

Лекции по алгоритмическим композициям

К. В. Воронцов

24 июня 2010 г.

Материал находится в стадии разработки, может содержать ошибки и неточности. Автор будет благодарен за любые замечания и предложения, направленные по адресу vokov@forecsys.ru, либо высказанные в обсуждении страницы «Машинное обучение (курс лекций, К.В.Воронцов)» вики-ресурса www.MachineLearning.ru.

Перепечатка фрагментов данного материала без согласия автора является плагиатом.

Содержание

1	Композиции алгоритмов	2
1.1	Понятие композиции алгоритмов	2
1.2	Бустинг в задачах классификации	4
1.2.1	Алгоритм AdaBoost	4
1.2.2	Алгоритм AnyBoost	9
1.2.3	Голосование по большинству	9
1.2.4	Голосование по старшинству	13
1.3	Стохастические методы построения композиций	15
1.3.1	Бэггинг и метод случайных подпространств	15
1.3.2	Кооперативная коэволюция	17
1.4	Квазилинейные композиции (смеси алгоритмов)	20
1.4.1	Выпуклые функции потерь	22
1.4.2	Последовательный метод построения смеси	23
1.4.3	Иерархический метод построения смеси	26
1.5	Нелинейные монотонные композиции	28
1.5.1	Оптимизация базовых алгоритмов	30
1.5.2	Монотонная интерполяция и аппроксимация	33
1.6	Краткий обзор литературы	36

1 Композиции алгоритмов

При решении сложных задач классификации, регрессии, прогнозирования часто оказывается, что ни один из алгоритмов не обеспечивает желаемого качества восстановления зависимости. В таких случаях имеет смысл строить композиции алгоритмов, в которых ошибки отдельных алгоритмов взаимно компенсируются. Наиболее известные примеры композиций — простое и взвешенное голосование. В данной главе рассматриваются и другие типы композиций — голосование по старшинству, смеси и монотонные корректоры.

Напомним основные обозначения.

Рассматривается задача обучения по прецедентам $\langle X, Y, y^*, X^\ell \rangle$, где X — пространство объектов; Y — множество ответов; $y^*: X \rightarrow Y$ — неизвестная целевая зависимость; $X^\ell = (x_1, \dots, x_\ell)$ — обучающая выборка; $Y^\ell = (y_1, \dots, y_\ell)$ — вектор ответов на обучающих объектах, $y_i = y^*(x_i)$. Требуется построить алгоритм $a: X \rightarrow Y$, аппроксимирующий целевую зависимость y^* на всём множестве X .

§1.1 Понятие композиции алгоритмов

Наиболее общее определение алгоритмической композиции даётся в алгебраическом подходе Ю. И. Журавлёва [6]. Наряду с множествами X и Y вводится вспомогательное множество R , называемое *пространством оценок*. Рассматриваются алгоритмы, имеющие вид суперпозиции $a(x) = C(b(x))$, где функция $b: X \rightarrow R$ называется *алгоритмическим оператором*, функция $C: R \rightarrow Y$ — *решающим правилом*.

Многие алгоритмы классификации имеют именно такую структуру: сначала вычисляются оценки принадлежности объекта классам, затем решающее правило переводит эти оценки в номер класса. Значением оценки $b(x)$ может быть вероятность принадлежности объекта x классу, расстояние от объекта до разделяющей поверхности, степень уверенности классификации и т. п.

Опр. 1.1. *Композицией алгоритмов $a_t(x) = C(b_t(x))$, $t = 1, \dots, T$ называется суперпозиция алгоритмических операторов $b_t: X \rightarrow R$, корректирующей операции $F: R^T \rightarrow R$ и решающего правила $C: R \rightarrow Y$:*

$$a(x) = C(F(b_1(x), \dots, b_T(x))), \quad x \in X. \quad (1.1)$$

Алгоритмы a_t , а иногда и операторы b_t , называют *базовыми алгоритмами*.

Суперпозиции вида $F(b_1, \dots, b_T)$ являются отображениями из X в R , то есть, опять-таки, алгоритмическими операторами. Это позволяет строить иерархические композиции, применяя определение 1.1 рекурсивно.

Пространство оценок R вводится для того, чтобы расширить множество допустимых корректирующих операций. Можно было бы определить корректирующую операцию и как отображение $F: Y^T \rightarrow Y$, то есть комбинировать непосредственно ответы базовых алгоритмов. Однако в задачах классификации, когда множество Y конечно, число «разумных» корректирующих операций такого вида невелико, что ограничивает возможность оптимизации F под конкретную задачу [6]. Если же комбинировать ответы алгоритмических операторов, то операция F получает на входе более детальную информацию, не огрублённую решающим правилом.

Пример 1.1. В задачах классификации на два класса, $Y = \{-1, +1\}$, в качестве пространства оценок обычно используется множество действительных чисел $R = \mathbb{R}$. В этом случае алгоритмические операторы называют также *вещественнозначными классификаторами* (real-valued classifiers):

$$C(b(x)) = \text{sign } b(x).$$

Пример 1.2. В задачах классификации на M классов, $Y = \{1, \dots, M\}$, в качестве пространства оценок обычно используется $R = \mathbb{R}^M$. Алгоритмический оператор $b(x)$ выдаёт вектор оценок принадлежности объекта x каждому из классов, $b(x) = (b^1(x), \dots, b^M(x))$. Решающее правило C относит объект к тому классу, для которого оценка максимальна:

$$C(b(x)) \equiv C(b^1(x), \dots, b^M(x)) = \arg \max_{y \in Y} b^y(x).$$

В частности, именно такой вид имеют байесовские алгоритмы классификации из главы ?? и метрические алгоритмы из ??.

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:

$$b(x) = F(b_1(x), \dots, b_T(x)) = \sum_{t=1}^T \alpha_t b_t(x), \quad x \in X, \quad \alpha_t \in \mathbb{R}. \quad (1.2)$$

Если веса α_t неотрицательны и нормированы, $\sum_{t=1}^T \alpha_t = 1$, то композицию (1.2) называют *выпуклой комбинацией* базовых алгоритмов. Естественно предполагать, что вес α_t тем больше, чем выше качество базового алгоритма b_t .

В задачах классификации корректирующая операция (1.2) называется *взвешенным голосованием* (weighted voting).

Пример 1.3. Рассмотрим задачу классификации с двумя классами, $Y = \{-1, +1\}$, и двухслойную нейронную сеть, принимающую на входе n -мерный вектор признаков, $x = (x^1, \dots, x^n)$, $x \in X = \mathbb{R}^n$:

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

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 $F(b_1, \dots, b_T) = \sum_{t=1}^T w_t b_t$ и решающим правилом $C(b) = \text{sign } b$.

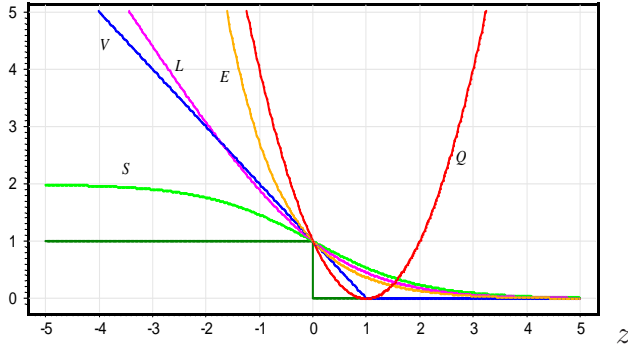


Рис. 1. Гладкие верхние аппроксимации пороговой функции потерь $[z < 0]$:

$S(z) = 2(1 + e^z)^{-1}$ — сигмоидная;

$L(z) = \log_2(1 + e^{-z})$ — логарифмическая;

$V(z) = (1 - z)_+$ — кусочно-линейная;

$E(z) = e^{-z}$ — экспоненциальная;

$Q(z) = (1 - z)^2$ — квадратичная.

§1.2 Бустинг в задачах классификации

Рассмотрим задачу классификации на два класса, $Y = \{-1, +1\}$. Допустим, что решающее правило фиксировано, $C(b) = \text{sign}(b)$, 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:

$$a(x) = C(F(b_1(x), \dots, b_T(x))) = \text{sign}\left(\sum_{t=1}^T \alpha_t b_t(x)\right), \quad x \in X. \quad (1.3)$$

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

$$Q_T = \sum_{i=1}^{\ell} \left[y_i \sum_{t=1}^T \alpha_t b_t(x_i) < 0 \right]. \quad (1.4)$$

Для упрощения задачи минимизации функционала Q_T введём две эвристики.

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 $\alpha_1 b_1(x), \dots, \alpha_{t-1} b_{t-1}(x)$ полагаются фиксированными.

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

Вторая эвристика широко используется в теории классификации. В частности, логарифмическая функция связана с принципом максимума правдоподобия и применяется в нейронных сетях (см. ??) и логистической регрессии (см. ??). Кусочно-линейная аппроксимация связана с принципом максимизации зазора между классами и применяется в методе опорных векторов (см. ??). Примеры других аппроксимаций показаны на Рис. 1. Все эти функции можно использовать, получая различные варианты бустинга. Начнём с экспоненциальной аппроксимации, которая исторически была самой первой.

1.2.1 Алгоритм AdaBoost

При использовании экспоненциальной аппроксимации $[y_i b(x_i) < 0] \leq e^{-y_i b(x_i)}$ эти две эвристики приводят к широко известному алгоритму AdaBoost [37].

Оценим функционал Q_T сверху:

$$\begin{aligned} Q_T &\leq \tilde{Q}_T = \sum_{i=1}^{\ell} \exp\left(-y_i \sum_{t=1}^T \alpha_t b_t(x_i)\right) = \\ &= \sum_{i=1}^{\ell} \underbrace{\exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t b_t(x_i)\right)}_{w_i} e^{-y_i \alpha_T b_T(x_i)}. \end{aligned} \quad (1.5)$$

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

Введём вектор нормированных весов $\tilde{W}^\ell = (\tilde{w}_1, \dots, \tilde{w}_\ell)$, где $\tilde{w}_i = w_i / \sum_{j=1}^{\ell} w_j$.

Определим два функционала качества алгоритма классификации b на обучающей выборке X^ℓ, Y^ℓ с нормированным вектором весов объектов $U^\ell = (u_1, \dots, u_\ell)$: суммарный вес ошибочных (negative) классификаций $N(b; U^\ell)$ и суммарный вес правильных (positive) классификаций $P(b; U^\ell)$:

$$\begin{aligned} N(b; U^\ell) &= \sum_{i=1}^{\ell} u_i [b(x_i) = -y_i]; \\ P(b; U^\ell) &= \sum_{i=1}^{\ell} u_i [b(x_i) = y_i]. \end{aligned}$$

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

Теорема 1.1. Пусть для любого нормированного вектора весов U^ℓ существует базовый алгоритм b , классифицирующий выборку хотя бы немного лучше, чем наугад: $P(b; U^\ell) > N(b; U^\ell)$. Тогда минимум функционала \tilde{Q}_T достигается при

$$\begin{aligned} b_T &= \arg \max_b \sqrt{P(b; \tilde{W}^\ell)} - \sqrt{N(b; \tilde{W}^\ell)}. \\ \alpha_T &= \frac{1}{2} \ln \frac{P(b_T; \tilde{W}^\ell)}{N(b_T; \tilde{W}^\ell)}. \end{aligned}$$

Доказательство.

Воспользуемся тождеством $e^{-\alpha b} = e^{-\alpha} [b=1] + e^{\alpha} [b=-1] + [b=0]$, справедливым для любых $\alpha \in \mathbb{R}$, $b \in \{-1, 0, +1\}$. Положим для краткости $\alpha = \alpha_T$ и $b_i = b_T(x_i)$. Тогда (1.5) примет вид

$$\begin{aligned} \tilde{Q}_T &= \left(e^{-\alpha} \underbrace{\sum_{i=1}^{\ell} \tilde{w}_i [b_i = y_i]}_P + e^{\alpha} \underbrace{\sum_{i=1}^{\ell} \tilde{w}_i [b_i = -y_i]}_N + \underbrace{\sum_{i=1}^{\ell} \tilde{w}_i [b_i = 0]}_{1-P-N} \right) \underbrace{\sum_{i=1}^{\ell} w_i}_{\tilde{Q}_{T-1}} = \\ &= (e^{-\alpha} P + e^{\alpha} N + (1 - P - N)) \tilde{Q}_{T-1}, \end{aligned}$$

Дифференцируя \tilde{Q}_T по параметру α и приравнявая нулю производную, получим оптимальное значение

$$\alpha_T = \frac{1}{2} \ln \frac{P}{N}.$$

Подставляя его обратно в \tilde{Q}_T , получим

$$\tilde{Q}_T = \tilde{Q}_{T-1} \left(1 - (\sqrt{P} - \sqrt{N})^2 \right). \quad (1.6)$$

Поскольку \tilde{Q}_{T-1} не зависит от α_T и b_T , минимизация \tilde{Q}_T эквивалентна либо максимизации $\sqrt{P} - \sqrt{N}$ при $P > N$, либо максимизации $\sqrt{N} - \sqrt{P}$ при $P < N$, однако второй случай исключён условием теоремы. ■

Требование, чтобы каждый базовый алгоритм классифицировал объекты хотя бы немного лучше, чем наугад, является достаточно слабым, и на практике, как правило, выполняется. Более того, этого оказывается достаточно, чтобы гарантировать сходимость алгоритма AdaBoost за конечное число шагов.

Теорема 1.2. *Если на каждом шаге метод обучения обеспечивает построение базового алгоритма b_t такого, что $\sqrt{P(b_t; \tilde{W}^\ell)} - \sqrt{N(b_t; \tilde{W}^\ell)} = \gamma_t > \gamma$ при некотором $\gamma > 0$, то за конечное число шагов будет построен корректный алгоритм $a(x)$.*

Доказательство.

Непосредственно из (1.6) вытекает сходимость функционала Q_T к нулю со скоростью геометрической прогрессии: $Q_{T+1} \leq \tilde{Q}_{T+1} \leq \tilde{Q}_1 (1 - \gamma^2)^T$. Наступит момент, когда значение \tilde{Q}_T окажется меньше 1. Но тогда функционал Q_T обратится в нуль, поскольку он может принимать только целые неотрицательные значения. ■

На практике сходимость может и не достигаться, если модель базовых алгоритмов не достаточно «богата». В таких случаях значения γ_t сходятся к нулю настолько быстро, что Q_T , наоборот, не успевают сойтись к нулю.

Реализация AdaBoost показана в Алгоритме 1.1 для важного частного случая, когда базовые алгоритмы не отказываются от классификации ($N + P = 1$).

Теорема 1.3. *Пусть базовые алгоритмы не отказываются от классификации, и для любого нормированного вектора весов W^ℓ существует базовый алгоритм b , классифицирующий выборку хотя бы немного лучше, чем наугад: $N(b; U^\ell) < \frac{1}{2}$. Тогда минимум функционала \tilde{Q}_T достигается при*

$$b_T = \arg \min_b N(b; \tilde{W}^\ell);$$

$$\alpha_T = \frac{1}{2} \ln \frac{1 - N(b_T; \tilde{W}^\ell)}{N(b_T; \tilde{W}^\ell)}.$$

В данном случае задача построения базового алгоритма b_T по обучающей выборке X^ℓ решается путём минимизации функционала взвешенной частоты ошибок, который отличается от стандартного эмпирического риска (частоты ошибок) только тем, что объекты учитываются с весами w_i . Стандартные методы минимизации эмпирического риска, как правило, легко обобщаются на этот случай.

Некоторых комментариев заслуживает правило мультипликативного пересчёта весов на шаге 5. Оно вытекает непосредственно из определения w_i в (1.5). Вес объекта увеличивается в e^{α_t} раз, когда b_t допускает на нём ошибку, и во столько же раз уменьшается, когда b_t правильно классифицирует x_i . Таким образом, непосредственно перед настройкой базового алгоритма наибольший вес накапливается у тех объектов, которые чаще оказывались трудными для предыдущих алгоритмов.

Алгоритм 1.1. AdaBoost — построение линейной комбинации классификаторов

Вход:

X^ℓ, Y^ℓ — обучающая выборка; T — максимальное число базовых алгоритмов;

Выход:

базовые алгоритмы и их веса $\alpha_t b_t$, $t = 1, \dots, T$;

- 1: инициализация весов объектов: $w_i := 1/\ell$, $i = 1, \dots, \ell$;
 - 2: **для всех** $t = 1, \dots, T$, пока не выполнен критерий останова
 - 3: $b_t := \arg \min_b N(b; W^\ell)$;
 - 4: $\alpha_t := \frac{1}{2} \ln \frac{1 - N(b_t; W^\ell)}{N(b_t; W^\ell)}$;
 - 5: пересчёт весов объектов: $w_i := w_i \exp(-\alpha_t y_i b_t(x_i))$, $i = 1, \dots, \ell$;
 - 6: нормировка весов объектов: $w_0 := \sum_{j=1}^{\ell} w_j$; $w_i := w_i/w_0$, $i = 1, \dots, \ell$;
-

Фильтрация выбросов. После построения некоторого количества базовых алгоритмов (скажем, пары десятков) имеет смысл проанализировать распределение весов объектов. Объекты с наибольшими весами w_i , скорее всего, являются шумовыми выбросами, которые стоит исключить из выборки, после чего начать построение композиции заново. Вообще, бустинг можно использовать как универсальный метод фильтрации выбросов перед применением любого другого метода классификации.

Обобщающая способность бустинга. В течение последних 10 лет бустинг остаётся одним из наиболее популярных методов машинного обучения, наряду с нейронными сетями и машинами опорных векторов. Основные причины — простота, универсальность, гибкость (возможность построения различных модификаций), и, главное, неожиданно высокая обобщающая способность.

Бустинг над решающими деревьями считается одним из наиболее эффективных методов с точки зрения качества классификации. Во многих экспериментах наблюдалось практически неограниченное уменьшение частоты ошибок на независимой тестовой выборке по мере наращивания композиции. Более того, качество на тестовой выборке часто продолжало улучшаться даже после достижения безошибочного распознавания всей обучающей выборки [23]. Это перевернуло существовавшие долгое время представления о том, что для повышения обобщающей способности необходимо ограничивать сложность алгоритмов. На примере бустинга стало понятно, что хорошим качеством могут обладать сколь угодно сложные композиции, если их «правильно настраивать».

Впоследствии феномен бустинга получил теоретическое обоснование. Оказалось, что взвешенное голосование не увеличивает эффективную сложность алгоритма, а лишь сглаживает ответы базовых алгоритмов. Количественные оценки обобщающей способности бустинга формулируются в терминах отступа [18]. Эффективность бустинга объясняется тем, что по мере добавления базовых алгоритмов увеличиваются отступы обучающих объектов $M_i = y_i \sum_{t=1}^T \alpha_t b_t(x_i)$. Причём бустинг продолжает «раздвигать» классы даже после достижения безошибочной классификации обучающей выборки.

К сожалению, теоретические оценки обобщающей способности [18] дают лишь качественное обоснование феномену бустинга. Хотя они существенно точнее более общих оценок Вапника-Червоненкиса [42], всё же они сильно завышены, и требуемая длина обучающей выборки оценивается величиной порядка $10^4 \dots 10^6$.

Более основательные эксперименты показали, что иногда бустинг всё же переобучается [36, 35]. По мнению автора статьи [25] причины эффективности бустинга до конца ещё не поняты и его дальнейшее улучшение остаётся открытой проблемой.

Достоинства AdaBoost.

- 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 w_i принимают наибольшие значения.

Недостатки AdaBoost.

- AdaBoost склонен к переобучению при наличии значительного уровня шума в данных. Экспоненциальная функция потерь слишком сильно увеличивает веса «наиболее трудных» объектов, на которых ошибаются многие базовые алгоритмы. Однако именно эти объекты чаще всего оказываются шумовыми выбросами. В результате AdaBoost начинает настраиваться на шум, что ведёт к переобучению. Проблема решается путём удаления выбросов или применения менее «агрессивных» функций потерь. В частности, алгоритм LogitBoost использует логистическую функцию [24].
- AdaBoost требует достаточно длинных обучающих выборок. Другие методы линейной коррекции, в частности, бэггинг, способны строить алгоритмы сопоставимого качества по меньшим выборкам данных.
- Жадная стратегия последовательного добавления приводит к построению неоптимального набора базовых алгоритмов. Для улучшения композиции можно периодически возвращаться к ранее построенным алгоритмам и обучать их заново. Для улучшения коэффициентов α_t можно оптимизировать их ещё раз по окончании процесса бустинга с помощью какого-нибудь стандартного метода построения линейной разделяющей поверхности. Рекомендуется использовать для этой цели SVM — машины опорных векторов [43].
- Бустинг может приводить к построению громоздких композиций, состоящих из сотен алгоритмов. Такие композиции исключают возможность содержательной интерпретации, требуют больших объёмов памяти для хранения базовых алгоритмов и существенных затрат времени на вычисление классификаций.

1.2.2 Алгоритм AnyBoost

Алгоритм бустинга несложно обобщается на тот случай, когда пороговая функция потерь оценивается сверху произвольной невозрастающей функцией отступа $\mathcal{L}(M)$, не обязательно экспонентой, а базовые алгоритмы b_t возвращают произвольные вещественные значения, не обязательно ± 1 . По-прежнему рассматривается взвешенное голосование (1.3), но вместо (1.5) записывается оценка

$$Q_T \leq \tilde{Q}_T = \sum_{i=1}^{\ell} \mathcal{L} \left(\underbrace{y_i \sum_{t=1}^T \alpha_t b_t(x_i)}_{M_T(x_i)} \right) = \sum_{i=1}^{\ell} \mathcal{L} (M_{T-1}(x_i) + y_i \alpha_T b_T(x_i)). \quad (1.7)$$

Рассмотрим функцию потерь \mathcal{L} как функцию параметра α_T ,

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

и линеаризуем её в окрестности значения $\alpha_T = 0$, разложив в ряд Тейлора и отбросив старшие члены, начиная с квадратичного: $\lambda(\alpha_T) \approx \lambda(0) + \alpha_T \lambda'(0)$. Это приведёт к линеаризации функционала \tilde{Q}_T по параметру α_T :

$$\tilde{Q}_T \approx \sum_{i=1}^{\ell} \mathcal{L} (M_{T-1}(x_i)) - \alpha_T \sum_{i=1}^{\ell} \underbrace{-\mathcal{L}'(M_{T-1}(x_i))}_{w_i} y_i b_T(x_i),$$

где w_i — веса объектов. Если параметр α_T фиксирован, то для минимизации \tilde{Q}_T необходимо строить базовый алгоритм b_T , исходя из принципа *явной максимизации отступа* (direct optimization of margin) [33, 32]:

$$\sum_{i=1}^{\ell} w_i y_i b(x_i) \rightarrow \max_b.$$

После того, как b_T построен, параметр α_T определяется путём одномерной минимизации функционала (1.7). Итерации этих двух шагов приводят к обобщённому алгоритму бустинга AnyBoost [17]. Его подробная запись приведена в Алгоритме 1.2.

Утв. 1.4. AnyBoost переходит в AdaBoost при $b_t: X \rightarrow \{-1, +1\}$ и $\mathcal{L}(M) = e^{-M}$.

1.2.3 Голосование по большинству

Рассмотрим задачу классификации с двумя классами $Y = \{-1, +1\}$, пространством оценок $R = \mathbb{R}$ и решающим правилом $C(b) = \text{sign}(b)$. В качестве корректирующей операции возьмём простое голосование. Рассмотрим функционал качества композиции a , равный числу ошибок на обучении:

$$Q(a; X^\ell) = \sum_{i=1}^{\ell} [y_i a(x_i) < 0] = \sum_{i=1}^{\ell} \underbrace{[y_i b_1(x_i) + \dots + y_i b_T(x_i) < 0]}_{M_{iT}}$$

Алгоритм 1.2. AnyBoost — обобщённый бустинг с произвольной функцией потерь

Вход:

X^ℓ, Y^ℓ — обучающая выборка; T — максимальное число базовых алгоритмов;

Выход:

базовые алгоритмы и их веса $\alpha_t b_t$, $t = 1, \dots, T$;

- 1: инициализация отступов: $M_i := 0$, $i = 1, \dots, \ell$;
 - 2: **для всех** $t = 1, \dots, T$, пока не выполнен критерий останова
 - 3: $b_t := \arg \max_b \sum_{i=1}^{\ell} w_i y_i b(x_i)$, где $w_i = -\mathcal{L}'(M_i)$, $i = 1, \dots, \ell$;
 - 4: $\alpha_t := \arg \min_{\alpha > 0} \sum_{i=1}^{\ell} \mathcal{L}(M_i + \alpha b_t(x_i) y_i)$;
 - 5: пересчёт отступов: $M_i := M_i + \alpha_t b_t(x_i) y_i$; $i = 1, \dots, \ell$;
-

Обозначим через M_{it} *отступ* (margin) объекта x_i , вычисленный для композиции из первых t базовых алгоритмов:

$$M_{it} = y_i b_1(x_i) + \dots + y_i b_t(x_i).$$

Если отступ отрицателен, $M_{it} < 0$, то композиция первых t базовых алгоритмов допускает ошибку на объекте x_i . Чтобы компенсировать ошибки композиции, будем обучать базовый алгоритм b_{t+1} не на всей выборке X^ℓ , а только на объектах с наименьшими значениями M_{it} . Или, что то же самое, будем минимизировать функционал $Q(b_{t+1}; W^\ell)$ с весами объектов $w_i = [M_{it} \leq M_0]$.

Выбирать параметр M_0 следует так, чтобы в обучающую выборку попало не слишком мало объектов (иначе будут строиться базовые алгоритмы слишком низкого качества), но и не слишком много (иначе будут строиться почти одинаковые алгоритмы). Поэтому вместо M_0 удобнее ввести параметр длины обучающих подвыборок ℓ_1 и подбирать его оптимальное значение.

Фиксация длины обучения. Упорядочим объекты выборки X^ℓ по возрастанию значений отступов M_{it} . Объекты с одинаковыми отступами M_{it} , если таковые имеются, договоримся располагать в случайном порядке. Возьмём первые ℓ_1 объектов упорядоченной выборки, и положим для них $w_i = 1$, для остальных объектов положим $w_i = 0$. Данный вариант представлен в Алгоритме 1.3.

Параметр ℓ_1 можно задавать априори или подбирать по критерию минимума ошибок на скользящем контроле. Увеличение ℓ_1 повышает качество базовых алгоритмов, но уменьшает их различность. В предельном случае, когда $\ell_1 = \ell$, все базовые алгоритмы становятся одинаковыми и применение коррекции теряет смысл. Оптимизация параметра ℓ_1 позволяет найти компромисс между качеством и различностью базовых алгоритмов.

Обратим внимание, что в описании алгоритма вместо M_{it} фигурирует M_i , так как при реализации можно записывать $M_{i,t+1}$ на место, занимаемое M_{it} .

Оптимизация длины обучения. Вместо фиксации параметра ℓ_1 можно формировать несколько обучающих подвыборок различной длины, и на каждой итерации находить оптимальное значение ℓ_1 . Рассмотрим подробнее шаг 6 Алгоритма 1.3.

Алгоритм 1.3. Построение композиции для голосования по большинству

Параметры:

X^ℓ, Y^ℓ — обучающая выборка; μ — метод обучения базовых алгоритмов;
 ℓ_1 — длина обучающих подвыборок;

- 1: инициализировать веса и отступы: $w_i := 1; M_i := 0$ для всех $i = 1, \dots, \ell$;
 - 2: **для всех** $t = 1, \dots, T$, пока не выполнен критерий останова
 - 3: $b_t := \mu(X^\ell, Y^\ell, W^\ell)$;
 - 4: $M_i := M_i + y_i b_t(x_i)$ для всех $i = 1, \dots, \ell$;
 - 5: упорядочить выборку X^ℓ по возрастанию значений отступов M_i ;
 - 6: выбрать ℓ_1 ;
 - 7: $w_i := [i \leq \ell_1]$ для всех $i = 1, \dots, \ell$;
-

В качестве начального приближения разумно взять оптимальное значение ℓ_1 с предыдущей итерации. Затем начать уменьшать ℓ_1 , удаляя по одному объекту в порядке убывания отступов M_{it} . Затем увеличивать ℓ_1 , добавляя по одному объекту в порядке возрастания M_{it} . По каждой подвыборке производится обучение нового базового алгоритма. В итоге выбирается тот вариант базового алгоритма, для которого функционал качества всей композиции принимает оптимальное значение. Качество композиции предпочтительно оценивать на контрольных данных, если имеется такая возможность.

Построение алгоритма путём удаления/добавления одного из обучающих объектов для многих методов (в частности, для SVM) производится намного эффективнее, чем перенастройка алгоритма заново по всей выборке. Для таких методов описанный способ позволяет довольно быстро находить оптимальное значение параметра ℓ_1 на каждой итерации.

Идентификация выбросов. В практических задачах выборка, как правило, содержит некоторое количество шумовых выбросов, на которых даже «хорошие» алгоритмы будут допускать ошибки. Такие объекты было бы разумно исключить из выборки, так как попытка настройки на шум приводит к искажению разделяющей поверхности и ухудшению качества классификации. В нашем случае обучение базового алгоритма b_{t+1} по объектам с наименьшими отступами как раз и означает «настройку на шум». Проблема решается путём введения ещё одного параметра $\ell_0 < \ell_1$. Обучающая подвыборка формируется на шаге 7 исходя из условия $w_i := [\ell_0 \leq i \leq \ell_1]$. Параметр ℓ_0 можно либо фиксировать, либо подбирать путём оптимизации на каждой итерации, аналогично ℓ_1 .

Вещественные веса объектов. Альтернативная эвристика заключается в том, чтобы не менять состав обучающей выборки, а лишь увеличивать веса тех объектов, на которых часто ошибались предыдущие алгоритмы. Например, в методе *взвешенного большинства* (weighted majority) вместо шага 7 применяется мультипликативный пересчёт весов [31]:

$$w_i = \begin{cases} w_i/\gamma, & \text{если } y_i b_t(x_i) > 0; \\ w_i\gamma, & \text{если } y_i b_t(x_i) < 0. \end{cases}$$

Легко видеть, что тогда на t -й итерации $w_i = \gamma^{-Mit}$. Здесь $\gamma > 1$ — параметр метода настройки, который приходится задавать априори.

Для применения мультипликативного пересчёта необходимо, чтобы метод обучения базовых алгоритмов минимизировал функционал $Q(b_t; W^\ell)$ при произвольном векторе весов W^ℓ . Далеко не все стандартные методы способны учитывать веса, хотя соответствующее обобщение в большинстве случаев строится несложно.

Преобразование простого голосования во взвешенное. Если T различных базовых алгоритмов уже построены, то вектор оценок $(b_1(x), \dots, b_T(x))$ можно принять за новое признаковое описание объекта x и построить линейную разделяющую гиперплоскость в пространстве R^T по той же обучающей выборке X^ℓ .

Для этого годится любой стандартный метод обучения линейных классификаторов: линейный дискриминант Фишера (раздел ??), логистическая регрессия (раздел ??), однослойный перцептрон (раздел ??), или метод опорных векторов SVM (раздел ??). Последний имеет преимущество, поскольку он максимизирует величину зазора между классами, что способствует более надёжной классификации.

Желательно, чтобы применяемый линейный метод допускал введение дополнительного ограничения на коэффициенты $\alpha_t \geq 0$. Отрицательный коэффициент α_t свидетельствует о том, что оператор b_t выдаёт ответы настолько ненадёжные, что их надо было бы учитывать с противоположным знаком. Такие базовые алгоритмы лучше вовсе исключить из композиции, обнулив α_t .

Пример 1.4. Самый простой линейный метод — «наивный» байесовский классификатор (стр. ??). Он исходит из предположения, что $b_1(x), \dots, b_T(x)$ являются независимыми случайными величинами. Нетрудно доказать, что в этом случае коэффициенты взвешенного голосования вычисляются по явной формуле [30]:

$$\alpha_t = \ln \frac{1 - p_t}{p_t}, \quad t = 1, \dots, T,$$

где p_t — вероятность ошибок базового алгоритма b_t на обучающей выборке. Чем меньше ошибок допускает алгоритм b_t , тем больше его вес α_t . В качестве оценки вероятности часто берут частоту ошибок ν_t , или $\nu_t + 1/\ell$, чтобы знаменатель не обращался в нуль. Если алгоритм b_t допускает более половины ошибок, то $\alpha_t < 0$. Такой алгоритм лучше вовсе исключить из композиции, положив $\alpha_t = 0$.

Обобщение на большое число классов. Пусть теперь число классов произвольно, $|Y| = M$. Базовые алгоритмы $b_t: X \rightarrow \mathbb{R}$ всё равно будем строить так, чтобы они решали двухклассовую задачу, отделяя заданный класс c_t от всех остальных классов. Обозначим через $T_y = \{t: c_t = y\}$ множество индексов всех базовых алгоритмов, предназначенных для отделения класса y . Запишем алгоритм простого голосования, применив решающее правило из Примера 1.2:

$$a(x) = \arg \max_{y \in Y} \frac{1}{|T_y|} \sum_{t \in T_y} b_t(x).$$

Соответственно изменится в Алгоритме 1.3 и вычисление отступов M_{it} :

$$M_{it} = \frac{1}{|T_{y_i}|} \sum_{t \in T_{y_i}} b_t(x) - \max_{y \in Y \setminus \{y_i\}} \frac{1}{|T_y|} \sum_{t \in T_y} b_t(x).$$

В остальном все рассуждения и Алгоритм 1.3 остаются теми же.

1.2.4 Голосование по старшинству

Рассмотрим задачу классификации с произвольным числом классов, $|Y| = M$. Положим $R = \{0, 1\}$. Будем использовать базовые алгоритмы, осуществляющие классификацию на 2 непересекающихся класса, $b_t: X \rightarrow \{0, 1\}$. В качестве корректирующей операции возьмём голосование по старшинству, см. Алгоритм ???. Решающие правила использовать не будем.

Рассмотрим, что происходит при добавлении базового алгоритма b_t . Если $b_t(x) = 1$, то говорят, что « b_t выделяет объект x », и объект x относится к классу c_t . Обозначим через U_t подмножество объектов выборки, не выделенных ни одним из предыдущих базовых алгоритмов:

$$U_t = \{x_i \in X^\ell : b_1(x_i) = \dots = b_{t-1}(x_i) = 0\}.$$

Если $x_i \in X^\ell \setminus U_t$, то объект x_i уже классифицирован, и значение $b_t(x_i)$ не влияет на ответ композиции $a(x_i)$. Для таких объектов вес w_i положим равным нулю.

Если $x_i \in U_t$ и $y_i \neq c_t$, то ошибка базового алгоритма, $b_t(x_i) = 1$, приведёт к ошибке всей композиции: $a(x_i) = c_t \neq y_i$. Положим для таких объектов $w_i = 1$.

Если $x_i \in U_t$ и $y_i = c_t$, то ошибка базового алгоритма, $b_t(x_i) = 0$, приведёт к тому, что композиция из первых t алгоритмов откажется от классификации x_i . Эта ошибка ещё может быть исправлена следующими алгоритмами. Поэтому для таких объектов положим $w_i = \lambda$, где $\lambda \in [0, 1]$ — величина штрафа за отказ.

Итак, минимизация функционала $Q(a) = Q(F(b_1, \dots, b_t))$ по базовому алгоритму b_t , относящему объекты к классу c_t , эквивалентна минимизации функционала

$$Q(b_t) = \sum_{i=1}^{\ell} w_i [b_t(x_i) \neq [y_i = c_t]].$$

Это стандартный функционал взвешенного числа ошибок. Для его минимизации применим стандартный метод обучения $b_t = \mu(X^\ell, Z^\ell, W^\ell)$, передав ему вектор весов объектов $W^\ell = (w_1, \dots, w_\ell)$ и бинарный вектор ответов $Z^\ell = (z_1, \dots, z_\ell)$, где $z_i = [y_i = c_t]$. В результате базовый алгоритм b_t будет стремиться выделить как можно больше объектов «своего» класса c_t и как можно меньше «чужих» объектов.

Алгоритм 1.4 описывает весь процесс построения композиции.

Стратегии выбора параметра λ . Параметр λ позволяет выбрать «золотую середину» между двумя крайними стратегиями: выделить хоть сколько-нибудь объектов «своего» класса ($\lambda = 0$) и в точности отделить все «свои» объекты от всех остальных ($\lambda = 1$). Первая задача, как правило, оказывается слишком простой, а вторая — слишком сложной. При выборе параметра λ можно руководствоваться следующими соображениями.

- Представляется разумным брать $\lambda \ll 1$, поскольку покрытие «чужого» объекта гарантирует ошибку, а не-покрытие своего объекта может быть компенсировано последующими базовыми алгоритмами. Например, можно начинать со значения $\lambda = 0.1$, постепенно увеличивая его от итерации к итерации.

Алгоритм 1.4. Построение композиции для голосования по старшинству

Параметры:

X^ℓ, Y^ℓ — обучающая выборка; μ — метод обучения базовых алгоритмов;
 λ — штраф за отказ от классификации;

- 1: инициализировать веса объектов: $w_i := 1$ для всех $i = 1, \dots, \ell$;
 - 2: **для всех** $t = 1, \dots, T$, пока не выполнен критерий останова
 - 3: выбрать класс c_t ;
 - 4: вычислить бинарный вектор ответов Z^ℓ для настройки b_t :
 $z_i := [y_i = c_t]$ для всех $i = 1, \dots, \ell$;
 - 5: **если** $w_i \neq 0$ **то** $w_i := (1 - z_i) + \lambda z_i$ для всех $i = 1, \dots, \ell$;
 - 6: $b_t := \mu(X^\ell, Z^\ell, W^\ell)$;
 - 7: **если** $b_t(x_i) = 1$ **то** $w_i := 0$ для всех $i = 1, \dots, \ell$;
-

- Уменьшение λ способствует росту числа базовых алгоритмов и в конечном итоге может приводить к переобучению. С другой стороны, увеличение λ приводит к переупрощению композиции за счёт увеличения числа ошибок. Таким образом, параметр λ позволяет управлять сложностью и обобщающей способностью композиции. Для практического определения оптимального значения λ можно использовать скользящий контроль.

Стратегии формирования последовательности c_1, \dots, c_T . В Алгоритме 1.4 стратегия выбора класса c_t на шаге 4 преднамеренно не фиксирована, так как возможны различные варианты, выбор которых определяется особенностями задачи.

- Выбирается тот класс, в котором осталось больше непокрытых объектов.
- Выбирается тот класс, для которого удаётся получить лучшее значение функционала качества. При этом время работы алгоритма увеличивается в M раз.
- Задаётся приоритетный порядок классов $Y = \{v_1, \dots, v_M\}$, разбивающий последовательность базовых алгоритмов на списки по T_m алгоритмов для каждого из классов v_m , $m = 1, \dots, M$:

$$\underbrace{b_{v_1 1}(x), \dots, b_{v_1 T_1}(x)}_{\text{за класс } v_1}, \underbrace{b_{v_2 1}(x), \dots, b_{v_2 T_2}(x)}_{\text{за класс } v_2}, \dots, \underbrace{b_{v_M 1}(x), \dots, b_{v_M T_M}(x)}_{\text{за класс } v_M}.$$

При этом базовые алгоритмы внутри блоков оказываются независимыми — их можно свободно переставлять местами и интерпретировать по отдельности.

- Задаётся приоритетный порядок классов $Y = \{v_1, \dots, v_M\}$, как в предыдущем случае. Однако теперь для каждого класса строится независимый список базовых алгоритмов, способный отделять объекты данного класса от объектов всех остальных классов. Для этого перед построением списка для следующего класса все объекты с нулевыми весами возвращаются в обучающую выборку. Соответственно, модифицируется правило пересчёта весов на шаге 5:

$$\text{если } w_i \neq 0 \text{ или } c_t \neq c_{t-1} \text{ то } w_i := (1 - z_i) + \lambda z_i;$$

При такой стратегии композиция становится ещё лучше интерпретируемой, поскольку теперь и блоки можно переставлять местами и интерпретировать по отдельности. Во многих приложениях независимость интерпретации базовых алгоритмов от их положения в композиции является заметным преимуществом.

О кусочно-линейных разделяющих поверхностях. Если базовые алгоритмы строят линейные разделяющие поверхности, то комитет старшинства является кусочно-линейной разделяющей поверхностью.

Пример 1.5. Задача на плоскости: $X = \mathbb{R}^2$, $Y = \{o, \times\}$, $b_t(x)$ — полуплоскости.

§1.3 Стохастические методы построения композиций

1.3.1 Бэггинг и метод случайных подпространств

Базовые алгоритмы, составляющие линейную комбинацию, должны быть в достаточной степени различными, чтобы их погрешности компенсировали друг друга. Нет никакого смысла складывать одинаковые (или почти одинаковые) алгоритмы.

В бустинге и других последовательных методах различность достигается благодаря пересчёту весов объектов. Но возможна и другая стратегия повышения различности, когда базовые алгоритмы настраиваются независимо друг от друга на случайно выбранных подмножествах обучающей выборки, либо на различных случайных подмножествах признаков. Ещё один способ обеспечить различность — выбирать случайные начальные приближения при оптимизации вектора параметров (обычно так поступают при настройке нейронных сетей), либо применять стохастические алгоритмы оптимизации. Полученные базовые алгоритмы объединяются в композицию с помощью простого или взвешенного голосования. В случае взвешенного голосования для настройки коэффициентов α_t применяются стандартные линейные методы.

Бэггинг. Метод *бэггинга* (bagging, bootstrap aggregation) был предложен Л. Брейманом в 1996 году [19]. Из исходной обучающей выборки длины ℓ формируются различные обучающие подвыборки той же длины ℓ с помощью бутстрепа — случайного выбора с возвращениями. При этом некоторые объекты попадают в подвыборку по несколько раз, некоторые — ни разу. Можно показать, что доля объектов, оказавшихся в каждой подвыборке, стремится к $1 - e^{-1} \approx 0.632$ при $\ell \rightarrow \infty$. Базовые алгоритмы, обученные по подвыборкам, объединяются в композицию с помощью простого голосования.

Эффективность бэггинга объясняется двумя обстоятельствами. Во-первых, благодаря различности базовых алгоритмов, их ошибки взаимно компенсируются при голосовании. Во-вторых, объекты-выбросы могут не попадать в некоторые обучающие подвыборки. Тогда алгоритм, построенный по подвыборке, может оказаться даже точнее алгоритма, построенного по полной выборке.

Бэггинг особенно эффективен на малых выборках, когда исключение даже небольшой доли обучающих объектов приводит к построению существенно различных базовых алгоритмов. В случае сверхбольших избыточных выборок приходится строить подвыборки меньшей длины $\ell_0 \ll \ell$, при этом возникает задача подбора оптимального значения ℓ_0 .

Алгоритм 1.5. Бэггинг и RSM

Параметры:

- ℓ' — длина обучающих подвыборок;
 - n' — длина признакового подописания;
 - ε_1 — порог качества базовых алгоритмов на обучении;
 - ε_2 — порог качества базовых алгоритмов на контроле;
-

- 1: **для всех** $t = 1, \dots, T$, пока не выполнен критерий останова
 - 2: $U :=$ случайное подмножество X^ℓ длины ℓ' ;
 - 3: $\mathcal{G} :=$ случайное подмножество \mathcal{F} длины n' ;
 - 4: $b_t := \mu(\mathcal{G}, U)$;
 - 5: **если** $Q(b_t, U) > \varepsilon_1$ или $Q(b_t, X^\ell \setminus U) > \varepsilon_2$ **то**
 - 6: не включать b_t в композицию;
-

RSM. В *методе случайных подпространств* (random subspace method, RSM) базовые алгоритмы обучаются на различных подмножествах признакового описания, которые также выделяются случайным образом [39]. Этот метод предпочтителен в задачах с большим числом признаков и относительно небольшим числом объектов, а также при наличии избыточных неинформативных признаков. В этих случаях алгоритмы, построенные по части признакового описания, могут обладать лучшей обобщающей способностью по сравнению с алгоритмами, построенными по всем признакам.

Обобщение бэггинга и RSM приводит к Алгоритму 1.5. Пусть объекты описываются набором признаков $\mathcal{F} = \{f_1, \dots, f_n\}$ и метод обучения $\mu(\mathcal{G}, U)$ строит алгоритмический оператор по обучающей подвыборке $U \subseteq X^\ell$, используя только часть признакового описания $\mathcal{G} \subseteq \mathcal{F}$. Выборка объектов на шаге 2 может производиться как с возвращениями, так и без возвращений; вообще говоря, это ещё один (бинарный) параметр алгоритма. Числа $\ell' = |U| \leq \ell$ и $n' = |\mathcal{G}| \leq n$ являются параметрами метода. Алгоритм 1.5 соответствует бэггингу при $\mathcal{G} = \mathcal{F}$, и методу случайных подпространств при $U = X^\ell$.

Шаги 5–6 осуществляют фильтрацию неудачных базовых алгоритмов. Базовый алгоритм b_t признаётся неудачным, если его качество на обучающей подвыборке хуже заданного порога ε_1 , либо качество на контрольных данных $X^\ell \setminus U$ хуже ε_2 . Это может происходить в тех случаях, когда в обучающей выборке случайно оказалось слишком много шумовых выбросов, или среди признаков оказалось слишком мало информативных.

Ни один из методов построения линейных композиций не является однозначно лучшим. Как обычно, для каждого метода можно найти прикладные задачи, на которых он превосходит остальные. Приведём некоторые выводы, вытекающие из эмпирического сравнения бустинга, бэггинга и RSM [39].

- Бустинг работает лучше на больших обучающих выборках. При этом он лучше воспроизводит границы классов сложной формы.
- Бэггинг и RSM предпочтительны для коротких обучающих выборок.

- RSM безусловно предпочтительнее в тех случаях, когда признаков больше, чем объектов и/или когда среди признаков много неинформативных.
- Бэггинг и RSM допускают эффективное распараллеливание, в то время как бустинг выполняется строго последовательно.

1.3.2 Кооперативная коэволюция

Как и в предыдущем разделе, будем полагать, что объекты описываются набором признаков $\mathcal{F} = \{f_1, \dots, f_n\}$, и задан метод обучения μ , позволяющий строить базовые алгоритмы $b = \mu(\mathcal{G}, U)$ по различным подвыборкам $U \subseteq X^\ell$ и различным частям признакового описания $\mathcal{G} \subseteq \mathcal{F}$.

Общая идея применения *генетических алгоритмов* для глобальной оптимизации функционала качества $Q(a)$ в некотором сложно устроенном пространстве $a \in A$ заключается в следующем. Формируется популяция индивидов — конечное множество $\Pi(t) \subset A$. Над ними производятся генетические операции, порождающее новое поколение индивидов. Обычно это бинарная операция рекомбинации (скрещивания) и унарная — мутации. Из большого числа порождённых индивидов в следующее поколение $\Pi(t+1)$ отбираются только наиболее *адаптивные* — наилучшие с точки зрения функционала качества $Q(a)$. Эволюционный процесс смены поколений останавливается, когда качество лучшего индивида перестаёт улучшаться.

Для построения композиции обучаемых алгоритмов хорошо подходит специфическая модель эволюции, называемая *кооперативной коэволюцией* [34]. Как всякий эволюционный алгоритм, он представляет собой итерационный процесс смены поколений, см. Алгоритм 1.6.

Инициализация нулевого поколения производится случайным образом.

На t -м поколении строится не одна, а $p(t)$ популяций $\Pi_1(t), \dots, \Pi_{p(t)}(t)$. Каждая популяция $\Pi_j(t)$ представляет собой множество индивидов. Каждый индивид кодируется $(\ell + n)$ -мерным бинарным вектором, составленным из характеристических векторов подмножества объектов $X' \subseteq \{x_1, \dots, x_\ell\}$ и подмножества признаков $G' \subseteq \{g_1, \dots, g_n\}$. Таким образом, каждому индивиду $v_j \in \Pi_j(t)$ соответствует пара подмножеств (G', X') , к которой можно применить метод обучения и получить базовый алгоритм $b_j = \mu(G', X') \equiv \mu(v_j)$.

Для оценки *адаптивности* (fitness) индивида v_j вычисляется оценка качества композиции, в которой на j -м месте стоит алгоритм b_j , на остальных местах — наиболее адаптивные алгоритмы b_s^* , взятые по одному из каждой популяции $\Pi_s(t)$, $s \neq j$:

$$\varphi(v_j) = Q(F(b_1^*, \dots, b_{j-1}^*, \mu(v_j), b_{j+1}^*, \dots, b_{p(t)}^*), X^\ell),$$

Адаптивность оценивается для каждого индивида в каждой популяции.

Затем производится селекция индивидов. В каждой популяции отбирается ограниченное количество наиболее адаптивных индивидов. К ним применяются генетические операции рекомбинации (crossover), мутации и селекции, в результате которых порождается новое поколение $\Pi_j(t+1)$, $j = 1, \dots, p(t)$. Из каждого поколения t отбирается и запоминается лучшая композиция вида $F(b_1, \dots, b_{p(t)})$.

В момент смены поколений может приниматься решение об увеличении или уменьшении числа популяций. Если популяция даёт слишком малый вклад в композицию в течение некоторого числа поколений, то она удаляется. Если процесс эволюции вошел в состояние стагнации, то есть качество лучших композиций

Алгоритм 1.6. Коэволюционное обучение алгоритмической композиции CCEL

Вход:

- T_{\max} — максимальное число поколений;
- N_0 — размер основной популяции;
- N_1 — размер промежуточной популяции;
- N_2 — размер элиты, переходящей в следующее поколение без изменений;

Выход:

композиция $a = F(b_1, \dots, b_p)$;

- 1: начальное число популяций: $p(1) := 1$;
 - 2: создать популяцию: $\Pi_1(1) := \text{Инициализация}(N_0)$;
 - 3: **для всех** поколений $t := 1, \dots, T_{\max}$
 - 4: **для всех** популяций $\Pi_j(t)$, $j := 1, \dots, p(t)$
 - 5: породить промежуточную популяцию:
 - $\Pi'_j := \text{Рекомбинация}(\Pi_j(t), N_1)$;
 - $\Pi''_j := \text{Мутация}(\Pi'_j) \cup \text{Селекция}(\Pi_j(t), N_2)$;
 - 6: **для всех** $v_j \in \Pi''_j$
 - 7: оценить адаптивность индивида $\varphi(v_j)$;
 - 8: $Q_t := \min\{Q_t, \varphi(v_j)\}$;
 - 9: запомнить лучшего индивида в популяции:
 - $v_j^* := \arg \min_{v \in \Pi''_j} \varphi(v)$; $b_j^* := \mu(v_j^*)$;
 - 10: отобрать лучших индивидов в следующее поколение:
 - $\Pi_j(t+1) := \text{Селекция}(\Pi''_j, N_0)$;
 - 11: **если** $\text{Вклад}(\Pi_j) < \delta$ **то**
 - 12: удалить популяцию Π_j ; $p(t+1) := p(t) - 1$;
 - 13: **если** $\text{Стагнация}(Q, t)$ **то**
 - 14: $p(t+1) := p(t) + 1$; добавить $\Pi_{p(t+1)}(t+1) := \text{Инициализация}(N_0)$;
 - 15: **если** $\text{Останов}(Q, t)$ **то**
 - 16: **выход**;
 - 17: **вернуть** композицию $F(b_1^*, \dots, b_{p(t)}^*)$, на которой достигается $\min_t Q_t$.
-

перестало заметно изменяться, то создаётся новая популяция. Если и эти меры не помогают выйти из стагнации, то процесс эволюции останавливается.

Основное отличие кооперативной коэволюции от стандартных генетических алгоритмов в том, что функция адаптивности оценивает не качество алгоритма b_j в отдельности, а его полезность для композиции. В ходе эволюции базовые алгоритмы обучаются кооперировать друг с другом с целью наилучшего решения поставленной задачи. При этом каждая популяция (следовательно, каждый базовый алгоритм) специализируется в своей области объектов и в своём подпространстве признаков.

Описанный метод получил название CCEL — Cooperative Coevolution Ensemble Learner [3]. Он имеет большое число параметров и большую свободу выбора различных эвристик. Рассмотрим некоторые из них более подробно.

$\text{Инициализация}(N_0)$ — процедура, создающая N_0 индивидов. Каждый индивид формируется случайным образом. Для этого задаются два параметра — вероятность включения объекта q_X и вероятность включения признака q_G .

Селекция(Π, N) — генетическая операция, отбирающая N наиболее адаптивных индивидов популяции Π . В методе CSEL она используется дважды. На шаге 5 — для переноса лучших индивидов (*элиты*) из популяции родителей в популяцию потомков. На шаге 11 — для естественного отбора, при котором из N_1 потомков только N_0 лучших переносятся в следующее поколение.

Рекомбинация(Π, N_1) — генетическая операция, порождающая N_1 новых индивидов путём попарного скрещивания индивидов популяции Π . Родительские пары выбираются случайным образом, возможно, с поощрением наиболее адаптивных индивидов. Для формирования потомка в методе CSEL используется так называемый *равномерный кроссинговер* (uniform crossover) — каждый бит потомка равновероятно выбирается у одного из родителей.

Мутация(Π) — генетическая операция, производящая случайные изменения в индивидах популяции Π . В качестве параметра алгоритма задаётся вероятность инверсии бита в бинарном коде индивида.

Вклад(Π_j) — функция, оценивающая вклад популяции Π_j в композицию. При использовании *проверки изъятием* строятся две композиции — с участием и без участия j -го базового алгоритма. Соответственно, вычисляются оценки их качества: Q_{jt} и \bar{Q}_{jt} . Вклад j -й популяции вычисляется как средняя разность этих оценок за последние d итераций: $\frac{1}{d} \sum_{\tau=t-d+1}^t (\bar{Q}_{j\tau} - Q_{j\tau})$. Если вклад слишком мал, популяция целиком удаляется. Метод проверки изъятием — надёжный, но ресурсоёмкий. Возможен альтернативный приём: если строится линейная корректирующая операция, то вклад можно оценивать как среднее значение веса j -го базового алгоритма за последние d итераций: $\frac{1}{d} \sum_{\tau=t-d+1}^t \alpha_j(\tau)$. Если это значение окажется меньше заданного достаточно малого порога, популяция целиком удаляется.

Стагнация(Q, t) — критерий, проверяющий наступление стагнации в последовательности $Q = \{Q_t\}$ в момент времени t . Например, это может быть отсутствие заметных улучшений качества на протяжении последних d поколений: $Q_{t-d} - Q_t < \varepsilon$, где d, ε — параметры алгоритма. При наступлении стагнации создаётся новая популяция, в надежде на то, что увеличение сложности композиции сможет привести к улучшению её качества.

Останов(Q, t) — критерий останова, проверяющий наступление длительной стагнации по условию: $Q_{t-D} - Q_t < \varepsilon$, где $D \gg d$ — параметр алгоритма. Должно состояться по меньшей мере несколько безуспешных попыток добавить ещё одну популяцию, прежде чем станет ясно, что дальнейшее изменение структуры композиции не способно её улучшить.

Преимущества метода CSEL.

- Эксперименты на реальных данных показали, что качество классификации (обобщающая способность) CSEL не хуже, чем у бустинга или бэггинга [3, 28].
- CSEL строит очень короткие композиции. Как правило, от 3 до 6 базовых алгоритмов бывает достаточно, тогда как бустинг и бэггинг набирают сотни алгоритмов для достижения сопоставимого качества.
- CSEL применим к любым базовым алгоритмам и методам их обучения.
- Автоматически отбираются информативные объекты и признаки, что даёт дополнительную информацию для понимания задачи.

- Процесс обучения CSEL является *алгоритмом с произвольным временем работы* (anytime algorithm). Обучение может быть в любой момент прервано для получения лучшей из найденных композиций, и вновь возобновлено для поиска ещё лучшего решения. Применение таких алгоритмов предпочтительно в тех случаях, когда имеется простаивающий вычислительный ресурс.

Недостатки метода CSEL.

- Метод CSEL сложен для реализации и имеет большое количество параметров, которые приходится задавать из эвристических соображений.
- Если не использовать распараллеливания, CSEL работает очень долго. Практически применимыми оказываются только очень простые и быстрые методы обучения μ , например, наивный байесовский классификатор (стр. ??).

§1.4 Квазилинейные композиции (смеси алгоритмов)

Линейные корректирующие операции естественным образом обобщаются на тот случай, когда веса базовых алгоритмов не постоянны и зависят от положения объекта x в пространстве X .

Поставим каждому базовому алгоритму $b_t(x)$ в соответствие *функцию компетентности* $g_t(x)$, принимающую значения из отрезка $[0, 1]$. Определим композицию так, чтобы ответ базового алгоритма $b_t(x)$ на объекте x учитывался с весом $g_t(x)$.

Опр. 1.2. *Квазилинейная корректирующая операция есть функция $F: \mathbb{R}^{2T} \rightarrow \mathbb{R}$,*

$$F(b_1, \dots, b_T, g_1, \dots, g_T) = \sum_{t=1}^T g_t b_t. \quad (1.8)$$

Соответственно, алгоритмическая композиция имеет вид

$$a(x) = C\left(\sum_{t=1}^T g_t(x) b_t(x)\right),$$

где C — фиксированное решающее правило. Чем больше значение $g_t(x)$, тем с большим весом учитывается ответ алгоритма $b_t(x)$ на объекте x . Множество

$$\Omega_t = \{x \in X: g_t(x) > g_s(x), s = 1, \dots, T, s \neq t\}$$

называется *областью компетентности* базового алгоритма $b_t(x)$.

Основная идея смесей заключается в применении принципа «разделяй и властвуй». Используя достаточно простые базовые алгоритмы $b_t(x)$ и функции компетентности $g_t(x)$, можно восстанавливать довольно сложные зависимости.

Понятие области компетентности было введено Растригиным в [10]. В англоязычной литературе композиции вида (1.8) принято называть *смесью экспертов* (mixture of experts, ME), базовые алгоритмы — *экспертами*, функции компетентности — *шлюзами* (gates) [16]. Шлюзы определяют, к каким экспертам должен быть направлен объект x . По-русски «смесь экспертов» звучит не очень удачно, поэтому

будем говорить о «смесях алгоритмов». Произведения $g_t(x)b_t(x)$ называются *компонентами смеси*.

Естественным требованием к шлюзам является условие нормировки:

$$\sum_{t=1}^T g_t(x) = 1, \quad \text{для любого } x \in X. \quad (1.9)$$

Если оно выполнено, то для любого объекта x коррекция сводится к усреднению ответов базовых алгоритмов с некоторыми весами, причём результат никогда не выходит за пределы отрезка $[\min_t b_t(x), \max_t b_t(x)]$.

Чтобы гарантировать выполнение условия нормировки (1.9), используют функцию «мягкого максимума» $\text{SoftMax}: \mathbb{R}^T \rightarrow \mathbb{R}^T$:

$$\tilde{g}_t(x) = \text{SoftMax}_t(g_1(x), \dots, g_T(x)) = \frac{e^{\beta g_t(x)}}{e^{\beta g_1(x)} + \dots + e^{\beta g_T(x)}}.$$

Эта функция переводит вектор (g_1, \dots, g_T) в нормированный вектор $(\tilde{g}_1, \dots, \tilde{g}_T)$. При $\beta \rightarrow \infty$, все компоненты \tilde{g}_t стремятся к нулю, кроме одной, соответствующей максимальному значению $\max_t g_t$, которая стремится к единице. Чем больше значение параметра β , тем «чётче» границы областей компетентности.

В задачах классификации с пороговым решающим правилом $C(b) = [b > 0]$ или $C(b) = \text{sign}(b)$ делать нормировку, вообще говоря, не обязательно, поскольку для классификации важен только знак выражения (1.8).

Задача обнаружения нетипичных объектов. Если значение $g_t(x)$ меньше некоторого порога для всех $t = 1, \dots, T$, то можно полагать, что объект x попадает в «область неуверенности», где все алгоритмы не компетентны. Как правило, это связано с тем, что в данной области обучающих прецедентов просто не было. Таким образом, смеси алгоритмов позволяют решать задачу обнаружения нетипичных объектов или *новизны* (novelty detection). Иногда эту задачу называют *классификацией с одним классом* (one-class classification).

Вероятностная интерпретация. Функции $g_t(x)$ можно интерпретировать как нечёткие множества, а при условии нормировки — и как вероятностные распределения. Если для байесовского классификатора расписать апостериорные вероятности классов, то функции компетентности приобретут смысл априорных вероятностей:

$$a(x) = \arg \max_{y \in Y} \lambda_y \mathbb{P}\{y|x\} = \arg \max_{y \in Y} \lambda_y \sum_{t=1}^T \underbrace{\mathbb{P}(t)}_{g_t(x)} \underbrace{\mathbb{P}\{y|t, x\}}_{b_{ty}(x)}.$$

Здесь предполагается, что каждый базовый алгоритм $b_t(x)$ возвращает вектор апостериорных вероятностей $(\mathbb{P}\{y|t, x\})_{y \in Y}$. Затем квазилинейная корректирующая операция переводит T таких векторов в один вектор апостериорных вероятностей $(\mathbb{P}\{y|x\})_{y \in Y}$, и к нему применяется решающее правило $\arg \max_y$.

Вероятностная интерпретация приводит к EM-алгоритму, который обрастает массой технических усложнений [16, 27, 44]. Мы рассмотрим более простые оптимизационные методы, основанные на применении выпуклых функций потерь.

Вид функций компетентности определяется спецификой предметной области. Например, может иметься априорная информация о том, что при больших и малых значениях некоторого признака $f(x)$ поведение целевой зависимости принципиально различно. Тогда имеет смысл использовать весовую функцию вида

$$g(x; \alpha, \beta) = \sigma(\alpha f(x) + \beta),$$

где $\alpha, \beta \in \mathbb{R}$ — свободные параметры, $\sigma(z) = (1 + e^{-z})^{-1}$ — сигмоидная функция, позволяющая описать плавный переход между двумя областями компетентности.

Если априорной информации нет, то в качестве «универсальных» областей компетентности берут области простой формы. Например, в пространстве $X = \mathbb{R}^n$ можно взять полуплоскости

$$g(x; \alpha, \beta) = \sigma(x^\top \alpha + \beta)$$

или сферические гауссианы

$$g(x; \alpha, \beta) = \exp(-\beta \|x - \alpha\|^2),$$

где $\alpha \in \mathbb{R}^n$, $\beta \in \mathbb{R}$ — свободные параметры, которые настраиваются по выборке аналогично тому, как настраиваются параметры базовых алгоритмов.

1.4.1 Выпуклые функции потерь

Преимущество смесей в том, что они позволяют разбить всё пространство X на области, в каждой из которых задача решается относительно просто, и затем «склеить» эти решения в единую композицию. Основной технический приём, позволяющий решить эти относительно простые подзадачи по-отдельности, независимо друг от друга, связан с использованием выпуклых функций потерь.

Рассмотрим функционал (??), характеризующий качество алгоритмических операторов при заданном векторе весов объектов $W^\ell = (w_1, \dots, w_\ell)$.

Гипотеза 1.1. Функция потерь $\tilde{L}(b, y)$ является выпуклой по b , то есть для любых $b_1, b_2 \in \mathbb{R}$, $y \in Y$ и неотрицательных g_1, g_2 , таких, что $g_1 + g_2 = 1$, выполняется

$$\tilde{L}(g_1 b_1 + g_2 b_2, y) \leq g_1 \tilde{L}(b_1, y) + g_2 \tilde{L}(b_2, y).$$

Условие выпуклости имеет прозрачную интерпретацию: потери растут не медленнее, чем величина отклонения от правильного ответа y . Во многих задачах требование выпуклости является естественным. В частности, ему удовлетворяет квадратичная функция $\tilde{L}(b, y) = (b - y)^2$, применяемая в регрессионных задачах. К сожалению, пороговая функция $\tilde{L}(b, y) = [by < 0]$, применяемая в задачах классификации при $Y = \{-1, 1\}$, не является выпуклой. Однако некоторые её непрерывные аппроксимации выпуклы, см. Рис. 1 на стр. 4:

$$\tilde{L}(b, y) = [by < 0] \leq \begin{cases} e^{-by} & \text{— экспоненциальная;} \\ \log_2(1 + e^{-by}) & \text{— логарифмическая;} \\ (1 - by)_+ & \text{— кусочно-линейная.} \end{cases}$$

Замена пороговой функции потерь на непрерывную является распространённой практикой. В частности, логарифмическая аппроксимация используется в логистической регрессии (см. ??), экспоненциальная — в алгоритме AdaBoost (см. §1.2), кусочно-линейная — в методе опорных векторов SVM (см. ??). Непрерывные функции потерь имеют важное достоинство — они штрафуют обучающие объекты за приближение к границе классов, что способствует увеличению зазора между классами и повышению надёжности классификации. Таким образом, требование выпуклости функции потерь не слишком обременительно и даже естественно как для задач регрессии, так и для классификации.

Итак, пусть выполнено условие нормировки (1.9) и функция потерь \tilde{L} выпукла. Вместо функционала $Q(a)$ будем минимизировать его верхнюю оценку, которая непосредственно вытекает из условия выпуклости:

$$Q(a) = \sum_{i=1}^{\ell} \tilde{L} \left(\sum_{t=1}^T \tilde{g}_t(x_i) b_t(x_i), y_i \right) \leq \sum_{t=1}^T \underbrace{\sum_{i=1}^{\ell} \tilde{g}_t(x_i) \tilde{L}(b_t(x_i), y_i)}_{Q(b_t; W^\ell)}. \quad (1.10)$$

Функционал $Q(a)$ распадается на сумму T функционалов, каждый из которых зависит только от b_t и \tilde{g}_t . Если зафиксировать функции компетентности \tilde{g}_t , то все базовые алгоритмы b_t можно настроить по-отдельности, независимо друг от друга. Это стандартная задача обучения базовых алгоритмов. Затем можно зафиксировать базовые алгоритмы и настроить их функции компетентности. Для настройки всей композиции организуется итерационный процесс, в котором эти два шага выполняются по очереди. Остаётся только понять, каким образом компоненты будут добавляться в смесь, как строить для них начальные приближения, и когда прекращать порождение новых компонент.

Далее рассматриваются два итерационных метода построения смесей, отличающиеся способом добавления компонент, — последовательный и иерархический.

1.4.2 Последовательный метод построения смеси

Алгоритм 1.7 реализует процесс последовательного построения смеси.

Первый базовый алгоритм $b_1(x)$ настраивается по всей выборке с помощью стандартного метода обучения. Затем выполняется процедура `LearnInitG`, которая настраивает функцию компетентности $g_1(x)$, пытаясь аппроксимировать область, на которой $b_1(x)$ чаще даёт верные ответы:

$$\text{LearnInitG}(b, X^\ell, Y^\ell) = \arg \min_g \sum_{i=1}^{\ell} (g(x_i) - z_i)^2,$$

где $z_i = \psi(L(b(x_i), y_i))$ — величина потери, приведённая к отрезку $[0, 1]$ с помощью некоторой монотонно невозрастающей функции ψ . Например, можно полагать

$z_i = [b(x_i) = y_i]$ — в задачах классификации;

$z_i = [|b(x_i) - y_i| < \varepsilon]$ при некотором $\varepsilon > 0$ — в задачах регрессии.

Алгоритм 1.7. Последовательное построение смеси алгоритмов

Параметры:

- ℓ_0 — допустимое число ошибок;
 δ — пороговое значение функции потерь;
-

- 1: начальное приближение первой компоненты смеси:
 $b_1 := \mu(X^\ell, Y^\ell);$
 $g_1 := \text{LearnInitG}(b_1, X^\ell, Y^\ell);$
 - 2: **для всех** $t = 1, \dots, T - 1$, пока не выполнен критерий останова:
 - 3: подмножество объектов, на которых композиция $a_t(x)$ допускает ошибки:
 $X^{(t)} := \{x_i \in X^\ell: L(a_t(x_i), y_i) > \delta\};$
 - 4: **если** $|X^{(t)}| \leq \ell_0$ **то**
 - 5: **вернуть** $(a_t);$
 - 6: начальное приближение t -й компоненты смеси:
 $b_{t+1} := \mu(X^{(t)}, Y^{(t)});$
 $g_{t+1} := \text{LearnInitG}(b_{t+1}, X^\ell, Y^\ell);$
 - 7: **для всех** $k = 1, 2, \dots$, пока не стабилизируется значение $Q(a_{t+1}; X^\ell, Y^\ell)$
 - 8: **для всех** $j = 1, \dots, t + 1$
 - 9: $w_i := g_j(x_i)$ для всех $i = 1, \dots, \ell;$
 - 10: перестройка j -й компоненты смеси на k -й итерации:
 $b_j := \mu(X^\ell, Y^\ell, W^\ell);$
 $g_j := \text{LearnG}_j(X^\ell, Y^\ell);$
 - 11: **вернуть** $(a_T);$
-

Далее в цикле по t происходит добавление новых компонент. При входе в цикл на шаге 3 предполагается, что смесь из t компонент уже построена:

$$a_t(x) = C(g_1(x)b_1(x) + \dots + g_t(x)b_t(x)), \quad x \in X.$$

Начальное приближение для $(t + 1)$ -й компоненты строится так же, как и на шаге 1, только теперь базовый алгоритм b_{t+1} обучается не на полном объеме данных, а на подвыборке $X^{(t)}$, составленной из «самых плохих» объектов, на которых композиция $a_t(x)$ выдала наименее точные ответы. Если таких объектов набирается менее ℓ_0 , построение смеси завершается.

Затем внутренний цикл итераций поочередно подстраивает все базовые алгоритмы и их функции компетентности (шаги 7–10). Согласно (1.10) обучение базового алгоритма b_j сводится к минимизации функционала $Q(b_j; W^\ell)$ с помощью стандартного метода обучения μ .

Покажем, что функции компетентности также можно настраивать по отдельности, применяя для этого стандартные методы обучения. Зафиксируем все базовые алгоритмы и все функции компетентности, кроме $g_j(x)$, и введём обозначения:

$$A_{ji} = \sum_{s=1, s \neq j}^{t+1} g_s(x_i)b_s(x_i); \quad G_{ji} = \sum_{s=1, s \neq j}^{t+1} g_s(x_i); \quad b_{ji} = b_j(x_i); \quad i = 1, \dots, \ell.$$

Тогда функционал качества Q будет зависеть только от функции g_j , и его можно минимизировать по g_j :

$$\text{LearnG}_j(X^\ell, Y^\ell) = \arg \min_g \sum_{i=1}^{\ell} \tilde{L}(A_{ji} + b_{ji}g_j(x_i), y_i).$$

Рассмотрим более подробно процедуру настройки LearnG_j , отдельно для задач классификации и регрессии.

Настройка функций компетентности в задачах классификации. Допустим для простоты, что $Y = \{-1, +1\}$, и базовые алгоритмы возвращают только два возможных значения -1 и $+1$. Для настройки функции компетентности $g_j(x)$ в смеси $a_{t+1}(x)$ будем минимизировать функционал Q с экспоненциальной функцией потерь:

$$\begin{aligned} Q(a_{t+1}) &= \sum_{i=1}^{\ell} \exp(-A_{ji}y_i - g_j(x_i)b_{ji}y_i) = \\ &= \sum_{i=1}^{\ell} \underbrace{\exp(-A_{ji}y_i)}_{\tilde{w}_i} \exp(-g_j(x_i) \underbrace{b_{ji}y_i}_{\tilde{y}_i}) = \sum_{i=1}^{\ell} \tilde{w}_i \exp(-g_j(x_i)\tilde{y}_i). \end{aligned}$$

Таким образом, настройка функции компетентности g_j сводится к задаче классификации, которая решается путём минимизации функционала качества Q с весами \tilde{w}_i и экспоненциальной функцией потерь. Формулы для весов объектов $\tilde{w}_i = \exp(-A_{ji}y_i)$ и модифицированных ответов $\tilde{y}_i = b_{ji}y_i$ позволяют дать этой задаче очевидную интерпретацию: функция компетентности обучается выделению объектов, на которых базовый алгоритм $b_j(x)$ не допускает ошибки. При этом бóльший вес получают «наиболее трудные» объекты, на которых смесь всех остальных компонент имеет большой отрицательный отступ.

Настройка функций компетентности в регрессионных задачах. Рассмотрим случай $Y = \mathbb{R}$. Зафиксируем все базовые алгоритмы и все функции компетентности кроме $g_j(x)$. Заметим, что в обозначениях, введённых выше, $A_{ji}/G_{ji} = a_t^{(j)}(x_i)$, где $a_t^{(j)}$ — композиция, полученная из $a_{t+1}(x)$ путём исключения j -й компоненты. Будем называть $a_t^{(j)}$ «смесью всех остальных компонент».

Для настройки функции компетентности $g_j(x)$ в смеси $a_{t+1}(x)$ будем минимизировать функционал среднеквадратичного отклонения:

$$\begin{aligned} Q(g_j) &= \sum_{i=1}^{\ell} (a_{t+1}(x_i) - y_i)^2 = \sum_{i=1}^{\ell} \left(\frac{A_{ji} + g_j(x_i)b_{ji}}{G_{ji} + g_j(x_i)} - y_i \right)^2 = \\ &= \sum_{i=1}^{\ell} \left(\frac{g_j(x_i)}{G_{ji} + g_j(x_i)} \left(b_{ji} - \frac{A_{ji}}{G_{ji}} \right) - \left(y_i - \frac{A_{ji}}{G_{ji}} \right) \right)^2 = \\ &= \sum_{i=1}^{\ell} \underbrace{\left(b_{ji} - \frac{A_{ji}}{G_{ji}} \right)^2}_{\tilde{w}_i} \left(\frac{g_j(x_i)}{G_{ji} + g_j(x_i)} - \underbrace{\frac{y_i - A_{ji}/G_{ji}}{b_{ji} - A_{ji}/G_{ji}}}_{\tilde{y}_i} \right)^2. \end{aligned}$$

Настройка функции компетентности $g_j(x)$ сводится к нелинейной регрессионной задаче со стандартным функционалом среднеквадратичной ошибки. Выполненные выше несложные преобразования позволили выделить веса объектов \tilde{w}_i и модифицированные ответы \tilde{y}_i . Их смысл достаточно очевиден. Веса \tilde{w}_i близки к нулю на тех объектах x_i , где базовый алгоритм $b_j(x_i)$ мало отличается от смеси всех остальных компонент. Вполне разумно, что функционал Q остаётся нечувствительным к значениям функции компетентности $g_j(x_i)$ на таких объектах. Модифицированные ответы \tilde{y}_i представляют отношение отклонений правильного ответа y_i и значения $b_j(x_i)$ от значения, которое даёт смесь всех остальных компонент $a_t^{(j)}(x_i)$. Если эти отклонения разного знака, то базовый алгоритм b_j даёт менее точный ответ, чем смесь всех остальных компонент. В этом случае значение функции компетентности $g_j(x_i)$ должно быть близко к нулю. Если же эти отклонения одного знака, то базовый алгоритм b_j пытается компенсировать ошибку, допущенную всеми остальными компонентами. В этом случае его компетентность должна быть близка к единице.

Недостатки последовательного метода.

- Низкая скорость обучения. При добавлении каждой новой компоненты смеси приходится перестраивать все предыдущие компоненты, что ведёт к существенным затратам времени. Этого недостатка лишены иерархические методы.

1.4.3 Иерархический метод построения смеси

Смесь двух алгоритмов. Рассмотрим сначала простой случай, когда композиция состоит из двух базовых алгоритмов, функции компетентности удовлетворяют требованию нормировки, решающее правило фиксировано:

$$a(x) = C(g_1(x)b_1(x) + g_2(x)b_2(x)), \quad g_1(x) + g_2(x) = 1, \quad x \in X.$$

Гипотеза 1.1 позволяет оценить сверху функционал $Q(a; W^\ell)$ суммой двух функционалов стандартного вида:

$$Q(a; W^\ell) \leq \tilde{Q}(a; W^\ell) = \underbrace{\sum_{i=1}^{\ell} g_1(x_i) w_i \tilde{L}(b_1(x_i), y_i)}_{Q(b_1; W_1^\ell)} + \underbrace{\sum_{i=1}^{\ell} g_2(x_i) w_i \tilde{L}(b_2(x_i), y_i)}_{Q(b_2; W_2^\ell)}.$$

Если функции компетентности $g_1(x)$ и $g_2(x)$ известны, то минимизация функционалов $Q(b_1; W_1^\ell)$ и $Q(b_2; W_2^\ell)$ может быть выполнена порознь. Для этого достаточно применить стандартный метод обучения μ :

$$\begin{aligned} b_1 &= \mu(X^\ell, Y^\ell, W_1^\ell), & W_1^\ell &= (w_i g_1(x_i))_{i=1}^{\ell}; \\ b_2 &= \mu(X^\ell, Y^\ell, W_2^\ell), & W_2^\ell &= (w_i g_2(x_i))_{i=1}^{\ell}. \end{aligned}$$

Верно и обратное: если известны базовые алгоритмы $b_1(x)$ и $b_2(x)$, то можно оценить функции $g_1(x)$ и $g_2(x)$. В силу условия нормировки $g_2(x) = 1 - g_1(x)$ достаточно найти функцию $g_1(x)$, доставляющую минимум функционалу

$$\tilde{Q}(g_1) = \sum_{i=1}^{\ell} g_1(x_i) \underbrace{w_i (\tilde{L}(b_1(x_i), y_i) - \tilde{L}(b_2(x_i), y_i))}_{d_i = \text{const}(g_1)}. \quad (1.11)$$

Алгоритм 1.8. М2Е — построение смеси двух базовых алгоритмов

Вход:

X^ℓ, Y^ℓ — обучающая выборка;

$W^\ell = (w_1, \dots, w_\ell)$ — исходный вектор весов объектов;

$b_0(x)$ — начальное приближение одного из базовых алгоритмов;

Выход:

алгоритмическая композиция: $a(x) = b_1(x)g_1(x) + b_2(x)g_2(x)$;

1: $g_1 := \arg \min_g \sum_{i=1}^{\ell} g(x_i) w_i \tilde{L}(b_0(x_i), y_i)$;

2: **повторять**

3: настройка двух базовых алгоритмов:

$b_1 := \mu(X^\ell, Y^\ell, W_1^\ell)$ с весами $w_{1i} = w_i g_1(x_i)$;

$b_2 := \mu(X^\ell, Y^\ell, W_2^\ell)$ с весами $w_{2i} = w_i (1 - g_1(x_i))$;

4: настройка функции компетентности:

$d_i := w_i (\tilde{L}(b_1(x_i), y_i) - \tilde{L}(b_2(x_i), y_i))$, для всех $i = 1, \dots, \ell$;

$g_1 := \arg \min_g \sum_{i=1}^{\ell} d_i g(x_i)$;

5: оценка качества композиции:

$Q := \sum_{i=1}^{\ell} w_i L(b_1(x_i)g_1(x_i) + b_2(x_i)g_2(x_i), y_i)$;

6: **пока** не стабилизируется значение Q ;

Если известен параметрический вид функции $g_1(x)$, то задача легко решается градиентными методами, аналогичными методам обучения нейронных сетей.

Итак, знание функций компетентности позволяет настроить оба базовых алгоритма, а знание базовых алгоритмов позволяет аппроксимировать функции компетентности. Остаётся только запустить итерационный процесс, в котором эти две задачи решаются поочерёдно. Итерации можно начинать с построения начального приближения $b_0(x)$ для одного из базовых алгоритмов. Алгоритм 1.8 описывает процедуру М2Е (mixture of 2 experts) подробно.

Иерархическая смесь алгоритмов. Метод М2Е легко приспособить для построения сколь угодно сложных смесей, состоящих из произвольного числа алгоритмов.

Идея заключается в следующем. Сначала строится смесь двух алгоритмов и анализируется качество работы каждого из них на своей области компетентности. Если качество алгоритма недостаточно, то он заменяется смесью двух новых алгоритмов, разбивающих его область $g(x)$ на две новые области с функциями компетентности $g(x)g_1(x)$ и $g(x)g_2(x)$, причём $g_1(x) + g_2(x) = 1$, чтобы по-прежнему выполнялось условие нормировки. Процесс дробления областей компетентности продолжается до тех пор, пока в смеси остаются алгоритмы неудовлетворительного качества. В результате получается бинарное дерево алгоритмов, называемое *иерархической смесью алгоритмов* (hierarchical mixture of experts, НМЕ). Алгоритм 1.9 более подробно описывает детали рекурсивной реализации этой идеи.

Способ построения НМЕ напоминает синтез бинарного решающего дерева (см. раздел ??). Главное отличие в том, что здесь функции компетентности производят нечёткое разделение пространства объектов на области. Ещё одно существенное отличие — в применяемом критерии ветвления.

Алгоритм 1.9. Рекурсивный алгоритм построения иерархической смеси

Вход:
 X^ℓ, Y^ℓ — обучающая выборка;

1: **ПРОЦЕДУРА** Разветвить (W^ℓ, b_0):
 $W^\ell = (w_1, \dots, w_\ell)$ — веса обучающих объектов;

 b_0 — алгоритм, заменяемый смесью $a(x) = b_1(x)g_1(x) + b_2(x)g_2(x)$;
2: $b_1, g_1, b_2, g_2 := M2E(W^\ell, b_0)$;3: $W_1^\ell = (w_i g_1(x_i))_{i=1}^\ell$;4: **если** КритерийВетвления (W_1^ℓ, b_1) **то**5: Разветвить (W_1^ℓ, b_1);6: $W_2^\ell = (w_i g_2(x_i))_{i=1}^\ell$;7: **если** КритерийВетвления (W_2^ℓ, b_2) **то**8: Разветвить (W_2^ℓ, b_2);

Основной алгоритм

9: $W^\ell := (1/\ell, \dots, 1/\ell)$;10: $b_0 := \mu(X^\ell, Y^\ell, W^\ell)$;11: Разветвить (W^ℓ, b_0);

Критерии ветвления. Базовый алгоритм расщепляется на два, когда качество его работы оказывается неудовлетворительным. Существуют различные способы оценить качество алгоритма, но мы рассмотрим только два.

- Относительная точность на обучающей выборке X^ℓ хуже заданного порога ε :

$$\text{КритерийВетвления}(W^\ell, b) := \left[Q(b; W^\ell) > \varepsilon \sum_{i=1}^\ell w_i \right].$$

Недостатком этого критерия является необходимость задания параметра ε .

- Точность на заранее выделенной контрольной выборке X^k хуже, чем у родительского алгоритма:

$$\text{КритерийВетвления}(W^\ell, b) := \left[Q(b; X^k, Y^k, W^k) > Q(b_0; X^k, Y^k, W^k) \right].$$

Данный критерий оценивает склонность рассматриваемой ветки иерархии к переобучению. Недостатком этого критерия является необходимость выделения части объектов в контрольную выборку.

§1.5 Нелинейные монотонные композиции

Монотонная коррекция является ещё одним обобщением линейной. Произвольная линейная корректирующая операция $F(b_1, \dots, b_T) = \alpha_1 b_1 + \dots + \alpha_T b_T$ с неотрицательными коэффициентами α_t является монотонным отображением из R^T в Y . Множество произвольных (не обязательно линейных) монотонных функций существенно шире, поэтому монотонные корректирующие операции обладают большими

возможностями для настройки. В то же время, монотонность является более естественным требованием к корректирующей операции, нежели линейность. Монотонность F как отображения $R^T \rightarrow Y$ означает, что одновременное увеличение значений $b_1(x), \dots, b_T(x)$ не должно приводить к уменьшению значения $F(b_1(x), \dots, b_T(x))$. Например, в алгоритмах классификации значение $b_t(x)$ часто имеет смысл вероятности того, что объект x принадлежит некоторому фиксированному классу. Было бы странно, если бы повышение этой оценки одновременно для всех базовых алгоритмов сопровождалось понижением значения $F(b_1(x), \dots, b_T(x))$, полученного в результате коррекции. Требование линейности связано с более жестким предположением, что веса базовых алгоритмов постоянны и не зависят от x — весьма сомнительная эвристика, если учесть, что базовые алгоритмы могут иметь различные области компетентности в пространстве объектов.

Опр. 1.3. Пусть R и Y — частично упорядоченные множества. Монотонные функции вида $F: R^T \rightarrow Y$ будем называть *монотонными корректирующими операциями*.

Заметим, что в данном случае решающие правила не используются, и корректирующие операции действуют непосредственно в Y . Соответственно, искомый алгоритм a имеет вид $a(x) = F(b_1(x), \dots, b_T(x))$.

Лемма о проведении монотонной функции через заданные точки.

Опр. 1.4. Пусть U, V — произвольные частично упорядоченные множества. Совокупность пар $(u_i, v_i)_{i=1}^{\ell}$ из $U \times V$ называется *монотонной*, если из $u_j \leq u_k$ следует $v_j \leq v_k$ для всех $j, k = 1, \dots, \ell$.

Лемма 1.5. Пусть U, V — произвольные частично упорядоченные множества. Монотонная функция $f: U \rightarrow V$ такая, что $f(u_i) = v_i$ для всех $i = 1, \dots, \ell$, существует тогда и только тогда, когда совокупность $(u_i, v_i)_{i=1}^{\ell}$ монотонна.

Доказательство.

Необходимость вытекает из определения монотонной функции: если f монотонна, то совокупность $(u_i, f(u_i))_{i=1}^{\ell}$ монотонна.

Докажем достаточность. Предполагая, что совокупность $(u_i, v_i)_{i=1}^{\ell}$ монотонна, построим функцию f в классе кусочно-постоянных функций. Определим для произвольного $u \in U$ множество индексов $I(u) = \{k: u_i \leq u\}$ и положим

$$f(u) = \begin{cases} \min_{i=1, \dots, \ell} v_i, & \text{если } I(u) = \emptyset; \\ \max_{i \in I(u)} v_i, & \text{если } I(u) \neq \emptyset. \end{cases}$$

Докажем, что функция f монотонна. Для произвольных u и u' из $u \leq u'$ следует $I(u) \subseteq I(u')$, значит $f(u) \leq f(u')$.

Докажем, что $f(u_i) = v_i$ для всех $i = 1, \dots, \ell$. Множество $I(u_i)$ не пусто, так как $i \in I(u_i)$. Для любого $k \in I(u_i)$ в силу монотонности $(u_i, v_i)_{i=1}^{\ell}$ из $u_k \leq u_i$ следует $v_k \leq v_i$. Но тогда $\max_{k \in I(u_i)} v_k$ достигается при $k = i$, откуда следует $f(u_i) = v_i$. ■

Доказательство леммы конструктивно, однако предложенный способ построения функции f мало пригоден для практических нужд. В регрессионных задачах

(когда $Y = \mathbb{R}$) желательно, чтобы функция f была гладкой или хотя бы непрерывной, здесь же f кусочно постоянна. В задачах классификации (когда $Y = \{0, 1\}$) желательно, чтобы разделяющая поверхность проходила как можно дальше от точек выборки, здесь же разделяющая полоса целиком относится к классу 0. Более практичные конструкции будут показаны ниже.

1.5.1 Оптимизация базовых алгоритмов

Обозначим через u_i вектор значений базовых алгоритмов на объекте x_i , через f_i — значение, выданное алгоритмом $a(x)$ на объекте x_i :

$$\begin{aligned} u_i &= (b_1(x_i), \dots, b_t(x_i)); \\ f_i &= a(x_i) = F(b_1(x_i), \dots, b_t(x_i)) = F(u_i); \quad i = 1, \dots, \ell. \end{aligned}$$

В новых обозначениях условие корректности алгоритма $a(x_i) = y_i$ примет вид

$$F(u_i) = y_i, \quad i = 1, \dots, \ell. \quad (1.12)$$

Опр. 1.5. Пусть V — произвольное упорядоченное множество. Пара индексов (j, k) называется дефектной для функции $b: X \rightarrow V$, если $y_j < y_k$ и $b(x_j) \geq b(x_k)$. Дефектом функции $b(x)$ называется множество всех её дефектных пар:

$$\mathbb{D}(b) = \{(j, k): y_j < y_k \wedge b(x_j) \geq b(x_k)\}.$$

Дефектная пара обладает тем свойством, что для любой монотонной функции F выполняется $F(b(x_j)) \geq F(b(x_k))$, следовательно, алгоритм $a(x) = F(b(x))$ допустит ошибку хотя бы на одном из двух объектов x_j, x_k , какой бы ни была F .

Опр. 1.6. Совокупным дефектом операторов b_1, \dots, b_t называется множество

$$\mathbb{D}_T(b_1, \dots, b_t) = \mathbb{D}(b_1) \cap \dots \cap \mathbb{D}(b_t) = \{(j, k): y_j < y_k \wedge u_j \geq u_k\}.$$

Будем также пользоваться сокращённым обозначением $\mathbb{D}_t = \mathbb{D}_t(b_1, \dots, b_t)$.

Для любой пары (j, k) из совокупного дефекта и любой монотонной функции F выполняется $F(u_j) \geq F(u_k)$, следовательно, алгоритм $a(x) = F(b_1(x), \dots, b_t(x))$ допустит ошибку хотя бы на одном из двух объектов x_j, x_k .

Теорема 1.6. Монотонная функция $F: R^t \rightarrow Y$, удовлетворяющая условию корректности 1.12 существует тогда и только тогда, когда совокупный дефект операторов b_1, \dots, b_t пуст. При этом дефект алгоритма $a = F(b_1, \dots, b_t)$ также пуст.

Доказательство.

Справедлива следующая цепочка равносильных утверждений:

- а) совокупный дефект пуст: $\mathbb{D}_t(b_1, \dots, b_t) = \emptyset$;
- б) для любых j, k не выполняется $(u_k \leq u_j) \wedge (y_j < y_k)$ (согласно Опр. 1.6);
- в) для любых j, k справедлива импликация $(u_k \leq u_j) \rightarrow (y_k \leq y_j)$;
- г) совокупность пар $(u_i, y_i)_{i=1}^{\ell}$ монотонна (согласно Опр. 1.4);
- д) существует монотонная функция $F: R^t \rightarrow Y$ такая, что $F(u_i) = y_i$ для всех $i = 1, \dots, \ell$ (согласно Лемме 1.5).

Из утверждения д) следует, что условия $y_j < y_k$ и $F(u_j) \geq F(u_k)$ не могут выполняться одновременно, значит, дефект $\mathbb{D}(F(b_1, \dots, b_t))$ также пуст. ■

Итак, чтобы гарантировать корректность композиции $a = F(b_1, \dots, b_t)$, достаточно построить операторы b_1, \dots, b_t , совокупный дефект которых пуст. Согласно определению 1.6 добавление каждого нового оператора в композицию может только уменьшать дефект, поскольку $\mathbb{D}_t \subseteq \mathbb{D}_{t-1}$. Если пара (j, k) удовлетворяет условиям

$$b_t(x_j) < b_t(x_k), \quad (j, k) \in \mathbb{D}_{t-1}, \quad (1.13)$$

то она уже не может принадлежать \mathbb{D}_t , так как добавление оператора b_t увеличивает размерность векторов u_k и u_j на единицу, при этом они становятся несравнимыми. Итак, выполнение условия (1.13) приводит к *устранению дефектной пары* (j, k) .

Теорема 1.7 (о сходимости). Пусть на первом шаге процесса (??) построен оператор b_1 , и семейство операторов B выбрано так, что для любой подвыборки X^{2m} длины $2m$, $m \geq 1$, найдётся оператор $b \in B$ и монотонная корректирующая операция F , удовлетворяющие системе ограничений

$$F(b(x_i)) = y_i, \quad x_i \in X^{2m}.$$

Тогда итерационный процесс (??-??) приводит к построению корректной композиции $a = F(b_1, \dots, b_T)$ за конечное число шагов $T \leq \lceil \mathbb{D}(b_1)/m \rceil + 1$.

Доказательство.

Рассмотрим t -й шаг, $t \geq 2$, итерационного процесса (??). Если $|\mathbb{D}_{t-1}| > m$, то выберем некоторое m -элементное подмножество совокупного дефекта $\Delta_{t-1} \subseteq \mathbb{D}_{t-1}$. Если $|\mathbb{D}_{t-1}| \leq m$, то положим $\Delta_{t-1} = \mathbb{D}_{t-1}$. Рассмотрим подмножество объектов выборки, образующих всевозможные дефектные пары из Δ_{t-1} :

$$U = \{x_i \in X^\ell \mid \exists k: (k, i) \in \Delta_{t-1} \text{ или } (i, k) \in \Delta_{t-1}\}.$$

Очевидно, мощность U не превышает $2m$. Согласно условию теоремы существует оператор $b_t \in B$ и монотонная корректирующая операция F , удовлетворяющие системе ограничений $F(b_t(x_i)) = y_i$ при всех $x_i \in U$. Но тогда для оператора b_t выполняется также система ограничений

$$b_t(x_j) < b_t(x_k), \quad (j, k) \in \Delta_{t-1}.$$

Докажем это от противного: пусть $b_t(x_k) \leq b_t(x_j)$, тогда $F(b_t(x_k)) \leq F(b_t(x_j))$, следовательно, $y_k \leq y_j$, что противоречит условию $y_j < y_k$, входящему в определение дефектной пары (j, k) .

Если выбирать операторы b_t , $t = 2, 3, \dots$ указанным способом, то мощность совокупного дефекта будет уменьшаться, как минимум, на m на каждом шаге итерационного процесса: $|\mathbb{D}_t| \leq |\mathbb{D}_{t-1}| - m$. Для полного устранения дефекта потребуется не более $\lceil \mathbb{D}(b_1)/m \rceil$ операторов, не считая b_1 . ■

Процесс последовательного построения базовых алгоритмов организуется по принципу *наискорейшего устранения дефекта*. Каждый следующий оператор оптимизируется таким образом, чтобы он устранял как можно больше дефектных пар из совокупного дефекта всех предыдущих операторов.

Проблема в том, что решение системы неравенств вида (1.13) не является стандартной задачей обучения, поскольку каждое неравенство в этой системе относится к паре объектов, а не к отдельному объекту. Тем не менее, задача может быть сведена к стандартной: оказывается, мощность совокупного дефекта можно оценить сверху обычным функционалом числа ошибок (??), если специальным образом задать веса обучающих объектов.

Следующая теорема показывает, что в случае классификации вес i -го объекта можно положить равным числу пар из совокупного дефекта \mathbb{D}_{t-1} , в которых участвует данный объект.

Теорема 1.8. Если $Y = \{0, 1\}$ и функция потерь имеет вид $\tilde{L}(b, y) = [b > 0] \neq y$, то справедлива оценка

$$|\mathbb{D}_t(b_1, \dots, b_t)| \leq \sum_{i=1}^{\ell} w_i \tilde{L}(b_i(x_i), y_i); \quad (1.14)$$

$$w_i = |D(i)|;$$

$$D(i) = \{x_k \in X^\ell \mid (k, i) \in \mathbb{D}_{t-1} \text{ или } (i, k) \in \mathbb{D}_{t-1}\}; \quad i = 1, \dots, \ell. \quad (1.15)$$

Доказательство.

Обозначим через $\beta_i = b_t(x_i)$ значение t -го оператора на i -м обучающем объекте, через $L_i = \tilde{L}(\beta_i, y_i) = [[\beta_i > 0] \neq y_i]$ — значение функции потерь, равное 1, если оператор b_t допускает ошибку на i -м объекте, 0 в противном случае.

Для любых действительных β_j, β_k справедливо неравенство

$$[\beta_j \geq \beta_k] \leq [\beta_j > 0] + [\beta_k \leq 0].$$

Если $y_j < y_k$, то из двухэлементности множества Y следует:

$$y_j = 0; \quad [\beta_j > 0] = [[\beta_j > 0] \neq 0] = [[\beta_j > 0] \neq y_j] = L_j;$$

$$y_k = 1; \quad [\beta_k \leq 0] = [[\beta_k > 0] \neq 1] = [[\beta_k > 0] \neq y_k] = L_k.$$

Используя представление $\mathbb{D}_t = \mathbb{D}_{t-1} \cap \mathbb{D}(b_t)$, распишем мощность совокупного дефекта в виде суммы по парам объектов и применим оценку $[\beta_j \geq \beta_k] \leq L_j + L_k$:

$$\begin{aligned} |\mathbb{D}_t| &= \sum_{(j,k) \in \mathbb{D}_{t-1}} [(j, k) \in \mathbb{D}(b_t)] = \sum_{(j,k) \in \mathbb{D}_{t-1}} [\beta_j \geq \beta_k] \leq \sum_{(j,k) \in \mathbb{D}_{t-1}} (L_j + L_k) = \\ &= \sum_{\substack{j=1 \\ y_j=0}}^{\ell} L_j \sum_{k=1}^{\ell} [(j, k) \in \mathbb{D}_{t-1}] + \sum_{\substack{k=1 \\ y_k=1}}^{\ell} L_k \sum_{j=1}^{\ell} [(j, k) \in \mathbb{D}_{t-1}] = \\ &= \sum_{i=1}^{\ell} L_i \sum_{k=1}^{\ell} [(k, i) \in \mathbb{D}_{t-1} \text{ или } (i, k) \in \mathbb{D}_{t-1}] = \sum_{i=1}^{\ell} L_i |D(i)|. \end{aligned}$$

■

В случае регрессии справедлива аналогичная теорема [2].

Теорема 1.9. Если $Y = \mathbb{R}$ и функция потерь имеет вид $\tilde{L}(b, y) = (b - y)^2$, то справедлива та же оценка (1.14), только веса определяются по-другому: $w_i = h_i^{-2} |D(i)|$, где $h_i = \frac{1}{2} \min_{k \in D(i)} |y_i - y_k|$, $D(i)$ определяется согласно (1.15).

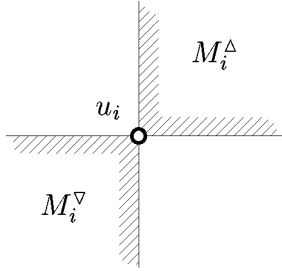


Рис. 2. Верхний (M_i^Δ) и нижний (M_i^∇) конусы вектора u_i .

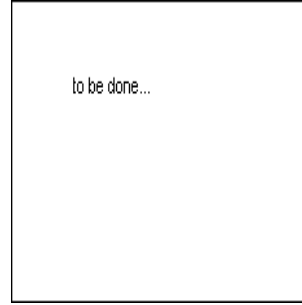


Рис. 3. Расстояние от вектора u до ближайшего вектора верхнего конуса.

1.5.2 Монотонная интерполяция и аппроксимация

Если совокупный дефект операторов b_1, \dots, b_t пуст, то существует монотонная функция F такая, что $F(u_i) = y_i$ для всех $i = 1, \dots, \ell$. Задача построения такой функции называется задачей *монотонной интерполяции*.

Если же дефект не пуст, то совокупность пар $(u_i, y_i)_{i=1}^\ell$ не является монотонной, и система ограничений $F(u_i) = y_i$ несовместна. Задача построения монотонной функции F , приближенно удовлетворяющей условиям $F(u_i) = y_i$, называется задачей *монотонной аппроксимации*. Она решается в два этапа.

На первом этапе находятся значения $f_i \in Y$, $i = 1, \dots, \ell$, наименее отклоняющиеся от y_i , для которых последовательность $(u_i, f_i)_{i=1}^\ell$ монотонна:

$$\begin{cases} \sum_{i=1}^{\ell} (f_i - y_i)^2 \rightarrow \min; \\ f_j \leq f_k, \text{ для всех } (i, j): u_j \leq u_k. \end{cases}$$

Это задача квадратичного программирования с линейными ограничениями-неравенствами. Для её решения могут быть применены стандартные методы, например, метод активных ограничений.

На втором этапе решается задача монотонной интерполяции — строится монотонная функция, удовлетворяющая условиям $F(u_i) = f_i$ для всех $i = 1, \dots, \ell$.

Таким образом, в обоих случаях приходится решать задачу монотонной интерполяции. Рассмотрим её более подробно.

Задача монотонной интерполяции. Задана выборка — монотонная совокупность пар $(u_i, f_i)_{i=1}^\ell$, где $u_i \in \mathbb{R}^t$, $f_i \in \mathbb{R}$. Требуется построить монотонную функцию $F: \mathbb{R}^t \rightarrow \mathbb{R}$ проходящую через все точки выборки: $F(u_i) = f_i$, $i = 1, \dots, \ell$. Предполагается, что среди значений f_i есть различные (задача не вырождена).

Определим *верхний конус* M_i^Δ и *нижний конус* M_i^∇ для каждого из векторов u_i , как показано на Рис. 2:

$$\begin{aligned} M_i^\Delta &= \{u \in \mathbb{R}^t \mid u_i \leq u\}; \\ M_i^\nabla &= \{u \in \mathbb{R}^t \mid u \leq u_i\}. \end{aligned}$$

Пусть $\mu: \mathbb{R}_+^t \rightarrow \mathbb{R}_+$ — некоторая неубывающая функция, принимающая нулевое значение только в точке $(0, \dots, 0)$. Например, можно взять одну из функций

$$\mu(z_1, \dots, z_t) = \begin{cases} \max(z_1, \dots, z_t); \\ z_1 + \dots + z_t; \\ \sqrt{z_1^2 + \dots + z_t^2}; \end{cases}$$

В то же время, функция $\min(z_1, \dots, z_t)$ не подходит, так как она принимает нулевое значение не только в нуле.

Определим функции расстояния от произвольного вектора $u = (u^1, \dots, u^t)$ до верхнего и нижнего конусов векторов $u_i = (u_i^1, \dots, u_i^t)$:

$$\begin{aligned} r_i^\Delta(u) &= \mu((u_i^1 - u^1)_+, \dots, (u_i^t - u^t)_+); \\ r_i^\nabla(u) &= \mu((u^1 - u_i^1)_+, \dots, (u^t - u_i^t)_+); \end{aligned}$$

где $(z)_+ = [z > 0]z$. Расстояние $r_i^\Delta(u)$ монотонно не возрастает по u и обращается в нуль на верхнем конусе M_i^Δ . Расстояние $r_i^\nabla(u)$ монотонно не убывает по u и обращается в нуль на нижнем конусе M_i^∇ . Если функция μ непрерывна, то функции r_i^Δ и r_i^∇ также непрерывны.

Для любого γ из полуинтервала $[\min_i f_i, \max_i f_i)$ определим расстояния от произвольного вектора $u = (u^1, \dots, u^t)$ до ближайшего верхнего и до ближайшего нижнего конусов:

$$\begin{aligned} h^\Delta(u, \gamma) &= \min_{i: f_i > \gamma} r_i^\Delta(u); \\ h^\nabla(u, \gamma) &= \min_{i: f_i \leq \gamma} r_i^\nabla(u). \end{aligned}$$

Интерполяция бинарной монотонной функции. В задаче классификации на два класса $Y = \{0, 1\}$ функции h^Δ, h^∇ сразу позволяют построить искомую монотонную корректирующую операцию $F(u)$. Вектор u будет относиться к классу 1, если расстояние до ближайшего верхнего конуса класса 1 меньше, чем расстояние до ближайшего нижнего конуса класса 0.

Теорема 1.10. *Для задачи классификации, $Y = \{0, 1\}$, функция*

$$F(u) = [h^\Delta(u, \gamma) \leq h^\nabla(u, \gamma)]$$

при любом $\gamma \in [0, 1)$ определена на всём пространстве \mathbb{R}^t , монотонно не убывает, принимает значения из Y и удовлетворяет условиям $F(u_i) = f_i$, для всех $i = 1, \dots, \ell$.

Доказательство можно найти в [2].

Рис. 4 поясняет геометрический смысл функции $F(u)$: она представляет собой монотонную «ступеньку», равную единице на объединении верхних конусов класса 1, и нулю на объединении нижних конусов класса 0. Разделяющая поверхность проходит посередине между этими областями. График на Рис. 4 построен для двумерного случая $t = 2$, при $\mu(z^1, \dots, z_t) = (z_1^2 + \dots + z_t^2)^{-1/2}$.

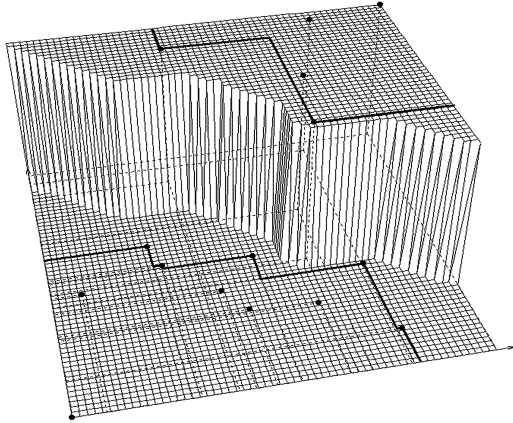


Рис. 4. Дискретная монотонная ступенька $F(u)$ для задачи классификации.

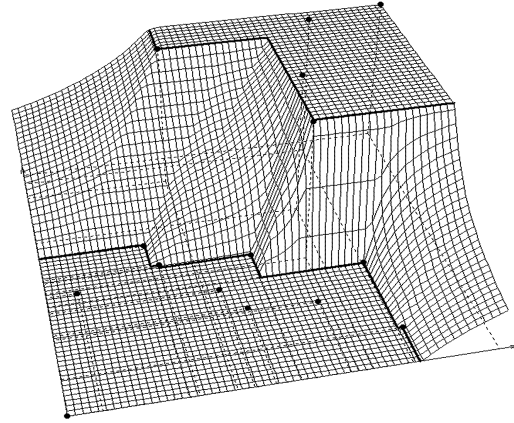


Рис. 5. Непрерывная монотонная ступенька $\Phi(u, \gamma)$ для задачи восстановления регрессии.

Интерполяция непрерывной монотонной функции. В случае восстановления регрессии задача осложняется тем, что алгоритм $a(x)$, а значит и корректирующая операция $F(u)$, должны быть достаточно гладкими или хотя бы непрерывными. Это естественное требование возникает в большинстве практических задач.

Введём вспомогательную функцию

$$\Phi(u, \gamma) = \frac{h^\nabla(u, \gamma)}{h^\nabla(u, \gamma) + h^\Delta(u, \gamma)}, \quad u \in \mathbb{R}^t, \quad \gamma \in \mathbb{R}.$$

Нетрудно показать, что эта функция непрерывна, монотонно не убывает, и на векторах u_i , $i = 1, \dots, \ell$ принимает значения либо 0, либо 1:

$$\Phi(u_i, \gamma) = [f_i > \gamma].$$

Функция $\Phi(u, \gamma)$ представляет собой непрерывный аналог дискретной монотонной «ступеньки», рассмотренный выше, что хорошо видно из сравнения Рис. 4 и Рис. 5.

Идея построения непрерывной монотонно неубывающей функции, проходящей через заданные точки, заключается в том, чтобы «поставить друг на друга» $\ell - 1$ непрерывных ступенек вида $\Phi(u, \gamma)$.

Теорема 1.11. Пусть $Y = \mathbb{R}$ и точки выборки пронумерованы по возрастанию значений: $f_1 \leq f_2 \leq \dots \leq f_\ell$. Тогда функция

$$F(u) = f_1 + \sum_{i=1}^{\ell-1} (f_{i+1} - f_i) \Phi(u, f_i)$$

определена на всём пространстве \mathbb{R}^t , монотонно не убывает и удовлетворяет условиям $F(u_i) = f_i$ для всех $i = 1, \dots, \ell$. Если функция $\mu(u)$ непрерывна, то функция $F(u)$ также непрерывна.

Доказательство можно найти в [2].

Заметим, что монотонная функция $F(u)$ в общем случае не является гладкой, её производная может претерпевать разрывы в местах «склейки» ступенек.

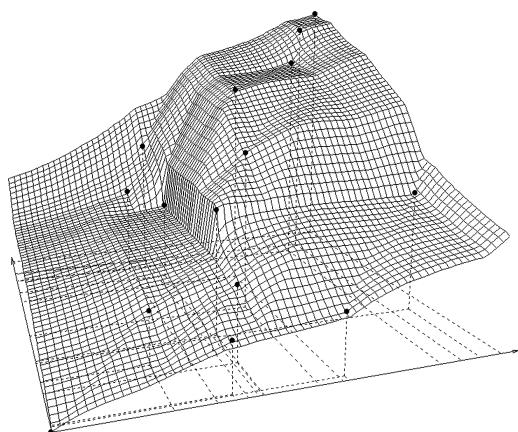


Рис. 6. Монотонная корректирующая операция $F(u)$ в задаче восстановления регрессии.

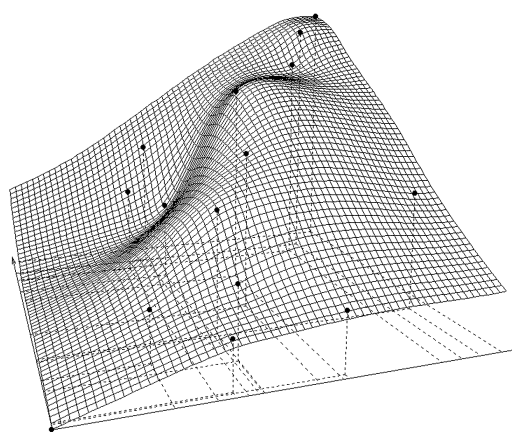


Рис. 7. Классический (немонотонный) гладкий сплайн, построенный по монотонной выборке.

На Рис. 6 показан пример функции $F(u)$, построенной по двумерным модельным данным. Для сравнения на Рис. 7 показан классический гладкий интерполяционный сплайн, построенный по той же монотонной выборке. Заметим, что он не является монотонной функцией.

§1.6 Краткий обзор литературы

Общая теория алгоритмических композиций над произвольными моделями алгоритмов и произвольными семействами корректирующих операций была впервые предложена Журавлёвым в *алгебраическом подходе* к проблеме распознавания [4, 5]. До этого были известны только наиболее простые методы комбинирования алгоритмов, такие как взвешенное голосование и комитетные системы [8]. При этом, как правило, рассматривались вполне конкретные модели базовых алгоритмов, а корректирующие операции не обладали возможностью настройки. Фактически, эти работы оставались на уровне обычных эвристических приёмов и не носили характера систематической научной теории. Большое разнообразие методов комбинирования алгоритмов также появилось существенно позже.

Методы, основанные на идее *областей компетентности*, были предложены Растригиным [10] и позже Джорданом и Джакобсом и др. в [16].

В 1988 в работах Кернса и Валианта [29] была поставлена следующая проблема. Будем под *слабой обучаемостью* (weak learnability) понимать возможность построения (за полиномиальное время) алгоритма, вероятность ошибок которого лишь немного меньше 50%; под *сильной обучаемостью* (strong learnability) будем понимать возможность построения (опять-таки, за полиномиальное время) алгоритма, вероятность ошибок которого сколь угодно мало отличается от нуля при $\ell \rightarrow \infty$. Была выдвинута гипотеза, что понятия сильной и слабой обучаемости эквивалентны, то есть что любую слабую модель можно *усилить* (boost). Эта гипотеза была теоретически подтверждена в работе Шапира [38], который предложил первый алгоритм *бустинга* (boosting). Годом позже Фройнд предложил свой алгоритм [21]. Оба алгоритма основывались на взвешенном голосовании, но были неудобны для практического применения. Лишь пятью годами позже им удалось разработать алгоритм AdaBoost, получивший впоследствии широкую известность благодаря простоте и высокой эф-

фективности [22]. Аналогичные методы, но с несколько иной стратегией оптимизации базовых операторов, были разработаны Уолпертом [45] и Брейманом [19]. Следующим обобщением стало введение нелинейных весовых функций в [41]. Затем были предложены нелинейные монотонные корректирующие операции [1, 2, 15]

По отношению к алгебраическому подходу перечисленные методы являются частными случаями, отличаясь, главным образом, видом корректирующей операции. Алгебраический подход позволяет «порождать» такого рода методы с общих теоретических позиций. Наиболее абстрактным разделом алгебраического подхода является *теория универсальных и локальных ограничений* Рудакова. Она даёт критерии полноты, позволяющие строить семейства базовых алгоритмов и корректирующих операций минимальной достаточной сложности [13, 11, 12, 14].

В современных обзорах по методам распознавания [26] и алгоритмическим композициям [40] подчеркивается, что построение композиций, в которых различные алгоритмы компенсируют недостатки друг друга, является одним из наиболее перспективных направлений машинного обучения.

Резюме

1. *Основные свойства композиций*, отличающие их от обычных алгоритмов обучения по прецедентам.
 - Алгоритмическая композиция объединяет базовые алгоритмы, способные самостоятельно решать ту же исходную задачу.
 - Композиция не знает внутреннего устройства базовых алгоритмов. Для неё это «чёрные ящики», имеющие только две функции: обучения по заданной выборке и вычисления ответа для заданного объекта. Это свойство удобно с технологической точки зрения, так как для построения базовых алгоритмов можно задействовать уже имеющиеся стандартные методы обучения.
 - Композиция позволяет получать высокое качество обучения, недостижимое для отдельных базовых алгоритмов.
2. *Два основных принципа* построения алгоритмических композиций.
 - *Специализация.* Пространство объектов делится на области, в каждой из которых строится свой алгоритм, специализирующийся на объектах только этой области. Исходная задача разбивается на более простые подзадачи по принципу «разделяй и властвуй». К таким методам относятся *комитеты старшинства* [9] и *смеси алгоритмов* [27].
 - *Усреднение.* В этом случае корректирующая операция не получает информации о том, в какой области пространства находится объект, и работает только с ответами, выданными базовыми алгоритмами. Если базовые алгоритмы достаточно различны, то их погрешности компенсируются в результате усреднения. Причём усреднение следует понимать в обобщённом смысле, это не обязательно среднее арифметическое, и даже не обязательно линейная операция. На идее усреднения основаны *комитеты большинства*, *бустинг* [23], *бэггинг* [19], *монотонная коррекция* [2].
3. *Основные стратегии* построения алгоритмических композиций.

- *Последовательная оптимизация.* Базовые алгоритмы строятся по очереди, и каждый следующий старается компенсировать недостатки предыдущих. Это жадная стратегия. Она не гарантирует построения наилучшей композиции, но на практике оказывается достаточно удобной. К таким методам относятся простейшие методы построения комитетов большинства и старшинства, бустинг [37], монотонная коррекция [1].
- *Параллельная оптимизация.* Базовые алгоритмы настраиваются независимо друг от друга. Чтобы они не получались слишком похожими, настройка производится по различным частям обучающей выборки, либо по различным частям признакового описания, либо при различных начальных приближениях. Типичными представителями этого подхода являются бэггинг [20], метод случайных подпространств (random subspace method) [39] и генетические алгоритмы [3]. В отличие от последовательной оптимизации, эти алгоритмы допускают эффективную реализацию на параллельных вычислительных устройствах.
- *Глобальная оптимизация* одновременно всех базовых алгоритмов является тяжёлой многоэкстремальной задачей. К тому же, она требует знания внутреннего устройства алгоритмов, что затрудняет применение стандартных методов обучения. На практике применяются разнообразные усовершенствования параллельных и последовательных методов, которые не являются «глобальными» в полном смысле слова. Например, при построении смесей экспертов базовые алгоритмы и их функции компетентности перебираются поочерёдно и многократно с помощью итерационного процесса, напоминающего EM-алгоритм [27]. В другом методе с помощью специального генетического алгоритма оптимизируются подмножества объектов и признаков, на которых настраиваются базовые алгоритмы [3]. Фактически, это развитие идей бэггинга и метода случайных подпространств.
- *Алгебраический подход*, описанный в теоретических работах Журавлёва и его учеников [6, 7], позволяет строить корректные алгоритмические композиции чисто алгебраическими методами, вообще не прибегая к оптимизации. Этот подход крайне продуктивен при исследовании вопросов полноты моделей алгоритмов вида (1.1). Однако он плохо приспособлен для практического построения алгоритмов, так как не позволяет управлять сложностью композиции и склонен к переобучению. Более практичные схемы, разработанные в рамках алгебраического подхода, используют упомянутую выше стратегию последовательной оптимизации [1, 2].

Упражнения

Упр. 1.1. Вывести формулу для коэффициентов взвешенного голосования при «наивном байесовском» предположении, что базовые алгоритмы являются независимыми случайными величинами, см. 1.4.

Упр. 1.2. Доказать, что при случайном выборе с возвращениями ℓ раз из ℓ объектов отобранными окажутся $1 - e^{-1}$ объектов при $\ell \rightarrow \infty$.