Лекции по алгоритмическим композициям

К. В. Воронцов

21 декабря 2007 г.

Содержание

1 Композиции алгоритмов

При решении сложных задач классификации, регрессии и прогнозирования часто возникает следующая ситуация. Одна за другой предпринимаются попытки построить алгоритм, восстанавливающий искомую зависимость, однако качество всех построенных алгоритмов оставляет желать лучшего. В таких случаях имеет смысл объединить несколько алгоритмов в композицию, в надежде на то, что погрешности этих алгоритмов взаимно скомпенсируются.

Масса вопросов возникает при построении композиций. При каких условиях качество композиции окажется лучше, чем у отдельных базовых алгоритмов? Как настраивать базовые алгоритмы, учитывая, что они будут работать в составе композиции? Возможно ли приспособить для их настройки стандартные методы обучения? Как обойтись минимальным числом базовых алгоритмов? Эти и другие вопросы разбираются в данной главе.

Напомним основные обозначения.

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

§1.1 Понятие алгоритмической композиции

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 $b\colon X \to R$ называется *алгоритмическим оператором*, функция $C\colon R \to Y$ — *решающим правилом*.

Многие алгоритмы классификации имеют именно такую структуру: сначала вычисляются оценки принадлежности объекта классам, затем решающее правило переводит эти оценки в номер класса. Значение оценки, как правило, характеризует степень уверенности классификации. В одних алгоритмах это вероятность принадлежности объекта заданному классу, в других — расстояние от объекта до разделяющей поверхности. Возможны и другие интерпретации оценок.

Опр. 1.1. *Алгоритмической композицией, составленной из алгоритмических операторов $b_t\colon X \to R$, $t = 1, \ldots, T$, корректирующей операции $F\colon R^T \to R$ и решающего правила $C\colon R \to Y$ называется алгоритм $a\colon X \to Y$ вида*

$$a(x) = C\big(F(b_1(x), \ldots, b_T(x))\big), \quad x \in X. \tag{1.1}$$

Базовыми алгоритмами будем называть функции $a_t(x) = C(b_t(x))$, а при фиксированном решающем правиле C — и сами операторы $b_t(x)$.

Заметим, что суперпозиции вида $F(b_1, \ldots, b_T)$ являются отображениями из X в R, то есть, опять-таки, алгоритмическими операторами.

1.1.1 Пространства оценок и решающие правила

Пространство оценок R вводится для того, чтобы расширить множество допустимых корректирующих операций. Можно было бы определить корректирующую операцию и как отображение $F\colon Y^T \to Y$, то есть комбинировать непосредственно ответы базовых алгоритмов. Однако в задачах классификации, когда множество Y конечно, число «разумных» корректирующих операций такого вида настолько мало, что возможность оптимизации F под конкретную задачу практически исключается [6]. Положение меняется, если комбинировать ответы алгоритмических операторов. Тогда операция F получает на входе более детальную информацию, не огрублённую решающим правилом, что способствует повышению качества классификации.

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

$$Y = \{0, 1\}, \quad C(b) = [b > 0], \quad a(x) = [b(x) > 0].$$

Иногда удобнее брать

$$Y = \{-1, +1\}, \quad C(b) = \operatorname{sign}(b), \quad a(x) = \operatorname{sign} b(x).$$

Пример 1.2. Решающие правила C могут иметь настраиваемые параметры. Например, в задачах классификации часто используется *пороговое решающее правило*

$$C(b) = [b > \theta] \quad \text{или} \quad C(b) = \operatorname{sign}(b - \theta), \quad \theta \in \mathbb{R}.$$

Как правило, сначала строится оператор b при $\theta = 0$, затем отдельно подбирается оптимальное значение θ.

Пример 1.3. В задачах классификации на M классов, $Y = \{1, \ldots, M\}$, пространством оценок является множество M-мерных действительных векторов, $R = \mathbb{R}^M$. Алгоритмический оператор $b(x)$ выдаёт вектор оценок принадлежности объекта x каждому из классов, $b(x) = \big(b^{(1)}(x), \ldots, b^{(M)}(x)\big)$. Решающее правило C относит объект к тому классу, для которого оценка максимальна:

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

В частности, именно такой вид имеют байесовские алгоритмы классификации из главы **??** и метрические алгоритмы из **??**. Далее будем часто ограничиваться задачами классификации на два класса, так как обобщение на произвольное число классов строится несложно — путём перехода от скалярных оценок к векторным.

Пример 1.4. В работах Ю. И. Журавлёва рассматривались двухпараметрические решающие правила для задач классификации с M классами. Как и в примере 1.3, объект x относится к тому классу y, для которого оценка $b^{(y)}$ максимальна. Но если выполняется хотя бы одно из двух условий

$$b^{(y)} < \delta_1 + \max_{z \in Y \setminus y} b^{(z)}; \qquad b^{(y)} < \delta_2 \sum_{z \in Y \setminus y} b^{(z)};$$

то алгоритм отказывается от классификации объекта x. Отказ означает, что уверенная классификация данного объекта невозможна. Необходимо либо привлекать дополнительную информацию об объекте, либо обращаться за помощью к эксперту. Если цена ошибки сильно выше цены отказа, то оптимизация границы зоны отказов с помощью параметров δ_1 и δ_2 может существенно снизить средние потери.

Пример 1.5. В задачах регрессии множество Y уже достаточно богато, обычно $Y = \mathbb{R}$, поэтому использовать решающее правило нет особого смысла. Общая формула (1.1) остаётся в силе, если положить $R = Y$ и $C(b) \equiv b$.

1.1.2 Корректирующие операции

Пока мы только перечислим основные виды корректирующих операций. Методы их настройки будут рассмотрены в следующих параграфах.

Простое голосование. Простейшим примером корректирующей операции является среднее арифметическое:

$$b(x) = F\big(b_1(x), \ldots, b_T(x)\big) = \frac{1}{T}\sum_{t=1}^{T} b_t(x), \quad x \in X. \tag{1.2}$$

В задачах классификации такая корректирующая операция называется *простым голосованием* (simple voting) или *голосованием по большинству* (majority voting). Если $Y = \{-1, +1\}$ и операторы b_t принимают только два возможных значения -1 и $+1$, то объект x будет относиться к тому классу, к которому его относит большинство базовых алгоритмов.

Взвешенное голосование. Корректирующая операция F может иметь свободные параметры, которые необходимо настраивать по обучающей выборке наряду с параметрами базовых алгоритмов. Примером является взвешенное среднее, называемое также *линейной комбинацией* базовых алгоритмов:

$$b(x) = F\big(b_1(x), \ldots, b_T(x)\big) = \sum_{t=1}^{T} \alpha_t b_t(x), \quad x \in X, \quad \alpha_t \in \mathbb{R}. \tag{1.3}$$

Если параметры α_t неотрицательны и нормированы, $\sum_{t=1}^{T} \alpha_t = 1$, то композицию (1.3) называют *выпуклой комбинацией* базовых алгоритмов. В выпуклых комбинациях гарантируется, что значение $b(x)$ не выйдет за границы отрезка $\big[\min\limits_t b_t(x), \max\limits_t b_t(x)\big]$ для любого $x \in X$. При этом веса α_t выражают степень доверия к соответствующим базовым алгоритмам.

В задачах классификации корректирующая операция (1.3) называется *взвешенным голосованием* (weighted voting).

Голосование по старшинству. Положим $R = \{0, 1\}$. *Голосованием по старшинству* (seniority voting) называется корректирующая операция $F\colon R^T \to Y$, вычисляемая согласно Алгоритму 1.1. Если $b_1(x) = 1$, то объект x относится к классу c_1, иначе право голоса передаётся следующему по старшинству алгоритму b_2. Если $b_2(x) = 1$,

Алгоритм 1.1. Классификация объекта $x \in X$ комитетом старшинства

1: **для всех** $t = 1, \ldots, T$
2: **если** $b_t(x) = 1$ **то вернуть** c_t;
3: **вернуть** c_0.

то объект x относится к классу c_2, и так далее. Передача права голоса продолжается до тех пор, пока один из базовых алгоритмов не примет решения. Если же все T алгоритмов возвращают нулевое значение, выдаётся ответ c_0, означающий отказ от классификации данного объекта. Решающее правило C в данном случае не используется, формально можно положить $C(y) \equiv y$.

Данный способ построения композиций вводился многократно под разными названиями: комитет с логикой старшинства [32], решающий список правил [36], машина покрывающих множеств [31]. Последнее название связано с тем, что при $b_t(x) = 1$ говорят «правило b_t *покрывает* (или *выделяет*) объект x».

Смесь алгоритмов. Дальнейшее обобщение линейных комбинаций связано с предположением, что веса базовых алгоритмов не постоянны и зависят от положения объекта x в пространстве X.

Квазилинейная комбинация базовых алгоритмов $b_t(x)$ с *функциями компетентности* $g_t(x)$, $t = 1, \ldots, T$, есть отображение $F\colon \mathbb{R}^{2T} \to \mathbb{R}$ следующего вида:

$$b(x) = F\big(b_1(x), g_1(x), \ldots, b_T(x), g_T(x)\big) = \sum_{t=1}^{T} g_t(x) b_t(x). \tag{1.4}$$

Обычно к функциям компетентности предъявляются требования неотрицательности и нормированности: для любого $x \in X$

$$g_t(x) \geqslant 0;$$
$$g_1(x) + \cdots + g_T(x) = 1;$$

В этом случае чем больше значение $g_t(x)$, тем с бóльшим весом учитывается ответ алгоритма $b_t(x)$ на объекте x.

Если функция $g_t(x)$ принимает только два значения $\{0, 1\}$, то множество всех $x \in X$, для которых $g_t(x) = 1$, называется *областью компетентности* базового алгоритма $b_t(x)$. В общем случае функция $g_t(x)$ описывает область компетентности как нечёткое множество, и значение $g_t(x) \in [0, 1]$ рассматривается как степень принадлежности объекта x области компетентности t-го базового алгоритма.

Понятие области компетентности было введено Растригиным в [10]. В англоязычной литературе композиции вида (1.11) принято называть *смесями 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*.

Монотонная корректирующая операция. Пусть R и Y — частично упорядоченные множества и корректирующая операция $F\colon R^T \to Y$ является монотонной функцией. Согласно определению монотонности это означает, что для любых $(b_1,\ldots,b_T)$ и $(b'_1,\ldots,b'_T)$ из R^T если $b_t \leqslant b'_t$, $t=1,\ldots,T$, то $F(b_1,\ldots,b_T) \leqslant F(b'_1,\ldots,b'_T)$. Решающие правила в данном случае не используются. Соответственно, искомый алгоритм имеет вид $a(x) = F(b_1(x),\ldots,b_T(x))$.

Линейная выпуклая корректирующая операция является частным случаем монотонной. Поэтому монотонность можно рассматривать как ещё одно обобщение взвешенного голосования. Требование монотонности представляется даже более естественным, чем линейность. Монотонность F означает, что одновременное увеличение оценок $b_1(x),\ldots,b_T(x)$ не может приводить к уменьшению значения $F(b_1(x),\ldots,b_T(x))$. В то же время, линейность связана с более жёстким ограничением: вклады базовых алгоритмов в композицию должны быть постоянны и не меняться на всём пространстве X.

1.1.3 Анализ разброса и смещения

Если X — вероятностное пространство, то b_t можно рассматривать как случайные величины. Если ошибки операторов b_t независимы, то при простом голосовании квадратичная ошибка случайной величины $b(x) = b_1(x) + \cdots + b_T(x)$ уменьшается со скоростью $O(T^{1/2})$ при $T \to \infty$. Это простейшее обоснование корректирующих операций, основанных на принципе голосования.

На практике предполагать независимость ошибок, конечно же, нельзя, поскольку базовые алгоритмы настраиваются на решение одной и той же задачи. Если требование независимости нарушается, ошибки могут уменьшаться существенно медленнее или даже увеличиваться с ростом T.

Далее мы будем рассматривать различные приёмы, повышающие *различность* (diversity) базовых алгоритмов. С одной стороны, это ведёт к ухудшению их качества по-отдельности, но с другой стороны, делает их более независимыми и может способствовать улучшению качества композиции в целом. Основная трудность при практическом построении композиций — выдержать компромисс между различностью и качеством базовых алгоритмов.

1.1.4 Функционалы качества

Введём функционал качества алгоритма $a\colon X \to Y$ на выборке X^ℓ с заданными на ней ответами Y^ℓ и весами объектов $W^\ell = (w_1,\ldots,w_\ell)$:

$$Q(a; X^\ell, Y^\ell, W^\ell) = \sum_{i=1}^{\ell} w_i L\big(a(x_i), y_i\big), \tag{1.5}$$

где $L\colon Y \times Y \to \mathbb{R}_+$ — функция потерь. Примеры различных функций потерь для задач классификации и регрессии приведены в параграфе **??**.

Пусть решающее правило C фиксировано. Тогда можно ввести функцию потерь в пространстве оценок $\widetilde{L}(b, y^*) = L(C(b), y^*)$ и оценивать качество не только самих алгоритмов, но и алгоритмических операторов:

$$Q(b; X^\ell, Y^\ell, W^\ell) = \sum_{i=1}^{\ell} w_i \widetilde{L}\big(b(x_i), y_i\big). \tag{1.6}$$

Будем полагать, что в нашем распоряжении имеется некоторый *стандартный метод обучения* базовых алгоритмов μ, решающий задачу минимизации функционала качества $Q(b)$ в заранее заданном семействе алгоритмических операторов B:

$$\mu(X^\ell, Y^\ell, W^\ell) = \arg\min_{b \in B} Q(b; X^\ell, Y^\ell, W^\ell).$$

§1.2 Последовательное обучение базовых алгоритмов

Процесс последовательного обучения базовых алгоритмов применяется, пожалуй, чаще всего при построении композиций. Рассмотрим сначала этот процесс в наиболее общем виде, как показано в Алгоритме 1.2.

На первом шаге с помощью стандартного метода обучения μ строится первый базовый алгоритм b_1. Если его качество удовлетворяет, то дальнейшее построение композиции лишено смысла, и процесс на этом завершается. В противном случае алгоритм b_1 фиксируется и строится второй алгоритм b_2, при одновременной оптимизации корректирующей операции F. На t-м шаге базовый алгоритм b_t и корректирующая операция F оптимизируются при фиксированных $b_1, \ldots, b_{t-1}$:

$$b_1 = \arg\min_{b} Q(b; X^\ell, Y^\ell); \tag{1.7}$$

$$b_2 = \arg\min_{b,F} Q(F(b_1, b); X^\ell, Y^\ell);$$

$$\ldots$$

$$b_t = \arg\min_{b,F} Q(F(b_1, \ldots, b_{t-1}, b); X^\ell, Y^\ell). \tag{1.8}$$

Во многих случаях для решения задачи (1.8) удаётся приспособить стандартные методы обучения, решающие задачу более простого вида (1.7). Для этого на вход стандартного метода обучения $\mu(X^\ell, Y^\ell, W^\ell)$ подаются модифицированные векторы весов W^ℓ и ответов Y^ℓ. Конкретный способ модификации зависит от типа задачи (классификация или регрессия) и вида корректирующей операции F. Для каждого частного случая формулы пересчёта весов и ответов выводятся отдельно. Забегая немного вперёд, заметим, что модификация весов, как правило, сводится к увеличению весов у наиболее «трудных» объектов, на которых чаще ошибались предыдущие базовые алгоритмы, а модификация ответов — к аппроксимации невязки $y_i - a(x_i)$ вместо аппроксимации исходных ответов y_i.

Базовый алгоритм b_t, оптимальный на t-м шаге, перестаёт быть оптимальным после добавления следующих алгоритмов. Процесс (1.7)–(1.8) можно обобщить, чередуя добавление новых алгоритмов с перенастройкой предыдущих:

$$b_k = \arg\min_{b,F} Q(F(b_1, \ldots, b_{k-1}, b, b_{k+1}, \ldots, b_t); X^\ell, Y^\ell), \quad 1 \leqslant k < t.$$

По способам решения данная задача мало чем отличается от задачи построения последнего базового алгоритма (1.8). Пример итерационного процесса с возвратами будет рассмотрен для одного частного случая в разделе 1.4.2.

Критерии останова могут использоваться различные, в зависимости от специфики задачи; возможно также совместное применение нескольких критериев:

- Построено заданное количество базовых алгоритмов T.

Алгоритм 1.2. Построение алгоритмической композиции путём последовательного обучения базовых алгоритмов

Вход:
X^ℓ, Y^ℓ — обучающая выборка; μ — метод обучения базовых алгоритмов;
T — максимальное число базовых алгоритмов в композиции;
Выход:
алгоритмическая композиция $F(b_1, \ldots, b_T)$;

1: инициализировать веса: $w_i := 1$ для всех $i = 1, \ldots, \ell$;
2: **для всех** $t = 1, \ldots, T$, пока не выполнен критерий останова
3: $b_t := \mu(X^\ell, Y^\ell, W^\ell)$;
4: модифицировать веса W^ℓ и/или ответы Y^ℓ с учётом ошибок, допущенных b_t;

- Достигнута заданная точность на обучающей выборке:

$$Q(F(b_1, \ldots, b_t); X^\ell, Y^\ell) \leqslant \varepsilon.$$

- Достигнутую точность на контрольной выборке X^k не удаётся улучшить на протяжении последних d шагов: $t - t^* \geqslant d$, где d — параметр алгоритма,

$$t^* = \arg\min_{s=1,\ldots,t} Q(F(b_1, \ldots, b_s); X^k, Y^k).$$

Выполнение этого критерия считается признаком переобучения. В качестве окончательного решения берётся композиция, построенная на t^*-м шаге.

Ниже будут рассмотрены различные методы последовательного построения алгоритмических композиций, отличающиеся видом корректирующей операции и способом сведения задачи (1.8) к задаче (1.7).

Последовательное построение базовых алгоритмов проще всего реализуется, когда корректирующая операция не имеет собственных настраиваемых параметров. К таким методам относится голосование по большинству и по старшинству.

1.2.1 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:

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

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

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

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 $i = 1, \ldots, \ell$;
2: **для всех** $t = 1, \ldots, T$, 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 $i = 1, \ldots, \ell$;
5: упорядочить выборку X^ℓ по возрастанию значений отступов M_i;
6: выбрать ℓ_1;
7: $w_i := [i \leqslant \ell_1]$ для всех $i = 1, \ldots, \ell$;

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 $w_i = [M_{it} \leqslant M_0]$.

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.

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 $w_i := [\ell_0 \leqslant i \leqslant \ell_1]$. 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 [30]:

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

Легко видеть, что тогда на t-й итерации $w_i = \gamma^{-M_{it}}$. 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 $\big(b_1(x), \ldots, b_T(x)\big)$ 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 персептрон (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 $\alpha_t \geqslant 0$. 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 1.6. Самый простой линейный метод — «наивный» байесовский классификатор (стр. **??**). Он исходит из предположения, что $b_1(x), \ldots, b_T(x)$ являются независимыми случайными величинами. Нетрудно доказать, что в этом случае коэффициенты взвешенного голосования вычисляются по явной формуле [29]:

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

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 $b_t \colon X \to \mathbb{R}$ всё равно будем строить так, чтобы они решали двухклассовую задачу, отделяя заданный класс c_t от всех остальных классов. Обозначим через $T_y = \{t \colon c_t = y\}$ множество индексов всех базовых алгоритмов, предназначенных для отделения класса y. Запишем алгоритм простого голосования, применив решающее правило из Примера 1.3:

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

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.

1.2.2 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, $b_t \colon X \to \{0, 1\}$. В качестве корректирующей операции возьмём голосование по старшинству, см. Алгоритм 1.1. 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:

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

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 $i = 1, \ldots, \ell$;
2: **для всех** $t = 1, \ldots, T$, 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 $i = 1, \ldots, \ell$;
5: **если** $w_i \neq 0$ **то** $w_i := (1 - z_i) + \lambda z_i$ для всех $i = 1, \ldots, \ell$;
6: $b_t := \mu(X^\ell, Z^\ell, W^\ell)$;
7: **если** $b_t(x_i) = 1$ **то** $w_i := 0$ для всех $i = 1, \ldots, \ell$;

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 $Q(a) = Q\big(F(b_1, \ldots, b_t)\big)$ по базовому алгоритму b_t эквивалентна минимизации функционала

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

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 $W^\ell = (w_1, \ldots, w_\ell)$ и бинарный вектор ответов $Z^\ell = (z_1, \ldots, z_\ell)$, 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 скомпенсировано последующими базовыми алгоритмами. Например, можно начинать со значения $\lambda = 0.1$, постепенно увеличивая его от итерации к итерации.

- 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 $c_1, \ldots, c_T$. 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 $Y = \{v_1, \ldots, v_M\}$, разбивающий последовательность базовых алгоритмов на списки по T_m алгоритмов для каждого из классов v_m, $m = 1, \ldots, M$:

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

- Задаётся приоритетный порядок классов $Y = \{v_1, \ldots, v_M\}$, 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 7:

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

  Во многих приложениях такие композиции легче поддаются содержательной интерпретации.

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

Пример 1.7. Задача на плоскости: $X = \mathbb{R}^2$, $Y = \{\circ, \times\}$, $b_t(x)$ — полуплоскости.

1.2.3 Бустинг в задачах классификации

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

Искомая алгоритмическая композиция имеет вид:

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

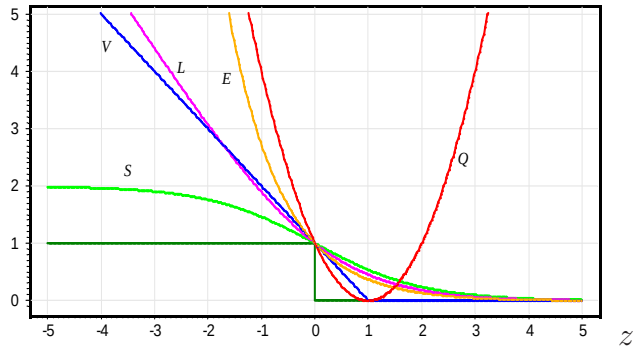

Рис. 1. Гладкие верхние аппроксимации пороговой функции потерь $[z < 0]$:
$S(z) = 2(1 + e^z)^{-1}$ — сигмоидная;
$L(z) = \log_2(1+e^{-z})$ — логарифмическая;
$V(z) = (1 - z)_+$ — кусочно-линейная;
$E(z) = e^{-z}$ — экспоненциальная;
$Q(z) = (1 - z)^2$ — квадратичная.

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

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

Для упрощения задачи минимизации функционала Q_T введём две эвристики.

Эвристика 1. При добавлении в композицию слагаемого $\alpha_t b_t(x)$ будем оптимизировать только базовый алгоритм b_t и коэффициент при нём α_t, не трогая предыдущих слагаемых $\alpha_j b_j(x)$, $j = 1, \ldots, t-1$.

Эвристика 2. Чтобы решить задачу оптимизации параметра α_t аналитически, аппроксимируем пороговую функцию потерь в функционале Q_t какой-нибудь её 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 (**??**) и логистической регрессии (**??**). Кусочно-линейная аппроксимация связана с принципом оптимальной разделяющей поверхности (максимизации зазора между классами) и применяется в методе опорных векторов (**??**). Примеры других аппроксимаций показаны на Рис. 1.

Процесс последовательного обучения базовых алгоритмов при экспоненциальной аппроксимации приводит к широко известному алгоритму AdaBoost [37].

Оценим функционал Q_T сверху:

$$Q_T \leqslant \widetilde{Q}_T = \sum_{i=1}^{\ell} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t b_t(x_i) \right) =$$
$$= \sum_{i=1}^{\ell} \underbrace{\exp\left(-y_i \sum_{t=1}^{T-1} \alpha_t b_t(x_i) \right)}_{w_i} \exp(-y_i \alpha_T b_T(x_i)). \qquad (1.9)$$

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

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

Теорема 1.1. *Пусть Q — стандартный функционал качества алгоритма классификации b на обучающей выборке X^ℓ, Y^ℓ с нормированным вектором весов объектов U^ℓ:*

$$Q(b;U^\ell)=\sum_{i=1}^{\ell}u_i\big[y_ib(x_i)<0\big],\qquad\sum_{i=1}^{\ell}u_i=1.$$

Пусть $\min\limits_b Q(b;W^\ell)<1/2$ для любого нормированного вектора весов W^ℓ. Тогда минимум функционала $\widetilde{Q}_T$ достигается при

$$b_T=\arg\min_b Q(b;\widetilde{W}^\ell);$$
$$\alpha_T=\frac{1}{2}\ln\frac{1-Q(b_T;\widetilde{W}^\ell)}{Q(b_T;\widetilde{W}^\ell)}.$$

Доказательство.

Выразим $\widetilde{Q}_T$ через функционал качества базового алгоритма. Это легко сделать, если воспользоваться очевидным разложением $e^{-\alpha\beta}=e^{-\alpha}[\beta=1]+e^{\alpha}[\beta=-1]$, справедливым для любых $\alpha\in\mathbb{R}$ и $\beta\in\{-1,1\}$:

$$\widetilde{Q}_T=\Bigg(e^{-\alpha_T}\underbrace{\sum_{i=1}^{\ell}\tilde{w}_i[b_T(x_i)=y_i]}_{1-Q}+e^{\alpha_T}\underbrace{\sum_{i=1}^{\ell}\tilde{w}_i[b_T(x_i)\neq y_i]}_{Q}\Bigg)\underbrace{\sum_{i=1}^{\ell}w_i}_{\widetilde{Q}_{T-1}}=$$
$$=\big(e^{-\alpha_T}(1-Q)+e^{\alpha_T}Q\big)\widetilde{Q}_{T-1},$$

где введено краткое обозначение $Q=Q(b_T;\widetilde{W}^\ell)$. Дифференцируя функционал $\widetilde{Q}_T$ по параметру α_T и приравнивая нулю производную, получим оптимальное значение

$$\alpha_T=\frac{1}{2}\ln\frac{1-Q}{Q}.$$

Подставляя его обратно в $\widetilde{Q}_T$, получим

$$\widetilde{Q}_T=\widetilde{Q}_{T-1}\sqrt{4Q(1-Q)}. \tag{1.10}$$

Поскольку $\widetilde{Q}_{T-1}$ фиксировано, минимизация $\widetilde{Q}_T$ эквивалентна минимизации Q при $Q<1/2$ и максимизации Q при $Q>1/2$, однако второй случай не проходит по условию теоремы. ■

Требование $Q(b_t;\widetilde{W}^\ell)<1/2$ означает, что метод обучения должен обеспечивать построение алгоритма $b_t(x)$, доля ошибок которого на обучающей выборке хотя бы немного меньше $1/2$. Иными словами, от каждого базового алгоритма требуется, чтобы он классифицировал объекты хотя бы немного лучше, чем наугад. Это достаточно слабое требование, и на практике оно, как правило, выполняется. Более того, этого условия 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 $Q(b_t;\widetilde{W}^\ell)<1/2-\gamma$ при некотором $\gamma>0$, то AdaBoost гарантирует построение корректного алгоритма $a(x)$ за конечное число шагов.*

Алгоритм 1.5. AdaBoost — построение линейной комбинации классификаторов

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

Доказательство.

Непосредственно из (1.10) вытекает сходимость функционала Q_T к нулю со скоростью геометрической прогрессии: $Q_{T+1} \leqslant \widetilde{Q}_{T+1} \leqslant \widetilde{Q}_1(1 - 4\gamma^2)^{\frac{T}{2}}$. Наступит момент, когда значение $\widetilde{Q}_T$ окажется меньше 1. Но тогда функционал Q_T обратится в нуль, поскольку он может принимать только целые неотрицательные значения. ■

Реализация AdaBoost показана в Алгоритме 1.5. Это очень простой алгоритм, в котором буквально повторяются формулы, доказанные в теореме.

Некоторых комментариев заслуживает правило мультипликативного пересчёта весов на шаге 5. Оно вытекает непосредственно из определения w_i в (1.9). 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.

Замечание 1.1. 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 [22]. 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 [17]. Эффективность бустинга объясняется тем, что по мере добавления базовых алгоритмов увеличиваются отступы обучающих объектов $M_i = y_i a(x_i)$. Причём бустинг продолжает «раздвигать» классы даже после достижения безошибочной классификации обучающей выборки.

К сожалению, теоретические оценки обобщающей способности [17] 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 $10^4 \ldots 10^6$.

Более основательные эксперименты показали, что иногда бустинг всё же переобучается [35, 34]. По мнению автора статьи [24] 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.
- Собственные накладные расходы бустинга невелики. Время построения композиции практически полностью определяется временем обучения базовых алгоритмов.
- Возможность идентифицировать объекты, являющиеся шумовыми выбросами. 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 [23].
- 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.

§1.3 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 алгоритмах 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 [18]. 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 $\ell \to \infty$. 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 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

Алгоритм 1.6. 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** $t = 1, \ldots, T$, пока не выполнен критерий останова
2: $U :=$ случайное подмножество X^ℓ длины ℓ';
3: $\mathscr{G} :=$ случайное подмножество $\mathscr{F}$ длины n';
4: $b_t := \mu(\mathscr{G}, U)$;
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;

строить подвыборки меньшей длины $\ell_0 \ll \ell$, при этом возникает задача подбора оптимального значения ℓ_0.

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 оказываться точнее алгоритмов, построенных по всем признакам.

Обобщение бэггинга и RSM приводит к Алгоритму 1.6. Пусть объекты описываются набором признаков $\mathscr{F} = \{f_1, \ldots, f_n\}$ и метод обучения $\mu(\mathscr{G}, U)$ строит алгоритмический оператор по обучающей подвыборке $U \subseteq X^\ell$, используя только часть признакового описания $\mathscr{G} \subseteq \mathscr{F}$. Числа $\ell' = |U| \leqslant \ell$ и $n' = |\mathscr{G}| \leqslant n$ являются параметрами метода. Алгоритм 1.6 соответствует бэггингу при $\mathscr{G} = \mathscr{F}$, 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 $\mathscr{F} = \{f_1, \ldots, f_n\}$, и задан метод обучения μ, позволяющий строить базовые алгоритмы $b = \mu(\mathscr{G}, U)$ по различным подвыборкам $U \subseteq X^\ell$ и различным частям признакового описания $\mathscr{G} \subseteq \mathscr{F}$.

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* [33]. Как всякий эволюционный алгоритм, он представляет собой итерационный процесс смены поколений, см. Алгоритм 1.7.

Инициализация нулевого поколения производится случайным образом.

На t-м поколении строится не одна, а $p(t)$ популяций $\Pi_1(t), \ldots, \Pi_{p(t)}(t)$. Каждая популяция $\Pi_j(t)$ представляет собой множество индивидов. Каждый индивид кодируется $(\ell + n)$-мерным бинарным вектором, составленным из характеристических векторов подмножества объектов $X' \subseteq \{x_1, \ldots, x_\ell\}$ и подмножества признаков $G' \subseteq \{g_1, \ldots, g_n\}$. 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 $\Pi_s(t)$, $s \neq j$:

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

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 $\Pi_j(t+1)$, $j = 1, \ldots, p(t)$. Из каждого поколения t отбирается и запоминается лучшая композиция вида $F(b_1, \ldots, b_{p(t)})$.

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 1.7. 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 $a = F(b_1, \ldots, b_p)$;

начальное число популяций: $p(1) := 1$;
создать популяцию: $\Pi_1(1) := \texttt{Инициализация}(N_0)$;
для всех поколений $t := 1, \ldots, T_{\max}$
  для всех популяций $\Pi_j(t)$, $j := 1, \ldots, p(t)$
    породить промежуточную популяцию:
      $\Pi'_j := \texttt{Рекомбинация}(\Pi_j(t), N_1)$;
      $\Pi''_j := \texttt{Мутация}(\Pi'_j) \cup \texttt{Селекция}(\Pi_j(t), N_2)$;
    для всех $v_j \in \Pi''_j$
      оценить адаптивность индивида $\varphi(v_j)$;
      $Q_t := \min\{Q_t, \varphi(v_j)\}$;
      запомнить лучшего индивида в популяции:
        $v^*_j := \arg\min\limits_{v \in \Pi''_j} \varphi(v)$; $b^*_j := \mu(v^*_j)$;
    отобрать лучших индивидов в следующее поколение:
      $\Pi_j(t+1) := \texttt{Селекция}(\Pi''_j, N_0)$;
    если $\texttt{Вклад}(\Pi_j) < \delta$ **то**
      удалить популяцию Π_j; $p(t+1) := p(t) - 1$;
  если $\texttt{Стагнация}(Q, t)$ **то**
    $p(t+1) := p(t) + 1$; добавить $\Pi_{p(t+1)}(t+1) := \texttt{Инициализация}(N_0)$;
  если $\texttt{Останов}(Q, t)$ **то**
    выход;
вернуть композицию $F\big(b^*_1, \ldots, b^*_{p(t)}\big)$, на которой достигается $\min\limits_t Q_t$.

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.

$\texttt{Инициализация}(N_0)$ — 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 CCEL 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 CCEL 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: $\frac{1}{d}\sum_{\tau=t-d+1}^{t}\big(\bar{Q}_{j\tau} - Q_{j\tau}\big)$. 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 CCEL.

- Эксперименты на реальных данных показали, что качество классификации (обобщающая способность) CCEL не хуже, чем у бустинга или бэггинга [3, 27].
- CCEL 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.
- CCEL применим к любым базовым алгоритмам и методам их обучения.
- Автоматически отбираются информативные объекты и признаки, что даёт дополнительную информацию для понимания задачи.

- Процесс обучения CCEL 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 CCEL.

- Метод CCEL сложен для реализации и имеет большое количество параметров, которые приходится задавать из эвристических соображений.
- Если не использовать распараллеливания, CCEL 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* $F\colon \mathbb{R}^{2T} \to \mathbb{R}$,

$$F\big(b_1,\dots,b_T,g_1,\dots,g_T\big) = \sum_{t=1}^{T} g_t b_t. \tag{1.11}$$

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 бóльшим весом учитывается ответ алгоритма $b_t(x)$ на объекте x. Множество

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

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 (1.11) принято называть *смесями 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:

$$\sum_{t=1}^{T} g_t(x) = 1, \quad \text{для любого } x \in X. \tag{1.12}$$

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 $\big[\min\limits_t b_t(x), \max\limits_t b_t(x)\big]$.

Чтобы гарантировать выполнение условия нормировки (1.12), используют функцию «мягкого максимума» $\mathrm{SoftMax}\colon \mathbb{R}^T \to \mathbb{R}^T$:

$$\tilde{g}_t(x) = \mathrm{SoftMax}_t\big(g_1(x), \ldots, g_T(x)\big) = \frac{e^{\beta g_t(x)}}{e^{\beta g_1(x)} + \cdots + e^{\beta g_T(x)}}.$$

Эта функция переводит вектор $(g_1, \ldots, g_T)$ в нормированный вектор $(\tilde{g}_1, \ldots, \tilde{g}_T)$. При $\beta \to \infty$, 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 $C(b) = \mathrm{sign}(b)$ делать нормировку, вообще говоря, не обязательно, поскольку для классификации важен только знак выражения (1.11).

Задача обнаружения нетипичных объектов. Если значение $g_t(x)$ меньше некоторого порога для всех $t = 1, \ldots, T$, 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:

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

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

Вероятностная интерпретация приводит к EM-алгоритму, который обрастает массой технических усложнений [16, 26, 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

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

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

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

$$\widetilde{L}(g_1b_1 + g_2b_2, y) \leqslant g_1\widetilde{L}(b_1,y) + g_2\widetilde{L}(b_2,y).$$

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 $\widetilde{L}(b,y) = (b-y)^2$, применяемая в регрессионных задачах. К сожалению, пороговая функция $\widetilde{L}(b,y) = [by < 0]$, применяемая в задачах классификации при $Y = \{-1, 1\}$, не является выпуклой. Однако некоторые её непрерывные аппроксимации выпуклы, см. Рис. 1 на стр. 14:

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

Замена пороговой функции потерь на непрерывную является распространённой практикой. В частности, логарифмическая аппроксимация используется в логистической регрессии (см. **??**), экспоненциальная — в алгоритме AdaBoost (см. 1.2.3), 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 (1.12) и функция потерь $\widetilde{L}$ выпукла. Вместо функционала $Q(a)$ будем минимизировать его верхнюю оценку, которая непосредственно вытекает из условия выпуклости:

$$Q(a) = \sum_{i=1}^{\ell} \widetilde{L}\Big(\sum_{t=1}^{T} \tilde{g}_t(x_i) b_t(x_i), y_i\Big) \leqslant \sum_{t=1}^{T} \underbrace{\sum_{i=1}^{\ell} \tilde{g}_t(x_i) \widetilde{L}\big(b_t(x_i), y_i\big)}_{Q(b_t; W^\ell)}. \tag{1.13}$$

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. 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 1.8 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:

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

где $z_i = \psi\big(L(b(x_i), y_i)\big)$ — 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;

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

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

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

Алгоритм 1.8. Последовательное построение смеси алгоритмов

Параметры:
ℓ_0 — допустимое число ошибок;
δ — пороговое значение функции потерь;

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

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 объёме 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 поочерёдно подстраивает все базовые алгоритмы и их функции компетентности (шаги 7–10). Согласно (1.13) 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:

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

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

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

Рассмотрим более подробно процедуру настройки $\texttt{LearnG}_j$, 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:

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

Таким образом, настройка отдельной функции компетентности сводится к задаче классификации, которая решается путём минимизации стандартного функционала качества Q. 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:

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

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:

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

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

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

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:

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

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

$$\widetilde{Q}(g_1) = \sum_{i=1}^{\ell} g_1(x_i)\underbrace{w_i\big(\widetilde{L}(b_1(x_i),y_i) - \widetilde{L}(b_2(x_i),y_i)\big)}_{d_i=\text{const}(g_1)}. \tag{1.14}$$

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 1.9 описывает процедуру M2E (mixture of 2 experts) подробно.

Алгоритм 1.9. M2E — построение смеси двух базовых алгоритмов

Вход:
X^ℓ, Y^ℓ — обучающая выборка;
$W^\ell = (w_1, \ldots, w_\ell)$ — исходный вектор весов объектов;
$b_0(x)$ — начальное приближение одного из базовых алгоритмов;
Выход:
алгоритмическая композиция: $a(x) = b_1(x)g_1(x) + b_2(x)g_2(x)$;

$g_1 := \arg\min_g \sum_{i=1}^{\ell} g(x_i) w_i \widetilde{L}(b_0(x_i), y_i)$;
повторять
  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 $w_{2i} = w_i(1 - g_1(x_i))$;
  настройка функции компетентности:
    $d_i := w_i\big(\widetilde{L}(b_1(x_i), y_i) - \widetilde{L}(b_2(x_i), y_i)\big)$, для всех $i = 1, \ldots, \ell$;
    $g_1 := \arg\min_g \sum_{i=1}^{\ell} d_i g(x_i)$;
  оценка качества композиции:
    $Q := \sum_{i=1}^{\ell} w_i L\big(b_1(x_i)g_1(x_i) + b_2(x_i)g_2(x_i), y_i\big)$;
пока не стабилизируется значение Q;

Иерархическая смесь алгоритмов. Метод M2E 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, HME). Алгоритм 1.10 более подробно описывает детали рекурсивной реализации этой идеи.

Способ построения HME 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.

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:

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

Недостатком этого критерия является необходимость задания параметра ε.

Алгоритм 1.10. Рекурсивный алгоритм построения иерархической смеси алгоритмов

Вход:
X^ℓ, Y^ℓ — обучающая выборка;

ПРОЦЕДУРА `Разветвить` (W^ℓ, b_0):
  $W^\ell = (w_1, \ldots, w_\ell)$ — веса обучающих объектов;
  b_0 — алгоритм, заменяемый смесью $a(x) = b_1(x)g_1(x) + b_2(x)g_2(x)$;
$b_1, g_1, b_2, g_2 := M2E(W^\ell, b_0)$;
$W_1^\ell = \big(w_i g_1(x_i)\big)_{i=1}^\ell$;
если `КритерийВетвления` (W_1^ℓ, b_1) **то**
  `Разветвить` (W_1^ℓ, b_1);
$W_2^\ell = \big(w_i g_2(x_i)\big)_{i=1}^\ell$;
если `КритерийВетвления` (W_2^ℓ, b_2) **то**
  `Разветвить` (W_2^ℓ, b_2);

Основной алгоритм
$W^\ell := (1/\ell, \ldots, 1/\ell)$;
$b_0 := \mu(X^\ell, Y^\ell, W^\ell)$;
`Разветвить` (W^ℓ, b_0);

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

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

  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 $F(b_1, \ldots, b_T) = \alpha_1 b_1 + \cdots + \alpha_T b_T$ 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 бóльшими возможностями для настройки. В то же время, монотонность является более естественным требованием к корректирующей операции, нежели линейность. Монотонность F как отображения $R^T \to Y$ означает, что одновременное увеличение значений $b_1(x), \ldots, b_T(x)$ не должно приводить к уменьшению значения $F(b_1(x), \ldots, b_T(x))$. 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 $F(b_1(x), \ldots, b_T(x))$, 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 $F\colon R^T \to Y$ 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 $a(x) = F(b_1(x), \ldots, b_T(x))$.

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 $u_j \leqslant u_k$ следует $v_j \leqslant v_k$ для всех $j, k = 1, \ldots, \ell$.*

Лемма 1.3. *Пусть U, V — произвольные частично упорядоченные множества. Монотонная функция $f\colon U \to V$ такая, что $f(u_i) = v_i$ для всех $i = 1, \ldots, \ell$, 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 $I(u) = \{k\colon u_i \leqslant u\}$ и положим

$$f(u) = \begin{cases} \min\limits_{i=1,\ldots,\ell} v_i, & \text{если } I(u) = \varnothing; \\ \max\limits_{i \in I(u)} v_i, & \text{если } I(u) \neq \varnothing. \end{cases}$$

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

Докажем, что $f(u_i) = v_i$ для всех $i = 1, \ldots, \ell$. 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 $u_k \leqslant u_i$ следует $v_k \leqslant v_i$. Но тогда $\max\limits_{k \in I(u_i)} v_k$ 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:

$$u_i = (b_1(x_i), \ldots, b_t(x_i));$$
$$f_i = a(x_i) = F(b_1(x_i), \ldots, b_t(x_i)) = F(u_i); \quad i = 1, \ldots, \ell.$$

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

$$F(u_i) = y_i, \quad i = 1, \ldots, \ell. \tag{1.15}$$

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

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

Дефектная пара обладает тем свойством, что для любой монотонной функции F выполняется $F(b(x_j)) \geqslant F(b(x_k))$, 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 $b_1, \ldots, b_t$ называется множество*

$$\mathbb{D}_T(b_1, \ldots, b_t) = \mathbb{D}(b_1) \cap \ldots \cap \mathbb{D}(b_t) = \{(j, k)\colon y_j < y_k \wedge u_j \geqslant u_k\}.$$

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

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

Теорема 1.4. *Монотонная функция $F\colon R^t \to Y$, удовлетворяющая условию корректности 1.15 существует тогда и только тогда, когда совокупный дефект операторов $b_1, \ldots, b_t$ пуст. При этом дефект алгоритма $a = F(b_1, \ldots, b_t)$ также пуст.*

Доказательство.

Справедлива следующая цепочка равносильных утверждений:

а) совокупный дефект пуст: $\mathbb{D}_t(b_1, \ldots, b_t) = \varnothing$;

б) для любых j, k не выполняется $(u_k \leqslant u_j) \wedge (y_j < y_k)$ (согласно Опр. 1.6);

в) для любых j, k справедлива импликация $(u_k \leqslant u_j) \to (y_k \leqslant y_j)$;

г) совокупность пар $(u_i, y_i)_{i=1}^{\ell}$ монотонна (согласно Опр. 1.4);

д) существует монотонная функция $F\colon R^t \to Y$ такая, что $F(u_i) = y_i$ для всех $i = 1, \ldots, \ell$ (согласно Лемме 1.3).

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

Итак, чтобы гарантировать корректность композиции $a = F(b_1, \ldots, b_t)$, достаточно построить операторы $b_1, \ldots, b_t$, 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

$$b_t(x_j) < b_t(x_k), \quad (j, k) \in \mathbb{D}_{t-1}, \tag{1.16}$$

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 (1.16) приводит к *устранению дефектной пары* (j,k).

Теорема 1.5 (о сходимости). *Пусть на первом шаге процесса (1.7) построен оператор b_1, и семейство операторов B выбрано так, что для любой подвыборки X^{2m} длины $2m$, $m \geqslant 1$, 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 (1.7–1.8) приводит к построению корректной композиции $a = F(b_1, \ldots, b_T)$ за конечное число шагов $T \leqslant \lceil \mathbb{D}(b_1)/m \rceil + 1$.

Доказательство.

Рассмотрим t-й шаг, $t \geqslant 2$, итерационного процесса (1.8). Если $|\mathbb{D}_{t-1}| > m$, то выберем некоторое m-элементное подмножество совокупного дефекта $\Delta_{t-1} \subseteq \mathbb{D}_{t-1}$. Если $|\mathbb{D}_{t-1}| \leqslant m$, то положим $\Delta_{t-1} = \mathbb{D}_{t-1}$. Рассмотрим подмножество объектов выборки, образующих всевозможные дефектные пары из Δ_{t-1}:

$$U = \left\{ x_i \in X^\ell \;\middle|\; \exists k\colon (k,i) \in \Delta_{t-1} \text{ или } (i,k) \in \Delta_{t-1} \right\}.$$

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 $b_t(x_k) \leqslant b_t(x_j)$, тогда $F(b_t(x_k)) \leqslant F(b_t(x_j))$, следовательно, $y_k \leqslant y_j$, что противоречит условию $y_j < y_k$, входящему в определение дефектной пары (j,k).

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

Процесс последовательного построения базовых алгоритмов можно организовать 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 (1.16) 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 (1.5), 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 1.6. *Если $Y = \{0, 1\}$ и функция потерь имеет вид $\widetilde{L}(b, y) = \big[[b > 0] \neq y\big]$, то справедлива оценка*

$$|\mathbb{D}_t(b_1, \ldots, b_t)| \leqslant \sum_{i=1}^{\ell} w_i \, \widetilde{L}(b_t(x_i), y_i); \tag{1.17}$$

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

$$D(i) = \big\{x_k \in X^\ell \;\big|\; (k, i) \in \mathbb{D}_{t-1} \text{ или } (i, k) \in \mathbb{D}_{t-1}\big\}; \quad i = 1, \ldots, \ell. \tag{1.18}$$

Доказательство.

Обозначим через $\beta_i = b_t(x_i)$ значение t-го оператора на i-м обучающем объекте, через $L_i = \widetilde{L}(\beta_i, y_i) = \big[[\beta_i > 0] \neq y_i\big]$ — 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

$$[\beta_j \geqslant \beta_k] \leqslant [\beta_j > 0] + [\beta_k \leqslant 0].$$

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

$$y_j = 0; \qquad [\beta_j > 0] = \big[[\beta_j > 0] \neq 0\big] = \big[[\beta_j > 0] \neq y_j\big] = L_j;$$
$$y_k = 1; \qquad [\beta_k \leqslant 0] = \big[[\beta_k > 0] \neq 1\big] = \big[[\beta_k > 0] \neq y_k\big] = L_k.$$

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 $[\beta_j \geqslant \beta_k] \leqslant L_j + L_k$:

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

■

В случае регрессии справедлива аналогичная теорема [2].

Теорема 1.7. *Если $Y = \mathbb{R}$ и функция потерь имеет вид $\widetilde{L}(b, y) = (b - y)^2$, то справедлива та же оценка (1.17), только веса определяются по-другому: $w_i = h_i^{-2}|D(i)|$, где $h_i = \frac{1}{2} \min\limits_{k \in D(i)} |y_i - y_k|$, $D(i)$ определяется согласно (1.18).*

1.5.2 Монотонная интерполяция и аппроксимация

Если совокупный дефект операторов $b_1, \ldots, b_t$ пуст, то существует монотонная функция F такая, что $F(u_i) = y_i$ для всех $i = 1, \ldots, \ell$. 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

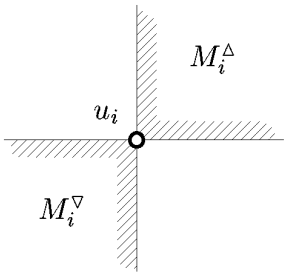

Рис. 2. Верхний ($M_i^{\triangle}$) и нижний ($M_i^{\triangledown}$) конусы вектора u_i.

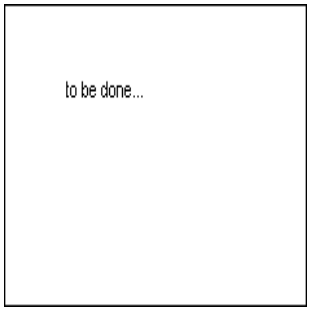

Рис. 3. Расстояние от вектора u до ближайшего верхнего конуса.

функции F, приближенно удовлетворяющей условиям $F(u_i) = y_i$, называется задачей *монотонной аппроксимации*. Она решается в два этапа.

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

$$\begin{cases} \sum\limits_{i=1}^{\ell} (f_i - y_i)^2 \to \min; \\ f_j \leqslant f_k, \text{ для всех } (i,j)\colon u_j \leqslant u_k. \end{cases}$$

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 $i = 1, \ldots, \ell$.

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 $F\colon R^t \to R$ проходящую через все точки выборки: $F(u_i) = f_i$, $i = 1, \ldots, \ell$. Предполагается, что среди значений f_i есть различные (задача не вырождена).

Определим *верхний конус* $M_i^{\triangle}$ и *нижний конус* $M_i^{\triangledown}$ для каждого из векторов u_i, как показано на Рис. 2:

$$M_i^{\triangle} = \{u \in R^t \mid u_i \leqslant u\};$$
$$M_i^{\triangledown} = \{u \in R^t \mid u \leqslant u_i\}.$$

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

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

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

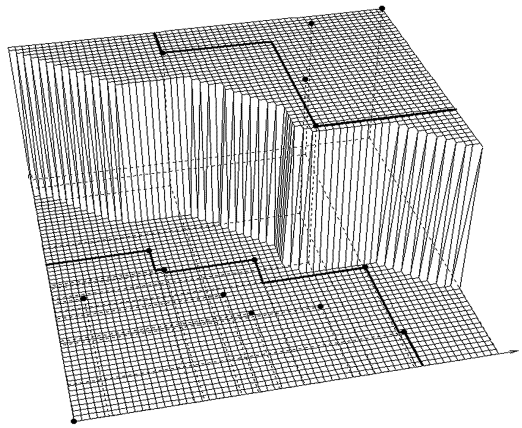

Рис. 4. Дискретная монотонная ступенька $F(u)$ для задачи классификации.

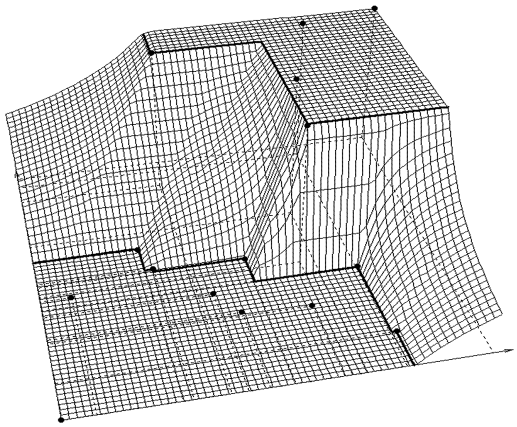

Рис. 5. Непрерывная монотонная ступенька $\Phi(u,\gamma)$ для задачи восстановления регрессии.

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

$$r_i^{\triangle}(u) = \mu\left((u_i^1 - u^1)_+,\ldots,(u_i^t - u^t)_+\right);$$
$$r_i^{\triangledown}(u) = \mu\left((u^1 - u_i^1)_+,\ldots,(u^t - u_i^t)_+\right);$$

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

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

$$h^{\triangle}(u,\gamma) = \min_{i:f_i>\gamma} r_i^{\triangle}(u);$$
$$h^{\triangledown}(u,\gamma) = \min_{i:f_i\leqslant\gamma} r_i^{\triangledown}(u).$$

Интерполяция бинарной монотонной функции. В задаче классификации на два класса $Y = \{0,1\}$ функции $h^{\triangle}$, $h^{\triangledown}$ 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 1.8. *Для задачи классификации, $Y = \{0,1\}$, функция*

$$F(u) = [h^{\triangle}(u,\gamma) \leqslant h^{\triangledown}(u,\gamma)]$$

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 $i = 1,\ldots,\ell$.

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 $\mu(z^1, \ldots, z_t) = (z_1^2 + \cdots + z_t^2)^{-1/2}$.

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

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

Нетрудно показать, что эта функция непрерывна, монотонно не убывает, и на векторах u_i, $i = 1, \ldots, \ell$ 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 1.9. *Пусть $Y = \mathbb{R}$ и точки выборки пронумерованы по возрастанию значений: $f_1 \leqslant f_2 \ldots \leqslant f_\ell$. 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 $i = 1, \ldots, \ell$. 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.

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

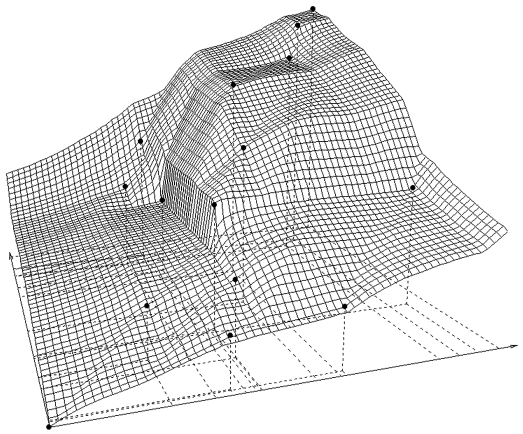

Рис. 6. Монотонная корректирующая операция $F(u)$ в задаче восстановления регрессии.

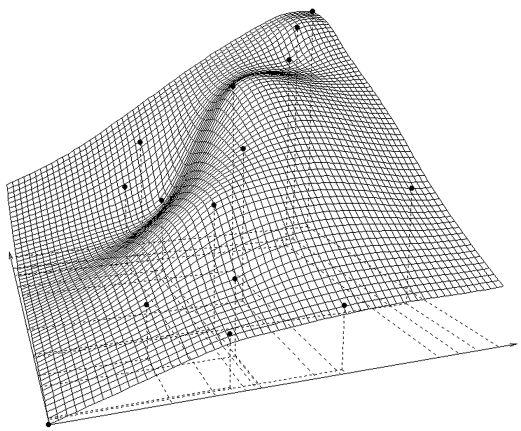

Рис. 7. Классический (немонотонный) гладкий сплайн, построенный по монотонной выборке.

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 [28] 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 $\ell \to \infty$. 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 [20]. 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 эффективности [21]. Аналогичные методы, но с несколько иной стратегией оптимизации базовых операторов, были разработаны Уолпертом [45] и Брейманом [18]. 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 [25] и алгоритмическим композициям [40] подчеркивается, что построение композиций, в которых различные алгоритмы компенсируют недостатки друг друга, является одним из наиболее перспективных направлений машинного обучения.

§1.7 Выводы

Основные свойства композиций, отличающие их от обычных алгоритмов.

- 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 очень удобно с технологической точки зрения: для построения базовых алгоритмов можно задействовать богатый арсенал стандартных методов обучения.
- Композиция позволяет получать высокое качество обучения, недостижимое для отдельных базовых алгоритмов.

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 экспертов [26].
- *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 [22], бэггинг [18], монотонная коррекция [2].

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 [19], 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 перестраиваются поочерёдно и многократно с помощью итерационного процесса, напоминающего EM-алгоритм [26]. 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. 1.6.

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

Список литературы

[1] *Воронцов К. В.* О проблемно-ориентированной оптимизации базисов задач распознавания // *ЖВМ и МФ.* — 1998. — Т. 38, № 5. — С. 870–880.
http://www.ccas.ru/frc/papers/voron98jvm.pdf.

[2] *Воронцов К. В.* Оптимизационные методы линейной и монотонной коррекции в алгебраическом подходе к проблеме распознавания // *ЖВМ и МФ.* — 2000. —

Т. 40, № 1. — С. 166–176.
http://www.ccas.ru/frc/papers/voron00jvm.pdf.

[3] *Воронцов К. В., Каневский Д. Ю.* Коэволюционный метод обучения алгоритмических композиций // *Таврический вестник информатики и математики.* — 2005. — № 2. — С. 51–66.
http://www.ccas.ru/frc/papers/voron05twim.pdf.

[4] *Журавлёв Ю. И.* Непараметрические задачи распознавания образов // *Кибернетика.* — 1976. — № 6.

[5] *Журавлёв Ю. И.* Экстремальные алгоритмы в математических моделях для задач распознавания и классификации // *Доклады АН СССР. Математика.* — 1976. — Т. 231, № 3.

[6] *Журавлёв Ю. И.* Об алгебраическом подходе к решению задач распознавания или классификации // *Проблемы кибернетики.* — 1978. — Т. 33. — С. 5–68.
http://www.ccas.ru/frc/papers/zhuravlev78prob33.pdf.

[7] *Журавлёв Ю. И., Рудаков К. В.* Об алгебраической коррекции процедур обработки (преобразования) информации // *Проблемы прикладной математики и информатики.* — 1987. — С. 187–198.
http://www.ccas.ru/frc/papers/zhurrud87correct.pdf.

[8] *Мазуров В. Д.* Комитеты системы неравенств и задача распознавания // *Кибернетика.* — 1971. — № 3.

[9] *Мазуров В. Д.* Метод комитетов в задачах оптимизации и классификации. — М.: Наука, 1990.

[10] *Растригин Л. А., Эренштейн Р. Х.* Коллективные правила распознавания. — М.: Энергия, 1981. — Р. 244.

[11] *Рудаков К. В.* Полнота и универсальные ограничения в проблеме коррекции эвристических алгоритмов классификации // *Кибернетика.* — 1987. — № 3. — С. 106–109.

[12] *Рудаков К. В.* Симметрические и функциональные ограничения в проблеме коррекции эвристических алгоритмов классификации // *Кибернетика.* — 1987. — № 4. — С. 73–77.
http://www.ccas.ru/frc/papers/rudakov87symmetr.pdf.

[13] *Рудаков К. В.* Универсальные и локальные ограничения в проблеме коррекции эвристических алгоритмов // *Кибернетика.* — 1987. — № 2. — С. 30–35.
http://www.ccas.ru/frc/papers/rudakov87universal.pdf.

[14] *Рудаков К. В.* О применении универсальных ограничений при исследовании алгоритмов классификации // *Кибернетика.* — 1988. — № 1. — С. 1–5.
http://www.ccas.ru/frc/papers/rudakov88universal.pdf.

[15] *Рудаков К. В., Воронцов К. В.* О методах оптимизации и монотонной коррекции в алгебраическом подходе к проблеме распознавания // *Докл. РАН.* — 1999. — Т. 367, № 3. — С. 314–317.
http://www.ccas.ru/frc/papers/rudvoron99dan.pdf.

[16] Adaptive mixtures of local experts / R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton // *Neural Computation.* — 1991. — no. 3. — Pp. 79–87.

[17] Boosting the margin: a new explanation for the effectiveness of voting methods / R. E. Schapire, Y. Freund, W. S. Lee, P. Bartlett // *Annals of Statistics.* — 1998. — Vol. 26, no. 5. — Pp. 1651–1686.
http://citeseer.ist.psu.edu/article/schapire98boosting.html.

[18] *Breiman L.* Bagging predictors // *Machine Learning.* — 1996. — Vol. 24, no. 2. — Pp. 123–140.
http://citeseer.ist.psu.edu/breiman96bagging.html.

[19] *Breiman L.* Arcing classifiers // *The Annals of Statistics.* — 1998. — Vol. 26, no. 3. — Pp. 801–849.
http://citeseer.ist.psu.edu/breiman98arcing.html.

[20] *Freund Y.* Boosting a weak learning algorithm by majority // COLT: Proceedings of the Workshop on Computational Learning Theory. — Morgan Kaufmann Publishers, 1990.
http://citeseer.ist.psu.edu/freund95boosting.html.

[21] *Freund Y., Schapire R. E.* A decision-theoretic generalization of on-line learning and an application to boosting // European Conference on Computational Learning Theory. — 1995. — Pp. 23–37.
http://citeseer.ist.psu.edu/article/freund95decisiontheoretic.html.

[22] *Freund Y., Schapire R. E.* Experiments with a new boosting algorithm // International Conference on Machine Learning. — 1996. — Pp. 148–156.
http://citeseer.ist.psu.edu/freund96experiments.html.

[23] *Friedman J., Hastie T., Tibshirani R.* Additive logistic regression: a statistical view of boosting: Tech. rep.: Dept. of Statistics, Stanford University Technical Report, 1998.
http://citeseer.ist.psu.edu/friedman98additive.html.

[24] *Grove A. J., Schuurmans D.* Boosting in the limit: Maximizing the margin of learned ensembles // AAAI/IAAI. — 1998. — Pp. 692–699.
http://citeseer.ist.psu.edu/grove98boosting.html.

[25] *Jain A. K., Duin R. P. W., Mao J.* Statistical pattern recognition: A review // *IEEE Transactions on Pattern Analysis and Machine Intelligence.* — 2000. — Vol. 22, no. 1. — Pp. 4–37.
http://citeseer.ist.psu.edu/article/jain00statistical.html.

[26] *Jordan M. I., Jacobs R. A.* Hierarchical mixtures of experts and the EM algorithm // *Neural Computation.* — 1994. — no. 6. — Pp. 181–214.
http://citeseer.ist.psu.edu/article/jordan94hierarchical.html.

[27] *Kanevskiy D. Y., Vorontsov K. V.* Cooperative coevolutionary ensemble learning // Multiple Classifier Systems: 7th International Workshop, Prague, Czech Republic, May 23-25, 2007. — Lecture Notes in Computer Science. Springer-Verlag, 2007. — Pp. 469–478.
http://www.ccas.ru/frc/papers/kanevskiy07ccel.pdf.

[28] *Kearns M., Valiant L. G.* Cryptographic limitations on learning Boolean formulae and finite automata // Proc. of the 21st Annual ACM Symposium on Theory of Computing. — 1989. — Pp. 433–444.
http://citeseer.ist.psu.edu/kearns89cryptographic.html.

[29] *Kuncheva L.* Combining pattern classifiers. — John Wiley & Sons, Inc., 2004.

[30] *Littlestone N., Warmuth M. K.* The weighted majority algorithm // IEEE Symposium on Foundations of Computer Science. — 1989. — Pp. 256–261.
http://citeseer.ist.psu.edu/littlestone92weighted.html.

[31] *Marchand M., Shawe-Taylor J.* Learning with the set covering machine // Proc. 18th International Conf. on Machine Learning. — Morgan Kaufmann, San Francisco, CA, 2001. — Pp. 345–352.
http://citeseer.ist.psu.edu/452556.html.

[32] *Osborne M. L.* The seniority logic: A logic for a committee machine // *IEEE Trans. on Comp.* — 1977. — Vol. C-26, no. 12. — Pp. 1302–1306.

[33] *Potter M. A., De Jong K. A.* Cooperative coevolution: An architecture for evolving coadapted subcomponents // *Evolutionary Computation.* — 2000. — Vol. 8, no. 1. — Pp. 1–29.
http://citeseer.ist.psu.edu/potter00cooperative.html.

[34] *Ratsch G., Onoda T., Muller K. R.* An improvement of adaboost to avoid overfitting // Advances in Neutral Information Processing Systems, Kitakyushu, Japan. — 1998. — Pp. 506–509.
http://citeseer.ist.psu.edu/6344.html.

[35] *Ratsch G., Onoda T., Muller K.-R.* Soft margins for AdaBoost // *Machine Learning.* — 2001. — Vol. 42, no. 3. — Pp. 287–320.
http://citeseer.ist.psu.edu/ratsch00soft.html.

[36] *Rivest R. L.* Learning decision lists // *Machine Learning.* — 1987. — Vol. 2, no. 3. — Pp. 229–246.
http://citeseer.ist.psu.edu/rivest87learning.html.

[37] *Schapire R.* The boosting approach to machine learning: An overview // MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA. — 2001.
http://citeseer.ist.psu.edu/schapire02boosting.html.

[38] *Schapire R. E.* The strength of weak learnability // *Machine Learning.* — 1990. — Vol. 5. — Pp. 197–227.
http://citeseer.ist.psu.edu/schapire90strength.html.

[39] *Skurichina M., Duin R. P. W.* Limited bagging, boosting and the random subspace method for linear classifiers // *Pattern Analysis & Applications.* — 2002. — no. 5. — Pp. 121–135.
http://citeseer.ist.psu.edu/skurichina02limited.html.

[40] *Tresp V.* Committee machines // Handbook for Neural Network Signal Processing / Ed. by Y. H. Hu, J.-N. Hwang. — CRC Press, 2001.
http://citeseer.ist.psu.edu/tresp01committee.html.

[41] *Tresp V., Taniguchi M.* Combining estimators using non-constant weighting functions // Advances in Neural Information Processing Systems / Ed. by G. Tesauro, D. Touretzky, T. Leen. — Vol. 7. — The MIT Press, 1995. — Pp. 419–426.
http://citeseer.ist.psu.edu/tresp95combining.html.

[42] *Vapnik V.* Statistical Learning Theory. — Wiley, New York, 1998.

[43] *Vapnik V.* The nature of statistical learning theory. — 2 edition. — Springer-Verlag, New York, 2000.

[44] *Waterhouse S. R., Robinson A. J.* Classification using hierarchical mixtures of experts // Proceedings of the 1994 IEEE Workshop on Neural Networks for Signal Processing IV. — Long Beach, CA: IEEE Press, 1994. — Pp. 177–186.
http://citeseer.ist.psu.edu/waterhouse94classification.html.

[45] *Wolpert D. H.* Stacked generalization // *Neural Networks.* — 1992. — no. 5. — Pp. 241–259.
http://citeseer.ist.psu.edu/wolpert92stacked.html.