Санкт-Петербургский государственный университет

Прикладная математика и информатика

Кафедра статистического моделирования

Композиция методов. Бустинг.

Романова Елизавета

Горбачук Анна

Сидоренко Денис

Санкт-Петербург

2019

Оглавление

Введение

Пусть мы всё также находимся в рамках терминов задачи обучения с учителем. То есть в общем случае также наблюдается некоторый отклик Y, который в случае классификации принимает значения из $\{1, \ldots, k\}$, где k — число классов. А в случае регрессии принимает вещественные значения. Также у нас имеется p признаков $X_1, \ldots, X_p$. По предположению между признаками и откликами есть некоторая зависимость, которую требуется найти.

К сожалению в ряде задач, имеющими сложные нелинейности часто оказывается, что одиночные алгоритмы восстановления функции зависимости между признаками и откликом не дают требуемого качества.

В своё время возникла простая идея композиции алгоритмов, которая сначала получила эмпирическое, а затем и теоретическое подтверждение.

1. Постановка задачи

В общем определении, следуя стандартной постановке задаче, в бустинге вводят дополнительное пространство оценок R, которое работает только с алгоритмами, представленными в виде: $a(x) = C(b(x))$, где функция $b : X \to R$ является алгоритмическим оператором, а функция $C : R \to Y$ — решающим правилом. Введём изменённую по отношению к стандартной постановку задачи.

Определение 1. *Композицией T алгоритмов $a_t(x) = C(b_t(x)), t = 1, \ldots, T$ называется суперпозиция алгоритмических операторов $b_t : X \to R$, корректирующей операции $F : R^{\mathrm{T}} \to R$ и решающего правила*

$$C : R \to Y : a(x) = C(F(b_1(x), \ldots, b_T(x))), x \in X$$

Замечание 1. *Алгоритмы $a_1(x), \ldots, a_T(x)$ (а иногда и операторы b_t) называются базовыми. А $\mathcal{B}(\Theta) = \{b(.;\theta)|\theta \in \Theta\}$ — параметризованное множество базовых алгоритмов. Выбор базового алгоритма: выбор такого $\theta \in \Theta$ и $b(x) = b(x;\theta) \in \mathcal{B}$.*

Корректирующее правило $F : Y^{\mathrm{T}} \to Y$ может комбинировать как ответы алгоритмов, так и в случае классификации "оценки"принадлежности определённым классам, а в случае одних и тех же алгоритмов — усреднённые веса по алгоритму.

Как правило в качестве базовых алгоритмов выступают: 1) Решающие деревья(неглубокие(2-8)) — используются чаще всего; 2) Пороговые правила(data stumps); 3) Решающие "пни"(дерево глубины 1) на некоторых задачах дают результаты значительно выше своих более глубоких конкурентов.

Задача: Подбор оптимальных(в смысле рассматриваемой функции потерь) базовых алгоритмов $\{b_t(x)\}_{t=1}^T$.

2. Примеры

2.1. Примеры пространств оценок

1. В задаче классификации на два класса: $Y = \{-1, 1\}$ в качестве пространства оценок обычно используется множество действительных чисел $R = \mathbb{R}$. В этом случае алгоритмические операторы называются вещественнозначными классификаторами: $C(b(x)) = \operatorname{sign} b(x)$. Пояснение: в данном случае предполагается, что ответы базовых алгоритмов больше 0 соответствует 1, а меньше 0 — -1.

2. В задаче классификации на M классов, $Y = \{1, \ldots, M\}$, в качестве подпространства оценок обычно используется $R = \mathbb{R}^M$. Алгоритмический оператор $b(x)$ выдаёт вектор оценок принадлежности объекта x каждому из классов, $b(x) = b^1(x), \ldots, b^M(x)$. Решающее правило C относит объект к тому классу, для которого оценка максимальна: $C(b(x)) === C(b^1(x), \ldots, b^M(x)) = \operatorname{argmax}_{y \in Y} b^y(x)$ Пояснение: такой вид имеют байесовские алгоритмы классификации и метрические алгоритмы.

3. В задачах регрессии множество Y уже достаточно богато, обычно $Y = \mathbb{R}$, поэтому использовать решающее правило нет особого смысла. В этом случае обычно полагают $R = \mathbb{R}$, $C(b) \equiv b$.

4.

2.2. Примеры корректирующих операций

- Простое голосование (Simple Voting):

$$F(b_1(x), \ldots, b_T(x)) = \frac{1}{T} \sum_{t=1}^{T} b_t(x), \quad x \in X.$$

Пояснение: по сути означает усреднение всех предсказаний при регрессии или классификации, если ответ алгоритма - вероятность. Если ответ алгоритма классификации класс, то берётся класс, за который наиболее часто голосовали базовые алгоритмы.

- Взвешенное голосование (Weighted Voting):

$$F(b_1(x), \ldots, b_T(x)) = \sum_{t=1}^{T} \alpha_t b_t(x), \quad x \in X, \quad \alpha_t \in R.$$

Пояснение: имеет смысл только, когда в качестве голосований используются разные алгоритмы.

- Смесь алгоритмов (Mixture of Experts):

$$F(b_1(x), \ldots, b_T(x)) = \sum_{t=1}^{T} g_t(x) b_t(x), \quad x \in X, \quad g_t : X \to \mathbb{R}.$$

2.3. Взвешенное голосование

Корректирующая операция F может иметь параметры, настраиваемые по обучающей выборке, наряду с параметрами базовых алгоритмов. Например, в линейной комбинации настраиваются веса α_t базовых алгоритмов:

$$b(x) = F(b_1(x), \ldots, b_T(x)) = \sum_{t=1}^{T} \alpha_t b_t(x), \quad x \in X, \quad \alpha_t \in R. \tag{1}$$

Если веса α_t неотрицательны и нормированы, $\sum_{t=1}^{T} \alpha_t = 1$, то композицию (1) называют **выпуклой комбинацией** базовых алгоритмов.

В задачах классификации корректирующая операция (1) называется **взвешенным голосованием** (weighted voting).

2.4. Пример с нейронной сетью

Рассмотрим задачу классификации с двумя классами$Y = \{-1, +1\}$ и двухслойную нейронную сеть, принимающую на входе n-мерный вектор признаков $x = (x^1, \ldots, x^n), x \in X = \mathbb{R}$:

$$a(x) = \text{sign}(\sum_{t=1}^{T} w_t \sigma(\sum_{j=1}^{n} w_{jt} x^j))$$

, где T — число нейронов скрытого слоя, w_{jt} — вес связи между j–м признаком и t–м нейроном скрытого слоя, w_t–вес связи между t–м нейроном скрытого слоя и выходным

нейроном, σ – функция активации. Такая сеть может рассматриваться как композиция алгоритмических операторов вида $b_t(x) = \sigma(\sum_{j=1}^{n} w_{jt}x^j)$ с линейной корректирующей операцией $F(b_1, \ldots, b_T) = \sum_{t=1}^{T} w_t b_t$ и решающим правилом $C(b) = \operatorname{sign} b$.

2.5. Бустинг в задачах классификации

Рассмотрим задачу классификации на два класса, $Y = \{-1, +1\}$. Допустим, что решающее правило фиксировано, $C(b) = sign(b)$, базовые алгоритмы возвращают ответы $-1, 0, +1$.

Ответ $b_t(x) = 0$ означает, что базовый алгоритм b_t отказывается от классификации объекта x, и ответ $b_t(x)$ не учитывается в композиции.

Искомая алгоритмическая композиция имеет вид:

$$a(x) = C(F(b_1(x), \ldots, b_T(x)) = sign\left(\sum_{t=1}^{T} \alpha_t b_t(x)\right), \quad x \in X. \tag{2}$$

Определим функционал качества композиции как число ошибок, допускаемых ею на обучающей выборке:

$$Q_T = \sum_{i=1}^{n} \left[y_i \sum_{t=1}^{T} \alpha_t b_t(x_i) < 0 \right]. \tag{3}$$

Для упрощения задачи минимизации функционала Q_T введём две эвристики (не полностью математически обоснованные, но при этом практически полезные алгоритмы).

Эвристика 1. При добавлении в композицию слагаемого $\alpha_t b_t(x)$ оптимизируется только базовый алгоритм b_t и коэффициент при нём α_t, а все предыдущие слагаемые $\alpha_1 b_1(x), \ldots, \alpha_{t-1} b_{t-1}(x)$ полагаются фиксированными.

Эвристика 2. Пороговая функция потерь в функционале Q_t аппроксимируется (заменяется) непрерывно дифференцируемой оценкой сверху.

Вторая эвристика широко используется в теории классификации.

2.6. Аппроксимации

Прежде чем приступать к описанию алгоритма Adaboost следует сказать, что существует множество гладких(дифференцируемых) аппроксимаций пороговой функции потерь. Некоторые примеры:

1. $S(z) = 2(1 + \exp^z)^{-1}$ — сигмоидная;
2. $L(z) = \log_2(1 + \exp^{-z})$ — логарифмическая;
3. $(1 - z)_+$ —кусочно–линейная;
4. $\exp^{-z}$ —экспоненциальная;
5. $(1 - z^2)$ —квадратичная.

3. Adaboost

При использовании экспоненциальной аппроксимации $[y_i b(x_i) < 0] \leq e^{?y_i b(x_i)}$ эти две эвристики приводят к алгоритму Adaboost.

Оценим функционал Q_T сверху:

$$Q_T \leq \widetilde{Q}_T = \sum_{i=1}^{n} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t b_t(x_i)\right) =$$

$$= -\sum_{i=1}^{n} \underbrace{\exp\left(-y_i \sum_{t=1}^{T} \alpha_t b_t(x_i)\right)}_{\omega_i} e^{?y_i \alpha_T b_T(x_i)}.$$

Заметим, что введённые здесь веса объектов ω_i не зависят от $\alpha_T b_T$ и могут быть вычислены перед построением базового алгоритма b_T.

Введём вектор нормированных весов $\widetilde{W}^n = \widetilde{\omega}_1, \ldots, \widetilde{\omega}_n$, где $\widetilde{\omega}_i = \omega_i / \sum_{j=1}^{n} \omega_j$.

Определим два функционала качества алгоритма классификации b на обучающей выборке $X^n = (x_i, y_i)_{i=1}^n$ с нормированным вектором весов объектов $U^n = (u_1, \ldots, u_n)$: суммарный вес ошибочных (negative) классификаций $N(b; U^n)$ и суммарный вес правильных (positive) классификаций $P(b; U^n)$:

$$N(b; U^n) = \sum_{i=1}^{n} u_i [b(x_i) = -y_i],$$

$$P(b; U^n) = \sum_{i=1}^{n} u_i [b(x_i) = y_i].$$

Заметим, что $1 - N - P$ есть суммарный вес отказов от классификации. Если отказов нет, то $N + P = 1$.

Теорема Гаусса-Маркова 1. *Пусть $\mathcal{B}$ — достаточно богатое семейство базовых алгоритмов. Пусть для любого нормированного вектора весов U^n существует алгоритм $b \in \mathcal{B}$, классифицирующий выборку хотя бы немного лучше, чем наугад: $P(b;U^n) > N(b;U^n)$.*

Тогда минимум функционала $\widetilde{Q}_T$ достигается при

$$b_T = \arg\max_{b\in\mathcal{B}} \sqrt{P(b;\widetilde{W}^n)} - \sqrt{N(b;\widetilde{W}^n)},$$

$$a_t = \frac{1}{2}\ln\frac{P(b_t;\widetilde{W}^n)}{N(b_t;\widetilde{W}^n)}.$$

3.1. Алгоритм

Вход: $X^n = (x_i, y_i)_{i=1}^n$ - обучающая выборка, T - максимальное число базовых алгоритмов.

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

1. инициализация весов объектов:

   $\omega_i := 1/n$, $i = 1, \ldots, n$;

2. для всех $t = 1, \ldots, T$, пока не выполнен критерий остановки:

3. обучить базовый алгоритм:

   $b_t := \arg\min_{b\in\mathcal{B}} N(b; W^n)$;

4. $a_t := \frac{1}{2}\ln\frac{1-N(b_t;W^n)}{N(b_t;W^n)}$;

5. пересчет весов объектов:

   $\omega_i := \omega_i e^{?\alpha_t y_i b_t(x_i)}$, $i = 1, \ldots, n$;

6. нормировка весов объектов:

   $\omega_0 := \sum_{j=1}^n \omega_j$; $\omega_i := \omega_i/\omega_0$, $i = 1, \ldots, n$.

3.2. Достоинства и недостатки

Достоинства

- Хорошая обобщающая способность. В реальных задачах (не всегда, но часто) удаётся строить композиции, превосходящие по качеству базовые алгоритмы. Обобщающая способность может улучшаться (в некоторых задачах) по мере увеличения числа базовых алгоритмов.
- Простота реализации.
- Накладные расходы бустинга невелики. Время построения композиции практически полностью определяется временем обучения базовых алгоритмов.
- Возможность идентифицировать выбросы. Это "наиболее трудные"объекты x_i, для которых в процессе наращивания композиции веса ω_i принимают наибольшие значения.

Недостатки AdaBoost.

- AdaBoost склонен к переобучению при наличии значительного уровня шума в данных.
- AdaBoost требует достаточно длинных обучающих выборок.
- Бустинг может приводить к построению громоздких композиций, состоящих из сотен алгоритмов. Такие композиции исключают возможность содержательной интерпретации, требуют больших объёмов памяти и существенных затрат времени.
- Жадная стратегия последовательного добавления приводит к построению неоптимального набора базовых алгоритмов.

4. AnyBoost

Возьмём $Y = \{-1; +1\}$, $b_t : X \to \mathbb{R}$, $C(b) = sign(b)$;

$\mathrm{L}(M)$ — функция потерь, гладкая функция отступа M;

$M_T(x_i) = y_i \sum_{t=1}^{T} \alpha_t b_t(x_i)$ — отступ композиции на объекте x_i;

Оценка сверху для числа ошибок композиции:

$$Q_T \leq \widetilde{Q}_T = \sum_{i=1}^{n} \mathrm{L}(M_{T-1}(x_i) + y_i \alpha_T b_T(x_i)) \to \min_{\alpha, b \in \mathcal{B}}.$$

Рассмотрим функцию потерь L как функцию параметра α_T,

$$\lambda(\alpha_T) = \mathrm{L}(M_{T-1}(x_i) + y_i \alpha_T b_T(x_i))$$

и линеаризуем её в окрестности значения $\alpha_T = 0$, разложив в ряд Тейлора и отбросив старшие члены: $\lambda(\alpha_T) \approx \lambda(0) + \alpha_T \lambda'(0)$.

Это приведет к линеаризация функционала $\widetilde{Q}_T$ по α_T:

$$\widetilde{Q}_T \approx \sum_{i=1}^{n} \mathrm{L}(M_{T-1}(x_i)) - \alpha \sum_{i=1}^{n} \underbrace{-\mathrm{L}'(M_{T-1}(x_i))}_{\omega_i} y_i b(x_i) \to \min_{b \in \mathcal{B}},$$

где w_i — веса объектов.

4.1. Принцип явной максимизации отступов.

Минимизация линеаризованного $\widetilde{Q}_T$ при фиксированном α:

$$\widetilde{Q}_T \approx \sum_{i=1}^{n} \mathrm{L}(M_{T-1}(x_i)) - \alpha \sum_{i=1}^{n} \omega_i y_i b(x_i) \to \min_{b \in \mathcal{B}}$$

приводит к принципу явной максимизации отступов (direct optimization of margin, DOOM):

$$\sum_{i=1}^{n} \omega_i y_i b(x_i) \to \max_{b \in \mathcal{B}}.$$

Затем α определяется путём одномерной минимизации $\widetilde{Q}_T$.

Итерации этих двух шагов приводят к алгоритму AnyBoost.

Замечание. AnyBoost переходит в AdaBoost в частном случае, при $b_t : X \to \{-1, 0, +1\}$ и $\mathrm{L}(M) = e^{-M}$.

Вход: $X^n = (x_i, y_i)_{i=1}^n$ - обучающая выборка, T - максимальное число базовых алгоритмов.

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

1. инициализация отступов: $M_i := 0$, $i = 1, \ldots, n$;
2. для всех $t = 1, \ldots, T$, пока не выполнен критерий остановки:
3. вычислить веса объектов:

$\omega_i = -\mathrm{L}'(M_i)$, $i = 1, \ldots, n$;

4. обучить базовый алгоритм согласно принципу DOOM: $b_t := \arg\max_{b \in \mathcal{B}} \sum_{i=1}^{n} \omega_i y_i b(x_i)$;

5. решить задачу одномерной минимизации: $a_t := \arg\max_{\alpha} \sum_{i=1}^{n} \mathrm{L}(M_i + \alpha b_t(x_i) y_i)$;

6. пересчет отступов:

$M_i := M_i + \alpha b_t(x_i) y_i;\ i = 1, \ldots, n.$

5. Градиентный бустинг

Ранее мы изучили бэггинг и случайные леса — подходы к построению композиций, которые независимо обучают каждый базовый алгоритм по некоторому подмножеству обучающих данных. При этом возникает ощущение, что мы используем возможности объединения алгоритмов не в полную силу, и можно было бы строить их так, чтобы каждая следующая модель исправляла ошибки предыдущих. Даже после рассмотренных AdaBoost и AnyBoost это ощущение не пропадало. Ниже мы рассмотрим метод, который реализует эту идею — градиентный бустинг, предложенный Фридманом [1]. Этот подход является наиболее популярным методом "из коробки"на сегодняшний день.

6. Бустинг в задаче регрессии

Рассмотрим задачу минимизации квадратичного функционала:

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

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

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

где базовые алгоритмы b_n принадлежат некоторому семейству $\mathbb{A}$.

Построим первый базовый алгоритм:

$$b_1(x) := \mathrm{argmin}_{b \in \mathbb{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - y_i)^2$$

Решение такой задачи не представляет трудностей для многих семейств алгоритмов. Теперь мы можем посчитать остатки на каждом объекте — расстояния от ответа нашего алгоритма до истинного ответа:

$$s_i^{(1)} = y_i - b_1(x_i)$$

Если прибавить эти остатки к ответам построенного алгоритма, то он не будет допускать ошибок на обучающей выборке. Значит, будет разумно построить второй алгоритм так, чтобы его ответы были как можно ближе к остаткам:

$$b_2(x) := \operatorname{argmin}_{b \in \mathbb{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - s_i^{(1)})^2$$

Каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:

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

$$b_N(x) := \operatorname{argmin}_{b \in \mathbb{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - s_i^{(N)})^2$$

Описанный метод прост в реализации, хорошо работает и может быть найден во многих библиотеках — например, в `scikit-learn`.

Заметим, что остатки могут быть найдены как антиградиент функции потерь по ответу модели, посчитанный в точке ответа уже построенной композиции:

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

Получается, что выбирается такой базовый алгоритм, который как можно сильнее уменьшит ошибку композиции — это свойство вытекает из его близости к антиградиенту функционала на обучающей выборке. Попробуем разобраться с этим свойством подробнее, а также попытаемся обобщить его на другие функции потерь.

7. Градиентный бустинг

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

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

Заметим, что в композиции имеется начальный алгоритм $b_0(x)$. Как правило, коэффициент γ_0 при нем берут равным единице, а сам алгоритм выбирают очень простым, например:

- нулевым $b_0(x) = 0$;

- возвращающим самый популярный класс (в задачах классификации):

$$b_0(x) = \text{argmax}_{y \in \mathbb{Y}} \sum_{i=1}^{\ell} [y_i = y]$$

- возвращающим средний ответ (в задачах регрессии):

$$b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$$

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

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

Ответим в первую очередь на следующий вопрос: если бы в качестве алгоритма $b_N(x)$ мы могли выбрать совершенно любую функцию, то какие значения ей следовало бы принимать на объектах обучающей выборки? Иными словами, нам нужно понять, какие числа $s_1, \ldots, s_\ell$ надо выбрать для решения следующей задачи:

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

Понятно, что можно требовать $s_i = y_i - a_{N-1}(x_i)$, но такой подход никак не учитывает особенностей функции потерь $L(y, z)$ и требует лишь точного совпадения предсказаний и истинных ответов. Более разумно потребовать, чтобы сдвиг s_i был противоположен производной функции потерь в точке $z = a_{N-1}(x_i)$:

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

В этом случае мы сдвинемся в сторону скорейшего убывания функции потерь. Заметим, что вектор сдвигов $s = (s_1, \ldots, s_\ell)$ совпадает с антиградиентом:

$$\left(-\left.\frac{\partial L}{\partial z}\right|_{z = a_{N-1}(x_i)} \right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i)\big|_{z_i = a_{N-1}(x_i)}$$

При таком выборе сдвигов s_i мы, по сути, сделаем один шаг градиентного спуска, двигаясь в сторону наискорейшего убывания ошибки на обучающей выборке. Отметим, что речь идет о градиентном спуске в ℓ-мерном пространстве предсказаний алгоритма на объектах обучающей выборки. Поскольку вектор сдвига будет свой на каждой

итерации, правильнее обозначать его как $s_i^{(N)}$, но для простоты будем иногда опускать верхний индекс.

Итак, мы поняли, какие значения новый алгоритм должен принимать на объектах обучающей выборки. По данным значениям в конечном числе точек необходимо построить функцию, заданную на всем пространстве объектов. Это классическая задача обучения с учителем, которую мы уже хорошо умеем решать. Один из самых простых функционалов — среднеквадратичная ошибка. Воспользуемся им для поиска базового алгоритма, приближающего градиент функции потерь на обучающей выборке:

$$b_N(x) = \text{argmin}_{b \in \mathbb{A}} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2$$

Отметим, что здесь мы оптимизируем квадратичную функцию потерь независимо от функционала исходной задачи — вся информация о функции потерь L находится в антиградиенте s_i, а на данном шаге лишь решается задача аппроксимации функции по ℓ точкам. Разумеется, можно использовать и другие функционалы, но среднеквадратичной ошибки, как правило, оказывается достаточно. Ещё одна причина для использования среднеквадратичной ошибки состоит в том, что от алгоритма требуется как можно точнее приблизить направление наискорейшего убывания функционала (то есть направление $(s_i)_i$); совпадение направлений вполне логично оценивать через косинус угла между ними, который напрямую связан со среднеквадратичной ошибкой.

После того, как новый базовый алгоритм найден, можно подобрать коэффициент при нем по аналогии с наискорейшим градиентным спуском:

$$\gamma_N = \text{argmin}_{\gamma \in \mathbb{R}} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma b_N(x_i))$$

Описанный подход с аппроксимацией антиградиента базовыми алгоритмами и называется градиентным бустингом. Данный метод представляет собой поиск лучшей функции, восстанавливающей истинную зависимость ответов от объектов, в пространстве всех возможных функций. Ищем мы данную функцию с помощью «псевдоградиентного» спуска — каждый шаг делается вдоль направления, задаваемого некоторым базовым алгоритмом. При этом сам базовый алгоритм выбирается так, чтобы как можно лучше приближать антиградиент ошибки на обучающей выборке.

8. Регуляризация

Сокращение шага. На практике оказывается, что градиентный бустинг очень быстро строит композицию, ошибка которой на обучении выходит на асимптоту, после чего начинает настраиваться на шум и переобучаться. Это явление можно объяснить одной из двух причин:

- Если базовые алгоритмы очень простые (например, решающие деревья небольшой глубины), то они плохо приближают вектор антиградиента. По сути, добавление такого базового алгоритма будет соответствовать шагу вдоль направления, сильно отличающегося от направления наискорейшего убывания. Соответственно, градиентный бустинг может свестись к случайному блужданию в пространстве.

- Если базовые алгоритмы сложные (например, глубокие решающие деревья), то они способны за несколько шагов бустинга идеально подогнаться под обучающую выборку — что, очевидно, будет являться переобучением, связанным с излишней сложностью семейства алгоритмов.

Хорошо зарекомендовавшим себя способом решения данной проблемы является *сокращение шага*: вместо перехода в оптимальную точку в направлении антиградиента делается укороченный шаг

$$a_N(x) = a_{N-1}(x) + \eta\gamma_N b_N(x),$$

где $\eta \in (0, 1]$ — темп обучения [1]. Как правило, чем меньше темп обучения, тем лучше качество итоговой композиции. Сокращение шага, по сути, позволяет понизить доверие к направлению, восстановленному базовым алгоритмом.

Также следует обратить внимание на число итераций градиентного бустинга. Хотя ошибка на обучении монотонно стремится к нулю, ошибка на контроле, как правило, начинает увеличиваться после определенной итерации. Оптимальное число итераций можно выбирать, например, по отложенной выборке или с помощью кросс-валидации.

Стохастический градиентный бустинг. Еще одним способом улучшения качества градиентного бустинга является внесение рандомизации в процесс обучения базовых алгоритмов [2]. А именно, алгоритм b_N обучается не по всей выборке X, а лишь по ее случайному подмножеству $X^k \subset X$. В этом случае понижается уровень шума в

обучении, а также повышается эффективность вычислений. Существует рекомендация брать подвыборки, размер которых вдвое меньше исходной выборки.

9. Функции потерь

9.1. Регрессия

При вещественном целевом векторе, как правило, используют квадратичную функцию потерь, формулы для которой уже были приведены в разделе 6. Другой вариант — модуль отклонения $L(y,z) = |y - z|$, для которого антиградиент вычисляется по формуле

$$s_i^{(N)} = -\operatorname{sign}(a_{N-1}(x_i) - y_i).$$

9.2. Классификация

В задаче классификации с двумя классами разумным выбором является логистическая функция потерь, с которой уже сталкивались при изучении линейных методов:

$$L(y,z) = \log(1 + \exp(-yz)).$$

Задача поиска базового алгоритма с ней принимает вид

$$b_N = \operatorname{argmin}_{b \in \mathbb{A}} \sum_{i=1}^{\ell} \left(b(x_i) - \frac{y_i}{1 + \exp(y_i a_{N-1}(x_i))} \right)^2.$$

Логистическая функция потерь имеет интересную особенность, связанную со взвешиванием объектов. Заметим, что ошибка на N-й итерации может быть записана как

$$\begin{aligned} Q(a_N) &= \sum_{i=1}^{\ell} \log\left(1 + \exp(-y_i a_N(x_i))\right) = \\ &= \sum_{i=1}^{\ell} \log\left(1 + \exp(-y_i a_{N-1}(x_i)) \exp(-y_i \gamma_N b_N(x_i))\right). \end{aligned}$$

Если отступ $y_i a_{N-1}(x_i)$ на i-м объекте большой положительный, то данный объект не будет вносить практически никакого вклада в ошибку, и может быть исключен из всех вычислений на текущей итерации без потерь. Таким образом, величина

$$w_i^{(N)} = \exp(-y_i a_{N-1}(x_i))$$

может служить мерой важности объекта x_i на N-й итерации градиентного бустинга.

10. Градиентный бустинг над деревьями

Считается, что градиентный бустинг над решающими деревьями — один из самых универсальных и сильных методов машинного обучения, известных на сегодняшний день. В частности, на градиентном бустинге над деревьями основан MatrixNet — алгоритм ранжирования компании Яндекс [3].

Вспомним, что решающее дерево разбивает все пространство на непересекающиеся области, в каждой из которых его ответ равен константе:

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

где $j = 1, \ldots, J_n$ — индексы листьев, R_j — соответствующие области разбиения, b_{nj} — значения в листьях. Значит, на N-й итерации бустинга композиция обновляется как

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

Видно, что добавление в композицию одного дерева с J_N листьями равносильно добавлению J_N базовых алгоритмов, представляющих собой предикаты вида $[x \in R_j]$. Если бы вместо общего коэффициента γ_N был свой коэффициент γ_{Nj} при каждом предикате, то мы могли бы его подобрать так, чтобы повысить качество композиции. Если подбирать свой коэффициент γ_{Nj} при каждом слагаемом, то потребность в b_{Nj} отпадает, его можно просто убрать:

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

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

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

В некоторых случаях оптимальные коэффициенты могут быть найдены аналитически — например, для квадратичной и абсолютной ошибки.

Рассмотрим теперь логистическую функцию потерь. В этом случае нужно решить задачу

$$F_j^{(N)}(\gamma) = \sum_{x_i \in R_j} \log\left(1 + \exp\left(-y_i(a_{N-1}(x_i) + \gamma)\right)\right) \to \min_\gamma.$$

Данная задача может быть решена лишь с помощью итерационных методов, аналитической записи для оптимального γ не существует. Однако на практике обычно нет необходимости искать точное решение — оказывается достаточным сделать лишь один шаг метода Ньютона-Рафсона из начального приближения $\gamma_{Nj} = 0$. Можно показать, что в этом случае

$$\gamma_{Nj} = \frac{\partial F_j^{(N)}(0)}{\partial \gamma} \Bigg/ \frac{\partial^2 F_j^{(N)}(0)}{\partial \gamma^2} = -\sum_{x_i \in R_j} s_i^{(N)} \Bigg/ \sum_{x_i \in R_j} |s_i^{(N)}|(1 - |s_i^{(N)}|).$$

Смещение и разброс. В случайных лесах используются глубокие деревья, поскольку от базовых алгоритмов требуется низкое смещение; разброс же устраняется за счёт усреднения ответов различных деревьев. Бустинг работает несколько иначе — в нём каждый следующий алгоритм целенаправленно понижает ошибку композиции, и даже при использовании простейших базовых моделей композиция может оказаться достаточно сложной. Более того, итоговая композиция вполне может оказаться переобученной при большом количестве базовых моделей. Это означает, что благодаря бустингу можно понизить смещение моделей, а разброс либо останется таким же, либо увеличится. Из-за этого, как правило, в бустинге используются неглубокие решающие деревья (3-6 уровней), которые обладают большим смещением, но не склонны к переобучению.

11. Взвешивание объектов

Одним из первых широко распространённых методов построения композиций является AdaBoost, в котором оптимизируется экспоненциальная функция потерь $L(y, z) = e^{-yz}$. Благодаря её свойствам удаётся свести задачу поиска базового алгоритма к минимизации доли неверных ответов с весами при объектах. Эти веса возникают и в градиентном бустинге при использовании экспоненциальной функции потерь:

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

Найдем компоненты ее антиградиента после $(N-1)$-й итерации:

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

Заметим, что антиградиент представляет собой ответ на объекте, умноженный на его вес. Если все веса будут равны единице, то следующий базовый классификатор будет просто настраиваться на исходный целевой вектор $(y_i)_{i=1}^{\ell}$; штраф за выдачу ответа, противоположного правильному, будет равен 4 (поскольку при настройке базового алгоритма используется квадратичная функция потерь). Если же какой-либо объект будет иметь большой отступ, то его вес окажется близким к нулю, и штраф за выдачу любого ответа будет равен 1.

Отметим, что многие функционалы ошибки классификации выражаются через отступы объектов:

$$L(a_{N-1}, X^{\ell}) = \sum_{i=1}^{\ell} L(a_{N-1}(x_i), y_i) = \sum_{i=1}^{\ell} \tilde{L}(y_i a_{N-1}(x_i)).$$

В этом случае антиградиент принимает вид

$$s_i = y_i \underbrace{\left(-\frac{\partial \tilde{L}(y_i a_{N-1}(x_i))}{\partial a_{N-1}(x_i)} \right)}_{w_i},$$

то есть тоже взвешивает ответы с помощью ошибки на них.

12. Влияние шума на обучение

Выше мы находили формулу для антиградиента при использовании экспоненциальной функции потерь:

$$s_i = y_i \underbrace{\exp\left(-y_i \sum_{n=1}^{N-1} \gamma_n b_n(x_i) \right)}_{w_i}.$$

Заметим, что если отступ на объекте большой и отрицательный (что обычно наблюдается на шумовых объектах), то вес становится очень большим, причем он никак не ограничен сверху. В результате базовый классификатор будет настраиваться исключительно на шумовые объекты, что может привести к неустойчивости его ответов и переобучению.

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

$$L(a, X^{\ell}) = \sum_{i=1}^{\ell} \log\left(1 + \exp\left(-y_i a(x_i)\right)\right).$$

Найдем ее антиградиент после $(N-1)$-го шага:

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

Теперь веса ограничены сверху единицей. Если отступ на объекте большой отрицательный (то есть это выброс), то вес при нем будет близок к единице; если же отступ на объекте близок к нулю (то есть это объект, на котором классификация неуверенная, и нужно ее усиливать), то вес при нем будет примерно равен $1/2$. Таким образом, вес при шумовом объекте будет всего в два раза больше, чем вес при нормальных объектах, что не должно сильно повлиять на процесс обучения.

13. Методы оптимизации второго порядка

(дополнительный материал)

Как мы выяснили выше, градиентный бустинг осуществляет градиентный спуск в пространстве прогнозов алгоритма на обучающей выборке. Здесь может возникнуть вполне логичный вопрос: а почему бы не воспользоваться другим, более эффективным методом оптимизации? Наиболее явными кандидатами являются методы оптимизации второго порядка — например, метод Ньютона. При оптимизации числовой функции $Q(w)$ шаг в методе Ньютона осуществляется по формуле

$$w^{(n)} = w^{(n-1)} - H^{-1}(w^{(n-1)})\nabla_w Q(w^{(n-1)}),$$

где $H(w)$ — матрица вторых производных, которая также называется матрицей Гессе.

Этим же подходом можно воспользоваться и в градиентном бустинге. Нам нужно как можно сильнее уменьшить значение следующей функции путем подбора сдвигов s_i:

$$Q(s) = \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i)$$

Мы уже находили вектор градиента:

$$\nabla_s Q(s) = g = \left(\frac{\partial L}{\partial z}\bigg|_{z=a_{N-1}(x_i)} \right)_{i=1}^{\ell}$$

Заметим, что матрица вторых производных тут будет диагональной, поскольку каждая переменная s_i входит лишь в одно отдельное слагаемое:

$$H = \operatorname{diag}\left(\frac{\partial^2 L}{\partial z^2}\bigg|_{z=a_{N-1}(x_1)}, \ldots, \frac{\partial^2 L}{\partial z^2}\bigg|_{z=a_{N-1}(x_\ell)} \right)$$

Мы здесь пользуемся функционалом, который представляет собой сумму ошибок на всех объектах. Такое представление подходит во многих задачах, но не является максимально общим. Дело в том, что в некоторых ситуациях необходимо использовать функционалы, которые измеряют качество сортировки объектов алгоритмом — это, иными словами, функционалы качества ранжирования. Их особенность заключается как раз в том, что матрица вторых производной уже не будет диагональной.

Зная градиент и матрицу Гессе, мы можем выписать формулу для сдвигов s:

$$s = -H^{-1}g$$

Поскольку обращение матрицы является неустойчивой операцией, правильнее будет находить вектор сдвигов через систему линейных уравнений

$$Hs = -g$$

В случае с диагональной матрицей Гессе данный подход сводится к домножению каждой компоненты вектора антиградиента на некоторые коэффициенты. В общем же случае такой подход может оказаться слишком сложным, поскольку при больших размерах выборки матрица Гессе будет получаться слишком большой для эффективной работы. После того, как рассчитан вектор сдвигов, процедура будет такой же, как и раньше — обучаем алгоритм на данные сдвиги, находим коэффициент при нем, добавляем в композицию.

На похожей идее основан метод LogitBoost, который настраивает композицию с использованием логистической функции потерь, исходя из несколько иных предположений. Использование метода Ньютона приводит к тому, что базовый алгоритм настраивается на взвешенный функционал, что может затруднять обучение. Более того, формулы для весов получаются не вполне устойчивыми, и нередко в них происходит деление на очень маленькое число. Чтобы избежать этого, вводится ряд достаточно грубых эвристик.

Обратим внимание, что трюк с переподбором прогнозов в листьях базовых решающих деревьев похож на применение метода Ньютона. Допустим, мы как-то выбрали сдвиги s_i и обучили на них решающее дерево $b_n(x)$. После этого на объекте x_i обучающей выборки будет сделан сдвиг $b_n(x_i)$. Сдвиги будут одинаковыми на тех объектах, которые попали в один и тот же лист дерева. Если сделать переподбор, то сдвиги будут изменены так, чтобы как можно сильнее уменьшать исходный функционал ошибки. По

сути, благодаря этому сдвиги подбираются индивидуально под группы объектов, что близко к использованию методов второго порядка с диагональной матрицей Гессе — там тоже выставляются индивидуальные коэффициенты при компонентах сдвига.

Список литературы

1. *Friedman, Jerome H.* (2001). Greedy Function Approximation: A Gradient Boosting Machine. // Annals of Statistics, 29(5), p. 1189–1232.
2. *Friedman, Jerome H.* (1999). Stochastic Gradient Boosting. // Computational Statistics and Data Analysis, 38, p. 367–378.
3. *Gulin, A., Karpovich, P.* (2009). Greedy function optimization in learning to rank. `http://romip.ru/russir2009/slides/yandex/lecture.pdf`