Введение в машинное обучение

Национальный исследовательский университет "Высшая школа экономики"
Yandex School of Data Analysis

Неофициальный конспект по курсу.

2 апреля 2021 г.

1 Решающие деревья

1.1 Описание метода

Решающие деревья — это целый класс методов машинного обучения.

Эти методы предназначены исходно для решения задач классификации, очень редко для задач регрессии. Изначально решающие деревья возникли как попытка **формализовать человеческое мышление** при принятии решений.

Например, решающим деревом мы можем описать рассуждения врача, который за 5-6 вопросов отбирает нужные признаки и может поставить некоторый диагноз. При этом какие-то признаки исключают друг друга и целые группы симптомов, отсюда и получается *дерево*.
Конечно, на практике не все так просто, но все же.

1.2 Задача классификации (обучение с учителем)

Напомним, что в общем случае мы занимаемся задачами обучения с учителем:

Задача восстановления зависимости $y : X \to Y$, $|Y| < \infty$
по точкам обучающей выборки $(x_i, y_i), i = 1, \dots, l$.

Дано: векторы $x_i = (x_i^1, \dots, x_i^n)$ — объекты обучающей выборки
$y_i = y(x_i)$ — классификации, ответы учителя, $(i = 1, \dots, l)$

$$\begin{pmatrix} x_1^1 & \dots & x_1^n) \\ \dots & \dots & \dots \\ x_l^1 & \dots & x_l^n) \end{pmatrix} \xrightarrow{y*} \begin{pmatrix} y_1 \\ \dots \\ y_l \end{pmatrix}$$

Найти: функцию $a(x)$, способную классифицировать объекты произвольной тестовой выборки $\tilde{x}_i = (\tilde{x}_i^1, \dots, \tilde{x}_i^n)$, $i = 1, \dots, k$:

$$\begin{pmatrix} \tilde{x}_1^1 & \dots & \tilde{x}_1^n) \\ \dots & \dots & \dots \\ \tilde{x}_k^1 & \dots & \tilde{x}_k^n) \end{pmatrix} \xrightarrow{a?} \begin{pmatrix} a(\tilde{x}_1) \\ \dots \\ a(\tilde{x}_k) \end{pmatrix}$$

При этом в нашей задаче мы будем строить *алгоритм классификации* в виде **дерева**.

1.3 Определение бинарного решающего дерева

Бинарное дерево — это ациклический граф, в котором есть два типа вершин: либо вершина соединена с двумя дочерними (*внутренний тип*), либо вершина не соединена ни с одной (*листовой/терминальный тип*).

Бинарное решающее дерево — алгоритм классификации $a(x)$, задающийся бинарном деревом. В нем определена *дополнительная* информация, связанная с каждой внутренней вершиной и каждой лиственной вершиной:

1) $\forall v \in V_{\text{внутр}} \to$ предикат $\beta_v : X \to \{0, 1\}$, $\beta_v \in \mathscr{B}$,

2) $\forall v \in V_{\text{лист}} \to$ имя класса $c_v \in Y$,

где $\mathscr{B}$ — множество бинарных признаков или предикатов
(например, вида $\beta(x) = [x^j \geqslant \theta_j]$, $x^j \in \mathbb{R}$)

Предикат β_v (для **внутренней** вершины v) способен, взяв объект, сказать, *«дальше мы отправим его в левую ветвь, или в правую ветвь дерева?»*.

В каждой из **листовой** вершин мы будем записывать *имя* или *метку* одного из классов (элемент множества ответов Y). То есть дошли до листа — гарантируем (в меру своих возможностей) отношение объекта к определенному классу.

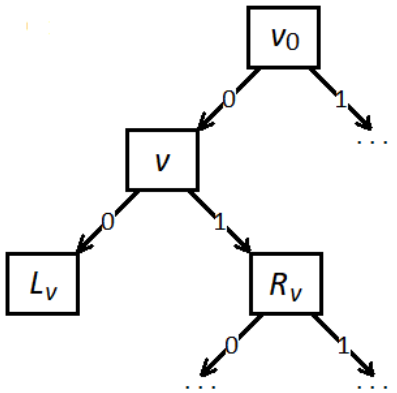
Предикаты для внутренних вершин мы берем из некоторого множества бинарных предикатов $\mathscr{B}$.

В простейшем случае, если все признаки *бинарны* — это будут просто исходные признаки.

Если признаки *вещественные*, то мы будем строить предикаты в виде бинарного условия *«значение признака меньше/больше некоторого порогового значения θ_j»* (см. пример выше).

Принятие решений нашим решающим бинарным деревом в итоге, на самом-то деле, можно описать достаточно простым алгоритмом спуска по дереву:

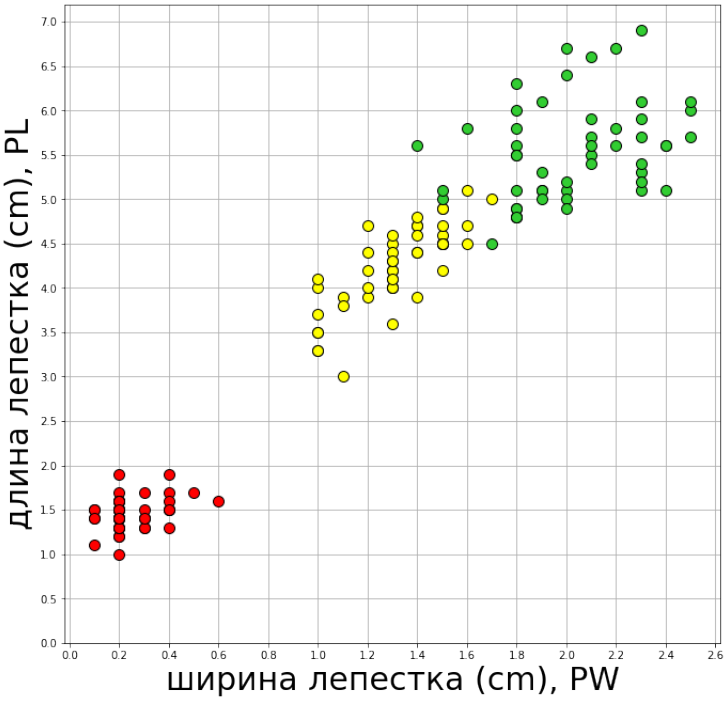
1: $v := v_0$;
2: пока $v \in V_{\text{внутр}}$
3: если $\beta_v(x) = 1$ то
4: переход вправо: $v := R_v$;
5: иначе
6: переход влево: $v := L_v$;
7: вернуть c_v.



Нам достаточно пустить объект по этому дереву, начиная с корня. В каждой вершине мы проверяем значение предиката на данном объекте, переходим в нужную ветвь, и так идем до некоторой листовой вершины, которая и говорит нам о классе объекта.

1.3.1 Пример. Задача Фишера

Задача Фишера (классическая задача) о классификации цветков ириса на 3 класса. В выборке по 50 объектов каждого класса, имеется 4 признака:

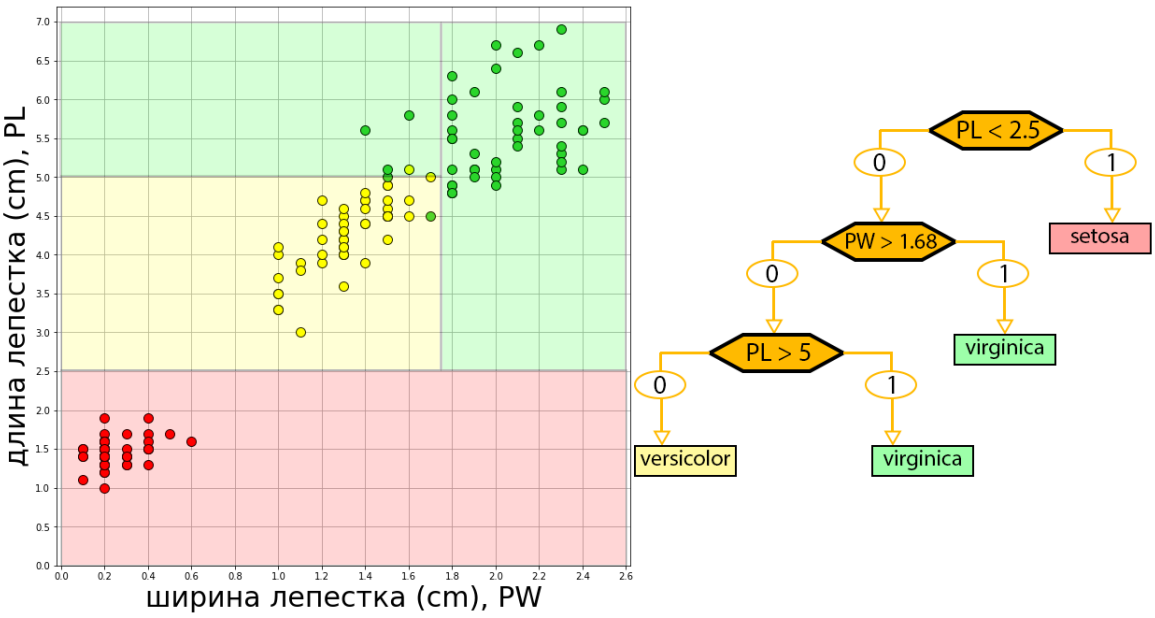


На графике:
в осях двух самых информативных признаков (из 4)
два класса разделились без ошибок, на третьем 3 ошибки.

Гладя на эту картинку можно в принципе вручную построить **решающее бинарное дерево**:

1. Отсекаем тот класс, который лежит в нижней области — $PL < 2.5$;
2. Отсекаем правую часть множества зеленых точек — $PW > 1.68$;
3. Осталось отделить желтые от нескольких зеленых. Заметим, что для желтых (большинства выполняется условие $PL < 5$.

Таким образом:



Итого 3 шага для того, чтобы представить алгоритм классификации в виде решающего дерева. Можно заметить, что при таком алгоритме у нас получилось 3-4 ошибки.

Замечание. Существует и альтернативный способ представления решающего дерева в виде списка правил:

setosa $r_1(x) = [PL \leqslant 2.5]$

virginica $r_2(x) = [PL > 2.5] \wedge [PW > 1.68]$

virginica $r_3(x) = [PL > 5] \wedge [PW \leq 1.68]$

versicolor $r_4(x) = [PL > 2.5] \wedge [PL \leq 5] \wedge [PW < 1.68]$

Дерево нам разделило пространство двух признаков на 4 не пересекающиеся области, а их мы можем описать условиями, которые их и ограничивают.

Такие правила называются **логическими закономерностями**, и часто они характеризуются тем, что каждое такое правило выделяет какую-то область в пространстве объектов, где содержатся объекты только одного класса или преимущественно одного класса.

1.4 Алгоритм построения решающего дерева

- **Жадный алгоритм построения дерева ID3**

Классическая процедура для построения решающего дерева **ID3 — Induction of Decision Tree**:

1: LearnID3 ($U \subseteq X^l$);

2: если все объекты из U лежат в одном классе $c \in Y$ то

3: вернуть новый лист $v, c_v := c$;

4: найти предикат с максимальной информативностью:
$\beta := arg \max\limits_{\beta \in \mathscr{B}} I(\beta, U)$;

5: разбить выборку на две части $U = U_0 \sqcup U_1$ по предикату β:
$U_0 := \{x \in U : \beta(x) = 0\}$
$U_0 := \{x \in U : \beta(x) = 1\}$

6: если $U_0 = \emptyset$ или $U_1 = \emptyset$ то

7: вернуть новый лист $v, c_v :=$ Мажоритарный класс(U);

8: создать новую внутреннюю вершину v: $\beta_v := \beta$;
Построить левое поддерево: $L_v :=$ LearnID3(U_0);
Построить правое поддерево: $R_v :=$ LearnID3(U_1);

9: вернуть v;

Разберем алгоритм по пунктам:

- Программа принимает на вход некоторую подвыборку обучающей выборки. В первый раз (процедура рекурсивная) мы подаем на вход всю выборку:

  1: LearnID3 ($U \subseteq X^l$);

- Если все объекты в подвыборке оказались одного класса, мы можем вернуть из процедуры лист (дальше классифицировать некуда; куда этот лист девается — об этом дальше):

  2: если все объекты из U лежат в одном классе $c \in Y$ то

  3: вернуть новый лист $v, c_v := c$;

- Дальше самый важный шаг. На этом этапе мы имеем выборку, в которой точно находятся объекты разных классов (мы только что проверили обратный случай). Попытаемся найти предикат, который максимально хорошо выделял бы нам объекты какого-то одного класса, или просто отделял их совсем от других классов.

  Для того, чтобы определить, насколько предикат информативный, существует множество **критериев информативности** – в данном случае обозначается как I. В общем-то именно из-за выбора критерия информативности возникает большое разнообразие методов построения решающих деревьев.

  4: найти предикат с максимальной информативностью:

  $\beta := arg \max\limits_{\beta \in \mathscr{B}} I(\beta, U)$;

- Так как мы строим бинарное дерево, и предикаты у нас тоже бинарные, мы можем спокойно разделить нашу выборку на две подвыборки — для одной предикат будет выполняться, для другой нет:

  5: разбить выборку на две части $U = U_0 \sqcup U_1$ по предикату β:

  $U_0 := \{x \in U : \beta(x) = 0\}$

  $U_0 := \{x \in U : \beta(x) = 1\}$

- На всякий случай проверим, а не получилась ли одна из подвыборок пустой. Если такое случится, это будет значит, что *несмотря на то*, что в подвыборке U были объекты *разных* классов, мы *не смогли* найти предикат, который как-либо разделил бы их.

  Нужно вводить новые предикаты, но это уже не задача алгоритма построения дерева, да и не всегда возможно (ошибки могут быть всегда), поэтому тут мы вернем новую листовую вершину, которой присвоим преобладающий в этой подвыборке класс (которого больше всего).

  6: если $U_0 = \emptyset$ или $U_1 = \emptyset$ то

  7: вернуть новый лист $v, c_v :=$ Мажоритарный класс(U);

- К этому моменту мы можем гарантировать, что обе подвыборки не пустые, поэтому мы можем создать внутреннюю вершину, которой присвоим наш разделивший выборку предикат. К этой вершине мы можем построить две ветви, которые будут строиться рекурсивно из наших подвыборок:

  8: создать новую внутреннюю вершину v: $\beta_v := \beta$;

  Построить левое поддерево: $L_v :=$ LearnID3(U_0);

  Построить правое поддерево: $R_v :=$ LearnID3(U_1);

- Поддеревья готовы, вершина подготовлена, можем ее спокойно вернуть из процедуры:

  9: вернуть v;

1.4.1 Варианты критериев ветвления

Рассмотрим все же, какие бывают **критерии ветвления** *(информативности)*:

- **Критерий Джини:**

$$I(\beta, X^l) = \#\{(x_i, x_j) : \ (y_i = y_j) \text{ и } (\beta(x_i) = \beta(x_j))\}.$$

  Он показывает, сколько есть пар объектов, лежащих в одном и том же классе, которые вместе идут в одну из дочерних вершин. У этих объектов должны совпадать метки классов и значения предикатов.

  По сути критерий измеряет то, насколько часто объекты одних классов объединяются.

- **D-критерий В.И.Донского:**

$$I(\beta, X^l) = \#\{(x_i, x_j) : \ (y_i \neq y_j) \text{ и } (\beta(x_i) \neq \beta(x_j))\}.$$

  По сути просто противоположная критерию Джини стратегия — подсчитываем число пар объектов, которые лежали в разных классах и они ушли в разные ветви.

  По сути измеряет, насколько данный предикат обладает способностью разделять объекты разных классов.

- Энтропийный критерий:

$$I(\beta, X^l) = \sum_{c \in Y} h\left(\frac{P_c}{l}\right) - \frac{p}{l} h\left(\frac{p_c}{p}\right) - \frac{l-p}{l} h\left(\frac{P_c - p_c}{l-p}\right),$$

где $h(z) = -z log_2(z)$,
$P_c(X^l) = \#\{x_i : \ y_i = c\}$,
$p_c(X^l) = \#\{x_i : \ (y_i = c) \text{ и } (\beta(x_i) = 1)\}$,
$p(X^l) = \#\{x_i : \ \beta(x_i) = 1\}$,

Достаточно сложный, поэтому разбирать его не будем. Впрочем, на практике часто оказывается, что этот критерий очень похож на критерий Джини и работает примерно так же...

1.5 Обработка пропусков

При построении решающих деревьев часто появляется важная проблема – **пропуски в данных**.

Если возвращаться к примеру с врачом и пациентом, то пациент так и может ответить: *«Я не знаю»*. Так и мы будем обращаться к признаку объекта и просто не сможем получить ответ на свой вопрос.

И вот решающие деревья достаточно удобно и эффективно, как мы потом увидим, решают эту проблему.

Чтобы решить проблему с пропусками, нам понадобится сделать некоторые уточнения как на стадии обучения, так и на стадии классификации:

На стадии обучения:

- $\beta_v(x)$ не определено $\Rightarrow x_i$ **исключается** из U для $I(\beta, U)$.

  Таким образом "информативность"предиката будет вычисляться без этого *объекта* (важно, что именно объекта!, а не исключение признака для всех объектов).

- $q_v = \frac{|U_0|}{|U|}$ — оценка **вероятности** левой ветви (попадания в левую ветвь), $\forall v \in V_{\text{внутр}}$.

  Вероятность для правой ветви, очевидно, всегда находится как $1 - q_v$. Зачем это вообще нужно? Увидим дальше на этапе классификации.

- $P(y|\ x, v) = \frac{1}{|U|} \#x_i \in U : \ y_i = y$ для всех $v \in V_{\text{лист}}$.

  По сути это функция-интерпретация **условной вероятности** для листовых вершин — условная вероятность класса y для данного объекта x, который дошел до листовой вершины v. По сути просто доля объектов обучающей выборки, которые дошли до этой вершины и принадлежат классу y

На стадии классификации

Мы имеем объект x, который пропускаем по нашему дереву сверху-вниз. Его обработка с учетом пропусков производится следующим образом:

- $\beta_v(x)$ **не определено** $\Rightarrow$ **пропорциональное распределение**:

$$P(y|\ x, v) = q_v P(y|\ x, L_v) + (1 - q_v)P(y|\ x, R_v).$$

  То есть мы пропускаем объект *и в левое поддерево, и в правое*. По сути мы посчитаем вероятность **каждого класса** для данного объекта x, дошедшего до вершины v, **учитывая** то, с какой вероятностью он может попасть в левое и правое поддерево — **для этого** мы и рассчитывали q_v.

- $\beta_v(x)$ **определено** $\Rightarrow$ либо налево, либо направо:

$$P(y|\ x, v) = \big(1 - \beta_v(x)\big)P(y|\ x, L_v) + \beta_v(x)P(y|\ x, R_v).$$

  То есть значение предиката определено, и объект x однозначно идет либо налево, либо направо — не забываем, что $\beta_v(x) = \{0, 1\}$, поэтому "либо то, либо то"мы контролируем домножением на $1 - \beta_v(x)$ и $\beta_v(x)$ соответственно, так как они принимают значения $\{1, 0\}$.

  При этом мы все еще мыслим вероятностями, чтобы получить общий удобный случай.

- Окончательное решение — **наиболее вероятный класс**:

$$a(x) = arg\ \max_{y \in Y} P(y|\ x, v_0).$$

  В данном случае v_0 — корневая вершина. То есть в процессе всего нашего обхода мы в итоге посчитаем условные вероятности для каждого класса **для корневой вершины**, и на основе этого можем выбрать класс, к которому объект, очевидно, максимально вероятно принадлежит.

  Этот принцип называется **«принцип максимума апостериорной вероятности**.

Таким образом мы получаем полностью рабочую схему решающего дерева, включающего обработку пропусков.

1.6 Достоинства и недостатки дерева ID3

Достоинства:

- Интерпретируемость и простота классификации.
- Гибкость: можно варьировать множество $\mathscr{B}$ (множество предикатов).
- Допустимость разнотипных данных и данных с пропусками.
- Трудоемкость линейна по длине выборки $O(|\mathscr{B}|hl)$ (как по признакам, так и по количеству объектов).
- Не бывает отказов от классификации (опять же, из-за обработки пропусков).

Недостатки (вытекают из жадности)**:**

- Жадный ID3 переусложняет структуру дерева, и, как следствие, сильно переобучается.

  Решения, которые принимаются в каждой внутренней вершине дерева, локально оптимальны, но не оптимальны глобально. Полным перебором мы всегда можем найти более компактное дерево, вероятно, даже с меньшим числом ошибок.

- Фрагментация выборки: чем дальше v от корня, тем меньше статистическая надежность выбора β_v, c_v.

  Когда мы доходим до вершин, близких терминальным (листам), мы часто имеем дело с высокой *фрагментацией* выборки, так как туда доходит *небольшое* число объектов, и решения о выборе предиката или соотнесения с классом становятся *статистически ненадежными*.

- Высокая чувствительность к шумам, к составу выборки, к критерию информативности.

  Часто малое изменение выборки приводит к совершенно иному построению дерева: где-то выберется иначе предикат и по цепочке изменения будут идти по всему поддереву.

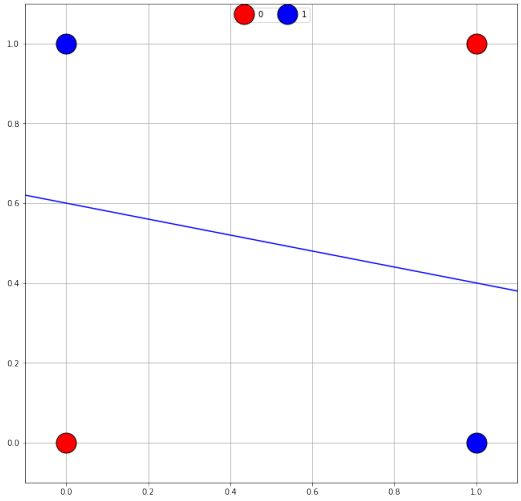
1.7 Способы устранения недостатков решающих деревьев

Только что мы уже сказали, что основная проблема решающего дерева ID3 в **переусложнении** дерева. Посмотрим явно на эту проблему на примере синтетической задачи классификации *«задача исключающего ИЛИ»*.

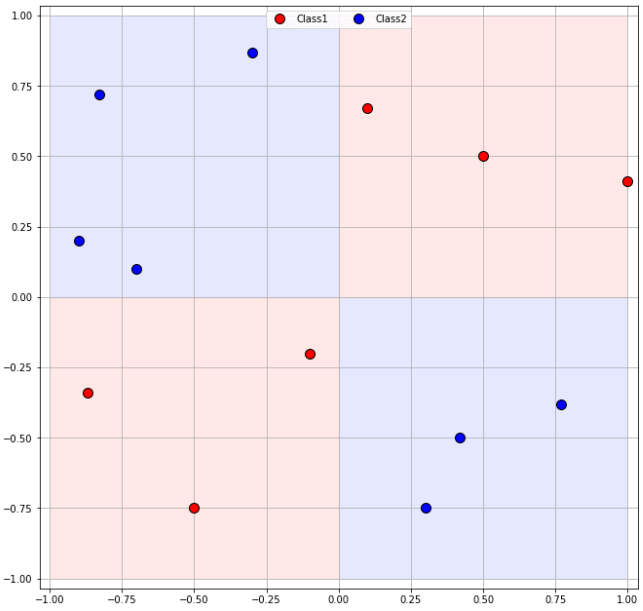
1.7.1 Пример. Задача исключающего ИЛИ

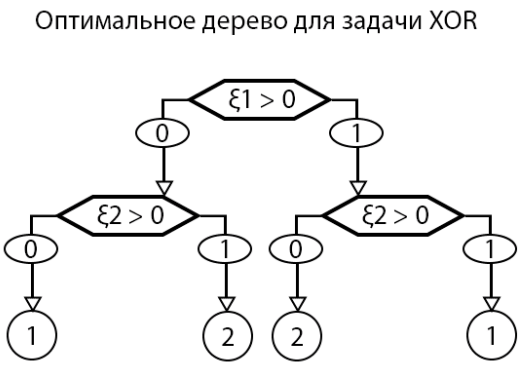
Данная задача не решается, например, линейным классификатором (то есть не может быть разделена прямой на два класса без ошибок), поэтому для нас она более чем подойдет.

Задача ставится однозначно: имеется логическая функция XOR. Если нарисовать точки значений в плоскости (a, b), где a — первый аргумент, а b — второй, то получим:

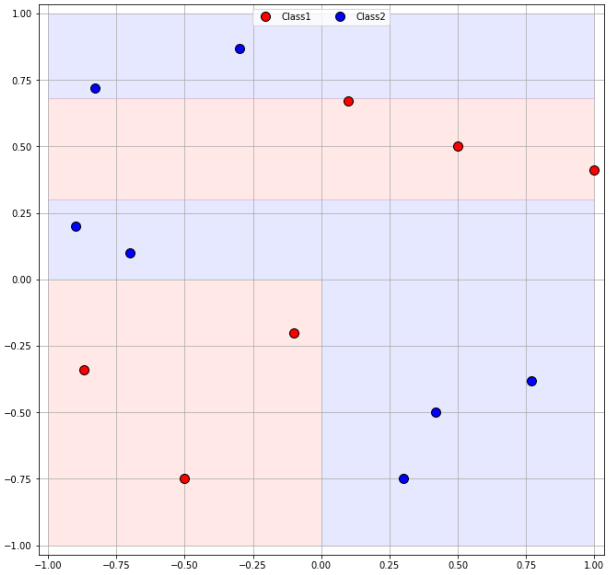


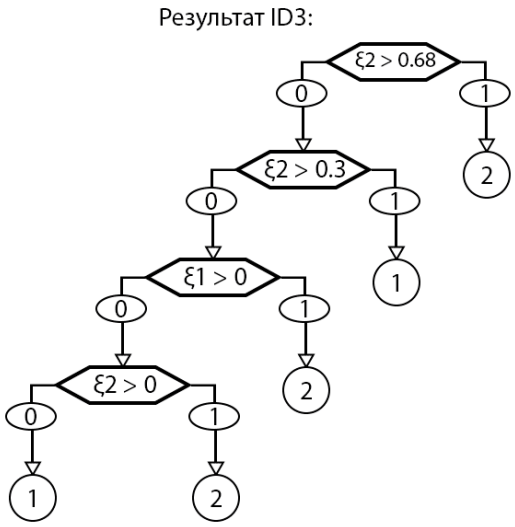
Линия проведена для примера, что одной линией эти случаи не разделить. Казалось бы, это все можно просто вычислить. Но если мы будем иметь какие-то объекты с вещественными признаками, которые тоже надо будет как-то классифицировать, то мы получим следующую картину:





Соответственно, по этой картине мы можем вполне выделить оптимальное решающее дерево (там же на рисунке, если вдруг не заметно). Однако алгоритм ID3 даст нам совершенно иную картину, и не сказать, что неправильную, но уж точно не оптимизированную:





Жадный алгоритм стремится отделить два класса, а не разделить выборки поровну, где и там, и там будет примерно одинаковое количество объектов из каждого класса, чтобы только на следующем уровне разделить их окончательно. Таким образом, двухуровневое дерево жадным алгоритмом в данном случае никак не построить.

Как бороться с такой ситуацией?

1.7.2 Усечение дерева (pruning). Алгоритм C4.5

Данный подход один из самый удачных и прижившихся. Он вошел в состав процедуры C4.5, которая имеет свободнодоступную реализацию (выложена в интернете). Алгоритм был предложен тем же автором, что и создал ID3.

Идея следующая: если наша проблема заключается в переобучении и переусложнении структуры дерева, значит нам нужна *контрльная выборка данных* для оценки *обобщающей способности.*

Можно было бы делать это в процессе постройки дерева, но в данном случае алгоритм работает уже над построенным деревом.

Мы разобьем обучающую выборку на две части: *обучающую* и *контрольную* — обычно это делают в соотношении 2:1, но можно любое, например:

X^k — независимая контрольная выборка, $k \approx 0.5l$.

Далее давайте опишем сам алгоритм, зная, что у нас уже есть построенное дерево и контрольная выборка:

1: для всех $v \in V_{\text{внутр}}$

3: если $S_v = \emptyset$ то

4: вернуть новый лист $v, c_v := (U)$;

5: число ошибок при классификации S_v четырьмя способами:

$r(v)$ — поддеревом, растущим из вершины v;

$r_L(v)$ — поддеревом левой дочерней вершины L_v;

$r_R(v)$ — поддеревом правой дочерней вершины R_v;

$r_c(v)$ — к классу $c \in Y$.

6: в зависимости от того, какое из них минимально:

сохранить поддерево v;

заменить поддерево v поддеревом L_v;

заменить поддерево v поддеревом R_v;

заменить поддерево v листом $c_v := arg \min\limits_{c \in Y} r_c(v)$.

Разберем данный алгоритм по пунктам:

- Пропускаем нашу контрольную выборку через дерево, и для каждой внутренней вершины сохраняем подмножество объектов, которые дошли до данной вершины.

  1: для всех $v \in V_{\text{внутр}}$

  2: $S_v :=$ подмножество объектов X^k, дошедших до v;

- Для каждой внутренней вершины постараемся понять, нужна она, или ее можно заменить на лист, или упростить поддерево, которое растет из этой внутренней вершины...

  Если оказалось, что множество контрольных объектов, дошедших до данной вершины *пусто*, то упростим ее и заменим её листовой вершиной, и выберем в этой листовой вершине тот класс, к которому относится большинство объектов *обучающей выборки* (контрольных же сюда дошло ноль, так что только по обучающей выборке нам и остается делать выводы).

  3: если $S_v = \emptyset$ то

  4: вернуть новый лист $v, c_v := (U)$;

- Дальше мы должны принять решение, чем можно было бы заменить эту внутреннюю вершину. Способов это выбрать много, но алгоритм усечения предусматривает только 4 варианта:

  – мы оставляем поддерево *без изменений*;

  – заменяем поддерево её же *левым* поддеревом;

  – заменяем поддерево её же *правым* поддеревом;

  – заменяем *терминальной* вершиной.

Выбор из этих четырех вариантов, на самом-то деле, достаточно простой. Надо просто для каждого варианта классифицировать объекты контрольной выборки (точнее объекты S_v, для данной вершины мы уже обрезали недошедшие), и просто выбрать тот вариант, где *число ошибок минимально.*

5: число ошибок при классификации S_v четырьмя способами:

$r(v)$ — поддеревом, растущим из вершины v;

$r_L(v)$ — поддеревом левой дочерней вершины L_v;

$r_R(v)$ — поддеревом правой дочерней вершины R_v;

$r_c(v)$ — к классу $c \in Y$.

6: в зависимости от того, какое из них минимально:

сохранить поддерево v;

заменить поддерево v поддеревом L_v;

заменить поддерево v поддеревом R_v;

заменить поддерево v листом $c_v := arg \min\limits_{c \in Y} r_c(v)$.

Таким образом мы можем вершину за вершиной упростить дерево. *Порядок* просмотра вершин зависит от реализации, в целом он не принципиален (в плане того, что алгоритм не сломается).

1.7.3 CART: деревья регрессии и классификации

CART — **C**lassification and **R**egression **T**ree.

В этом дереве тоже есть очень похожий на $ID3$ жадный рекурсивный алгоритм построения дерева, тоже применяются стратегии усечения, но алгоритм обобщен на случаи регрессии.

Для регрессии используется следующее обобщение: $Y = \mathbb{R}, c_v \in \mathbb{R}$.

А в качестве *критерия информативности* мы используем другой критерий. Самое простое и естественное — это использовать *критерий среднеквадратичной ошибки* (тот же, что и в методе наименьших квадратов):

Пусть U_v — по-прежнему является множеством объектов обучающей выборки, дошедших до вершины v.

Значения в терминальных вершинах — МНК-решение (усредним значение ответов на всех объектах):

$$c_v := \hat{y}(U_v) = \frac{1}{|U_v|} \sum_{x_i \in U_v} y_i$$

Критерий информативности — среднеквадратичная ошибка

$$I(\beta, U_v) = \sum_{x_i \in U_v} (\hat{y}_i(\beta) - y_i)^2,$$

где $\hat{y}_i(\beta) = \beta(x_i)\hat{y}(U_{v1}) + (1 - \beta(x_i))\hat{y}(U_{v0})$ — прогноз после ветвления β и разбиения $U_v = U_{v0} \sqcup U_{v1}$.

WARN: плохо понятая часть, need help! То есть выбор предиката осуществляется по тому, насколько хорошо он будет классифицировать то дерево, которое будет состоять из вершины и двух поддеревьев $v0$ и $v1$ (каждое из которых имеет свое среднее значение), в которые мы будем отправлять наши объекты x_i (выбор ветви зависит от результата предиката $\beta(x_i)$).

Так произойдет разбиение и процесс построения дерева пойдет рекурсивно.

В итоге дерево регрессии строит **кусочно-постоянную функцию**, потому что в каждом листе ответ, который будет выдавать этот алгоритм, является константой (среднее значение). Это означает, что дерево регрессии разделило все пространство объектов на какие-то куски, и в каждом куске выдается свое константное значение.

1.7.4 CART: критерий Minimal Cost-Complexity Pruning

Возвращаясь к вопросу оптимизации структуры дерева, для **CART** придуман очень эффективный критерий, который позволяет упростить структуру дерева.

Здесь можно выписать критерий, который состоит из двух частей. Первое — это среднеквадратичная ошибка. Второе — **штрафное слагаемое**. Таким образом получаем:

- Среднеквадратичная ошибка со штрафом за сложность дерева:

$$C_\alpha = \sum_{x_i=1}^{l} (\hat{y} - y_i)^2 + \alpha |V_{\text{лист}}| \to min$$

  По сути *штрафное слагаемое* в данном случае — это *количество листьев* в дереве, домноженное на какой-то коэффициент, который называется *константой регуляризации*.

Таким образом, мы можем выделить следующие моменты минимизации данной величины:

- При увеличении α дерево последовательно упрощается.

  Причем последовательность вложенных деревьев единственна (это строго доказываемый факт).

- Из этой последовательности выбирается дерево с минимальной ошибкой на тестовой выборке (Hold-Out).

- Для случая классификации используется аналогичная стратегия усечения, с критерием Джини.

2 Резюме

Преимущества решающих деревьев:

- интерпретируемость,
- допускаются разнотипные данные,
- возможность обхода пропусков;

Недостатки решающих деревьев

- переобучение,
- фрагментация
- неустойчивость к шуму, составу выборки, критерию;

Способы устранения этих недостатков:

- редукция,
- композиции (леса) деревьев (*в следующих лекциях*).