Содержание

1 Random Forest

Метод случайных лесов основан на бэггинге над решающими деревьями. Алгоритм построения случайного леса выглядит следующим образом:

Для $b = 1, \ldots, B$

1. Построить bootstrap-выбороку X_b^{*n}, размером N,
2. Рекурсивно строим решающее дерево T_b на основе полученных X_b^{*n}, пока не достигнем критерия остановки n_{min} следуя дальнейшим шагам для каждого листа:
   - случайно выбираем m признаков из p признаков,
   - выбираем признак X_j дающий лучшее разбиение из имеющихся m,
   - делим узел на два дочерних узла.

Объединяем построенные деревья $\{T_b\}_{b=1}^{B}$ в композицию.

В задачах регрессии: $\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

В задачах классификации: Пусть $\hat{C}_b(x)$ - предсказание b-того класса дерева случайного леса. Тогда $\hat{C}_{rf}^{B}(x) = majorityvote\{\hat{C}_b(x)\}_1^B$

В случайных лесах корреляция между деревьями понижается путем рандомизации по двум направлениям: по объектам и по признакам. Во-первых, каждое дерево обучается по бутстрапированной подвыборке. Во-вторых, в каждой вершине разбиение ищется по подмножеству признаков. Вспомним, что при построении дерева последовательно происходит разделение вершин до тех пор, пока не будет достигнуто идеальное качество на обучении. Каждая вершина разбивает выборку по одному из признаков относительно некоторого порога. В случайных лесах признак, по которому производится разбиение, выбирается не из всех возможных признаков, а лишь из их случайного подмножества размера m.

Рекомендуется в задачах классификации брать $m = \sqrt{p}$, а в задачах регрессии — $m = p/3$. Также рекомендуется в задачах классификации строить каждое дерево до тех пор, пока в каждом листе не окажется по одному объекту, а в задачах регрессии — пока в каждом листе не окажется по пять объектов.

Случайные леса — один из самых сильных методов построения композиций. На практике он может работать немного хуже градиентного бустинга, но при этом он гораздо более прост в реализации.

2 Boosting

Ранее мы изучили бэггинг и случайные леса — подходы к построению композиций, которые независимо обучают каждый базовый алгоритм по неко-

торому 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. Ниже мы рассмотрим метод, который реализует эту идею — градиентный бустинг. Он работает для любых дифференцируемых функций потерь и является одним из наиболее мощных и универсальных на сегодняшний день.

Пусть X — множество объектов, Y — множество ответов
$y : X \to Y$ — неизвестная зависимость.

Дано: обучающая выборка — $X^n = (x_i, y_i)_{i=1}^n$,
$y_i = y(x_i), i = 1, \ldots, n$ — известные ответы.

Требуется построить алгоритм $a(x) = C(b(x))$, аппроксимирующий целевую зависимость y на всём множестве X на основе алгоритмов $b_1(x), \ldots, b_T(x)$ (в качестве алгоритмов могут выступать например решающие деревья).

2.1 Бустинг в задаче регрессии

Рассмотрим задачу минимизации квадратичного функционала:
$\frac{1}{2}\sum_{i=1}^{l}(a(x_i) - y_i)^2 \to \min\limits_{a}$

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 принадлежат некоторому семейству $\mathbf{A}$.

Первый базовый алгоритм: $b_1(x) := \underset{b\in A}{argmin}\frac{1}{2}\sum_{i=1}^{l}(b(x_i) - y_i)^2$.

Решение такой задачи не представляет трудностей для многих семейств алгоритмов. Теперь мы можем посчитать остатки на каждом обхекте — растояния от ответа нашего алгоритма до истинного ответа: $s_i^{(1)} = y_i - b_i(x)$

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:
$b_2(x) := \underset{b\in A}{argmin}\frac{1}{2}\sum_{i=1}^{l}(b(x_i) - s^{(1)})^2$

Таким образом, каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:
$s_i^{(N)} = y_i - \sum_{n=1} N - 1 b_n(x_i) = y_i - a_{N-1}(x_i), \qquad i = 1, \ldots, l$
$b_N(x) := \underset{b\in A}{argmin}\frac{1}{2}\sum_{i=1}^{l}(b(x_i) - s_i^{(N)})^2$

Также, 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.

2.2 Бустинг в задаче 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:

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

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

$$Q_T = \sum_{i=1}^{n} \left[y_i \sum_{t=1}^{T} \alpha_t b_t(x_i) < 0 \right]. \tag{2}$$

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.

2.3 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.

2.4 Adaboost

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

Оценим функционал Q_T сверху:

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

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

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 $N + P = 1$.

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

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

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

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

2.5 AnyBoost

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

$L(M)$ — 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:

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

Рассмотрим функцию потерь L как функцию параметра α_T,

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

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:

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

где w_i — веса объектов.

2.6 Принцип явной максимизации отступов.

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

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

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 $L(M) = e^{-M}$.

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:
   $\omega_i = -Le'(M_i)$, $i = 1, \ldots, n$;
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: $a_t := \arg\max_{\alpha} \sum_{i=1}^{n} L(M_i + \alpha b_t(x_i) y_i)$;
6. пересчет отступов:
   $M_i := M_i + \alpha b_t(x_i) y_i$; $i = 1, \ldots, n$.

2.7 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.

Примеры выбора алгоритма $b_0(x)$:

1. Нулевой: $b_0(x) = 0$.

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

3. Возвращающий средний ответ (в задачах регрессии): $b_0(x) = \frac{1}{l} \sum_{i=1} l y_i$

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

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 $s_1, \ldots, s_l$ надо выбрать для решения следующей задачи:
$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + s_i\right) \to \underset{s_1, \ldots, s_l}{min}$

Можно, например, выбрать $s_i = y_i - a_{N-1}(x_i)$, но данный подход не учитывает особенностей функции потерть $L(y, z)$.

Другой вариант — 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 таком случае мы сдвинемся в сторону скорейшего убывания функции потерь. Заметим, что вектор сдвигов $s = s_1, \ldots, s_l$ совпадает с антиградиентом:
$$\left(- \left. \frac{\partial L}{\partial z} \right|_{z = a_{N-1}(x_i)} \right)_{i=1}^{l} = - \bigtriangledown_z \sum_{i=1}^{l} L(y_i, z_i)|_{z = a_{N-1}(x_i)}$$

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

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

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

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.

2.8 Регуляризация

2.8.1 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.

Сокращение шага представляет собой следующее: 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. 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 кроссвалидации.

2.8.2 Стохастический градиентный бустинг

Еще одним способом улучшения качества градиентного бустинга является внесение рандомизации в процесс обучения базовых алгоритмов. 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.

2.9 Функции потерь

2.9.1 Регрессия

Одна ищ функций потерь — квадратичная, была рассмотрена в разделе 6.1. Другой вариант — модуль отклонения $L(y, z) = |y - z|$, для которого антиградиент вычисляется по формуле

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

2.9.2 Классификация

В задаче классификации с двумя классами разумным выбором является логистическая функция потерь:

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

Задача поиска базового алгоритма с ней принимает вид

$b_N = \underset{b \in A}{argmin} \sum_{i=1}^{l} (b(x_i) - \frac{y_i}{1+exp(y_i a_{N-1}(x_i))})^2$

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

$Q(a_N) = \sum_{i=1}^{l} log(1+exp(-y_i a_N(x_i))) = \sum_{i=1}^{l} log(1+exp(-y_i a_{N-1}(x_i))exp(-y_i \gamma_n b_N(x_i)))$

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 $w_i^{(N)} = exp(-y_i a_{N-1} x(i))$ можем случить мерой важности объекта x_i на N-й итерации градиентного бустинга.

2.10 Градиентный бустинг над деревьями

Считается, что градиентный бустинг над решающими деревьями — один из самых универсальных и сильных методов машинного обучения, известных на сегодняшний день.

Как известно, 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 в N-й итерации бустинга композиция обновляется как

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

Видно, что добавление в композицию одного дерева с J_N листьями равносильно добавлению J_N базовых алгоритмов, представляющих собой предикаты. Мы можем улучшить качество композиции, подобрав свой коэффициент при каждом из предикатов:

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

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

$\gamma_{Nj} = \arg\min\limits_{\gamma} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma), \qquad j = 1, \dots J_N$

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

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 $\gamma_{Nj} = 0$.

2.10.1 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.

2.11 Взвешивание объектов

Одним из первых широко распространённых методов построения композиций является AdaBoost, в котором оптимизируется экспоненциальная функция потерь $L(y, z) = exp(-yz)$. 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:

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

Компоненты ее антиградиента после $N - 1$ итерации:

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

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 $(y_i)_{i=1}^l$;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.

2.12 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:

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

Ее антиградиент после $N-1$ шага:

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

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.

3 XGBoost

Мы уже разобрались с двумя типами методов построения композиций — бустингом и бэггингом, и познакомились с градиентным бустингом и случайным лесом, которые являются наиболее яркими представителями этих классов. На практике реализация градиентного бустинга оказывается очень непростой задачей, в которой успех зависит от множества тонких моментов. Мы рассмотрим конкретную реализацию градиентного бустинга — пакет XGBoost, который считается одним из лучших на сегодняшний день.

3.1 Градиентный бустинг. Альтернативный подход

Из раздела 2.7 имеем вектор сдвигов:

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

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

$$b_N(x) = \underset{b \in A}{argmin} \sum_{i=1}^{l} (b(x_i) - s_i)^2 \tag{3}$$

Ранее, мы аргументировали использование среднеквадратичной функции потерь тем, что она наиболее проста для оптимизации. Теперь же, найдем более адекватное обоснование этому выбору.

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 $a_{N-1}(x_i)$:

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

,

где h_i — вторые производные по сдвигам: $h_i = \frac{\partial^2}{\partial z^2} L(y_i, z)|_{a_{N-1}(x_i)}$

Первое слагаемое не зависит от нового базового алгоритма, и поэтому его можно выкинуть. Получаем функционал

$$\sum_{i=1}^{l} \left(-s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \right) \tag{4}$$

Покажем, что он очень похож на среднеквадратичный из формулы 3. Преобразуем его:

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

Видно, что последняя формула совпадает с 4 с точностью до константы, если положить $h_i = 1$. Таким образом, в обычном градиентном бустинге мы используем аппроксимацию второго порядка при обучении очередного базового алгоритма, и при этом отбрасываем информацию о вторых производных (то есть считаем, что функция имеет одинаковую кривизну по всем направлениям).

3.2 Регуляризация

Будем далее работать с функционалом 4. Он измеряет лишь ошибку композиции после добавления нового алгоритма, никак при этом не штрафуя за излишнюю сложность этого алгоритма. Ранее мы решали проблему переобучения путем ограничения глубины деревьев, но можно подойти к вопросу и более гибко. Мы выясняли, что 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.

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

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

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

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

Каждое слагаемое здесь можно минимизировать по b_j независимо. Заметим, что отдельное слагаемое представляет собой параболу относительно b_j, благодаря чему 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

$$H(b) = -\frac{1}{2}\sum_{j=1}^{J}\frac{S_j^2}{H_j + \lambda} + \gamma J \tag{5}$$

3.3 Обучение решающего дерева

Мы получили функционал $H(b)$, который для заданной структуры дерева вычисляет минимальное значение ошибки (4), которую можно получить путем подбора коэффициентов в листьях. Заметим, что он прекрасно подходит на роль критерия информативности — с его помощью можно принимать решение, какое разбиение вершины является наилучшим! Значит, с его помощью мы можем строить дерево. 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

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

За счет этого мы будем выбирать структуру дерева так, чтобы оно как можно лучше решало задачу минимизации исходной функции потерь. При этом можно ввести вполне логичный критерий останова: вершину нужно объявить листом, если даже лучшее из разбиений приводит к отрицательному значению функционала Q.

3.4 Заключение

Итак, градиентный бустинг в XGBoost имеет ряд важных особенностей.

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.

4. Критерий останова при обучении дерева также зависит от оптимального сдвига.