УНИВЕРСИТЕТ
ИННОПОЛИС

Ансамбли моделей

Воробьёва Мария

- maria.vorobyova.ser@gmail.com
- @SparrowMaria

План лекции

1) Бустинг. Виды

2) Градиентный бустинг

Повторение

Бэггинг. Финальный алгоритм формировался как усреднение по всем алгоритмам.

Все алгоритмы были равнозначны

$$a(x) = \frac{1}{T} \sum_{i=1}^{T} a_i(x)$$

Минусы такого подхода:

- все алгоритмы должны быть независимы
- все взвешивается одним и тем же коэффициентов

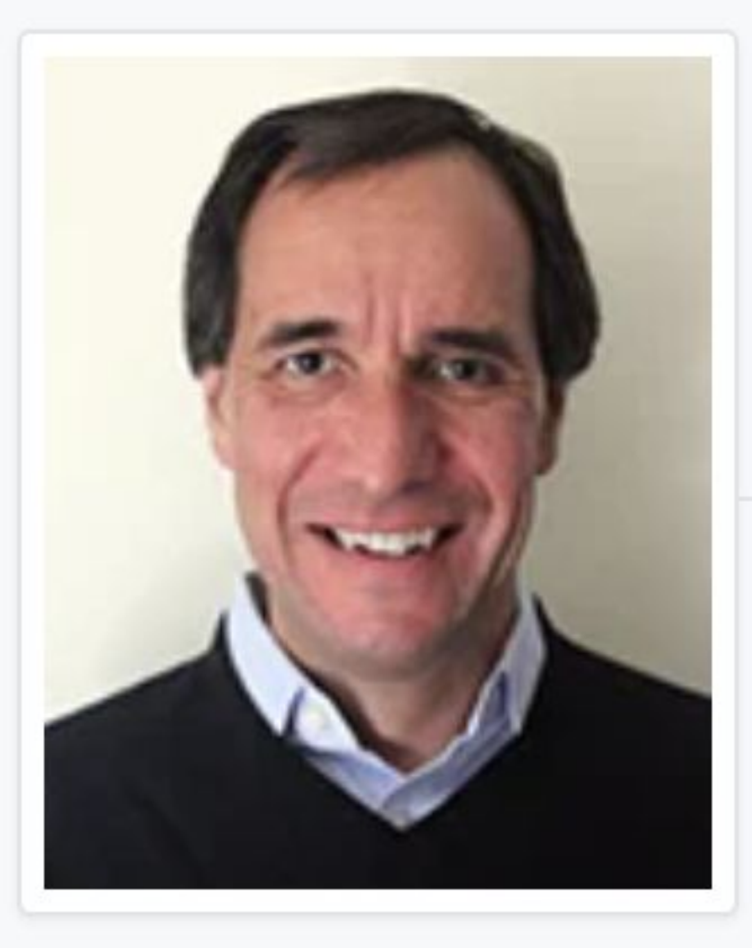
Как появился бустинг?

В 1995 году Йоав Фройнд (Yoav Freund) и Роберт Шапир (Robert Schapire) предложили более

общую схему для композиции алгоритмов – с разными весами

$a(x) = \sum_{i=1}^{T} w_i a_i(x)$, где w_i - это веса, $a_i(x)$ - это базовые алгоритмы

Yoav Freund



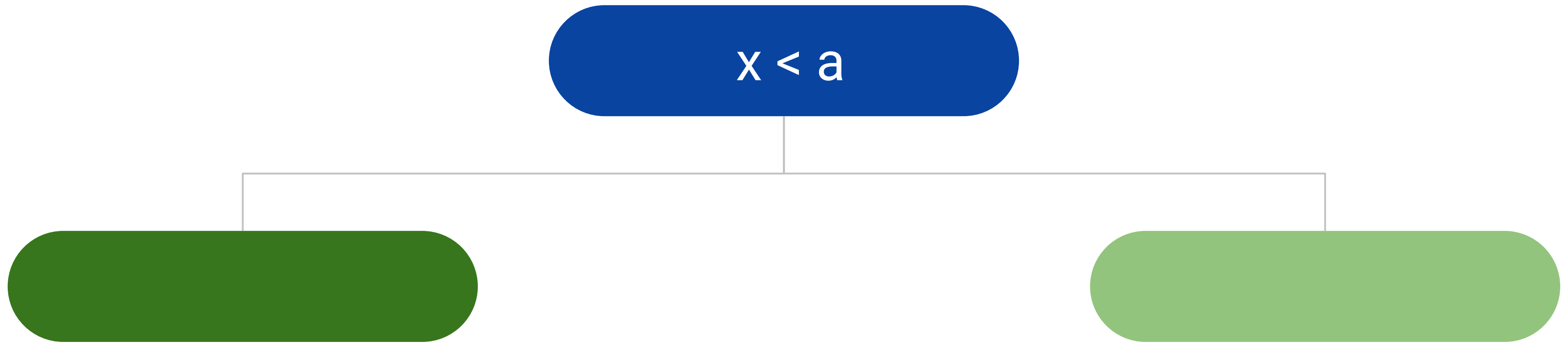
Robert Schapire

AdaBoost

1. **Инициализация весов**: На первом шаге, каждому обучающему примеру присваивается начальный вес `w[i] = 1/N`, где `N` - общее количество обучающих примеров. Эти веса определяют, насколько каждый пример важен для обучения

2. **Итеративный процесс** (Последовательное обучение базовых моделей:):

- Adaboost строит ансамбль из нескольких базовых моделей (называемых "слабыми учениками"), часто используют решаюшие пни.
- Алгоритм последовательно обучает базовые модели на основе весовых коэффициентов и ошибок предыдущих моделей.



AdaBoost

На каждой итерации (рассмотрим на примере классификации)**:**

1. Обучается базовая модель a_t на обучающих данных с весами $w[i]$
2. Вычисляется взвешенная ошибка e_t базовой модели a_t на текущих весах, как сумма весов тех примеров, на которых модель ошиблась

$$e_t = \sum_{i=1}^{N} w[i] \cdot \delta(y[i], a_t(x[i]))$$

где $\delta(y[i], a_t(x[i]))$ индикативная функция, равная 1, если $y[i] \neq a_t(x[i]$ и равная 0, если $y[i] = a_t(x[i]$

$y[i]$ фактические значения y на объектах

3. Вычисляется вес для базовой модели $\alpha_t = \frac{1}{2}\ln\left(\frac{1-e_t}{e_t}\right)$

AdaBoost

4. Вычисляется вес для базовой модели

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-e_t}{e_t}\right)$$

5. Обновляются веса обучающих примеров:

$$w[i] = w[i] \cdot \exp(-\alpha_t \cdot y[i] \cdot a_t(x[i]))$$

6. Веса нормализуются так, чтобы сумма стала равна 1

$$w_i = \frac{w_i}{\sum_{i=1}^{N} w[i]}$$

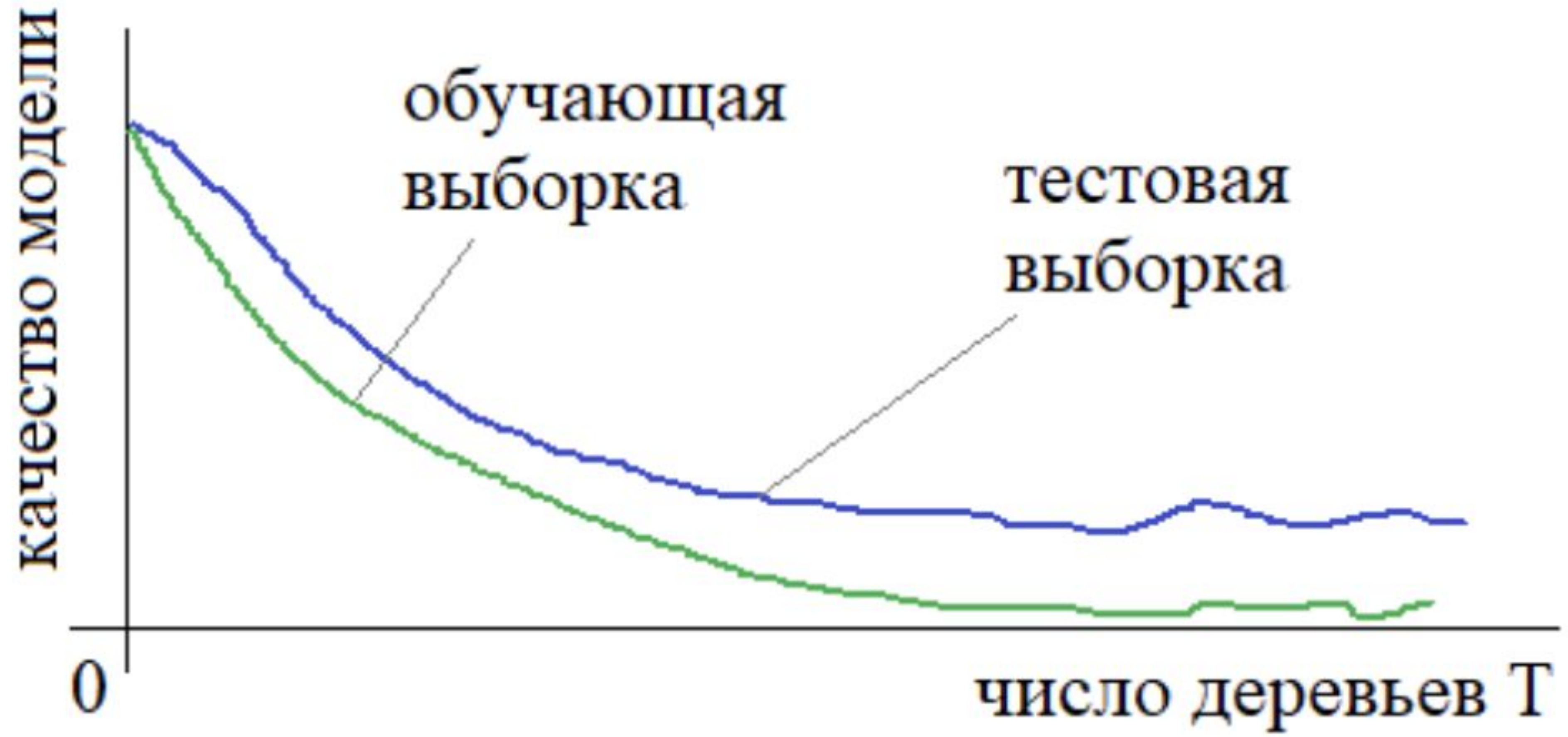
AdaBoost

Ансамбль моделей A(x) создается путем взвешенной комбинации предсказаний базовых

моделей a(x):

$$A(x) = \sum_{t=1}^{T} \alpha_t \cdot a_t(x)$$

жадная стратегия - все, что было ранее найдено, фиксируется и никак не меняется



AdaBoost

1) Инициализация весов (на первой итерации все наблюдения равны для нас)

x	y	weights
24	-1	0.2
96	1	0.2
2	1	0.2
10	-1	0.2
5	1	0.2

2) Применим первый базовый алгоритм - решаюший пень: x<10

x	y	weights	predictions
24	-1	0.2	-1
96	1	0.2	-1
2	1	0.2	1
10	-1	0.2	-1
5	1	0.2	1

AdaBoost

3) Подсчет ошибки

x	y	weights	predictions	incorrect_predictions
24	-1	0.2	-1	False
96	1	0.2	-1	True
2	1	0.2	1	False
10	-1	0.2	-1	False
5	1	0.2	1	False

взвешенная ошибка 0.2

считаем вес для базовой модели по формуле $\alpha_t = \frac{1}{2}\ln\left(\frac{1-e_t}{e_t}\right)$,

для наших данных вес для базовой модели равен 0.693

AdaBoost

3) пересчитываем веса по формуле $w[i] = w[i] \cdot \exp(-\alpha_t \cdot y[i] \cdot a_t(x[i]))$

x	y	weights	predictions	incorrect_predictions	weights_upd
24	-1	0.2	-1	False	0.1
96	1	0.2	-1	True	0.4
2	1	0.2	1	False	0.1
10	-1	0.2	-1	False	0.1
5	1	0.2	1	False	0.1

4) нормализуем веса, так чтобы их сумма стала равна 1

x	y	weights	predictions	incorrect_predictions	weights_upd	weights_upd_norm
24	-1	0.2	-1	False	0.1	0.125
96	1	0.2	-1	True	0.4	0.500
2	1	0.2	1	False	0.1	0.125
10	-1	0.2	-1	False	0.1	0.125
5	1	0.2	1	False	0.1	0.125

AdaBoost

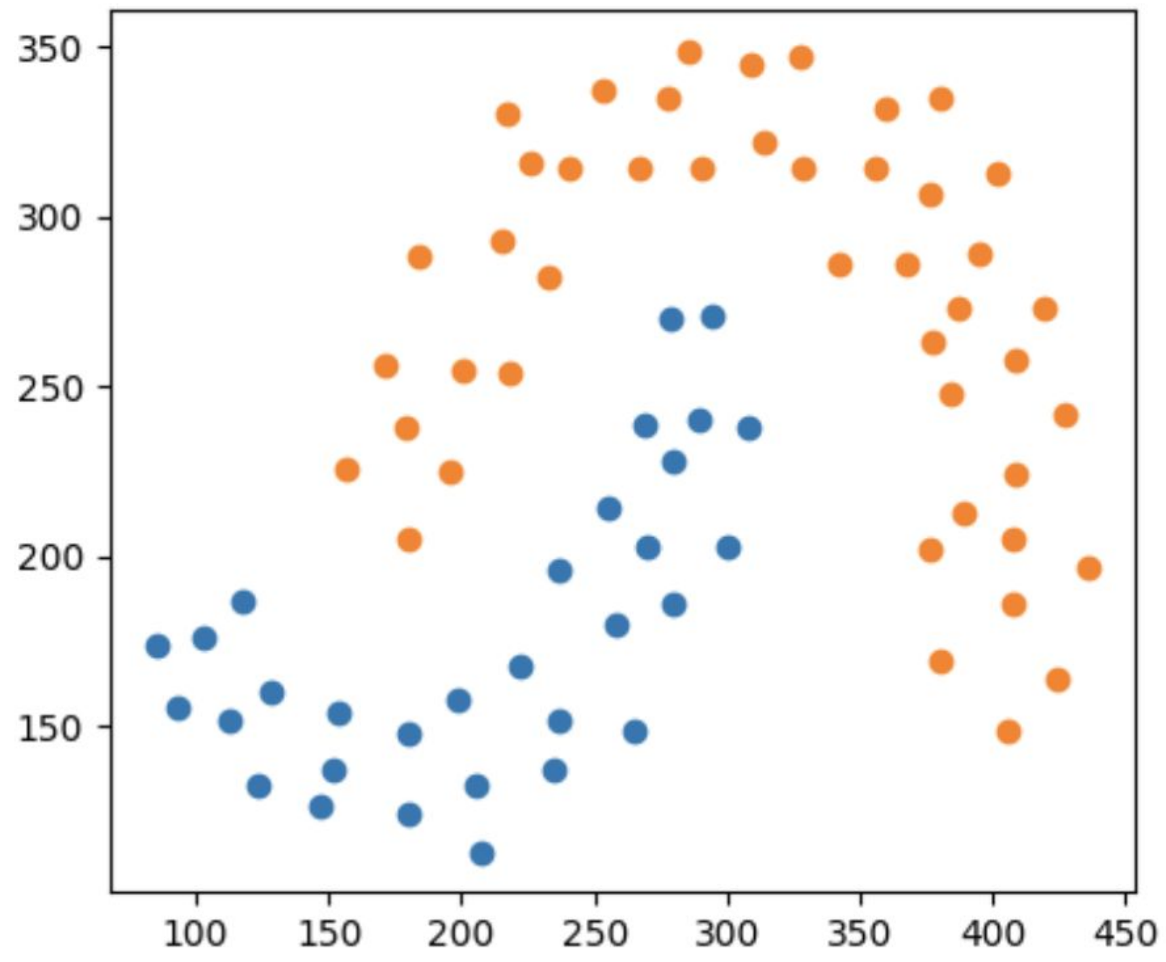
Там, где наш базовый алгоритм ошибся, вес увеличился сильнее всего, для объектов, где алгоритм не ошибся, вес снизился

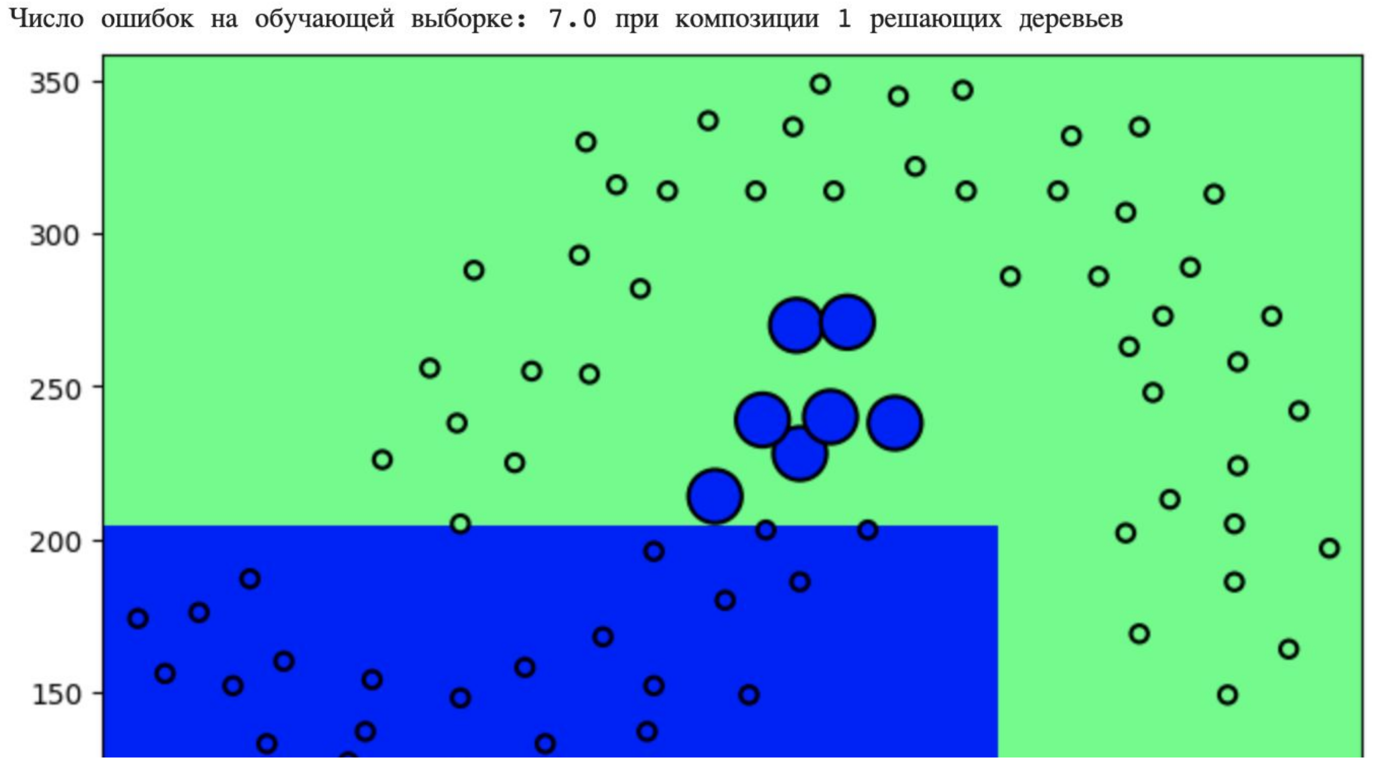
Далее процесс повторяем

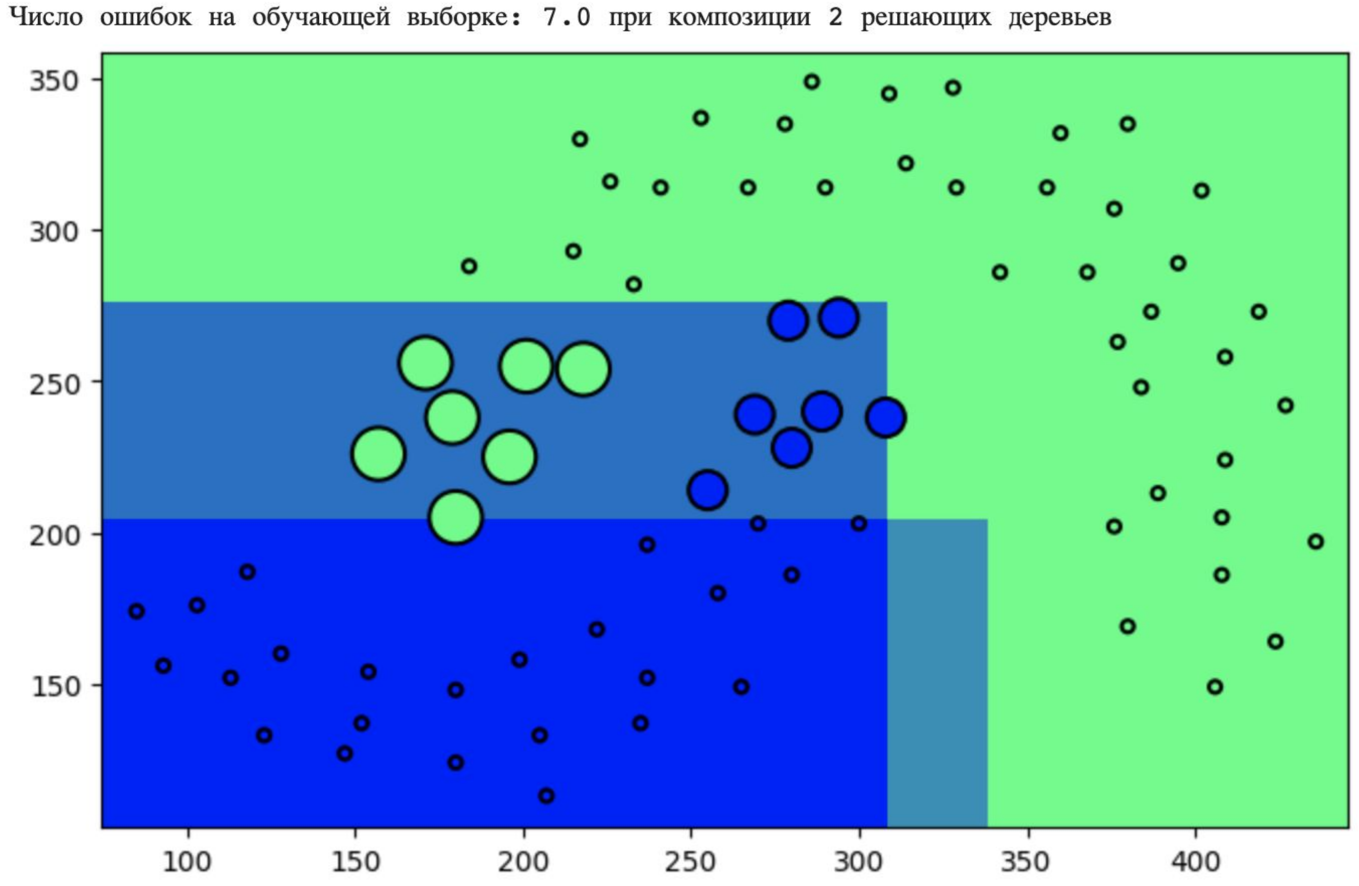
x	y	weights	predictions	incorrect_predictions	weights_upd	weights_upd_norm
24	-1	0.2	-1	False	0.1	0.125
96	1	0.2	-1	True	0.4	0.500
2	1	0.2	1	False	0.1	0.125
10	-1	0.2	-1	False	0.1	0.125
5	1	0.2	1	False	0.1	0.125

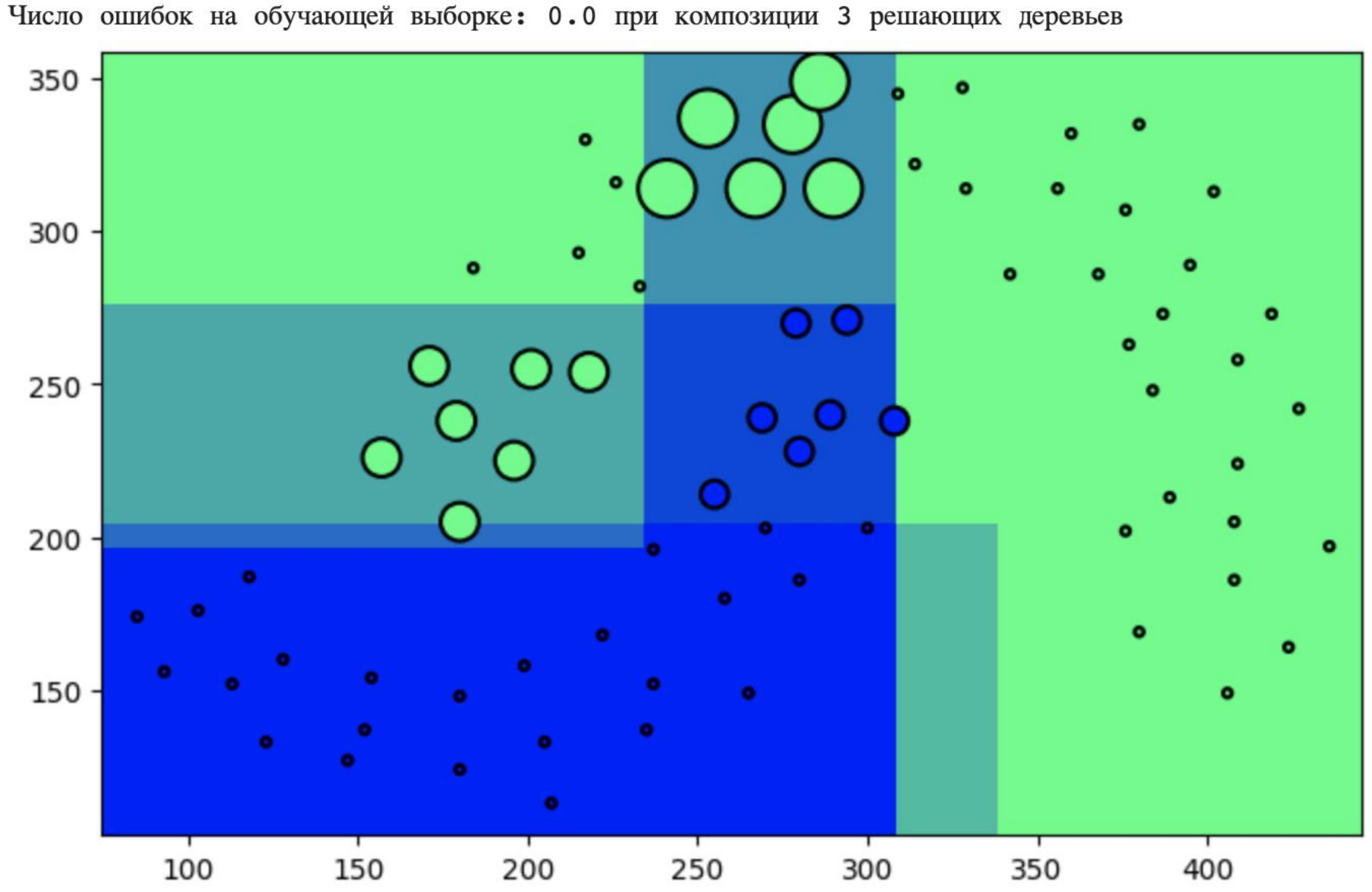
Процесс обучения AdaBoost останавливается после выполнения **заранее заданного числа итераций (слабых учителей)** или когда достигается **удовлетворительная точность** на обучающем наборе данных. Также может применяться **ранняя остановка**, если дополнительные итерации не улучшают производительность алгоритма.

AdaBoost. Игрушечный пример









https://proproprogs.ru/ml/ml-vvedenie-v-boosting-algoritm-adaboost-pri-klassifikacii

AdaBoost. Можно ли его улучшить?

Можно использовать разные функции потерь

$L(y, f(x)) = \exp(-y \cdot f(x))$ - Экспоненциальная функция потерь (AdaBoost)

$L(y, f(x)) = \ln(1 + \exp(-2y \cdot f(x)))$ - Логарифмическая функция потерь (LogitBoost)

$L(y, f(x)) = (y - f(x))^2$ - Квадратичная функция потерь (GentleBoost)

$L(y, f(x)) = \exp\left(-\frac{(y-f(x))^2}{2\sigma^2}\right)$ - Гауссовская функция потерь (BrownBoost):

AdaBoost. Можно ли его улучшить?

INNOPOLIS UNIVERSITY

Jerome Isaac Friedman

Можно придумать много других функций для разработки (синтеза) новых алгоритмов бустинга. И здесь возникает естественный вопрос. А можно ли создать универсальный алгоритм бустинга, который бы работал с произвольной гладкой и дифференцируемой функцией потерь?

Оказывается ДА, можно!

И такой подход получил название **градиентного бустинга.**

Впервые градиентный бустинг представил Jerome Friedman (Джером Фридман) в 1999 году

Градиентный бустинг. Алгоритм

INNOPOLIS UNIVERSITY

1. **Инициализация:**
    - Инициализировать ансамбль средним предсказанием: $F_0(x) = avg$.
    - Вычислить начальные остатки: $r_{i0} = y_i - F_0(x_i)$, где y_i - истинное значение, x_i - обучающий пример.
2. **Для каждой итерации t от 1 до T , где T - количество базовых моделей:**.

    a. Обучение базовой модели a_t на обучающих данных, предсказывающей остатки r_{it}.

    b. Вычислить множитель γ_t путем решения задачи оптимизации:

    $\gamma_t = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, F_{t-1}(x_i) + \gamma \cdot a_t(x_i))$

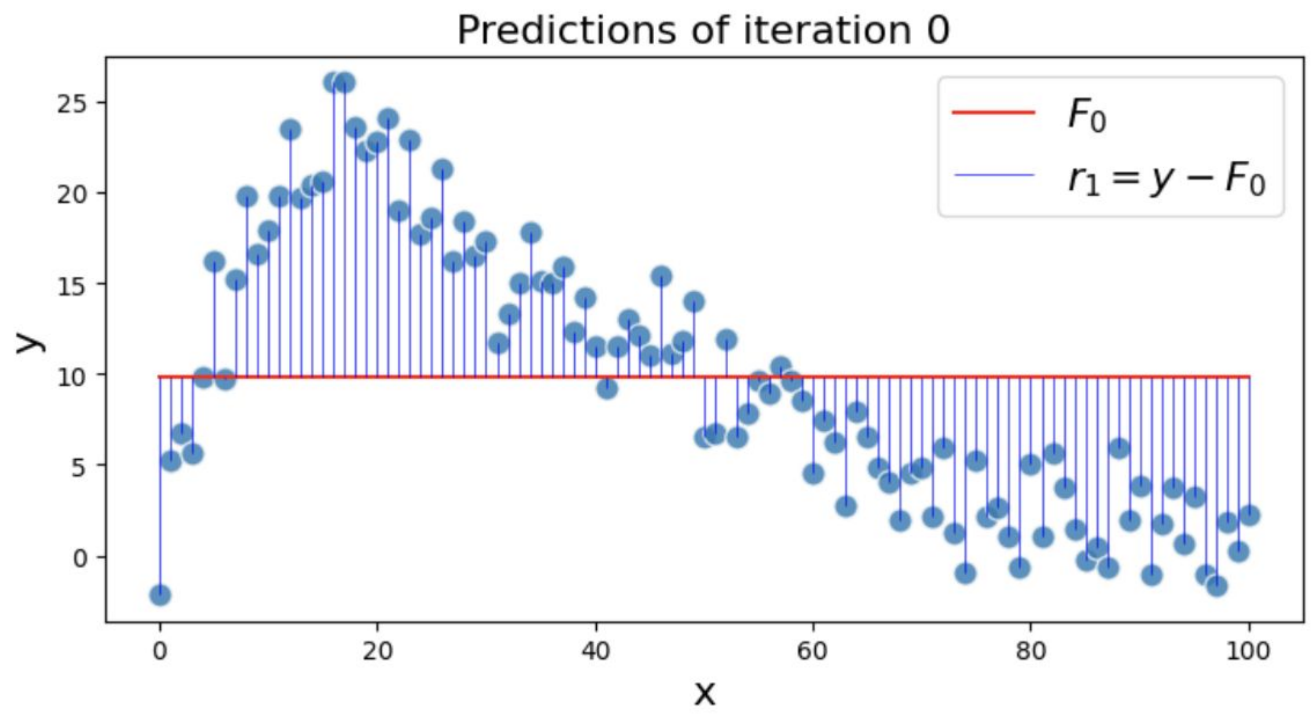
    где L - функция потерь (например, квадратичная), y_i - истинное значение,

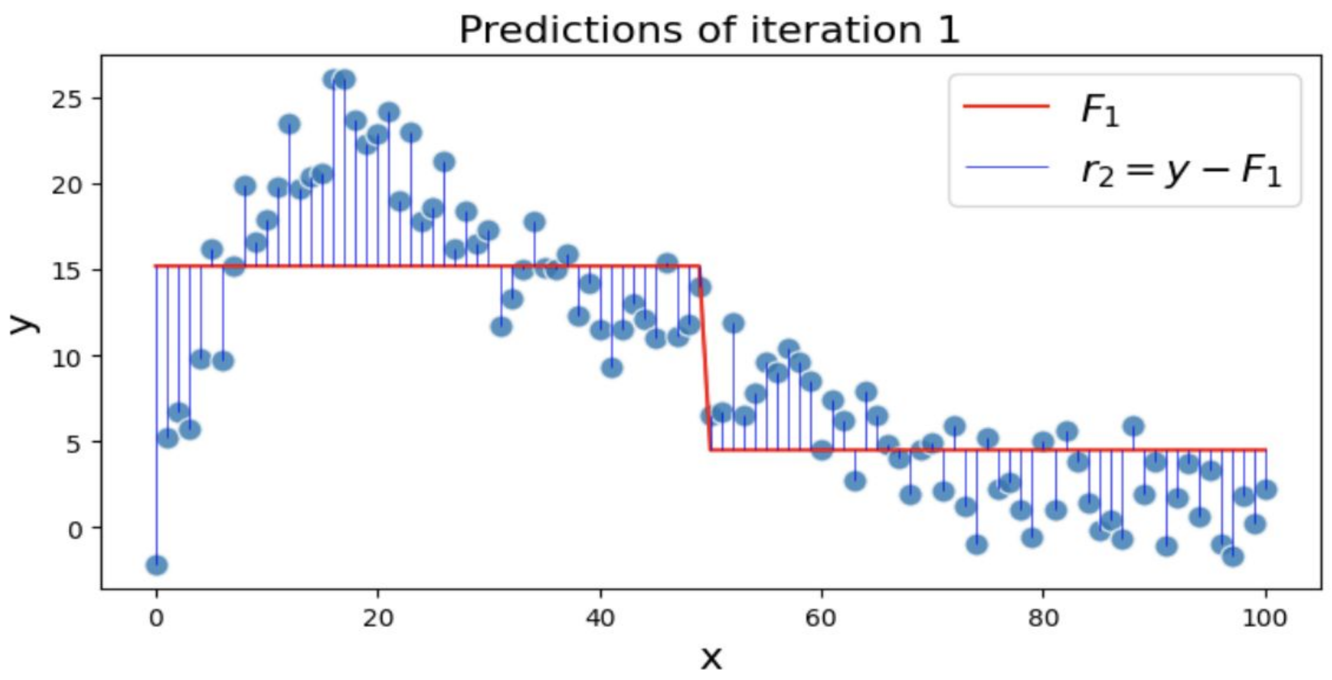
    $F_{t-1}(x_i)$ - предсказание ансамбля на предыдущей итерации, $a_t(x_i)$ - предсказание базовой модели a_t.

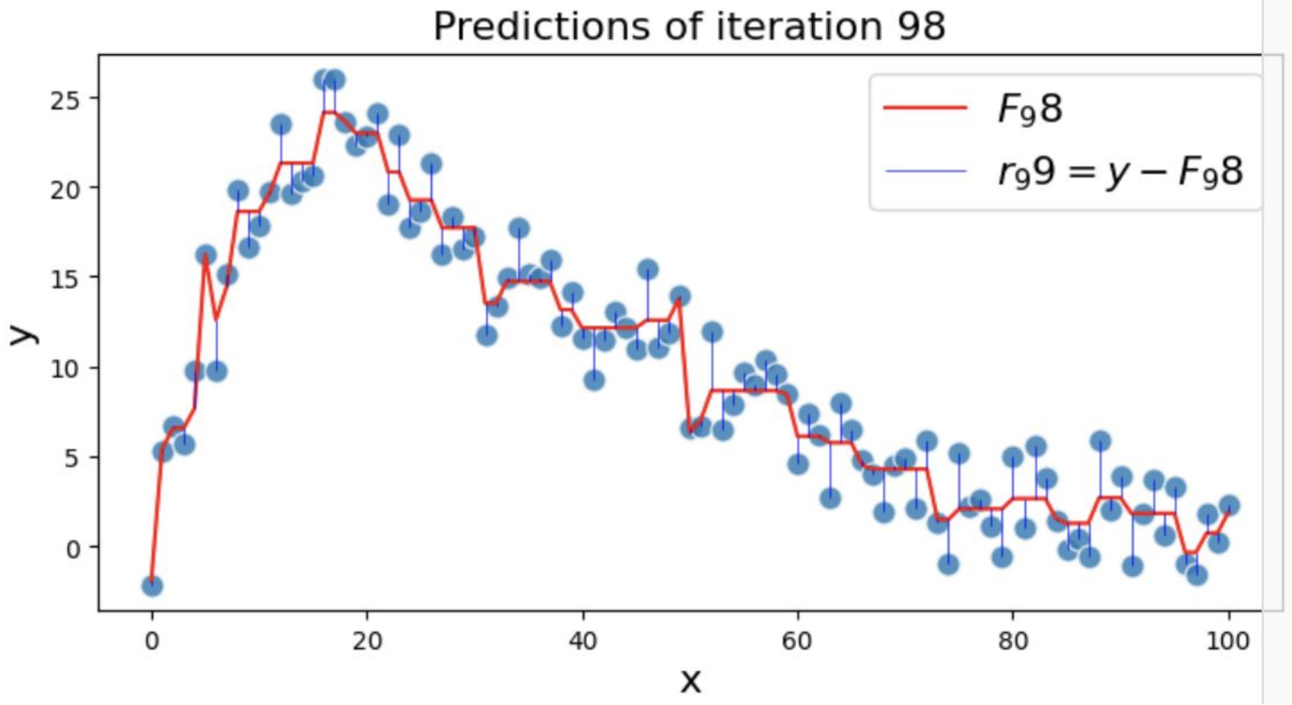
    c. Обновить ансамбль: $F_t(x) = F_{t-1}(x) + \gamma_t \cdot a_t(x)$.

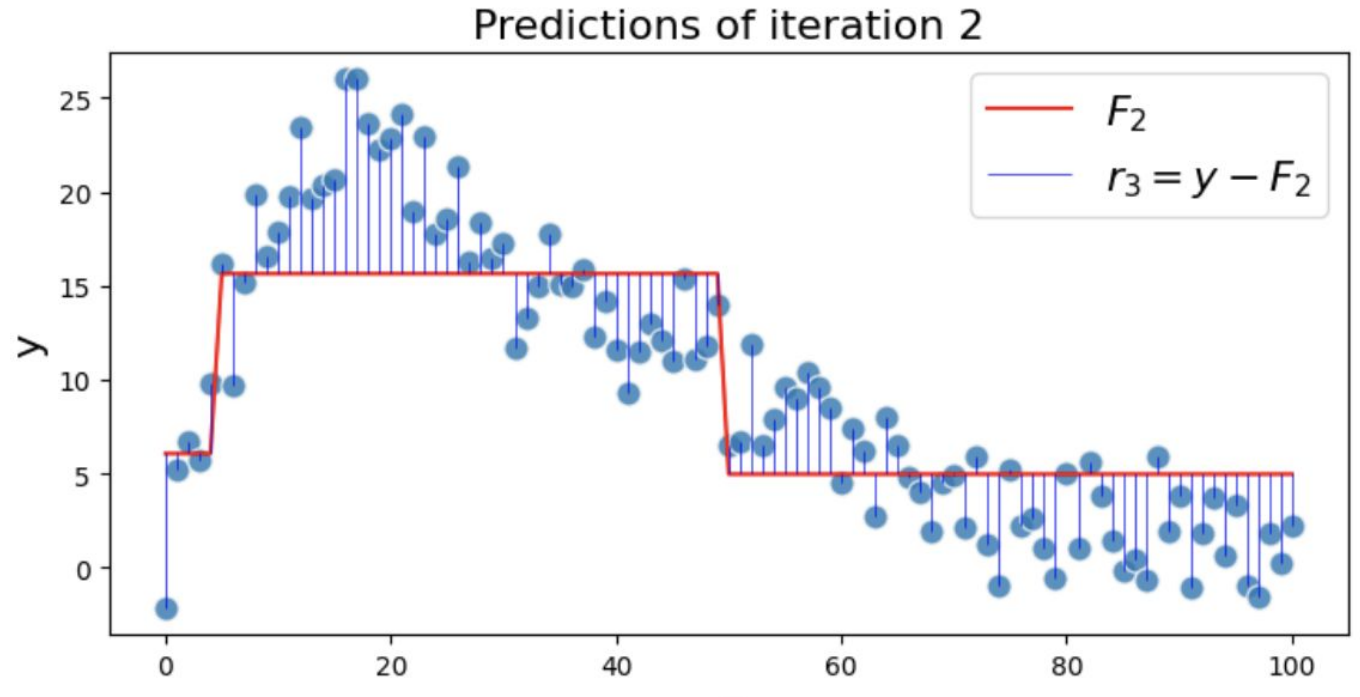
    d. Обновить остатки: $r_{it} = y_i - F_t(x_i)$.
3. **Окончательное предсказание:** Итоговое предсказание для нового примера x: $F_T(x) = F_0(x) + \sum_{t=1}^{T} \gamma_t \cdot a_t(x)$

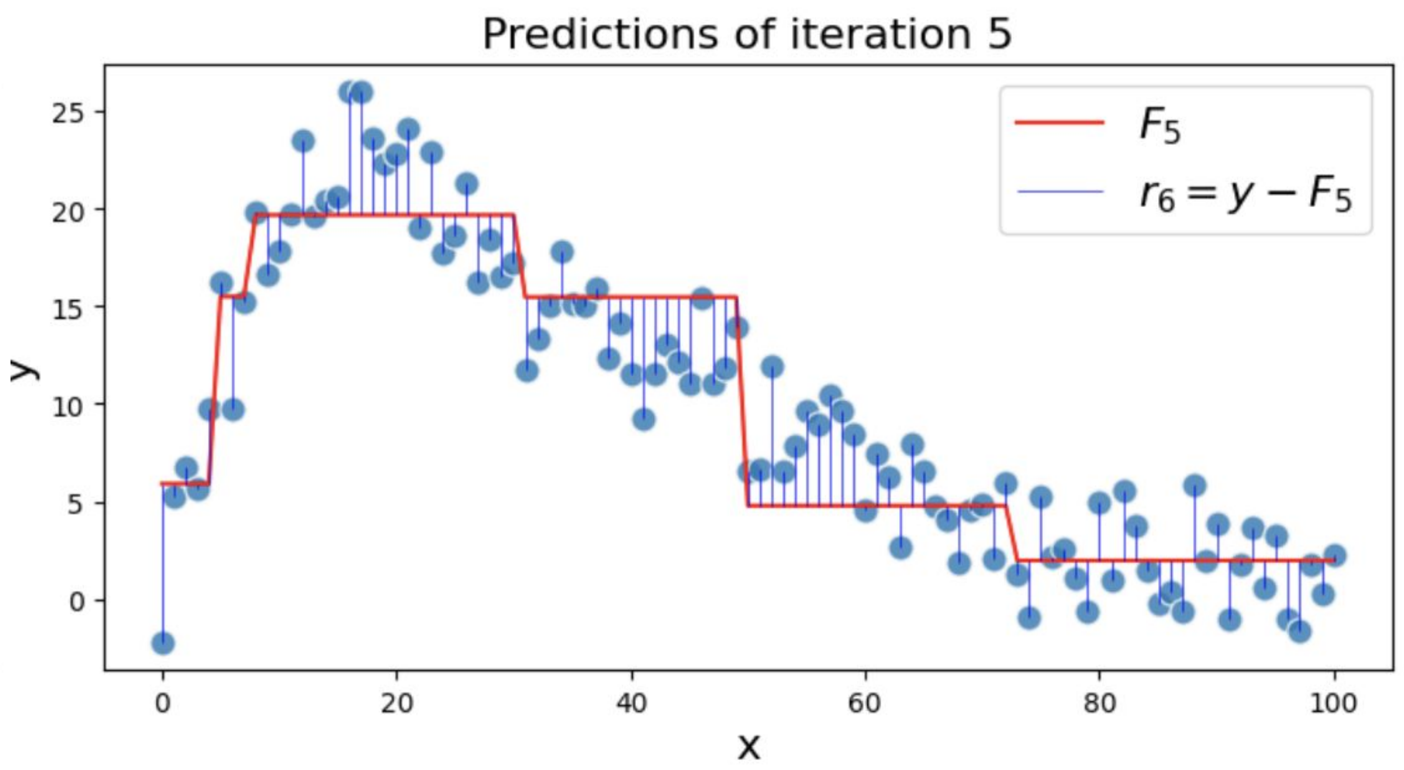
Градиентный бустинг. ДЕМО











https://colab.research.google.com/drive/1_fWOmF8Dm8hv-YFOREC4BkWOmnHak-zl?usp=sharing

Градиентный бустинг & градиентный спуск

Градиентный спуск:

- Градиентный спуск используется **для оптимизации функции**, минимизируя её по направлению наискорейшего убывания градиента
- Веса параметров модели (например, веса в линейной регрессии) обновляются в направлении, противоположном градиенту функции потерь, с учетом некоторого коэффициента (learning rate).

Градиентный бустинг:

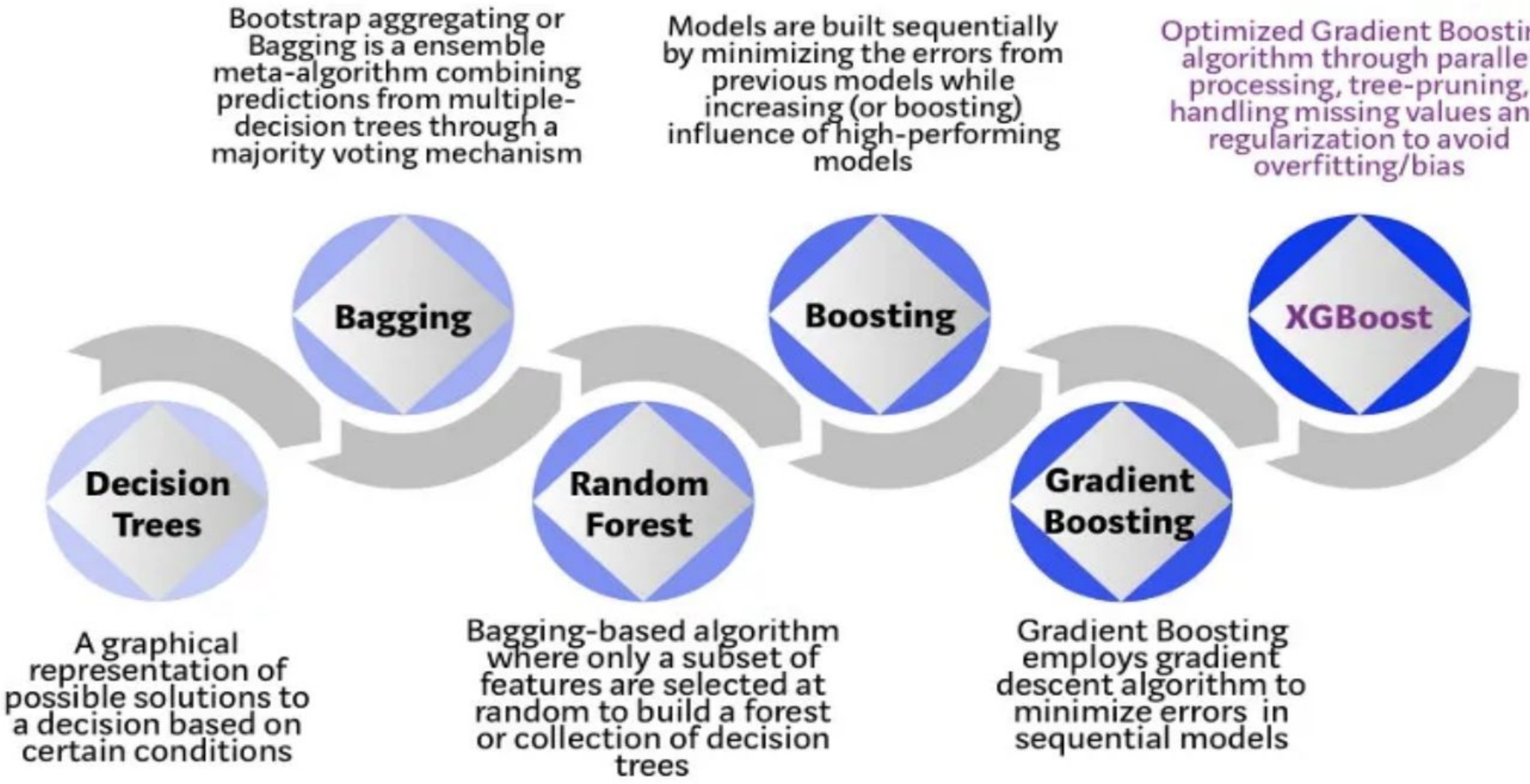
- Градиентный бустинг строит ансамбль слабых моделей, таких как деревья решений. Каждое следующее дерево обучается на остатках предыдущей композиции моделей
- **Градиенты функции потерь по остаткам** на каждой итерации указывают на то, **какие направления** требуется **скорректировать в ансамбле,** чтобы улучшить его предсказания.
- Новая слабая модель добавляется к ансамблю таким образом, чтобы она аппроксимировала градиент функции потерь. Это делается с учетом градиента и при помощи оптимизации, чтобы новая модель учла ошибки, допущенные предыдущими моделями.

Таким образом, хотя оба метода используют градиенты, их цели и подходы к применению градиентов различны.

Градиентный спуск - это метод оптимизации, а градиентный бустинг - метод построения ансамблей моделей, который использует **градиенты для коррекции предсказаний ансамбля** на каждом шаге.

Градиентный бустинг. Можно ли улучшить?

XGBoost, которая расшифровывается как Extreme Gradient Boosting (Экстремальный градиентный бустинг), предложенной Тяньци Ченом и Карлосом Гестрином в 2014 году.



Эволюция алгоритмов, использующих деревья поиска решений

1) Из 29 победивших решений на Kaggle за 2015 год, в 17 использовался XGBoost
2) В восьми из этих 17 решений использовался только XGBoost, а в остальных девяти — XGBoost в сочетании с нейросетями.

ссылка на стаью
https://arxiv.org/pdf/1603.02754.pdf

XGBoost. Что улучшили?

Улучшения алгоритма:

1. **Использует вторые производные.** Разбиение узлов дерева: Для выбора наилучшего разбиения (или узла) на каждом этапе построения дерева, XGBoost рассчитывает оптимальное разбиение на основе прироста функции потерь. Прирост функции потерь можно оценить с помощью градиентов и Гессианов. Формула для расчета прироста ΔLoss для разбиения узла:

$$\Delta\text{Loss} = \frac{1}{2}\left(\frac{(G^2)_{left}}{H_{left}+\lambda} + \frac{(G^2)_{right}}{H_{right}+\lambda} - \frac{(G^2)}{H+\lambda}\right) - \gamma$$

где:

- G — сумма градиентов.
- H — сумма Гессианов.
- λ — L2-регуляризация.
- γ — штраф за сложность дерева.

XGBoost. Что улучшили?

Улучшения алгоритма:

1. **Апроксимационный поиск точек расщепления (глобальный метод):**
   i. расчет квантилей
   ii. дальше этот набор точек предлагается на каждом следующем этапе
2. **Параллелизация**: В XGBoost построение деревьев основано на параллелизации. Это возможно благодаря взаимозаменяемой природе циклов, используемых для построения базы для обучения: внешний цикл перечисляет листья деревьев, внутренний цикл вычисляет признаки
3. **Кросс-валидация:** Алгоритм использует свой собственный метод кросс-валидации на каждой итерации. То есть, нам не нужно отдельно программировать этот поиск и определять количество итераций бустинга для каждого запуска
4. **Регуляризация:** Штрафует сложные модели, используя как регуляризацию LASSO (L1), так и Ridge-регуляризацию (L2), для того, чтобы избежать переобучения.

УНИВЕРСИТЕТ
ИННОПОЛИС

ВОПРОСЫ И ОТВЕТЫ