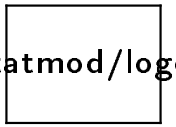
Решающие деревья. Random Forest

Попов Владимир
Шаповал Егор
Леонович Роман



Санкт-Петербург
2022 г.

Бустинг в задаче регрессии

Рассмотрим задачу минимизации квадратичного функционала:

$$\frac{1}{2}\sum_{i=1}^{l}(a(x_i)-y_i)^2 \to \min_a$$

Будем искать итоговый алгоритм в виде суммы базовых моделей $b_n(x)$:

$$a_N(x) = \sum_{n=1}^{N} b_n(x),$$

где базовые алгоритмы $b_n \in \mathbf{A}$.

Первый базовый алгоритм:

$b_1(x) := \frac{1}{2}\sum_{i=1}^{l}(b_1(x_i)-y_i)^2 \to \min_{b_1}.$

Остатки на каждом объекте: $s_i^{(1)} = y_i - b_1(x_i)$

$b_2(x) := \frac{1}{2}\sum_{i=1}^{l}(b(x_i)-s_i^{(1)})^2 \to \min_{b_2}$

Бустинг в задаче регрессии

Таким образом, каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:

$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} b_n(x_i) = y_i - a_{N-1}(x_i), \quad i = 1, \ldots, l$$

$$b_N(x) := \frac{1}{2} \sum_{i=1}^{l} (b(x_i) - s_i^{(N)})^2 \to \min_{b_N}$$

Пример

age	square footage	location	price
5	1500	5	480
11	2030	12	1090
14	1442	6	350
8	2501	4	1310
12	1300	9	400
10	1789	11	506

Рис.: Пример данных

Пример

age	square footage	location	price	residuals
5	1500	5	480	-208
11	2030	12	1090	402
14	1442	6	350	-338
8	2501	4	1810	622
12	1300	9	400	-288
10	1789	11	500	-188

Рис.: Подсчет остатков

Пример

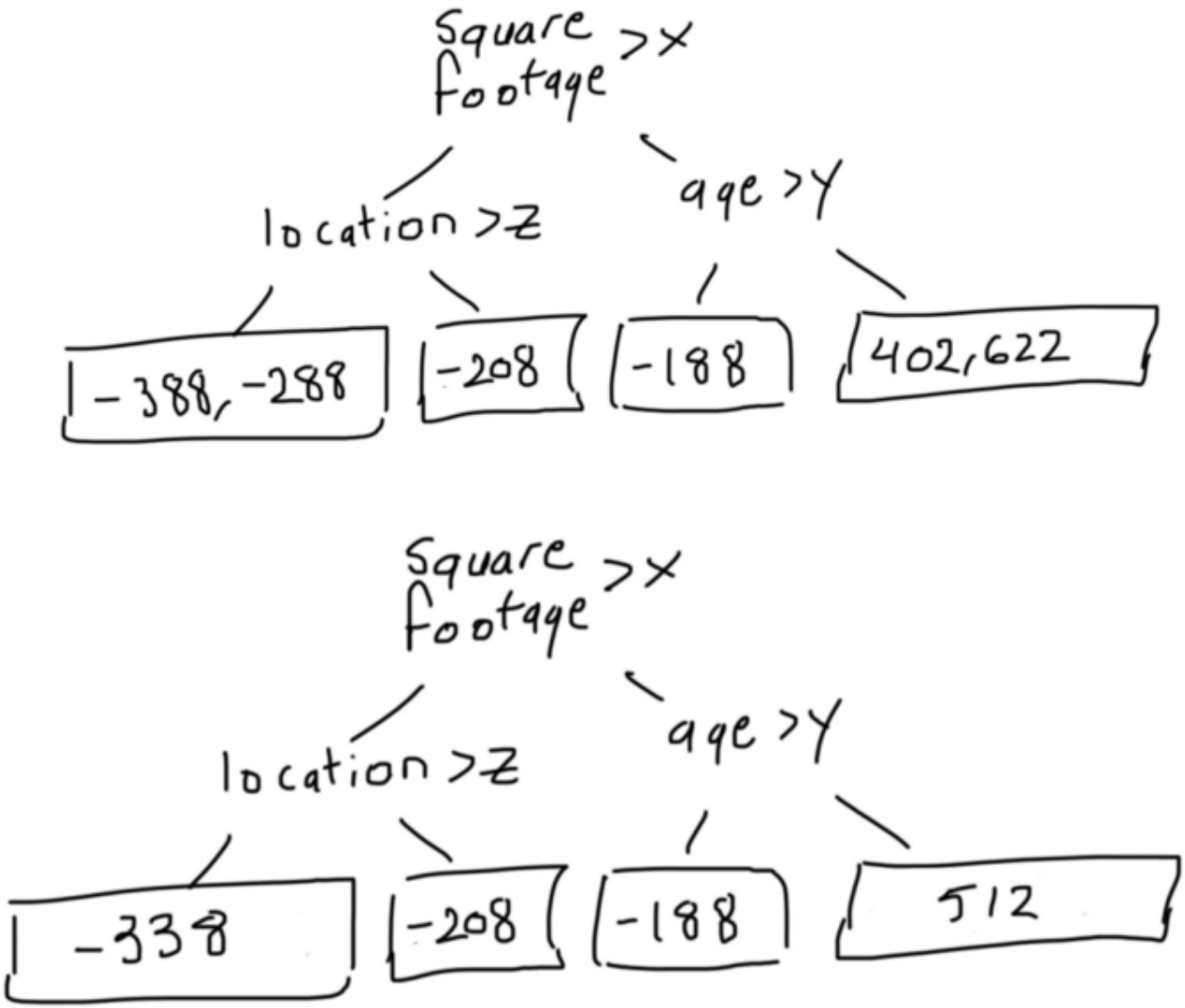


Рис.: Построение дерева

Пример

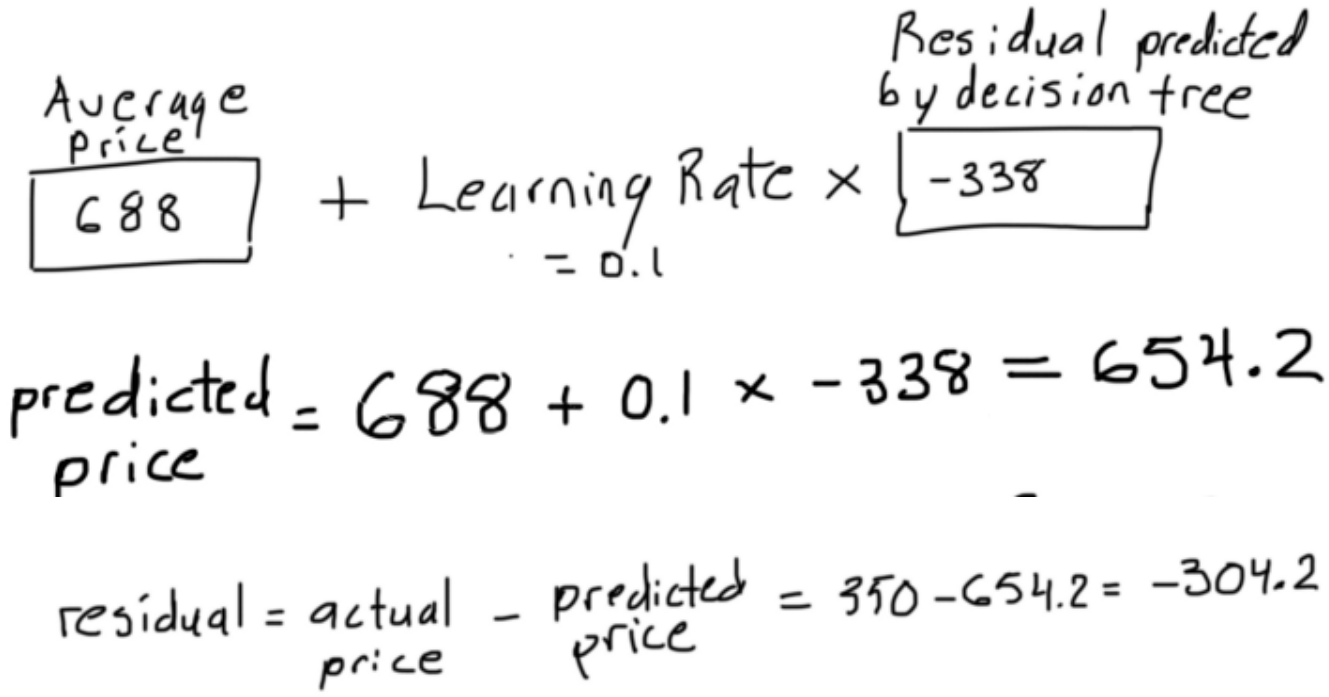


Рис.: Подсчет остатков

Пример

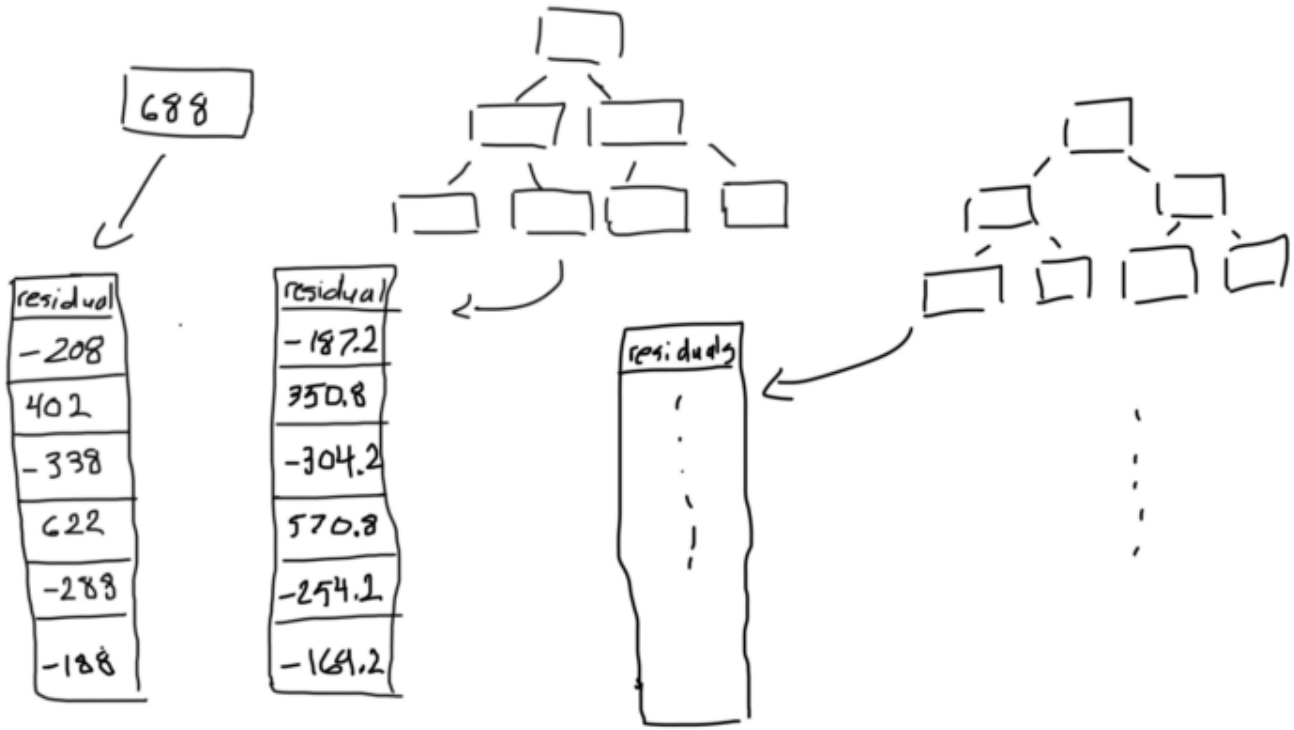


Рис.: Проводим следующие итерации

Градиентный бустинг

Пусть дана некоторая дифференцируемая функция потерь $L(y, z)$.
Будем строить взвешенную сумму базовых алгоритмов:

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

Примеры выбора алгоритма $b_0(x)$:

1. Нулевой: $b_0(x) = 0$.
2. Возвращающий самый популярный класс (в задачах классификации):
   $b_0(x) = \underset{y \in \mathbb{Y}}{argmax} \sum_{i=1}^{l} [y_i = y]$
3. Возвращающий средний ответ (в задачах регрессии):
   $b_0(x) = \frac{1}{l} \sum_{i=1} l y_i$

Градиентный бустинг

Допустим, мы построили композицию $a_{N-1}(x)$ из $N-1$ алгоритма, и хотим выбрать следующий абзовый алгоритм $b_N(x)$ так, чтобы как можно сильнее уменьшить ошибку:

$$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)\right) \to \min_{b_N, \gamma_N}$$

Функция алгоритма

Какие числа $s_1, \ldots, s_l$ надо выбрать для решения следующей задачи?

$$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + s_i\right) \to \min_{s_1,\ldots,s_l}$$

- $s_i = y_i - a_{N-1}(x_i)$
- $s_i = -\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}$

В этом случае сдвиг s_i будет противоположен производной функции потерь в точке $z = a_{N-1}(x_i)$

Проблемы градиентного бустинга

- Если базовые алгоритмы очень простые, то они плохо приближают вектор антиградиента. Соответственно, градиентный бустинг может свестись к случайному блужданию в пространстве.
- Если базовые алгоритмы сложные, то они способны за несколько шагов бустинга идеально подогнаться под обучающую выборку.
  Решение:
  - Сокращение шага
    $a_N(x) = a_{N-1}(x) + \eta\gamma_N b_N(x)$, где $\eta \in (0, 1]$ — темп обучения.
  - Стохастический градиентный бустинг

Градиентный бустинг над деревьями

$$b_n(x) = \sum_{j=1}^{J_n} b_{nj}[x \in R_j],$$

где $j = 1, \ldots, J_n$ — индексы листьев, R_j — соответствующие области разбиения, b_{nj} — значения в листьях.

В N-й итерации бустинга композиция обновляется как

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{Nj}[x \in R_j]$$

Можно улучшить качество композиции:

$$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + \sum_{j=1}^{J_N} \gamma_{Nj}[x \in R_j]\right) \to \min_{\{\gamma_{Nj}\}_{j=1}^{J_N}}$$

Градиентный бустинг над деревьями

Так как области разбиения R_j не пересекаются, данная задача распадается на J_N независимых подзадач:

$$\gamma_{Nj} = \underset{\gamma}{argmin} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma), \qquad j = 1, \ldots, J_N$$

Имеем вектор сдвигов, который показывает, как нужно скорректировать ответы композиции на обучающей выборке, чтобы как можно сильнее уменьшить ошибку:

$$\left(-\frac{\partial L}{\partial z}\Big|_{z=a_{N-1}(x_i)} \right)_{i=1}^{l} = -\nabla z \sum_{i=1}^{l} L(y_i, z_i)|_{z=a_{N-1}(x_i)}$$

После этого новый базовый алгоритм обучается путем минимизации среднеквадратичного отклонения от вектора сдвигов s

$$b_N(x) = \underset{b \in \mathbf{A}}{argmin} \sum_{i=1}^{l} (b(x_i) - s_i)^2 \qquad (1)$$

Альтернативный подход

Мы хотим найти алгоритм $b(x)$, решающий следующую задачу:

$$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + b(x_i)\right) \to \min_b$$

Разложим функцию L в каждом слагаемом в ряд Тейлора до второго члена с центром в ответе композиции $a_{N-1}(x_i)$:

$$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + b(x_i)\right) \approx$$

$$\approx \sum_{i=1}^{l} \left(L\left(y_i, a_{N-1}(x_i)\right) - s_i b(x_i) + \tfrac{1}{2} h_i b^2(x_i)\right),$$

где h_i — вторые производные по сдвигам:

$$h_i = \frac{\partial^2}{\partial z^2} L(y_i, z)|_{a_{N-1}(x_i)}$$

Альтернативный подход

Так как первое слагаемое не зависит от нового базового алгоритма, то его можно выкинуть:

$$\sum_{i=1}^{l}\left(-s_i b(x_i) + \frac{1}{2}h_i b^2(x_i)\right) \tag{2}$$

Преобразуем среднеквадратичный функционал из формулы (1):

$$\sum_{i=1}^{l}(b(x_i) - s_i)^2 = 2\sum_{i=1}^{l}(-s_i b(x_i) + \frac{1}{2}b^2(x_i)) \to \min_b$$

Регуляризация

Будем далее работать с функционалом (2). Дерево $b(x)$ можно описать формулой

$$b(x) = \sum_{j=1}^{J} b_j \, [x \in R_j]$$

Его сложность зависит от двух показателей:

1. Число листьев J. Чем больше листьев имеет дерево, тем сложнее его разделяющая поверхность, тем больше у него параметров и тем выше риск переобучения.
2. Норма коэффициентов в листьях $||b||_2^2 = \sum_{j=1}^{J} b_j^2$. Чем сильнее коэффициенты отличаются от нуля, тем сильнее данный базовый алгоритм будет влиять на итоговый ответ композиции.

Регуляризация

Добавляя регуляризаторы, штрафующие за оба этих вида сложности, получаем следующую задачу:

$$\sum_{i=1}^{l}(-s_i b(x_i) + \tfrac{1}{2}b^2(x_i)) + \gamma J + \tfrac{\lambda}{2}\sum_{j=1}^{J} b_j^2 \to \min_b$$

Так как дерево $b(x)$ дает одинаковые ответы на объектах, попадающих в один лист, то можно упростить функционал:

$$\sum_{j=1}^{J}\left\{\underbrace{\left(-\sum_{i\in R_j} s_i\right)}_{=-S_j} b_j + \frac{1}{2}\left(\lambda + \underbrace{\sum_{i\in R_j} h_i}_{=H_j}\right) b_j^2 + \gamma\right\} \to \min_b$$

Регуляризация

Можно аналитически найти оптимальные коэффициенты в листьях:
$b_j = \frac{S_j}{H_j + \lambda}$
Подставляя данное выражение обратно в функционал, получаем, что ошибка дерева с оптимальными коэффициентами в листьях вычисляется по формуле

$$H(b) = -\frac{1}{2}\sum_{j=1}^{J}\frac{S_j^2}{H_j + \lambda} + \gamma J \tag{3}$$

— Критерий качества структуры дерева

Обучение решающего дерева

Мы получили функционал $H(b)$, который для заданной структуры дерева вычисляет минимальное значение ошибки (2), которую можно получить путем подбора коэффициентов в листьях. Будем выбирать разбиение $[x_j < t]$ в вершине R так, чтобы оно решало следующую задачу максимизации:

$$Q = H(R) - H(R_l) - H(R_r) \to max$$

где информативность вычисляется по формуле

$$H(R) = -\frac{1}{2} \left(\sum_{(h_i, s_i) \in R} s_j \right)^2 \Bigg/ \left(\sum_{(h_i, s_i) \in R} h_j + \lambda \right) + \gamma$$

Особенности XGBoost

1. Базовый алгоритм приближает направление, посчитанное с учетом вторых производных функции потерь.
2. Функционал регуляризуется — добавляются штрафы за количество листьев и за норму коэффициентов.
3. При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига.

Смещение и разброс

В случайных лесах:

- Используются глубокие деревья, поскольку от базовых алгоритмов требуется низкое смещение.
- Разброс устраняется за счёт усреднения ответов различных деревьев.

В бустинге:

- Каждый следующий алгоритм понижает ошибку композиции.
- Переобучение при большом количестве базовых моделей.
- Можно понизить смещение моделей, а разброс либо останется таким же, либо увеличится.
- Используются неглубокие решающие деревья.

Функции потерь регрессии

- $L(y, z)(y - z)^2$ — Gaussian loss (L_2 loss)
- $L(y, z) = |y - z|$ — Laplacian loss (L_1 loss)
- $L(y, z) = \begin{cases} (1 - \alpha)\,|y - z| & |y - z| \leq 0 \\ \alpha\,|y - z| & |y - z| \leq 0 \end{cases}, \alpha \in (0, 1)$ — Quantile loss (L_q loss)

Функции потерь регрессии

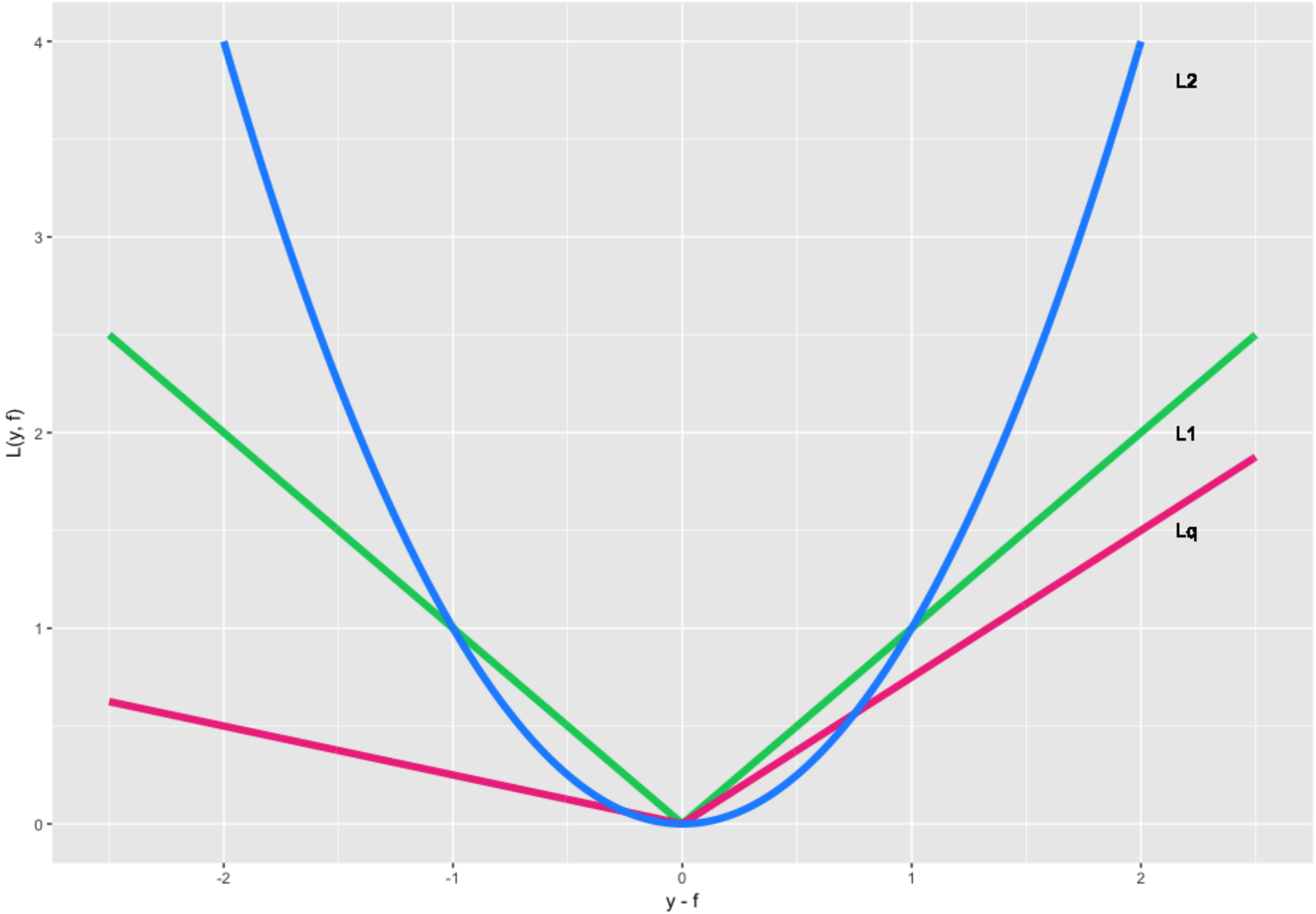


Рис.: Функции потерь регрессии

Функции потерь классификации

- $L(y, z) = log(1 + e^{-2yz})$ — Logistic loss. Ммы штрафуем даже корректно предсказанные метки классов. Оптимизируя эту функцию потерь, мы можем продолжать "раздвигать"классы и улучшать классификатор даже если все наблюдения предсказаны верно. Это самая стандартная и часто используемая функция потерь в бинарной классификации.
- $L(y, z) = e^{-yz}$ — Adaboost loss. Эта функция потерь очень похожа на Logistic loss, но имеет более жесткий экспоненциальный штраф на ошибки классификации и используется реже.

Функции потерь классификации

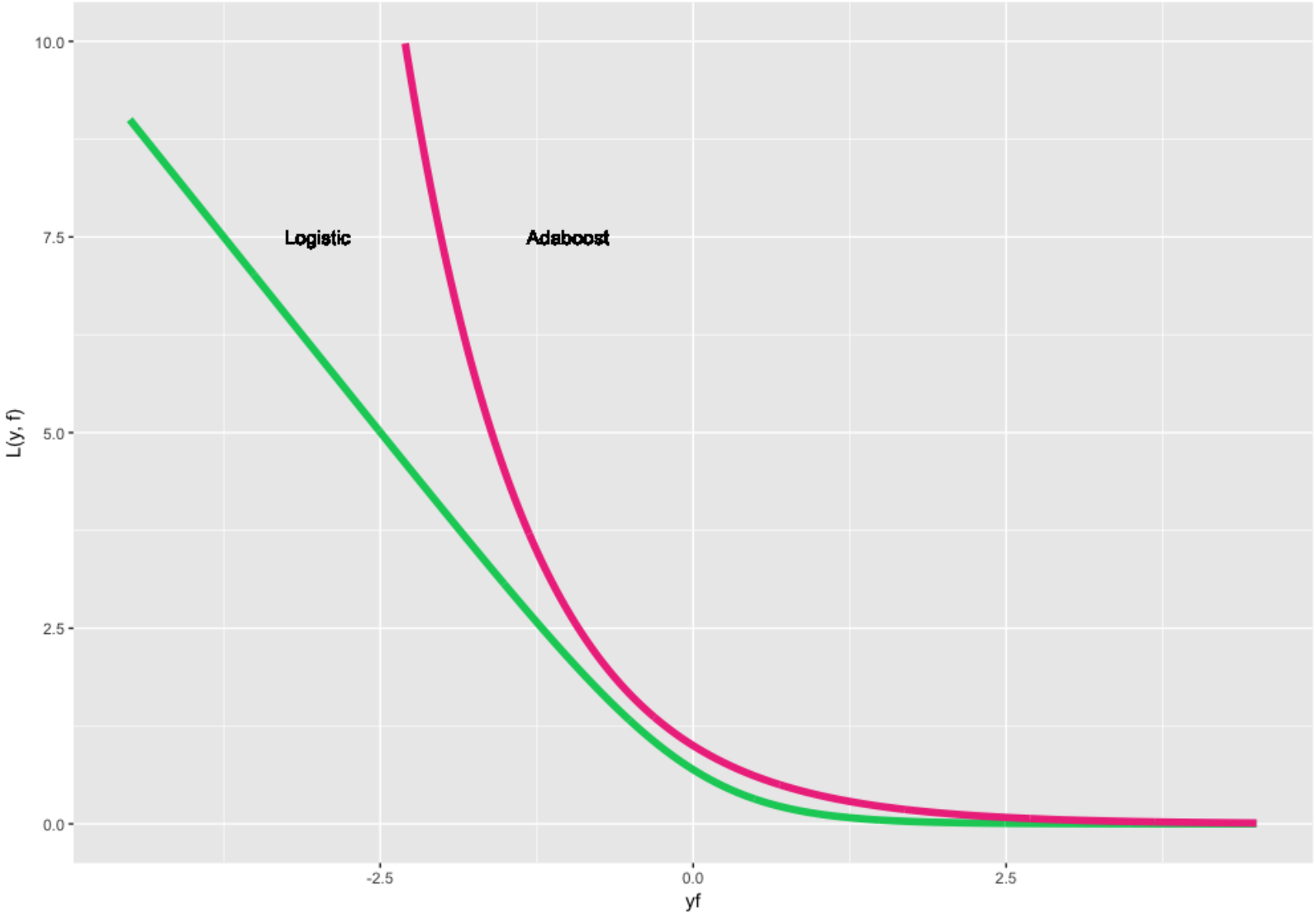


Рис.: Функции потерь классификации

Взвешивание объектов

AdaBoost: $L(y, z) = exp(-yz)$

$$L(a, X) = \sum_{i=1}^{l} exp\left(-y_i \sum_{n=1}^{N} \gamma_n b_n(x_i)\right)$$

Компоненты ее антиградиента после $N-1$ итерации:

$$s_i = -\frac{\partial L(y_i, z)}{\partial z}\Big|_z = a_{N-1}(x_i) = y_i \underbrace{exp\left(-y_i \sum_{n=1}^{N-1} \gamma_n b_n(x_i)\right)}_{w_i}$$

Влияние шума на обучение

Рассмотрим теперь логистическую функцию потерь, которая также может использоваться в задачах классификации:

$$L(a, X^l) = \sum_{i=1}^{l} log(1 + exp(y_i a(x_i)))$$

Ее антиградиент после $N - 1$ шага:

$$s_i = y_i \underbrace{\frac{1}{1 + exp(y_i a_{N-1}(x_i))}}_{w_i^{(N)}}$$