



Артем Груздев

Прогнозное моделирование в R и Python

Модуль 3. Построение деревьев решений
CART с помощью пакета R rpart



Москва — 2017

СОДЕРЖАНИЕ

Модуль 3 Построение деревьев решений CART с помощью пакета R rpart	3
Лекция 3.1 Знакомство с методом CART	3
3.1.1 Описание алгоритма	3
3.1.2 Неоднородность	6
3.1.3 Метод отсечения ветвей на основе меры стоимости-сложности с перекрестной проверкой	8
3.1.4 Обработка пропущенных значений	9
3.1.5 Иллюстрация работы метода CART на конкретных примерах	10
Лекция 3.2 Построение и интерпретация дерева классификации CART	29
3.2.1 Подготовка данных	29
3.2.2 Построение и интерпретация модели классификации	32
3.2.3 Работа с диаграммой дерева	39
3.2.4 Прунинг дерева классификации CART	48
3.2.5 Оценка качества модели	52
Лекция 3.3 Построение и интерпретация дерева регрессии CART	55
3.3.1 Подготовка данных	55
3.3.2 Построение и интерпретация модели регрессии	57
3.2.3 Прунинг и оценка качества дерева регрессии CART	59
Вопросы к модулю 3	65

Модуль 3 Построение деревьев решений CART с помощью пакета R rpart

Лекция 3.1 Знакомство с методом CART

Алгоритм метода CART был опубликован в 1984 году в книге «Classification and regression trees» («Деревья классификации и регрессии»), авторами которой являлись Лео Брейман (Калифорнийский университет в Беркли), Джером Фридман (Стэнфордский университет), Ричард Олшен (Калифорнийский университет в Беркли) и Чарльз Стоун (Стэнфордский университет).

Как уже ясно из названия, CART включает два вида анализа: деревья классификации для категориальных зависимых переменных и регрессионные деревья для количественных зависимых переменных. Эти виды анализа отличаются в деталях, но используют общий принцип. CART выполняет последовательные бинарные разбиения данных на основе выбранного критерия. В отличие от CHAID, в CART для выбора предиктора расщепления не применяются статистические критерии. Вместо этого в каждом узле при расщеплении данных используется предиктор, обеспечивающий наибольшее улучшение по выбранному критерию. Ключевой элемент в методе CART – это отсечение ветвей дерева, известный под названием «отсечение с минимизацией стоимости-сложности» (minimal cost-complexity pruning). Деревья, построенные с помощью метода CART, имеют тенденцию быть слишком большими, а результаты не повторяются с необходимой степенью устойчивости. Авторы метода пришли к выводу, что если разрешить построение максимально большого дерева, но затем отсечь его ветви, используя более сложный критерий, то в результате будет построено меньшее дерево лучшего качества. Построение дерева с последующим отсечением его ветвей стало основой метода CART.

3.1.1 Описание алгоритма

Алгоритм CART строит дерево, итеративно применяя к каждому узлу, начиная с корневого, процедуры выбора наилучшего расщепления предиктора, выбора наилучшего расщепления узла и остановки. В качестве критерия расщепления алгоритм использует уменьшение неоднородности, для категориальных зависимых переменных – уменьшение меры Джини, для количественных зависимых переменных – уменьшение девианса. Наилучшее расщепление – это расщепление, максимально уменьшающее меру Джини или девианс.

Этап 1. Выбор наилучшего расщепления предиктора

1. Алгоритм начинает с поиска наилучшего расщепления для каждого предиктора. Для количественных и порядковых предикторов алгоритм выполняет сортировку значений в порядке возрастания. Затем алгоритм разбивает предиктор по всем возможным точкам расщепления (разделяющим значениям). Если имеются 6 различных значений возраста, они будут упорядочены и для них будет созданы 5 точек расщепления.

Например, есть значения возраста 18, 35, 20, 16, 11, 10. Они будут упорядочены: 10, 11, 16, 18, 20 и 35. Будет рассмотрено пять расщепляющих значений:

$< 11 \text{ } s \geq 11$

$< 16 \text{ } s \geq 16$

$< 18 \text{ } s \geq 18$

$< 20 \text{ } s \geq 20$

$< 35 \text{ } s \geq 35$

Кроме того, часто применяется стратегия, когда в качестве возможной точки расщепления рассматривается среднее по каждой паре упорядоченных смежных значений. В данном случае точки расщепления будут выглядеть так:

$< 10,5 \text{ } s \geq 10,5$

$< 13,5 \text{ } s \geq 13,5$

$< 17 \text{ } s \geq 17$

$< 19 \text{ } s \geq 19$

$< 27,5 \text{ } s \geq 27,5$

Именно эта стратегия используется в пакете R `part`.

В каждой возможной точке расщепления переменной вся выборка наблюдений может быть разбита на два дочерних узла: левый и правый. Все наблюдения, у которых значение предиктора меньше разделяющего значения, относятся в левый дочерний узел. Все наблюдения, у которых значение предиктора больше или равно разделяющему значению, относятся в правый дочерний узел.

Для номинального предиктора его категории делятся всеми возможными способами на две группы.

2. Алгоритм вычисляет уменьшение неоднородности для каждого варианта расщепления (см. *раздел 3.1.2 «Неоднородность»*).

3. В качестве наилучшего расщепления предиктора алгоритма выбирается расщепление, лучше всего минимизирующее неоднородность.

Вышеописанные шаги повторяются для всех остальных переменных.

Этап 2. Выбор наилучшего расщепления узла

1. Из наилучших расщеплений предикторов, полученных на первом этапе, алгоритм выбирает расщепление предиктора, максимизирующее критерий расщепления для корневого узла или узла-потомка (проще говоря, из лучших расщеплений выбираем лучшее).

2. Алгоритм расщепляет узел, используя найденное наилучшее расщепление для него, если это позволяют правила остановки. Обратите внимание, что каждый предиктор может неоднократно использоваться для расщепления в ветви дерева. Например, может быть выполнено расщепление по переменной *Возраст* в значении 60 лет, а затем в узле-потомке снова может быть выполнено расщепление по этой переменной. Таким образом, могут моделироваться сложные зависимости между непрерывным предиктором и зависимой переменной, несмотря на то что выполняются только бинарные расщепления.

Этап 3. Остановка

Алгоритм проверяет, нужно ли прекратить построение дерева, в соответствии со следующими правилами остановки.

1. Если узел стал однородным, то есть все наблюдения в узле имеют одинаковые значения зависимой переменной, узел не разбивается.
2. Если при выполнении разбиения уменьшение ошибки модели становится меньше порогового значения штрафа за сложность, процесс построения дерева останавливается.
3. Если количество наблюдений в родительском узле меньше заданного пользователем минимума наблюдений в родительском узле, узел не разбивается.
4. Если минимальное количество наблюдений в терминальном узле меньше заданного пользователем минимума наблюдений в терминальном узле, узел не разбивается.
5. Если текущая глубина дерева достигает заданной пользователем максимальной глубины дерева, процесс построения дерева останавливается.

ПРИМЕЧАНИЕ

В пакете R `gpart` с помощью ряда параметров вспомогательной функции `gpart.control` можно изменить некоторые вышеперечисленные правила остановки:

- `sr` задает штраф за сложность, если разбиение уменьшает ошибку модели на значение, меньшее порогового значения `sr`, оно не принимается и дерево останавливается в росте;
- `minsplit` задает минимальное количество наблюдений в родительском узле перед расщеплением, по умолчанию 20;
- `minbucket` задает минимальное количество наблюдений в терминальном узле, по умолчанию используется округленное значение `minsplit/3`;
- `maxdepth` задает максимальную глубину дерева (количество уровней дерева, лежащих ниже корневого узла), по умолчанию равна 30.

3.1.2 Неоднородность

Критерий расщепления, используемый для построения дерева CART, называется **неоднородность (impurity)**.

Применительно к дереву классификации CART под неоднородностью понимается неоднородность распределения классов зависимой переменной в узлах-потомках. Однородным узлом является тот, в котором все наблюдения относятся к одному и тому же классу зависимой переменной, в то время как узел с максимальной неоднородностью содержит равное количество наблюдений во всех классах зависимой переменной. Допустим, есть узел, и он разбит на два класса. Максимальная неоднородность в узле будет достигнута при разбиении его на два подмножества по 50 примеров, а минимальная неоднородность – при разбиении на 100 и 0 примеров.

Наиболее популярная мера неоднородности для деревьев классификации – **мера Джини**. В основе меры Джини лежат возведенные в квадрат вероятности, с которыми наблюдения будут отнесены к каждому классу зависимой переменной.

Общая формула для вычисления меры Джини выглядит так:

$$Gini(t) = 1 - \sum_{k=1}^K p_k^2$$

где:

K – количество классов зависимой переменной;

k – класс зависимой переменной;

p_k – вероятность k -того класса зависимой переменной в t -ом узле.

Для бинарной зависимой переменной мера Джини принимает вид:

$$Gini(t) = 1 - p_1^2 - p_0^2$$

где:

p_1^2 – вероятность класса 1 (положительного класса) в t -ом узле;

p_0^2 – вероятность класса 0 (отрицательного класса) в t -ом узле.

Когда наблюдения в узле равномерно распределены по категориям, мера Джини принимает свое максимальное значение (для бинарной зависимой переменной максимальное значение меры Джини равно 0,5). Когда все наблюдения в узле принадлежат к одной и той же категории, мера Джини равна 0.

Узел с распределением (1, 0)	Мера Джини = $1 - 1^2 - 0^2 = 0$
Узел с распределением (0,5, 0,5)	Мера Джини = $1 - 0,5^2 - 0,5^2 = 0,5$
Узел с распределением (0,7, 0,3)	Мера Джини = $1 - 0,7^2 - 0,3^2 = 0,42$

Кроме меры Джини часто используется энтропия. Она вычисляется по формуле:

$$E(t) = -\sum_{k=1}^K p_k \times \log_2 p_k$$

Применительно к дереву регрессии CART под неоднородностью понимается степень разброса значений количественной зависимой переменной вокруг среднего значения. Более точно, речь идет о **среднеквадратичной ошибке** – сумме квадратов остатков (разностей между фактическими значениями зависимой переменной и ее средним значением) в конкретном узле, поделенной на количество наблюдений в этом узле.

$$MSE(t) = \frac{1}{n_t} \sum_i (y_{it} - \bar{y}_t)^2$$

где:

y_{it} – фактическое значение зависимой переменной для i -того наблюдения в t -ом узле;

\bar{y}_t – среднее значение зависимой переменной для t -ом узле;

n_t – количество наблюдений в t -ом узле.

При построении дерева CART расщепляет узел на дочерних узла по предиктору, который обеспечивает наибольшее уменьшение неоднородности, для этого неоднородность родительского узла сравнивается со взвешенным средним значением неоднородностей дочерних узлов:

$$\Delta i = i_P - \left(\frac{n_L}{n_P} i_L + \frac{n_R}{n_P} i_R \right)$$

где:

Δi – уменьшение неоднородности;

i_P – неоднородность родительского узла;

$\left(\frac{n_L}{n_P} i_L + \frac{n_R}{n_P} i_R \right)$ – взвешенное среднее значение неоднородностей дочерних узлов;

n_L – количество наблюдений в левом дочернем узле;

n_R – количество наблюдений в правом дочернем узле;

n_P – количество наблюдений в родительском узле;

i_L – неоднородность левого дочернего узла;

i_R – неоднородность правого дочернего узла.

Изменение неоднородности от родительского узла к дочерним узлам называется *улучшением* и отображается на диаграмме дерева.

3.1.3 Метод отсечения ветвей на основе меры стоимости-сложности с перекрестной проверкой

При использовании критерия неоднородности для построения дерева возникает следующая проблема. Увеличивая размеры дерева, почти всегда можно уменьшить неоднородность. Любое дерево будет иметь нулевую неоднородность, если оно построено достаточно большим. В частности, если в каждом терминальном узле имеется только одно наблюдение, то неоднородность равна нулю. По мере увеличения размеров дерево становится более сложным, а ошибка модели уменьшается. Однако в процессе построения дерева возникает момент, когда при расщеплении узла добавление переменной увеличивает сложность, но не уменьшает ошибку модели. Это ведет к построению излишне сложного, детализированного дерева, происходит переобучение модели. В итоге, несмотря на высокий процент правильных прогнозов, большие деревья из-за сложности дают статистически неустойчивые результаты. Поэтому необходимо найти баланс между сложностью и ошибкой модели.

Для решения этих проблем разработчики CART ввели меру стоимости-сложности, которая включает штраф, возрастающий с увеличением размера дерева. Эта функция для дерева (или его ветви) обычно выражается как

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

где $R(T)$ – ошибка модели, рассчитанная по тем же данным, по которым строилось дерево; α – коэффициент штрафа; $|T|$ – количество терминальных узлов дерева (или ветви) T .

Дерево большего размера будет иметь большую меру стоимости-сложности за счет слагаемого $\alpha|T|$. Для того, чтобы мера стоимости-сложности улучшилась, ошибка модели («стоимость» ошибки) должна уменьшиться в большей степени, чем штраф за сложность (в пакете `R` `graft` штраф за сложность регулируется параметром `cp`).

Мера стоимости-сложности была протестирована в качестве критерия построения дерева, однако авторы (Брейман, Фридман и др.) констатировали, что построенные таким способом деревья все еще не вполне удовлетворительны – они недостаточно стабильны. Решение этой проблемы привело, в свою очередь, к методу отсечения ветвей на основе критерия максимального уменьшения меры стоимости-сложности (`cost-complexity pruning`). Его суть сводится к следующему. Сначала строим максимально большое дерево (с небольшим числом наблюдений в узлах – от 1 до 5). Затем отсекаем у него ветви на основе меры стоимости-сложности. Выбираем простейшее дерево с наименьшим числом узлов,

ошибка которого находится в пределах одной стандартной ошибки от минимальной ошибки, достигнутой на этапе построения дерева. В качестве ошибки для дерева классификации берется ошибка классификации, а для дерева регрессии таковой будет сумма квадратов остатков.

В дальнейшем этот метод был вновь усовершенствован авторами. В ходе экспериментов было установлено, что управление отсечением и отбор модели необходимо осуществлять, оценивая качество модели не на обучающей выборке, а на контрольных блоках перекрестной проверки (Лео Брейман рекомендовал использовать 10-блочную перекрестную проверку). Мы выбираем дерево, которое дает наименьшую кросс-валидационную ошибку. Кросс-валидационной ошибкой является ошибка классификации (для дерева классификации) или сумма квадратов остатков (для дерева регрессии), усредненная по всем контрольным блокам перекрестной проверки. Метод отсечения на основе меры стоимости-сложности с перекрестной проверкой как раз и реализован в пакете `grat`.

3.1.4 Обработка пропущенных значений

В методе CART пропущенные значения обрабатываются с использованием переменных-суррогатов. Таким образом, если наблюдение имеет пропущенное значение в переменной, по которой осуществляется разбиение узла, то для выбора дочернего узла, к которому относится данное наблюдение, используется его значение для наилучшей переменной-суррогата. Наилучшей переменной-суррогатом является альтернативная предикторная переменная, дающая наиболее близкое (с использованием меры связи) разбиение к тому, которое дает исходный предиктор.

3.1.5 Иллюстрация работы метода CART на конкретных примерах

3.1.5.1 Дерево классификации

Предположим, есть данные по клиентам микрофинансовой организации и известно, выплатили они займ или нет (категориальная зависимая переменная *Просрочка*). Для удобства расчетов представим, что наш набор данных состоит всего из 5 наблюдений. В качестве потенциальных предикторов фигурируют две переменные: *Возраст* и *Пол*. Переменная *Пол* является номинальной, переменная *Возраст* является количественной. Необходимо классифицировать клиентов на тех, кто не уйдет в просрочку, и тех, кто уйдет в нее. Схематично наши исходные данные представлены на рис. 3.1.

Имеется набор данных (корневой узел)														
Возраст	Пол	Наличие просрочки	УЗЕЛ 0											
70	Мужской	Да	<table><tr><th colspan="3">Просрочка</th></tr><tr><td>Нет</td><td>2</td><td>40%</td></tr><tr><td>Да</td><td>3</td><td>60%</td></tr></table>			Просрочка			Нет	2	40%	Да	3	60%
Просрочка														
Нет	2	40%												
Да	3	60%												
64	Мужской	Да												
69	Женский	Да												
68	Мужской	Нет												
65	Женский	Нет												

Рис. 3.1 Исходные данные перед началом работы CART (корневой узел)

На первом этапе (рис. 3.2) алгоритм CART ищет наилучшую точку расщепления по количественному предиктору *Возраст* (отсортировав значения по возрастанию) и номинальному предиктору *Пол*. В каждой рассматриваемой точке расщепления родительский узел гипотетически разбивается на два дочерних узла (в левый записываются наблюдения со значениями, которые меньше точки расщепления, в правый – наблюдения со значениями, которые больше или равны точке расщепления) и вычисляется взвешенное среднее значение неоднородностей дочерних узлов (используется мера Джини или энтропия). Наилучшей точкой расщепления для каждого предиктора будет такая точка, которая дает наименьшее взвешенное среднее значение неоднородностей дочерних узлов.

CART ищет по каждому предиктору наилучшую точку расщепления, дающую наименьшее взвешенное среднее значение неоднородностей дочерних узлов

Наблюдения со значениями < точка расщепления отправляются в левый узел



Наблюдения со значениями ≥ точка расщепления отправляются в правый узел

вычисление взвешенных средних значений неоднородностей дочерних узлов

Предиктор *Возраст*

Значения	64	65	68	69	70
Точки расщепления	64,5	66,5	68,5	69,5	

Точки расщепления	<64,5	≥ 64,5	<66,5	≥ 66,5	<68,5	≥ 68,5	<69,5	≥ 69,5
Категория <i>Нет</i>	0	2	1	1	2	0	2	0
Категория <i>Да</i>	1	2	1	2	1	2	2	1
Взвешенное среднее значение неоднородностей дочерних узлов	0,4		0,464		0,267		0,4	

гипотетическое разбиение

Просрочка		
Нет	2	40%
Да	3	60%

Возраст

<64,5			≥ 64,5		
Просрочка			Просрочка		
Нет	2	67%	Нет	0	0%
Да	1	33%	Да	2	100%
Джини 0,445			Джини 0		
$1 - 0,67^2 - 0,33^2$			$1 - 0^2 - 1^2$		

Взвешенное среднее значение неоднородностей дочерних узлов
 $(3/5) * 0,445 + (2/5) * 0 = 0,267$

Предиктор *Пол*

Значения	Мужской	Женский
Категория <i>Нет</i>	1	1
Категория <i>Да</i>	2	1
Взвешенное среднее значение неоднородностей дочерних узлов	0,467	

Рис. 3.2 Поиск наилучших расщеплений предикторов для корневого узла дерева классификации CART

Взвешенное среднее значение неоднородностей дочерних узлов рассчитывается следующим образом. Сначала вычисляются оценки неоднородности дочерних узлов. Затем оценка неоднородности левого дочернего узла умножается на вес левого дочернего узла (долю наблюдений, попавших в дочерний левый узел, взятую от общего количества наблюдений в родительском узле). Потом оценка неоднородности правого дочернего узла умножается на вес правого дочернего узла (долю наблюдений, попавших в дочерний правый узел, взятую от общего количества наблюдений в родительском узле). Наконец, произведения суммируются и получается взвешенное среднее неоднородностей дочерних узлов. Все вышесказанное можно проиллюстрировать формулой:

$$\Delta i = i_p - \left(\frac{n_L}{n_p} i_L + \frac{n_R}{n_p} i_R \right)$$

где:

Δi – уменьшение неоднородности;

i_p – неоднородность родительского узла;

$\left(\frac{n_L}{n_p} i_L + \frac{n_R}{n_p} i_R \right)$ – взвешенное среднее значение неоднородностей дочерних узлов;

n_L – количество наблюдений в левом дочернем узле;

n_R – количество наблюдений в правом дочернем узле;

n_p – количество наблюдений в родительском узле;

i_L – неоднородность левого дочернего узла;

i_R – неоднородность правого дочернего узла.

Для предиктора *Возраст* наилучшей точкой расщепления становится точка 68,5, которая дает наименьшее взвешенное среднее значение неоднородностей дочерних узлов, равное 0,267. Для предиктора *Пол* такой наилучшей точкой расщепления автоматически становится разбиение на мужчин и женщин.

На втором этапе (рис. 3.3) алгоритм CART выбирает наилучшую точку расщепления из набора наилучших точек расщепления, вычисленных по каждому предиктору на первом этапе, и разбивает по ней узел. Наилучшей точкой расщепления считается точка, минимизирующая взвешенное среднее значение неоднородностей дочерних узлов. В данном случае такой точкой будет точка 68,5, полученная для предиктора *Возраст*. Алгоритм вычисляет улучшение. Улучшение рассчитывается как разность между неоднородностью родительского узла и

минимальным взвешенным средним значением неоднородностей дочерних узлов.

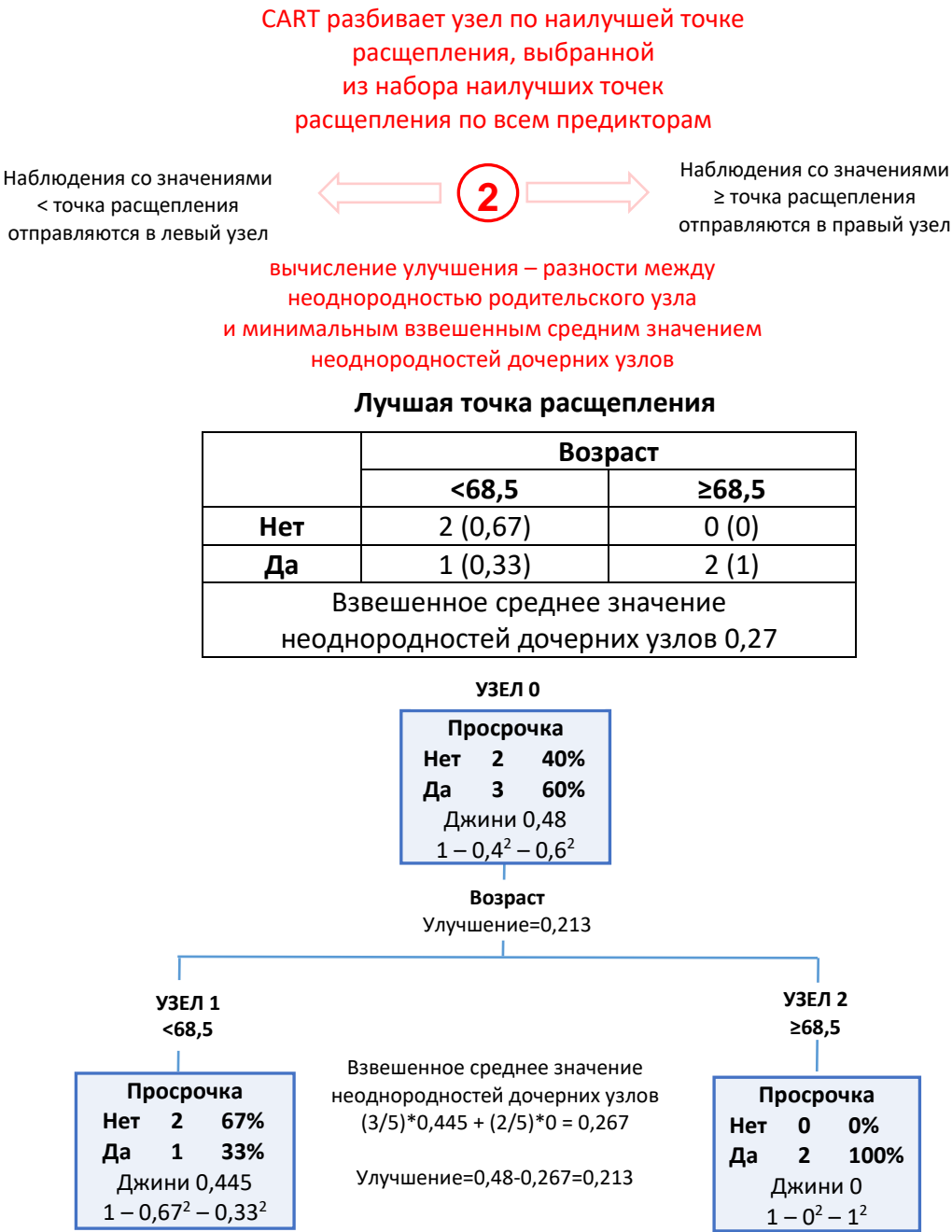


Рис. 3.3 Выбор наилучшего расщепления для корневого узла дерева классификации CART

Алгоритм спускается на уровень ниже и для полученных узлов 1 и 2 повторяет вышеописанные шаги. Сначала он рассматривает узел 1 (рис. 3.4).

Имеется поднабор данных (узел 1)

Возраст	Пол	Наличие просрочки	УЗЕЛ 1											
64	Мужской	Да	<table><tr><th colspan="3">Просрочка</th></tr><tr><td>Нет</td><td>2</td><td>67%</td></tr><tr><td>Да</td><td>1</td><td>33%</td></tr></table>			Просрочка			Нет	2	67%	Да	1	33%
Просрочка														
Нет	2	67%												
Да	1	33%												
68	Мужской	Нет												
65	Женский	Нет												

Рис. 3.4 Поднабор данных (узел 1)

Снова для каждой точки расщепления по каждому предиктору вычисляется взвешенное среднее значение неоднородностей дочерних узлов. Для каждого предиктора выбираем точку расщепления, которая дает наименьшее взвешенное среднее значение неоднородностей дочерних узлов.

Предиктор *Возраст*

Значения	64	65	68
Точки расщепления	64,5	66,5	

Точки расщепления	<64,5	≥ 64,5	<66,5	≥ 66,5
Категория <i>Нет</i>	0	2	1	1
Категория <i>Да</i>	1	0	1	0
Взвешенное среднее значение неоднородностей дочерних узлов	0		0,33	

Предиктор *Пол*

Значения	Мужской	Женский
Нет	1	1
Да	1	0
Взвешенное среднее значение неоднородностей дочерних узлов	0,33	

Рис. 3.5 Поиск наилучших расщеплений предикторов для узла 1 дерева классификации CART

Снова выбираем наилучшую точку расщепления из набора наилучших точек расщепления, вычисленных по всем предикторам, и разбиваем по ней узел. Алгоритм вновь вычисляет улучшение.

Лучшая точка расщепления

	Возраст	
	<64,5	≥64,5
Нет	0 (0)	2 (1)
Да	1 (1)	0 (0)
Взвешенное среднее значение неоднородностей дочерних узлов 0		

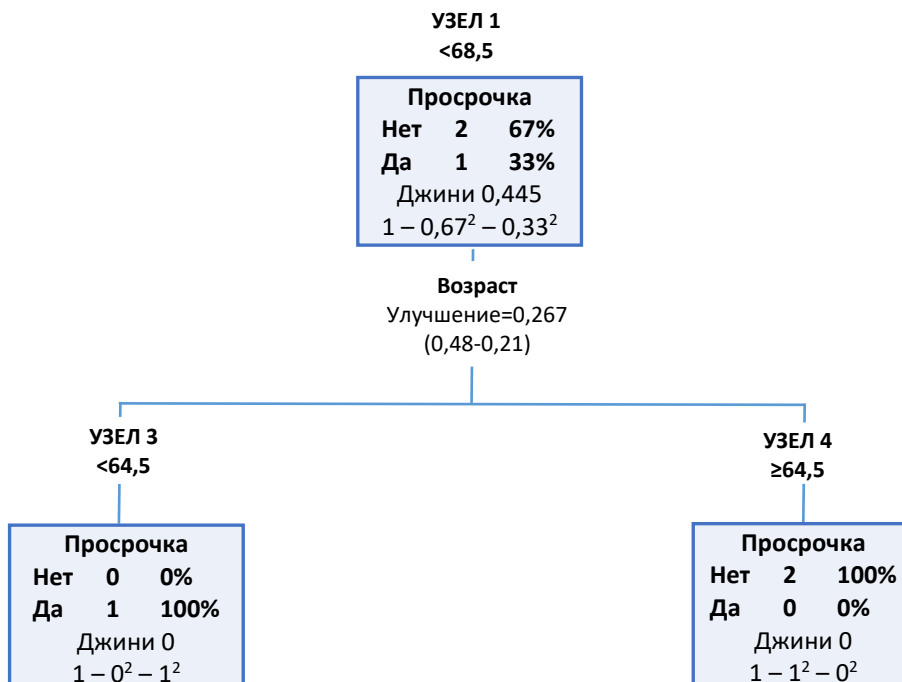


Рис. 3.6 Выбор наилучшего расщепления для узла 1 дерева классификации CART

Затем алгоритм рассматривает узел 2. Поскольку он является однородным, он не разбивается (срабатывает правило остановки) и становится терминальным узлом.

Алгоритм спускается на уровень ниже и рассматривает узлы 3 и 4. Поскольку узлы 3 и 4 тоже являются однородными, они не разбиваются (вновь срабатывает правило остановки) и становятся терминальными узлами.

Итоговое дерево классификации CART показано на рис. 3.7.

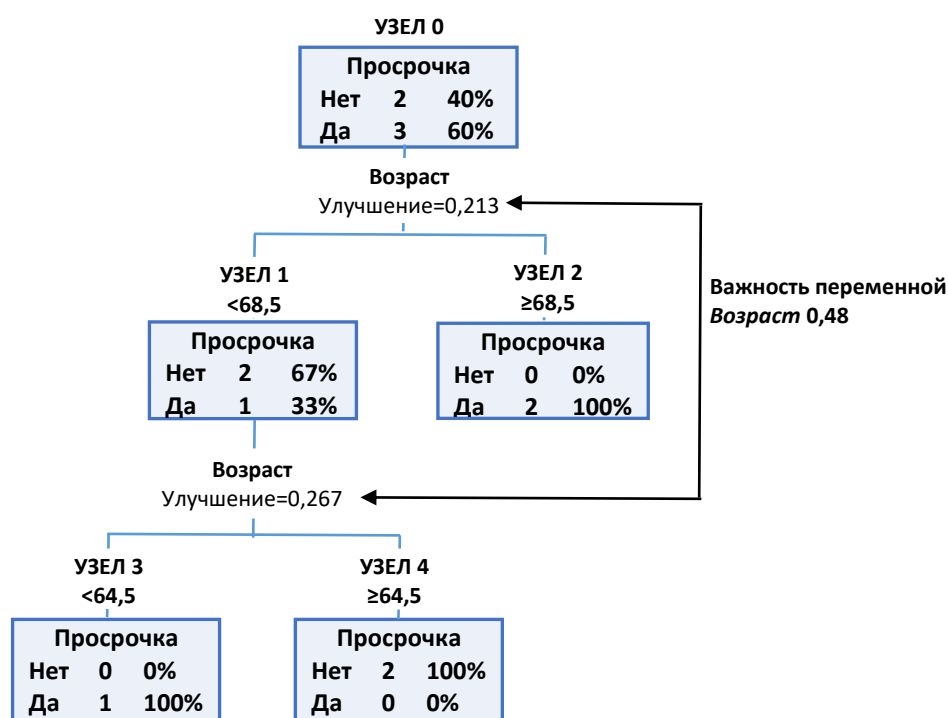


Рис. 3.7 Итоговое дерево классификации CART

Обратите внимание, метод CART позволяет вычислить важности предикторов. Данная информация может быть полезна, когда у вас имеется большое количество переменных и необходимо выбрать наиболее важные для включения в прогнозную модель (например, в модель логистической регрессии). Важность предиктора – это сумма улучшений, вызванных применением данного предиктора в качестве основной расщепляющей переменной или переменной-суррогата. Каждый из предикторов при каждом расщеплении берется с весом. Вес зависит от того, применялся ли предиктор в качестве основной расщепляющей переменной (предиктор, по которому был расщеплен родительский узел, имеет вес 1) или в качестве суррогата (вес зависит от ранга суррогата). Если допускается использовать 5 суррогатов, то первый суррогат на расщепление будет иметь наибольший вес среди прочих, а пятый – самый низкий.

Кроме того, часто вычисляют нормализованную важность для переменной, которая определяется формулой:

$$\text{Нормализованная важность} = 100 \times (\text{Важность} / \text{Максимальная важность})$$

Поэтому наиболее важный предиктор имеет нормализованное значение важности, равное 100.

Давайте вычислим важность предиктора *Возраст* в рамках нашего игрушечного примера. Она будет равна сумме улучшений – уменьшений неоднородности, когда в качестве предиктора использовалась переменная *Возраст*. Складываем наши улучшения 0,213 и 0,267, получаем 0,48. В данном случае важность предиктора *Возраст* равна неоднородности корневого узла. Это обозначает, что неоднородность корневого узла удалось полностью снизить за счет использования одной переменной *Возраст*.

Нетрудно увидеть недостаток важности. По сути важность складывается из частоты использования переменной в качестве предиктора разбиения, то есть чаще наиболее важными будут переменные, по которым можем быть рассмотрено больше вариантов разбиения и у них больше шансов стать предиктором разбиения. Поэтому наиболее важными переменными чаще будут переменные с большим количеством уникальных значений.

3.1.5.2 Особенности реализации дерева классификации CART в пакете R `rpart`

В пакете `rpart` на первом этапе для каждой рассматриваемой точки расщепления вместо взвешенного среднего значения неоднородностей дочерних узлов алгоритм вычисляет взвешенное уменьшение неоднородности при разбиении родительского узла на дочерние узлы. В качестве меры неоднородности используется либо мера Джини, либо энтропия. Сначала вычисляем разницу между неоднородностью родительского узла и неоднородностью левого дочернего узла, затем разницу между неоднородностью родительского узла и неоднородностью правого дочернего узла. Затем эти разницы умножаем на размеры узлов и полученные результаты складываем. Вышеперечисленные действия можно свести к следующей формуле:

$$n_L(i_P - i_L) + n_R(i_P - i_R)$$

где:

i_P – неоднородность родительского узла;

i_L – неоднородность левого дочернего узла;

i_R – неоднородность правого дочернего узла;

n_L – количество наблюдений в левом дочернем узле;

n_R – количество наблюдений в правом дочернем узле.

В итоге в качестве наилучшей рассматривается точка расщепления, которая дает наибольшее взвешенное уменьшение неоднородности.

CART ищет по каждому предиктору наилучшую точку расщепления, дающую наибольшее взвешенное уменьшение неоднородности

Наблюдения со значениями < точка расщепления отправляются в левый узел



Наблюдения со значениями ≥ точка расщепления отправляются в правый узел

вычисление взвешенных уменьшений неоднородности

Предиктор *Возраст*

Значения	64	65	68	69	70
Точки расщепления	64,5	66,5	68,5	69,5	

Точки расщепления	<64,5	≥ 64,5	<66,5	≥ 66,5	<68,5	≥ 68,5	<69,5	≥ 69,5
Категория <i>Нет</i>	0	2	1	1	2	0	2	0
Категория <i>Да</i>	1	2	1	2	1	2	2	1
Взвешенное уменьшение неоднородности	0,4		0,065		1,065		0,4	

гипотетическое разбиение

Просрочка		
Нет	2	40%
Да	3	60%

Возраст

<64,5			≥ 64,5		
Просрочка			Просрочка		
Нет	2	67%	Нет	0	0%
Да	1	33%	Да	2	100%
Джини 0,445			Джини 0		
$1 - 0,67^2 - 0,33^2$			$1 - 0^2 - 1^2$		

Взвешенное уменьшение неоднородности

$$3 \cdot (0,48 - 0,445) + 2 \cdot (0,48 - 0) = 1,065$$

Предиктор *Пол*

Значения	
Категория <i>Нет</i>	
Категория <i>Да</i>	
Взвешенное уменьшение неоднородности	

Мужской	Женский
1	1
2	1
0,065	

Рис. 3.8 Поиск наилучших расщеплений предикторов для корневого узла дерева классификации CART в пакете rpart

На втором этапе алгоритм CART выбирает наилучшую точку расщепления из набора наилучших точек расщепления, вычисленных по всем предикторам на первом этапе, и разбивает по ней узел. Наилучшей точкой расщепления считается точка, максимизирующая взвешенное уменьшение неоднородности. Максимальное взвешенное уменьшение неоднородности будет считаться улучшением. Кроме того, вычисляются т.н. альтернативные улучшения – взвешенные уменьшения неоднородности, которые можно было получить при использовании других переменных в качестве предиктора расщепления.

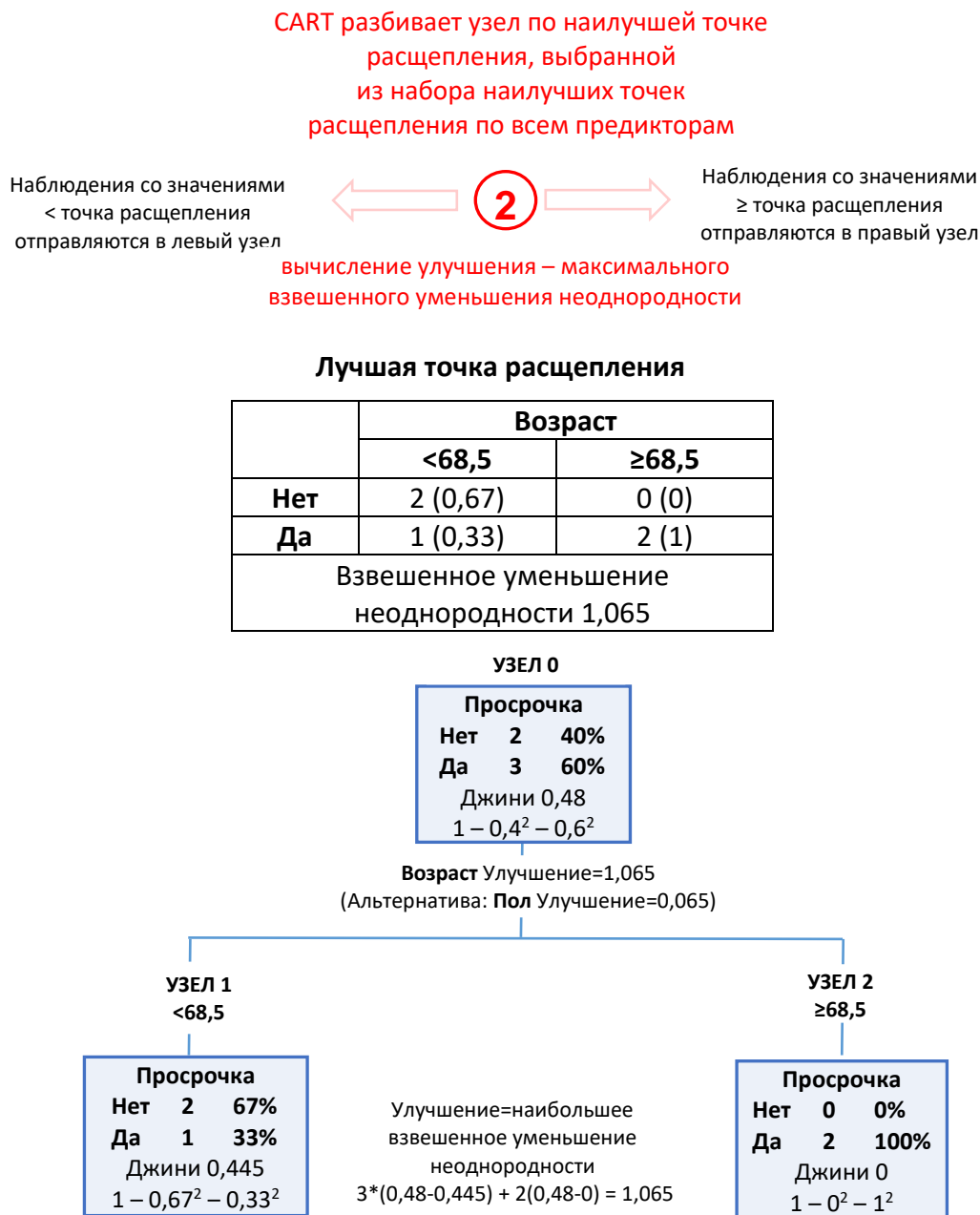


Рис. 3.9 Выбор наилучшего расщепления для корневого узла дерева классификации CART в пакете R rpart

Для полученных узлов алгоритм повторяет вышеописанные шаги, и так до тех пор, пока не сработают правила остановки. Итоговое дерево классификации CART показано на рис. 3.10.

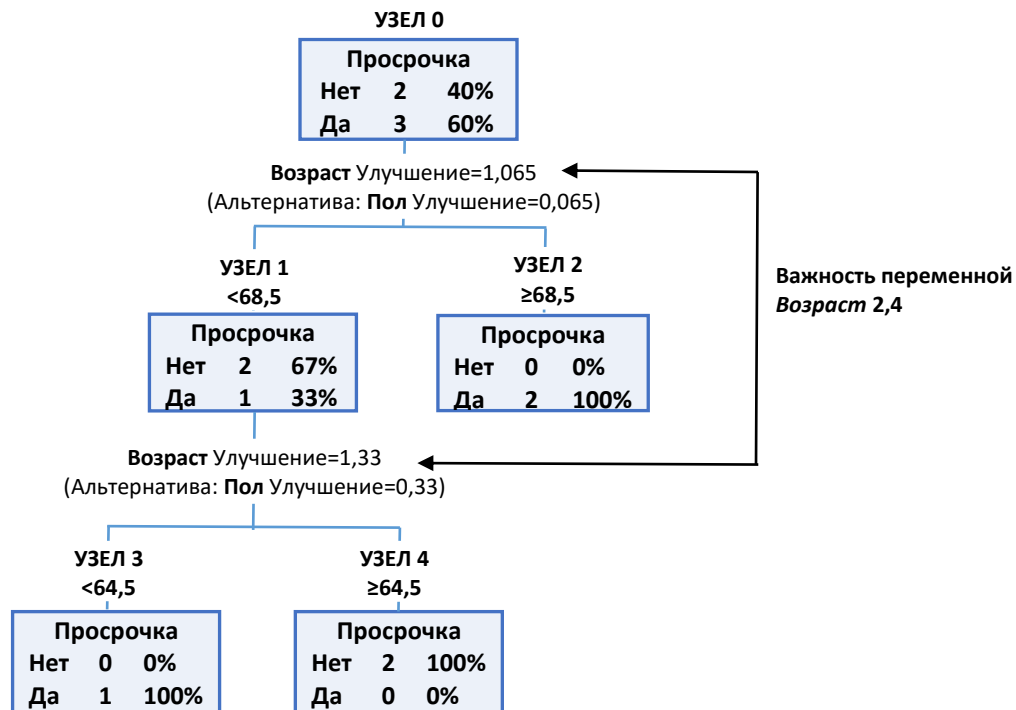


Рис. 3.10 Итоговое дерево классификации CART в пакете `rpart`

Как видно, построенное дерево классификации идентично дереву классификации, которое мы строили ранее, за исключением лишь того, что теперь улучшение является наибольшим взвешенным уменьшением неоднородности, а не разностью между неоднородностью родительского узла и минимальным взвешенным средним значением неоднородностей дочерних узлов. Кроме того, выводится информация об альтернативных улучшениях.

Давайте с помощью пакета `rpart` автоматически построим дерево классификации CART для рассматриваемого набора. Наш набор хранится в файле `CART_classification.csv`.

```
# загружаем данные
data <- read.csv2("C:/Trees/CART_classification.csv")

# устанавливаем пакет rpart
# install.packages("rpart")

# загружаем пакет rpart
library(rpart)

# строим модель дерева классификации CART
set.seed(42)
model<-rpart(default~., method='class', data,
              control=rpart.control(minsplit = 1, minbucket = 1, cp = 0.01))

# выводим краткую информацию
# о модели CART
model
```

Сводка 3.1. Краткая информация о построенном дереве классификации CART в пакете `rpart`

`n= 5`

`node), split, n, loss, yval, (yprob)`
 * denotes terminal node

```
1) root 5 2 Да (0.6000000 0.4000000)
  2) age>=68.5 2 0 Да (1.0000000 0.0000000) *
  3) age< 68.5 3 1 Нет (0.3333333 0.6666667)
    6) age< 64.5 1 0 Да (1.0000000 0.0000000) *
    7) age>=64.5 2 0 Нет (0.0000000 1.0000000) *
```

правило разбиения
количество наблюдений
количество неправильно предсказанных
предсказанный класс
вероятности классов

Мы видим, что дерево классификации CART, построенное с помощью пакета `rpart`, идентично дереву классификации CART, которое мы строили вручную. Сначала происходит разбиение по переменной *Возраст* в точке 68,5 и затем вновь по переменной *Возраст*, но уже в точке 64,5.

3.1.5.3 Дерево регрессии

Предположим, есть данные по клиентам микрофинансовой организации и известно количество дней просрочки по каждому. Для удобства расчетов представим, что наш набор данных состоит всего из 5 наблюдений. В качестве потенциальных предикторов фигурируют две переменные: *Возраст* и *Пол*. Переменная *Пол* является номинальной, переменная *Возраст* является количественной. Необходимо спрогнозировать глубину просрочки по каждому клиенту. Схематично наши исходные данные представлены на рис. 3.11.

Имеется набор данных (корневой узел)			УЗЕЛ 0	
Возраст	Пол	Количество дней просрочки	Количество дней просрочки	
70	Мужской	Да	Среднее	15,8
64	Мужской	Да	N	5
69	Женский	Да		
68	Мужской	Нет		
65	Женский	Нет		

Рис. 3.11 Исходные данные перед началом работы CART (корневой узел)

На первом этапе (рис. 3.12) алгоритм CART ищет наилучшую точку расщепления по количественному предиктору *Возраст* (отсортировав значения по возрастанию) и номинальному предиктору *Пол*. В каждой рассматриваемой точке расщепления родительский узел гипотетически разбивается на два дочерних узла (в левый записываются наблюдения со значениями, которые меньше точки расщепления, в правый – наблюдения со значениями, которые больше или равны точке расщепления) и вычисляется взвешенное среднее значение

неоднородностей дочерних узлов (используется среднеквадратичная ошибка). Наилучшей точкой расщепления для каждого предиктора будет такая точка, которая дает наименьшее взвешенное среднее значение неоднородностей дочерних узлов.

CART ищет по каждому предиктору наилучшую точку расщепления, дающую наименьшее среднее взвешенное значение неоднородностей дочерних узлов

Наблюдения со значениями < точка расщепления отправляются в левый узел



Наблюдения со значениями ≥ точка расщепления отправляются в правый узел

вычисление взвешенных средних значений неоднородностей дочерних узлов

Предиктор *Возраст*

Значения	64	65	68	69	70
Точки расщепления	64,5	66,5	68,5	69,5	

УЗЕЛ 0		
y_{iP}	\bar{y}_P	$(y_{iP} - \bar{y}_P)^2$
20	15,8	17,64
23	15,8	51,84
12	15,8	14,44
9	15,8	46,24
15	15,8	0,64
MSE 26,16		
$MSE(P) = \frac{1}{n_P} \sum_i (y_{iP} - \bar{y}_P)^2 = \frac{1}{5} \times 130,8 = 26,16$		

УЗЕЛ 1 <64,5		
y_{iL}	\bar{y}_L	$(y_{iL} - \bar{y}_L)^2$
23	23	0
MSE 0		
$MSE(L) = \frac{1}{n_L} \sum_i (y_{iL} - \bar{y}_L)^2 = \frac{1}{1} \times 0 = 0$		

УЗЕЛ 2 ≥ 64,5		
y_{iR}	\bar{y}_R	$(y_{iR} - \bar{y}_R)^2$
20	14	36
12	14	4
9	14	25
15	14	1
MSE 16,5		
$MSE(R) = \frac{1}{n_R} \sum_i (y_{iR} - \bar{y}_R)^2 = \frac{1}{4} \times 66 = 16,5$		

Взвешенное среднее значение неоднородностей дочерних узлов
 $(1/5) \times 0 + (4/5) \times 16,5 = 13,2$

Предиктор *Пол*

Значения	Мужской	Женский
Взвешенное среднее значение неоднородностей дочерних узлов	22,6	

Рис. 3.12 Поиск наилучших расщеплений предикторов для корневого узла дерева регрессии CART

Для предиктора *Возраст* наилучшей точкой расщепления становится точка 64,5, которая дает наименьшее взвешенное среднее значение неоднородностей дочерних узлов, равное 12,96. Для предиктора *Пол* такой наилучшей точкой расщепления автоматически становится разбиение на мужчин и женщин.

На втором этапе (рис. 3.13) алгоритм CART выбирает наилучшую точку расщепления из набора наилучших точек расщепления, вычисленных по каждому предиктору на первом этапе, и разбивает по ней узел. Наилучшей точкой расщепления считается точка, минимизирующая взвешенное среднее значение неоднородностей дочерних узлов. В данном случае такой точкой будет точка 64,5, полученная для предиктора *Возраст*. Алгоритм вычисляет улучшение. Улучшение рассчитывается как разность между неоднородностью родительского узла и минимальным взвешенным средним значением неоднородностей дочерних узлов.

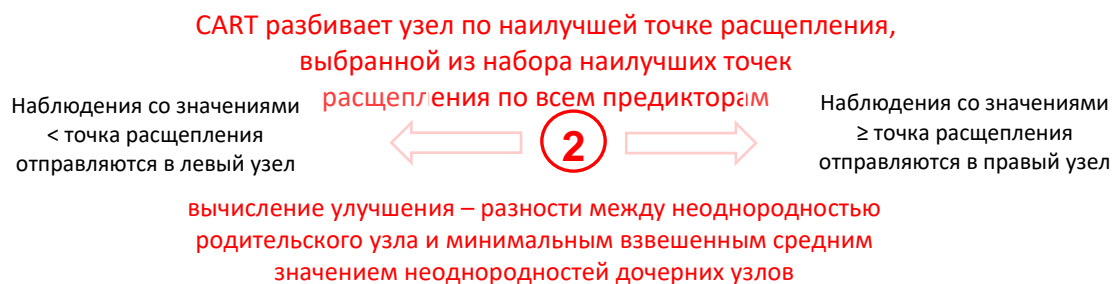


Рис. 3.13 Выбор наилучшего расщепления для корневого узла дерева регрессии CART

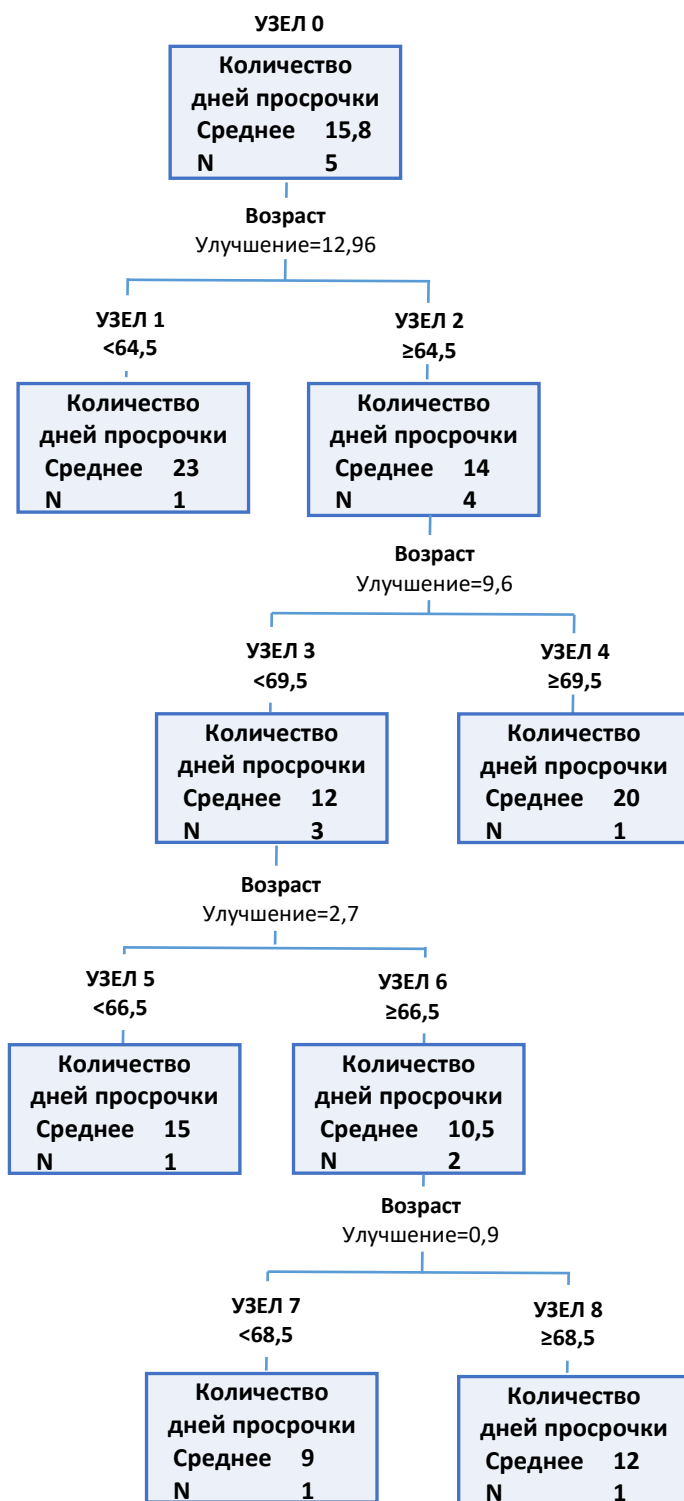


Рис. 3.14 Итоговое дерево регрессии CART

Для полученных узлов алгоритм повторяет вышеописанные шаги, и так до тех пор, пока не сработают правила остановки. Итоговое дерево регрессии CART показано на рис. 3.14.

3.1.5.4 Особенности реализации дерева регрессии CART в пакете R `rpart`

В пакете `rpart` на первом этапе для каждой рассматриваемой точки расщепления алгоритм вычисляет уменьшение девианса (суммы квадратов остатков) при разбиении родительского узла на дочерние узлы. Сначала вычисляем девианс родительского узла, затем девиансы дочерних узлов. Делим девианс левого дочернего узла на девианс родительского узла. Делим девианс правого дочернего узла на девианс родительского узла. Результаты складываем и вычитаем из 1.

Вышеперечисленные действия можно свести к следующей формуле:

$$1 - \left(\frac{D_L}{D_P} + \frac{D_R}{D_P} \right)$$

где:

i_P – девианс родительского узла;

i_L – девианс левого дочернего узла;

i_R – девианс правого дочернего узла.

В итоге в качестве наилучшей рассматривается точка расщепления, которая дает наибольшее уменьшение девианса при разбиении родительского узла на дочерние узлы.

CART ищет по каждому предиктору наилучшую точку расщепления, дающую наибольшее уменьшение девианса при разбиении родительского узла на дочерние

Наблюдения со значениями < точка расщепления отправляются в левый узел



Наблюдения со значениями \geq точка расщепления отправляются в правый узел

вычисление уменьшений девианса

Предиктор *Возраст*

Значения	64	65	68	69	70
Точки расщепления	64,5	66,5	68,5	69,5	

УЗЕЛ 0		
y_{ip}	\bar{y}_p	$(y_{ip} - \bar{y}_p)^2$
20	15,8	17,64
23	15,8	51,84
12	15,8	14,44
9	15,8	46,24
15	15,8	0,64

Девианс 130,8

УЗЕЛ 1 <64,5		
y_{il}	\bar{y}_l	$(y_{il} - \bar{y}_l)^2$
23	23	0

Девианс 0

УЗЕЛ 2 $\geq 64,5$		
y_{ir}	\bar{y}_r	$(y_{ir} - \bar{y}_r)^2$
20	14	36
12	14	4
9	14	25
15	14	1

Девианс 66

Уменьшение девианса
 $1 - (0/130,8 + 66/130,8) = 0,495$

Предиктор *Пол*

Значения	Мужской	Женский
Уменьшение девианса	0,135	

Рис. 3.15 Поиск наилучших расщеплений предикторов для корневого узла дерева регрессии CART в пакете rpart

Для предиктора *Возраст* наилучшей точкой расщепления становится точка 64,5, которая дает наибольшее уменьшение девианса, равное 0,495. Для предиктора *Пол* такой наилучшей точкой расщепления автоматически становится разбиение на мужчин и женщин.

На втором этапе алгоритм CART выбирает наилучшую точку расщепления из набора наилучших точек расщепления, вычисленных по

всем предикторам на первом этапе, и разбивает по ней узел. Наилучшей точкой расщепления считается точка, максимизирующая уменьшение девианса. Максимальное уменьшение девианса при разбиении родительского узла на дочерние узлы будет считаться улучшением. Кроме того, вычисляются т.н. альтернативные улучшения – уменьшения девианса, которые можно было получить при использовании других переменных в качестве предиктора расщепления.



Рис. 3.16 Выбор наилучшего расщепления для корневого узла дерева регрессии CART в пакете rpart

Для полученных узлов алгоритм повторяет вышеописанные шаги, и так до тех пор, пока не сработают правила остановки. Итоговое дерево регрессии CART показано на рис. 3.17.

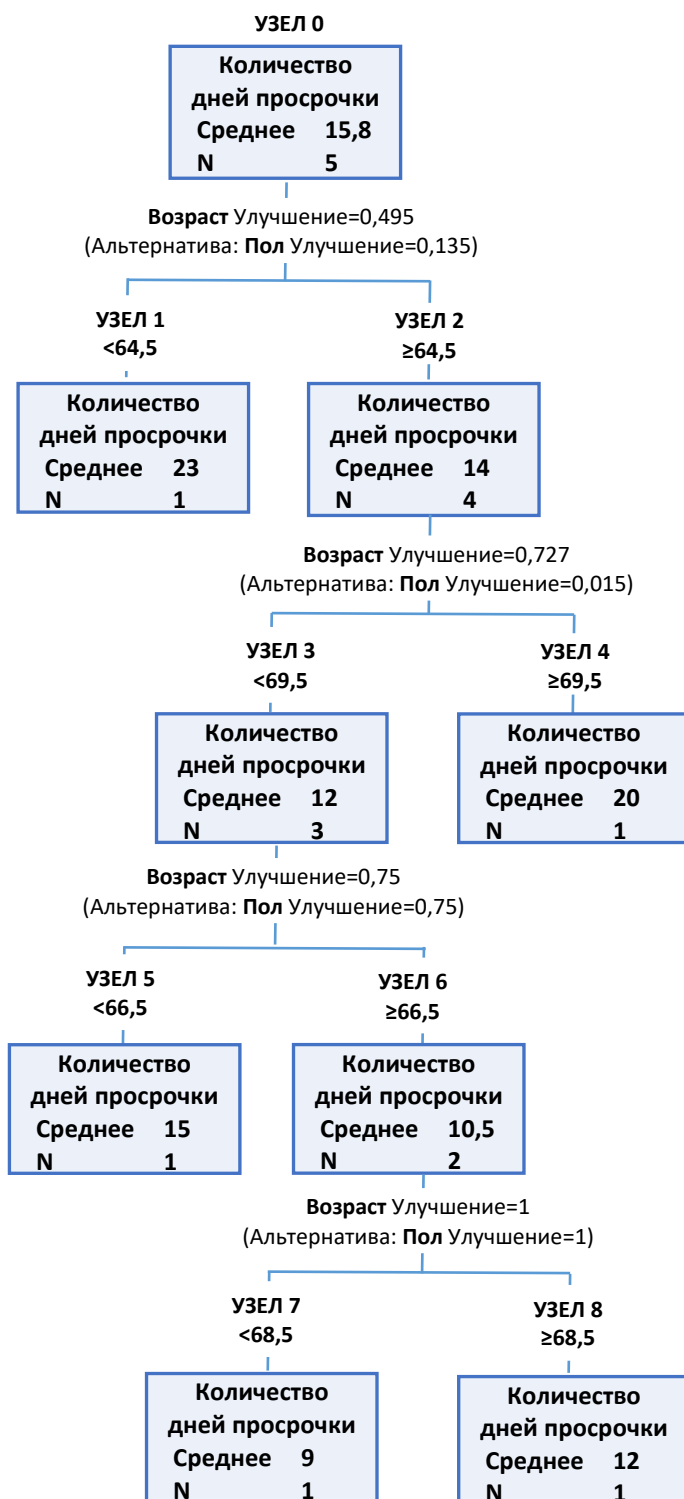


Рис. 3.17 Итоговое дерево регрессии CART в пакете gart

Как видно, построенное дерево регрессии идентично дереву регрессии, которое мы строили ранее, за исключением лишь того, что теперь улучшение – это наибольшее уменьшение девианса. Кроме того, выводится информация об альтернативных улучшениях.

Давайте с помощью пакета `rpart` автоматически построим дерево классификации CART для рассматриваемого набора. Он записан в файле `CART_regression.csv`.

```
# загружаем данные
data <- read.csv2("C:/Trees/CART_regression.csv")
str(data)

# строим модель дерева регрессии CART
set.seed(42)
model<-rpart(days_of_delinquency~., method='anova', data,
             control=rpart.control(minsplit = 1, minbucket = 1, cp = 0.01))

# выводим краткую информацию
# о модели регрессии CART
model
```

Сводка 3.2. Краткая информация о построенном дереве регрессии CART в пакете `rpart`

n= 5

```
node), split, n, deviance, yval
* denotes terminal node

1) root 5 130.8 15.8
 2) age>=64.5 4 66.0 14.0
   4) age< 69.5 3 18.0 12.0
     8) age>=66.5 2 4.5 10.5
       16) age< 68.5 1 0.0 9.0 *
       17) age>=68.5 1 0.0 12.0 *
     9) age< 66.5 1 0.0 15.0 *
   5) age>=69.5 1 0.0 20.0 *
 3) age< 64.5 1 0.0 23.0 *
```

правило разбиения
количество наблюдений
девианс
спрогнозированное значение

Из сводки 3.2 видно, что дерево регрессии CART, построенное с помощью пакета `rpart`, идентично дереву регрессии CART, которое мы строили вручную.

Лекция 3.2 Построение и интерпретация дерева классификации CART

3.2.1 Подготовка данных

Данные, которыми мы воспользуемся для построения дерева классификации CART, записаны в файле `Response.csv`. Исходная выборка содержит записи о 30259 клиентах, классифицированных на два класса: 0 — отклика нет (17170 клиентов) и 1 — отклик есть (13089 клиентов). По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

- номинальный предиктор *Ипотечный кредит* [`mortgage`];
- номинальный предиктор *Страхование жизни* [`life_ins`];
- номинальный предиктор *Кредитная карта* [`cre_card`];

- номинальный предиктор *Дебетовая карта* [deb_card];
- номинальный предиктор *Мобильный банк* [mob_bank];
- номинальный предиктор *Текущий счет* [curr_acc];
- номинальный предиктор *Интернет-доступ к счету* [internet];
- номинальный предиктор *Индивидуальный займ* [perloan];
- номинальный предиктор *Наличие сбережений* [savings];
- номинальный предиктор *Пользование банкоматом за последнюю неделю* [atm_user];
- номинальный предиктор *Пользование услугами онлайн-маркетплейса за последний месяц* [markpl];
- количественный предиктор *Возраст* [age];
- количественный предиктор *Давность клиентской истории* [cus_leng];
- номинальная зависимая переменная *Отклик на предложение новой карты* [response].

Запустим программу R. Запишем данные в датафрейм **data**, чтобы начать работу с ними.

```
# загружаем данные
data <- read.csv2("C:/Trees/Response.csv")
```

Проверим, как выглядят наши переменные.

```
# смотрим типы переменных
str(data)
```

Сводка 3.3. Датафрейм data

```
'data.frame':    30259 obs. of  14 variables:
 $ mortgage: int  0 1 1 1 1 0 0 1 1 1 ...
 $ life_ins: int  0 1 1 1 1 0 0 1 1 1 ...
 $ cre_card: int  0 0 0 1 0 0 0 0 0 1 ...
 $ deb_card: int  0 1 1 1 1 1 1 1 1 1 ...
 $ mob_bank: int  0 1 0 0 0 1 0 0 0 0 ...
 $ curr_acc: int  0 0 0 1 0 1 0 0 0 1 ...
 $ internet: int  0 0 0 0 0 0 0 0 0 0 ...
 $ perloan : int  0 0 0 0 1 0 0 0 0 0 ...
 $ savings : int  0 0 0 0 0 0 0 0 0 0 ...
 $ atm_user: int  0 1 0 1 1 0 1 1 1 0 ...
 $ markpl  : int  0 0 1 1 0 0 0 0 1 0 ...
 $ age     : int  18 18 18 18 18 18 18 18 18 18 ...
 $ cus_leng: int  1 2 2 2 2 2 1 1 3 2 ...
 $ response: int  0 1 1 1 0 1 1 1 0 1 ...
```

Сводка 3.3 показывает, что все переменные, включая категориальные, представлены в виде векторов типа **integer**. Как мы уже знаем, что в R для категориальной переменной существует вектор типа **factor**. Поэтому все целочисленные векторы, соответствующие категориальным переменным (это все переменные, кроме переменных *age* и *cus_leng*), мы должны преобразовать в факторы.

```
# преобразовываем переменные в факторы, за исключением
# переменных с индексами 12 и 13
data[, -c(12:13)] <- lapply(data[, -c(12:13)], factor)
```

Снова проверим, как выглядят переменные.

```
str(data)
```

Сводка 3.4. Таблица данных data после преобразования

```
'data.frame': 30259 obs. of 14 variables:
 $ mortgage: Factor w/ 2 levels "0","1": 1 2 2 2 2 1 1 2 2 2 ...
 $ life_ins: Factor w/ 2 levels "0","1": 1 2 2 2 2 1 1 2 2 2 ...
 $ cre_card: Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 1 1 2 ...
 $ deb_card: Factor w/ 2 levels "0","1": 1 2 2 2 2 2 2 2 2 2 ...
 $ mob_bank: Factor w/ 2 levels "0","1": 1 2 1 1 1 2 1 1 1 1 ...
 $ curr_acc: Factor w/ 2 levels "0","1": 1 1 1 2 1 2 1 1 1 2 ...
 $ internet: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ perloan : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
 $ savings : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ atm_user: Factor w/ 2 levels "0","1": 1 2 1 2 2 1 2 2 2 1 ...
 $ markpl : Factor w/ 2 levels "0","1": 1 1 2 2 1 1 1 1 2 1 ...
 $ age : int 18 18 18 18 18 18 18 18 18 18 ...
 $ cus_leng: int 1 2 2 2 2 2 1 1 3 2 ...
 $ response: Factor w/ 2 levels "0","1": 1 2 2 2 1 2 2 2 1 2 ...
```

Теперь категориальные переменные правильно записаны как векторы типа **factor**.

Разбиваем набор данных на обучающую и контрольную выборки.

```
# задаем стартовое значение генератора случайных чисел, чтобы
# каждый раз получать одно и то же разбиение
# на обучающую и контрольную выборки
set.seed(100)

# разбиваем набор на обучающую и контрольную выборки
ind <- sample(2, nrow(data), replace=TRUE, prob=c(0.7, 0.3))
train <- data[ind==1,]
valid <- data[ind==2,]
```

Проверяем наличие пропусков.

```
# проверяем наличие пропусков в обучающей выборке
sapply(train, function(x) sum(is.na(x)))
```

Сводка 3.5. Отчет по пропущенным значениям в датафрейме train

```
mortgage life_ins cre_card deb_card mob_bank curr_acc internet perloan savings atm_user
0 0 0 0 0 0 0 0 0 0
markpl age cus_leng response
0 0 0 0
```

```
# проверяем наличие пропусков в контрольной выборке
sapply(valid, function(x) sum(is.na(x)))
```

Сводка 3.6. Отчет по пропущенным значениям в датафрейме valid

```
mortgage life_ins cre_card deb_card mob_bank curr_acc internet perloan savings atm_user
0 0 0 0 0 0 0 0 0 0
markpl age cus_leng response
0 0 0 0
```

Пропусков нет, можно приступать к моделированию.

3.2.2 Построение и интерпретация модели классификации

Чтобы построить дерево CART, необходимо воспользоваться функцией `rpart`. Функция `rpart` имеет общий вид:

```
rpart(formula, data, method, control)
```

где

<code>formula=</code>	Задаёт формулу в формате <i>Зависимая переменная ~ Предиктор1 + Предиктор2 + Предиктор3 + др.</i>
<code>data=</code>	Задаёт таблицу данных для анализа
<code>method=</code>	'class' – для деревьев классификации; 'anova' – для деревьев регрессии.
<code>control=</code>	Задаёт критерии роста дерева CART с помощью вспомогательной функции <code>rpart.control</code> : <ul style="list-style-type: none">• <code>minsplit</code> – минимальное число наблюдений в узле перед расщеплением, по умолчанию равно 20;• <code>cp</code> (complexity parameter) – пороговое значение штрафа за сложность, по умолчанию 0,01. Если разбиение уменьшает ошибку модели на значение, меньшее порогового значения <code>cp</code>, оно не принимается и дерево останавливается в росте;• <code>minbucket</code> – минимальное количество наблюдений в терминальном узле, по умолчанию равен <code>minsplit/3</code>;• <code>xval</code> – количество разбиений исходной выборки на обучающую и контрольную для кроссвалидации, по умолчанию 10.

Поскольку наша зависимая переменная *response* является категориальной, строится дерево классификации.

Построим модель дерева классификации по всем исходным предикторам, уменьшив при этом пороговое значение штрафа за сложность `cp`.

```
set.seed(42)
# строим модель классификации, уменьшив пороговое
# значение штрафа за сложность cp
model<-rpart(response~., method='class', control=rpart.control(cp=0.001),
              data=train)
```

Выведем информацию о качестве модели.

```
# выводим информацию о качестве модели
printcp(model)
```

Сводка 3.7. Оценка качества дерева классификации, используется ошибка классификации

Classification tree:

```
rpart(formula = response ~ ., data = train, method = "class",
      control = rpart.control(cp = 0.001))
```

Variables actually used in tree construction:

```
[1] age      atm_user cre_card curr_acc cus_leng markpl  mob_bank
```

Root node error: 9229/21313 = 0.43302

n= 21313

	CP	nsplit	rel error	xerror	xstd
1	0.3634197	0	1.00000	1.00000	0.0078380
2	0.0716221	1	0.63658	0.63658	0.0070684
3	0.0393325	3	0.49334	0.49334	0.0064835
4	0.0167949	4	0.45400	0.45400	0.0062867
5	0.0117022	5	0.43721	0.43721	0.0061971
6	0.0018420	6	0.42551	0.42551	0.0061327
7	0.0013002	7	0.42366	0.42670	0.0061394
8	0.0012461	8	0.42236	0.42540	0.0061321
9	0.0010835	10	0.41987	0.42551	0.0061327
10	0.0010000	11	0.41879	0.42616	0.0061363

сложность
номер разбиения
относительная ошибка
кросс-валидационная ошибка
стандартное отклонение кросс-валидационной ошибки

В сводке 3.7 приводится сложность и ошибка модели в зависимости от использованного количества расщеплений. В целом приведенная таблица позволяет увидеть, как с ростом дерева увеличивается его сложность и уменьшается ошибка классификации.

Variables actually used in tree construction показывает переменные, которые были включены в модель дерева. Такими переменными стали *Возраст [age]*, *Пользование банкоматом за последнюю неделю[atm_user]*, *Кредитная карта [cre_card]*, *Текущий счет [curr_acc]*, *Давность клиентской истории [cus_leng]*, *Пользование услугами онлайн-маркетплейса за последний месяц [markpl]*, *Мобильный банк [mob_bank]*.

Root node error показывает базовую ошибку классификации. Это доля миноритарной категории зависимой переменной в исходной выборке.

Параметр **CP (complexity parameter)** показывает изменение сложности модели с ростом дерева. В данном случае 12-е расщепление уменьшает ошибку модели на значение, меньшее порогового значения сложности **cp**. Проще говоря, «стоимость» 12-го расщепления больше его сложности, оно не принимается и дерево останавливается в росте после 11 расщеплений.

Параметр **nsplit** показывает количество расщеплений на каждом этапе построения дерева. В последней строке в столбце **nsplit** приводится итоговое количество расщеплений. Количество терминальных узлов всегда на единицу больше количества расщеплений. В нашем случае дерево имеет 12 терминальных узлов.

Параметр `rel_error` показывает относительную ошибку классификации (долю наблюдений, неправильно классифицированных дерево с данным количеством расщеплений). Он отмасштабирован так, чтобы первый узел имел относительную ошибку классификации, равную 1. Умножение показателя на базовую ошибку классификации позволяет получить абсолютную ошибку классификации. Например, вычислим абсолютную ошибку классификации для `sr=0.0716221`, получаем $0,63658 \times 0,43 = 0,274$ или 27,4%. Это обозначает, что модель на обучающей выборке неправильно классифицировала 27,4% наблюдений. Несмотря на возрастание сложности дерева данный показатель может уменьшаться, поэтому его называют «оптимистичным» и обычно не используют для оценки размера дерева.

Параметр `xerror` показывает кросс-валидационную ошибку. Для дерева классификации кроссвалидационная ошибка – это ошибка классификации, усредненная по контрольным блокам перекрестной проверки. Как и `rel_error`, показатель отмасштабирован так, чтобы первый узел имел кросс-валидационную ошибку, равную 1. Умножение показателя на базовую ошибку классификации позволяет получить абсолютную кросс-валидационную ошибку классификации. Например, вычислим абсолютную кросс-валидационную ошибку классификации для `sr=0.0716221`, получаем $0,63658 \times 0,43 = 0,274$ или 27,4%. Это обозначает, что используя модель на новых данных, мы получим 27,4% неверно классифицированных наблюдений.

Параметр `xstd` показывает стандартное отклонение для кросс-валидационной ошибки.

Теперь выведем развернутые результаты построения модели.

```
# выводим подробную информацию о модели  
summary(model)
```

Поскольку полученная сводка является довольно большой, будем приводить и пояснять ее по частям.

Сводка 3.8. Развернутые результаты: оценка сложности и качества модели

Call:

```
rpart(formula = response ~ ., data = train, method = "class",
      control = rpart.control(cp = 0.001))
n= 21313
```

	CP	nsplit	rel error	xerror	xstd
1	0.363419655	0	1.0000000	1.0000000	0.007838009
2	0.071622061	1	0.6365803	0.6365803	0.007068423
3	0.039332539	3	0.4933362	0.4933362	0.006483491
4	0.016794886	4	0.4540037	0.4540037	0.006286660
5	0.011702243	5	0.4372088	0.4372088	0.006197144
6	0.001842020	6	0.4255066	0.4255066	0.006132723
7	0.001300249	7	0.4236645	0.4266985	0.006139363
8	0.001246072	8	0.4223643	0.4253982	0.006132118
9	0.001083541	10	0.4198721	0.4255066	0.006132723
10	0.001000000	11	0.4187886	0.4261567	0.006136347

В сводке 3.8 приводится уже знакомая таблица с оценкой сложности и качества модели.

Сводка 3.9. Развернутые результаты: важность переменных

Variable importance								
cus_leng	age	atm_user	savings	curr_acc	cre_card	markpl	mortgage	internet
61	25	6	2	1	1	1	1	1

Сводка 3.9 показывает важность переменной в модели. Наиболее важным предиктором оказалась *Давность клиентской истории [cus_leng]*. В данном случае важность переменной – это процент использования данной переменной в качестве предиктора разбиения. В данном случае переменная *Давность клиентской истории* использовалась в 61% разбиений.

Сводка 3.10. Развернутые результаты: характеристики узлов (фрагмент)

```
Node number 3: 14746 observations,      complexity param=0.07162206
predicted class=1 expected loss=0.3862742 P(node) =0.6918782
class counts: 5696 9050
probabilities: 0.386 0.614
left son=6 (10670 obs) right son=7 (4076 obs)
Primary splits:
  cus_leng < 1.5 to the right, improve=868.1191, (0 missing)
  age      < 44.5 to the right, improve=562.7721, (0 missing)
  atm_user splits as RL,      improve=518.9053, (0 missing)
  curr_acc splits as RL,      improve=480.0109, (0 missing)
  markpl  splits as RL,      improve=447.7176, (0 missing)
```

спрогнозированный класс
доля неверно классифицированных наблюдений
доля наблюдений в узле от исходной выборки
частоты классов
вероятности классов

Сводка 3.10 показывает характеристики каждого узла дерева.

Predicted class показывает спрогнозированную категорию зависимой переменной для узла. Например, рассмотрим узел 3. Он содержит 14746

наблюдений. Для данного узла спрогнозирован класс *Есть отклик* (класс 1).

Expected loss показывает долю неверно классифицированных наблюдений от общего числа наблюдений в узле. Например, для узла 3 доля неверно классифицированных наблюдений составляет $5696/14746=0,386$ или 38,6%.

P(node) показывает долю наблюдений в узле от исходной выборки. Например, доля наблюдений в узле 3 от исходной выборки составляет $14746/21313=0,692$ или 69,2%.

Class counts показывает распределение наблюдений по классам. В узле 3 5696 наблюдений – это неоткликнувшиеся клиенты и 9050 наблюдений – это откликнувшиеся клиенты. Поскольку большая часть наблюдений в узле 3 – это откликнувшиеся клиенты, для данного узла была спрогнозирована класс *Есть отклик* (класс 1).

Probabilities показывает распределение вероятностей по категориям. Для каждого клиента, относящегося к узлу 3, предсказываются вероятность отсутствия отклика 0.386 и вероятность наличия отклика 0.614.

Left son и **right son** показывают, как узел расщепляется на узлы-потомки. Например, узел 3 в свою очередь разбивается на два узла-потомка – узел 6 и узел 7.

Primary splits показывают переменную, которая была использована для расщепления узла, а также альтернативные переменные, которые можно было использовать для расщепления. Для каждой переменной приводятся значения **improve**, т.е. улучшения. Под улучшением понимается взвешенное уменьшение неоднородности, которое можно было получить при использовании данной переменной в качестве предиктора расщепления. Оно вычисляется по формуле:

$$n_L(i_P - i_L) + n_R(i_P - i_R)$$

где:

i_P – неоднородность родительского узла;

i_L – неоднородность левого дочернего узла;

i_R – неоднородность правого дочернего узла;

n_L – количество наблюдений в левом дочернем узле;

n_R – количество наблюдений в правом дочернем узле.

Surrogate splits показывают переменные-заменители (суррогаты), разбиения, задаваемые которыми, наиболее близки к разбиению, задаваемому исходным предиктором. Суррогаты применяются, если переменная, используемая для разбиения, имеет пропущенные значения.

На практике удобнее выводить краткие результаты построения модели.

```
# выводим краткую информацию о модели
print(model)
```

Сводка 3.11. Краткие результаты (фрагмент)

n= 21313

node), split, n, loss, yval, (yprob)
* denotes terminal node

```
1) root 21313 9229 0 (0.5669779 0.4330221)
 2) cus_leng>=2.5 6567 179 0 (0.9727425 0.0272575) *
 3) cus_leng< 2.5 14746 5696 1 (0.3862742 0.6137258)
 6) cus_leng>=1.5 10670 5253 1 (0.4923149 0.5076851)
 12) atm_user=1 5234 1956 0 (0.6262896 0.3737104)
    24) curr_acc=1 4423 1473 0 (0.6669681 0.3330319)
      48) markpl=1 3380 1010 0 (0.7011834 0.2988166) *
      49) markpl=0 1043 463 0 (0.5560882 0.4439118)
        98) cre_card=1 636 248 0 (0.6100629 0.3899371)
          196) mob_bank=0 524 187 0 (0.6431298 0.3568702) *
          197) mob_bank=1 112 51 1 (0.4553571 0.5446429) *
          99) cre_card=0 407 192 1 (0.4717445 0.5282555) *
    25) curr_acc=0 811 328 1 (0.4044390 0.5955610)
      50) markpl=1 357 170 0 (0.5238095 0.4761905)
        100) age>=29.5 225 98 0 (0.5644444 0.4355556) *
        101) age< 29.5 132 60 1 (0.4545455 0.5454545) *
        51) markpl=0 454 141 1 (0.3105727 0.6894273) *
 13) atm_user=0 5436 1975 1 (0.3633186 0.6366814)
    26) age>=44.5 2163 900 0 (0.5839112 0.4160888)
      52) cre_card=1 1439 484 0 (0.6636553 0.3363447) *
      53) cre_card=0 724 308 1 (0.4254144 0.5745856) *
    27) age< 44.5 3273 712 1 (0.2175374 0.7824626) *
 7) cus_leng< 1.5 4076 443 1 (0.1086850 0.8913150) *
```

правило разбиения
количество наблюдений в узле
количество неверно классифицированных наблюдений
спрогнозированный класс
вероятности классов

В сводке 3.11 для каждого узла выводится:

- **node)** – номер узла;
- **split** – переменная, использованная для расщепления узла;
- **n** – количество наблюдений в узле;
- **loss** – количество неверно классифицированных в узле;
- **yval** – спрогнозированный класс зависимой переменной для узла;
- **yprobs** – вероятности классов зависимой переменной в узле.

Знаком * отмечены терминальные узлы.

Например, узел, соответствующий правилу **cus_leng>=2.5**, содержит 6567 наблюдений. Для него дан прогноз 0 – отсутствие отклика, поскольку вероятность отсутствия отклика равна 0,97. 179 клиентов, которые откликнулись, неверно классифицированы. Таким образом, параметр **loss** (количество неверно классифицированных) равен 179.

Часто бывает необходимость вывести непосредственно сами правила разбиения. Их можно вывести с помощью функции `asRules` пакета `rattle`. Давайте установим и загрузим пакет `rattle`.

```
# устанавливаем пакет rattle
# install.packages("rattle")

# загружаем пакет rattle
library(rattle)
```

Теперь выводим правила.

```
# выводим непосредственно
# правила разбиения
asRules(model)
```

Сводка 3.12. Правила разбиения, полученные с помощью функции `asRules` пакета `rattle` (фрагмент)

```
Rule number: 7 [response=1 cover=4076 (19%) prob=0.89]
  cus_leng< 2.5
  cus_leng< 1.5
```

```
Rule number: 27 [response=1 cover=3273 (15%) prob=0.78]
  cus_leng< 2.5
  cus_leng>=1.5
  atm_user=0
  age< 44.5
```

номер правила
спрогнозированный класс
количество наблюдений в узле
доля наблюдений в узле от исходной выборки
вероятность положительного класса
правило разбиения

Сначала выводится номер правила, затем спрогнозированный класс, количество наблюдений в узле, доля наблюдений в узле от исходной выборки, вероятность положительного класса. Затем приводятся разбиения, в результате которых был получен узел. 7-е правило было получено в результате серии вышеприведенных разбиений.

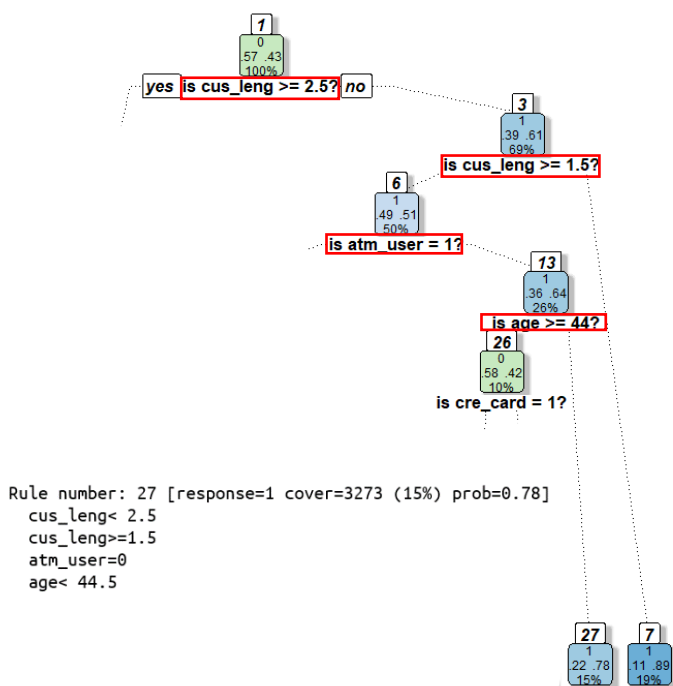


Рис. 3.18 Серия разбиений, в результате которых был получен узел 27

Теперь вычислим важности переменных.

```
# вычисляем важности переменных
model$variable.importance
```

Сводка 3.13. Важности переменных

cus_leng	age	atm_user	savings	curr_acc	cre_card
3993.5915311	1619.0428937	368.8030898	101.8555196	94.4689653	87.4262605
markpl	mortgage	internet	life_ins	mob_bank	deb_card
79.1442601	67.7550491	63.3971021	15.2147548	6.5070892	0.3054504

В данном случае важность предиктора – это сумма улучшений, вызванных применением данного предиктора в качестве основной расщепляющей переменной или переменной-суррогата. Наиболее важными стали количественные переменные *cus_leng* и *age*. Опять же помним про недостаток важности, она по сути складывается из частоты использования переменной, которая зависит от количества рассматриваемых точек расщепления, переменные с большим количеством уникальных значений будут получать преимущество при таком подходе.

3.2.3 Работа с диаграммой дерева

Для построения диаграммы дерева воспользуемся пакетами `gpart.plot` и `rattle`. Устанавливаем и загружаем пакет `gpart.plot`.

```
# устанавливаем пакет rpart.plot
# install.packages("rpart.plot")

# загружаем пакеты rpart.plot
library(rpart.plot)
```

Теперь с помощью функции `fancyRpartPlot` пакета `rattle` строим диаграмму дерева. Параметр `branch` задает форму ветвей. Для него можно значение в диапазоне от 0 (Λ-образные ветви) до 1 (Π-образные ветви). Параметр `space` и `yspace` настраивают горизонтальные и вертикальные размеры боксов, отображающих узлы. Параметр `split.cex` позволяет настроить размер шрифта для правил разбиения. Параметр `nn.cex` задает размер номеров узлов. Параметр `nn.font` задает тип шрифта для номеров узлов: 1 задает обычный шрифт, 2 задает жирный шрифт, 3 задает курсив (используется по умолчанию), 4 задает жирный курсив. Параметр `split.prefix` позволяет вставить текст перед определением узла. Параметр `split.suffix` позволяет вставить текст после определения узла.

```
# строим диаграмму дерева, branch задает
# форму ветвей, space и yspace
# настраивают горизонтальные и вертикальные
# размеры боксов, отображающих узлы,
# split.cex настраивает размер
# шрифта для определений узлов,
# nn.cex задает размер номеров узлов,
# nn.font задает тип шрифта
# для номеров узлов,
# split.prefix позволяет вставить
# текст перед определением узла,
# split.suffix позволяет вставить
# текст после определения узла
fancyRpartPlot(model,
  branch=0.5,
  space=0.001, yspace=0.001,
  split.cex=1.5,
  nn.cex=1, nn.font=4,
  split.prefix="is ",
  split.suffix="?")
```

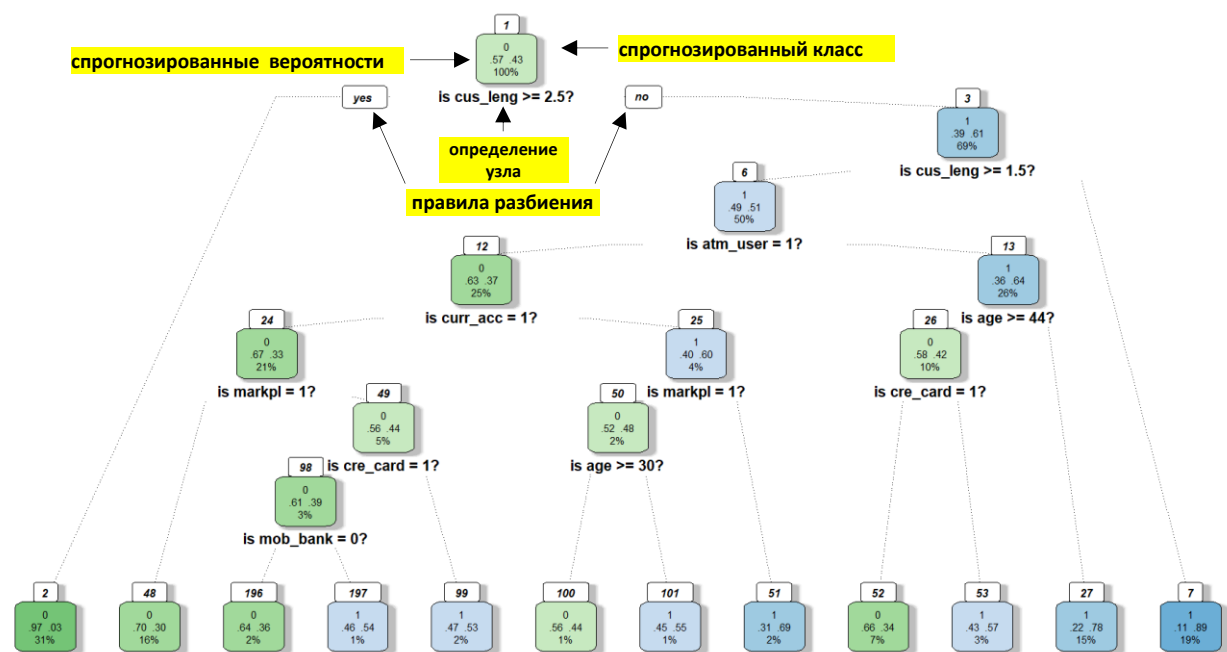


Рис. 3.19 Диаграмма дерева классификации CART

Диаграмма дерева (рисунок 3.19) наглядно показывает структуру дерева. В каждом узле первой записывается спрогнозированный класс зависимой переменной, затем спрогнозированные вероятности отсутствия и наличия отклика, а также доля наблюдений в узле от исходной выборки. Зеленым цветом отмечены узлы, для которых предсказан класс *Нет отклика* (0), а синим цветом – узлы, для которых предсказан класс *Есть отклик* (1).

Теперь расскажем, как давать определения узлам. Например, есть определение *is cus_leng >= 1?* (*Давность клиентской истории больше или равна 2,5 годам?*) и от него отходят левая и правая ветви. Левая ветвь всегда соответствует ответу *Да*, а правая – ответу *Нет*. Это означает, что левый узел – это клиенты, которые 2,5 года и больше пользуются услугами банка, а правый узел – клиенты, которые меньше 2,5 лет пользуются услугами банка. Возьмем другой пример. Есть определение *is mob_bank = 0?* (*Клиент не пользуется мобильным банком?*). Это означает, что левый узел – это клиенты, не пользующиеся мобильным банком, а правый узел – это клиенты, пользующиеся мобильным банком. Для правильной интерпретации диаграммы дерева все же лучше использовать результаты сводки 3.11. Главным предиктором отклика стала переменная *Давность клиентской истории [cus_leng]*. Она делит корневой узел (исходное множество клиентов) на два узла – покупателей с клиентской историей ≥ 2.5 (узел 2) и покупателей с клиентской историей < 2.5 (узел 3). При этом левому узлу-потомку (узлу 2) присвоен класс *Нет отклика*, а правому узлу-потомку (узлу 3) присвоен класс *Есть отклик*.

Узел 3 (покупатели с клиентской историей < 2.5) снова расщепляется по переменной *Давность клиентской истории* [*cus_leng*] на узел 6 (покупатели с клиентской историей ≥ 1.5) и узел 7 (покупатели с клиентской историей < 1.5). Обоим узлам присвоен класс *Есть отклик*. Кроме того, узел 7 является конечным и уже не расщепляется.

Узел 6 (покупатели с клиентской историей ≥ 1.5) разбивается по переменной *Пользование банкоматом за последнюю неделю* [*atm_user*] на узел 12 (клиенты, которые пользовались банкоматом за последнюю неделю) и узел 13 (клиенты, которые не пользовались банкоматом за последнюю неделю). Узлу 12 присвоен класс *Нет отклика*, а находящемуся справа узлу 13 присвоен класс *Есть отклик*.

Узел 12 (клиенты, которые пользовались банкоматом за последнюю неделю) разбивается по переменной *Наличие текущего счета* [*curr_acc*] на узел 24 (клиенты, у которых есть текущий счет) и узел 25 (клиенты, у которых нет текущего счета). Для узла 24 прогнозируется класс *Нет отклика*, а находящемуся справа узлу 25 присваивается класс *Есть отклик*.

Не вдаваясь подробно в интерпретацию дерева, можно отметить, что дерево имеет достаточно сложную, ветвистую структуру.

По умолчанию диаграмма дерева строится сверху вниз. С помощью параметра *yflip* мы можем изменить способ построения диаграммы.

```
# строим диаграмму дерева снизу вверх,  
# используя значение yflip=TRUE  
fancyRpartPlot(model,  
  branch=0.5,  
  space=0.001, yspace=0.001,  
  split.cex=1.5,  
  nn.cex=1, nn.font=4,  
  split.prefix="is ",  
  split.suffix="?", yflip=TRUE)
```

Теперь наше дерево построено снизу вверх (рис. 3.20).

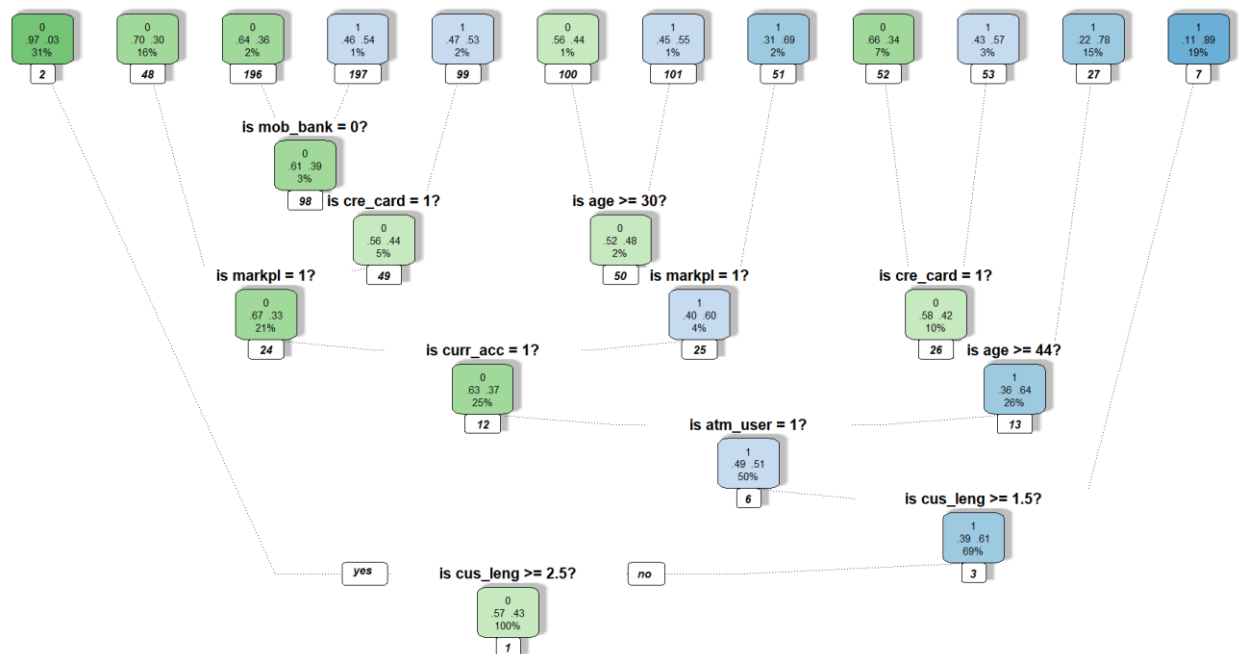


Рис. 3.20 Диаграмма дерева классификации CART (построение дерева снизу вверх)

С помощью параметра `xflip` мы можем поменять определения узлов на противоположные. Например, определение `is cus_leng>=1?` поменяется на определение `is cus_leng<1?`. Определение `is mob_bank=0` поменяется на определение `is mob_bank=1?`.

```
# строим диаграмму дерева, поменяв определения
# узлов на противоположные с помощью
# значения xflip=TRUE
fancyRpartPlot(model,
  branch=0.5,
  space=0.001, yspace=0.001,
  split.cex=1.5,
  nn.cex=1, nn.font=4,
  split.prefix="is ",
  split.suffix="?", xflip=TRUE)
```

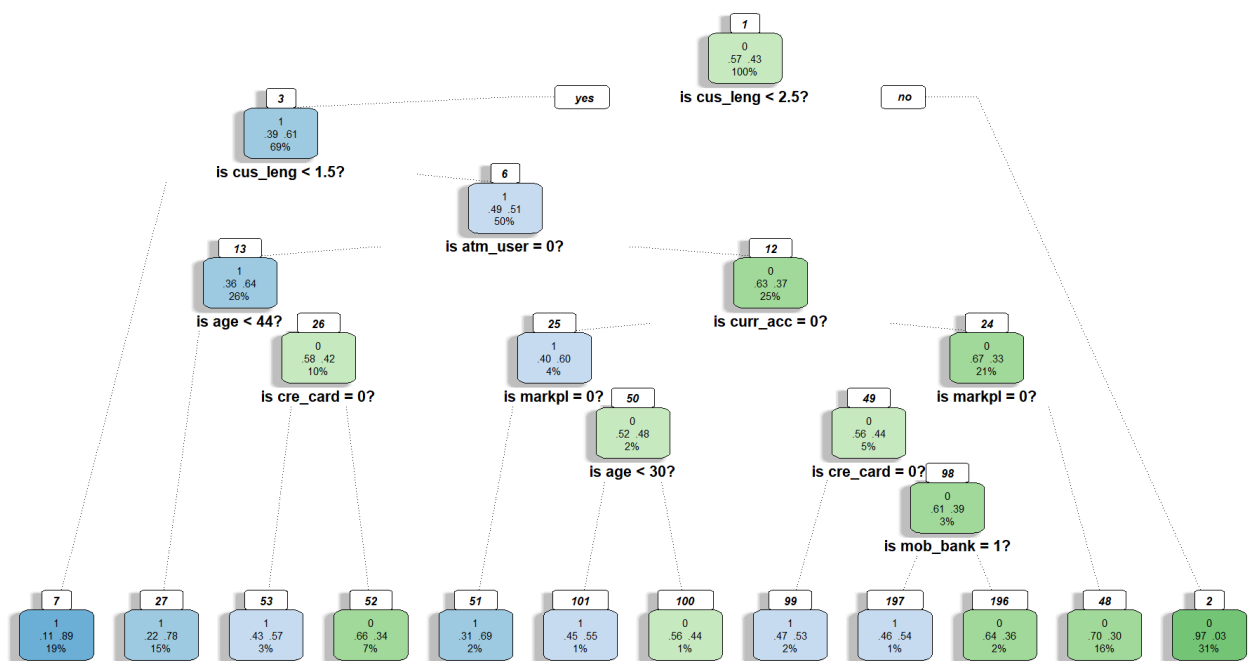


Рис. 3.21 Диаграмма дерева классификации CART (замена определений узлов на противоположные)

С помощью параметров `yes.text` и `no.text` мы можем поменять подписи `yes` и `no`, а с помощью параметра `yesno` можно изменить характер вывода этих подписей: `0` подавляет вывод подписей `yes` и `no`, `1` выводит подписи `yes` и `no` для самого верхнего разбиения (используется по умолчанию), `2` выводит подписи `yes` и `no` для всех разбиений. Давайте поменяем подписи `yes` и `no` на русские `да` и `нет`.

```

# строим диаграмму дерева, меняя подписи
# yes и no на русские да и нет
fancyRpartPlot(model,
  branch=0.5,
  space=0.001, yspace=0.001,
  split.cex=1.5,
  nn.cex=1, nn.font=4, split.prefix="is ",
  split.suffix="?", yes.text="Да", no.text="Нет")

```

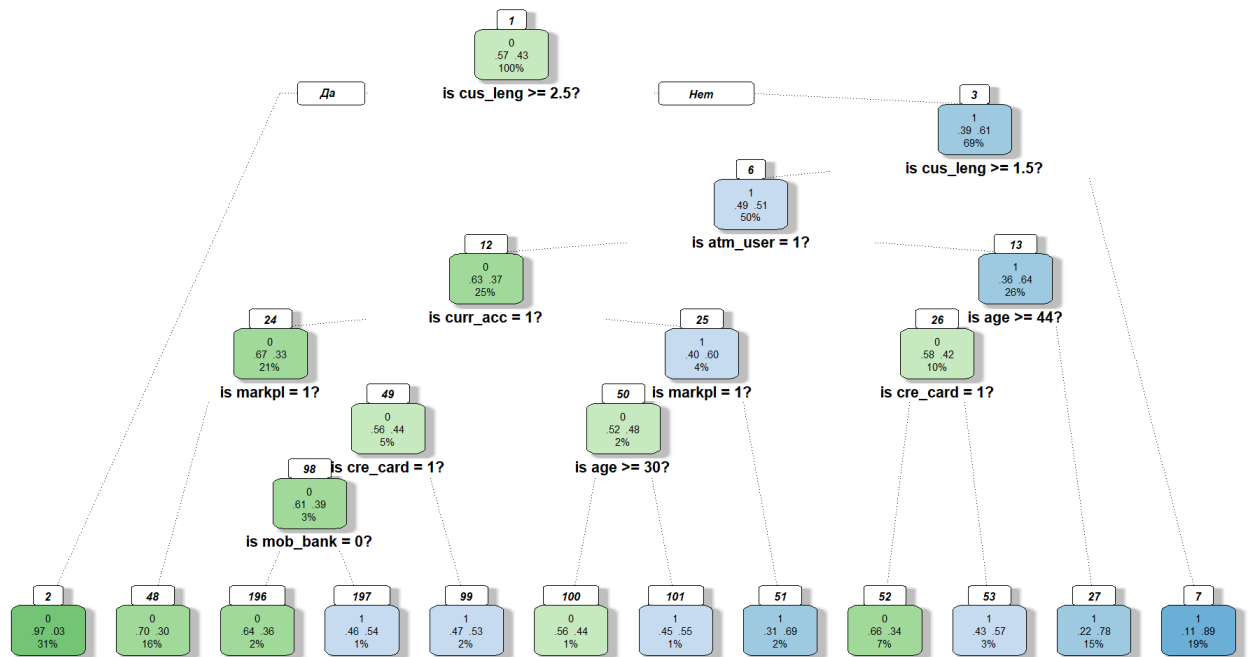



Рис. 3.22 Диаграмма дерева классификации CART
(замена подписей «yes» и «no» на русские «да» и «нет»)

Параметр `left` задает, в какую часть разбиения (левую или правую) относить узел, если условие верно. По умолчанию для этого параметра используется значение `TRUE`, означающее, что в левую часть разбиения у нас попадает узел, соответствующий положительному ответу на наш вопрос. Теперь в левую часть разбиения будем относить узел, соответствующий отрицательному ответу на наш вопрос.

```
# строим диаграмму дерева, теперь в левую
# часть разбиения будем записывать узел,
# соответствующий отрицательному ответу
# на вопрос
fancyRpartPlot(model,
  branch=0.5,
  space=0.001, yspace=0.001,
  split.cex=1.5,
  nn.cex=1, nn.font=4, split.prefix="is ",
  split.suffix="?", yes.text="Да", no.text="Нет", left=FALSE)
```

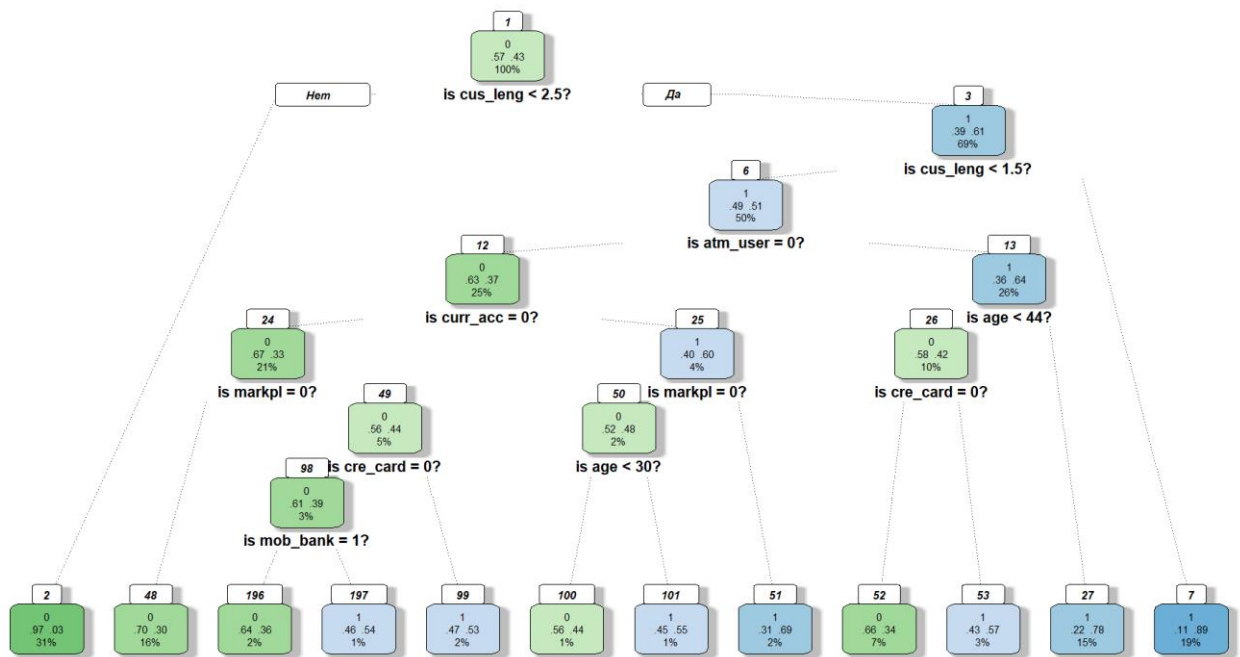


Рис. 3.23 Диаграмма дерева классификации CART (каждый узел, соответствующий отрицательному ответу на вопрос, записан в левой части разбиения)

Построить диаграмму дерева еще можно с помощью пакета `partykit`. Давайте установим и загрузим его.

```
# устанавливаем пакет partykit
# install.packages("partykit")
```

```
# загружаем пакет partykit
library(partykit)
```

Теперь переводим нашу модель дерева – объект класса `gpart` в объект класса `party` и визуализируем последний с помощью базовой функции `plot`.

```
# переводим модель дерева - объект класса
# gpart в объект класса party
model_party <- as.party(model)
```

```
# строим дерево party
plot(model_party)
```

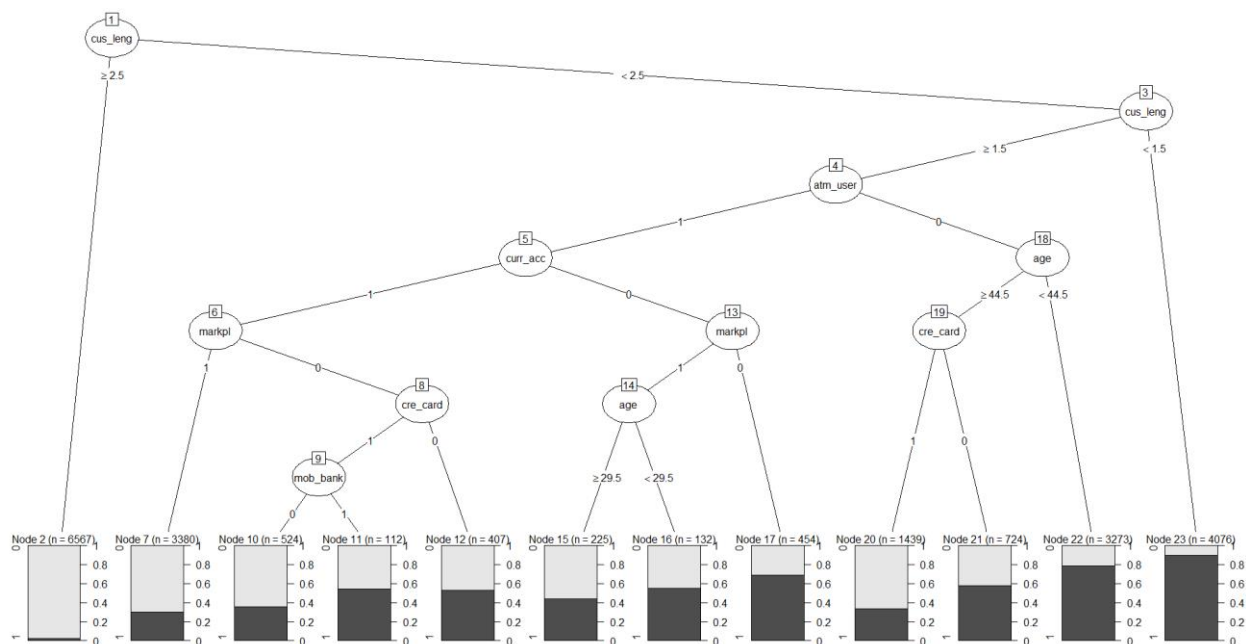


Рис. 3.24 Диаграмма дерева классификации CART, построенная с помощью пакета partykit

Если дерево является слишком большим, можно вывести лишь часть дерева – какой-то определенный узел. Например, выведем диаграмму для узла 18.

```
# выводим диаграмму для узла 18
model18 <- model_party[18]
plot(model18)
```

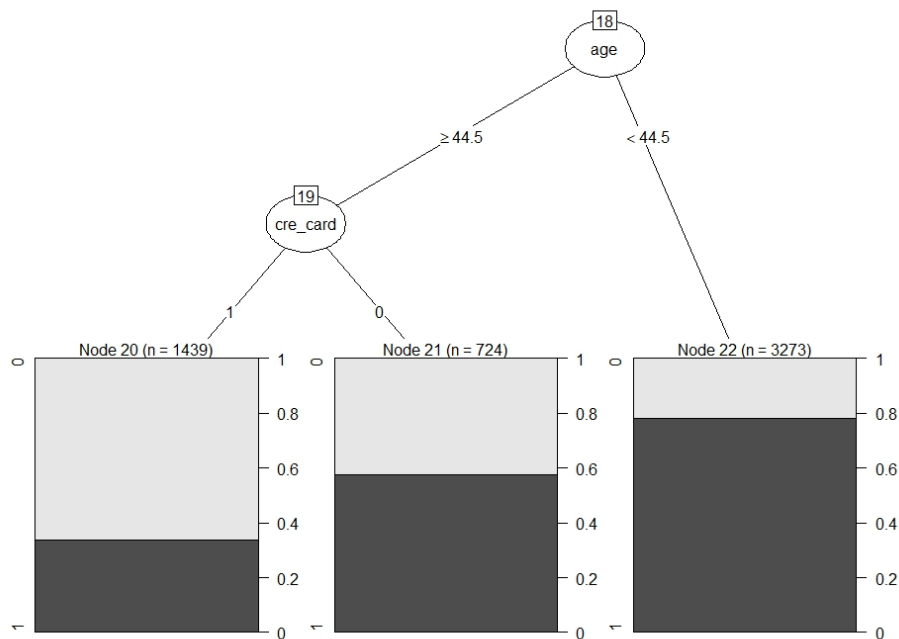


Рис. 3.25 Диаграмма для узла 18

Иногда возникает необходимость взглянуть на наблюдения, попавшие в конкретный узел. Для этого можно воспользоваться функцией `data_party` пакета `partykit`. Например, посмотрим, какие наблюдения относятся к узлу 11. Для краткости выведем первые 10 наблюдений узла 11.

```
# выясним, какие наблюдения
# относятся к узлу 11
node11 <- data_party(model_party, 11)

# выведем первые 5 наблюдений узла 11
head(node11, 5)
```

Сводка 3.14. Первые 5 наблюдений узла 11

	response	mortgage	life_ins	cre_card	deb_card	mob_bank	curr_acc	internet	perloan
422	0	1	1	1	1	1	1	0	0
893	1	0	1	1	1	1	1	0	0
1342	1	1	1	1	1	1	1	0	0
2056	1	0	1	1	1	1	1	0	0
2216	0	0	1	1	0	1	1	0	0

	savings	atm_user	markpl	age	cus_leng	(fitted)	(response)
422	0	1	0	21	2	11	0
893	0	1	0	24	2	11	1
1342	1	1	0	27	2	11	1
2056	0	1	0	28	2	11	1
2216	1	1	0	28	2	11	0

3.2.4 Прунинг дерева классификации CART

Вернемся к сводке, где приводится информация о сложности и качестве модели.

```
# выводим информацию о качестве модели
printcp(model)
```

Сводка 3.15. Оценка качества дерева классификации, используется ошибка классификации

Classification tree:

```
rpart(formula = response ~ ., data = train, method = "class",
      control = rpart.control(cp = 0.001))
```

Variables actually used in tree construction:

```
[1] age      atm_user cre_card curr_acc cus_leng markpl  mob_bank
```

Root node error: 9229/21313 = 0.43302

n= 21313

	CP	nsplit	rel error	xerror	xstd
1	0.3634197	0	1.00000	1.00000	0.0078380
2	0.0716221	1	0.63658	0.63658	0.0070684
3	0.0393325	3	0.49334	0.49334	0.0064835
4	0.0167949	4	0.45400	0.45400	0.0062867
5	0.0117022	5	0.43721	0.43721	0.0061971
6	0.0018420	6	0.42551	0.42551	0.0061327
7	0.0013002	7	0.42366	0.42670	0.0061394
8	0.0012461	8	0.42236	0.42540	0.0061321
9	0.0010835	10	0.41987	0.42551	0.0061327
10	0.0010000	11	0.41879	0.42616	0.0061363

сложность
номер разбиения
относительная ошибка
кросс-валидационная ошибка
стандартное отклонение кросс-валидационной ошибки

Давайте взглянем на столбец `rel error`. Мы видим, что с ростом числа разбиений и соответственно сложности дерева относительная ошибка модели, которая вычисляется на обучающей выборке, постепенно уменьшается. Однако если мы взглянем на столбец `xerror`, то здесь все будет не так радужно. Мы видим, что до определенного момента кросс-валидационная ошибка, вычисляемая по контрольным блокам перекрестной проверки уменьшается, а затем начинает возрастать. Таким образом, мы строим все более сложное дерево, которое начинает хуже прогнозировать наблюдения, не входившие в обучающее множество. Возникает необходимость найти оптимальное значение `cp`.

На практике используются два метода поиска оптимального значения `cp`. При первом методе берем такое значение `cp`, при котором в первый раз получили минимальную кросс-валидационную ошибку. Этот метод чаще всего применяют в тех случаях, когда имеется несколько `cp`, дающих одинаковую или примерно одинаковую наименьшую кросс-валидационную ошибку. В рамках данного метода оптимальным значением `cp` будет `cp=0.0012461`.

При втором методе мы находим наименьшее значение кросс-валидационной ошибки, прибавляем к нему стандартное отклонение кросс-валидационной ошибки, получаем пороговый уровень. Мы должны найти такое значение `cp`, которое дает наименьшее число разбиений и меньше этого порогового уровня. Например, в нашем случае минимальным значением кросс-валидационной ошибки будет значение 0.42540, прибавляем 0.0061321, получаем 0,4315321. В рамках данного

метода оптимальным значением `cp` будет `cp=0.0018420`. Этот метод учитывает изменчивость кросс-валидационной ошибки, возникающую в результате перекрестной проверки.

Ну и помимо методов полезно рассуждать логически. Мы видим, что начиная с `nsplit=6` кроссвалидационная ошибка достигает 0,42551 и перестает монотонно снижаться, при `nsplit=7` она становится больше, затем при `nsplit=8` уменьшается и потом снова начинает возрастать. Мы могли бы взять дерево с `nsplits=8`, дающее наименьшую кросс-валидационную ошибку 0,42540, однако скорее всего, данный результат представляет собой своего рода всплеск (вспоминаем про изменчивость кросс-валидационной ошибки в силу использования перекрестной проверки), ему не стоит доверять и лучше выбрать дерево с меньшим количеством расщеплений, то есть дерево с `nsplits=6` и `cp=0.0018420`. Чтобы убедиться в правильности выбранного решения, построим график зависимости кросс-валидационной ошибки от числа расщеплений и сложности модели.

```
# строим график зависимости кросс-валидационной
# ошибки от числа расщеплений и сложности модели
plotcp(model)
```

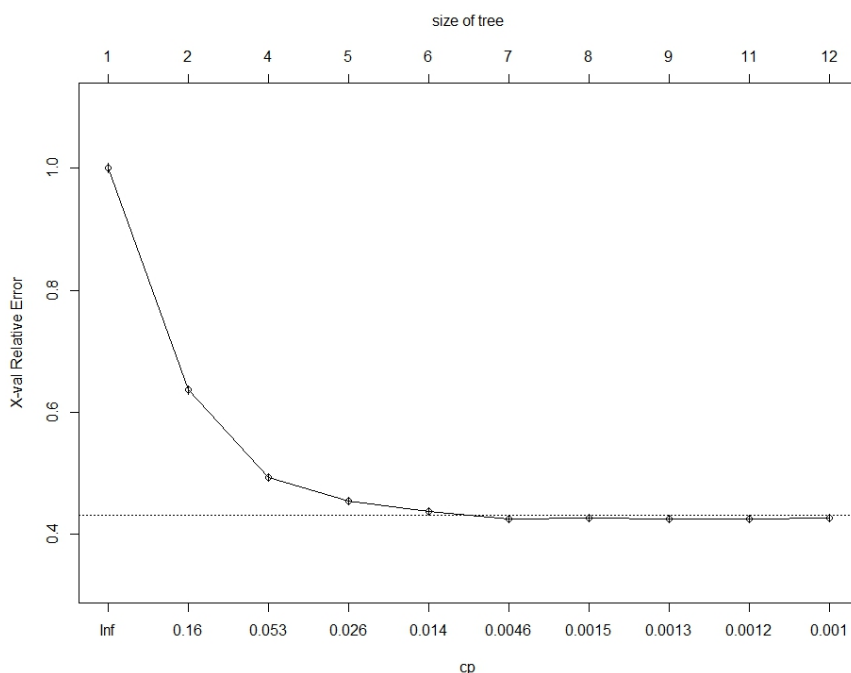


Рис. 3.26 График зависимости кросс-валидационной ошибки от числа расщеплений и сложности модели

На рис. 3.26 видно, что достигнув определенного размера дерева (`size of tree=8`), кросс-валидационная ошибка максимально уменьшается и остается на некотором постоянном уровне. Можно остановиться на значении кросс-валидационной ошибки, которое наблюдается при `cp=0.0015` и `nsplit=11`. Выберем `cp=0.0015`.

Теперь осуществим прунинг с помощью функции `prune` пакета `rpart`, установив новое пороговое значение сложности:

```
# строим обрезанное дерево
model2 <- prune(model, cp=0.0015)
```

Для наглядности выведем диаграмму обрезанного дерева (рис. 3.24).

```
# выводим диаграмму обрезанного дерева
fancyRpartPlot(model2,
  branch=0.5,
  space=0.001, yspace=0.001,
  split.cex=1.5,
  nn.cex=1, nn.font=4,
  split.prefix="is ",
  split.suffix="?")
```

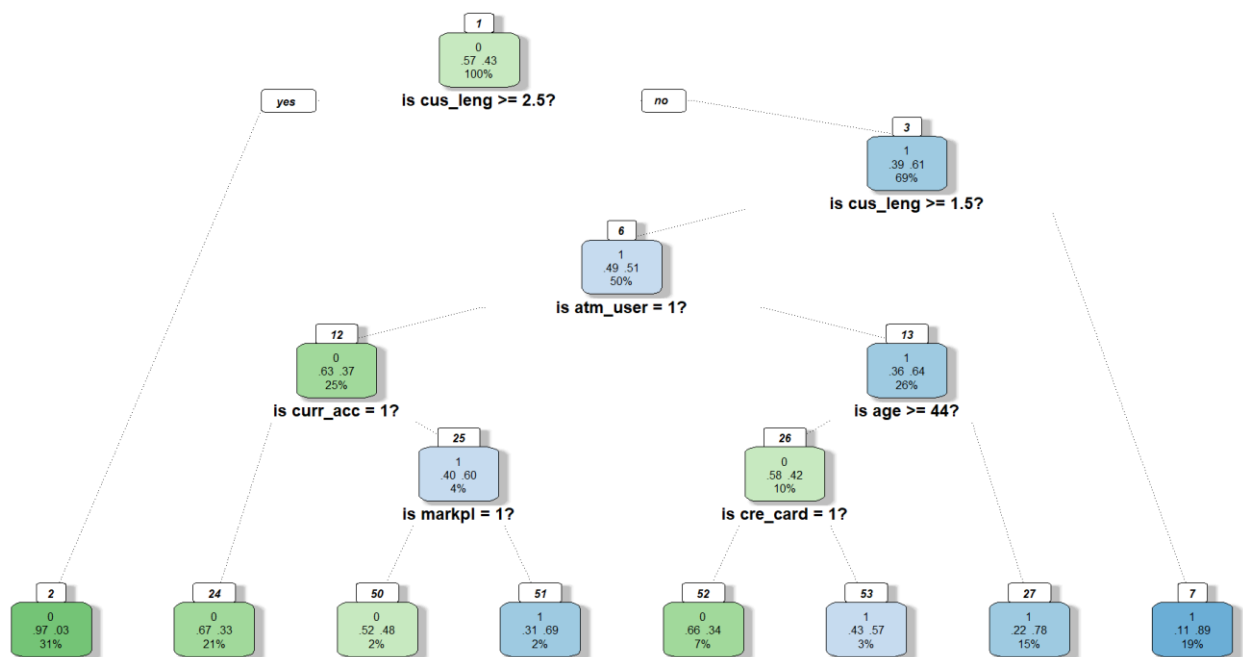


Рис. 3.27 Диаграмма дерева классификации CART (после прунинга)

Если вы затрудняетесь с определением оптимального размера, можно автоматически выбрать `cp` с наименьшей кросс-валидационной ошибкой, воспользовавшись следующим программным кодом:

```
# автоматически выбираем оптимальное
# значение cp
model3 <- prune(model, cp=model$cp.table[
  which.min(model$cp.table[, "xerror"]), "CP"])
```

Результат прунинга очевиден: при сохранении той же самой правильности модели мы получили более простое дерево.

3.2.5 Оценка качества модели

Теперь оценим качество обрезанного дерева классификации на контрольной выборке. Нас будет интересовать дискриминирующая способность и правильность.

```
# оцениваем дискриминирующую способность
# обрезанного дерева на контрольной выборке
library(pROC)
prob_valid <- predict(model2, valid, type="prob")
roc_valid <- roc(valid$response, prob_valid[,2], ci=TRUE)

# выводим доверительный интервал AUC
roc_valid
```

Сводка 3.16. Доверительный интервал AUC для обрезанного дерева классификации CART на контрольной выборке

Call:

```
roc.default(response = valid$response, predictor = prob_valid[, 2], ci = TRUE)
```

Data: prob_valid[, 2] in 5086 controls (valid\$response 0) < 3860 cases (valid\$response 1).

Area under the curve: 0.8812

95% CI: 0.8745-0.888 (DeLong)

Теперь выведем ROC-кривую для нашего обрезанного дерева.

```
# выводим ROC-кривую
plot.roc(roc_valid)
```

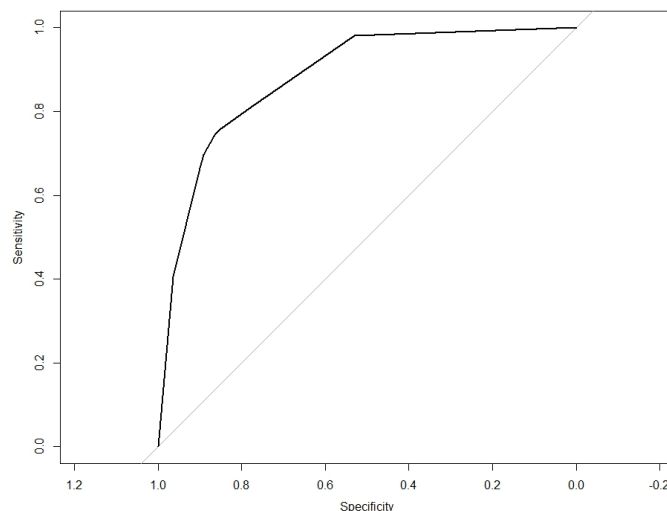


Рис. 3.28 ROC-кривая для обрезанного дерева классификации CART на контрольной выборке

До этого момента мы строили ROC-кривую с помощью пакета `pROC`. В среде R есть еще один прекрасный пакет для отрисовки ROC-кривой и вычисления AUC. Речь идет о пакете `ROCR`. Давайте установим и загрузим его.


```
# устанавливаем пакет ROCR
# install.packages("ROCR")
# загружаем пакет ROCR
library(ROCR)
```

Все начинается с создания объекта `prediction`. Мы создаем его с помощью функции `prediction` пакета `ROCR`. Функция `prediction` имеет общий вид:

```
prediction(predictions, labels)
```

где

predictions	Задаёт вектор, матрицу, список или датафрейм, содержащий спрогнозированные вероятности интересующего класса
labels	Задаёт вектор, матрицу, список или датафрейм, содержащий фактические метки классов. Должен иметь ту же самую размерность, что и predictions

```
# создаем объект prediction
pred <- prediction(prob_valid[,2], valid$response)
```

А затем нам нужно воспользоваться функцией `performance` пакета `ROCR`. Функция `performance` имеет общий вид:

```
prediction(prediction.obj, measure)
```

где

prediction.obj	Задаёт объект <code>prediction</code>
measure	Задаёт метрику качества. Метрик может быть несколько. Для построения ROC-кривой нужно задать " tpr " – чувствительность и " fpr " – 1-специфичность

Мы строим ROC-кривую, передав функции `performance` объект `prediction` (в данном случае он называется `pred`) и необходимые метрики. Параметр `lty` задаёт тип диагональной линии, обозначающей бесполезный классификатор.

```
# строим ROC-кривую
plot(performance(pred, "tpr", "fpr"))
abline(0, 1, lty = 4)
```

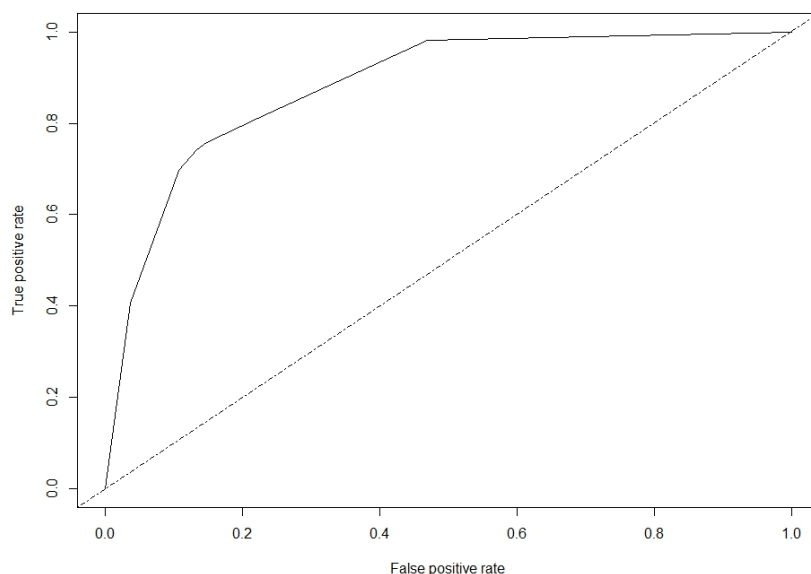


Рисунок 3.29 ROC-кривая для обрезанного дерева классификации CART, построенная с помощью пакета ROCR

Теперь вычислим значение AUC

```
# вычисляем AUC
AUC = performance(pred, "auc")@y.values[[1]]
AUC
```

Сводка 3.17. Значение AUC для обрезанного дерева классификации CART, вычисленное с помощью пакета ROCR

```
[1] 0.8812102
```

Теперь вычислим правильность модели на контрольной выборке. Для этого нам нужно получить спрогнозированные значения (классы) зависимой переменной и построить матрицу ошибок. Обратите внимание, что для получения спрогнозированных значений для параметра `type` функции `predict` нужно указать значение `"class"`.

```
# получаем спрогнозированные значения зависимой переменной
# для контрольной выборки
predvalue <- predict(model2, valid, type="class")

# строим матрицу ошибок
table(valid$response, predvalue)
```

Сводка 3.18. Матрица ошибок для обрезанного дерева классификации CART

```
predvalue
  0    1
0 4405 681
1  997 2863
```

Количество верных прогнозов делим на общее количество прогнозов $(4405+2863)/(4405+681+997+2863)=7268/8946=0,812$ или 81,2%.

Лекция 3.3 Построение и интерпретация дерева регрессии CART

3.3.1 Подготовка данных

Теперь построим дерево регрессии CART. На этот раз будем предсказывать задолженность по кредитной карте. Данные записаны в файле *Creddebt.csv*. Исходная выборка содержит записи о 5000 клиентах. По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

- количественный предиктор *Возраст в годах [age]*;
- порядковый предиктор *Уровень образования [ed]*
- количественный предиктор *Срок занятости на последнем месте работы в месяцах [employ]*;
- количественный предиктор *Срок проживания по последнему адресу [address]*;
- количественный предиктор *Ежемесячный доход в тысячах рублей [income]*;
- количественный предиктор *Отношение суммы долговых обязательств к доходу [debtinc]*;
- количественная зависимая переменная *Задолженность по кредитной карте в тысячах рублей [creddebt]*.

Запустим программу R. Запишем данные в объект **data**, чтобы начать работу с ними.

```
# загружаем данные
data <- read.csv2("C:/Trees/Creddebt.csv")
```

Проверим, как выглядят наши переменные.

```
# смотрим типы переменных
str(data)
```

Сводка 3.19. Таблица данных data

```
'data.frame':    5000 obs. of  7 variables:
 $ age      : int  41 30 40 41 57 45 36 39 43 34 ...
 $ ed       : Factor w/ 5 levels "Высшее, ученая степень",...: 5 3 3 3 3 3 3 3 5 ...
 $ employ   : int  17 13 15 15 7 0 1 20 12 7 ...
 $ address  : int  12 8 14 14 37 13 3 9 11 12 ...
 $ income   : num  35.9 46.7 61.8 72 25.6 28.1 19.6 80.5 68.7 33.8 ...
 $ debtinc  : num  11.9 17.9 10.6 29.7 15.9 ...
 $ creddebt : num  0.5 1.35 3.44 4.17 1.5 0.92 1.21 1.85 1.43 1.42 ...
```

Порядковую переменную *ed*, ставшую в R фактором, преобразуем в упорядоченный фактор.

```
# порядковую переменную ed преобразовываем
# в упорядоченный фактор
data$ed <- ordered(data$ed, levels = c("Неполное среднее", "Среднее", "Среднее специальное",
                                       "Незаконченное высшее", "Высшее, ученая степень"))
```

Снова проверим, как выглядят переменные.

```
# смотрим типы переменных
str(data)
```

Сводка 3.20. Таблица данных data после преобразования

```
'data.frame':    5000 obs. of  7 variables:
 $ age      : int  41 30 40 41 57 45 36 39 43 34 ...
 $ ed       : Ord.factor w/ 5 levels "Неполное среднее"<...: 3 1 1 1 1 1 1 1 3 ...
 $ employ   : int  17 13 15 15 7 0 1 20 12 7 ...
 $ address  : int  12 8 14 14 37 13 3 9 11 12 ...
 $ income   : num  35.9 46.7 61.8 72 25.6 28.1 19.6 80.5 68.7 33.8 ...
 $ debtinc  : num  11.9 17.9 10.6 29.7 15.9 ...
 $ creddebt : num   0.5 1.35 3.44 4.17 1.5 0.92 1.21 1.85 1.43 1.42 ...
```

Разбиваем набор данных на обучающую и контрольную выборки.

```
# задаем стартовое значение генератора случайных чисел, чтобы
# каждый раз получать одно и то же разбиение
# на обучающую и контрольную выборки
set.seed(100)

# разбиваем набор на обучающую и контрольную выборки
ind <- sample(2, nrow(data), replace=TRUE, prob=c(0.7, 0.3))
tr <- data[ind==1,]
val <- data[ind==2,]
```

Проверяем наличие пропусков.

```
# проверяем наличие пропусков в обучающей выборке
sapply(tr, function(x) sum(is.na(x)))
```

Сводка 3.21. Отчет по пропущенным значениям в датафрейме tr

age	ed	employ	address	income	debtinc	creddebt
0	0	0	0	0	0	0

```
# проверяем наличие пропусков в контрольной выборке
sapply(val, function(x) sum(is.na(x)))
```

Сводка 3.22. Отчет по пропущенным значениям в датафрейме val

age	ed	employ	address	income	debtinc	creddebt
0	0	0	0	0	0	0

3.3.2 Построение и интерпретация модели регрессии

Поскольку наша зависимая переменная *creddebt* является количественной, строится дерево регрессии. Построим модель дерева классификации по всем исходным предикторам, вновь уменьшив пороговое значение штрафа за сложность *cp*. Обратите внимание, теперь для параметра *method* мы задаем значение 'anova', а не 'class'.

```
set.seed(42)
# строим модель регрессии, уменьшив пороговое
# значение штрафа за сложность cp
m <- rpart(creddebt ~ ., method='anova', control=rpart.control(cp=0.001),
           data=tr)
```

Выведем информацию о качестве модели.

```
# выводим информацию о качестве модели
printcp(m)
```

Сводка 3.23. Оценка качества дерева регрессии, используется сумма квадратов остатков (девианс)

Regression tree:

```
rpart(formula = creddebt ~ ., data = tr, method = "anova", control = rpart.control(cp = 0.001))
```

Variables actually used in tree construction:

```
[1] address debtinc employ income
```

Root node error: 38848/3510 = 11.068

n= 3510

	CP	nsplit	rel error	xerror	xstd
1	0.2059577	0	1.00000	1.00077	0.49690
2	0.0848917	1	0.79404	1.05816	0.49800
3	0.0484284	2	0.70915	0.89849	0.43597
4	0.0381777	3	0.66072	0.84844	0.43448
5	0.0223124	4	0.62254	0.81357	0.43444
6	0.0080364	5	0.60023	0.71401	0.36431
7	0.0079322	6	0.59220	0.70344	0.36429
8	0.0063324	7	0.58426	0.70085	0.36429
9	0.0058589	8	0.57793	0.69436	0.36428
10	0.0054694	9	0.57207	0.69082	0.36425
11	0.0050726	10	0.56660	0.69026	0.36425
12	0.0048728	11	0.56153	0.68006	0.36347
13	0.0046986	12	0.55666	0.67790	0.36346
14	0.0037390	13	0.55196	0.67588	0.36339
15	0.0032180	14	0.54822	0.66633	0.36335
16	0.0029638	15	0.54500	0.66420	0.36335
17	0.0025281	16	0.54204	0.66286	0.36335
18	0.0022085	17	0.53951	0.65964	0.36335
19	0.0018628	18	0.53730	0.65736	0.36334
20	0.0014130	19	0.53544	0.65807	0.36334
21	0.0012199	20	0.53403	0.65918	0.36334
22	0.0011118	21	0.53281	0.66044	0.36334
23	0.0010998	22	0.53169	0.66016	0.36334
24	0.0010371	23	0.53059	0.65968	0.36334
25	0.0010000	25	0.52852	0.65957	0.36334

сложность
номер разбиения
относительная ошибка
кросс-валидационная ошибка
стандартное отклонение кросс-валидационной ошибки

Root node error показывает базовую ошибку модели регрессии. Она вычисляется как общая сумма квадратов отклонений, поделенная на количество наблюдений. Общая сумма квадратов отклонений – это сумма квадратов отклонений фактических значений зависимой переменной от ее среднего значения (общая сумма квадратов отклонений). Давайте вычислим ее.

```
# вычисляем общую сумму квадратов отклонений
TSS <- sum((tr$creddebt - (mean(tr$creddebt)))^2)
TSS
```

Сводка 3.24. Общая сумма квадратов отклонений

38847.63

Параметр **rel error** эквивалентен статистике $1 - R^2$ в регрессионном анализе. R^2 или коэффициент детерминации показывает долю дисперсии зависимой переменной, объясненную моделью. Таким образом, **rel error** показывает долю дисперсии зависимой переменной, которую не смогла объяснить модель на обучающей выборке. Давайте подробнее поговорим об R^2 . Он вычисляется как единица минус отношение остаточной суммы квадратов отклонений к общей сумме квадратов отклонений. Остаточная сумма квадратов отклонений – это сумма квадратов отклонений фактических значений зависимой переменной от спрогнозированных (ее называют остаточной суммой квадратов отклонений). Общая сумма квадратов отклонений – это сумма квадратов отклонений фактических значений зависимой переменной от ее среднего.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

где

\bar{y}_i – среднее значение зависимой переменной;

\hat{y}_i – спрогнозированное значение зависимой переменной;

y_i – фактическое значение зависимой переменной.

Коэффициент детерминации меняется в диапазоне от 0 до 1. Чем ближе значение коэффициента детерминации к 1, тем меньше дисперсия случайной ошибки модели по сравнению с общей дисперсией зависимой переменной.

Например, вычислим R^2 модели на обучающей выборке для **ср=0.0018628** (именно при этом значении **ср** достигается минимальная кросс-валидационная ошибка). Вычитаем из 1 значение 0,53730 и получаем 0,4627. Это обозначает, что модель на обучающей выборке объяснила 46,3% дисперсии зависимой переменной. R^2 еще можно было

вычислить так: $0,53730$ умножаем на общую сумму квадратов отклонений 38848 , получаем 20873 , это будет остаточная сумма квадратов отклонений. Из 1 вычитаем остаточную сумму квадратов отклонений, поделенную на общую сумму квадратов отклонений $1-(20873/38848)=1-0,537=0,463$ или $46,3\%$.

Параметр `хеггор` – это кроссвалидационная ошибка. Для дерева регрессии кроссвалидационная ошибка – это сумма квадратов остатков, вычисленная для объектов, не участвовавших в обучении¹. С помощью этого показателя мы можем вычислить R^2 модели, если применим ее к наблюдениям, не участвовавшим в обучении. Например, вычислим R^2 модели по таким наблюдениям для `ср=0.0018628`. Вычитаем из 1 значение $0,65736$ и получаем $0,34264$. R^2 еще можно было вычислить так: $0,65736$ умножаем на общую сумму квадратов отклонений 38848 , получаем 25537 , это будет остаточная сумма квадратов отклонений. Из 1 вычитаем остаточную сумму квадратов отклонений, поделенную на общую сумму квадратов отклонений $1-(25537/38848)=1-0,657=0,343$ или $34,3\%$. Это обозначает, что если нашу модель использовать на новых данных, она объяснит $34,3\%$ дисперсии зависимой переменной. В целом можно сказать, что наша модель имеет неудовлетворительное качество.

3.2.3 Прунинг и оценка качества дерева регрессии CART

Чтобы выбрать оптимальное значение `ср`, построим график зависимости кроссвалидационной ошибки от числа расщеплений и сложности модели.

```
# строим график зависимости кросс-валидационной  
# ошибки от числа расщеплений и сложности модели  
plotcp(m)
```

¹ В этом смысле она эквивалентна PRESS статистике, которая часто используется в регрессионном анализе с целью оценить подгонку модели по наблюдениям, которые не использовались для оценивания параметров модели. PRESS статистика вычисляется как сумма квадратов остатков, предсказанных для этих наблюдений.

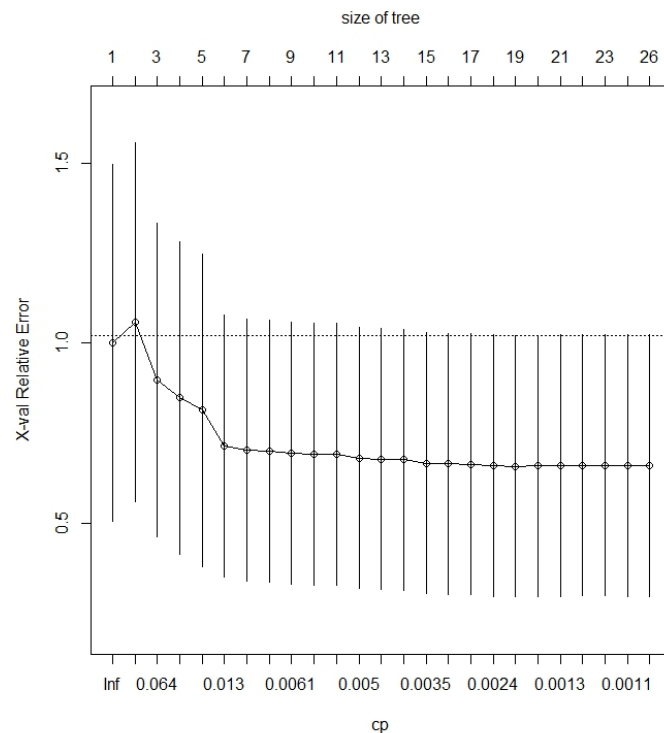


Рисунок 3.30 График зависимости кросс-валидационных ошибок от числа расщеплений и сложности модели

На графике мы видим, что по достижении `size of tree=19` кросс-валидационная ошибка перестает уменьшаться. В целом график довольно сложен и поэтому можно автоматически вычислить оптимальное значение `cp`, а затем по этому значению обрезать дерево.

```
# находим оптимальное значение cp
cp=m$cpstable[which.min(m$cpstable[, "xerror"]),"CP"]
cp
```

Сводка 3.25. Оптимальное значение `cp`

```
[1] 0.001862835
```

```
# выполняем прунинг
m2 <- prune(m, cp=0.001862835)
```

Выводим диаграмму обрезанного дерева.

```
# строим диаграмму обрезанного
# дерева регрессии
fancyRpartPlot(m2,
  branch=0.5,
  space=0.1, yspace=0.01,
  split.cex=1,
  nn.cex=0.5, nn.font=1,
  split.prefix="is ",
  split.suffix="?")
```

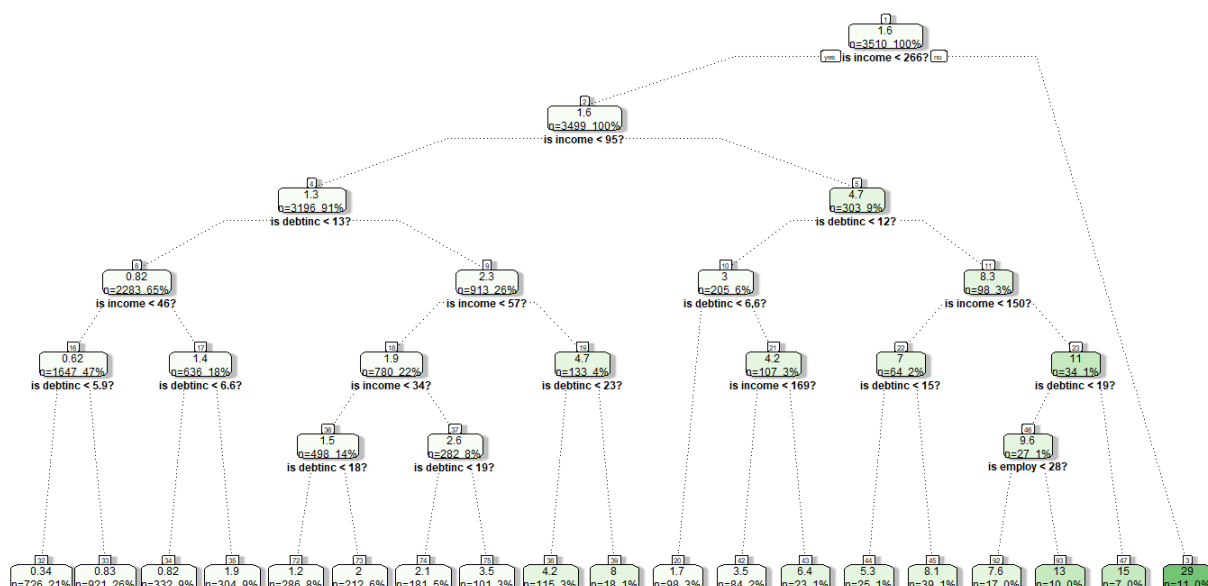



Рис. 3.31 Диаграмма дерева регрессии CART (после прунинга)

В узлах дерева регрессии первым записывается спрогнозированное среднее значение зависимой переменной, количество наблюдений и доля наблюдений в узле от исходной выборки.

Главным предиктором задолженности по кредитной карте стала переменная *Ежемесячный доход в тысячах рублей [income]*. Она делит корневой узел на два узла – клиентов с доходом меньше 266 тыс. рублей (узел 2) и клиентов, у которых доход равен 266 тыс. рублям или выше (узел 3). Средний размер задолженности у клиентов с доходом < 266 тыс. рублей равен 1,6 тыс. рублей, а средний размер задолженности у клиентов с доходом ≥ 266 тыс. рублей составляет 29 тыс. рублей. Примечательно, что узел 3 состоит всего из 11 наблюдений и является терминальным.

Узел 3 (клиенты с доходом < 266 тыс. рублей) вновь расщепляется по переменной *Ежемесячный доход в тысячах рублей [income]* на узел 4 (клиенты с доходом < 95 тыс. рублей) и узел 5 (клиенты с доходом ≥ 95 тыс. рублей). Средний размер задолженности у клиентов с доходом < 95 тыс. рублей равен 1,3 тыс. рублей, а средний размер задолженности у клиентов с доходом ≥ 95 тыс. рублей составляет 4,7 тыс. рублей.

Опять же не будем подробно вдаваться в интерпретацию дерева, а лучше в целом охарактеризуем его, исходя из диаграммы. Большое количество терминальных узлов очень небольших размеров указывает на то, что несмотря на прунинг дерево остается сложным, очень гибким и будет улавливать несущественные возмущения в данных, принимая их за сигнал. Все это говорит о нестабильности модели. Теперь вычислим девианс и среднеквадратичную ошибку модели на контрольной выборке.

Для этого нам нужно получить спрогнозированные значения зависимой переменной на контрольной выборке.

```
# вычисляем спрогнозированные значения зависимой  
# переменной на контрольной выборке  
predvalue <- predict(m2, val)
```

Получив спрогнозированные значения зависимой переменной, вычисляем девианс или сумму квадратов остатков. Для этого по каждому наблюдению получаем остаток – разность между фактическим и спрогнозированным значениями зависимой переменной, этот остаток возводим в квадрат, квадраты остатков складываем и получаем девианс.

```
# вычисляем девианс или сумму квадратов остатков  
# на контрольной выборке, из фактического значения  
# зависимой переменной вычитаем спрогнозированное  
# значение, возводим остаток в квадрат и квадраты  
# остатков суммируем  
D <- sum((val$creddebt - predvalue)^2)
```

Сводка 3.26. Девианс обрезанного дерева регрессии

```
[1] 6367.12
```

```
# вычисляем среднеквадратичную ошибку, сумму квадратов  
# делим на количество наблюдений  
MSE <- sum((val$creddebt - predvalue)^2)/nrow(val)
```

Сводка 3.27. Среднеквадратичная ошибка обрезанного дерева регрессии

```
[1] 4.273235
```

Кроме девианса и среднеквадратичной ошибки полезно взглянуть на коэффициент детерминации для контрольной выборки.

```
# вычисляем коэффициент детерминации  
TSS <- sum((val$creddebt - mean(val$creddebt))^2)  
RSS <- sum((val$creddebt - predvalue)^2)  
R2 <- 1 - (RSS/TSS)  
R2
```

Сводка 3.28. Коэффициент детерминации обрезанного дерева регрессии

```
[1] 0.1793884
```

Наша модель дерева объясняет лишь 18% дисперсии зависимой переменной, что указывает на ее неудовлетворительное качество. Давайте для сравнения построим модель линейной регрессии и вычислим для нее девианс, среднеквадратичную ошибку и коэффициент детерминации. При этом помним, чем меньше девианс и среднеквадратичная ошибка,

тем лучше качество модели, а для коэффициента детерминации все наоборот: чем он выше, тем лучше качество модели.

```
# строим модель линейной регрессии
linearMod <- lm(creddebt ~ ., data=tr)

# получаем спрогнозированные значения
# зависимой переменной с помощью
# модели линейной регрессии
predvalue_regr <- predict(linearMod, val)

# вычисляем девианс для модели линейной регрессии
D_regr <- sum((val$creddebt - predvalue_regr)^2)
D_regr
```

Сводка 3.29. Девианс модели линейной регрессии

```
[1] 4300.803
```

```
# вычисляем среднеквадратичную ошибку
# для модели линейной регрессии
MSE_regr <- sum((val$creddebt - predvalue_regr)^2)/nrow(val)
MSE_regr
```

Сводка 3.30. Среднеквадратичная ошибка модели линейной регрессии

```
[1] 2.886445
```

```
# вычисляем коэффициент детерминации
# для модели линейной регрессии
TSS <- sum((val$creddebt - (mean(val$creddebt)))^2)
RSS <- sum((val$creddebt - predvalue_regr)^2)
R2 <- 1 - (RSS/TSS)
R2
```

Сводка 3.31. Коэффициент детерминации модели линейной регрессии

```
[1] 0.4457009
```

Анализ метрик, полученных для линейной регрессии, показывает, что данная модель имеет более высокое качество по сравнению с моделью дерева. Модель линейной регрессии объясняет 45% дисперсии зависимой переменной (при этом можно добиться лучшего качества, воспользовавшись регуляризацией), тогда как модель дерева объясняет лишь 18%. Чем это обусловлено? Мы должны вспомнить то, о чем говорили в нашей вводной лекции по деревьям решений. Деревья решений более эффективны по сравнению с регрессионным анализом в тех случаях, когда взаимосвязи между предикторами и зависимой переменной являются нелинейными, переменные имеют несимметричные распределения, наблюдается большое количество коррелирующих между собой переменных, взаимодействия высоких порядков. Если же предпосылки регрессионного анализа выполняются, то логистическая регрессия (когда зависимая переменная является категориальной) или линейная регрессия (когда зависимая переменная является количественной) могут дать лучший результат. Это обусловлено тем, что деревья пытаются описать линейную связь между переменными путем

многократных разбиений по предикторам. CHAID делает это за счет расщепления сразу на несколько категорий, CART пытается уловить эту связь посредством серии бинарных делений и это может быть менее эффективно по сравнению с подбором параметров в регрессионном анализе. По-видимому, взаимосвязи между предикторами и зависимой переменной в нашем наборе носили линейный характер, распределения переменных были близки к нормальному распределению и применение линейной регрессии оказалось более эффективным. Кроме того, не забываем про недостаток CART – склонность к переобучению и нестабильность, которые не всегда можно устранить даже с помощью прунинга. Однако если на основе исходной выборки случайным образом создать множество немного отличающихся друг от друга выборок, построить по ним деревья регрессии CART, позволив каждому переобучаться на какой-то своей определенной части данных и при этом для поиска наилучшего расщепления каждого узла использовать не все предикторы, а лишь случайное подмножество, затем усреднить полученные результаты, можно добиться значительного улучшения качества. Этот подход реализован в рамках метода случайного леса для задачи регрессии. Давайте построим модель случайного леса для наших данных и вычислим R^2 модели на контрольной выборке.

```
# загружаем пакет randomForest для  
# построения случайного леса  
library(randomForest)  
  
# строим модель случайного леса  
set.seed(42)  
forestMod <- randomForest(creddebt ~ ., data=tr)  
  
# получаем спрогнозированные значения  
# зависимой переменной с помощью  
# модели случайного леса  
predvalue_forest <- predict(forestMod, val)  
  
# вычисляем коэффициент детерминации  
# для модели случайного леса  
TSS <- sum((val$creddebt - (mean(val$creddebt)))^2)  
RSS <- sum((val$creddebt - predvalue_forest)^2)  
R2 <- 1 - (RSS/TSS)  
R2
```

Сводка 3.32. Коэффициент детерминации
модели случайного леса

```
[1] 0.6296923
```

Модель случайного леса позволяет объяснить 63% дисперсии зависимой переменной.

Вопросы к модулю 3

1. Найдите `nsplit` с минимальной кросс-валидационной ошибкой. Ответом является конкретное значение `nsplit`.

Classification tree:

```
rpart(formula = default ~ ., data = data, method = "class", cp = 0.001)
```

Variables actually used in tree construction:

```
[1] children conttel credits gender jobtel marital
```

Root node error: 1407/3523 = 0.39938

n= 3523

	CP	nsplit	rel error	xerror	xstd
1	0.0097133	0	1.00000	1.00000	0.020661
2	0.0071073	3	0.97086	1.00569	0.020681
3	0.0046198	4	0.96375	1.00355	0.020673
4	0.0039090	6	0.95451	0.99787	0.020654
5	0.0031983	8	0.94670	0.99360	0.020639
6	0.0028429	10	0.94030	0.99360	0.020639
7	0.0021322	11	0.93746	0.98081	0.020592
8	0.0017768	13	0.93319	0.97512	0.020571
9	0.0014215	15	0.92964	0.97157	0.020557
10	0.0010661	19	0.92395	0.97726	0.020579
11	0.0010000	23	0.91969	0.97299	0.020562

2. Применительно к сводке, приведенной выше, выберите правильный вариант ответа.
- a. Модель неправильно классифицировала 36.8% наблюдений обучающей выборки
 - b. Модель правильно классифицировала 36.8% наблюдений обучающей выборки
 - c. Модель неправильно классифицировала 38.8% наблюдений обучающей выборки
 - d. Модель правильно классифицировала 38.8% наблюдений обучающей выборки
 - e. Модель правильно классифицировала 92% наблюдений обучающей выборки
 - f. Модель правильно классифицировала 97% наблюдений обучающей выборки
3. Применительно к сводке, приведенной выше, выберите правильный вариант ответа.
- a. Используя модель на новых данных, мы получим 36.8% неверно классифицированных наблюдений
 - b. Используя модель на новых данных, мы получим 36.8% верно классифицированных наблюдений
 - c. Используя модель на новых данных, мы получим 97% верно классифицированных наблюдений

- d. Используя модель на новых данных, мы получим 38.8% неверно классифицированных наблюдений
- e. Используя модель на новых данных, мы получим 38.8% верно классифицированных наблюдений