Рассмотрим бинарное дерево, в котором:

каждой внутренней вершине v приписана функция (или предикат)

βv : X →{0, 1};

каждой листовой вершине v приписан прогноз cv ∈ Y (в случае с классификацией листу также может быть приписан вектор вероятностей).

Рассмотрим теперь алгоритм a(x), который стартует из корневой вершины v0 и вычисляет значение функции βv0. Если оно равно нулю, то алгоритм переходит в левую дочернюю вершину, иначе в правую, вычисляет значение предиката в новой вершине и делает переход или влево, или вправо. Процесс продолжается, пока не будет достигнута листовая вершина; алгоритм возвращает тот класс, который приписан этой вершине. Такой алгоритм называется бинарным решающим деревом.

На практике в большинстве случаев используются одномерные предикаты βv, которые сравнивают значение одного из признаков с порогом:

βv (x; j, t) = [xj < t].

Существуют и многомерные предикаты, например:

• линейные βv(x) = [⟨w, x⟩ < t];

• метрические βv(x) = [ρ(x, xv) < t], где точка xv является одним из объектов выборки любой точкой признакового пространства.

Многомерные предикаты позволяют строить ещё более сложные разделяющие поверхности, но очень редко используются на практике — например, из-за того, что усиливают и без того выдающиеся способности деревьев к переобучению. Далее мы будем говорить только об одномерных предикатах.

Легко убедиться, что для любой выборки можно построить решающее дерево, не допускающее на ней ни одной ошибки — даже с простыми одномерными предикатами можно сформировать дерево, в каждом листе которого находится ровно по одному объекту выборки. Скорее всего, это дерево будет переобученным и не сможет показать хорошее качество на новых данных. Можно было бы поставить задачу поиска дерева, которое является минимальным (с точки зрения количества листьев) среди всех деревьев, не допускающих ошибок на обучении — в этом случае можно было бы надеяться на наличие у дерева обобщающей способности. К сожалению, эта задача является NP-полной, и поэтому приходится ограничиваться жадными алгоритмами построения дерева.

Опишем базовый жадный алгоритм построения бинарного решающего дерева. Начнем со всей обучающей выборки X и найдем наилучшее ее разбиение на две части R1(j, t) = {x | xj < t} и R2(j, t) = {x | xj ≥ t} с точки зрения заранее заданного функционала качества Q(X, j, t). Найдя наилучшие значения j и t, создадим корневую вершину дерева, поставив ей в соответствие предикат [xj < t]. Объекты разобьются на две части — одни попадут в левое поддерево, другие в правое. Для каждой из этих подвыборок рекурсивно повторим процедуру, построив дочерние вершины для корневой, и так далее. В каждой вершине мы проверяем, не выполнилось

ли некоторое условие останова — и если выполнилось, то прекращаем рекурсию и объявляем эту вершину листом. Когда дерево построено, каждому листу ставится в соответствие ответ. В случае с классификацией это может быть класс, к которому относится больше всего объектов в листе, или вектор вероятностей (скажем, вероятность класса может быть равна доле его объектов в листе). Для регрессии это может быть среднее значение, медиана или другая функция от целевых переменных объектов в листе. Выбор конкретной функции зависит от функционала качества в исходной задаче.

Решающие деревья могут обрабатывать пропущенные значения — ситуации, в которых для некоторых объектов неизвестны значения одного или нескольких признаков. Для этого необходимо модифицировать процедуру разбиения выборки в вершине, что можно сделать несколькими способами.

После того, как дерево построено, можно провести его стрижку (pruning) — удаление некоторых вершин с целью понижения сложности и повышения обобщающей способности. Существует несколько подходов к стрижке, о которых мы немного упомянем ниже.

Таким образом, конкретный метод построения решающего дерева определяется:

1. Видом предикатов в вершинах;

2. Функционалом качества Q(X, j, t);

3. Критерием останова;

4. Методом обработки пропущенных значений;

5. Методом стрижки.

Также могут иметь место различные расширения, связанные с учетом весов объектов, работой с категориальными признакам и т.д. Ниже мы обсудим варианты каждого из перечисленных пунктов.

При построении дерева необходимо задать функционал качества, на основе которого осуществляется разбиение выборки на каждом шаге. Обозначим через Rm множество объектов, попавших в вершину, разбиваемую на данном шаге, а через Rℓ и Rr — объекты, попадающие в левое и правое поддерево соответственно при заданном предикате. Мы будем использовать функционалы следующего вида:

Q(Rm, j, s) = H(Rm) − |Rℓ | |Rm| H(Rℓ) − |Rr| |Rm| H(Rr).

Здесь H(R) — это критерий информативности (impurity criterion), который оценивает качество распределения целевой переменной среди объектов множества R. Чем меньше разнообразие целевой переменной, тем меньше должно быть значение критерия информативности — и, соответственно, мы будем пытаться минимизировать его значение. Функционал качества Q(Rm, j, s) мы при этом будем максимизировать. Как уже обсуждалось выше, в каждом листе дерево будет выдавать константу — вещественное число, вероятность или класс. Исходя из этого, можно предложить оценивать качество множества объектов R тем, насколько хорошо их целевые переменные предсказываются константой (при оптимальном выборе этой константы):

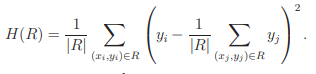


где L (y, c) — некоторая функция потерь. Далее мы обсудим, какие именно критерии информативности часто используют в задачах регрессии и классификации.

Регрессия Как обычно, в регрессии выберем квадрат отклонения в качестве функции потерь. В этом случае критерий информативности будет выглядеть как



Как известно, минимум в этом выражении будет достигаться на среднем значении целевой переменной. Значит, критерий можно переписать в следующем виде:



Мы получили, что информативность вершины измеряется её дисперсией — чем ниже разброс целевой переменной, тем лучше вершина. Разумеется, можно использовать и другие функции ошибки L — например, при выборе абсолютного отклонения мы получим в качестве критерия среднее абсолютное отклонение от медианы.

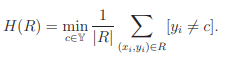
Обозначим через pk долю объектов класса k (k ∈ {1, . . . , K}), попавших в вершину R:



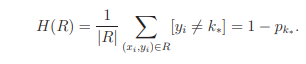
Через k∗ обозначим класс, чьих представителей оказалось больше всего среди объектов, попавших в данную вершину: .

Ошибка классификации

Рассмотрим индикатор ошибки как функцию потерь:

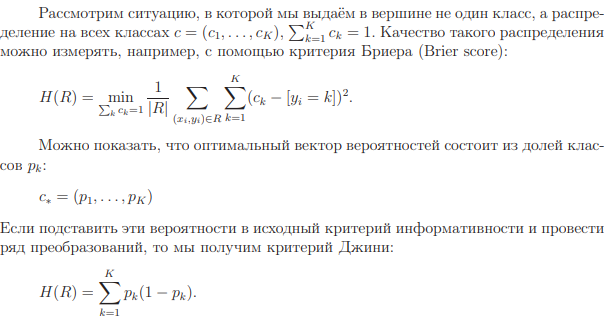


Легко видеть, что оптимальным предсказанием тут будет наиболее популярный класс k∗ — значит, критерий будет равен следующей доле ошибок:

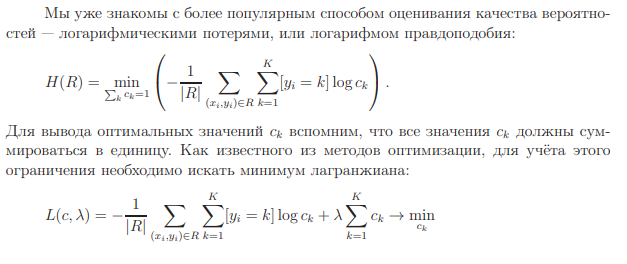


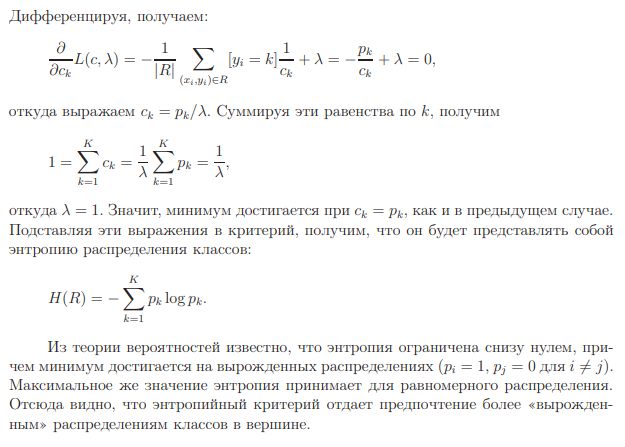
Данный критерий является достаточно грубым, поскольку учитывает частоту pk∗ лишь одного класса.

Критерий Джини



Энтропийный критерий





Критерии останова

Можно придумать большое количестве критериев останова. Перечислим некоторые ограничения и критерии:

• Ограничение максимальной глубины дерева.

• Ограничение минимального числа объектов в листе.

• Ограничение максимального количества листьев в дереве.

• Останов в случае, если все объекты в листе относятся к одному классу.

• Требование, что функционал качества при дроблении улучшался как минимум на s процентов.

С помощью грамотного выбора подобных критериев и их параметров можно существенно повлиять на качество дерева. Тем не менее, такой подбор является трудозатратным и требует проведения кросс-валидации.

Методы стрижки дерева

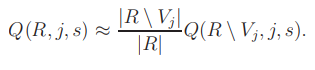
Стрижка дерева является альтернативой критериям останова, описанным выше. При использовании стрижки сначала строится переобученное дерево (например, до тех пор, пока в каждом листе не окажется по одному объекту), а затем производится оптимизация его структуры с целью улучшения обобщающей способности. Существует ряд исследований, показывающих, что стрижка позволяет достичь лучшего качества по сравнению с ранним остановом построения дерева на основе различных критериев. Тем не менее, на данный момент методы стрижки редко используются и не реализованы в большинстве библиотек для анализа данных. Причина заключается в том, что деревья сами по себе являются слабыми алгоритмами и не представляют большого интереса, а при использовании в композициях они либо должны быть переобучены (в случайных лесах), либо должны иметь очень небольшую глубину (в бустинге), из-за чего необходимость в стрижке отпадает. Одним из методов стрижки является cost-complexity pruning. Обозначим дерево, полученное в результате работы жадного алгоритма, через T0. Поскольку в каждом из листьев находятся объекты только одного класса, значение функционала R(T) будет минимально на самом дереве T0 (среди всех поддеревьев). Однако данный функционал характеризует лишь качество дерева на обучающей выборке, и чрезмерная подгонка под нее может привести к переобучению. Чтобы преодолеть эту проблему, введем новый функционал Rα(T), представляющий собой сумму исходного функционала R(T) и штрафа за размер дерева:



где |T| — число листьев в поддереве T, а α > 0 — параметр. Это один из примеров регуляризованных критериев качества, которые ищут баланс между качеством классификации обучающей выборки и сложностью построенной модели. Можно показать, что существует последовательность вложенных деревьев с одинаковыми корнями: TK ⊂ TK−1 ⊂ · · · ⊂ T0, (здесь TK — тривиальное дерево, состоящее из корня дерева T0), в которой каждое дерево Ti минимизирует критерий (5.1) для α из интервала α ∈ [αi , αi+1), причем 0 = α0 < α1 < · · · < αK < ∞. Эту последовательность можно достаточно эффективно найти путем обхода дерева. Далее из нее выбирается оптимальное дерево по отложенной выборке или с помощью кросс-валидации.

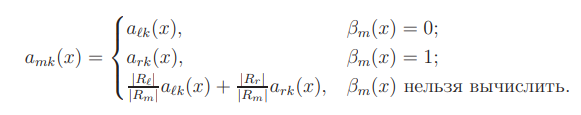
Обработка пропущенных значений

Одним из основных преимуществ решающих деревьев является возможность работы с пропущенными значениями. Рассмотрим некоторые варианты. Пусть нам нужно вычислить функционал качества для предиката β(x) = = [xj < t], но в выборке R для некоторых объектов не известно значение признака j — обозначим их через Vj . В таком случае при вычислении функционала можно просто проигнорировать эти объекты, сделав поправку на потерю информации от этого:



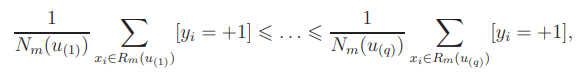
Затем, если данный предикат окажется лучшим, поместим объекты из Vj как в левое, так и в правое поддерево. Также можно присвоить им при этом веса

|Rℓ|/|R| в левом поддереве и |Rr|/|R| в правом. В дальнейшем веса можно учитывать, добавляя их как коэффициенты перед индикаторами [yi = k] во всех формулах. На этапе применения дерева необходимо выполнять похожий трюк. Если объект попал в вершину, предикат которой не может быть вычислен из-за пропуска, то прогнозы для него вычисляются в обоих поддеревьях, и затем усредняются с весами, пропорциональными числу обучающих объектов в этих поддеревьях. Иными словами, если прогноз вероятности для класса k в поддереве Rm обозначается через amk(x), то получаем такую формулу:



Другой подход заключается в построении суррогатных предикатов в каждой вершине. Так называется предикат, который использует другой признак, но при этом дает разбиение, максимально близкое к данному. Отметим, что нередко схожее качество показывают и гораздо более простые способы обработки пропусков — например, можно заменить все пропуски на ноль. Для деревьев также разумно будет заменить пропуски в признаке на числа, которые превосходят любое значение данного признака. В этом случае в дереве можно будет выбрать такое разбиение по этому признаку, что все объекты с известными значениями пойдут в левое поддерево, а все объекты с пропусками — в правое.

Учет категориальных признаков Самый очевидный способ обработки категориальных признаков — разбивать вершину на столько поддеревьев, сколько имеется возможных значений у признака (multi-way splits). Такой подход может показывать хорошие результаты, но при этом есть риск получения дерева с крайне большим числом листьев. Рассмотрим подробнее другой подход. Пусть категориальный признак xj имеет множество значений Q = {u1, . . . , uq}, |Q| = q. Разобьем множество значений на два непересекающихся подмножества: Q = Q1⊔Q2, и определим предикат как индикатор попадания в первое подмножество: β(x) = [xj ∈ Q1]. Таким образом, объект будет попадать в левое поддерево, если признак xj попадает в множество Q1, и в первое поддерево в противном случае. Основная проблема заключается в том, что для построения оптимального предиката нужно перебрать 2q−1 − 1 вариантов разбиения, что может быть не вполне возможным. Оказывается, можно обойтись без полного перебора в случаях с бинарной классификацией и регрессией [1]. Обозначим через Rm(u) множество объектов, которые попали в вершину m и у которых j-й признак имеет значение u; через Nm(u) обозначим количество таких объектов. 9 В случае с бинарной классификацией упорядочим все значения категориального признака на основе того, какая доля объектов с таким значением имеет класс +1:



после чего заменим категорию u(i) на число i, и будем искать разбиение как для вещественного признака. Можно показать, что если искать оптимальное разбиение по критерию Джини или энтропийному критерию, то мы получим такое же разбиение, как и при переборе по всем возможным 2q−1 − 1 вариантам.