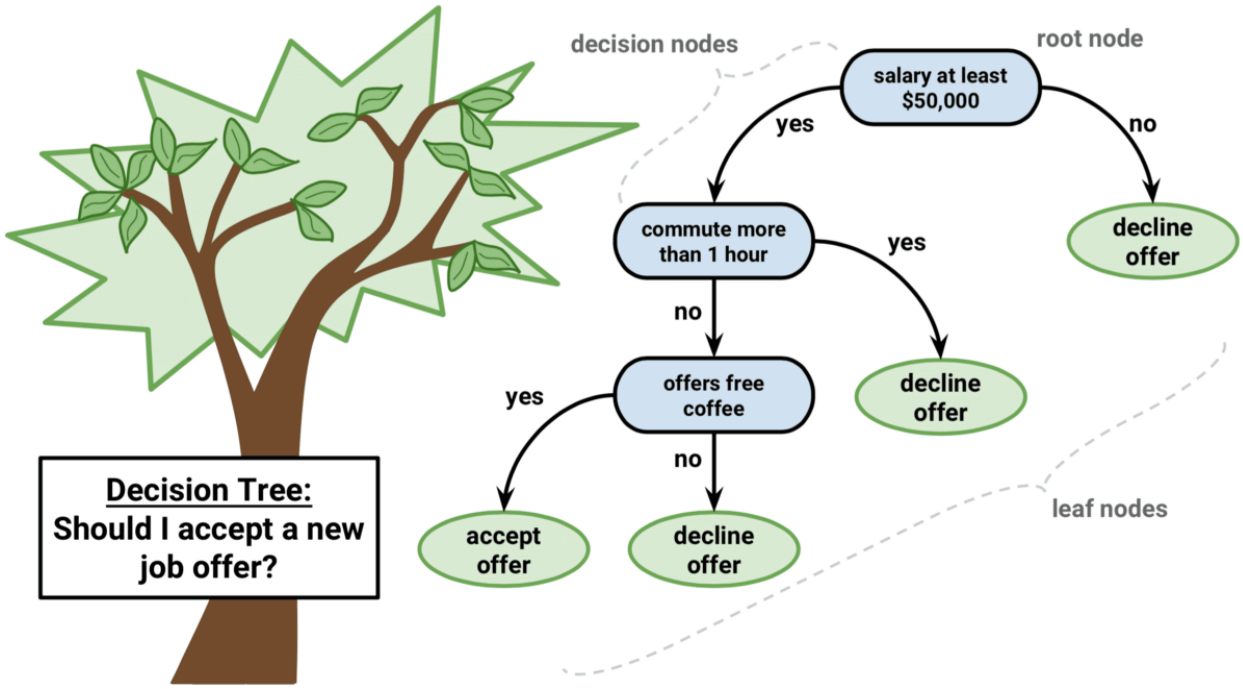
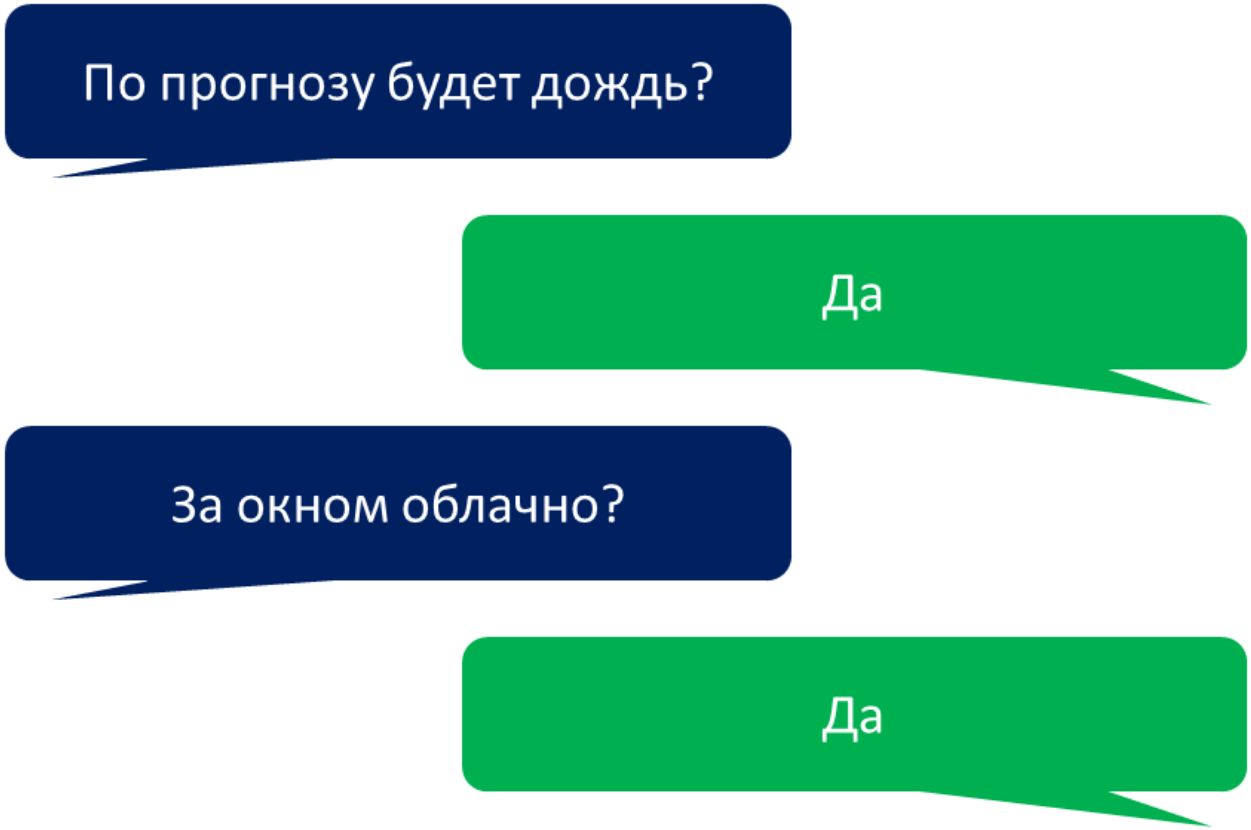
Лекция 7. Дерево решений (Decision Tree)

Давайте поговорим об одном из самых популярных методов в машинном обучении - деревья решений. Базово данное семейство алгоритмов используется для задачи классификации, но есть и некоторые адаптации для решения задачи регрессии.



В чем же основная суть алгоритма Дерево решений? На экране изображен хрестоматийный пример с решающим деревом по принятию оффера. Задается ряд вопросов к HR и после того, как прошли по всем ветвям ответов, можно принимать финальное решение.

Но давайте чуть более приближенно к принятию решений в повседневной жизни. Например, как вы определяете, стоит ли взять с собой зонт утром или нет. Скорее всего, вы посмотрите в прогноз погоды, затем выглянете в окно и если в прогнозе указано, что днем с высокой вероятностью будет дождь и за окном вы увидели, что собираются тучи, то скорее всего, вы зонт с собой возьмете. Так что же произошло, прежде чем вы приняли решение? Верно, вы задали несколько вопросов (приложению и погоде за окном) и на основании ответов приняли решение.



Конечно, это очень усеченный пример и на практике, особенно в профессиональной сфере, количество вопросов, которые нужно задать, чтобы прийти к решению, сильно больше. Таким образом, каждый раз задавая какой-то вопрос и предполагая бинарные ответы (да/нет, больше/меньше), мы как-будто бы строим некоторое дерево ответов и в итоге приходим к финальному решению. Ровно такую же логику и воспроизвели в модели!

Мы по-прежнему работаем с обучающей выборкой, и наша основная задача - выбрать и обучить такой алгоритм классификации, который бы хорошо работал не только на обучающей выборке, но и на новых данных.

Простейшим деревом решения является бинарное решающее дерево. В бинарном дереве есть два типа вершин:

Соединена с двумя дочерними вершинами (внутренняя)

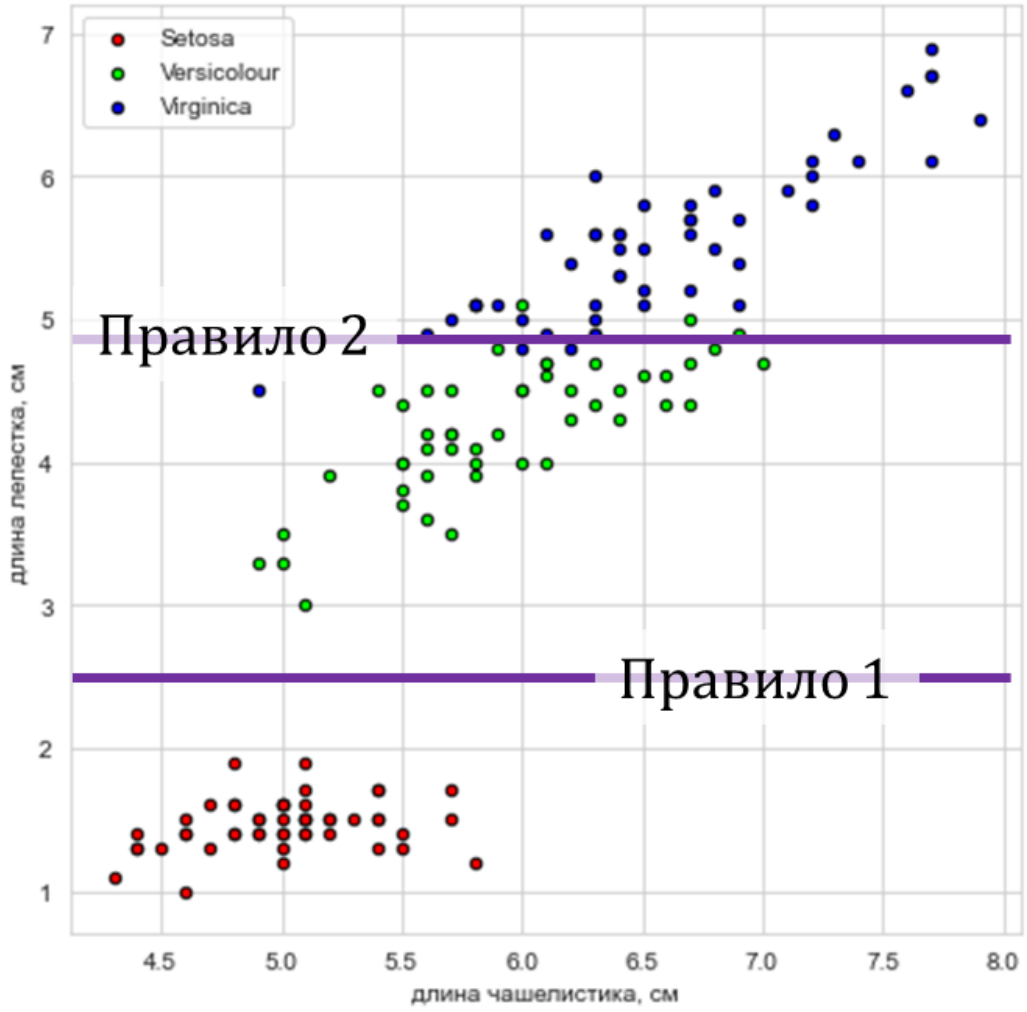
Не соединена ни с одной вершиной (листовая)

Получается, что вершины бывают двух типов: внутренние и листовые. Во внутренней вершине находится предикат, который на основании признаков объекта определяет по какой ветви дерева объект пойдет дальше - это тот самый вопрос, который мы задаем и получаем от системы ответ.

$$B(x_i, value) = [x_i \leq value]$$

Это будет происходить до тех пор, пока объект не доберется до листовой вершины, цель которой - присвоить объекту метку, которая в этой вершине находится. А как формировать предикаты, т.е. предикаты, на основании которых происходит деление на подмножества? Эти предикаты мы будем искать

на основании нашего исходного датасета. Когда признаки бинарны, то и предикат искать просто: левая ветвь это первое значение признака, правая - второе. Ситуация усложняется когда признаки вещественные, в этом случае нам необходимо найти некоторое пороговое значение, которое превратим в бинарное решающее правило: если значение признака больше или равно, чем некое пороговое значение θ, то левая ветвь, иначе - правая. В итоге получается следующая логика: мы имеем наш объект и построенное дерево (содержащее внутри себя множество бинарных решающих деревьев), и начинаем пропускать этот объект через наше дерево, начиная с корня. И на каждой внутренней вершине проверяем значение предиката на данном объекте. Если это значение равно 1, то объект переходит в правую ветвь, а в противном случае он переходит в левую. И так происходит до тех пор, пока объект не упрется в какую-то листовую вершину, в которой сидит метка класса, и тогда решающее дерево относит этот объект к этому классу. Вот так вот просто выглядит алгоритм принятия решения.



Нагляднее всего проиллюстрировать на примере. За основу возьмем датасет, где хранится информация о длине и ширине лепестка и чашелистика 3 сортов Ириса. Возьмем визуализацию в осях длина чашелистика и длина лепестка. Глазами видно, что классы очень хорошо разделимы. Первое правило сформировать очень просто: выделим в листовую вершину объекты, у которых длина лепестка менее 2.5, это наш X_0 (сорт **Setosa**, точки красного цвета). Теперь мы сформировали X_1 (точки зеленого и синего

цветов) и следующим шагом нам необходимо разделить его на подмножества. Теперь давайте отделим эти два класса следующим правилом: длина лепестка меньше 4.9. При помощи предиката мы смогли разделить наше двумерное пространство на 3 части, которые не пересекаются между собой, в этом ключевая особенность этого алгоритма: объект однозначно классифицируется. Эти правила называются логическими закономерностями, и часто они характеризуются тем, что каждое такое правило выделяет какую-то область в пространстве объектов, в которой находятся объекты либо только одного какого-то класса, либо преимущественно объекты одного класса, вот как в данном случае получилось с красными точками — одним из видов ириса, который на первом же шаге надежно отделился.

Давайте теперь более формально опишем что же такое решающее дерево. Исходно имеем бинарное дерево:

- у каждой внутренней вершины есть предикат

$$P_v : X \to \{0, 1\};$$

- у каждой листовой вершины v есть прогноз метки $C_v \in y$, где y — область значений целевой переменной (в случае классификации листу может быть также приписан вектор вероятностей классов).

В ходе предсказания осуществляется проход по этому дереву к некоторому листу. Для каждого объекта выборки x движение начинается из корня. В очередной внутренней вершине v проход продолжится в правую ветвь, если $P_v(x) = 1$, и влево, если $P_v(x) = 0$. Проход продолжается до момента, пока не будет достигнут листовая вершина, на которой объекту будет присвоена метка..

В внутренних вершинах используется простое правило:

$$P_v(x_i, value) = [x_i \leq value]$$

Из структуры дерева решений следует несколько интересных свойств:

- Дерево решений может работать только в диапазоне значений обучающей выборки;
- Для поиска оптимального решения невозможно использовать градиентные методы оптимизации из-за невозможности поиска градиента на кусочно-линейном функционале;
- Дерево склонно к переобучению, если задать слишком большую глубину: т.е. получится, что в каждой листовой вершине будет находиться по одному объекту и модель не будет обладать обобщающей способностью;
- Алгоритм очень интерпретируемый: всегда можно выписать все предикаты в виде правил, это очень любят заказчики, т.е. те люди, которые не погружаются в математику алгоритмов, но по долгу службы вынуждены принимать решения.

Алгоритм построения дерева решений

С общей концепцией работы алгоритма разобрались, теперь необходимо понять, а как именно выбираются признаки, на основании которых будет формироваться предикат во внутренней вершине дерева? Существует большое множество подходов как именно построить дерево решений, но мы поговорим о некотором стандарте при решении этой задачи - алгоритм **ID3 (Induction of Decision Tree)**. Этот подход был разработан в 1986 году Россом Куинланом и у него есть некоторые важные особенности:

1. Этот подход работает только с категориальными данными и не может работать с числовыми данными

2. Это рекурсивный алгоритм
3. При обучении он использует критерий информационной энтропии - об этом мы поговорим чуть позднее.

Принцип работы ID3:

Давайте проговорим общий принцип работы ID3-подобных алгоритмов. Пусть X-исходное множество обучающей выборки, X_m-множество объектов текущего листа. Отметим, что на первом шаге $X_m = X$. Давайте в общем смысле опишем каким образом происходит работа одного из самых популярных алгоритмов построения деревьев **ID3.**

1. Создаём вершину v
2. Если выполнен *критерий остановки* $StopCrit(X_m)$, то останавливаемся, объявляем эту вершину листовой и ставим ей в соответствие ответ $Res(X_m)$, после чего возвращаем её.
3. Если критерий остановки не выполнен, то находим предикат, который определит наилучшее разбиение текущего множества объектов X_m на две подвыборки X_0 и X_1, максимизируя *критерий ветвления* $Branching(X_m, i, value)$.
4. Далее рекурсивно повторим процедуру.

Как вы уже заметили, внутри алгоритма есть несколько гиперпараметров, которые необходимо задать до того, как будет запущена модель:

- $Res(X_m)$, вычисляющая ответ для листа по попавшим в него объектам из обучающей выборки, может быть, например:
  - для классификации — меткой самого частого класса
  - для регрессии — средним, медианой или другой статистикой;
- $StopCrit(X_m)$ — функция, которая решает, нужно ли продолжать ветвление или пора остановиться.
  - Для остановки можно использовать, например, глубину дерева, или же однородность меток внутри листовой вершины
- Критерий ветвления $Branching(X_m, i, value)$ -математический фундамент данного алгоритма. Обычно эта функция количественно оценивает, насколько улучшится некоторая финальная метрика качества дерева в случае, если получившиеся два узла будут листовыми, по сравнению с ситуацией, когда текущая внутренняя вершина будет листовой. По причине разнообразия этих критериев и существует большое количество методов построения решающих деревьев.

Общих правил по выбору гиперпараметров не существует, но есть некоторые рекомендации, про которые мы и поговорим.

Критерии ветвления

Начнем с основного функционала любой модели машинного обучения, а именно с функции потерь, она стандартно обозначается как $L(y_i, c)$

Как мы и проговорили, при поиске оптимального предиката, мы сравниваем функционал для двух состояний: если мы текущую вершину оставляем узловой, и когда мы превращаем её во внутреннюю на основании предиката. Сравниваем мы это на основании функции потерь:

$$\frac{1}{|X_m|}\sum_{(x_i,y_i)\in X_m} L(y_i,c)$$

Минимизация этого функционала

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

обычно называют **информативностью**, или **impurity**. Чем она ниже, тем лучше объекты в листе можно приблизить константным значением.

Похожим образом можно определить информативность решающего пня. Пусть, как и выше, X_0— множество объектов, попавших в левую вершину, а X_1 — в правую; пусть также c_0и c_1- константы, которые предсказываются в этих вершинах. Тогда функция потерь для всего пня в целом будет равна

$$\frac{1}{|Xm|}(\sum_{x_i\in X_0)} L(y_i,c_0) + \sum_{x_i\in X_1} L(y_i,c_1)) = \frac{1}{|Xm|}(|X_0|\frac{1}{|X_0|}\sum_{x_i\in X_0} L(y_i,c_0) + |X_1|\frac{1}{|X_1|}\sum_{x_i\in X_1} L(y_i,c_1))$$

Тогда минимальная сумма:

$$\frac{|X_0|}{|X_m|}H(X_0) + \frac{|X_1|}{|X_m|}H(X_1)$$

С общим видом критерия ветвления разобрались, а как же он будет выглядеть для прикладных задач? Отдельно отметим, что критериев существует большое количество, мы рассмотрим лишь часть из них, но самые популярные.

Классификация

Misclassification Error

Пусть в нашей задаче k классов, а q_k — доля объектов класса k в текущей вершине X_m:

$$q_k = \frac{1}{|Xm|}\sum_{(x_i,y_i)\in X_m} L(y_i,c)$$

Чаще всего функция потерь для классификации:

$$L(y_i,c) = I[y_i \neq c]$$

Упрощенно предположим, что листовая вершина в качестве ответа возвращает метку класса, тогда

$$H_{MisErr}(X_m) = min_{c\in y}\frac{1}{|Xm|}\sum_{(x_i,y_i)\in X_m} I[y_i \neq c]$$

Тогда выражение для информативности будет выглядеть так (для случая, когда в листовой вершине k_{max}:

$$H_{MisErr}(X_m) = \frac{1}{|Xm|}\sum_{(x_i,y_i)\in X_m} I[y_i \neq k_{max}] = 1 - p_{k_{max}}$$

Критерий Джини

Мы уже рассматривали, что во многих прикладных задачах выгоднее прогнозировать не конкретную метку класс, а вероятность принадлежности объекта классам $\{c_1, c_2, ..., c_k\}$. Теперь вопрос с функцией потерь, как её посчитать? В рассматриваемом критерии предлагается считать MSE для вероятностей:

$$H_{\text{Джини}}(X_m) = min_{\sum_k c_k = 1}\frac{1}{|Xm|}\sum_{(x_i,y_i)\in X_m}\sum_{l=1}^{k}(c_l - I[y_i = l])^2$$

Оптимум - вектор c, состоящий из локальных частот классов $\{p_1, p_2, ..., p_k\}, p_i = \frac{1}{|Xm|}\sum_i I[y_i = k]$. Объединим выражения:

$$H_{\text{Джини}}(X_m) = \sum_{l=1}^{k} p_l(1 - p_l)$$

$H_{\text{Джини}}(X_m)$ - математическое ожиданию числа неправильно классифицированных объектов в случае, если мы будем приписывать им случайные метки из дискретного распределения, заданного вероятностями $\{p_1, p_2, ..., p_k\}$.

Регрессия

MAE

Минимизация абсолютной ошибки

$$L(y_i, c) = |y_i - c|$$

Для минимизации абсолютной ошибки необходимо минимизировать медиану целевого признака, тогда

$$H_{MAE}(X_m) = \frac{1}{|Xm|}(y_i - y_{median})$$

MSE

Минимизация среднеквадратичной ошибки

$$L(y_i, c) = (y_i - c)^2$$

Информативность листа будет выглядеть следующим образом:

$$H_{MSE}(X_m) = \frac{1}{|Xm|}min_{c\in Y}\sum_{(x_i,y_i)\in X_m}(y_i - c)^2$$

Теперь нам нужно опорное значение (прогноз константой), в этом случае MSE среднего:

$$c = \frac{\sum y_i}{|X_m|}$$

Объединим с предыдущей формулой:

$$H_{MSE}(X_m) = \sum_{(x_i, y_i) \in X_m} \frac{(y_i - \bar{y})^2}{|X_m|}, \text{ где } \bar{y} = \frac{\sum_i^N y_i}{|X_m|}$$

Оценка значения в каждом листе — это среднее, а предикаты минимизируют дисперсию в узловых вершинах.

Одной из частых проблем обучения моделей машинного обучения является наличие пропусков в данных. И оказывается, решающие деревья очень хорошо и эффективно обходят проблему пропусков в данных. Рассмотрим что делать в случае с пропусками на этапе обучения модели. Когда некоторые объекты из-за пропусков не могут участвовать в вычислении критерия ветвления, то давайте просто уберем его из расчета и вместо этого посчитаем в каждой внутренней вершине, как часто объекты в этой внутренней вершине уходят в левую и правую ветви. Самое простое - вычислить оценку вероятности каждой из ветвей. Таким образом получена условная вероятность каждого класса для данного объекта, который дошел до каждой листовой вершины. Проще говоря, на основании частотных расчетов, мы получили оценку условной вероятности. А теперь этап промышленной работы модели: нам нужно классифицировать объект: начинаем пропускать объект через предикаты решающего дерева, и в какой-то момент он наталкивается на вершину, в которой предикат не может быть вычислен. Тут мы как раз и будем использовать величину условной вероятности. Мы можем определить значение условной вероятности класса для данного объекта в внутренней вершине через условные вероятности этого класса для этого объекта в левой и правой ветвях.

Подытожим

Решающее дерево - логический алгоритм, который при обучении формирует набор правил, которые на основании предикатов принимают решение по какой ветви направить анализируемый объект, который таким образом спускается до финальной листовой вершины. Мощным преимущество этого алгоритма является интерпретируемость: мы можем изобразить все правила в виде древовидной структуры и проанализировать как именно выглядит модель. Конечно, это применимо для деревьев небольшой глубины, но тем не менее, довольно часто это является ключевым фактором при выборе модели.

Одним из основных алгоритмов построения дерева является **ID3**. Само по себе построение дерева решения основывается на жадном алгоритме:

1. Создаём вершину
2. Если выполнен любой из *критериев остановки* , то прекращаем работу алгоритма, присваиваем этой вершине метку листовой и ставим ей метку соответствующего класса, после чего возвращаем её.
3. В противном случае: находим предикат (иногда ещё говорят *сплит, правило*), который определит наилучшее разбиение текущего множества объектов S на две подвыборки X_0 и X_1, максимизируя при этом *критерий ветвления* .
4. Для X_0 и X_1 рекурсивно повторим процедуру.

Это очень гибкий метод, мы можем по-разному задавать множество предикатов, из которого мы выбираем предикаты во внутренних вершинах. Вплоть до того, что это могут быть какие-то другие модели классификации, главное, чтобы они возвращали ответ «да» или «нет». Мы можем приспособить этот алгоритм для работы с данными с пропусками. Важно отметить, что у него не бывает отказов от классификации, любой объект так или иначе дойдет до какой-то листовой вершины и будет классифицирован. Т.к. ID3 работает по принципу жадного алгоритма, а это означает, что на каждом шаге ветвления мы принимаем локально оптимальные решения. Конечно, при помощи брут форса или полного перебора можно найти более оптимальное дерево, что с точки зрения структуры (количество узлов), так и с точки зрения точности. Но с учетом привычных размеров датасетов, которые исчисляются дестятками, сотнями тысяч записей, метод полного перебора тут реализовать невозможно. Поэтому, с учетом ограничений по вычислительным мощностям и времени, алгоритм, основанный на принятии локально-оптимальных решений, является наиболее привлекательным. Проще говоря, решающие деревья — это одна большая эвристика для решения NP-полной задачи, практически лишённая какой-либо стройной теоретической подоплёки. Также одним из возможных недостатков данного алгоритма является риск переобучения - в случае, если мы не ограничим глубину дерева, т.е. количество возможных предикатов, то вполне вероятна ситуация, когда в каждом из листовых узлов будет находиться по одному объекту и в этом случае ни о какой обобщающей способности алгоритма говорить уже не приходится.