Содержание

1 Решающие деревья

Пусть $\mathbf{X} \in \mathbb{R}^{n \times k}$ — матрица данных, n — число индивидов, k — число признаков, $Y \in \mathbb{R}^n$ — вектор отклика, X_i — вектор признака, $i \in 1:k$.

Определение 1. Дерево решений — бинарное дерево (V, E), в котором каждой $v \in V$, не являющейся листом соответствует функция, являющаяся предикатом $\beta_v : X \to \{0, 1\}$, а каждому листу $l \in V$ соответствует прогноз $f_l : X \to Y$.

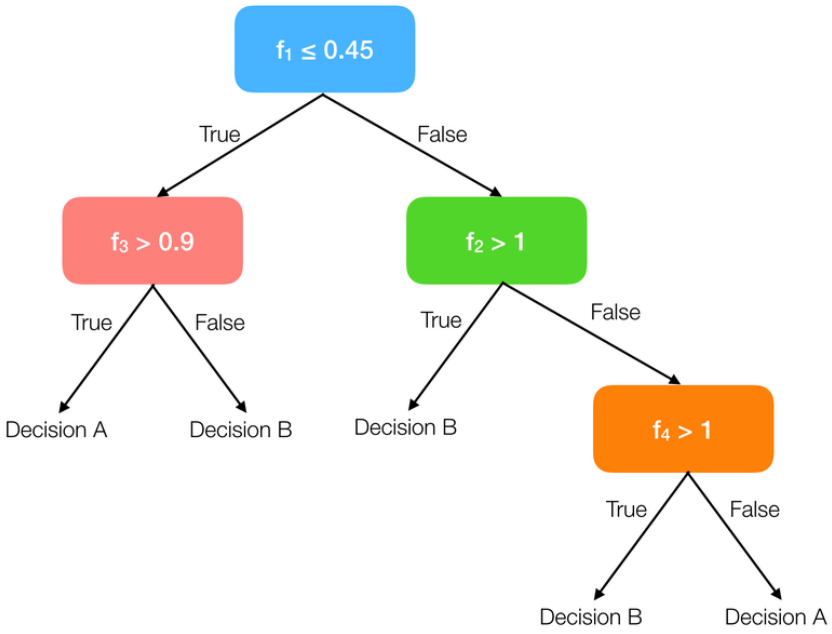

Рис. 1: Пример дерева решений, f_i — признаки

Деревья решений можно применять для классификации и регрессии.

1.1 Задача регрессии

Разобьём всевозможные значение X_i, $i \in 1:k$ на J непересекающихся множеств $\{R_j\}_{j=1}^J$, $R_j \in \mathbb{R}^k$. Предсказание для $x \in X$ равно

$$f(x) = \sum_{j=1}^{J} c_j \mathbb{1}(x \in R_j).$$

Выбирать набор R_j будем решая оптимизационную задачу минимизации

суммы квадратов остатков

$$\text{RSS} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - f(x_i))^2 \to \min_{R_1,\dots,R_J}.$$

Тогда оценка c_j равна

$$\widehat{c}_j = \frac{1}{|R_j|} \sum_{x_i \in R_j} y_i.$$

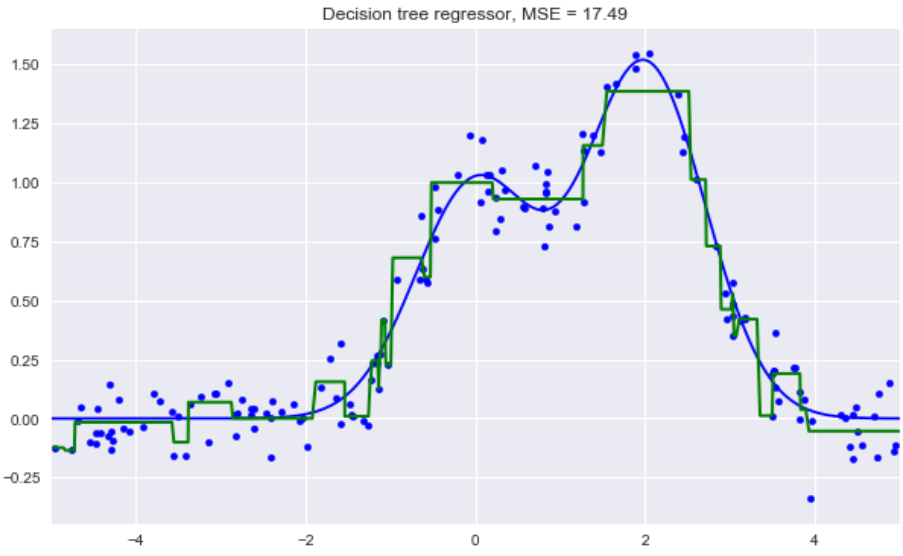

Рис. 2: Решающее дерево в задаче регрессии

1.2 Задача классификации

Обозначим за p_{jk} долю объектов класса $c \in Y$ в многомерном прямоугольнике $R_j \in \mathbb{R}^p$

$$\widehat{p}_{jc} = \frac{1}{N_j} \sum_{x_i \in R_j} \mathbb{1}(y_i = c)$$

.

$\{R_j\}_{j=1}^{J}$ — решение оптимизационной задачи

$$\text{ME} = 1 - \max_{j,c} \widehat{p}_{jc} \to \min_{R_1,\dots,R_J}.$$

Однако ME не является дифференцируемой функцией(?вроде дифференцируемость и не нужна при построении), поэтому используются другие функции:

- Индекс Джини $G(j) = \sum\limits_{k=1}^{K} \widehat{p}_{jc}(1 - \widehat{p}_{jc})$
- Кросс-энтропия $CI(j) = -\sum\limits_{k=1}^{K} \widehat{p}_{jc} \log \widehat{p}_{jc}$

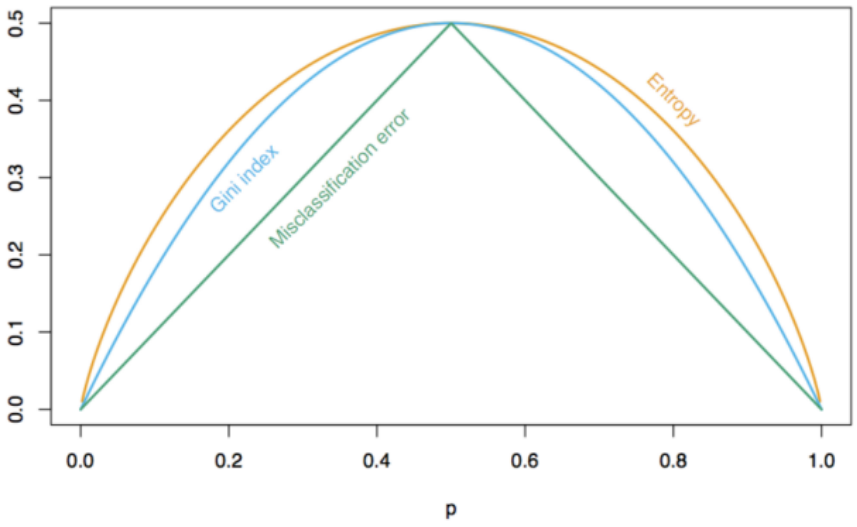

Рис. 3: Информационные индексы ME, G, CI в случае двух классов

Предсказание для объекта $x \in X$ равно

$$f(x) = \underset{c \in Y}{\operatorname{argmax}} \widehat{p}_{ck},$$

где j — индекс многомерного многоугольника, в который попадает x.

1.3 Алгоритм построения дерева

1. Выбираем признак X_j и порог s так, чтобы разбиение X^n на $R_1(j,s) = \{x \in X^n | X_j < s\}$ и $R_2(j,s) = \{x \in X^n | X_j \geqslant s\}$ решало оптимизационную задачу, соответствующую задаче классификации или регрессии:

$$\sum_{i: x_i \in R_1(j,s)} \text{error}(y_i, \hat{y}_{R_1}) + \sum_{i: x_i \in R_2(j,s)} \text{error}(y_i, \hat{y}_{R_2}) \to \min_{j,s},$$

где error — функция потерь, в случае регрессии сумма остатков, в случае классификации индекс Джини или кросс-энтропия.

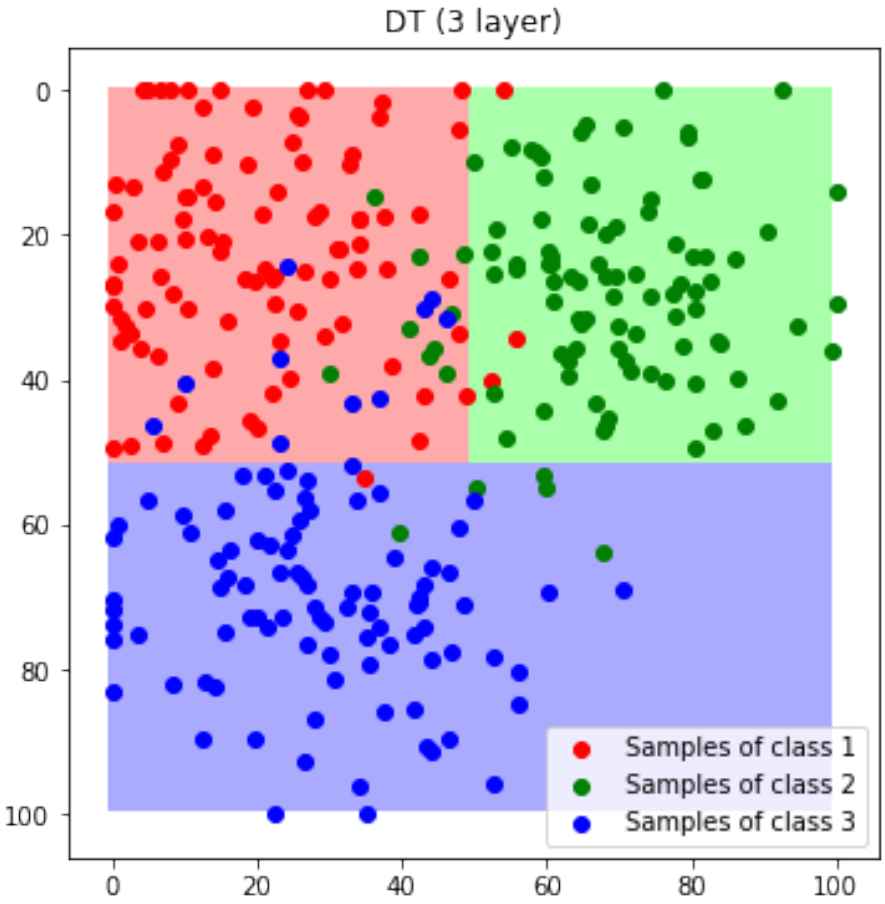

Рис. 4: Решающее дерево в задаче классификации

2. Разбиваем выборку на области R_1 и R_2, образуя две дочерние вершины.

3. Повторяем процедуру в пределах каждой получаемой области, пока не выполнится критерий остановки.

Очевидный критерий остановки — остановка в случае если все индивиды в листе принадлежат одному классу. Дополнительно можно задать следующие критерии остановки:

- Ограничение максимальной глубины дерева
- Ограничение минимального числа объектов в поддереве
- Ограничение максимального числа индивидов в листе

1.4 Стрижка деревьев

Очевидно, что можно построить дерево решений с нулевой ошибкой на тренировочной выборке. Достаточно разбить $\mathbb{R}^p$ на n областей R_j, $j \in 1:n$, получится глубокое дерево. Однако в таком случае ошибка на тестовой выборке может быть большой.

Для устранения такой проблемы деревья "стригут"(prune).

Пусть T_0 — большое построенное дерево. Можно в качестве T_0 взять дерево с нулевой ошибкой на тренировочной выборке. Рассмотрим поддерево $T \subset T_0$, в качестве оптимизируемой функции возьмём

$$Q_\alpha(T) = \text{error}(T) + \alpha|l(T)|,$$

где $\text{error}(T)$ — ошибка на тренировочной выборке, $|l(T)|$ — количество листьев в дереве, α — положительный гиперпараметр. α подбирается с помощью кросс-валидации. После подбора α строится поддерево T, на котором достигается минимум $Q_\alpha(T)$.

1.5 Обработка категориальных данных

Можно разбить вершину по категориям, однако такой подход может давать слишком глубокие деревья. Есть другой способ обработки категориальных значений, пусть категориальный признак имеет M значений $\mathbb{S} = \{s_i\}_{i=1}^M$, будем рассматривать всевозможные разбиения $\mathbb{S}$ на 2 дизъюнктных подмножества $\mathbb{S} = V \cup W$, $V \cap W = \emptyset$. В таком случае предикат в вершине есть $\mathbb{1}(x \in V)$. Проблема такого подхода заключается в том, что нужно перебрать $2^{M-1} - 1$ разбиений, но есть алгоритм, позволяющий избавиться от перебора всех множеств и дающий результат, совпадающий с результатом полного перебора.

1.6 Обработка пропусков

Есть несколько способов обработки пропусков:

- Убрать индивидов с пропусками (pairwise). Однако после выкидывания всех пропусков, размер выборки может сильно уменьшиться.
- На стадии выбора разбиения пропуски не учитываются и значения с пропусками спускаются в одно поддерево
- Не учитывать на шаге поиска разбиения пропуски, но спустить индивидов в оба поддерева с весами

1.7 Преимущества и недостатки

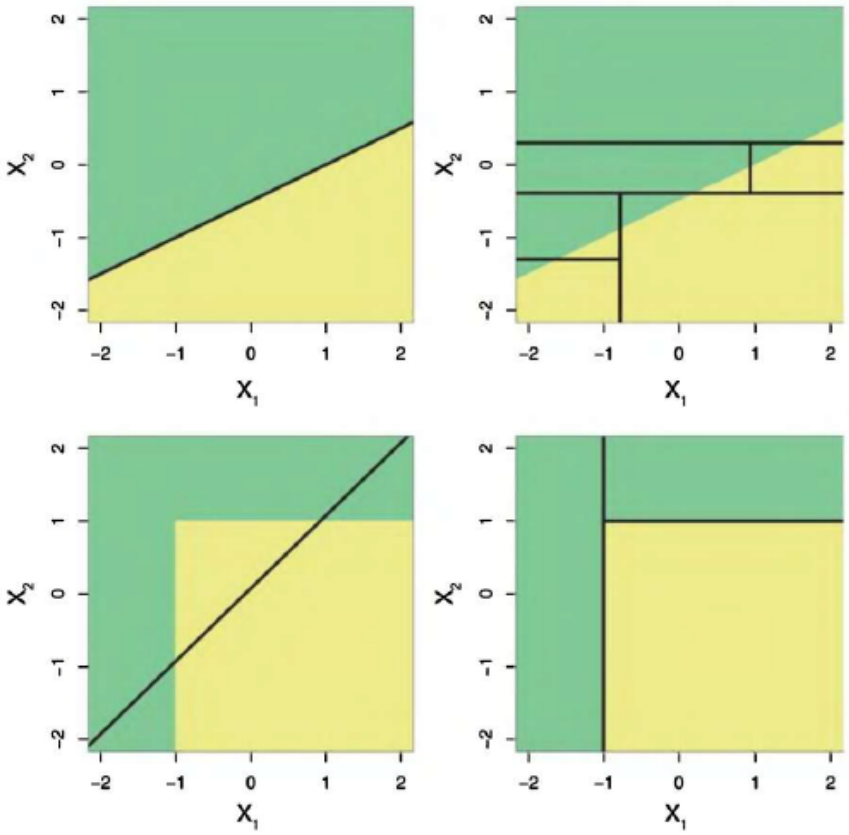

Рис. 5: Примеры решений задач классификации с линейной (верхний ряд) и нелинейной (нижний ряд) зависимостью. В левой части решение с помощью линейной модели, в правой — с помощью решающего дерева.

Преимущества:

- Простота интерпретации
- Пригодность и для задач регрессии, и для задач классификации
- Возможность работы с пропусками в данных

Недостатки:

- Основан на «жадном» алгоритме (решение является лишь локально оптимальным в каждом узле и может быть неоптимальным для всего дерева)
- Метод явялется неустойчивым и склонным к переобучению

2 Введение

Решающие деревья могут восстанавливать очень сложные закономерности, но при этом неустойчивы к малейшим изменениям в данных. Из-за этого сами по себе деревья не очень хороши, но при этом, как оказывается, при объединении в композицию они показывают очень хорошие результаты. Одним из подходов к построению композиций является бэггинг, который независимо строит несколько моделей и усредняет их ответы. Мы изучим инструмент, который поможет нам в анализе бэггинга — декомпозицию ошибки на компоненты смещения и разброса (bias-variance decomposition) — а затем перейдем к самим методам. Также существует другой подход к построению композиций, называемый бустингом, который строит модели последовательно, и каждая следующая модель исправляет ошибки предыдущей.

3 Бутстрап

Пусть дана конечная выборка $X = (x_i, y_i)$ с вещевственными ответами. Сгенерируем подвыборку с помощью бутстрапа. Равномерно возьмем из выборки l объектов с возвращением. Отметим, что из-за возвращения среди них окажутся повторы. Обозначим новую выборку через X_1. Повторим процедуру N раз и получим подвыборки $X_1, \ldots, X_N$. Обучим по каждой выборке модель линейной регрессии и получим базовые алгоритмы $b_1(x), \ldots, b_N(x)$.

Предположим, что существует истинная функция ответов для всех объектов $y(x)$ и задано распределение на объектах $p(x)$. Тогда мы можем записать ошибку каждой функции регрессии, как

$$\epsilon_j(x) = b_j(x) - y(x), j = 1, ..., N, \tag{1}$$

и записать матожидание среднеквадратичной ошибки:

$$\mathbb{E}_x \left(b_j(x) - y(x)\right)^2 = \mathbb{E}_x \epsilon_j^2(x) \tag{2}$$

Средняя ошибка построенных функций регрессии имеет вид
$E_1 = \frac{1}{N} \sum_{j=1}^{N} \mathbb{E}_x \epsilon_j^2(x)$
Предположим, что ошибки несмещены и некоррелированы:
$\mathbb{E}_x \epsilon_j(x) = 0$
$\mathbb{E}_x \epsilon_i(x)\epsilon_j(x) = 0,\ i \neq j$
Построим новую функцию регрессии, которая будет усреднять ответы построенных нами функций:
$a(x) = \frac{1}{N} \sum_{j=1}^{N} b_j(x)$
Ее среднеквадратичная ошибка:
$E_N = \mathbb{E}_x \left(\frac{1}{N} \sum_{j=1}^{N} b_j(x) - y(x)\right)^2 = \mathbb{E}_x \left(\frac{1}{N} \sum_{j=1}^{N} \epsilon_j(x)\right)^2 =$
$= \frac{1}{N^2} \mathbb{E}_x \left(\sum_{j=1}^{N} \epsilon_j^2(x) + \sum_{i \neq j} \epsilon_i(x)\epsilon_j(x)\right) = \frac{1}{N} E_1$

Усреднение ответов позволило уменьшит средний квадрат ошибки в N раз.

Это является идеальным случаем, так как на практике некоррелируемость ошибок редко случается.

4 Bias-Variance decomposition

Ошибка любой модели складывается из трех факторов: сложности самой выборки, сходства модели с истинной зависимостью ответов от объектов в выборке, и богатства семейства, из которого выбирается конкретная модель. Между этими факторами существует некоторый баланс, и уменьшение одного из них приводит к увеличению другого. Такое разложение ошибки носит название разложения на смещение и разброс.

Пусть задана выборка $X = (x_i, y_i)_{i=1}^l$ с вещественными ответами $y_i \in \mathbb{R}$. Будем считать что на пространстве $\mathbb{X} \times \mathbb{Y}$ существует распределение $p(x, y)$, из которого сгенерирована выборка X и все ответы на ней.

Рассмотрим квадратичную функцию потерь $L(y, a) = (y - a(x))^2$ и ее среднеквадратичный риск

$R(a) = \mathbb{E}_{x,y}\left[(y - a(x))^2\right] \int_{\mathbb{X}} \int_{\mathbb{Y}} p(x, y)(y - a(x))^2 dxdy$

Такой функционал усредняет ошибку модели в каждой точке пространства x и для каждого возможного ответа y, причём вклад пары (x, y), по сути, пропорционален вероятности получить её в выборке $p(x, y)$. Разумеется, на практике мы не можем вычислить данный функционал, поскольку распределение $p(x, y)$ неизвестно. Тем не менее, в теории он позволяет измерить качество модели на всех возможных объектах, а не только на обучающей выборке.

4.1 Минимум среднеквадратичного риска

???

4.2 Ошибка метода обучения

Для того, чтобы построить идеальную функцию регрессии, необходимо знать распределение на объектах и ответах $p(x, y)$, что, как правило, невозможно. На практике вместо этого выбирается некоторый метод обучения $\mu : (\mathbb{X} \times \mathbb{Y})^l \to \mathbf{A}$, который произвольной обучающей выборке ставит в соответствие некоторый алгоритм из семейства $\mathbf{A}$. В качестве меры качества метода обучения можно взять усредненный по всем выборкам среднеквадратичный риск алгоритма, выбранного методом μ по ввыборке:

$$L(\mu) = \mathbb{E}_X \left[\mathbb{E}_{x,y} \left[(y - \mu(X)(x)))^2 \right] \right] =$$

$$= \int_{(\mathbb{X}\times\mathbb{Y})^l} \int_{\mathbb{X}\times\mathbb{Y}} (y - \mu(X)(x))^2 \, p(x,y) \prod_{i=1}^{l} p(x_i, y_i) dx dy dx_1 dy_1, \dots dx_l dy_l \quad (3)$$

Здесь матожидание $\mathbb{E}_X[\cdot]$ берется по всем возможным выборкам $(x_1, y_1), ..., (x_l, y_l)$ из распределения $\prod_{i=1}^{l} p(x_i, y_i)$.

Среднеквадратичный риск на фиксированной выборке X можно расисать как

$\mathbb{E}_{x,y} = \left[(y - \mu(X))^2 \right] = \mathbb{E}_{x,y} \left[(y - \mathbb{E}[y|x])^2 \right] + \mathbb{E}_{x,y} \left[(\mathbb{E}[y|x] - \mu(X))^2 \right]$

Подставим эту формулу в (3).

$$L(\mu) = \mathbb{E}_X \left[\underbrace{\mathbb{E}_{x,y} \left[(y - \mathbb{E}[y|x])^2 \right]}_{\text{не завсисит от X}} + \mathbb{E}_{x,y} \left[(\mathbb{E}[y|x] - \mu(X))^2 \right] \right] = \quad (4)$$
$$= \mathbb{E}_{x,y} \left[(y - \mathbb{E}[y|x])^2 \right] + \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y|x] - \mu(X))^2 \right] \right]$$

Преобразовываем второе слагаемое:

$$\mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y|x] - \mu(X))^2 \right] \right] =$$
$$= \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y|x] - \mathbb{E}_X[\mu(X)] + \mathbb{E}_X[\mu(X)] - \mu(X))^2 \right] \right] =$$
$$= \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[\underbrace{(\mathbb{E}[y|x] - \mathbb{E}_X \mu(X))^2}_{\text{не завсисит от X}} \right] \right] + \mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}_X \mu(X) - \mu(X))^2 \right] \right] +$$
$$+ 2\mathbb{E}_{x,y} \left[\mathbb{E}_X \left[(\mathbb{E}[y|x] - \mathbb{E}[\mu(X)])(\mathbb{E}_X[\mu(X)] - \mu(X)) \right] \right] \quad (5)$$

Последнее слагаемое обращается в ноль.

Подставим (5) в 4.

$$L(\mu) = \underbrace{\mathbb{E}_{x,y} \left[(y - \mathbb{E}[y|x])^2 \right]}_{\text{шум}} +$$
$$+ \underbrace{\mathbb{E}_x \left[(\mathbb{E}_X[\mu(X)] - \mathbb{E}[y|x])^2 \right]}_{\text{смещение}} + \underbrace{\mathbb{E}_x \left[\mathbb{E}_X \left[(\mu(X) - \mathbb{E}_X[\mu(X)])^2 \right] \right]}_{\text{разброс}} \quad (6)$$

Первая компонента характеризует шум в данных и равна ошибке идеального алгоритма. Невозможно построить алгоритм, имеющий меньшую среднеквадратичную ошибку. Вторая компонента характеризует смещение (bias) метода обучения, то есть отклонение среднего ответа обученного алгоритма от ответа идеального алгоритма. Третья компонента характеризует дисперсию (variance), то есть разброс ответов обученных алгоритмов относительно среднего ответа.

Смещение показывает, насколько хорошо с помощью данных метода обучения и семейства алгоритмов можно приблизить оптимальный алгоритм. Как правило, смещение маленькое у сложных семейств (например, у деревьев) и большое у простых семейств (например, линейных классификаторов). Дисперсия показывает, насколько сильно может изменяться ответ обученного алгоритма в зависимости от выборки — иными словами, она характеризует чувствительность метода обучения к изменениям в выборке. Как правило, простые семейства имеют маленькую дисперсию, а сложные семейства — большую дисперсию.

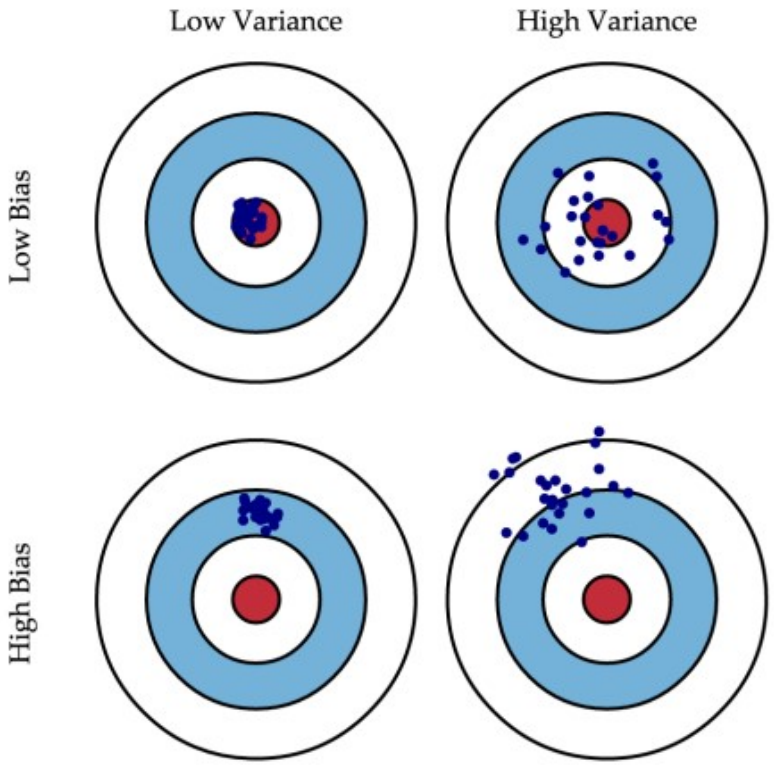

Рис. 6: Сдвиг и разброс разных моделей

На рис. 6 изображены модели с различными сдвигом и разбросом. Модели изображены синими точками, одна точка соответствует модели, обученной по одной из возможных обучающих выборок. Каждый круг характеризует качество модели — чем ближе точка к центру, тем меньше ошибок на контрольной выборке достигает данный алгоритм. Видно, что большой сдвиг соответствует тому, что в среднем точки не попадают в центр, то есть

в среднем они не соответствуют лучшей модели.

Большой разброс означает, что модель может попасть по качеству куда угодно — как в центр, так и в область с большой ошибкой.

5 Bagging

Деревья решений, страдают от высокой дисперсии. Это означает, что если разделить обучающие данные на две части случайным образом и применить дерево решений к обеим половинам, результаты, которые мы получим, могут быть совершенно разными. Напротив, процедура с низкой дисперсией даст схожие результаты при многократном применении к разным наборам данных; линейная регрессия имеет тенденцию к низкой дисперсии, если отношение n к p умеренно велико. Бэггинг, — это процедура для уменьшения дисперсии статистического метода обучения; она особенно полезна и часто используется в контексте деревьев решений.

Пусть у нас имеется набор из n независивых наблюдений $X_1, \dots X_n$, каждый из которых имеет дисперсию σ^2, дисперсия $\bar{X}$ будет равна σ^2/n. Другими словами, усреднение набора наблюдений снижает дисперсию.

Следовательно, естественный способ уменьшить дисперсию и, следовательно, увеличить точность предсказания статистического метода обучения заключается в том, чтобы взять множество обучающих наборов из выборки, построить отдельную модель предсказания, используя каждый набор обучения и усреднить полученные прогнозы.

$$a_N(x) = \frac{1}{N} \sum_{n=1}^{N} \hat{\mu}(x) \tag{7}$$

Однако на практике это не реализовать, потому что у нас нет нескольких сетов обучающих данных. Вместо этого, мы можем примеить бутстрап беря несколько сэмплов из обучающего набора данных. При таком подходе, мы генерируем B разных подвыборок с помощью бутсрапа. Затем мы обучаем модель на b-ом сете обучающих данных и усредняем прогнозы.

Это называется бэггингом.

Заметим, что в методе обучения для бэггинга появляется ещё один источник случайности — взятие подвыборки. Чтобы функционал качества $L(\mu)$ был детерминированным, мы будем далее считать, что матожидание$\mathbb{E}_X[\cdot]$ берётся не только по всем обучающим выборкам X, но ещё и по всем возможным подвыборкам $\hat{X}$, получаемым с помощью бутстрапа. Это вполне логичное обобщение, поскольку данное матожидание вводится в функционал именно для учёта случайностей, связанных с процедурой обучения модели.

Найдём смещение из разложения (6) для бэггинга:

$$\mathbb{E}_{x,y}\left[\left(\mathbb{E}_X\left[\frac{1}{N}\sum_{b=1}^{N}\hat{\mu}(X)(x)\right]-\mathbb{E}[y|x]\right)^2\right]=\mathbb{E}_{x,y}\left[\left(\frac{1}{N}\sum_{b=1}^{N}\mathbb{E}_X[\hat{\mu}(X)(x)]-\mathbb{E}[y|x]\right)^2\right]=$$
$$=\mathbb{E}_{x,y}\left[\left(\mathbb{E}_X\left[\hat{\mu}(X)(x)\right]-\mathbb{E}[y|x]\right)^2\right] \quad (8)$$

Мы получили, что смещение композиции, полученной с помощью бэггинга, совпадает со смещением одного базового алгоритма. Таким образом, бэггинг не ухудшает смещенность модели.

Теперь рассмотрим разброс. Из (6) он равен:

$$\frac{1}{N}\mathbb{E}_{x,y}\left[\mathbb{E}_X\left[\left(\hat{\mu}(X)(x)-\mathbb{E}_X[\hat{\mu}(X)(x)]\right)^2\right]\right]+$$
$$+\frac{N(N-1)}{N^2}\mathbb{E}_{x,y}\left[\mathbb{E}_X\left[\left(\hat{\mu}(X)(x)-\mathbb{E}_X[\hat{\mu}(X)(x)]\right)\times\left(\hat{\mu}(X)(x)-\mathbb{E}_X[\hat{\mu}(X)(x)]\right)\right]\right] \quad (9)$$

Первое слагаемое — это дисперсия одного базового алгоритма, деленная на длину композиции N. Второе — ковариация между двумя базовыми алгоритмами. Мы видим, что если базовые алгоритмы некоррелированы, то дисперсия композиции в N раз меньше дисперсии отдельных алгоритмов. Если же корреляция имеет место, то уменьшение дисперсии может быть гораздо менее существенным.

До сих пор мы описывали бэггинг в контексте регрессии, для предсказания количественного результата y. Как можно распространить бэггинг на задачу классификации, где Y является качественным? В этой ситуации существует существует несколько возможных подходов, но самый простой заключается в следующем. Для тестового сета, мы можем записать класс, предсказанный каждым из N деревьев, и сделать "*majority vote*— это наиболее часто встречающийся класс среди N предсказаний.

5.1 Out-of-Bag Error Estimation

Ключевой момент bagging модели заключается в том, что деревья многократно подгоняются к бутстреп-подмножествам наблюдений. В среднем каждое дерево сетки использует около двух третей наблюдений. Оставшаяся треть наблюдений, не использованная для подгонки данного дерева сетки, называется *out-of-bag* наблюдениями (OOB).

Мы можем предсказать ответ для i-го наблюдения, используя каждое из деревьев, в которых это наблюдение было OOB. Это даст около N/3 предсказаний для i-го наблюдения. Для того чтобы получить единое предсказание для i-го наблюдения, мы можем усреднить эти предсказанные ответы (при регрессионной модели) или взять majority vote (при классификации). Это приводит к единственному OOB-предсказанию для каждого наблюдения. Предсказание OOB может быть получено таким образом для каждого

из n наблюдений, из чего следует, что общий MSE OOB (для задачи регрессии) или ошибка классификации (для задачи классификации) может быть вычислена. Полученная ошибка OOB является достоверной оценкой тестовой ошибки для bagging модели, поскольку прогноз для каждого наблюдения предсказывается с использованием только тех деревьев, которые не подошли для данного наблюдения.

$$OOB = \sum_{i=1}^{l} L\left(y_i, \frac{1}{\sum_{n=1}^{N}\left[x_i \notin X_n\right]} \sum_{n=1}^{N} [x_i \notin X_n] b_n(x_i)\right)$$

5.2 Значимость переменных

Как говорилось ранее, бэггинг обычно приводит к повышению точности по сравнению с предсказанием с помощью одного дерева. Однако, к сожалению, интерпретировать полученную модель может быть сложно. Напомним, что одним из преимуществ деревьев решений является привлекательная и легко интерпретируемая диаграмма, которая получается в результате. Однако, когда мы объединяем большое количество деревьев, уже невозможно представить результирующую процедуру статистического обучения с помощью одного дерева, и уже не ясно, какие переменные являются наиболее важными для процедуры. Таким образом, группировка деревьев повышает точность прогнозирования точность прогноза за счет интерпретируемости.

Хотя набор деревьев построенных с помощью бэггинга гораздо сложнее интерпретировать чем отдельное дерево, можно получить общую сводку важности каждого предиктора, используя RSS (для деревьев регрессии) или индекс Джини (для деревьев классификации). В случае деревьев регрессии, мы можем записать общую сумму, на которую RSS уменьшается из-за разделяется для данного предиктора, усредненное по всем B деревьям. Большое значение указывает на важный предиктор. Аналогично, в контексте классификации деревьев, мы можем сложить общую сумму, на которую индекс Джини уменьшается в результате разделения по заданному предиктору, в среднем по всем B-деревьям.

6 Random Forest

Бэггинг позволяет объединить несмещенные, но чувствительные к обучающей выборке алгоритмы в несмещенную композицию с низкой дисперсией. Хорошим семейством базовых алгоритмов здесь являются решающие деревья — они достаточно сложны и могут достигать нулевой ошибки на любой выборке (следовательно, имеют низкое смещение), но в то же время легко переобучаются.

Метод случайных лесов основан на бэггинге над решающими деревьями.

Алгоритм: Ддля $n = 1, \ldots, N$

1. Сгенерировать выборку $\hat{X}_n$ с помощью бутстрапа.
2. Построить решающее дерево $b_n(x)$ по выборке $\hat{X}_n$.

- дерево строится, пока в каждом листе не окажется не более n_{min} объектов
- при каждом разбиении сначала выбирается m случайных признаков из p и оптимальное разделение ищется только среди них

3. Вернуть композицию $a_N(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x)$

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

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

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

7 Boosting

Ранее мы изучили бэггинг и случайные леса — подходы к построению композиций, которые независимо обучают каждый базовый алгоритм по некоторому подмножеству обучающих данных. При этом возникает ощущение, что мы используем возможности объединения алгоритмов не в полную силу, и можно было бы строить их так, чтобы каждая следующая модель исправляла ошибки предыдущих. Ниже мы рассмотрим метод, который реализует эту идею — градиентный бустинг. Он работает для любых дифференцируемых функций потерь и является одним из наиболее мощных и универсальных на сегодняшний день.

7.1 Бустинг в задаче регрессии

Рассмотрим задачу минимизации квадратичного функционала:

$\frac{1}{2}\sum_{i=1}^{l}(a(x_i) - y_i)^2 \to \min\limits_{a}$

Будем искать итоговый алгоритм в виде суммы базовых моделей $b_n(x)$:

$a_N(x) = \sum_{n=1}^{N} b_n(x)$, где базовые алгоритмы b_n принадлежат некоторому семейству $\mathbf{A}$.

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

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

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

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

Таким образом, каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:

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

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

Также, остатки могут быть найдены как антиградиент функции потерь по ответу модели, посчитанный в точке ответа уже построенной композиции:

$s_i^{(N)} = y_i - a_{N-1}(x_i) = -\frac{\partial}{\partial z} \frac{1}{2} (z - y_i)^2 \Big|_{z = a_{N-1}(x_i)}$

Получается, что выбирается такой базовый алгоритм, который как можно сильнее уменьшит ошибку композиции — это свойство вытекает из его близости к антиградиенту функционала на обучающей выборке.

7.2 Градиентный бустинг

Пусть дана некоторая дифференцируемая функция потерь $L(y, z)$. Будем строить взвешенную сумму базовых алгоритмов:

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

Заметим, что в композиции имеется начальный алгоритм $b_0(x)$. Как правило, коэффициент γ_0 при нем берут равным единице, а сам алгоритм выбирают очень простым.

Примеры выбора алгоритма $b_0(x)$:

1. Нулевой: $b_0(x) = 0$.
2. Возвращающий самый популярный класс (в задачах классификации):
   $b_0(x) = \underset{y \in Y}{argmax} \sum_{i=1}^{l} [y_i = y]$
3. Возвращающий средний ответ (в задачах регрессии): $b_0(x) = \frac{1}{l} \sum_{i=1} l y_i$

Допустим, мы построили композицию $a_{N-1}(x)$ из $N - 1$ алгоритма, и хотим выбрать следующий абзовый алгоритм $b_N(x)$ так, чтобы как можно сильнее уменьшить ошибку:

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

Ответим в первую очередь на следующий вопрос: если бы в качестве алгоритма $b_N(x)$ мы могли выбрать совершенно любую функцию, то какие значения ей следовало бы принимать на объектах обучающей выборки? Иными словами, нам нужно понять, какие числа $s_1, \ldots, s_l$ надо выбрать для решения следующей задачи:

$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + s_i\right) \to \min\limits_{s_1,\ldots,s_l}$

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

Другой вариант — потребовать чтобы сдвиг s_i был противоположен производной функции потерь в точке $z = a_{N-1}(x_i)$:

$s_i = -\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}$

В таком случае мы сдвинемся в сторону скорейшего убывания функции потерь. Заметим, что вектор сдвигов $s = s_1, \ldots, s_l$ совпадает с антиградиентом:

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

При таком выборе сдвигов s_i мы, по сути, сделаем один шаг градиентного спуска, двигаясь в сторону наискорейшего убывания ошибки на обучающей выборке. Отметим, что речь идет о градиентном спуске в l-мерном пространстве предсказаний алгоритма на объектах обучающей выборки. Поскольку вектор сдвига будет свой на каждой итерации, правильнее обозначать его как $s_i^{(N)}$, но для простоты будем иногда опускать верхний индекс.

Итак, мы поняли, какие значения новый алгоритм должен принимать на объектах обучающей выборки. По данным значениям в конечном числе точек необходимо построить функцию, заданную на всем пространстве объектов. Это классическая задача обучения с учителем, которую мы уже хорошо умеем решать. Один из самых простых функционалов — среднеквадратичная ошибка. Воспользуемся им для поиска базового алгоритма, приближающего градиент функции потерь на обучающей выборке:

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

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

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

Описанный подход с аппроксимацией антиградиента базовыми алгоритмами и называется градиентным бустингом. Данный метод представляет собой поиск лучшей функции, восстанавливающей истинную зависимость ответов от объектов, в пространстве всех возможных функций. Ищем мы данную функцию с помощью «псевдоградиентного» спуска — каждый шаг делается вдоль направления, задаваемого некоторым базовым алгоритмом. При этом сам базовый алгоритм выбирается так, чтобы как можно лучше приближать антиградиент ошибки на обучающей выборке.

7.3 Регуляризация

7.3.1 Сокращение шага

На практике оказывается, что градиентный бустинг очень быстро строит композицию, ошибка которой на обучении выходит на асимптоту, после чего начинает настраиваться на шум и переобучаться. Это явление можно объяснить одной из двух причин:

- Если базовые алгоритмы очень простые (например, решающие деревья небольшой глубины), то они плохо приближают вектор антиградиента. По сути, добавление такого базового алгоритма будет соответствовать шагу вдоль направления, сильно отличающегося от направления наискорейшего убывания. Соответственно, градиентный бустинг может свестись к случайному блужданию в пространстве
- Если базовые алгоритмы сложные (например, глубокие решающие деревья), то они способны за несколько шагов бустинга идеально подогнаться под обучающую выборку — что, очевидно, будет являться переобучением, связанным с излишней сложностью семейства алгоритмов.

Сокращение шага представляет собой следующее: вместо перехода в оптимальную точку в направлении антиградиента делается укороченный шаг

$a_N(x) = a_{N-1}(x) + \eta\gamma_N b_N(x)$,

где $\eta \in (0, 1]$ — темп обучения. Как правило, чем меньше темп обучения, тем лучше качество итоговой композиции. Сокращение шага, по сути, позволяет понизить доверие к направлению, восстановленному базовым алгоритмом.

Также следует обратить внимание на число итераций градиентного бустинга. Хотя ошибка на обучении монотонно стремится к нулю, ошибка на контроле, как правило, начинает увеличиваться после определенной итерации. Оптимальное число итераций можно выбирать, например, по отложенной выборке или с помощью кроссвалидации.

7.3.2 Стохастический градиентный бустинг

Еще одним способом улучшения качества градиентного бустинга является внесение рандомизации в процесс обучения базовых алгоритмов. А именно, алгоритм b_N обучается не по всей выборке X, а лишь по ее случайному подмножеству $X^k \subset X$. В этом случае понижается уровень шума в обучении, а также повышается эффективность вычислений. Существует рекомендация брать подвыборки, размер которых вдвое меньше исходной выборки.

7.4 Функции потерь

7.4.1 Регрессия

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

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

7.4.2 Классификация

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

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

Задача поиска базового алгоритма с ней принимает вид

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

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

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

Если отступ $y_i a_{N-1}(x_i)$ на i-м объекте большой положительный, то данный объект не будет вносить практически никакого вклада в ошибку, и может быть исключен из всех вычислений на текущей итерации без потерь.

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

7.5 Градиентный бустинг над деревьями

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

Как известно, решающее дерево разбивает все пространство на непересекающиеся области, в каждой из которых его ответ равен константе $b_n(x) = \sum_{j=1}^{J_n} b_{nj}[x \in R_j]$, где $j = 1, \ldots, J_n$ — индексы листьев, R_j — соответствующие области разбиения, b_{nj} — значения в листьях. Значит в N-й итерации бустинга композиция обновляется как

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

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

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

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

$\gamma_{Nj} = \underset{\gamma}{argmin} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma), \qquad j = 1, \ldots J_N$

В некоторых случаях оптимальные коэффициенты могут быть найдены аналитически — например, для квадратичной и абсолютной ошибки.

Рассмотрим теперь логистическую функцию потерь. В этом случае нужно решить задачу $F_j^{(N)}(\gamma) = \sum_{x_i \in R_j} log(1 + exp(-y_i(a_{N-1}(x_i) + \gamma))) \to \min_{\gamma}$.

Данная задача может быть решена лишь с помощью итерационных методов, аналитической записи для оптимального γ не существует. Однако на практике обычно нет необходимости искать точное решение — оказывается достаточным сделать лишь один шаг метода Ньютона-Рафсона из начального приближения $\gamma_{Nj} = 0$.

7.5.1 Смещение и разброс

В случайных лесах используются глубокие деревья, поскольку от базовых алгоритмов требуется низкое смещение; разброс же устраняется за счёт усреднения ответов различных деревьев. Бустинг работает несколько иначе — в нём каждый следующий алгоритм целенаправленно понижает ошибку композиции, и даже при использовании простейших базовых моделей композиция может оказаться достаточно сложной. Более того, итоговая композиция вполне может оказаться переобученной при большом количестве базовых моделей. Это означает, что благодаря бустингу можно понизить смещение моделей, а разброс либо останется таким же, либо увеличится. Из-за этого, как правило, в бустинге используются неглубокие решающие деревья (3-6 уровней), которые обладают большим смещением, но не склонны к переобучению.

7.6 Взвешивание объектов

Одним из первых широко распространённых методов построения композиций является AdaBoost, в котором оптимизируется экспоненциальная функция потерь $L(y, z) = exp(-yz)$. Благодаря её свойствам удаётся свести задачу поиска базового алгоритма к минимизации доли неверных ответов с весами при объектах. Эти веса возникают и в градиентном бустинге при использовании экспоненциальной функции потерь:

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

Компоненты ее антиградиента после $N-1$ итерации:

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

Заметим, что антиградиент представляет собой ответ на объекте, умноженный на его вес. Если все веса будут равны единице, то следующий базовый классификатор будет просто настраиваться на исходный целевой вектор $(y_i)_{i=1}^{l}$;штраф за выдачу ответа, противоположного правильному, будет равен 4 (поскольку при настройке базового алгоритма используется квадратичная функция потерь). Если же какой-либо объект будет иметь большой

отступ, то его вес окажется близким к нулю, и штраф за выдачу любого ответа будет равен 1.

7.7 Влияние шума на обучение

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

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

Заметим, что если отступ на объекте большой и отрицательный (что обычно наблюдается на шумовых объектах), то вес становится очень большим, причем он никак не ограничен сверху. В результате базовый классификатор будет настраиваться исключительно на шумовые объекты, что может привести к неустойчивости его ответов и переобучению. Рассмотрим теперь логистическую функцию потерь, которая также может использоваться в задачах классификации:

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

Ее антиградиент после $N-1$ шага:

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

Теперь веса ограничены сверху единицей. Если отступ на объекте большой отрицательный (то есть это выброс), то вес при нем будет близок к единице; если же отступ на объекте близок к нулю (то есть это объект, на котором классификация неуверенная, и нужно ее усиливать), то вес при нем будет примерно равен $1/2$. Таким образом, вес при шумовом объекте будет всего в два раза больше, чем вес при нормальных объектах, что не должно сильно повлиять на процесс обучения.

8 XGBoost

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

8.1 Градиентный бустинг. Альтернативный подход

Из раздела 7.2 имеем вектор сдвигов:

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

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

$$b_N(x) = \underset{b \in A}{argmin} \sum_{i=1}^{l} (b(x_i) - s_i)^2 \tag{10}$$

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

Мы хотим найти алгоритм $b(x)$, решающий следующую задачу:

$$\sum_{i=1}^{l} L\left(y_i, a_{N-1}(x_i) + b(x_i)\right) \to \min_b$$

Разложим функцию L в каждом слагаемом в ряд Тейлора до второго члена с центром в ответе композиции $a_{N-1}(x_i)$:

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

,

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

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

$$\sum_{i=1}^{l} \left(-s_i b(x_i) + \frac{1}{2} h_i b^2(x_i)\right) \tag{11}$$

Покажем, что он очень похож на среднеквадратичный из формулы 10. Преобразуем его:

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

Видно, что последняя формула совпадает с 11 с точностью до константы, если положить $h_i = 1$. Таким образом, в обычном градиентном бустинге мы используем аппроксимацию второго порядка при обучении очередного базового алгоритма, и при этом отбрасываем информацию о вторых производных (то есть считаем, что функция имеет одинаковую кривизну по всем направлениям).

8.2 Регуляризация

Будем далее работать с функционалом 11. Он измеряет лишь ошибку композиции после добавления нового алгоритма, никак при этом не штрафуя за излишнюю сложность этого алгоритма. Ранее мы решали проблему переобучения путем ограничения глубины деревьев, но можно подойти к вопросу и более гибко. Мы выясняли, что дерево $b(x)$ можно описать формулой

$$b(x) = \sum_{j=1}^{J} b_j\,[x \in R_j]$$

Его сложность зависит от двух показателей:

1. Число листьев J. Чем больше листьев имеет дерево, тем сложнее его разделяющая поверхность, тем больше у него параметров и тем выше риск переобучения.
2. Норма коэффициентов в листьях $||b||_2^2 = \sum_{j=1}^{J} b_j^2$. Чем сильнее коэффициенты отличаются от нуля, тем сильнее данный базовый алгоритм будет влиять на итоговый ответ композиции.

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

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

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

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

Каждое слагаемое здесь можно минимизировать по b_j независимо. Заметим, что отдельное слагаемое представляет собой параболу относительно b_j, благодаря чему можно аналитически найти оптимальные коэффициенты в листьях:

$b_j = \frac{S_j}{H_j + \lambda}$

Подставляя данное выражение обратно в функционал, получаем, что ошибка дерева с оптимальными коэффициентами в листьях вычисляется по формуле

$$H(b) = -\frac{1}{2}\sum_{j=1}^{J}\frac{S_j^2}{H_j + \lambda} + \gamma J \tag{12}$$

8.3 Обучение решающего дерева

Мы получили функционал $H(b)$, который для заданной структуры дерева вычисляет минимальное значение ошибки (11), которую можно получить путем подбора коэффициентов в листьях. Заметим, что он прекрасно подходит на роль критерия информативности — с его помощью можно принимать решение, какое разбиение вершины является наилучшим! Значит, с его помощью мы можем строить дерево. Будем выбирать разбиение $[x_j < t]$ в вершине R так, чтобы оно решало следующую задачу максимизации:

$Q = H(R) - H(R_l) - H(R_r) \to max$

где информативность вычисляется по формуле

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

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

8.4 Заключение

Итак, градиентный бустинг в XGBoost имеет ряд важных особенностей.

1. Базовый алгоритм приближает направление, посчитанное с учетом вторых производных функции потерь.
2. Функционал регуляризуется — добавляются штрафы за количество листьев и за норму коэффициентов.
3. При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига.
4. Критерий останова при обучении дерева также зависит от оптимального сдвига.