Семинары по композиционным методам

Евгений Соколов
sokolov.evg@gmail.com

28 марта 2014 г.

2 Композиционные методы машинного обучения

§2.1 Семейства базовых алгоритмов

В AdaBoost задача построения базового классификатора $b_N(x)$ сводится к поиску классификатора, минимизирующего взвешенную ошибку

$$\sum_{i=1}^{\ell} \tilde{w}_i [b_N(x_i) \neq y_i] \to \min_{b_N \in \mathcal{A}}. \tag{2.1}$$

Рассмотрим некоторые семейства базовых алгоритмов $\mathcal{A}$ и способы решения задачи (2.1).

Решающие пни. Одними из самых популярных базовых классификаторов являются *решающие пни*. Это деревья, состоящие из одной вершины:

$$b(x; j, t) = \begin{cases} +1, & x_j < t; \\ -1, & \text{иначе.} \end{cases}$$

Их параметрами являются номер признака j и порог t. Для такого семейства задача (2.1) решается путем полного перебора по всем параметрам j и t (в качестве вариантов для порога t можно рассматривать точки ровно между соседними значениями признака j, а также по точке слева от минимального и справа от максимального значения этого признака).

Решающие деревья. Многие критерии качества, используемые при построении решающих деревьев, выражаются через распределение объектов по классам. Допустим, мы сейчас рассматриваем вершину R_m, в которую попало N_m объектов. Тогда доля объектов k-го класса обозначается как

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} [y_i = k].$$

Используя данные величины, легко определить, например, энтропийный критерий качества:

$$H(p_m) = -\sum_{k=1}^{K} p_{mk} \log p_{mk}.$$

В случае, когда объекты имеют веса, можно немного модифицировать определение доли объектов k-го класса:

$$\tilde{p}_{mk} = \sum_{x_i \in R_m} \tilde{w}_i [y_i = k].$$

Легко проверить, что $\sum_{k=1}^{K} \tilde{p}_{mk} = 1$. Определения всех критериев не меняются, в них лишь используются модифицированные распределения $\tilde{p}_{mk}$.

Линейные классификаторы. Композиции линейных классификаторов имеют вид

$$a_N(x) = \sum_{n=1}^{N} \gamma_n \operatorname{sign}\langle w_n, x \rangle.$$

Рассмотрим способы настройки функционала (2.1). Допустим, мы выбрали некоторую верхнюю гладкую оценку для него:

$$\tilde{Q}(w) = \sum_{i=1}^{\ell} \tilde{w}_i \mathcal{L}(y_i, \langle w, x_i \rangle).$$

Если мы будем оптимизировать $\tilde{Q}(w)$ с помощью полного градиентного спуска, то никаких специальных трюков не требуется — веса будут уже учтены в градиенте. Если же мы выберем метод стохастического градиента, то можно учесть веса, выбирая на каждой итерации объект x_i с вероятностью $\tilde{w}_i$ вместо $1/\ell$.

Произвольный метод. Учет весов объектов можно встроить в любой метод обучения для любого семейства базовых классификаторов. Для этого достаточно производить обучение базовых алгоритмов по подвыборке, в которую объект x_i выбирается с вероятностью $\tilde{w}_i$.

§2.2 Многоклассовый AdaBoost

Рассмотрим обобщение AdaBoost на многоклассовый случай, предложенное в работе [1].

Пусть $X^\ell = \{x_1, \ldots, x_\ell\} \subset \mathbb{X}$ — выборка с K возможными ответами: $\mathbb{Y} = \{1, \ldots, K\}$. Ответ на i-м объекте обозначим через y_i. Мы воспользуемся K-мерным кодированием ответов: вместо номера класса c возьмем вектор $\vec{y} = (y_1, \ldots, y_K)$, в котором c-я координата равна единице, а все остальные равны $-\frac{1}{K-1}$:

$$y_k = \begin{cases} 1, & k = c, \\ -\frac{1}{K-1}, & k \neq c. \end{cases} \tag{2.2}$$

Каждый алгоритм теперь будет представлять собой K-мерный вектор $\vec{a}(x) = (a_1(x), \ldots, a_K(x))$, удовлетворяющий требованию $a_1(x) + \cdots + a_K(x) = 0$ [1]. В качестве ответа выдается тот класс, чья компонента максимальна:

$$a(x) = \arg\max_{k=1,\ldots,K} a_k(x).$$

[1] Если не ввести это требование, то классификаторы будут определены неоднозначно: прибавление константы ко всем компонентам $a_1(x), \ldots, a_K(x)$ никак не изменит сам алгоритм.

Рассмотрим следующее обобщение экспоненциальной функции потерь на наш случай:

$$q(\vec{a}, x, \vec{y}) = \exp\left(-\frac{1}{K}(y_1 a_1(x) + \cdots + y_K a_K(x))\right) = \exp\left(-\frac{1}{K}\langle \vec{y}, \vec{a}(x)\rangle\right).$$

Данный функционал штрафует за отрицательную корреляцию вектора ответов классификатора с истинным вектором ответов. Выбор именно такого функционала обусловлен тем, что минимум его матожидания достигается на алгоритмах $a_i(x)$, выдающих апостериорные вероятности классов (аналогичным свойством обладает классический AdaBoost).

Заметим, что в случае двух классов мы получим привычную нам постановку задачи AdaBoost. Действительно, если $K = 2$, то в любом векторе ответов $\vec{y}$ одна из компонент будет равна -1, а другая $+1$; для любого классификатора $\vec{a}(x) = (a_1(x), a_2(x))$ будет выполнено $a_1(x) = -a_2(x)$. Функция потерь примет вид

$$\begin{aligned} q(\vec{a}, x, \vec{y}) &= \exp\left(-\frac{1}{2}(y_1 a_1(x) + y_2 a_2(x))\right) = \\ &= \exp\left(-\frac{1}{2}(y_1 a_1(x) + (-y_1)(-a_1(x)))\right) = \\ &= \exp\left(-y_1 a_1(x)\right), \end{aligned}$$

то есть перейдет в стандартную экспоненциальную функцию потерь.

Пусть $\mathcal{A}$ — семейство базовых классификаторов. Потребуем, чтобы каждый из них возвращал вектор вида (2.2), то есть чтобы одна из компонент ответа была равна единице, а все остальные $-\frac{1}{K-1}$. Каждый базовый классификатор $\vec{b}(x)$ возвращает вектор, но ему можно однозначно поставить в соответствие многоклассовый классификатор $B : \mathbb{X} \to \{1, \ldots, K\}$ по правилу

$$B(x) = k \iff b_k(x) = 1,$$

и наоборот,

$$b_k(x) = \begin{cases} 1, & \text{если } B(x) = k, \\ -\frac{1}{K-1}, & \text{если } B(x) \neq k. \end{cases}$$

Будем строить композицию вида

$$\vec{a}_N(x) = \sum_{n=1}^{N} \gamma_n \vec{b}_n(x), \quad b_n \in \mathcal{A}.$$

Композицию будем наращивать последовательно, оптимизируя экспоненциальную функцию потерь:

$$\begin{aligned} Q(\vec{a}_N) &= \sum_{i=1}^{\ell} \exp\left(-\frac{1}{K}\langle \vec{y}_i, \vec{a}_{N-1}(x_i) + \gamma_N \vec{b}_N(x_i)\rangle\right) = \\ &= \sum_{i=1}^{\ell} \underbrace{\exp\left(-\frac{1}{K}\langle \vec{y}_i, \vec{a}_{N-1}(x_i)\rangle\right)}_{= w_i^{(N)}} \exp\left(-\frac{1}{K}\gamma_N \langle \vec{y}_i, \vec{b}_N(x_i)\rangle\right) \end{aligned}$$

Нормируя веса, получаем задачу

$$\sum_{i=1}^{\ell} \tilde{w}_i^{(N)} \exp\left(-\frac{1}{K}\gamma_N \langle \vec{y}_i, \vec{b}_N(x_i)\rangle\right) \to \min_{\gamma_N, \vec{b}_N}$$

Перейдем к многоклассовому классификатору $B_N(x)$, соответствующему $\vec{b}_N(x)$, и распишем в функционале скалярное произведение $\langle \vec{y}_i, \vec{b}_N(x_i)\rangle$:

$$\begin{aligned}
&\sum_{i=1}^{\ell} \tilde{w}_i^{(N)} \exp\left(-\frac{1}{K}\gamma_N \langle \vec{y}_i, \vec{b}_N(x_i)\rangle\right) = \\
&\quad= \sum_{i:\ y_i = B_N(x_i)} \tilde{w}_i^{(N)} \exp\left(-\frac{1}{K}\gamma_N \left((K-1)\frac{1}{(K-1)^2} + 1\right)\right) + \\
&\quad+ \sum_{i:\ y_i \neq B_N(x_i)} \tilde{w}_i^{(N)} \exp\left(-\frac{1}{K}\gamma_N \left((K-2)\frac{1}{(K-1)^2} - \frac{2}{K-1}\right)\right) = \\
&\quad= \sum_{i:\ y_i = B_N(x_i)} \tilde{w}_i^{(N)} e^{-\frac{\gamma_N}{K-1}} + \sum_{i:\ y_i \neq B_N(x_i)} \tilde{w}_i^{(N)} e^{\frac{\gamma_N}{(K-1)^2}} = \\
&\quad= e^{-\frac{\gamma_N}{K-1}} \underbrace{\sum_{i=1}^{\ell} \tilde{w}_i^{(N)}}_{=1} + \left(e^{\frac{\gamma_N}{(K-1)^2}} - e^{-\frac{\gamma_N}{K-1}}\right) \sum_{i=1}^{\ell} \tilde{w}_i^{(N)} [y_i \neq B_N(x_i)] = \\
&\quad= e^{-\frac{\gamma_N}{K-1}} + \left(e^{\frac{\gamma_N}{(K-1)^2}} - e^{-\frac{\gamma_N}{K-1}}\right) \varepsilon_N.
\end{aligned}$$

Если $\gamma_N > 0$, то получаем следующую задачу:

$$B_N^* = \arg\min \sum_{i=1}^{\ell} \tilde{w}_i^{(N)} [y_i \neq B_N(x_i)].$$

Таким образом, мы свели построение базового классификатора к минимизации взвешенного числа ошибок на обучающей выборке.

Найдем теперь γ_N. Продифференцируем функционал и приравняем к нулю:

$$\frac{\partial Q}{\partial \gamma_N} = -\frac{1}{K-1} e^{-\frac{\gamma_N}{K-1}} + \left(\frac{1}{(K-1)^2} e^{\frac{\gamma_N}{(K-1)^2}} + \frac{1}{K-1} e^{-\frac{\gamma_N}{K-1}}\right) \varepsilon_N = 0.$$

Решая уравнение, получаем

$$\gamma_N^* = \frac{(K-1)^2}{K}\left(\log\left(\frac{1-\varepsilon_N}{\varepsilon_N}\right) + \log(K-1)\right).$$

§2.3 AnyBoost

Ранее мы изучили алгоритм AdaBoost, который строит композицию путем минимизации экспоненциальной функции потерь. Известно, что он плохо работает на выборках с выбросами или с шумом: как правило, шумовые объекты имеют большой отрицательный отступ, экспоненциальная функция потерь сильно штрафует его, из-за чего базовые алгоритмы настраиваются именно на эти шумовые объекты. Одним

из возможных вариантов решения является использование других функций потерь, не так сильно штрафующих объекты-выбросы. Мы рассмотрим логистическую функцию потерь и ее оптимизацию с помощью алгоритма AnyBoost [2].

Пусть $X^\ell = \{x_1, \ldots, x_\ell\} \subset \mathbb{X}$ — выборка с бинарными ответами ($\mathbb{Y} = \{-1, +1\}$). Пусть дано семейство базовых классификаторов $\mathcal{A}$, каждый из которых является отображением из $\mathbb{X}$ в $\mathbb{Y}$. Будем строить композицию вида

$$a_N(x) = \operatorname{sign} \sum_{n=1}^{N} \gamma_n b_n(x), \quad b_n \in \mathcal{A}.$$

Будем наращивать композицию последовательно, оптимизируя на каждом шаге логистическую функцию потерь

$$\begin{aligned} Q(a_N) &= \sum_{i=1}^{\ell} \log \left(1 + \exp\left(-y_i \sum_{n=1}^{N} \gamma_n b_n(x_i)\right)\right) = \\ &= \sum_{i=1}^{\ell} \log\Big(1 + \exp\big(-y_i(a_{N-1}(x_i) + \gamma_n b_n(x_i))\big)\Big) \to \min_{b_N, \gamma_N}. \end{aligned}$$

В отличие от экспоненциальной функции потерь, слагаемые в данном функционале не получится разбить на отдельные множители, соответствующие базовым алгоритмам, получив в итоге взвешенную ошибку для b_N.

Попробуем выбрать новый базовый классификатор $b_N(x)$ так, чтобы функция потерь максимально быстро убывала в соответствующем ему направлении. Характеристикой скорости убывания функции в направлении $b_N(x)$ выберем производную по коэффициенту γ_N в точке $\gamma_N = 0$. Получаем:

$$\left.\frac{\partial Q}{\partial \gamma_N}\right|_{\gamma_N = 0} = \sum_{i=1}^{\ell} (-y_i b_N(x_i)) \frac{\exp\left(-y_i a_{N-1}(x_i)\right)}{1 + \exp\left(-y_i a_{N-1}(x_i)\right)} \to \min_{b_N}.$$

Заметим, что дробь в последнем выражении не зависит от b_N и γ_N, поэтому она по сути является весом для соответствующего объекта:

$$w_i^{(N)} = \frac{\exp\left(-y_i a_{N-1}(x_i)\right)}{1 + \exp\left(-y_i a_{N-1}(x_i)\right)}.$$

Как и в AdaBoost, отнормируем данные веса:

$$\tilde{w}_i^{(N)} = \frac{w_i^{(N)}}{\sum_{i=1}^{\ell} w_i^{(N)}}.$$

Мы получили, что поиск нового базового классификатора сводится к взвешенной максимизации отступов обучающих объектов:

$$b_N^*(x) = \arg\max_{b_N} \sum_{i=1}^{\ell} \tilde{w}_i^{(N)} y_i b_N(x_i).$$

Задачу можно представить и немного в другом виде. Пользуясь тождеством

$$b_N(x_i) y_i = 1 - 2[y_i \neq b_N(x_i)], \tag{2.3}$$

получаем задачу минимизации взвешенной ошибки:

$$b_N^*(x) = \arg\min_{b_N} \sum_{i=1}^{\ell} \tilde{w}_i^{(N)} [y_i \neq b_N(x_i)].$$

После того, как мы нашли базовый классификатор $b_N(x)$, остается найти коэффициент γ_N. Это делается путем решения одномерной задачи оптимизации

$$\gamma_N^* = \arg\min_{\gamma_N} \sum_{i=1}^{\ell} \log\Big(1 + \exp\big(-y_i(a_{N-1}(x_i) + \gamma_n b_n(x_i))\big)\Big).$$

Ранее мы вывели интересное свойство AdaBoost: n-й алгоритм имеет ошибку $1/2$ относительно распределения с $(n+1)$-го шага. Покажем, что оно выполняется и в AnyBoost.

Задача 2.1. *Покажите, что в AnyBoost взвешенная ошибка базового классификатора b_N относительно весов со следующего шага $\tilde{w}^{(N+1)}$ равна $1/2$.*

Решение.

$$\begin{aligned}
&\sum_{i=1}^{\ell} \tilde{w}_i^{(N+1)} [b_N(x_i) \neq y_i] = \{(2.3)\} = \frac{1}{2} - \frac{1}{2} \sum_{i=1}^{\ell} \tilde{w}_i^{(N+1)} y_i b_N(x_i) = \\
&\quad = \frac{1}{2} - \frac{1}{2\sum_{i=1}^{\ell} w_i^{(N+1)}} \sum_{i=1}^{\ell} w_i^{(N+1)} y_i b_N(x_i) = \\
&\quad = \frac{1}{2} - \frac{1}{2\sum_{i=1}^{\ell} w_i^{(N+1)}} \sum_{i=1}^{\ell} \frac{\exp\left(-y_i a_N(x_i)\right)}{1 + \exp\left(-y_i a_N(x_i)\right)} y_i b_N(x_i) = \\
&\quad = \frac{1}{2} - \frac{1}{2\sum_{i=1}^{\ell} w_i^{(N+1)}} \frac{\partial Q(a_N)}{\partial \gamma_N} = \\
&\quad = \{\gamma_N \text{ выбран как оптимум } Q, \text{ значит производная равна нулю}\} = \\
&\quad = \frac{1}{2}.
\end{aligned}$$

■

§2.4 Градиентный бустинг

Рассмотрим метод градиентного бустинга, который предлагает еще один способ построения композиции вида

$$a_N(x) = b_0 + \sum_{n=1}^{N} \gamma_n b_n(x), \quad b_0 \in \mathbb{R}, b_n \in \mathcal{A},$$

минимизирующей произвольный дифференцируемый функционал

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

Будем строить композицию по шагам, добавляя по одному базовому алгоритму за итерацию. Начнем с константного слагаемого b_0:

$$b_0 = \arg\min_{c \in \mathbb{R}} \sum_{i=1}^{\ell} L(y_i, c).$$

Это задача одномерной оптимизации, решение которой не должно представлять проблем [2].

Рассмотрим теперь задачу поиска базового алгоритма и коэффициента при нем:

$$(b_n, \gamma_n) = \arg\min_{b_n, \gamma_n} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)). \tag{2.4}$$

Здесь $b_N(x)$ — это функция из некоторого фиксированного семейства, для оптимизации по которой, вообще говоря, не существует эффективных методов. В AnyBoost предлагалось линеаризовать функцию потерь, получив тем самым задачу минимизации взвешенного числа ошибок на обучении.

В градиентном бустинге предлагается рассмотреть все алгоритмы не как функции, а как векторы их ответов на обучающей выборке. Обозначим через $z_i^{(N-1)}$ ответ композиции с $(N-1)$-го шага на объекте x_i:

$$z_i^{(N-1)} = a_{N-1}(x_i).$$

Критерий качества $Q(a)$ будем рассматривать не как функционал, определенный на множестве функций, а как функцию от ℓ аргументов — ответов композиции на объектах обучающей выборки:

$$\tilde{Q}(z_1, \ldots, z_\ell) = \sum_{i=1}^{\ell} L(y_i, z_i).$$

В такой постановке задача поиска следующего базового алгоритма и коэффициента при нем сводится к поиску вектора $\vec{\delta} = (\delta_i)_{i=1}^{\ell}$ и числа α, минимизирующих значения нашего нового функционала:

$$\sum_{i=1}^{\ell} L(y_i, z_i^{(N-1)} + \alpha\delta_i) \to \min_{\alpha, \vec{\delta}}.$$

Выберем α и $\vec{\delta}$, сделав один шаг скорейшего градиентного спуска в сторону минимума функционала:

$$\delta_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z = z_i^{(N-1)}} \equiv g_i^N, \quad i = 1, \ldots, \ell;$$

$$\alpha = \arg\min_{\alpha} \sum_{i=1}^{\ell} L(y_i, z_i^{(N-1)} + \alpha\delta_i).$$

[2] См., например, метод Брента.

Новые векторы ответов композиции записываются как

$$z_i^{(N)} = z_i^{(N-1)} + \alpha\delta_i.$$

Описанная процедура очень проста в реализации, однако на выходе мы получим лишь ответы композиции на объектах обучающей выборки. Чтобы получить на выходе полноценную композицию, будем на каждой итерации искать базовый алгоритм $b_N \in \mathcal{A}$, ответы которого на обучении максимально точно приближают антиградиент функции потерь:

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

Мы получили задачу наименьших квадратов, решать которую гораздо проще, чем исходную задачу (2.4). Отметим, что вместо квадратичной функции потерь можно выбирать и другую — главное, чтобы мы получили простую в решении задачу.

После того, как найден базовый алгоритм b_N, коэффициент при нем ищется с помощью одномерной оптимизации:

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

Задача 2.2. *Как будет выглядеть задача поиска базового алгоритма $b_N(x)$ в случае с квадратичной функцией потерь $L(y,z) = \frac{1}{2}(y-z)^2$?*

Решение. Найдем компоненты антиградиента g_i^N:

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

Значит, задача поиска базового алгоритма примет вид

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

Таким образом, в случае с квадратичной функцией потерь градиентный бустинг настраивает каждый следующий базовый алгоритм на остатки уже построенной композиции.

■

Регуляризация. 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 [3]. Как правило, чем меньше темп обучения, тем лучше качество итоговой композиции.

Стохастический градиентный бустинг. Еще одним способом улучшения качества градиентного бустинга является внесение рандомизации в процесс обучения базовых алгоритмов [4]. А именно, алгоритм b_N обучается не по всей выборке X^ℓ, а лишь по ее случайному подмножеству $X^k \subset X^\ell$. Существует рекомендация брать подвыборки, размер которых вдвое меньше исходной выборки.

Градиентный бустинг над решающими 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 [5].

Список литературы

[1] *Zhu, Ji et al.* (2009). Multi-class AdaBoost. // Statistics and Its Interface, 2, p. 177-186.

[2] *Mason, Llew et al.* (1999). Boosting Algorithms as Gradient Descent. // In proceeding of Advances in Neural Information Processing Systems 12.

[3] *Friedman, Jerome H.* (2001). Greed Function Approximation: A Gradient Boosting Machine. // Annals of Statistics, 29(5), p. 1189–1232.

[4] *Friedman, Jerome H.* (1999). Stochastic Gradient Boosting. // Computational Statistics and Data Analysis, 38, p. 367–378.

[5] *Gulin, A., Karpovich, P.* (2009). Greedy function optimization in learning to rank. http://romip.ru/russir2009/slides/yandex/lecture.pdf