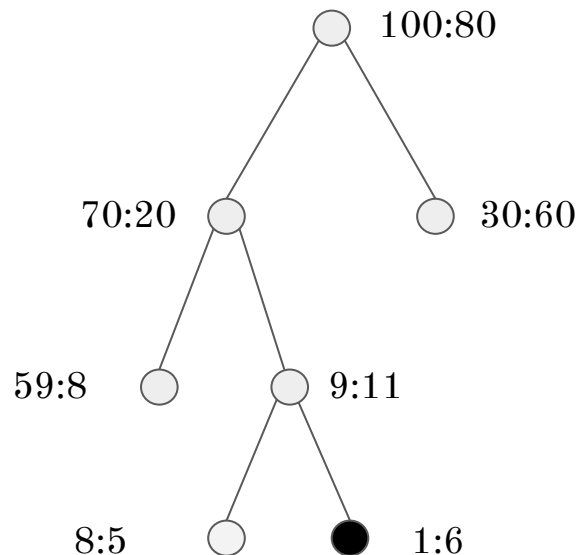
Обучение решающего дерева

- Находимся в красной вершине
- До красной вершины дошла часть объектов обучающей выборки
- Находим решающее правило так, чтобы объекты, дошедшие до красной вершины, хорошо разделялись по искомым классам



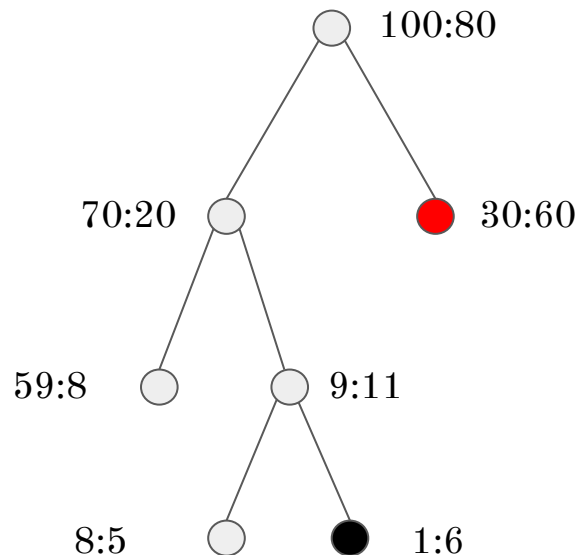
Обучение решающего дерева

- Находимся в красной вершине
- До красной вершины дошла часть объектов обучающей выборки
- Находим решающее правило так, чтобы объекты, дошедшие до красной вершины, хорошо разделялись по искомым классам
- Одна из нижних вершин стала терминальной



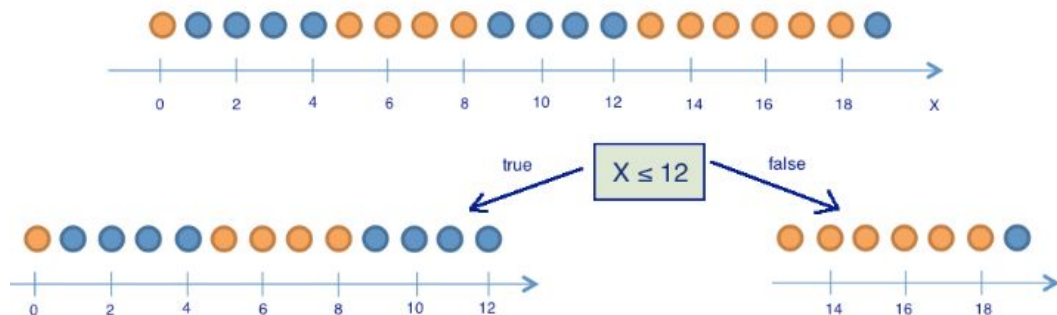
Обучение решающего дерева

- Находимся в красной вершине
- До красной вершины дошла часть объектов обучающей выборки
- Находим решающее правило так, чтобы объекты, дошедшие до красной вершины, хорошо разделялись по искомым классам
- Одна из нижних вершин стала терминальной
- Повторяем с другой вершиной



Как выбрать критерий ветвления?

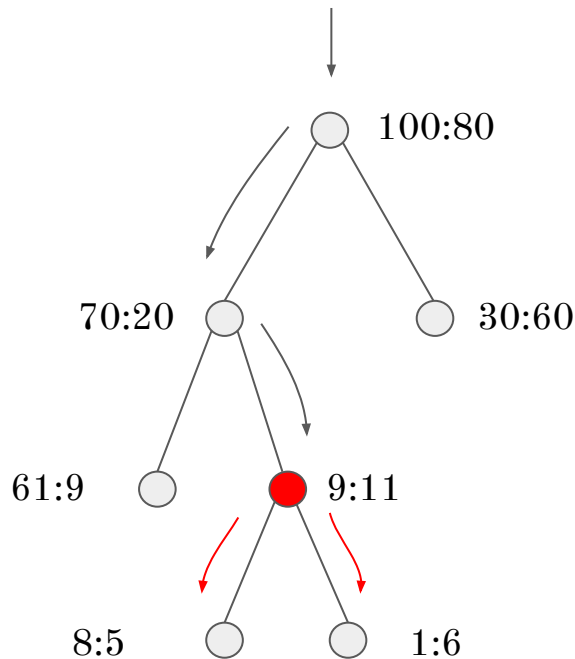
$$Q = 20, p_Q = \frac{9}{20}$$



$$L = 13, p_L = \frac{8}{13}$$

$$R = 7, p_R = \frac{1}{7}$$

$$\frac{L}{Q} H(p_L) + \frac{R}{Q} H(p_R) \rightarrow \min$$



Как выбрать критерий ветвления?

$$\frac{L}{Q} H(p_L) + \frac{R}{Q} H(p_R) \rightarrow \min$$

Возможные функции $H(q)$:

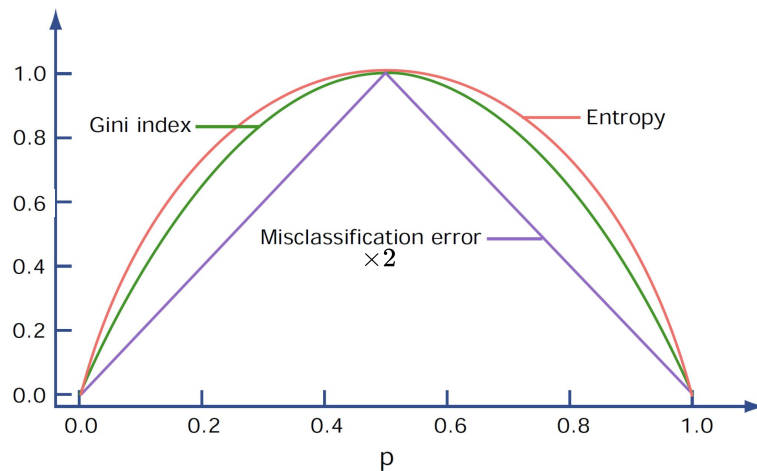
- Энтропия:

$$H(q) = -q \log q - (1 - q) \log(1 - q)$$

- Индекс Джини:

$$H(q) = 4q(1 - q)$$

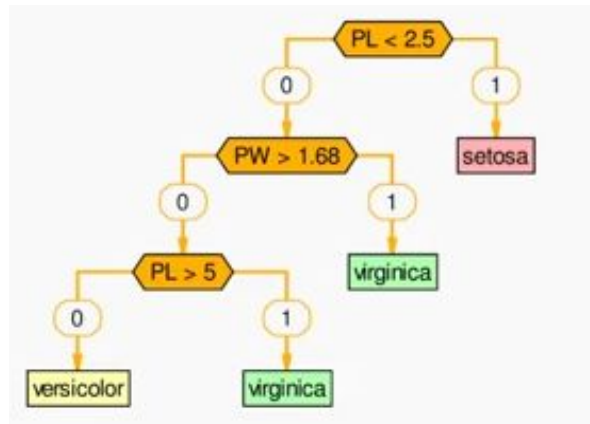
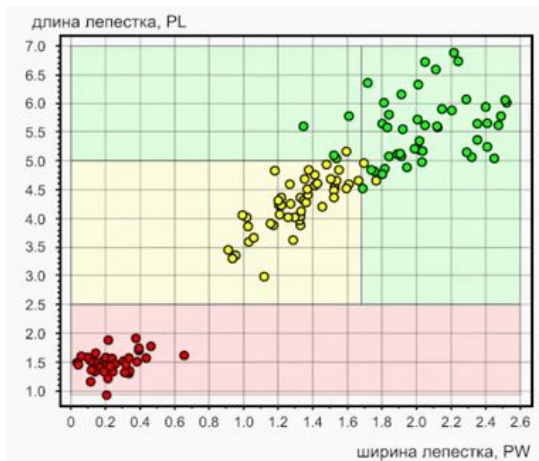
Затем решаем оптимизационную задачу
для поиска правила



Решающее дерево для Ирисов Фишера

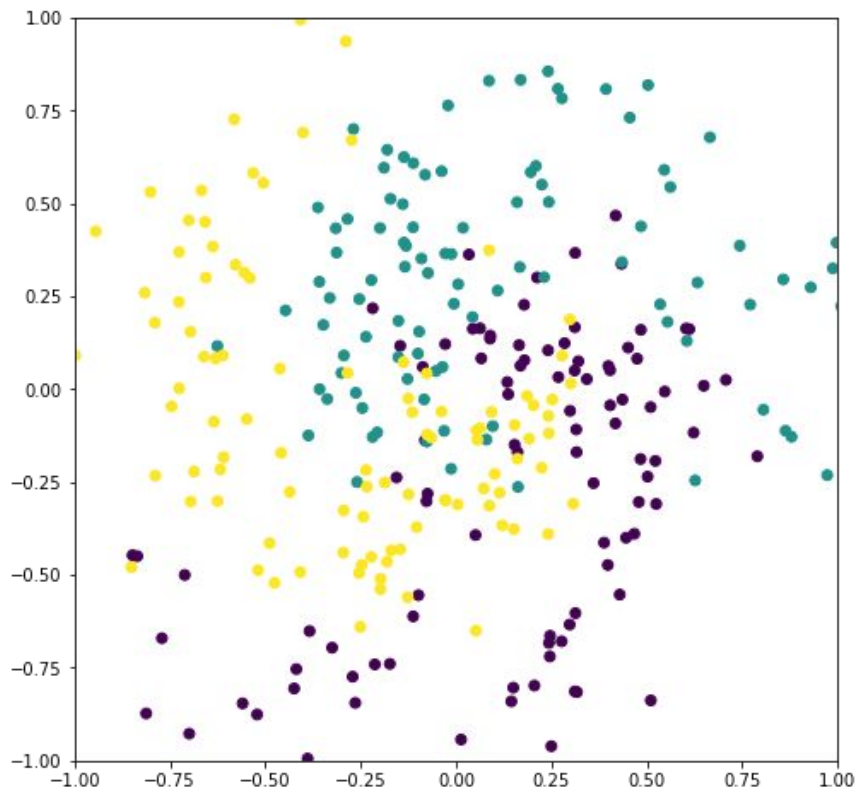
Задача Фишера о классификации ирисов на три класса.

В выборке по 50 объектов каждого класса, у каждого объекта 4 признака

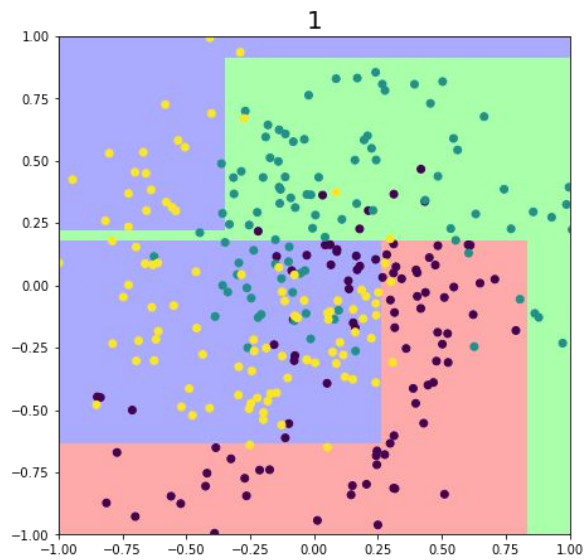


В осях двух самых информативных признаков два класса разделились без ошибок, на третьем — три ошибки.

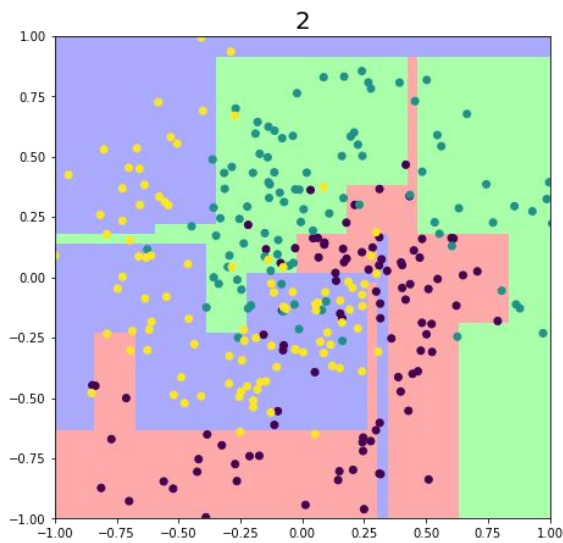
Недообучение и переобучение



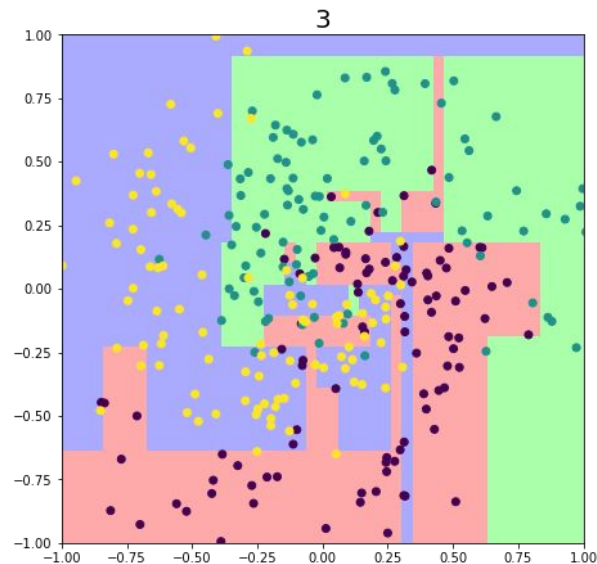
Недообучение и переобучение



depth = 3



depth = 7



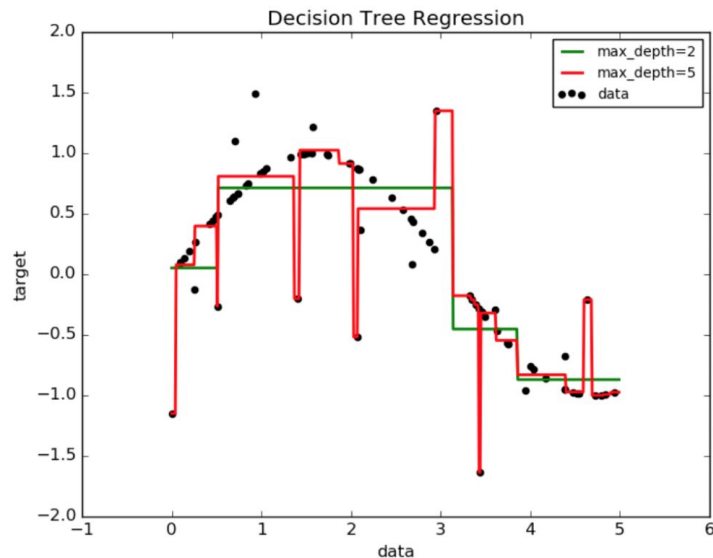
depth = 12

Параметры решающего дерева

- Критерий ветвления ('gini', 'entropy')
- Максимальная глубина
- Минимальный размер листа
- Стратегия сплита

Решающее дерево для регрессии

- В листах дерева вместо классов объектов находятся действительные числа
- Количество ответов решающего дерева ограничено количеством листовых вершин



Резюме: решающее дерево

Преимущества:

- простая идея и алгоритм обучения
- интерпретируемое
- возможна регуляризация
- обработка пропущенных значений

Недостатки:

- переобучается
- проблемы с регрессией
- сильно меняется в зависимости от параметров

Композиции алгоритмов



Алгоритмы машинного обучения

- Метод ближайших соседей
- Линейные алгоритмы
- Решающее дерево

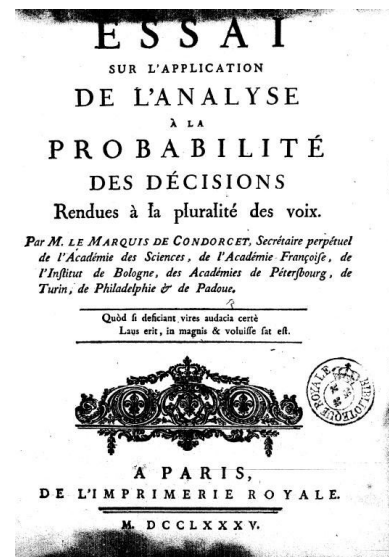
Алгоритмы машинного обучения

- Метод ближайших соседей
- Линейные алгоритмы
- Решающее дерево

Идея: построение композиции алгоритмов

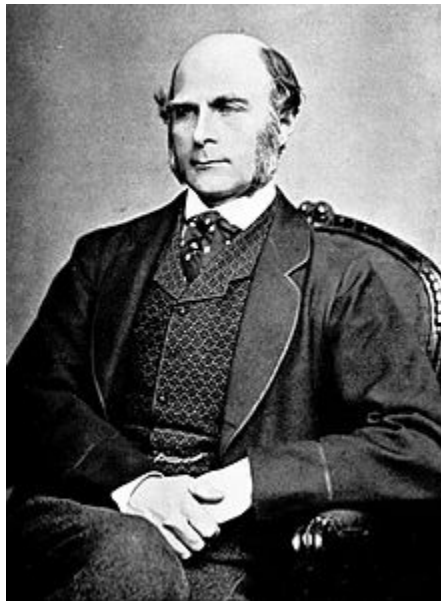
Принцип Кондорсе

- Если вероятность правильного решения члена жюри больше 0.5, то вероятность правильного решения присяжных в целом возрастает с увеличением количества членов жюри и стремится к единице.
- Если же вероятность быть правым меньше 0.5, то вероятность принятия правильного решения присяжными монотонно уменьшается и стремится к нулю с увеличением количества присяжных.



принцип Кондорсе, 1784

Фрэнсис Гальтон



Эксперимент Гальтона

- Собралось около 800 человек, которые попытались угадать вес быка на ярмарке. Бык весил 1198 фунтов.
- Ни один крестьянин не угадал точный вес быка
- Среднее предсказание оказалось равным 1197 фунтов.

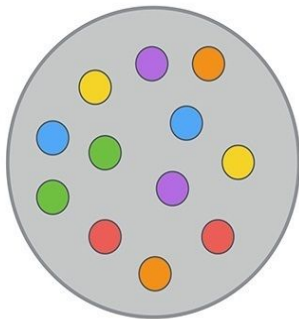


Метод простого голосования

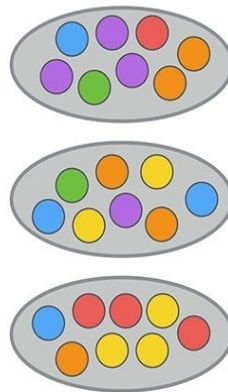
- a_1, a_2, \dots, a_n — несколько обученных алгоритмов
- Классификация: относим x к классу, за который проголосовало *большинство* из $a_1(x), a_2(x), \dots, a_n(x)$
- Регрессия: ответом является среднее значение $a_1(x), a_2(x), \dots, a_n(x)$

Бутстреп

изначальная выборка



бутстрепные выборки



алгоритмы

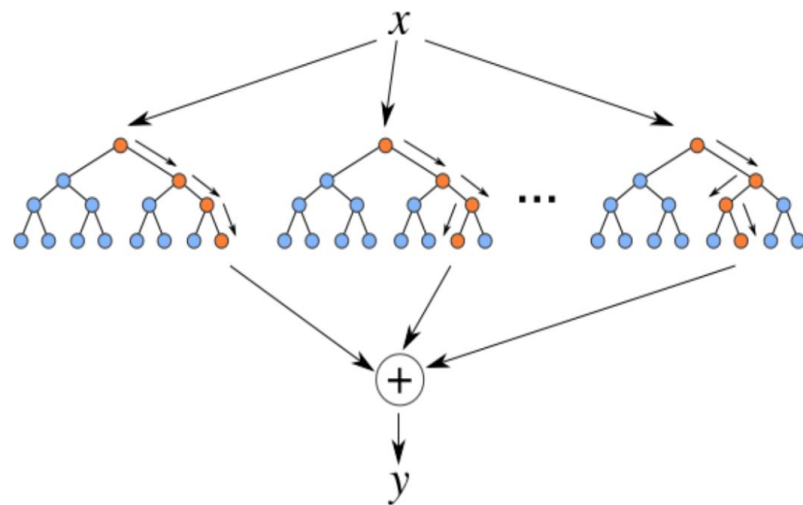
a_1

a_2

a_3

Бэггинг

- Бэггинг (bagging: bootstrap aggregation) — принцип построения композиции, основанный на простом голосовании
- 100 деревьев
- Бутстрепная выборка для каждого дерева
- Финальное решение принимается простым голосованием

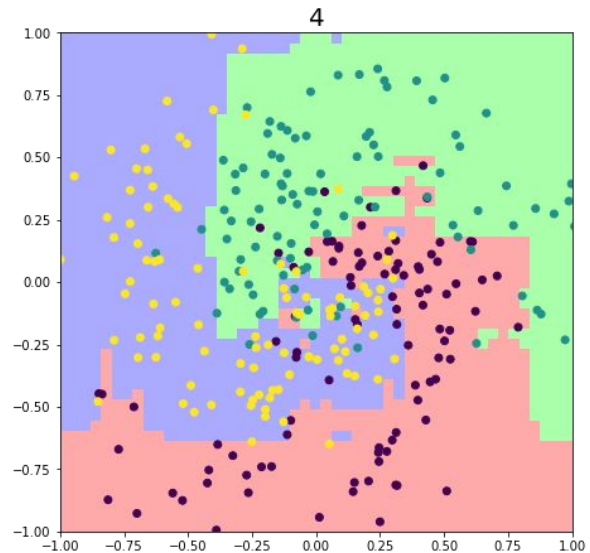
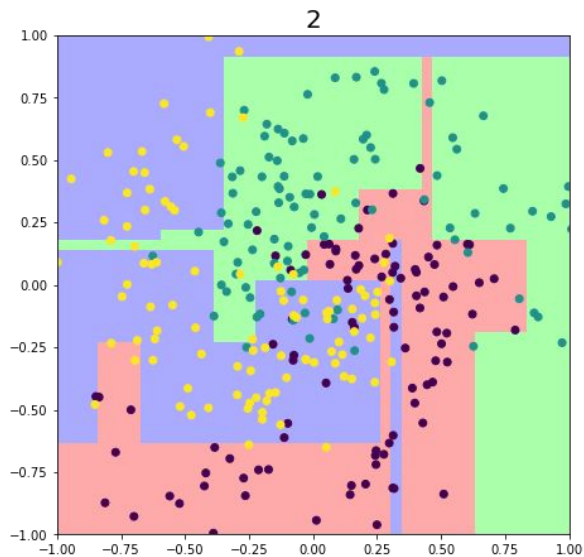


Случайный лес

Случайный лес — бэггинг над
решающими деревьями



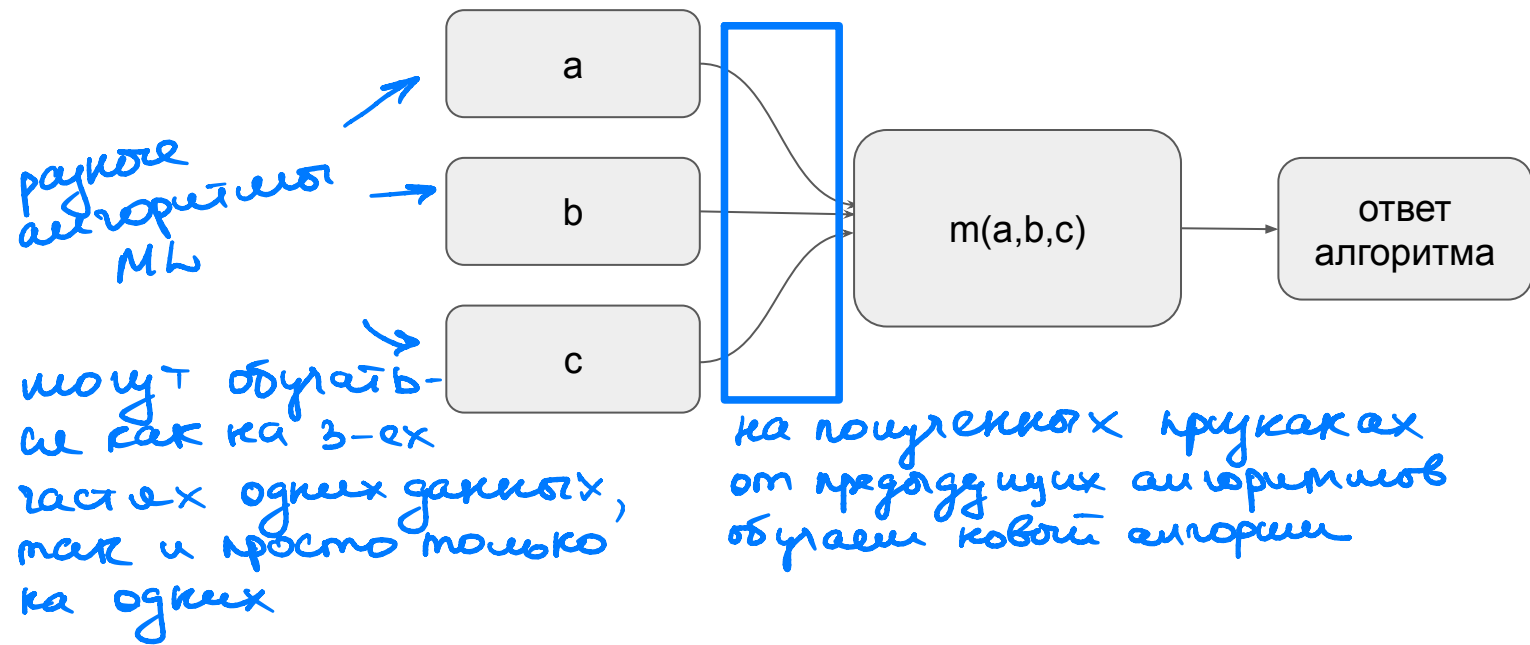
Случайный лес и решающее дерево



Недостатки случайного леса

- Слишком долгое и громоздкое вычисление
- В индустрии стараются обойтись без композиций
- Тем не менее, вычисления можно проводить параллельно

Стекинг



Бустинг

- Строим алгоритмы последовательно
- Каждый следующий алгоритм компенсирует ошибку всех предыдущих
- Принимаем решение взвешенным голосованием: *a-агрегация*

$$a(x) = c_1 a_1(x) + c_2 a_2(x) + \dots + c_n a_n(x)$$

- Сильный метод бустинга — градиентный бустинг над решающими деревьями
- Отличается от бустинга тем, что каждый последующий алгоритм учитывает ошибку предыдущего

Резюме

- Бэггинг, стекинг и бустинг используют принцип композиции
- Бэггинг принимает решение простым голосованием
- Стекинг обучает метаалгоритм над разноплановыми алгоритмами
- Бустинг строит базовые модели, компенсирующие ошибки предыдущих

Градиентный бустинг



Градиентный бустинг

Градиентный бустинг — эффективный способ построения композиции решающих деревьев

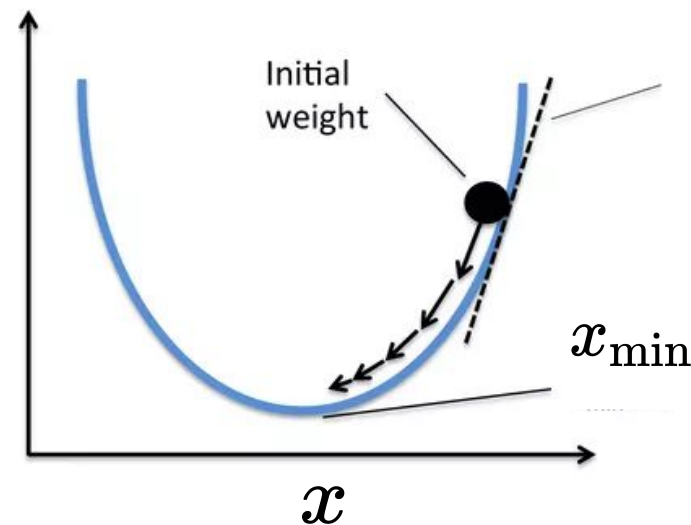
- Деревья строятся последовательно
- Каждое следующее дерево стремится компенсировать ошибку уже построенных
- Решение затем принимается взвешенным голосованием:

$$a(x) = c_1 a_1(x) + c_2 a_2(x) + \dots + c_n a_n(x)$$

Напоминание: градиентный спуск

- Минимизируем функцию $f(x)$
- Выбираем x_0 — начальное приближение
- Пусть x_n — текущая найденная точка
- Вычисляем градиент (производную) $f'(x_n)$
- Вычисляем следующее приближение:

$$x_{n+1} = x_n - \alpha f'(x_n)$$



Постановка задачи

Имеем:

- a_1, a_2, \dots, a_n — обучающая выборка
- $y = \{y^1, y^2, \dots, y^\ell\}$ — целевая переменная (действительное число)

Цель: построить алгоритм $a(x)$, оптимизирующий функцию потерь

$$Q(y, a) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(y^i, a(x^i))$$

Пример: квадратичная функция потерь

$$\mathcal{L}(y, a) = (a - y)^2$$

Построение композиции

пример.

$$c_k = \frac{1}{\sqrt{k}}$$

- Пусть уже построены a_1, a_2, \dots, a_k , а также константы c_1, c_2, \dots, c_k
- Хотим построить a_{k+1}
- Запишем функцию потерь для одного объекта

$$\mathcal{L}(y^i, a(x^i)) = \underbrace{\mathcal{L}(y^i, c_1 a_1(x^i) + \dots + c_k a_k(x^i))}_{\text{построено}} + \underbrace{c_{k+1} a_{k+1}(x^i)}_{\text{нужно построить}}$$

$$f\left(t + \Delta t\right) \rightarrow \min_t$$

$$\Delta t = -\alpha f'(t)$$

$$a_{k+1}(x^i) := -\mathcal{L}'_a(y^i, a)|_{a=c_1 a_1(x^i) + \dots + c_k a_k(x^i)}$$

Построение очередного дерева

Для одного объекта:

$$a_{k+1}(x^i) := \underbrace{-\mathcal{L}'_a(y^i, a)|_{a=c_1 a_1(x^i) + \dots + c_k a_k(x^i)}}_{\text{новая целевая переменная}}$$

Для нескольких объектов: решаем новую задачу машинного обучения, где $k+1$ -ое дерево “настраивается” по направлению *антиградиента функции потерь*

Константа в бустинге выбирается так же, как и в градиентном спуске

Градиентный бустинг для MSE

Антиградиент функции потерь для MSE:

$$\begin{aligned} -\mathcal{L}'_a(y^i, a)|_{a=c_1a_1(x^i)+\dots+c_ka_k(x^i)} &= \\ &= -2(a - y^i)|_{a=c_1a_1(x^i)+\dots+c_ka_k(x^i)} = \\ &= -2(c_1a_1(x^i) + \dots + c_ka_k(x^i) - y^i) \end{aligned}$$

Вывод. В случае MSE алгоритм градиентного бустинга настраивается на *разность ответа и текущего приближения*:

$$a_{k+1}(x^i) = -2(c_1a_1(x^i) + \dots + c_ka_k(x^i) - y^i)$$

Градиентный бустинг

- Базовые алгоритмы должны быть простыми и быстро обучаемыми
- Часто в качестве базовых алгоритмов используются решающие деревья
- Реализации градиентного бустинга над решающими деревьями:

dmlc
XGBoost

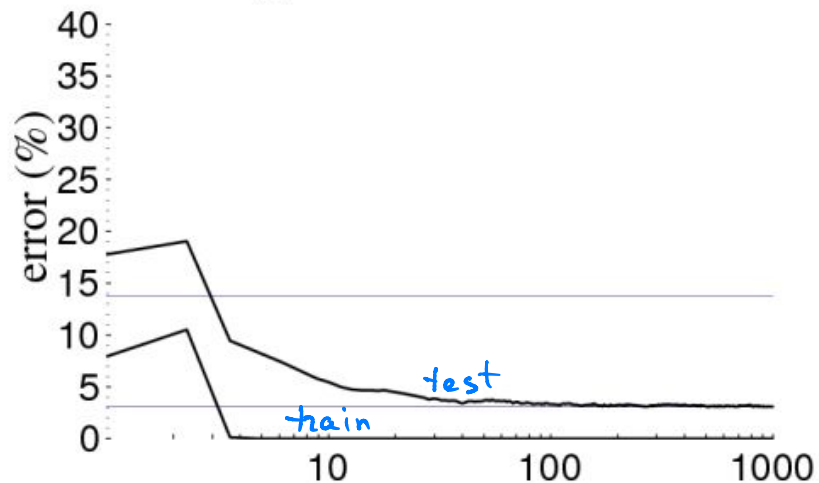
light
GBM



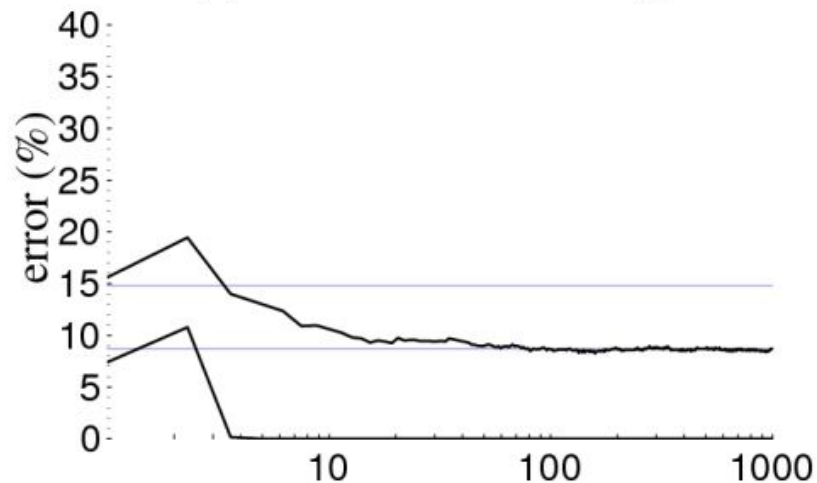
Yandex
CatBoost

Бустинг: качество от итерации

Задача UCI:letter



Задача UCI:satimage



Бустинг: преимущества и недостатки

- Преимущества:
 - Позволяет очень точно восстанавливать искомую функцию
 - Почти не переобучается
 - По сравнению со случайным лесом, способен компенсировать систематическую ошибку каждого из базовых алгоритмов, а не накапливает их
 - Работает с произвольной функцией потерь в задаче классификации, регрессии, ранжирования, ...
- Недостатки:
 - Медленный
 - Плохо интерпретируемый
 - Переобучение на выбросах при избыточном количестве алгоритмов
 - Нужна довольно большая обучающая выборка

The End

