Урок 7

Решающие деревья

7.1. Решающие деревья

Решающие деревья — это семейство алгоритмов, которое очень сильно отличается от линейных моделей, но в то же время играет важную роль в машинном обучении.

7.1.1. Линейные модели (обзор)

До этого момента изучались линейные модели. К особенностям линейных моделей относится следующее:

- Линейные модели быстро учатся. В случае со среднеквадратичной ошибкой для вектора весов даже есть аналитическое решение. Также легко применять для линейных моделей градиентный спуск.
- При этом линейные модели могут восстанавливать только простые зависимости из-за ограниченного количества параметров (степеней свободы).
- В то же время линейные модели можно использовать для восстановления нелинейных зависимостей за счет перехода к спрямляющему пространству, что является довольно сложной операцией.

Отдельно стоит отметить, что линейные модели не отражают особенности процесса принятия решений у людей. На самом деле, когда человек хочет понять ту или иную вещь, он будет задавать последовательность из простых вопросов, которые в итоге приведут его к какому-нибудь ответу.

7.1.2. Решающие деревья (пример 1)

Чтобы понять принцип работы решающих деревьев, полезно рассмотреть следующий сильно упрощенный пример.

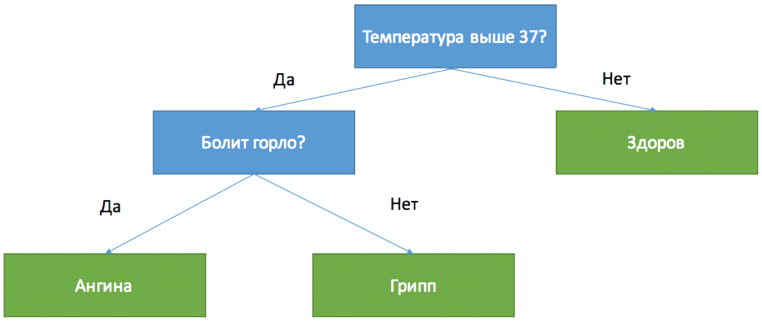


Рис. 7.1

Необходимо провести медицинскую диагностику. Врач, который проводит эту диагностику, знает только 2 заболевания — ангина и грипп. Поэтому сначала он спрашивает, какая температура у пациента. Если она меньше 37 градусов, он заключает, что пациент здоров, в ином случае — переходит к следующему вопросу, а

именно, спрашивает, болит ли у пациента горло. Если оно болит, врач ставит диагноз ангина, в ином случае — грипп.

Создатели курса не рекомендуют серьезно относиться к предложенному методу диагностики заболеваний.

7.1.3. Решающие деревья (пример 2)

Другой пример — для известной задачи определения того, выживет или не выживет тот или иной пассажир Титаника. Задача очень неплохо решается следующим решающим деревом:

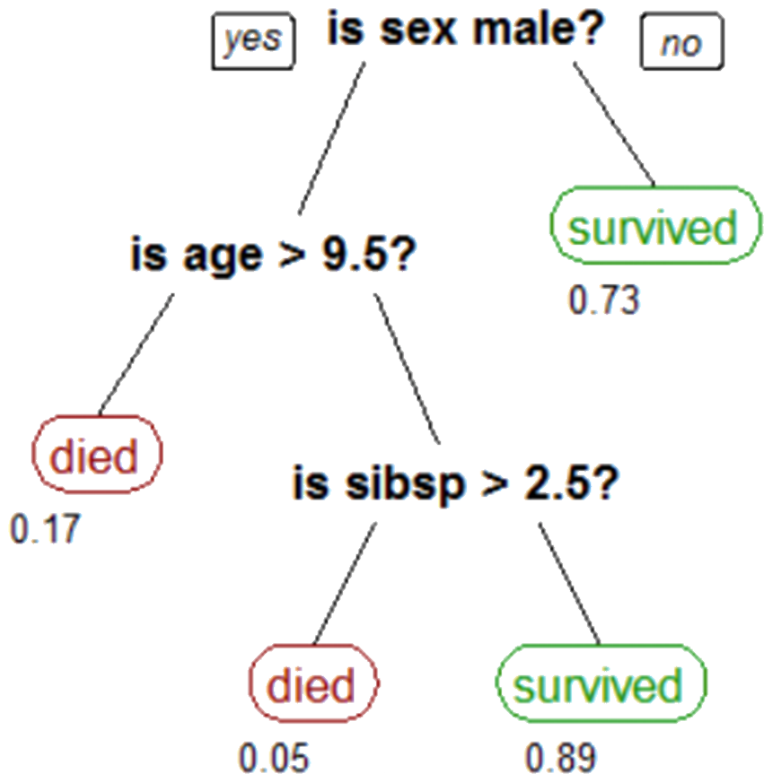


Рис. 7.2

В первую очередь спрашивается пол пассажира. Если это женщина, то решающее дерево сразу заявляет, что она выживает, и этот ответ верен в 73% случаев, и так далее.

7.1.4. Решающие деревья

Итак, были рассмотрены два примера решающих деревьев, которые представляли собой бинарные деревья, в каждой внутренней вершине записано условие, а в каждом листе дерева — прогноз. Строго говоря, не обязательно решающее дерево должно быть бинарным, но как правило используются именно бинарные.

Условия во внутренних вершинах выбираются крайне простыми. Наиболее частый вариант — проверить, лежит ли значение некоторого признака x^j левее, чем заданный порог t:

$$[x^j \leq t].$$

Это очень простое условие, которое зависит всего от одного признака, но его достаточно, чтобы решать многие сложные задачи.

Прогноз в листе является вещественным числом, если решается задача регрессии. Если же решается задача классификации, то в качестве прогноза выступает или класс, или распределение вероятностей классов.

7.1.5. Решающие деревья в задаче классификации

Пусть решается задача классификации с двумя признаками и тремя классами.

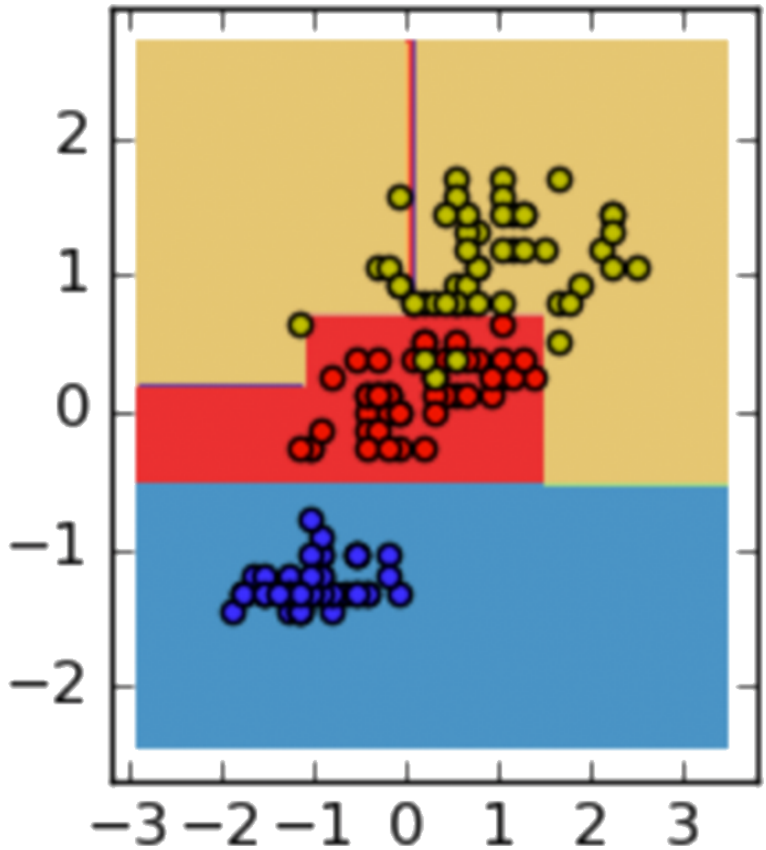


Рис. 7.3: Использование решающих деревьев в задачах классификации

Видно, что решающее дерево может очень неплохо отделить каждый класс от всех остальных. Видно, что разделяющая поверхность каждого класса кусочно-постоянная, и при этом каждая сторона поверхности параллельна оси координат, так как каждое условие сравнивает значение равно одного признака с порогом.

В то же время решающее дерево вполне может переобучиться: его можно сделать настолько глубоким, что каждый лист решающего дерева будет соответствовать ровно одному объекту обучающей выборки. В этом случае, если записать в каждом листе ответ соответствующего объекта, на обучающей выборке получается нулевая ошибка. Дерево получается явно переобученным. Вот пример такого дерева:

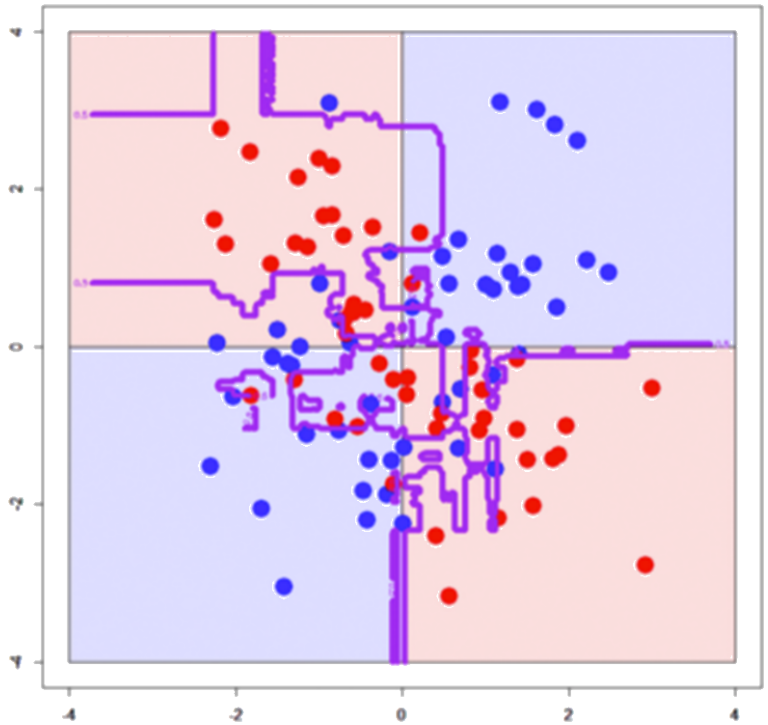


Рис. 7.4: Переобученное решающее дерево

Это дерево идеально отделило синий от красного класса, но разделяющая поверхность получилась безумно сложной — видно, что этот алгоритм переобучился и от него не будет никакой пользы на тестовой выборке.

7.1.6. Решающие деревья в задаче регрессии

Пусть решается задача регрессии с одним признаком, по которому нужно восстановить значение целевой переменной. Не очень глубокое дерево восстанавливает зависимость примерно так:

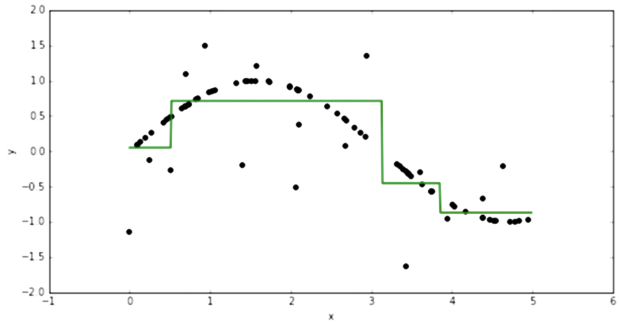


Рис. 7.5: Использование решающих деревьев в задачах регрессии

Восстановленная зависимость будет кусочно-постоянной, но в целом будет иметь неплохое качество.
При увеличении глубины дерева получившаяся функция будет иметь следующий вид:

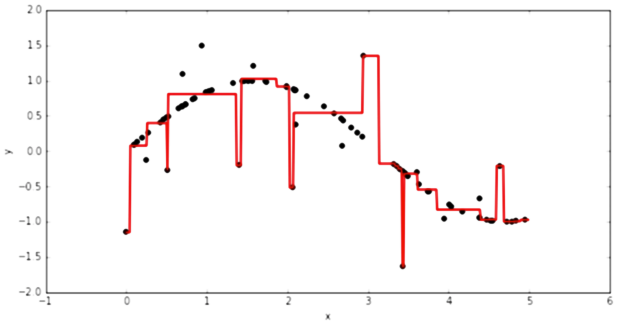


Рис. 7.6: Переобученное решающее дерево

Видно, что дерево подогналось под выбросы и его качество уже будет не таким хорошим. Дерево переобучилось из-за того, что его глубина слишком большая.

7.2. Обучение решающих деревьев

В данном разделе будет рассмотрен вопрос, как строить решающие деревья и как обучать их по конкретной выборке.

7.2.1. Переобучение деревьев

В предыдущем разделе было показано, что решающие деревья очень легко переобучаются. В том числе можно построить дерево, у которого каждый лист будет соответствовать одному объекту обучающей выборки.

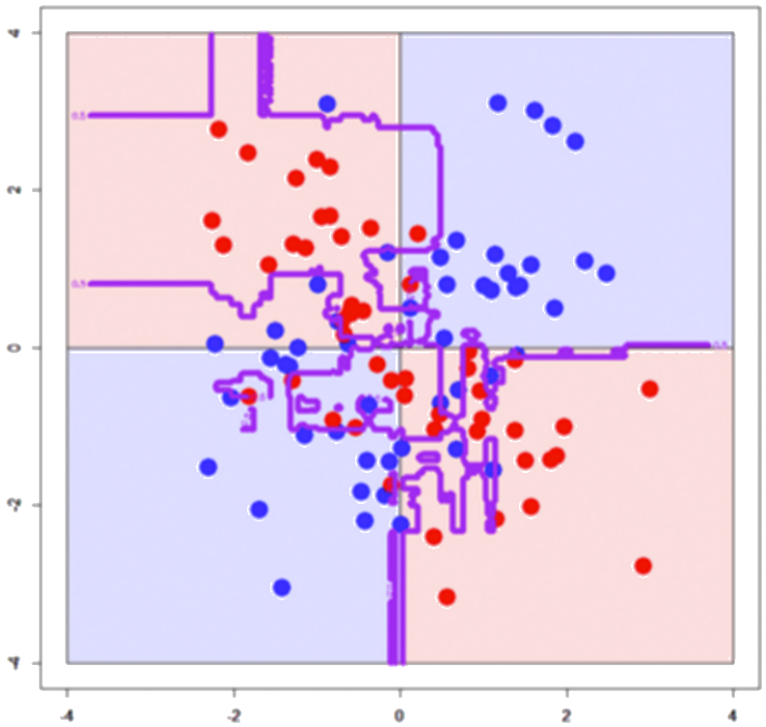


Рис. 7.7: Переобученное решающее дерево

Это дерево строит очень-очень сложную разделяющую поверхность и, очевидно, переобучено.

Поскольку всегда можно построить такое дерево, которое не ошибается на обучающей выборке и будет переобученным, имеет смысл искать минимальное (например, с минимальным числом листьев) дерево из имеющих нулевую ошибку. Но, к сожалению, задача отыскания такого дерева – NP-полная, то есть ее невозможно решить за разумное время.

7.2.2. Жадный способ построения

В машинном обучении применяется жадный способ построения решающего дерева от корня к листьям. Сначала выбирается корень, который разбивает выборку на две. Затем разбивается каждый из потомков этого корня и так далее. Дерево ветвится до тех пор, пока этого не будет достаточно.

Остается уточнить способ разбиения каждого потомка. Как было сказано ранее, в качестве условия в каждой вершине строящегося дерева будет использоваться простейшее условие: значение одного из признаков будет сравниваться с некоторым порогом.

Пусть в вершину m попало множество X_m объектов из обучающей выборки. Параметры в условии $[x^j \leq t]$ будут выбраны так, чтобы минимизировать данный критерий ошибки $Q(X_m, j, t)$, зависящий от этих параметров:

$$Q(X_m, j, t) \to \min_{j,t}.$$

Параметры j и t можно подбирать перебором. Действительно, признаков конечное число, а из всех возможных значений порога t можно рассматривать только те, при которых получаются различные разбиения. Можно показать, что таких значений параметра t столько, сколько различных значений признака x^j на обучающей выборке.

После того, как параметры были выбраны, множество X_m объектов из обучающей выборки разбивается на два множества

$$X_\ell = \left\{x \in X_m | [x^j \leq t]\right\}, \qquad X_r = \left\{x \in X_m | [x^j > t]\right\},$$

каждое из которых соответствует своей дочерней вершине.

Предложенную процедуру можно продолжить для каждой из дочерних вершин: в этом случае дерево будет все больше и больше углубляться. Такой процесс рано или поздно должен остановиться, и очередная дочерняя вершина будет объявлена листком, а не разделена пополам. Этот момент определяется критерием остановки. Существует много различных вариантов критерия остановки:

- Если в вершину попал только один объект обучающей выборки или все объекты принадлежат одному классу (в задачах классификации), дальше разбивать не имеет смысла.
- Можно также останавливать разбиение, если глубина дерева достигла определенного значения.

Возможные критерии останова будут обсуждаться позднее в этом уроке.

Если какая-то вершина не была поделена, а была объявлена листом, нужно определить прогноз, который будет содержаться в данном листе. В этот лист попала некоторая подвыборка X_m исходной обучающей выборки и требуется выбрать такой прогноз, который будет оптимален для данной подвыборки.

В задаче регрессии, если функционал — среднеквадратичная ошибка, оптимально давать средний ответ по этой подвыборке:

$$a_m = \frac{1}{|X_m|} \sum_{i \in X_m} y_i.$$

В задаче классификации оптимально возвращать тот класс, который наиболее популярен среди объектов в X_m:

$$a_m = \operatorname{argmax}_{y \in \mathbb{Y}} \sum_{i \in X_m} [y_i = y].$$

Если требуется указать вероятности классов, их можно указать как долю объектов разных классов в X_m:

$$a_{mk} = \frac{1}{|X_m|} \sum_{i \in X_m} [y_i = k].$$

7.3. Критерии информативности

В этом разделе речь пойдет о критериях информативности, с помощью которых можно выбирать оптимальное разбиение при построении решающего дерева.

7.3.1. Выбор критерия ошибки

Критерий ошибки записывается следующим образом:

$$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$$

и состоит из двух слагаемых, каждое из которых соответствует своему листу.

Функция $H(X)$ называется критерием информативности: ее значение должно быть тем меньше, чем меньше разброс ответов в X.

В случае регрессии разброс ответов — это дисперсия, поэтому критерий информативности в задачах регрессии записывается следующим образом:

$$H(X) = \frac{1}{|X|} \sum_{i \in X} \left(y_i - \bar{y}(X)\right)^2, \qquad \bar{y} = \frac{1}{|X|} \sum_{i \in X} y_i.$$

7.3.2. Критерий информативности Джини

Сформулировать критерий информативности для задачи классификации несколько сложнее. Пусть p_k — доля объектов класса k в выборке X:

$$p_k = \frac{1}{X} \sum_{i \in X} [y_i = k].$$

Критерий информативности Джини формулируется в терминах p_k:

$$H(X) = \sum_{k=1}^{K} p_k (1 - p_k).$$

Все слагаемые в сумме неотрицательные, поэтому критерий Джини также неотрицателен. Его оптимум достигается только в том случае, когда все объекты в X относятся к одному классу.

Одна из интерпретаций критерий Джини — это вероятность ошибки случайного классификатора. Классификатор устроен таким образом, что вероятность выдать класс k равна p_k.

7.3.3. Энтропийный критерий информативности

Еще один критерий информативности — энтропийный критерий:

$$H(X) = -\sum_{k=1}^{K} p_k \ln p_k.$$

В этом выражении полагается, что $0 \ln 0 = 0$.

Энтропийный критерий, как и критерий Джини, неотрицателен, а его оптимум также достигается только в том случае, когда все объекты в X относятся к одному классу.

Энтропийный критерий имеет интересный физический смысл. Он заключается в том, что показывает, насколько распределение классов в X отличается от вырожденного. Энтропия в случае вырожденного распределения равна 0: такое распределение характеризуется минимальной возможной степенью неожиданности. Напротив, равномерное распределение самое неожиданное, и ему соответствует максимальная энтропия.

7.4. Критерий останова и стрижка деревьев

В этом разделе речь пойдет о способах борьбы с переобучением деревьев, а именно о критериях останова и стрижке деревьев.

7.4.1. Критерий останова

Критерий останова используется, чтобы принять решение: разбивать вершину дальше или сделать листовой.

Худший случай решающего дерева — такое, в котором каждый лист соответствует своему объекту обучающей выборки. В этом случае дерево будет максимально переобученным и не будет обобщать информацию, полученную из обучающей выборки. Грамотно подобранный критерия останова позволяет бороться с переобучением.

Самый простой критерий останова проверяет, все ли объекты в вершине относятся к одному классу. Однако такой критерий останова может быть использован только в случае простых выборок, так как для сложных он остановится только тогда, когда в каждом листе останется примерно по одному объекту.

Гораздо более устойчивый и полезный критерий проверяет, сколько объектов оказалось в вершине, и разбиение продолжается, если это число больше, чем некоторое выбранное n. Соответственно, если в вершину попало $\leq n$ объектов, она становится листовой. Параметр n нужно подбирать.

Случай $n = 1$ является худшим случаем, описанным выше. При этом выбирать n нужно так, чтобы по n объектам, которые попали в вершину, можно было устойчиво построить прогноз. Существует рекомендация, что n нужно брать равным 5.

Еще один критерий, гораздо более грубый, заключается в ограничении на глубину дерева. Этот критерий хорошо себя зарекомендовал при построении композиций, когда много решающих деревьев объединяют в один сложный алгоритм. Об этом пойдет речь позже.

7.4.2. Стрижка деревьев

Существует и другой подход к борьбе с переобучением деревьев — стрижка. Он заключается в том, что сначала строится решающее дерево максимальной сложности и глубины, до тех пор, пока в каждой вершине не окажется по 1 объекту обучающей выборки.

После этого начинается «стрижка», то есть удаление листьев в этом дереве по определенному критерию. Например, можно стричь до тех пор, пока улучшается качество некоторой отложенной выборки.

Существует мнение, и это подкреплено многими экспериментами, что стрижка работает гораздо лучше, чем простые критерии, о которых говорилось раньше. Но стрижка — очень ресурсоёмкая процедура, так как, например, может потребоваться вычисление качества дерева на некоторой валидационной выборке на каждом шаге.

На самом деле, сами по себе деревья на сегодняшний день почти не используются, они бывают нужны только для построения композиции и объединения большого числа деревьев в один алгоритм. В случае с композициями такие сложные подходы к борьбе с переобучением уже не нужны, так как достаточно простых критериев останова, ограничения на глубину дерева или на число объектов в листе.

7.5. Решающие деревья и категориальные признаки

В этом разделе будет рассказано, как использовать категориальные признаки в решающих деревьях.

До этого момента использовалось следующее условие в вершине каждого дерева:

$$[x^j \leq t].$$

Очевидно, что такое условие можно записывать только для вещественных или бинарных признаков.

7.5.1. N-арные деревья

Подход, который позволяет включить категориальные признаки в деревья, состоит в том, чтобы строить n-арные деревья, то есть такие деревья, что из каждой вершины могут выходить до n ребер. Пусть необходимо разбить некоторую вершину по некоторому признаку. Если этот признак — вещественный или бинарный, то все еще можно использовать простое условие с порогом t, поэтому интерес представляет именно случай категориального признака.

Если x^j — категориальный признак, который может принимать значения

$$\{c_1, ..., c_n\},$$

можно разбить вершину на n вершин таким образом, что в i-ую дочернюю вершину идут объекты с $x^j = c_i$. Критерий ошибки такого разбиения строится по аналогии со случаем бинарного дерева. Поскольку X_n разбивается на n частей, а не на две, в выражении будет n слагаемых:

$$Q(X_m, j) = \sum_{i=1}^{n} \frac{|X_i|}{|X_m|} H(X_i) \to \min_j .$$

Таким образом, если вершину m нужно разбить, рассматриваются все возможные вещественные, бинарные и категориальные признаки. Для вещественных и бинарных признаков считается $Q(X_m, j, t)$, а для n-арных — так, как написано выше. Разбиение вершины будет происходить по тому признаку, для которого значение критерия ошибки будет минимальным.

Важное замечание состоит в том, что при делении вершины по категориальному признаку получается больше дочерних вершин, на которых, очень вероятно, будет достигаться более высокое качество и более низкое значение критерия информативности, поэтому, скорее всего, при таком подходе предпочтение почти всегда будет отдаваться разбиению по категориальным признакам с большим числом возможных значений. В результате получается много листьев в дереве, что почти гарантированно может привести к переобучению.

Однако это не всегда так. Если выборки настолько большие, что даже после разбиения по категориальному признаку в каждом поддереве будет оставаться много объектов, то такое дерево будет неплохо работать, поскольку оно будет восстанавливать сложные зависимости, и при этом не переобучаться при использовании должного критерия останова.

7.5.2. Бинарные деревья с разбиением множества значений

Другой подход позволяет не переходить к n-арным деревьям и продолжать работать с бинарными деревьями. Пусть также необходимо сделать разбиение вершины m, а категориальный признак x^j может принимать значения $C = \{c_1, ..., c_n\}$.

Для этого сначала необходимо разбить множество значений категориального признака на два не пересекающихся подмножества:

$$C = C_1 \cup C_2, \qquad C_1 \cap C_2 = \varnothing.$$

После того, как такое разбиение построено, условие в данной вершине будет выглядеть просто:

$$[x^j \in C_1]$$

Это условие проверяет, в какое из подмножеств попадает значение признака в данный момент на данном объекте. Главным вопросом остается то, как именно нужно разбивать множество C.

Полное количество возможных разбиений множества на два подмножества — 2^n. К счастью, есть хитрость, которая позволяет избежать полного перебора и, более того, работать с категориальным признаком как с вещественным. Для этого возможные значения категориального признака сортируются специальным образом $c_{(1)}, ..., c_{(n)}$ и заменяются на натуральные числа 1,...,n. После этого с данными признаком следует уже работать как с вещественным, а значение порога t будет определять разделение множества C на два подмножества.

Сортировать значения категориального признака в случае задачи бинарной классификации нужно по следующему принципу:

$$\frac{\sum_{i \in X_m} [x_i^j = c_{(1)}][y_i = +1]}{\sum_{i \in X_m} [x_i^j = c_{(1)}]} \leq \ ... \ \leq \frac{\sum_{i \in X_m} [x_i^j = c_{(n)}][y_i = +1]}{\sum_{i \in X_m} [x_i^j = c_{(n)}]}$$

Фактически, значения категориального признака сортируются по возрастанию доли объектов $+1$ класса среди объектов выборки X_n с соответствующим значением этого признака.

Для задачи регрессии сортировка происходит похожим образом, но вычисляется не доля объектов положительного класса, а средний ответ по всем объектам, у которых значение категориального признака равно c:

$$\frac{\sum_{i \in X_m} [x_i^j = c_{(1)}] y_i}{\sum_{i \in X_m} [x_i^j = c_{(1)}]} \leq \ ... \ \leq \frac{\sum_{i \in X_m} [x_i^j = c_{(n)}] y_i}{\sum_{i \in X_m} [x_i^j = c_{(n)}]}.$$

Главная особенность такого подхода состоит в том, что полученный результат полностью эквивалентен результату, который можно было бы получить в результате полного перебора. Это условие работает для критерия Джини, MSE и энтропийного критерия.

Урок 8

Случайные леса

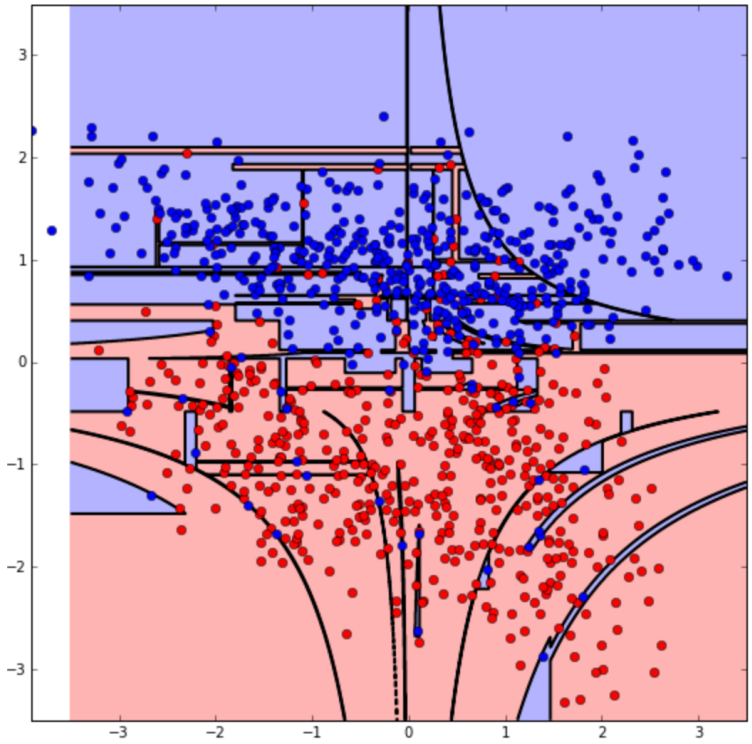
В прошлом уроке изучались решающие деревья и было установлено, что они способны восстанавливать очень сложные закономерности, следовательно, склонны к переобучению. Другими словами, деревья слишком легко подгоняются под обучающую выборку и получаются непригодными для построения прогнозов.

Но оказывается, решающие деревья очень хорошо подходят для объединения в композиции и построения одного непереобученного алгоритма на основе большого количества решающих деревьев.

8.1. Композиции деревьев

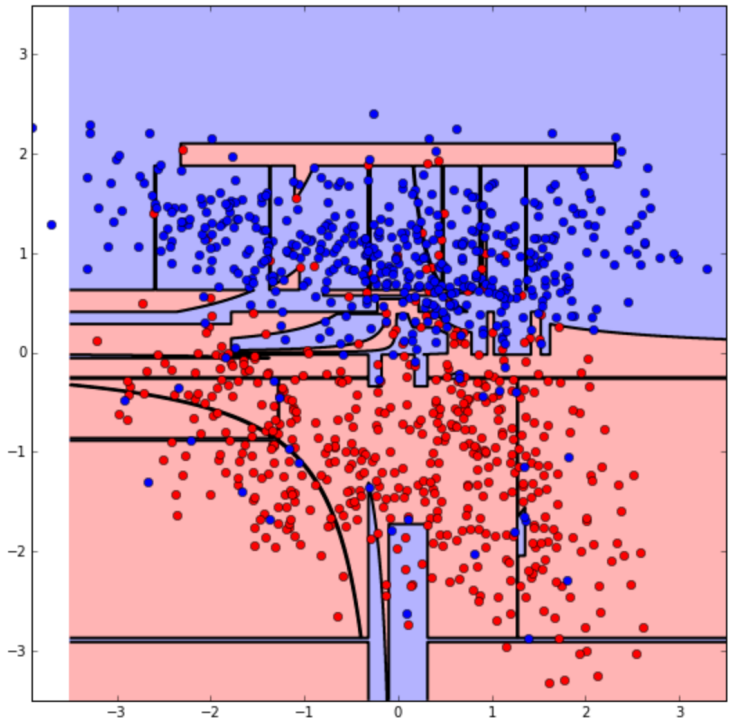
8.1.1. Основные недостатки решающих деревьев

Если взять сложную выборку и обучить на ней решающее дерево до конца, то есть пока в каждом из лепестков не останется по одному объекту, получившаяся разделяющая поверхность будет очень сложной:



Даже если какой-то объект попадает в «гущу другого класса», разделяющая поверхность пытается «уловить» его и выдать на нем правильный ответ.

Если немного изменить обучающую выборку, например выкинуть пару объектов, то обученное на получившейся выборке дерево все еще будет характеризоваться изрезанной и переобученной разделяющей поверхностью, но совершенно другой:



Разделяющая поверхность крайне неустойчива к изменению выборки. Другими словами, решающее дерево обладает следующими серьезными недостатками:

- сильно переобучается
- сильно меняется при небольшом изменении выборки

На самом деле, второй пункт можно будет превратить в достоинство с помощью композиции.

8.1.2. Композиция алгоритмов

Композиция — это объединение N алгоритмов $b_1(x), ..., b_N(x)$ в один. Идея заключается в том, чтобы обучить алгоритмы $b_1(x), ..., b_N(x)$, а затем усреднить полученные от них ответы:

$$a(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x).$$

Это выражение непосредственно является ответом в задаче регрессии. В задачах классификации нужно будет взять знак от получившегося выражения:

$$a(x) = \operatorname{sign}\frac{1}{N}\sum_{n=1}^{N} b_n(x),$$

Алгоритм $a(x)$, который возвращает среднее или знак среднего, называется композицией N алгоритмов $b_1(x), ..., b_N(x)$, а они сами называются базовыми алгоритмами.

Например, пусть при решении задачи классификации с двумя классами использовались 6 базовых алгоритмов, которые на некотором объекте x выдали следующие ответы:

$$-1, -1, 1, -1, 1, -1.$$

Ответ композиции этих 6 алгоритмов будет:

$$a(x) = \operatorname{sign}\left(-\frac{2}{6}\right) = -1.$$

Попросту говоря, объект был отнесен к классу -1, так как за этот вариант «проголосовало» большинство базовых алгоритмов.

8.1.3. Рандомизация

Чтобы построить композицию, нужно сначала обучить N базовых алгоритмов, причем их нельзя обучать на всей обучающей выборке, так как в этом случае они получаются одинаковыми, и в их усреднении не будет никакого смысла.

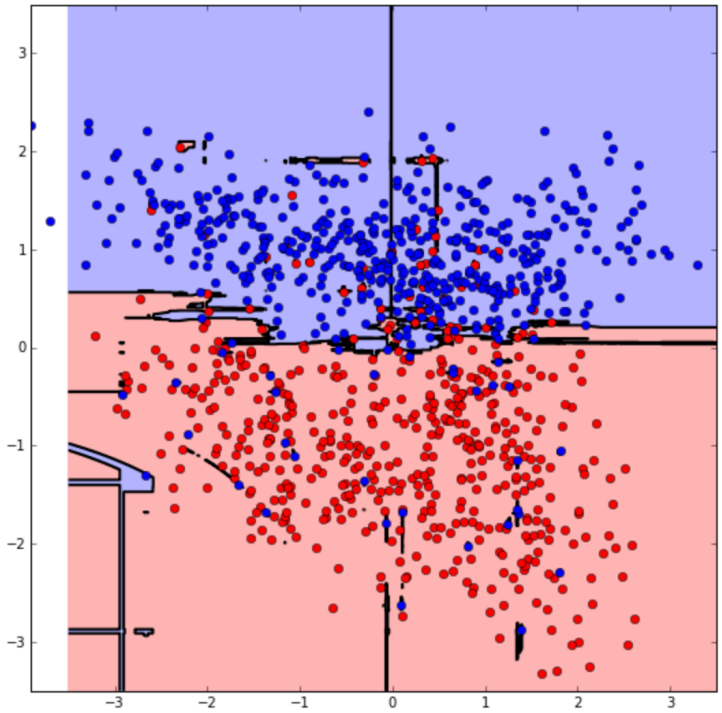
Использовать рандомизацию, то есть обучать базовые алгоритмы на разных подвыборках обучающей выборки, — это один из способов сделать базовые алгоритмы различными. А поскольку решающие деревья сильно меняются даже от небольших изменений обучающей выборки, такая рандомизация значительно повышает различность базовых алгоритмов.

Так называемый бутстрап — один из популярных подходов к построению подвыборок. Он заключается в том, что из обучающей выборки длины ℓ выбирают с возвращением ℓ объектов. При этом новая выборка также будет иметь размер ℓ, но некоторые объекты в ней будут повторятся, а некоторые объекты из исходной выборки в нее не попадут. Можно показать, что в бутстрапированной выборке будет содержаться в среднем 63% различных объектов исходной выборки.

Другой подход к рандомизации — генерация случайного подмножества обучающей выборки. Размер этого случайного подмножества является гиперпараметром. Например, можно случайно взять половину исходной выборки и обучить на ней базовый алгоритм. Этот подход несколько проигрывает бутстрапу, так как содержит гиперпараметр, в то время как бутстрап без какой-либо настройки выдает подвыборку.

8.1.4. Композиция деревьев

Если с помощью бутстрапа построить 100 базовых решающих деревьев и объединить их в композицию, разделяющая поверхность будет все еще сложная, но уже гораздо менее переобученная:



Разделяющая поверхность уже не подгоняется под большую часть попавших в гущу чужого класса объектов и в целом хорошо разделяет два класса. Увеличением количества базовых алгоритмов можно устранить оставшиеся погрешности.

8.2. Смещение и разброс

В этом разделе речь пойдет о разложении ошибки на шум, смещение и разброс. Эта техника позволяет глубже понять причины, почему усреднение алгоритмов позволяет повысить качество.

8.2.1. Разложение ошибки: шум, смещение и разброс

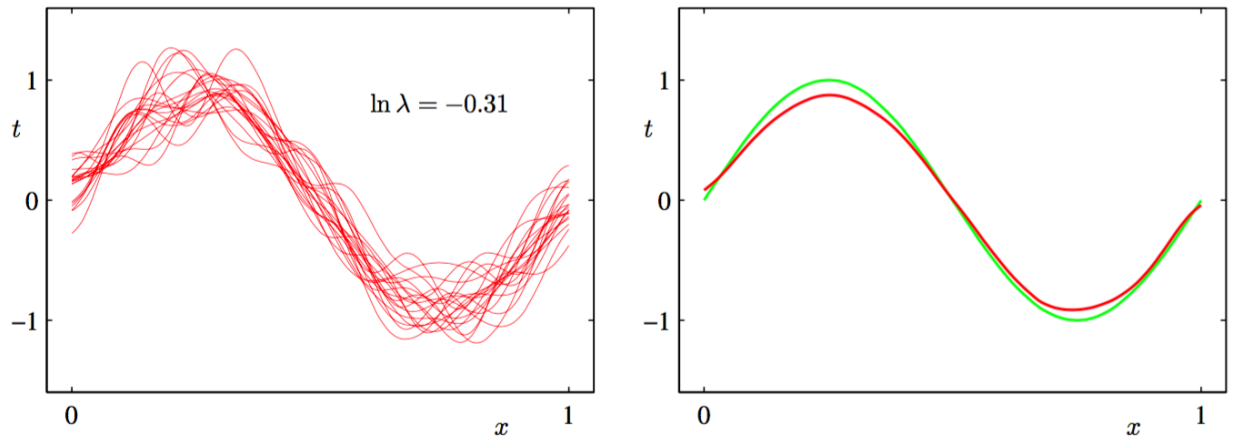
Ошибка алгоритма на новых тестовых данных складывается из трех компонент: шума, смещения и разброса. При этом все они характеризуют разные аспекты данных и модели, с помощью которой решается задача на этих данных:

- **Шум** — компонента ошибки алгоритма, которая будет проявляться даже на идеальной модели в этой задаче. Другими словами, шум является характеристикой данных и будет проявляться, какая бы модель не использовалась.

Пусть обучающая выборка генерируется из некоторого вероятностного распределения. На каждой конкретной обучающей выборке можно обучить некоторую модель и использовать обученную модель на тестовой выборке.

- **Смещение** — отклонение, усредненного по различным обучающим выборкам, прогноза заданной модели от прогноза идеальной модели.
- **Разброс** — дисперсия ответов моделей, обученных по различным обучающим выборкам. Разброс характеризует то, насколько сильно прогноз алгоритма зависит от конкретной обучающей выборки.

Продемонстрировать разложение ошибки на смещение и разброс можно на следующем примере. Рассматривается задача регрессии: требуется аппроксимировать истинную зависимость (изображена на правом графике зеленым) полиномом третьего порядка по обучающей выборке. Обучающая выборка представляет собой 10 случайных точек истинной зависимости, к которым был добавлен случайный шум. На левом графике изображены полиномы, получающиеся для различных обучающих выборок.



Усредненный полином (изображен красной линией на правом рисунке) практически идеально попадает в истинную зависимость, но каждый полином по отдельности существенно от нее отличается. Другими словами, используемое семейство алгоритмов обладает низким смещением, но довольно большим разбросом.

Линейные модели способны восстанавливать только линейные зависимости, а, следовательно, в случае нелинейных задач, которых подавляющее большинство, смещение при использовании таких алгоритмов будет большим. Разброс, наоборот, будет маленьким из-за малого числа параметров, сравнимого с количеством признаков. Вряд ли параметры линейной модели сильно поменяются при незначительном изменении обучающей выборки. Решающие деревья — полная противоположность. Они характеризуются низким смещением, то есть способны восстанавливать сложные закономерности, и большим разбросом: решающие деревья сильно меняются даже при небольших изменениях обучающей выборки.

8.2.2. Смещение и разброс композиции алгоритмов

При вычислении композиции базовых алгоритмов (с одинаковым смещением) смещение композиции совпадает со смещением отдельного базового алгоритма. Таким образом, поскольку деревья характеризуются низким

смещением, то же самое будет верно и для композиции деревьев. Следовательно, композиции деревьев тоже способны восстанавливать сложные закономерности.

Разброс композиции уже отличается от разброса одного базового алгоритма:

$$\left(\substack{\textbf{разброс}\\ \textbf{композиции}}\right) = \frac{1}{N}\left(\substack{\textbf{разброс одного}\\ \textbf{базового алгоритма}}\right) + \left(\substack{\textbf{корелляция между}\\ \textbf{базовыми алгоритмами}}\right).$$

Если базовые алгоритмы независимы, то есть их прогнозы не коррелируют между собой, выражение упрощается:

$$\left(\substack{\textbf{разброс}\\ \textbf{композиции}}\right) = \frac{1}{N}\left(\substack{\textbf{разброс одного}\\ \textbf{базового алгоритма}}\right).$$

Фактически, композиция достаточного количества некоррелированных алгоритмов может дать идеальный алгоритм. Но, к сожалению, базовые алгоритмы всегда получаются в той или иной степени коррелированы, так как обучаются на подвыборках одной выборки. Таким образом, возникает необходимость уменьшения корелляции базовых алгоритмов.

8.2.3. Уменьшение корреляции базовых алгоритмов

Существуют следующие два подхода по уменьшению корреляции базовых алгоритмов:

1. **Беггинг:** Обучение базовых алгоритмов происходит на случайных подвыборках обучающей выборки. Причем чем меньше размер случайной подвыборки, тем более независимыми получаются базовые алгоритмы.

2. **Метод случайных подпространств:** выбирается случайное подмножество признаков (столбцов матрицы «объекты–признаки») и очередной базовый алгоритм обучается только на этих признаках. Доля выбираемых признаков является гиперпараметром этого метода.

Два данных подхода — бэггинг и метод случайных подпространств — можно объединять и использовать одновременно.

8.3. Случайные леса

В этом разделе речь пойдет о случайных лесах, которые являются одним из лучших способов объединения деревьев в композиции.

8.3.1. Случайный лес

Ранее были получены следующие результаты:
- Ошибка может быть разложена на смещение и разброс.
- Смещение композиции близко к смещению одного базового алгоритма.
- Разброс при построении композиции уменьшается, причем тем сильнее, чем менее коррелированы базовые алгоритмы.

Рассмотренных в прошлый раз способов понижения корреляции между базовыми алгоритмами (бэггинг и метод случайных подпространств) оказывается недостаточно. Чтобы базовые алгоритмы были еще менее скореллированными, имеет смысл сделать случайным их процесс построения.

8.3.2. Рандомизация процесса построения решающих деревьев

Процесс построения решающих деревьев представляет собой жадный алгоритм, работающий до выполнения критерия останова.

Пусть на некотором шаге алгоритма необходимо разбить вершину m, в которой оказалась выборка X_m, на две. В качестве условия разбиения используется сравнение j-го признака с порогом t:

$$[x^j \leq t].$$

Параметры j и t выбираются исходя из условия минимизации функции ошибки $Q(X_m, j, t)$:

$$Q(X_m, j, t) \to \min_{j,t}.$$

Рандомизировать процесс построения можно, если в задаче поиска оптимальных параметров выбирать j из случайного подмножества признаков размера q. Оказывается, что этот подход действительно позволяет сделать деревья менее коррелированными.

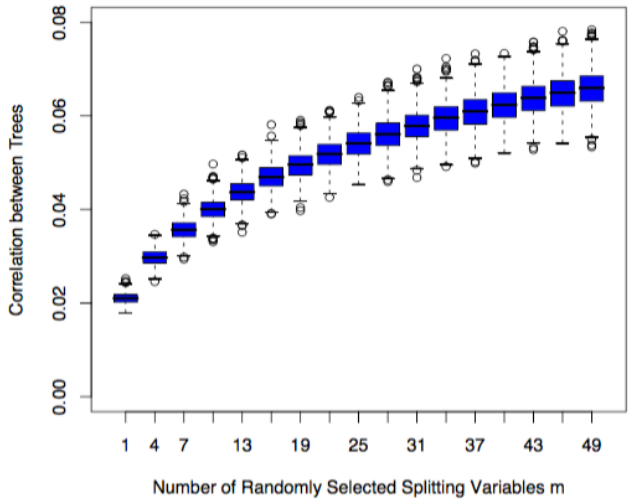


Рис. 8.1: Зависимость корреляции между деревьями от параметра q

По графику видно, что чем меньше «простор для выбора лучшего разбиения», то есть чем меньше q, тем меньше корреляции между получающимися решающими деревьями. Случай $q = 1$ соответствует абсолютно случайному выбору признака.

Для q есть некоторые рекомендации, которые неплохо работают на практике:

- В задаче регрессии имеет смысл брать $q = d/3$, то есть использовать треть от общего числа признаков.
- В задаче классификации имеет смысл брать $q = \sqrt{d}$.

8.3.3. Алгоритм построения случайного леса

Чтобы построить случайный лес из N решающих деревьев, необходимо:

1. Построить с помощью бутстрапа N случайных подвыборок $\tilde{X}_n$, $n = 1, ..., N$.
2. Каждая получившаяся подвыборка $\tilde{X}_n$ используется как обучающая выборка для построения соответствующего решающего дерева $b_n(x)$. Причем:
   - Дерево строится, пока в каждом листе окажется не более n_{min} объектов. Очень часто деревья строят до конца ($n_{min} = 1$), чтобы получить сложные и переобученные решающие деревья с низким смещением.
   - Процесс построения дерева рандомизирован: на этапе выбора оптимального признака, по которому будет происходить разбиение, он ищется не среди всего множества признаков, а среди случайного подмножества размера q.
   - Следует обратить особое внимание, что случайное подмножество размера q выбирается заново каждый раз, когда необходимо разбить очередную вершину. В этом состоит основное отличие такого подхода от метода случайных подпространств, где случайное подмножество признаков выбиралось один раз перед построением базового алгоритма.
3. Построенные деревья объединяются в композицию:
   - В задачах регрессии $a(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x)$;
   - В задачах классификации $a(x) = \text{sign}\, \frac{1}{N}\sum_{n=1}^{N} b_n(x)$.

Одна из особенностей случайных лесов: они не переобучаются при росте числа базовых алгоритмов.

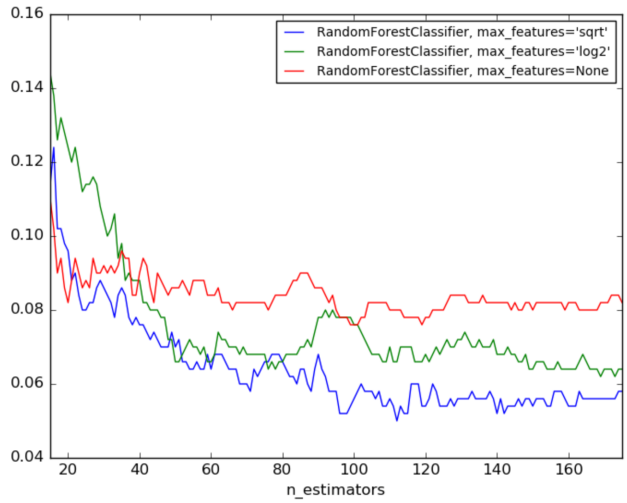


Рис. 8.2: Зависимость качества случайного леса от значения параметра q.

По графику видно, что ошибка на тесте сначала уменьшается с ростом числа базовых алгоритмов, а затем выходит на асимптоту. Не происходит роста ошибки при росте числа базовых алгоритмов.

8.4. Трюки со случайными лесами

Случайный лес обладает рядом интересных особенностей.

8.4.1. Возможность распараллеливания

Поскольку каждое дерево обучается независимо от всех остальных базовых решающих деревьев, его можно обучать на отдельном ядре или отдельном компьютере.

Фактически, данная задача допускает идеальное распараллеливание: скорость вычислений пропорциональна количеству задействованных вычислительных ядер.

8.4.2. Оценивание качества случайного леса

Каждое дерево из случайного леса обучается на бутстрапированной выборке, в которую попадают приблизительно 63% объектов полной выборки. Таким образом, около 37% объектов выборки не использовались при обучении этого дерева, а значит их можно использовать для оценки обобщающей способности случайного леса.

Такой подход носит название out-of-bag и позволяет оценивать качество леса без использования отложенной выборки или кросс-валидации. Формула для оценки качества случайного леса из N деревьев в рамках подхода out-of-bag имеет вид:

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

Эта формула устроена следующим образом. Для каждого объекта x_i из обучающей выборки вычисляется средний прогноз по тем деревьям, в обучающую выборку которых не входит объект x_i:

$$\frac{1}{\sum_{n=1}^{N}[x_i \notin X_n]} \sum_{n=1}^{N}[x_i \notin X_n] b_n(x_i).$$

Для полученного прогноза вычисляется значение ошибки. В качестве оценки качества случайного леса используется сумма таких значений для всех элементов выборки.

Также с помощью случайных лесов и out-of-bag можно отбирать наиболее важные признаки. Об этом пойдет речь в следующем курсе.

Урок 9

Градиентный бустинг

9.1. Композиции простых алгоритмов

В начале данного урока обсудим, почему случайные леса подходят не для всех задач.

9.1.1. Недостатки случайного леса

Случайный лес — композиция глубоких деревьев, которые строятся независимо друг от друга. Но такой подход имеет следующую проблему. Обучение глубоких деревьев требует очень много вычислительных ресурсов, особенно в случае большой выборки или большого числа признаков.

Если ограничить глубину решающих деревьев в случайном лесе, то они уже не смогут улавливать сложные закономерности в данных. Это приведет к тому, что сдвиг будет слишком большим.

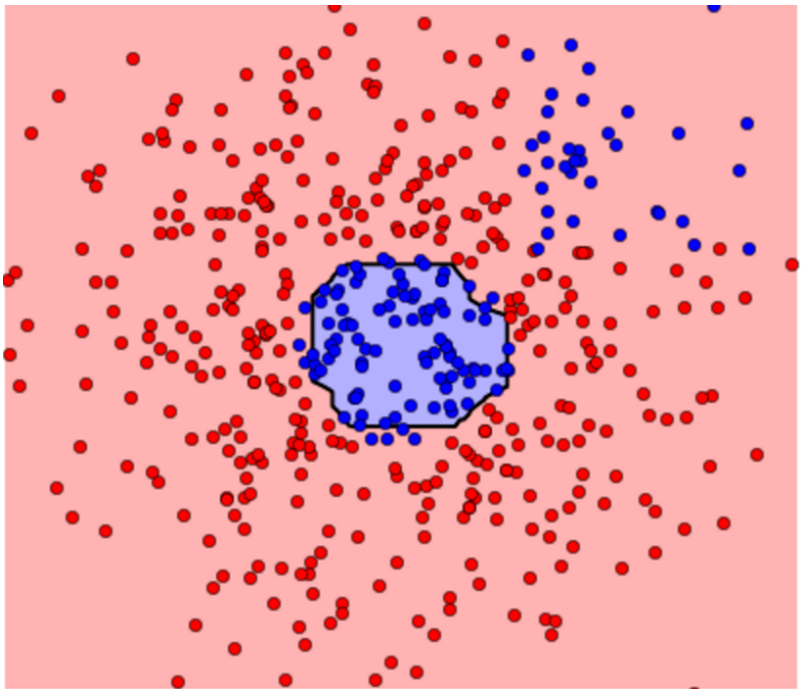


Рис. 9.1: Неглубокие деревья не способны улавливать все закономерности в данных. В данном случае синий класс состоит из двух групп объектов, но неглубокое дерево смогло уловить только центральную группу. На объектах из второй группы такое дерево ошибается.

Вторая проблема со случайным лесом состоит в том, что процесс построения деревьев является ненаправленным: каждое следующее дерево в композиции никак не зависит от предыдущих. Из-за этого для решения сложных задач необходимо огромное количество деревьев.

9.1.2. Бустинг: основная идея

Решить данные проблемы можно с помощью так называемого бустинга. Бустинг — это подход к построению композиций, в рамках которого:

- Базовые алгоритмы строятся последовательно, один за другим.
- Каждый следующий алгоритм строится таким образом, чтобы исправлять ошибки уже построенной композиции.

Благодаря тому, что построение композиций в бустинге является направленным, достаточно использовать простые базовые алгоритмы, например неглубокие деревья.

9.1.3. Бустинг на примере задачи регрессии

Пусть дана задача регрессии, в которой в качестве ошибки используется среднеквадратичная ошибка:

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2 .$$

Для начала необходимо обучить первый простой алгоритм (например, неглубокое решающее дерево):

$$b_1(x) = \text{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - y_i)^2 .$$

Такая задача минимизации квадратичной ошибки легко решается, например, градиентным спуском. Этот алгоритм будет первым алгоритмом в строящейся композиции.

Второй алгоритм должен быть обучен таким образом, чтобы композиция первого и второго алгоритмов:

$$b_1(x_i) + b_2(x_i)$$

имела наименьшую из возможных ошибку на обучающей выборке:

$$b_2(x) = \text{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b_1(x_i) + b(x_i) - y_i)^2 = \text{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - (y_i - b_1(x_i)))^2 .$$

Другими словами, алгоритм $b_2(x)$ улучшает качество работы алгоритма $b_1(x)$. Продолжая по аналогии на N шаге очередной алгоритм $b_N(X)$ будет определяться следующим образом:

$$b_N(x) = \text{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} \left(b(x_i) - \left(y_i - \sum_{n=1}^{N-1} b_n(x_i) \right) \right)^2 .$$

Процесс продолжается до тех пор, пока ошибка композиции $b_1(x) + ... + b_N(x)$ не будет устраивать.

9.2. Градиентный бустинг

Градиентный бустинг является одним из лучших способов направленного построения композиции на сегодняшний день. В градиентном бустинге строящаяся композиция

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

является суммой, а не их усреднением базовых алгоритмов $b_i(x)$. Это связано с тем, что алгоритмы обучаются последовательно и каждый следующий корректирует ошибки предыдущих.

Пусть задана функция потерь $L(y, z)$, где y — истинный ответ, z — прогноз алгоритма на некотором объекте. Примерами возможных функций потерь являются:

- среднеквадратичная ошибка (в задаче регрессии):

$$L(y, z) = (y - z)^2$$

- логистическая функция потерь (в задаче классификации):

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

9.2.1. Инициализация

В начале построения композиции по методу градиентного бустинга нужно ее инициализировать, то есть построить первый базовый алгоритм $b_0(x)$. Этот алгоритм не должен быть сколько-нибудь сложным и не стоит тратить на него много усилий. Например, можно использовать:

- алгоритм $b_0(x) = 0$, который всегда возвращает ноль (в задаче регрессии);
- более сложный $b_0(x) = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i$, который возвращает средний по всем элементам обучающей выборки истинный ответ (в задаче регрессии);
- алгоритм $b_0(x) = \text{argmax}_{y\in\mathbb{Y}} \sum_{i=1}^{\ell}[y_i = y]$, который всегда возвращает метку самого распространенного класса в обучающей выборке (в задаче классификации).

9.2.2. Обучение базовых алгоритмов

Обучение базовых алгоритмов происходит последовательно. Пусть к некоторому моменту обучены $N-1$ алгоритмов $b_1(x), ..., b_{N-1}(x)$, то есть композиция имеет вид:

$$a_{N-1}(x) = \sum_{n=1}^{N-1} b_n(x).$$

Теперь к текущей композиции добавляется еще один алгоритм $b_N(x)$. Этот алгоритм обучается так, чтобы как можно сильнее уменьшить ошибку композиции на обучающей выборке:

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

Сначала имеет смысл решить более простую задачу: определить, какие значения $s_1, ..., s_\ell$ должен принимать алгоритм $b_N(x_i) = s_i$ на объектах обучающей выборки, чтобы ошибка на обучающей выборке была минимальной:

$$F(s) = \sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + s_i\right) \to \min_s,$$

где $s = (s_1, ..., s_\ell)$ — вектор сдвигов.

Другими словами, необходимо найти такой вектор сдвигов s, который будет минимизировать функцию $F(s)$. Поскольку направление наискорейшего убывания функции задается направлением антиградиента, его можно принять в качестве вектора s:

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{N-1}(x_1)), \\ ... \\ -L'_z(y_\ell, a_{N-1}(x_\ell)) \end{pmatrix}.$$

Компоненты вектора сдвигов s, фактически, являются теми значениями, которые на объектах обучающей выборки должен принимать новый алгоритм $b_N(x)$, чтобы минимизировать ошибку строящейся композиции.

Обучение $b_N(x)$, таким образом, представляет собой задачу обучения на размеченных данных, в которой $\{(x_i, s_i)\}_{i=1}^{\ell}$ — обучающая выборка, и используется, например, квадратичная функция ошибки:

$$b_N(x) = \text{argmin}_b \frac{1}{\ell}\sum_{i=1}^{\ell} \left(b(x_i) - s_i\right)^2 .$$

Следует обратить особое внимание на то, что информация об исходной функции потерь $L(y, z)$, которая не обязательно является квадратичной, содержится в выражении для вектора оптимального сдвига s. Поэтому для большинства задач при обучении $b_N(x)$ можно использовать квадратичную функцию потерь.

9.3. Описание алгоритма градиентного бустинга

1. **Инициализация:** инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0.
2. **Шаг итерации:**
   (a) **Вычисляется вектор сдвига**
   $$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{n-1}(x_1)), \\ \dots \\ -L'_z(y_\ell, a_{n-1}(x_\ell)) \end{pmatrix}.$$
   (b) **Строится алгоритм**
   $$b_n(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2,$$
   параметры которого подбираются таким образом, что его значения на элементах обучающей выборки были как можно ближе к вычисленному вектору оптимального сдвига s.
   (c) **Алгоритм $b_n(x)$ добавляется в композицию**
   $$a_n(x) = \sum_{m=1}^{n} b_m(x)$$
3. Если не выполнен критерий останова (об этом будет рассказано далее), то выполнить еще один шаг итерации. Если критерий останова выполнен, остановить итерационный процесс.

9.4. Проблема переобучения градиентного бустинга

9.4.1. Проблема переобучения градиентного бустинга

На следующем графике изображена зависимость ошибки градиентного бустинга от числа используемых деревьев на обучающей и контрольной выборках.

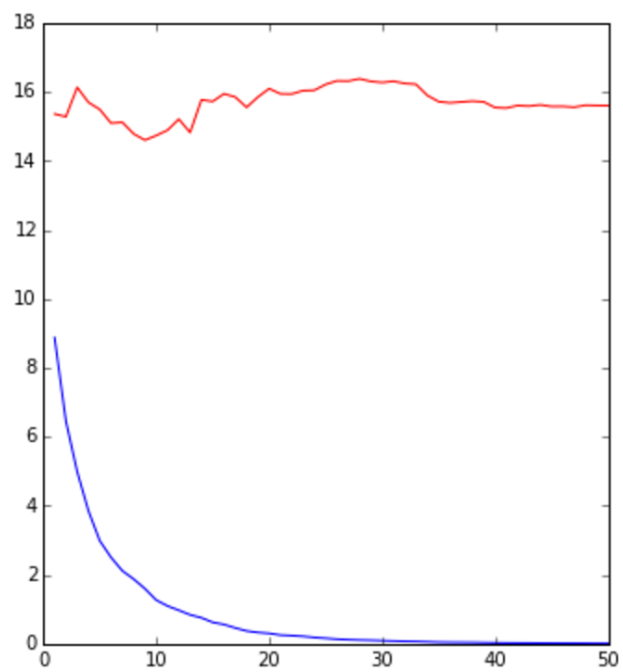


Рис. 9.2: Ошибка в зависимости от числа деревьев: синяя линия — ошибка на обучающей выборке. Красная линия — ошибка на контрольной выборке.

По мере увеличения числа деревьев ошибка на обучающей выборке постепенно уходит в 0. Ошибка на контрольной выборке существенно больше ошибки на обучающей выборке, достигает минимума примерно на 10 итерации, а затем начинает опять возрастать. Имеет место переобучение.

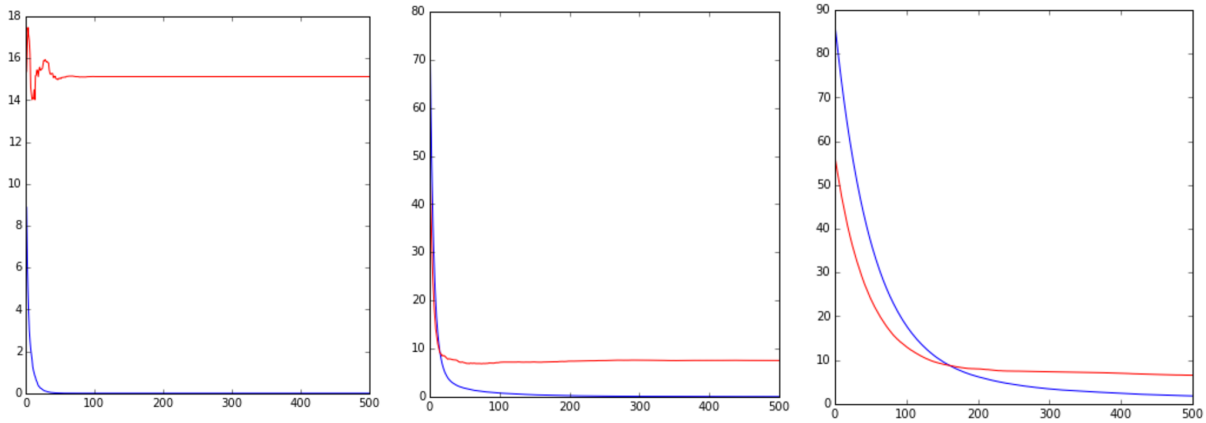
Это связано с тем, что базовый алгоритм пытается приблизить вектор антиградиента на обучающей выборке. Но в градиентном бустинге используются очень простые базовые алгоритмы, например невысокие решающие деревья, которые не могут хорошо аппроксимировать вектор антиградиента на обучающей выборке. Вектор, построенный алгоритмом, будет указывать не в сторону наискорейшего убывания ошибки, то есть вместо градиентного спуска можно получить случайное блуждание. Из-за этого и получается не очень хорошее качество на контроле.

9.4.2. Сокращение размера шага

Чтобы решить эту проблему, нужно «не доверять» направлению, которое построил базовый алгоритм и лишь чуть-чуть смещаться в сторону этого вектора:

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

где $\eta \in (0, 1]$ — длина шага. Это обеспечивает очень аккуратное движение в пространстве, что делает возможным нахождение локального минимума.



(a) Случай $\eta = 1$.

(b) Случай $\eta = 0.1$.

(c) Случай $\eta = 0.01$.

Рис. 9.3: Качество градиентного бустинга на обучающей выборке и контроле при различных η.

Как видно по графикам, при $\eta = 0.1$ качество на контрольной выборке уже существенно лучше, то есть в некотором смысле удалось побороть переобучение. При еще меньшей длине шага $\eta = 0.01$ градиентному бустингу требуется существенно больше итераций, чтобы достичь чуть-чуть большего качества. Таким образом:

- Чем меньше размер шага, тем больше нужно базовых алгоритмов, чтобы достичь хорошего качества, и тем больше времени занимает процесс.
- Чем меньше размер шага, тем лучшего качества можно достичь.

Другими словами, приходится выбирать: или быстро получить достаточно хорошее качество, или получить качество чуть-чуть лучше за большее время.

9.4.3. Подбор гиперпараметров

Размер шага η также является гиперпараметром градиентного бустинга, как и число итерации N. Эти гиперпараметры следует подбирать либо по отложенной выборке, либо по по кросс-валидации. Но подбирать сразу два гиперпараметра довольно сложно, поэтому обычно используют одну из двух следующих стратегий:

1. Зафиксировать размер шага η и подбирать число итераций N
2. Зафиксировать число итераций N и подбирать размер шага η

Обе стратегии неплохо работают, но если время и ресурсы не ограничены, можно попробовать подобрать оба параметра одновременно.

9.4.4. Стохастический градиентный бустинг

Бэггинг — еще один подход к борьбе с переобучением градиентного бустинга, который заключается в том, что каждый базовый алгоритм обучается не на всей выборке, а на некоторой ее случайной подвыборке. Такой подход еще называется стохастическим градиентным бустингом.

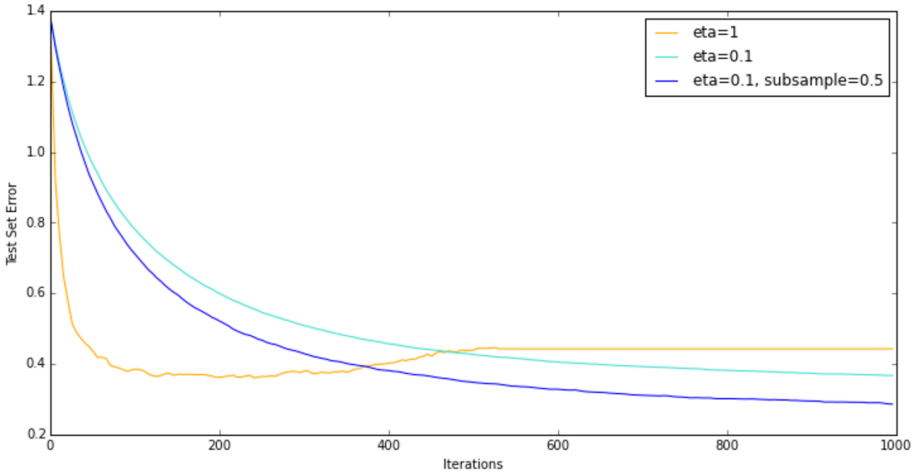


Рис. 9.4: Качество на контрольной выборке в зависимости от числа деревьев.

На графике изображены зависимость качества на контроле от числа деревьев:

- **Оранжевая кривая**: без применения сокращения шага и без бэггинга. Кривая обладает явно выраженным минимумом, после чего алгоритм начинает переобучаться.
- **Бирюзовая кривая**: при использовании сокращения шага с параметром $\eta = 0.1$ без бэггинга. В этом случае требуется больше итерации, чтобы достичь хорошего качества, но и при этом достигается более низкое значение ошибки. То есть метод сокращения шага действительно работает.
- **Синяя кривая**: при использовании сокращения шага с параметром $\eta = 0.1$ и бэггинга с размером подвыборки равным половине обучающей выборки. Форма кривой совпадает совпадает с формой бирюзовой кривой, но при этом за такое же количество итераций алгоритм достигает более низкой ошибки.

То есть использование стохастического градиентного бустинга позволяет уменьшить ошибку или достичь такой же ошибки при таком же числе итераций.

9.5. Градиентный бустинг для регрессии и классификации

9.5.1. Градиентный бустинг

В градиентном бустинге в качестве базовых алгоритмов, как правило, используются не очень глубокие (глубина выбирается от 2 до 8, обычно ближе к 2) решающие деревья. Использования таких не очень глубоких деревьев все же достаточно, чтобы восстанавливать сложные закономерности, поскольку бустинг строит направленную композицию. Чтобы решить возникающую в градиентном бустинге проблему с переобучением, необходимо использовать оба подхода борьбы с ним: сокращение шага и бэггинг.

9.5.2. Градиентный бустинг для регрессии

Типичный функционал ошибки в регрессии — это среднеквадратичная ошибка:

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} \left(a(x_i) - y_i\right)^2.$$

При этом функция потерь, которая измеряет ошибку для одного объекта:

$$L(y, z) = (z - y)^2, \qquad L'_z(y, z) = 2(z - y),$$

где z — это прогноз нашего алгоритма, а y — истинный ответ на данном объекте. Соответственно, вектор сдвигов, каждая компонента которого показывает, как нужно модифицировать ответ на каждом объекте обучающей выборки, чтобы уменьшить среднеквадратичную ошибку, имеет вид:

$$s = \begin{pmatrix} -2\Big(a_{N-1}(x_1) - y_1\Big) \\ \dots \\ -2\Big(a_{N-1}(x_\ell) - y_\ell\Big) \end{pmatrix}.$$

9.5.3. Градиентный бустинг для классификации

В задаче бинарной классификации ($\mathbb{Y} = \{-1, +1\}$) популярным выбором для функции потерь является логистическая функция потерь:

$$\sum_{i=1}^{n} \log\big(1 + \exp\big(-y_i a(x_i)\big)\big),$$

где $a(x) \in \mathbb{R}$ — оценка принадлежности положительному классу. Если $a(x) > 0$, классификатор относит объект x к классу $+1$, а при $a(x) \leq 0$ – к классу -1. Причем, чем больше $|a(x)|$, тем больше классификатор уверен в своем выборе. Функция потерь в этом случае записывается следующим образом:

$$L(y, z) = \log(1 + \exp(-yz)), \qquad L'_z(y, z) = -\frac{y}{1 + \exp(yz)}.$$

Вектор сдвигов s в этом случае будет иметь вид:

$$s = \begin{pmatrix} \frac{y_1}{1+\exp(y_1 a_{N-1}(x_1))} \\ \dots \\ \frac{y_\ell}{1+\exp(y_\ell a_{N-1}(x_\ell))} \end{pmatrix}.$$

Новый базовый алгоритм будет настраиваться таким образом, чтобы вектор его ответов на объектах обучающей выборки был как можно ближе к s. После того, как вычислен алгоритм $a_N(x)$, можно оценить вероятности принадлежности объекта x к каждому из классов:

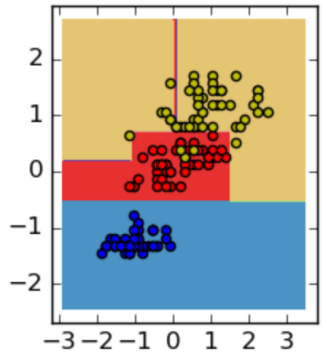
$$P(y = 1|x) = \frac{1}{1 + \exp\left(-a_N(x)\right)}, \qquad P(y = -1|x) = \frac{1}{1 + \exp\left(a_N(x)\right)}.$$

9.6. Градиентный бустинг для решающих деревьев

В этом разделе речь пойдет о том, как применять градиентный бустинг, если базовый алгоритм — это решающие деревья.

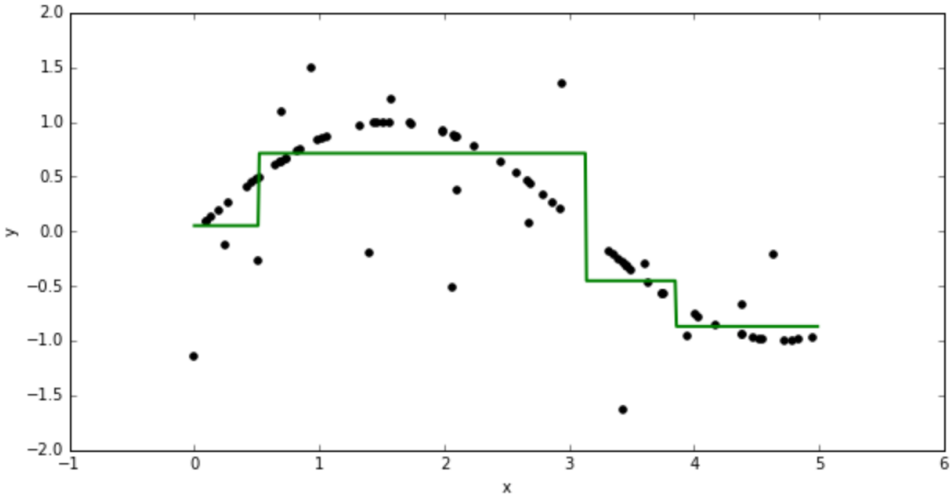
9.6.1. Поверхности, которые восстанавливают решающие деревья

В задаче классификации с тремя классами решающее дерево может разделить выборку примерно вот так:



Поскольку условия в каждой вершине дерева есть сравнение значения какого-то признака с порогом, решающее дерево выделяет каждый класс с помощью некой области, стороны которой параллельны осям координат.

В задаче регрессии функция, которую восстанавливает решающее дерево — кусочно постоянная:



В данном случае есть четыре участка, на каждом из которых функция возвращает постоянное значение.

Решающее дерево, таким образом, разбивает все пространство объектов на J областей:

$$R_1, R_2, ..., R_J,$$

в каждой из которых дерево возвращает постоянное предсказание. Пусть b_j — предсказание дерева в области R_j, тогда решающее дерево $b(x)$ можно записать в следующем виде:

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

Здесь $[x \in R_j]$ — индикатор того, что объект x попал в область R_j.

9.6.2. Градиентный бустинг для решающих деревьев

В градиентном бустинге каждый новый базовый алгоритм b_N прибавляется к уже построенной композиции:

$$a_N(x) = a_{N-1}(x) + b_N(x)$$

Если базовые алгоритмы — это решающие деревья

$$b_N(x) = \sum_{j=1}^{J} [x \in R_{Nj}] b_{Nj},$$

тогда новая композиция a_N будет выглядеть следующим образом:

$$a_N(x) = a_{N1}(x) + \sum_{j=1}^{J} [x \in R_{Nj}] b_{Nj}$$

Последнее выражение можно проинтерпретировать не только как прибавление одного решающего дерева, но и как прибавление J очень простых алгоритмов, каждый из которых возвращает постоянное значение в некоторой области и ноль во всем остальном пространстве. Можно подобрать каждый прогноз b_{Nj}, где N — номер дерева, j — номер листа в этом дереве, таким образом, чтобы он был оптимальным с точки зрения исходной функции потерь:

$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x) + \sum_{j=1}^{J} [x \in R_{Nj}] \, b_{Nj}\right) \to \min_{b_1,\ldots,b_J}$$

Можно показать, что данная задача распадается на J подзадач:

$$b_{Nj} = \operatorname{argmin}_{\gamma \in R} \sum_{x_i \in \mathbb{R}_j} L\left(y_i, a_{N-1}(x) + \gamma\right).$$

Такая задача часто решается аналитически или любым простым методом. Итак, структура базового решающего дерева (структура областей R_j) в градиентном бустинге настраивается минимизацией среднеквадратичной ошибки. Потом можно переподобрать ответы в листьях, то есть перенастроить их, так, чтобы они были оптимальны не с точки зрения среднеквадратичной ошибки (с помощью которой строилось дерево), а с точки зрения исходной функции потерь L. Это позволяет существенно увеличить скорость сходимости градиентного бустинга.

Например, в задаче классификации с логистической функцией потерь, практически оптимальные значения для прогнозов в листьях имеют вид:

$$b_{N_j} = -\frac{\sum_{x_i \in \mathbb{R}_j} s_i}{\sum_{x_i \in \mathbb{R}_j} |s_i|(1 - |s_i|)}.$$