Лекция 7
Решающие деревья

Е. А. Соколов
ФКН ВШЭ

5 ноября 2016 г.

Мы уделили достаточно много внимания линейным методам, которые обладают рядом важных достоинств: быстро обучаются, способны работать с большим количеством объектов и признаков, имеют небольшое количество параметров, легко регуляризуются. При этом у них есть и серьёзный недостаток — они могут восстанавливать только линейные зависимости между целевой переменной и признаками. Конечно, можно добавлять в выборку новые признаки, которые нелинейно зависят от исходных, но этот подход является чисто эвристическим, требует выбора типа нелинейности, а также всё равно ограничивает сложность модели сложностью признаков (например, если признаки квадратичные, то и модель сможет восстанавливать только зависимости второго порядка).

В данной лекции мы разберёмся с решающими деревьями — семейством моделей, которые позволяют восстанавливать нелинейные зависимости произвольной сложности.

Решающие деревья хорошо описывают процесс принятия решения во многих ситуациях. Например, когда клиент приходит в банк и просит выдать ему кредит, то сотрудник банка начинает проверять условия:

1. Какой возраст у клиента? Если меньше 18, то отказываем в кредите, иначе продолжаем.

2. Какая зарплата у клиента? Если меньше 50 тысяч рублей, то переходим к шагу 3, иначе к шагу 4.

3. Какой стаж у клиента? Если меньше 10 лет, то не выдаем кредит, иначе выдаем.

4. Есть ли у клиента другие кредиты? Если есть, то отказываем, иначе выдаем.

Такой алгоритм, как и многие другие, очень хорошо описывается решающим деревом. Это отчасти объясняет их популярность.

Первые работы по использованию решающих деревьев для анализа данных появились в 60-х годах, и с тех пор несколько десятилетий им уделялось очень большое внимание. Несмотря на свою интерпретируемость и высокую выразительную способность, деревья крайне трудны для оптимизации из-за свой дискретной структуры — дерево нельзя продифференцировать по параметрам и найти с помощью градиентного спуска хотя бы локальный оптимум. Более того, даже число параметров у них

не является постоянным и может меняться в зависимости от глубины, выбора критериев дробления и прочих деталей. Из-за этого все методы построения решающих деревьев являются жадными и эвристичными.

На сегодняшний день решающие деревья не очень часто используются как отдельные методы классификации или регрессии. В то же время, как оказалось, они очень хорошо объединяются в композиции — решающие леса, которые являются одними из наиболее сильных и универсальных моделей.

1 Определение решающего дерева

Рассмотрим бинарное дерево, в котором:

- каждой внутренней вершине v приписана функция (или предикат) $\beta_v : \mathbb{X} \to \{0, 1\}$;
- каждой листовой вершине v приписан прогноз $c_v \in Y$ (в случае с классификацией листу также может быть вектор вероятностей).

Рассмотрим теперь алгоритм $a(x)$, который стартует из корневой вершины v_0 и вычисляет значение функции β_{v_0}. 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 β_v, которые сравнивают значение одного из признаков с порогом:

$$\beta_v(x; j, t) = [x_j < t].$$

Существуют и многомерные предикаты, например:

- линейные $\beta_v(x) = [\langle w, x \rangle < t]$;
- метрические $\beta_v(x) = [\rho(x, x_v) < t]$, где точка x_v 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.

2 Построение деревьев

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 $R_1(j,t) = \{x \,|\, x_j \leqslant t\}$ и $R_2(j,t) = \{x \,|\, x_j \geqslant t\}$ 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 $[x_j < t]$. 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.

3 Критерии информативности

При построении дерева необходимо задать *функционал ошибки*, на основе которого осуществляется разбиение выборки на каждом шаге. Обозначим через R_m

множество объектов, попавших в вершину, разбиваемую на данном шаге, а через R_ℓ и R_r — объекты, попадающие в левое и правое поддерево соответственно при заданном предикате. Мы будем использовать функционалы следующего вида:

$$Q(R_m, j, s) = H(R_m) - \frac{|R_\ell|}{|R_m|} H(R_\ell) - \frac{|R_r|}{|R_m|} H(R_r).$$

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 $Q(R_m, j, s)$ 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):

$$H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} L(y_i, c),$$

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

§3.1 Регрессия

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

$$H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c)^2.$$

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

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \left(y_i - \frac{1}{|R|} \sum_{(x_j, y_j) \in R} y_j \right)^2.$$

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.

§3.2 Классификация

Обозначим через p_k долю объектов класса k ($k \in \{1, \ldots, K\}$), попавших в вершину R:

$$p_k = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k].$$

Через k_* обозначим класс, чьих представителей оказалось больше всего среди объектов, попавших в данную вершину: $k_* = \arg\max_k p_k$.

3.2.1 Ошибка классификации

Рассмотрим индикатор ошибки как функцию потерь:

$$H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i \neq c].$$

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

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i \neq k_*] = 1 - p_{k_*}.$$

Данный критерий является достаточно грубым, поскольку учитывает частоту p_{k_*} лишь одного класса.

3.2.2 Критерий Джини

Рассмотрим ситуацию, в которой мы выдаём в вершине не один класс, а распределение на всех классах $c = (c_1, \ldots, c_K)$, $\sum_{k=1}^{K} c_k = 1$. Качество такого распределения можно измерять, например, с помощью критерия Бриера (Brier score):

$$H(R) = \min_{\sum_k c_k = 1} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} (c_k - [y_i = k])^2.$$

Можно показать, что оптимальный вектор вероятностей состоит из долей классов p_k:

$$c_* = (p_1, \ldots, p_K)$$

Если подставить эти вероятности в исходный критерий информативности и провести ряд преобразований, то мы получим критерий Джини:

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

3.2.3 Энтропийный критерий

Мы уже знакомы с более популярным способом оценивания качества вероятностей — логарифмическими потерями, или логарифмом правдоподобия:

$$H(R) = \min_{\sum_k c_k = 1} \left(-\frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} [y_i = k] \log c_k \right).$$

Для вывода оптимальных значений c_k вспомним, что все значения c_k должны суммироваться в единицу. Как известного из методов оптимизации, для учёта этого ограничения необходимо искать минимум лагранжиана:

$$L(c, \lambda) = -\frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} [y_i = k] \log c_k + \lambda \sum_{k=1}^{K} c_k \to \min_{c_k}$$

Дифференцируя, получаем:

$$\frac{\partial}{\partial c_k} L(c, \lambda) = -\frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k] \frac{1}{c_k} + \lambda = -\frac{p_k}{c_k} + \lambda = 0,$$

откуда выражаем $c_k = p_k / \lambda$. Суммируя эти равенства по k, получим

$$1 = \sum_{k=1}^{K} c_k = \frac{1}{\lambda} \sum_{k=1}^{K} p_k = \frac{1}{\lambda},$$

откуда $\lambda = 1$. Значит, минимум достигается при $c_k = p_k$, как и в предыдущем случае. Подставляя эти выражения в критерий, получим, что он будет представлять собой энтропию распределения классов:

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

Из теории вероятностей известно, что энтропия ограничена снизу нулем, причем минимум достигается на вырожденных распределениях ($p_i = 1$, $p_j = 0$ для $i \neq j$). Максимальное же значение энтропия принимает для равномерного распределения. Отсюда видно, что энтропийный критерий отдает предпочтение более «вырожденным» распределениям классов в вершине.

4 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.

5 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 T_0. Поскольку в каждом из листьев находятся объекты только одного класса, значение функционала $R(T)$ будет минимально на самом дереве T_0 (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 $R_\alpha(T)$, представляющий собой сумму исходного функционала $R(T)$ и штрафа за размер дерева:

$$R_\alpha(T) = R(T) + \alpha|T|, \tag{5.1}$$

где $|T|$ — число листьев в поддереве T, а $\alpha \geqslant 0$ — 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:

$$T_K \subset T_{K-1} \subset \cdots \subset T_0,$$

(здесь T_K — тривиальное дерево, состоящее из корня дерева T_0), в которой каждое дерево T_i минимизирует критерий (5.1) для α из интервала $\alpha \in [\alpha_i, \alpha_{i+1})$, причем

$$0 = \alpha_0 < \alpha_1 < \cdots < \alpha_K < \infty.$$

Эту последовательность можно достаточно эффективно найти путем обхода дерева. Далее из нее выбирается оптимальное дерево по отложенной выборке или с помощью кросс-валидации.

6 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 $\beta(x) = [x_j < t]$, но в выборке R для некоторых объектов не известно значение признака j — обозначим их через V_j. В таком случае при вычислении функционала можно просто проигнорировать эти объекты, сделав поправку на потерю информации от этого:

$$Q(R, j, s) \approx \frac{|R \setminus V_j|}{|R|} Q(R \setminus V_j, j, s).$$

Затем, если данный предикат окажется лучшим, поместим объекты из V_j как в левое, так и в правое поддерево. Также можно присвоить им при этом веса $|R_\ell|/|R|$ в левом поддереве и $|R_r|/|R|$ в правом. В дальнейшем веса можно учитывать, добавляя их как коэффициенты перед индикаторами $[y_i = k]$ 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 R_m обозначается через $a_{mk}(x)$, то получаем такую формулу:

$$a_{mk}(x) = \begin{cases} a_{\ell k}(x), & \beta_m(x) = 0; \\ a_{rk}(x), & \beta_m(x) = 1; \\ \frac{|R_\ell|}{|R_m|} a_{\ell k}(x) + \frac{|R_r|}{|R_m|} a_{rk}(x), & \beta_m(x) \text{ нельзя вычислить.} \end{cases}$$

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.

7 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 x_j имеет множество значений $Q = \{u_1, \ldots, u_q\}$, $|Q| = q$. Разобьем множество значений на два непересекающихся подмножества: $Q = Q_1 \sqcup Q_2$, и определим предикат как индикатор попадания в первое подмножество: $\beta(x) = [x_j \in Q_1]$. Таким образом, объект будет попадать в левое поддерево, если признак x_j попадает в множество Q_1, и в первое поддерево в противном случае. Основная проблема заключается в том, что для построения оптимального предиката нужно перебрать $2^{q-1} - 1$ вариантов разбиения, что может быть не вполне возможным.

Оказывается, можно обойтись без полного перебора в случаях с бинарной классификацией и регрессией [1]. Обозначим через $R_m(u)$ множество объектов, которые попали в вершину m и у которых j-й признак имеет значение u; через $N_m(u)$ обозначим количество таких объектов.

В случае с бинарной классификацией упорядочим все значения категориального признака на основе того, какая доля объектов с таким значением имеет класс +1:

$$\frac{1}{N_m(u_{(1)})}\sum_{x_i\in R_m(u_{(1)})}[y_i=+1]\leqslant\ldots\leqslant\frac{1}{N_m(u_{(q)})}\sum_{x_i\in R_m(u_{(q)})}[y_i=+1],$$

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 $2^{q-1}-1$ вариантам.

Для задачи регрессии с MSE-функционалом это тоже будет верно, если упорядочивать значения признака по среднему ответу объектов с таким значением:

$$\frac{1}{N_m(u_{(1)})}\sum_{x_i\in R_m(u_{(1)})}y_i\leqslant\ldots\leqslant\frac{1}{N_m(u_{(q)})}\sum_{x_i\in R_m(u_{(q)})}y_i.$$

Именно такой подход используется в библиотеке Spark MLlib [1].

8 Методы построения деревьев

Существует несколько популярных методов построения деревьев:

- ID3: использует энтропийный критерий. Строит дерево до тех пор, пока в каждом листе не окажутся объекты одного класса, либо пока разбиение вершины дает уменьшение энтропийного критерия.
- C4.5: использует критерий Gain Ratio (нормированный энтропийный критерий). Критерий останова — ограничение на число объектов в листе. Стрижка производится с помощью метода Error-Based Pruning, который использует оценки обобщающей способности для принятия решения об удалении вершины. Обработка пропущенных значений осуществляется с помощью метода, который игнорирует объекты с пропущенными значениями при вычислении критерия ветвления, а затем переносит такие объекты в оба поддерева с определенными весами.
- CART: использует критерий Джини. Стрижка осуществляется с помощью Cost-Complexity Pruning. Для обработки пропусков используется метод суррогатных предикатов.

9 Решающие деревья и линейные модели

Как следует из определения, решающее дерево $a(x)$ разбивает всё признаковое пространство на некоторое количество непересекающихся подмножеств $\{J_1,\ldots,J_n\}$,

[1] http://spark.apache.org/docs/latest/mllib-decision-tree.html

и в каждом подмножестве J_j выдаёт константный прогноз w_j. Значит, соответствующий алгоритм можно записать аналитически:

$$a(x) = \sum_{j=1}^{n} w_j [x \in J_j].$$

Обратим внимание, что это линейная модель над признаками $([x \in J_j])_{j=1}^{n}$ — а ведь в начале лекции мы хотели избавиться от линейности! Получается, что решающее дерево с помощью жадного алгоритма подбирает преобразование признаков для данной задачи, а затем просто строит линейную модель над этими признаками. Далее мы увидим, что многие нелинейные методы машинного обучения можно представить как совокупность линейных методов и хитрых способов порождения признаков.

Список литературы

[1] *Hastie T., Tibshirani R., Friedman J.* (2009). The Elements of Statistical Learning.