Семинар: Решающие деревья

Федоров Артем Максимович
Качура Александр Сергеевич

October 2024

Введение

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).

Данная лекция призвана ввести новое для вас понятие Решающих деревьев, семейство моделей, которые позволяют восстанавливать нелинейные зависимости произвольной сложности путем разбиения пространства признаков на области гиперплоскостями. Решающие деревья хорошо и **интерпретируемо** 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.

Минутка истории

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

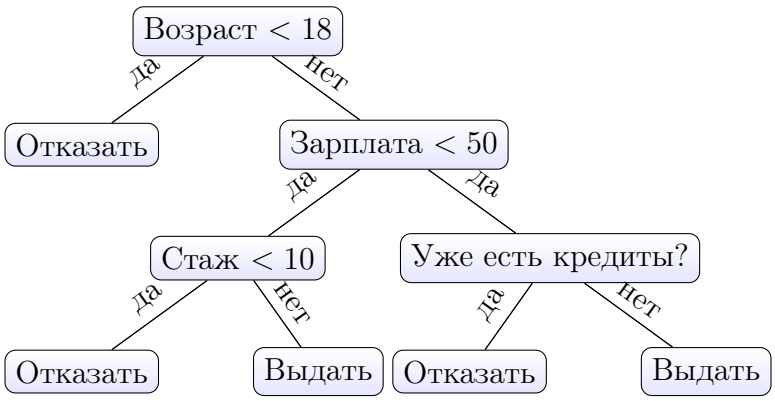


Рис. 1: Визуализация решаюшего дерева

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 Решающее дерево

Пусть рассматриваются объекты из пространства X с соответсвенными целевыми значениями Y. Рассмотрим граф-дерево $G = (E, V)$: пусть каждая вершина бует являться либо листом, либо иметь две дочерних вершины, при чем:

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

Рассмотрим алогритм $a(x)$, для каждого объекта из выборки стартующий из корневой вершины v_0, что вычисляет значения $\beta_{v_0} \Rightarrow$ Если значение равно нулю – алгоритм начинает рекурсивно работать на дереве с вершиной в левой дочерней вершине, иначе – с поддерева из правой вершины. Вычисляя значения каждого предиката в новых вершине, алгоритм рано или поздно дойдет до листовой вершины графа G, на каждом шагу делая шаги влево или вправо. Тогда такой алгоритм принято называть *Бинарным решающим деревом.*

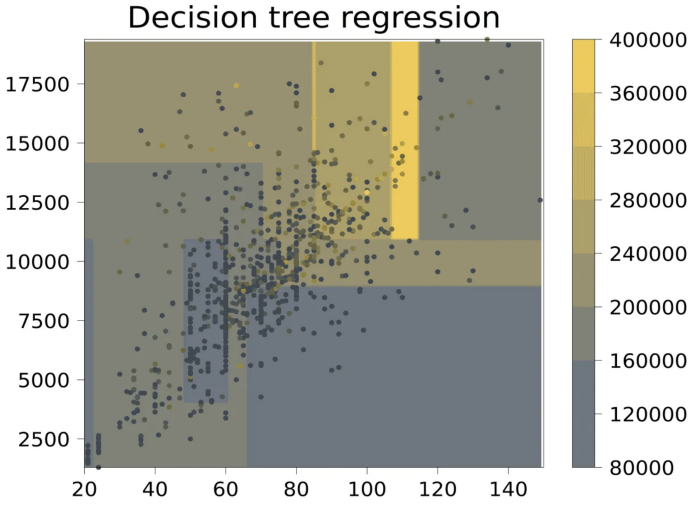
1.1 Какие фукнции β_v могут быть?

На практике в большинстве случаев используются одномерные предикаты β_v, сравнивающие значение одного из признаков объекта x с порогом:

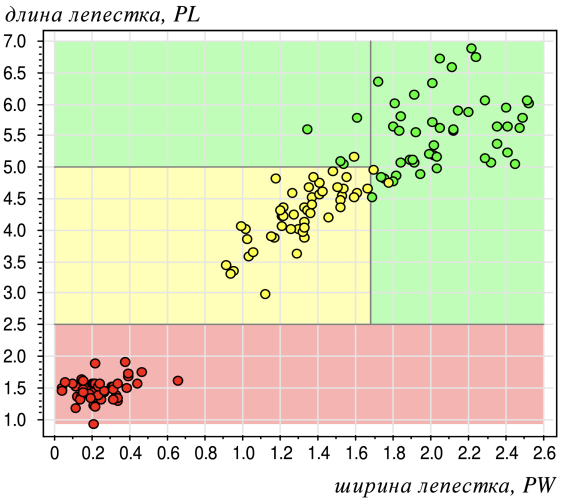
$$\beta_v(x_i, j, t) = [x_i^j < t], \text{ где } x_i^j \text{ i объект, j признак}$$

В природе существуют и более сложные предикаты. Мы можем построить функцию предиката как любое нелинейное преобразование, ограничением будет лишь сигнатура возвращаемого значения: индекс дочерней вершины (для бинарных деревьев – $\{0, 1\}$, для более сложных – сегмент из $\mathbb{N}$). Например:

- линейные по всему объекту $\beta_v(x) = [\langle w, x \rangle < t]$



(a) дерево регрессии



(b) дерево классификации в задаче Ирисов

- метрические $\beta_v(x) = [\rho(x, x_v) < t]$, где точка $x_v \in \mathrm{X}$

Многомерные предикаты позволяют строить ещё более сложные разделяющие поверхности. Из примера [рис. 5.3.2] получаемые разделяющие поверхности есть гиперплоскости, параллельные осям. Можно начать поворачивать пространство, и спользовать поверхности второго, третьего и далее порядков, но очень это редко используется на практике — например, из-за того, что усиливают и без того выдающиеся способности деревьев к переобучению. **Далее мы будем говорить только об одномерных предикатах.**

1.2 Что выдает дерево?

Вне зависимости от преикатов, формы (ветвления графа) решающего дерева действие по возвращению оценки на объекте для всех стратегий данного семейства одно и то же. Проследив путь от корня дерева до **одного единственного** листа, модель возвращает константу, запомненную на обучении для данного листа. Посмотрев на графовое дерево более внимательно и заметив, что для любого листа мы можем отследить точный путь до корня (из ацикличности и конечности) мы способны более формально с точки зрения математики описать решающее дерево в виде:

$$DecisionTree(x) = \sum_{\gamma \in \Gamma} a_\gamma \cdot \prod_{i=1}^{|\gamma|} \mathbb{1}\{\beta_{v_{j_i}}(x) \equiv \gamma_i\} = \sum_{i=0}^{N} a_i \mathbb{1}\{x \in \mathcal{U}_i\}$$

- Γ все листовые вершины дерева, γ путь до соответственного листа
- $\mathcal{U}_i$ области пространства, на которые дробид дерево пространство X

Оказывается, дерево есть простая функция из функционального анализа, из чего следует очень много приятных особенностей данных моделей, очень сильно пригодящиеся в последующем в методах ансамблирования.

1.3 Проблемы деревьев

Решающие деревья – это *чемпионы переобучения*. Легко убедиться, что для любой выборки без регуляризации на число областей разбиения пространства X 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. Такой оверфит деревьев выглядит и вызывается совершенно иным образом, нежели на в моделях, рассматриваемых вами ранее, хоть и приводит к одинаково печальному результату на тестовой выборке. Пример [рис. 1.2]

Другой проблемой деревьев можно назвать их непредсказуемое поведение: из-за разных способов обучения мы можем получить как оптимальное разбиение, так и неподлежащее никакому здоровому анализу. Классический пример это задача **XOR** [рис. 3]. Для справедливости требуетсям отметить, что данное поведение есть результат особенностей выбранной стратегии построения модели, о чем пойдет речь далее.

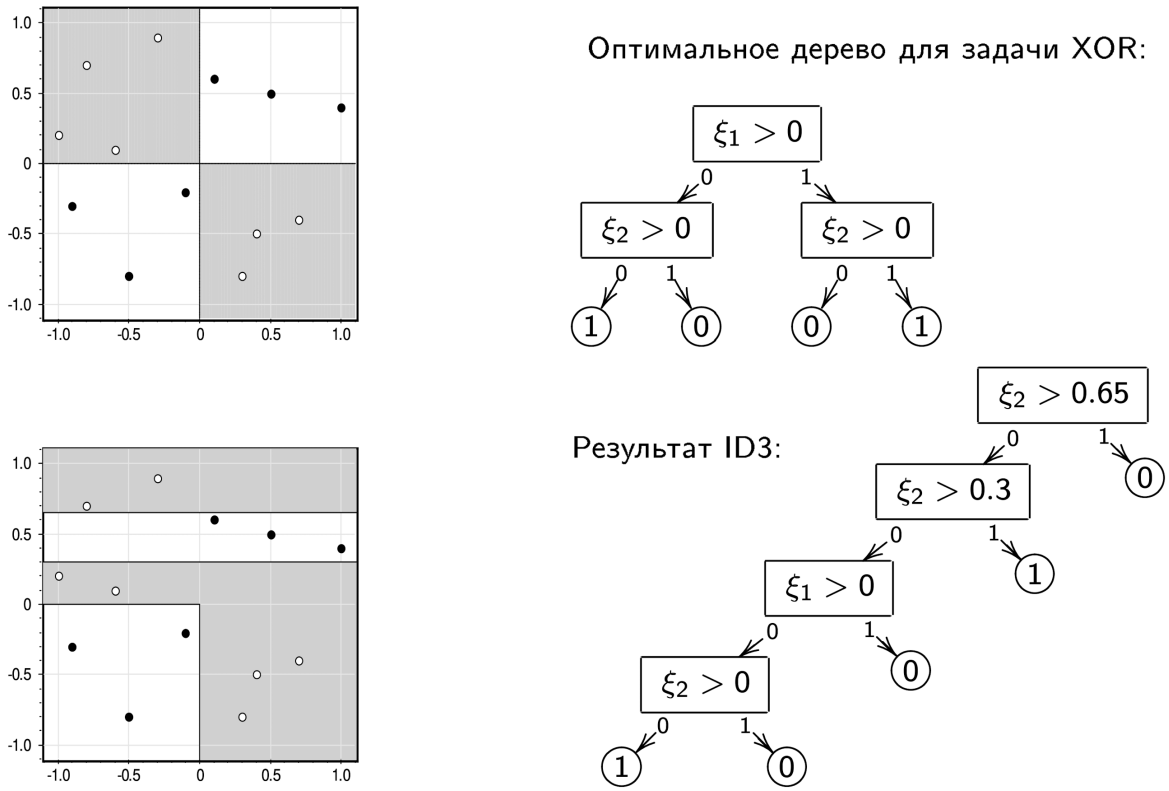


Рис. 3: XOR проблема

2 Построение деревьев

Для избежания обозначеных проблем 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 мы имеем право надеяться на наличие у дерева обобщающей способности, тем не менее, эта задача является NP-полной.

В таком случае выработано две тактики действия:

1. Разрешить себе искать не оптимальное решение, а просто достаточно хорошее. Начать можно с того, чтобы строить дерево с помощью **жадного алгоритма**, то есть не искать всю структуру сразу, а строить дерево этаж за этажом, на каждом этапе подразбивая области пространства так, что общий критерий качества улучшать наискорейшим образом.

2. Заняться оптимизацией с точки зрения computer science — наивную версию алгоритма (перебор наборов возможных предикатов и порогов) можно ускорить и асимптотиче-

ски, и в константу раз. Такой grid-search даже используется на практике, когда нам нужен относительно быстрый результат с малыми амбициями на конечное качество.

2.1 Жадный алгоритм

Введем новые понятия:

1. Будем обозначать за R множество объектов, попавших в вершину графа. Далее это значение будет восприниматься и как множество, и как функция.

2. Q есть *критерий информативности*, с помощью которого мы будем искать оптимальное разбиение – мера схожести объектов в каждой подобласти по целевой переменной.

Опишем базовый жадный алгоритм построения бинарного решающего дерева. Начнем со всей обучающей выборки X и найдем наилучшее ее разбиение на две части $R_1(j,t) = \{x \mid x^j < t\}$ и $R_2(j,t) = \{x \mid x^j \geq t\}$ 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.

Впрочем, есть и другие подходы. При этом строгой теории, которая бы связывала оптимальность выбора разных вариантов этих функций и разных метрик классификации и регрессии, в общем случае не существует и при построении модели приходится пользоваться наборами интуитивных соображений.

Конкретный метод построения дерева диктуется набором:

1. Семейством предикатов в вершинах
2. Функционалом качества $Q(X,j,t)$
3. Критерием останова
4. Методом обработки пропущенных значений
5. Методом стрижки (будет далее)

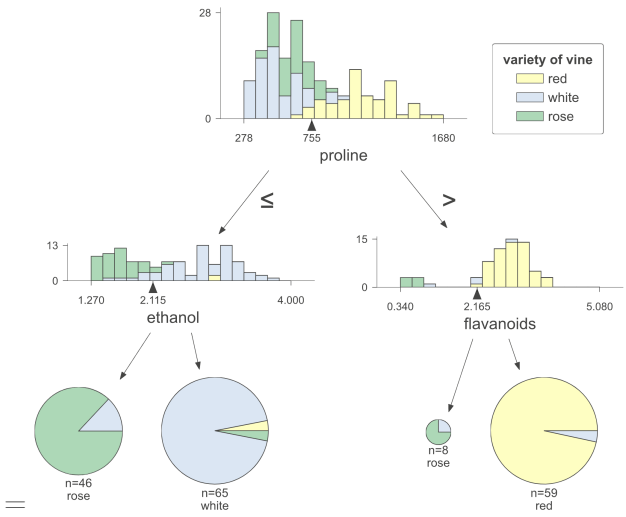


Рис. 4: Построенное дерево глубины 2 на датасете wines жадным алгоритмом

3 Критерий информативности

При построении дерева необходимо задать функционал качества Q. Выведем основную идею, закладываемую в каждый такой фукнционал. 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. Будем считать, в момент, когда мы ищем оптимальное разбиение множества R_m на множества $R_l \cup R_r$ мы решаем задачу оптимизации некоторой функции эмпирического риска с функцией потерь L, а значение, которым мы бы приближали целевую переменную данном листе константой c:

$$H(R_m) = \min_c \frac{1}{|R_m|} \sum_{(x,y)\in R_m} L(x, y, c)$$

Здесь $H(R)$ — это критерий информативности (impurity criterion), который оцени- вает качество распределения целевой переменной среди объектов множества R. Чем меньше разнообразие целевой переменной, тем меньше должно быть значение кри- терия информативности — и, соответственно, мы будем пытаться минимизировать его значение. Функционал качества $Q(R_m, j, s)$ мы при этом будем максимизировать.

3.1 Зачем уменьшать разнообразие целевой переменной в разбиениях?

Наша задача отгадать y_i за наименьшее количество-вопросов, задаваемых объекту x_i. Мы хотим с каждым «вопросом» оставлять как можно меньше неопределенности в ответе на объекте x_i. Тем самым, решая задачу минимизации функционала эмпирического риска на объектах из рассматриваемой области, мы бы ожидали, что в среднем мы бы уменьшили число вопросов, задаваемых объекту в дальнейшем.

Центральную роль в данной теме играет теория информации, а именно **энтропия**. Например, энтропия напрямую связана с числом вопросов к объекту, а уменьшая энтропию на распределении целевых переменных оставляемых в данной области мы бы автоматически добились нашего желания.

3.2 Связь с критерием качества

Казалось бы, мы конструктивно пришли к тому, как поставить задачу оптимищации. Однако это не совсем так: мы полностью не учитываем, на какие области R_l и R_r мы разделяем область R_m. Разделив лист дерева на два множества мы можем получить критерий информативности для каждой из данных областей. Рассмотрим, чему равна их разность:

$$\frac{1}{|R_m|}\left(\min_{c_l} \sum_{(x,y)\in R_l} L(x, y, c_l) + \min_{c_r} \sum_{(x,y)\in R_r} L(x, y, c_r)\right) = \frac{|R_l|}{|R_m|}H(R_l) + \frac{|R_r|}{|R_m|}H(R_r)$$

Тогда выйгрыш от разбиения данного листа будет равняться:

$$Q(R_m, j, s) = H(R_m) - \frac{|R_l|}{|R_m|}H(R_l) - \frac{|R_r|}{|R_m|}H(R_r) \longrightarrow \max_{\{j,s\}}$$

Теперь мы получили понятный механизм ветвления, где выбор H напрямую диктуется решаемой задачей. Вместе с тем мы теперь способны использовать оценку на Q как параметр останова, например, для вершин, где любой способ ветвления не способен привнести весомый вклад в качество модели $\Rightarrow$ объявляем вершину листом.

3.3 Если бы разбиение было бы не на две подобласти?

Пусть теперь $\beta_v : \mathrm{X} \to \{0, 1, \ldots, G\}$. Тогда несложно проверить, что Q приняла бы вид:

$$Q(R_m, j, s) = H(R_m) - \sum_{i=0}^{G} \frac{|R_i|}{|R_m|} H(R_i) \longrightarrow \max_{\{j,\vec{s}\}}$$

4 Регрессионные деревья

Самое очевидное и приятное применение деревьев как простой фукнции из функана. Как обычно, в регрессии выберем квадрат отклонения в качестве функции потерь. В этом случае критерий информативности будет выглядеть как

$$H(R) = \min_{c \in \mathrm{Y}} \frac{1}{|R_m|} \sum_{(x,y) \in R_m} (y - c)^2$$

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

$$H(R_m) = \frac{1}{|R_m|} \sum_{(x,y) \in R_m} \left(y - \frac{1}{|R_m|} \sum_{(x,y) \in R_m} y \right)^2$$

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.

5 Классификационные деревья

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

$$p_k = \frac{1}{|R_m|} \sum_{(x,y) \in R_m} \mathbb{1}\{y = k\}$$

Будем считать, что объекту, попавшему в данный лист, дерево будет присваивать именно этот вектор вероятностей $\vec{p}$ – вероятность его принадлежности к каждому из классов. Тогда рассмотрим самые популярные способы определения функции L для последующего определения H.

5.1 Ошибка классификации

Начнем с самого излюбленного метода введения функционала ошибки для классификации — частотного функционала ошибки. Через k_* обозначим класс, чьих представителей оказалось больше всего среди объектов, попавших в данную вершину: $k_* = arg\max_k p_k$. Рассмотрим индикатор ошибки как функцию потерь:

$$H(R_m) = \min_{c \in \mathrm{Y}} \frac{1}{|R_m|} \sum_{(x,y) \in R_m} \mathbb{1}\{y \neq c\}$$

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

$$H(R_m) = \frac{1}{|R_m|} \sum_{(x,y) \in R_m} \mathbb{1}\{y \neq k_*\} = 1 - p_{k_*}$$

Данный критерий является очень грубым, он не учитывает поведение вероятностей других классов (только p_{k_*}), что, конечно, дает нам основания верить, что со временем алгоритм сможет запомнить весь обучающий датасет, но не очень понятно, насколько это будет эффективно. Самым большим происшествием был бы факт, что на шагах мы получаем равномерные распределения меток в подобластях, что этот функционал не запрещает.

5.2 Энтропийный критерий

Чтобы исправить ошибки предыдущего подхода, обратимся напрямую к уже затронутой теме – минимизации энтропии целевой переменной при разбиении. А именно, пусть в вершине дерева предсказывается фиксированное распределение c, не зависящее от x, тогда правдоподобие имеет вид

$$P(y \mid x, c) = P(y \mid c) = \prod_{(x,y)\in R_m} P\left(y \mid c\right) = \prod_{(x,y)\in R_m} \prod_{k=1}^{K} c_k^{\mathbb{1}\{y=k\}}$$

Откуда уже сам критерий H принимает вид (логарифмируем и обращаем на -1):

$$H\left(R_m\right) = \min_{\sum_k c_k = 1} \left(-\frac{1}{|R_m|} \sum_{(x,y)\in R_m} \sum_{k=1}^{K} \mathbb{1}\{y=k\} \log c_k \right)$$

Воспользуемся теоремой ККТ (благо задача регулярна) и выпишем Лагранжиан:

$$L(c,\lambda) = -\frac{1}{|R_m|} \sum_{(x,y)\in R_m} \sum_{k=1}^{K} \log c_k \cdot \mathbb{1}\{y=k\} + \lambda \sum_{k=1}^{K} c_k \to \min_{c_k}$$

Дифференцируя, получаем:

$$\frac{\partial}{\partial c_k} L(c,\lambda) = -\frac{1}{|R_m|} \sum_{(x,y)\in R_m} \mathbb{1}\{y=k\} \frac{1}{c_k} + \lambda = -\frac{p_k}{c_k} + \lambda = 0$$

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

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

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 (частотные вероятности получаются из внесения мощности множества $|R_m|$ под знак суммирования):

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

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.

5.3 Критерий Джини

Подойдем к проблеме с другой стороны. Вернемся к семинару по вероятностным основам машинного обучения и вспомним, что функция потерь вида MSE над метками классов и вероятностями классов для объекта соответствует критерию вероятностной природы лосса. Такая метрика называется **критерием Бриера (Brier score)**:

$$H(R) = \min_{\sum_k c_k = 1} \frac{1}{|R_m|} \sum_{(x,y)\in R_m} \sum_{k=1}^{K} (c_k - \mathbb{1}\{y = k\})^2$$

Записав ККТ и решив его для данной задачи мы получим, что $c_k \equiv p_k$, однако сам критерий информативности H примен новую форму:

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

Критерий Джини допускает и следующую интерпретацию: $H(R_m)$ есть математическое ожидание частоты неправильно классифицированных объектов в случае, если мы будем приписывать им случайные метки из дискретного распределения, заданного вероятностями $(p_1, p_2, \ldots, p_K)$

5.3.1 Задача

Рассмотрим вершину m и объекты R, попавшие в нее. Поставим в соответствие вершине m алгоритм $a(x)$, который выбирает класс случайно, причем класс k выбирается с вероятностью p_k. Покажите, что матожидание частоты ошибок этого алгоритма на объектах из R_m равно индексу Джини.

$$\mathbb{E}\,\frac{1}{|R_m|} \sum_{(x,y)\in R_m} \mathbb{1}\{y \neq a(x)\} = \frac{1}{|R_m|} \sum_{(x,y)\in R_m} \mathbb{E}\,\mathbb{1}\{y \neq a(x)\} = \frac{1}{|R_m|} \sum_{(x,y)\in R_m} (1 - p_y) =$$
$$= \sum_{k=1}^{K} \frac{\sum_{(x,y)\in R_m} \mathbb{1}\{y = k\}}{|R_m|} (1 - p_k) = \sum_{k=1}^{K} p_k(1 - p_k)$$

5.3.2 Максимизация функционала Джини

Заметим, что в функционале качества Q член для R_m не зависит от разбиения. Проанализируем оставшуюся часть, обозначив долю объектов класса k в вершине m через p_{mk}, для этого просто раскрыв $p(1-p) = p - p^2$:

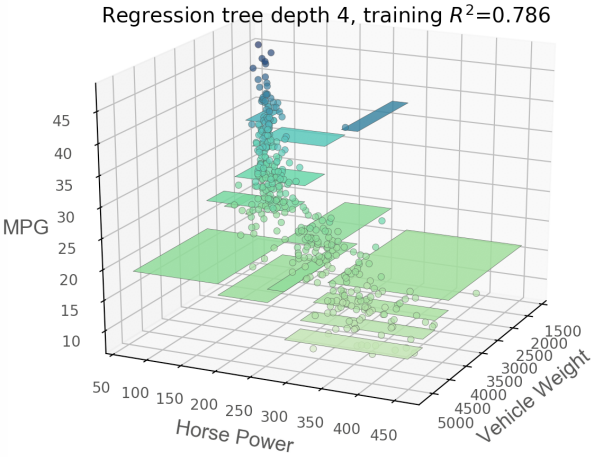
$$-\frac{|R_\ell|}{|R_m|} H(R_\ell) - \frac{|R_r|}{|R_m|} H(R_r) = -\frac{1}{|R_m|} \left(|R_\ell| - \sum_{k=1}^{K} p_{\ell k}^2 |R_\ell| + |R_r| - \sum_{k=1}^{K} p_{rk}^2 |R_r| \right) =$$
$$= \frac{1}{|R_m|} \left(\sum_{k=1}^{K} p_{\ell k}^2 |R_\ell| + \sum_{k=1}^{K} p_{rk}^2 |R_r| - |R_m| \right) = \{|R_m| \text{ не зависит от } j \text{ и } s\} =$$
$$= \sum_{k=1}^{K} p_{\ell k}^2 |R_\ell| + \sum_{k=1}^{K} p_{rk}^2 |R_r|$$

Запишем теперь в наших обозначениях число таких пар объектов (x, x_j), что оба объекта попадают в одно и то же поддерево, и при этом $y = y_j$. Число объектов класса k, попавших

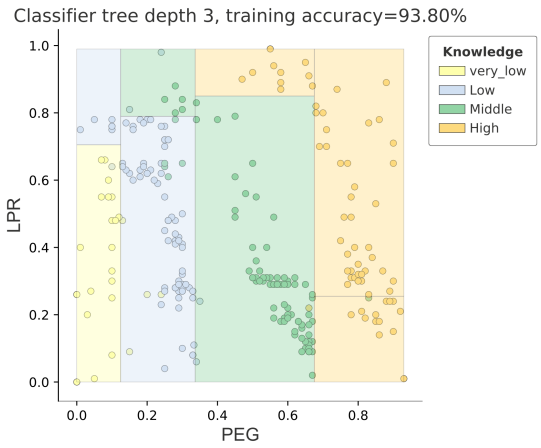
в поддерево ℓ, равно $p_{\ell k}|R_\ell|$; соответственно, число пар объектов с одинаковыми метками, попавших в левое поддерево, равно $\sum_{k=1}^{K} p_{\ell k}^2 |R_\ell|^2$. Интересующая нас величина равна

$$\sum_{k=1}^{K} p_{\ell k}^2 |R_\ell|^2 + \sum_{k=1}^{K} p_{rk}^2 |R_r|^2$$

Заметим, что данная величина очень похожа на полученное выше представление для функционала Джини. Таким образом, максимизацию критерия Джини можно условно интерпретировать как максимизацию числа пар объектов одного класса, оказавшихся в одном поддереве.



(a) двумерная регрессия в задаче по машинкам



(b) двумерная классификация в оценке знаний школьников

6 Выбор критерия

Рассмотрим простой пример с двумя классами. Пусть в текущую вершину попало 400 объектов первого класса и 400 объектов второго класса. Допустим, нужно сделать выбор между двумя разбиениями, одно из которых генерирует поддеревья с числом объектов $(300, 100)$ и $(100, 300)$ (первое число в паре — число объектов первого класса в подвыборке, второе — число объектов второго класса), а другое — с числом объектов $(200, 400)$ и $(200, 0)$. Оба разбиения дают ошибку классификации 0.25, но критерий Джини и энтропийный критерий отдадут предпочтение второму разбиению, что логично, поскольку правая вершина окажется листовой и сложность дерева окажется меньше.

В заключение отметим, что нет никаких четких правил для выбора функционала качества, и на практике лучше всего выбирать его с помощью кросс-валидации. Более того, это далеко не самый важный гиперпараметр — так, между критерием Джини и энтропийным критерием нет очень большой разницы с точки зрения результатов.

7 Критерии останова

Можно придумать большое количестве критериев останова, однако самые часто применяемые и доступные во всех профессиональных фреймворках, использующих деревья представлены ниже:

- 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 кроссвалидации.

8 Стрижка деревьев

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.

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 $\mathcal{R}(T)$ 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 $\mathcal{R}_\alpha(T)$, представляющий собой сумму исходного функционала $\mathcal{R}(T)$ и штрафа за размер дерева:

$$\mathcal{R}_\alpha(T) = \mathcal{R}(T) + \alpha|\Gamma|$$

где $|\Gamma|$ — число листьев в поддереве T, а $\alpha \geq 0$ — 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 $\mathcal{R}_\alpha(T)$ 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 кроссвалидации.

8.1 А нужен ли прунинг?

Ну как бы да, но не совсем. 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 ансамблях они либо **должны** (не удивляйтесь) быть переобучены (в случайных лесах), либо должны иметь очень небольшую глубину (в бустинге), из-за чего необходимость в стрижке отпадает.

Тем не менее существуют области, где применение чистых деревьев неотвратимо (там где нужен интерпретируемый, хотя и не рабочий ml). Поэтому методы стрижки могут пригодиться и на практике.

9 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 $B = \{u_1, \dots, u_q\}$, $|B| = q$. Разобьем множество значений на два непересекающихся подмножества: $B = B_1 \cup B_2$, и определим предикат как индикатор попадания в первое подмножество: $\beta(x) = [x_j \in B_1]$. Таким образом, объект будет попадать в левое поддерево, если признак x_j попадает в множество B_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. 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 $+1$:

$$\frac{1}{N_m(u_{(1)})} \sum_{x_i \in R_m(u_{(1)})} \mathbb{1}\{y_i = +1\} \leq \dots \leq \frac{1}{N_m(u_{(q)})} \sum_{x_i \in R_m(u_{(q)})} \mathbb{1}\{y_i = +1\}$$

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:

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

Именно такой подход используется в библиотеке Spark MLlib

10 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:

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

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 $|R_\ell|/|R_m|$ в левом поддереве и $|R_r|/|R_m|$ в правом. В дальнейшем веса можно учитывать, добавляя их как коэффициенты перед индикаторами $\mathbb{1}\{y_i = k\}$ 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.

11 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.

1. Материалы лекции по деревьям Е. А. Соколова
2. Лекции К. В. Воронцова
3. Yandex ML Handbook
4. Algorithms: ID3, C4.5, CART

Реализация решающего дерева

Введём обозначения:

- $X = (x_1, \ldots, x_\ell)$ — конечное упорядоченное множество объектов.
- $y = (y_1, \ldots, y_\ell)$ — конечное упорядоченное множество откликов.
- x_i^f — f-ый признак i-го объекта.
- $|z|$ — длина конечной последовательности z.
- I — критерий информативности (мера неопределённости, impurity).
- ΔI — информационный выигрыш (information gain).
- C — количество классов (для задачи классификации).

Сначала для фиксированного признака f определим функцию `get_threshold` нахождения наилучшего порога разбиения выборки по нему (см. Алгоритм 1).

Algorithm 1 Поиск наилучшего порога разбиения по признаку

function GET_THRESHOLD(X, y, f)
  if $|\{x^f | x \in X\}| = 1$ **then**
    return None, 0
  end if
  $\Delta I_{best} \leftarrow -\infty$
  for $th \in \{x^f | x \in X\}$ **do**
    $y_L \leftarrow \{y_i | x_i^f \leq th, i = \overline{1, |X|}\}$
    $y_R \leftarrow \{y_i | x_i^f > th, i = \overline{1, |X|}\}$
    $\Delta I \leftarrow I(y) - \frac{|y_L|}{|y|} I(y_L) - \frac{|y_R|}{|y|} I(y_R)$
    if $\Delta I > \Delta I_{best}$ **then**
      $\Delta I_{best} \leftarrow \Delta I$
      $th_{best} \leftarrow th$
    end if
  end for
  return th_{best}, ΔI_{best}
end function

С использованием функции `get_threshold` опишем функцию `get_best_split` нахождения наилучшего разбиения выборки (см. Алгоритм 2).

Функция `predict_leaf` в случае задачи классификации вычисляет частотное распределение откликов (см. Алгоритм 3), попавших в лист, а в случае задачи регрессии — арифметическое среднее этих откликов (см. Алгоритм 4).

Реализацию метода построения решающего дерева определим при помощи функции `fit` (см. Алгоритм 5).

Функция `predict` описывает вычисление для объекта x значения, предсказываемого для него с помощью дерева t (см. Алгоритм 6 для задачи классификации и Алгоритм 7 для задачи регрессии).

Algorithm 2 Поиск наилучшего разбиения

function GET_BEST_SPLIT(X, y)
  $\Delta I_{best} \leftarrow -\infty$
  for $f \in \overline{1, N_{features}}$ **do**
    $th, \Delta I \leftarrow$ GET_THRESHOLD(X, y, f)
    if $\Delta I > \Delta I_{best}$ **then**
      $\Delta I_{best} \leftarrow \Delta I$
      $th_{best} \leftarrow th$
      $f_{best} \leftarrow f$
    end if
  end for
end function
return f_{best}, th_{best}, ΔI_{best}

Algorithm 3 Вычисление предсказания в листе (задача классификации)

function PREDICT_LEAF(y)
  return $\left(\frac{\sum_{i=1}^{|y|}[y_i=c]}{|y|}\right)_{c=\overline{1,C}}$
end function

Algorithm 4 Вычисление предсказания в листе (задача регрессии)

function PREDICT_LEAF(y)
  return $\frac{\sum_{i=1}^{|y|} y_i}{|y|}$
end function

Algorithm 5 Построение решающего дерева

procedure FIT_ITERATION(X, y, t)
  if ($t[depth] \geq d_{\max}$) **or** ($|X| < N_{\min}$) **or** $|\{y_i \in y\}| < 2$ **then**
    $t[predictions] \leftarrow$ PREDICT_LEAF(y)
    return
  end if
  f, th, $\Delta I \leftarrow$ GET_BEST_SPLIT(X, y)
  if $\Delta I < \Delta I_{min}$ **then**
    $t[predictions] \leftarrow$ PREDICT_LEAF(y)
    return
  end if
  $t[feature] \leftarrow f$, $t[threshold] \leftarrow th$
  $X_L \leftarrow (x_i | x_i^f \leq th, i \in \overline{1, |X|})$, $y_L \leftarrow (y_i | x_i^f \leq th, i \in \overline{1, |X|})$
  $X_R \leftarrow (x_i | x_i^f > th, i \in \overline{1, |X|})$, $y_R \leftarrow (y_i | x_i^f > th, i \in \overline{1, |X|})$
  $t_L \leftarrow \{depth : t[depth] + 1\}$, $t_R \leftarrow \{depth : t[depth] + 1\}$
  $t[left] \leftarrow t_L$, $t[right] \leftarrow t_R$
  FIT_ITERATION(X_L, y_L, t_L)
  FIT_ITERATION(X_R, y_R, t_R)
end procedure
function FIT(X, y)
  $root \leftarrow \{depth : 0\}$
  FIT_ITERATION($X, y, root$)
  return $root$
end function

Algorithm 6 Предсказание с помощью решающего дерева (классификация)

function PREDICT(t, x)
  if $t[predictions] \neq null$ **then**
    return $\max_c(t[predictions]_c)$
  end if
  $f \leftarrow t[feature]$
  $th \leftarrow t[threshold]$
  if $x^f \leq th$ **then**
    return PREDICT($t[left]$, x)
  else
    return PREDICT($t[right]$, x)
  end if
end function

Algorithm 7 Предсказание с помощью решающего дерева (регрессия)

function PREDICT(t, x)
  if $t[predictions] \neq null$ **then**
    return $t[predictions]$
  end if
  $f \leftarrow t[feature]$
  $th \leftarrow t[threshold]$
  if $x^f \leq th$ **then**
    return PREDICT($t[left]$, x)
  else
    return PREDICT($t[right]$, x)
  end if
end function