Композиции классификаторов (часть 2)

К. В. Воронцов, А.В. Зухба
vokov@forecsys.ru
a__l@mail.ru

сентябрь 2016

Бустинг для задачи классификации с двумя классами

Возьмём $Y = \{\pm 1\}$, $b_t \colon X \to \{-1, 0, +1\}$, $C(b) = \mathrm{sign}(b)$.
$b_t(x) = 0$ — отказ (лучше промолчать, чем соврать).

Взвешенное голосование:

$$a(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t b_t(x)\right), \quad x \in X.$$

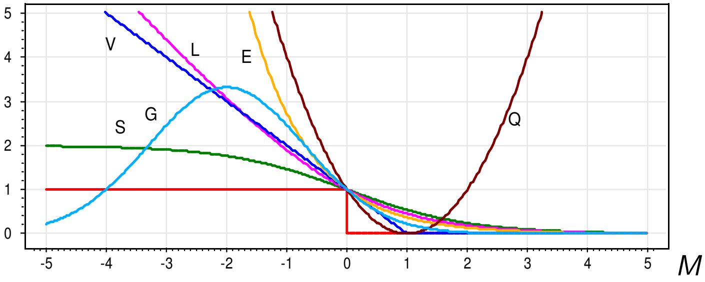
Функционал качества композиции — число ошибок на X^ℓ:

$$Q_T = \sum_{i=1}^{\ell} \left[y_i \sum_{t=1}^{T} \alpha_t b_t(x_i) < 0 \right].$$

Две основные эвристики бустинга:

- фиксация $\alpha_1 b_1(x), \ldots, \alpha_{t-1} b_{t-1}(x)$ при добавлении $\alpha_t b_t(x)$;
- гладкая аппроксимация пороговой функции потерь $[M \leqslant 0]$.

Гладкие аппроксимации пороговой функции потерь $[M < 0]$



$E(M) = e^{-M}$ — экспоненциальная (AdaBoost);
$L(M) = \log_2(1 + e^{-M})$ — логарифмическая (LogitBoost);
$Q(M) = (1 - M)^2$ — квадратичная (GentleBoost);
$G(M) = \exp\bigl(-cM(M + s)\bigr)$ — гауссовская (BrownBoost);
$S(M) = 2(1 + e^M)^{-1}$ — сигмоидная;
$V(M) = (1 - M)_+$ — кусочно-линейная (из SVM);

Градиентный бустинг для произвольной функции потерь

Линейная (выпуклая) комбинация базовых алгоритмов:

$$a(x) = \sum_{t=1}^{T} \alpha_t b_t(x), \quad x \in X, \quad \alpha_t \in \mathbb{R}_+.$$

Функционал качества с произвольной функцией потерь $\mathscr{L}(a, y)$:

$$Q(\alpha, b; X^\ell) = \sum_{i=1}^{\ell} \mathscr{L}\Bigl(\underbrace{\underbrace{\sum_{t=1}^{T-1} \alpha_t b_t(x_i)}_{f_{T-1,i}} + \alpha b(x_i)}_{f_{T,i}}, y_i\Bigr) \to \min_{\alpha, b}.$$

$f_{T-1} = \bigl(f_{T-1,i}\bigr)_{i=1}^{\ell}$ — текущее приближение
$f_T = \bigl(f_{T,i}\bigr)_{i=1}^{\ell}$ — следующее приближение

Friedman G. Greedy Function Approximation: A Gradient Boosting Machine. 1999.

Параметрическая аппроксимация градиентного шага

Градиентный метод минимизации $Q(f) \to \min$, $f \in \mathbb{R}^\ell$:

$$f_0 := \text{начальное приближение;}$$
$$f_{T,i} := f_{T-1,i} - \alpha g_i, \quad i = 1, \dots, \ell;$$

$g_i = \mathscr{L}'\big(f_{T-1,i}, y_i\big)$ — компоненты вектора градиента,
α — градиентный шаг.

Наблюдение: это очень похоже на одну итерацию бустинга!

$$f_{T,i} := f_{T-1,i} + \alpha b(x_i), \quad i = 1, \dots, \ell$$

Идея: будем искать такой базовый алгоритм b_T, чтобы вектор $\big(b_T(x_i)\big)_{i=1}^\ell$ приближал вектор антиградиента $\big(-g_i\big)_{i=1}^\ell$:

$$b_T := \arg\max_b \sum_{i=1}^{\ell} \big(b(x_i) + g_i\big)^2$$

Алгоритм градиентного бустинга (Gradient Boosting)

Вход: обучающая выборка X^ℓ; параметр T;

Выход: базовые алгоритмы и их веса $\alpha_t b_t$, $t = 1, \ldots, T$;

1: инициализация: $f_i := 0$, $i = 1, \ldots, \ell$;

2: **для всех** $t = 1, \ldots, T$

3: найти базовый алгоритм, приближающий градиент:

$$b_t := \arg\min_b \sum_{i=1}^{\ell} \bigl(b(x_i) + \mathscr{L}'(f_i, y_i)\bigr)^2;$$

4: решить задачу одномерной минимизации:

$$\alpha_t := \arg\min_{\alpha > 0} \sum_{i=1}^{\ell} \mathscr{L}\bigl(f_i + \alpha b_t(x_i), y_i\bigr);$$

5: обновить значения композиции на объектах выборки:

$f_i := f_i + \alpha_t b_t(x_i)$; $i = 1, \ldots, \ell$;

Стохастический градиентный бустинг (SGB)

Идея: на шагах 3–5 использовать не всю выборку X^ℓ, а случайную подвыборку без возвращений

Преимущества:

- улучшается качество
- улучшается сходимость
- уменьшается время обучения

Friedman G. Stochastic Gradient Boosting. 1999.

Алгоритм XGBoost

Вычисления шага с помощью методов второго порядка:

$$b_N = \arg\min_{b \in A} \sum_{i=1}^{\ell} \left(b(x_i) - \frac{s_i}{h_i} \right)^2$$

$$s = \left(-\frac{\partial L}{\partial z} \Big|_{z = a_{N-1}(x_i)} \right)_{i=1}^{\ell}, \quad h_i = \frac{\partial^2 L}{\partial z^2} \Big|_{z = a_{N-1}(x_i)}$$

Рассмотрим функцию L:

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

Отбросим первое слогаемое:

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

Алгоритм XGBoost

Базовый алгоритм — дерево $b(x)$, описывается формулой:

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

Сложность зависет от:

1. число листьем J
2. норма коэффицентов в листьях $\|b\|^2 = \sum_{j=1}^{J} b_j^2$

Добавим регуляризаторы, оштрафовав сложность:

$$L_1 = \sum_{i=1}^{\ell} \Big(-s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \Big) + \lambda J + \frac{\mu}{2} \sum_{j=1}^{J} b_j^2 \to \min_b$$

Алгоритм XGBoost

Дерево $b(x)$ дает одинаковые ответы на объектах:

$$L_1 = \sum_{j=1}^{J}\left\{\left(-\sum_{i\in R_j} s_i\right)b_j + \frac{1}{2}\left(\mu + \sum_{i\in R_j} h_i\right)b_j^2 + \lambda\right\}$$

Обозначим:

$$-S_j = -\sum_{i\in R_j} s_i, \quad H_j = \sum_{i\in R_j} h_i$$

Оптимальные коэффиценты и соответствующая ошибка дерева:

$$b_j = \frac{S_j}{H_j + \mu}, \quad H(b) = \frac{1}{2}\sum_{j=1}^{J}\frac{S_j^2}{H_j + \mu} + \lambda J$$

Критерий разбиения для дерева:

$$Q = H(b_l) + H(b_r) - H(b) - \lambda \to \max$$

Алгоритм XGBoost

- Базовый алгоритм приближает направление, посчитанное с учетом вторых производных функции потерь.
- Отклонение направления, построенного базовым алгоритмом, измеряется с помощью модифицированного функционала — из него удалено деление на вторую производную, за счет чего избегаются численные проблемы.
- Функционал регуляризуется: добавляются штрафы за количество листьев и за норму коэффициентов.
- При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига.
- Критерий останова при обучении дерева зависит от оптимального сдвига.

- Градиентный бустинг — наиболее общий из всех бустингов:
  — произвольная функция потерь
  — произвольное пространство оценок R
  — подходит для регрессии, классификации, ранжирования
- Стохастический вариант SGB — лучше и быстрее
- Чаще всего GB применяется к решающим деревьям
- Градиентный бустинг над ODT = Yandex.MatrixNet

Несколько эмпирических наблюдений:

- Веса алгоритмов не столь важны для выравнивания отступов.
- Веса объектов не столь важны для обеспечения различности.
- Не удаётся строить короткие композиции из «сильных» алгоритмов типа SVM (только длинные из слабых).

Почему бустинг и бэггинг работают?

- бэггинг уменьшает разброс
- бустинг уменьшает и смещение, и разброс
- композиции тем менее эффективны, чем сильнее коррелируют базовые алгоритмы
- случайный лес уменьшает корреляции базовых алгоритмов