Семинары по композиционным методам

Евгений Соколов
sokolov.evg@gmail.com

25 февраля 2015 г.

2 Композиционные методы машинного обучения

§2.1 Градиентный бустинг

Ранее мы изучили алгоритм AdaBoost, строящий композицию алгоритмов путем последовательной минимизации экспоненциальной функции потерь. Из плюсов этого подхода можно отметить аналитическую запись формул для поиска базовых алгоритмов, гарантии сходимости, направленность на максимизацию отступов. Минусы вытекают из вида экспоненциальной функции потерь — алгоритм неустойчив к выбросам, а также крайне тяжело адаптируем для других задач (регрессии, многоклассовой классификации). Ниже мы рассмотрим другой подход к построению композиций — градиентный бустинг, предложенный Фридманом [1]. Данный подход работает для любых функций потерь и является одним из наиболее мощных и универсальных на сегодняшний день.

2.1.1 Бустинг в задаче регрессии

Рассмотрим задачу минимизации квадратичной функции потерь:

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

Будем искать алгоритм в виде суммы базовых:

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

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

Построим первый базовый алгоритм:

$$b_1(x) := \arg\min_{b\in\mathcal{A}} \frac{1}{2}\sum_{i=1}^{\ell}(b(x_i) - y_i)^2.$$

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:

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

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

$$b_2(x) := \arg\min_{b \in \mathcal{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - z_i^{(1)})^2.$$

Каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:

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

$$b_N(x) := \arg\min_{b \in \mathcal{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - z_i^{(N)})^2.$$

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:

$$z_i^{(N-1)} = y_i - a_{N-1}(x_i) = -\frac{\partial}{\partial a} \frac{1}{2} \sum_{i=1}^{\ell} (a - y_i)^2 \Bigg|_{a = a_{N-1}(x_i)}.$$

Это наблюдение наводит нас на мысль, что аналогичным образом можно было бы строить композиции, оптимизирующие и другие функции потерь — достаточно заменить остатки $z_i^{(N)}$ на градиент нужного функционала. Именно так и работает градиентный бустинг, о котором пойдет речь ниже.

2.1.2 Градиентный спуск в функциональном пространстве

Пусть дана некоторая дифференцируемая функция потерь $L(y, z)$. Предположим сначала, что нам известно распределение на объектах и ответах $p(x, y)$. В этом случае мы можем записать функционал среднего риска

$$\Phi(a) = \int_{\mathbb{X}} \int_{\mathbb{Y}} L(y, a(x)) dy dx = \mathbb{E}_{x,y} L(y, a(x)) = \mathbb{E}_x \Big[\mathbb{E}_y L(y, a(x)) \,|\, x \Big].$$

Мы можем воспользоваться непараметрическим подходом и поставить задачу выбора наилучшего алгоритма a среди всех возможных функций от объектов. Минимизация функционала среднего риска эквивалентна минимизации условного матожидания в каждой точке:

$$\varphi(a(x)) = \mathbb{E}_y [L(y, a(x)) \,|\, x] \to \min_{a(x)}$$

Строить алгоритм будем с помощью градиентного спуска. Выберем первый алгоритм b_0 (начальное приближение), после чего мы сможем сделать градиентный шаг:

$$a_1(x) = b_0(x) - \gamma_1 g^{(1)}(x),$$

где

$$g^{(1)}(x) = \left.\frac{\partial \varphi(a)}{\partial a}\right|_{a=b_0(x)}.$$

Аналогично будет делаться и каждый следующий шаг:

$$g^{(N)}(x) = \left.\frac{\partial \varphi(a)}{\partial a}\right|_{a=a_{N-1}(x)};$$
$$a_N(x) = a_{N-1}(x) - \gamma_N g^{(N)}(x).$$

На каждом шаге коэффициент при градиенте находится из задачи одномерной оптимизации

$$\gamma_N = \arg\min_{\gamma} \Phi(a_{N-1}(x) - \gamma g_N(x)),$$

решение которой не должно представлять проблем [1]. Также можно показать, что при выполнении определенных условий регулярности градиент можно вычислять по формуле:

$$g^{(N)}(x) = \mathbb{E}_y \left[\frac{\partial L(y,a)}{\partial a} \mid x\right]_{a=a_{N-1}(x)}.$$

После N шагов градиентного спуска мы получим алгоритм $a_N(x)$, представляющий собой сумму функций:

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

где

$$b_n(x) = -g^{(N)}(x).$$

Таким образом, с помощью градиентного спуска мы построим композицию алгоритмов. Единственная проблема данного подхода состоит в том, что в большинстве случаев нам неизвестно распределение $p(x,y)$, а значит, мы не можем записать функционал $\Phi(a)$ и проводить его оптимизацию. Мы можем работать лишь с эмпирической функцией потерь, вычисляющей ошибку на обучающей выборке:

$$Q(a) = \sum_{i=1}^{\ell} L(y_i, a(x_i)).$$

В этом случае мы можем вычислить значения градиента лишь в точках из обучающей выборки:

$$g_i^{(N)} = \left.\frac{\partial L(y_i,a)}{\partial a}\right|_{a=a_{N-1}(x_i)}.$$

[1] См., например, метод Брента.

Чтобы сделать градиентный шаг и найти следующий базовый алгоритм, необходимо восстановить всю функцию $g^{(N)}(x)$. Будем приближать ее алгоритмами из базового семейства $\mathcal{A}$. Более конкретно, будем настраивать следующий базовый алгоритм b_N, оптимизируя квадратичные отклонения от антиградиентов в точках из обучающей выборки:

$$\sum_{i=1}^{\ell}(b(x) - (-g_i^{(N)}))^2 \to \min_{b\in\mathcal{A}}.$$

Отметим, что здесь мы оптимизируем квадратичную функцию потерь независимо от функционала L исходной задачи — вся информация о функции потерь L находится в градиенте $g_i^{(N)}$, а на данном шаге лишь решается задача аппроксимации функции по ℓ точкам.

После того, как базовый алгоритм b_N найден, остается лишь найти коэффициент при нем:

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

2.1.3 Регуляризация

Сокращение шага. На практике оказывается, что градиентный бустинг очень быстро строит композицию, дающую близкую к нулю ошибку на обучении, после чего начинает настраиваться на шум и переобучаться. Хорошо зарекомендовавшим себя способом решения этой 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.

Также следует обратить внимание на число итераций градиентного бустинга. Хотя ошибка на обучении монотонно стремится к нулю, ошибка на контроле, как правило, начинает увеличивать 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^ℓ, а лишь по ее случайному подмножеству $X^k \subset X^\ell$. В этом случае понижается уровень шума в обучении, а также повышается эффективность вычислений. Существует рекомендация брать подвыборки, размер которых вдвое меньше исходной выборки.

2.1.4 Функции потерь

Регрессия. При вещественном целевом векторе, как правило, используют квадратичную функцию потерь, формулы для которой уже были приведены в разделе 2.1.1.

Другой вариант — модуль отклонения $L(y,z) = |y-z|$, для которого градиент вычисляется по формуле

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

Классификация. В задаче классификации с двумя классами $\mathbb{Y} = \{+1, -1\}$ разумным выбором является настройка функции $p_+(x) \in [0.1]$, возвращающей вероятность класса $+1$. В этом случае мы можем измерить правдоподобие обучающей выборки при условии модели $p_+(x)$:

$$P(p_+) = \prod_{i=1}^{\ell} p_+(x_i)^{[y_i=1]}(1 - p_+(x_i))^{[y_i=-1]}.$$

Данное правдоподобие следует максимизировать. Гораздо удобнее минимизировать отрицательный логарифм правдоподобия:

$$Q(a) = -\sum_{i=1}^{\ell} ([y_i = 1] \log p_+(x_i) + [y_i = -1] \log(1 - p_+(x_i))) \to \min_{p_+(x)}. \tag{2.1}$$

Такая задача крайне неудобна — нам нужно искать алгоритм $p_+(x)$ с ограничением, что его ответ лежит на отрезке $[0,1]$. Будем вместо этого искать алгоритм $a(x) \in \mathbb{R}$, возвращающий любые вещественные числа, который связан с функцией $p_+(x)$ через сигмоидную функцию:

$$p_+(x) = \frac{1}{1 + \exp(-2a(x))}.$$

Подставим это в логарифм правдоподобия (2.1):

$$Q(a) = -\sum_{i=1}^{\ell} (-[y_i = 1] \log(1 + \exp(-2a(x_i))) - [y_i = -1] \log(1 + \exp(2a(x_i)))) =$$
$$= \sum_{i=1}^{\ell} \log(1 + \exp(-2y_i a(x_i))).$$

Полученная нами функция потерь называется *логистической*:

$$L(y, a(x)) = \log(1 + \exp(-2ya(x))).$$

Легко показать, что алгоритм возвращает логарифм отношения оценок вероятностей классов:

$$a(x) = \frac{1}{2} \log \frac{p_+(x)}{1 - p_+(x)}.$$

Оценки вероятностей классов вычисляются по формулам

$$\hat{P}(y = 1 \,|\, x) = \frac{1}{1 + \exp(-2a(x))};$$
$$\hat{P}(y = -1 \,|\, x) = \frac{1}{1 + \exp(2a(x))}.$$

Задача 2.1. *Как будет выглядеть задача поиска базового алгоритма $b_N(x)$ в случае с логистической функцией потерь?*

Решение. Найдем компоненты градиента $g_i^{(N)}$:

$$g_i^{(N)} = \frac{\partial L(y_i, a)}{\partial a}\bigg|_{a=a_{N-1}(x_i)} = -\frac{2y_i}{1+\exp(2y_i a_{N-1}(x_i))}. \tag{2.2}$$

Значит, задача поиска базового алгоритма примет вид

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

■

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

$$\begin{aligned} Q(a_N) &= \sum_{i=1}^{\ell} \log\left(1+\exp(-2y_i a_N(x_i))\right) = \\ &= \sum_{i=1}^{\ell} \log\left(1+\exp(-2y_i a_{N-1}(x_i))\exp(-2y_i\gamma_N b_N(x_i))\right). \end{aligned}$$

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

$$w_i^{(N)} = \exp(-2y_i a_{N-1}(x_i))$$

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

2.1.5 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:

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

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

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

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

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

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

$$\gamma_{Nj} = \arg\min_{\gamma} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma), \qquad j = 1, \ldots, J.$$

В некоторых случаях оптимальные коэффициенты могут быть найдены аналитически.

Задача 2.2. *Найдите оптимальные коэффициенты $\{\gamma_{Nj}\}_{j=1}^{J}$ для функционалов квадратичной и абсолютной ошибки.*

Решение. Требуется решить задачи

$$\sum_{x_i \in R_j} (a_{N-1}(x_i) + \gamma - y_i)^2 \to \min_{\gamma}$$

и

$$\sum_{x_i \in R_j} |a_{N-1}(x_i) + \gamma - y_i| \to \min_{\gamma}.$$

Известно, что решениями данных задач являются среднее и медиана остатков:

$$\gamma_1 = \frac{1}{|R_j|} \sum_{x_i \in R_j} (y_i - a_{N-1}(x_i));$$
$$\gamma_2 = \operatorname*{median}_{x_i \in R_j} \{y_i - a_{N-1}(x_i)\}.$$

Видно, что в случае с квадратичным функционалом дополнительную настройку коэффициентов можно не делать — деревья и так настраиваются на квадратичный функционал, и в листьях будет записано среднее значение ответа по попавшим в них объектам.

В случае с абсолютной функцией потерь настройка коэффициентов уже несет в себе пользу. Сами деревья настраиваются так, чтобы квадратичное отклонение от знака ошибки было минимально (т.е. минимизируется $(b(x) - \operatorname{sign}(y_i - a_{N-1}(x_i)))^2$), но затем мы изменяем значения в листьях так, чтобы они были оптимальны с точки зрения модуля отклонения. Безусловно, дерево можно сразу настраивать на модули отклонений, но такая процедура работает гораздо медленнее, чем настройка на квадратичные потери.

■

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

$$F_j^{(N)}(\gamma) = \sum_{x_i \in R_j} \log\left(1 + \exp\left(-2y_i(a_{N-1}(x_i) + \gamma)\right)\right) \to \min_{\gamma}.$$

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

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

2.1.6 Связь с AdaBoost

Попробуем применить градиентный бустинг к экспоненциальной функции потерь, на оптимизации которой основан AdaBoost:

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

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

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

Заметим, что антиградиент представляет собой ответ на объекте, умноженный на его вес (такой же, как в AdaBoost). 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

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

то есть тоже взвешивает ответы с помощью ошибки на них.

2.1.7 Влияние шума на обучение.

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

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

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:

$$L(a, X^\ell) = \sum_{i=1}^{\ell} \log\left(1 + \exp\left(-2y_i a(x_i)\right)\right).$$

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

$$-g_i^{(N)} = y_i \underbrace{\frac{2}{1 + \exp(2y_i a_{N-1}(x_i))}}_{=w_i^{(N)}}.$$

Теперь веса ограничены сверху двойкой. Если отступ на объекте большой отрицательный (то есть это выброс), то вес при нем будет близок к двойке; 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 единице. 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.

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