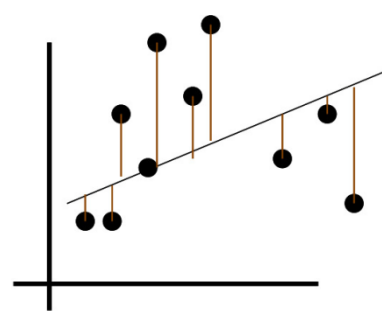
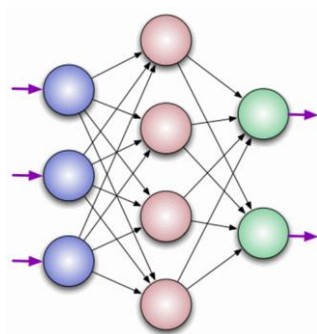
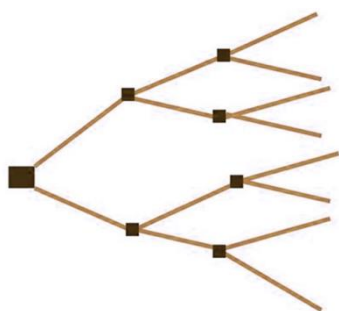




Артем Груздев

Прогнозное моделирование в R и Python

Модуль 2. Построение деревьев решений CHAID с помощью пакета R CHAID



СОДЕРЖАНИЕ

Модуль 2 Построение деревьев решений CHAID с помощью пакета R CHAID	3
Лекция 2.1 Знакомство с методом CHAID	3
2.1.1 Описание алгоритма	3
2.1.2 Немного о тесте хи-квадрат.....	5
2.1.3 Способы объединения категорий предикторов.....	7
2.1.4 Поправка Бонферрони	7
2.1.5 Иллюстрация работы CHAID на конкретном примере.....	8
Лекция 2.2 Предварительная подготовка данных перед построением модели дерева CHAID.....	14
2.2.1 Загрузка данных	14
2.2.2 Фиксация пустых строковых значений как пропусков	16
2.2.3 Изменение типов переменных	18
2.2.4 Обработка дублирующихся наблюдений	20
2.2.5 Вывод подробной информации о переменных	22
2.2.6 Нормализация строковых значений	25
2.2.7 Обработка редких категорий	28
2.2.8 Однократное случайное разбиение набора данных на обучающую и контрольную выборки для проверки модели.....	30
2.2.9 Импутация пропусков	34
2.2.10 Биннинг количественных переменных для получения порядковых предикторов	42
Лекция 2.3 Построение модели дерева CHAID и ее проверка	48
2.3.1 Построение модели и работа с диаграммой дерева	48
2.3.2 Получение прогнозов модели.....	51
2.3.3 Работа с матрицей ошибок.....	56
2.3.4 Знакомство с ROC-кривой и AUC.....	65
2.3.5 Построение ROC-кривой и вычисление AUC для модели	78
2.3.6 Вычисление интервальных оценок AUC для модели	79

Модуль 2 Построение деревьев решений CHAID с помощью пакета R CHAID

Лекция 2.1 Знакомство с методом CHAID

2.1.1 Описание алгоритма

Перед началом работы алгоритма CHAID необходимо преобразовать все имеющиеся количественные предикторы в порядковые переменные. Обычно их разбивают на 10 категорий одинакового объема.

Алгоритм приступает к построению дерева, итеративно применяя к каждому узлу, начиная с корневого, процедуры объединения категорий, расщепления узла и проверки правил остановки.

Этап 1. Объединение категорий

1. Для каждого предиктора с числом категорий больше двух¹ алгоритм ищет пару категорий, меньше всего статистически различающихся по зависимой переменной (т.е. с наибольшим p -значением). Для этого он вычисляет p -значения, выполняя тесты хи-квадрат Пирсона. Алгоритм строит двухвходовую таблицу сопряженности с категориями предиктора в качестве строк и категориями зависимой переменной в качестве столбцов и проверяет гипотезу о независимости двух переменных.

2. Найдя наибольшее p -значение для пары категорий, алгоритм сравнивает его с заданным уровнем значимости для объединения категорий.

Если p -значение:

- меньше или равно заданному уровню значимости для объединения категорий – алгоритм переходит к вычислению скорректированных p -значений для полученного набора категорий (шаг 3);
- больше уровня значимости для объединения категорий – эта пара объединяется в отдельную составную категорию, в результате формируется новый набор категорий предиктора и процесс начинается заново с поиска пары категорий с наибольшим p -значением.

ПРИМЕЧАНИЕ

В пакете R CHAID уровень значимости для объединения категорий можно настроить с помощью параметра `alpha2` вспомогательной функции `chaid_control`.

¹ Если предиктор имеет одну категорию, он исключается из анализа. Если предиктор имеет две категории, происходит переход к шагу 3.

(Опционно) Если новая составная категория состоит из трех и более исходных категорий, алгоритм находит внутри этой составной категории наилучшее бинарное расщепление, которое дает наименьшее p -значение. Алгоритм выполняет бинарное расщепление, если его p -значение не превышает уровня значимости для разбиения объединенных категорий.

ПРИМЕЧАНИЕ

В пакете R CHAID уровень значимости для разбиения уже объединенных категорий можно настроить с помощью параметра `alpha3` вспомогательной функции `chaid_control`.

3. Для сформированного набора категорий предиктора алгоритм вычисляет скорректированное p -значение как p -значение, умноженное на поправку Бонферрони. Поправка Бонферрони представляет собой число возможных способов, с помощью которых исходные категории предиктора могут быть объединены в итоговые категории.

Этап 2. Расщепление узла

После вычисления скорректированных p -значений для итоговых наборов категорий по всем предикторам алгоритм переходит к этапу расщепления узла.

1. На этапе расщепления алгоритм выбирает, какой предиктор обеспечит наилучшее разбиение узла. Для этого предиктор должен иметь наименьшее скорректированное p -значение (т.е. является наиболее статистически значимым).

2. Найдя предиктор с наименьшим скорректированным p -значением, алгоритм сравнивает его с заданным уровнем значимости для расщепления.

Если p -значение:

- меньше или равно заданному уровню значимости для расщепления – алгоритм разбивает узел с использованием данного предиктора (категории предиктора становятся дочерними узлами);
- больше заданного уровня значимости для расщепления, то алгоритм не расщепляет узел и узел рассматривается как терминальный.

ПРИМЕЧАНИЕ

В пакете R CHAID уровень значимости для расщепления узла можно настроить с помощью параметра `alpha4` вспомогательной функции `chaid_control`.

Этап 3. Остановка

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

1. Если текущая глубина дерева достигает заданной пользователем максимальной глубины дерева, процесс построения дерева останавливается.
2. Если количество наблюдений в родительском узле меньше заданного пользователем минимума наблюдений в родительском узле, узел не разбивается.
3. Если минимальное абсолютное количество наблюдений в терминальном узле меньше заданного пользователем минимума наблюдений в терминальном узле, узел не разбивается.
4. Если минимальная относительная частота наблюдений в терминальном узле меньше заданной пользователем минимальной относительной частоты наблюдений в терминальном узле, узел не разбивается.

ПРИМЕЧАНИЕ

В пакете R CHAID с помощью ряда параметров вспомогательной функции `chaid_control` можно изменить некоторые вышеперечисленные правила остановки:

- `minsplit` задает минимальное количество наблюдений в родительском узле перед расщеплением, по умолчанию 20;
- `minbucket` задает минимальное абсолютное количество наблюдений в терминальном узле, по умолчанию 7;
- `minprob` задает минимальную относительную частоту наблюдений в терминальном узле, по умолчанию 0.01;
- `maxheight` задает максимальную высоту дерева, по умолчанию равен -1, т.е. не используется.

2.1.2 Немного о тесте хи-квадрат

Предположим, алгоритм проверяет, различаются ли значимо категории предиктора *Семейное положение* *Холост* и *Женат* по зависимой переменной *Отклик*. Нулевая гипотеза звучит так: категории предиктора не отличаются друг от друга с точки зрения распределения категорий зависимой переменной. Альтернативная гипотеза заключается в том, что категории предиктора все же отличаются друг от друга по зависимой переменной. Строится двухвходовая таблица сопряженности, где строки являются категориями предиктора *Семейное положение*, а столбцы – категориями зависимой переменной *Отклик*. Наблюдения распределились так, как показано в таблице ниже:

	Нет отклика	Есть отклик	Всего
Холост	20	13	33
Женат	30	37	67
Всего	50	50	100

Согласно нулевой гипотезе ожидаемые частоты имеют вид:

	Нет отклика	Есть отклик	Всего
Холост	16,50	16,50	33
Женат	33,50	33,50	67
Всего	50	50	100

Хи-квадрат вычисляется по формуле:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

где O – наблюдаемые частоты, а E – ожидаемые частоты.

$$\chi^2 = \frac{(20 - 16,50)^2}{16,50} + \frac{(13 - 16,50)^2}{16,50} + \frac{(30 - 33,50)^2}{33,50} + \frac{(37 - 33,50)^2}{33,50}$$

$$= 0,74 + 0,74 + 0,37 + 0,37 = 2,216$$

Статистика хи-квадрат подчиняется распределению хи-квадрат со степенями свободы $df = (R - 1)(C - 1)$, где R и C – количество строк и столбцов в таблице сопряженности. В нашем случае количество степеней свобод будет равно $df = (2 - 1)(2 - 1) = 1$.

Чтобы выяснить, достаточно ли велико полученное значение хи-квадрат для отклонения нулевой гипотезы, вычисляем соответствующую ему p -значение. P -значение – это вероятность ошибки, заключающейся в отклонении нулевой гипотезы, когда она верна. Более строго, это вероятность того, что случайная величина, имеющая распределение хи-квадрат при условии верности нулевой гипотезы, примет значение, не меньшее, чем вычисленное значение хи-квадрат. Решение об отклонении нулевой гипотезы принимается в результате сравнения p -значения с определенным пороговым уровнем, который называют уровнем значимости. Обычно p -значение сравнивают с общепринятым стандартным уровнем значимости 0,05. Если найденное p -значение меньше уровня значимости, нулевую гипотезу отклоняют, в противном случае у нас нет оснований отвергнуть нулевую гипотезу.

P -значение находят по таблице распределения хи-квадрат, для удобства вычислений можно воспользоваться онлайн-сервисами типа <http://www.socscistatistics.com/tests/chisquare/>. В нашем случае значение хи-квадрат 2,216 с одной степенью свободы соответствует p -значению 0,1366. Таким образом, вероятность того, что статистика хи-квадрат примет вычисленное значение 2,216 и выше, когда категории предиктора *Семейное положение* не отличаются друг от друга с точки зрения распределения категорий зависимой переменной *Отклик*, составляет

0,1366. Это превышает уровень значимости 0,05. У нас нет оснований отвергнуть нулевую гипотезу. Можно сделать вывод, что категории переменной *Семейное положение* действительно не отличаются друг от друга с точки зрения распределения неоткликнувшихся и откликнувшихся клиентов.

Обратите внимание, фраза «нет оснований отклонить нулевую гипотезу» не тождественна фразе «принять нулевую гипотезу», которая является неверной. Нулевая гипотеза обычно имеет очень конкретную формулировку. Например, она может звучит так: нет статистически значимой разницы между средним значением выборки и средним значением генеральной совокупности. Если мы не можем отклонить нулевую гипотезу, обозначает ли это, что наши оценки равны? Вовсе не обязательно. Отсутствие оснований отклонить нулевую гипотезу означает, что нам просто не удалось найти статистически значимой разницы между этими оценками, но это вовсе не означает, что мы нашли равенство этих оценок друг другу.

2.1.3 Способы объединения категорий предикторов

Способ объединения категорий предиктора зависит от шкалы его измерения.

В номинальных предикторах можно объединять любые категории, если они не различаются значимо по зависимой переменной. Таким образом, для номинальных переменных ограничения на объединение категорий не накладываются.

В порядковых предикторах две категории могут быть объединены, только если к ним могут быть присоединены промежуточные категории. Например, переменная, представляющая группы по уровню доходов, может рассматриваться как порядковая. Людей с доходом менее 2000\$ имеет смысл объединять с теми, кто зарабатывает более 3000\$, только если к вновь образовавшейся группе можно также отнести людей с доходом от 2000\$ до 3000\$.

2.1.4 Поправка Бонферрони

Осуществляя поиск незначимых категорий предиктора для объединения, CHAID выполняет большое количество статистических тестов для различных комбинаций категорий предиктора. Однако число таких комбинаций зависит от количества категорий, которое у каждой переменной разное. По одной переменной может оцениваться 2 варианта объединения, рассматриваться 2 таблицы сопряженности и выполняться 2 статистических теста, а по другой переменной – 10 вариантов объединения, 10 таблиц сопряженности и 10 статистических тестов.

Вероятность того, что из 10 тестов хи-квадрат для второй переменной по крайней мере один из тестов дает ложное отклонение нулевой гипотезы составляет $1 - \prod_{i=1}^{10} (1 - \alpha_i)$. Групповая вероятность ошибки намного больше индивидуальной вероятности ошибки α_i . Например, если индивидуальная вероятность ошибки (α_i) по каждому тесту = 0,05, то групповая вероятность ошибки $1 - 0,95^{10} = 0,401$. Таким образом, при осуществлении множественных проверок гипотез при помощи хи-квадрат (одна проверка на каждое возможное объединение), p -значения недооценивают риск отклонения нулевой гипотезы, когда она верна. Например, вы можете сделать ошибочный вывод, что заемщики с разными профессиями отличаются по кредитоспособности, тогда как они на самом деле не отличаются.

Допустим, мы получили три p -значения: 0.01, 0.02 и 0.005. Если мы хотим, чтобы групповая вероятность ошибки при этом не превышала определенный уровень значимости α (например, 0.05), то, согласно методу Бонферрони, мы должны сравнить каждое из полученных p -значений не с α , а с α / m , где m – это количество возможных вариантов объединения I исходных категорий предиктора в g итоговых категорий. Деление исходного уровня значимости α на m – это и есть *классическая поправка Бонферрони*. В рассматриваемом примере каждое из полученных p -значений необходимо было бы сравнить с $0.05/3 = 0.017$. В результате мы получили, что p -значение для второй гипотезы (0.02) превышает 0.017 и, соответственно, у нас нет оснований отвергнуть эту гипотезу.

Вместо деления изначально принятого уровня значимости на число возможных вариантов объединения, мы могли бы умножить каждое исходное p -значение на это число и получить скорректированное p -значение. Именно этот вариант чаще всего используют в современных реализациях алгоритмов CHAID. Сравнив такие скорректированные p -значения с α , мы пришли бы к точно тем же выводам, что и при использовании классического варианта поправки Бонферрони:

$0.01 * 3 = 0.03 < 0.05$: гипотеза отклоняется;

$0.02 * 3 = 0.06 > 0.05$: нет оснований отклонить нулевую гипотезу;

$0.005 * 3 = 0.015 < 0.05$: гипотеза отклоняется.

В ряде случаев при умножении исходных p -значений на уровень значимости результат может превысить 1. По определению, вероятность не может быть больше 1, и если это происходит, то получаемое значение просто приравнивают к 1.

2.1.5 Иллюстрация работы CHAID на конкретном примере

Предположим, есть данные по клиентам микрофинансовой организации и известно, выплатили они займ или нет (категориальная зависимая

переменная *Просрочка*). В качестве потенциальных предикторов фигурируют четыре переменных: *Доход*, *Возраст*, *Сфера занятости*, *Пол*. Переменные *Пол*, *Сфера занятости* являются номинальными, переменная *Доход* и *Возраст* являются порядковыми. Переменная *Сфера занятости* принимает значения *Работает по найму*, *Свое дело*, *Учится или студент*, *Пенсионер*. Переменная *Доход* принимает значения *Менее 10 тыс. рублей*, *От 10 до 25 тыс. рублей*, *От 26 до 40 тыс. рублей*, *Более 40 тыс. рублей*. Переменная *Пол* принимает значения *Женщины* и *Мужчины*. Переменная *Возраст* принимает значения от 18 до 74 лет. Необходимо выяснить, какие группы клиентов с большей вероятностью выйдут в просрочку, чтобы сосредоточить внимание на них. Схематично наши исходные данные представлены на рисунке 2.1.

Имеется набор данных (корневой узел)

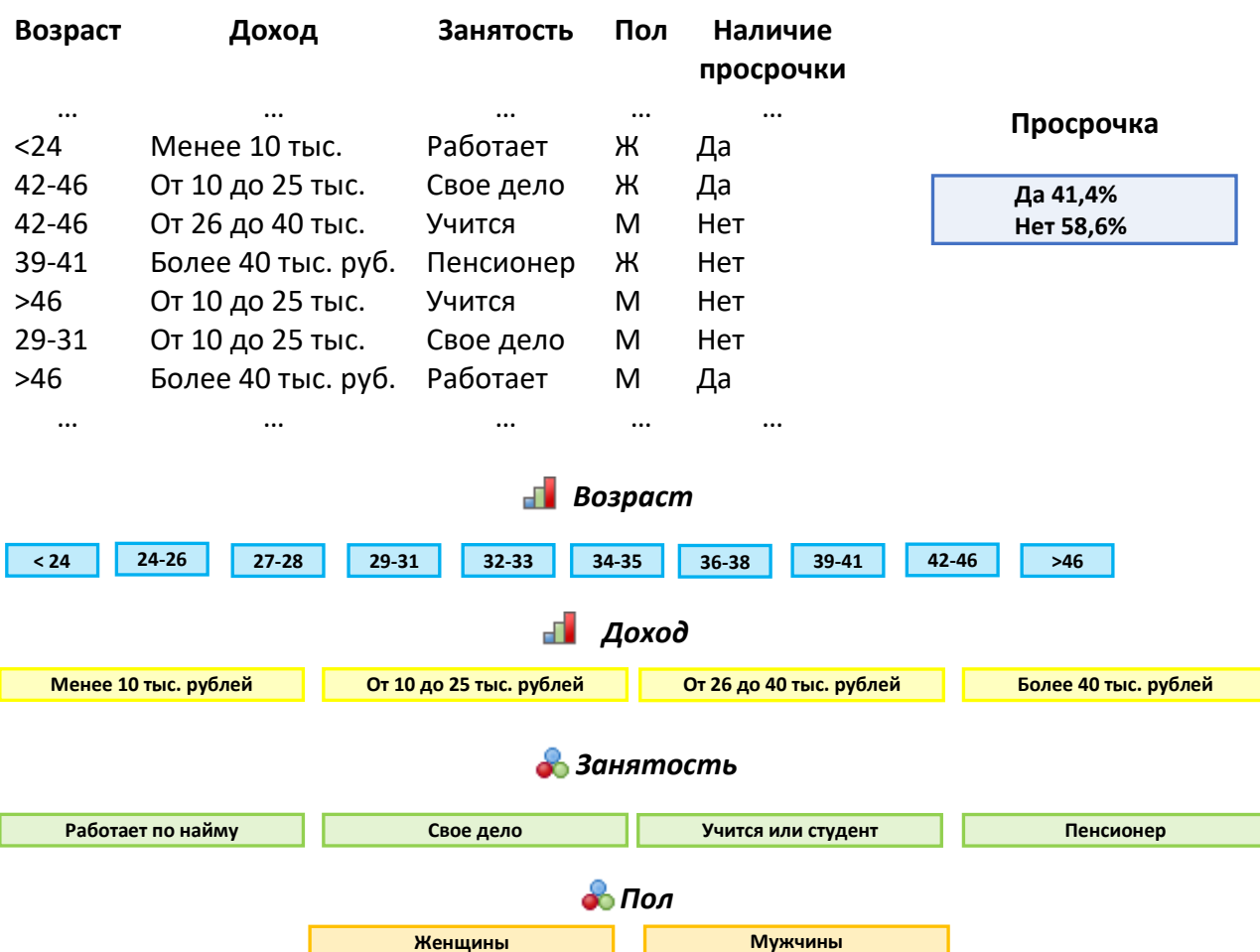


Рисунок 2.1 Исходные данные перед началом работы CHAID

Традиционным решением этой задачи могло бы быть построение набора таблиц сопряженности и анализ того, какие переменные по отдельности связаны с кредитоспособностью. Однако поскольку кредитоспособность может зависеть от комбинаций демографических характеристик, то

необходимо изучить трехвходовые (трехмерные), четырехвходовые таблицы и, возможно, таблицы еще большей размерности, что потребует больших временных и интеллектуальных затрат. Конечно, для построения модели взаимосвязи переменных в многовходовой таблице сопряженности можно воспользоваться лог-линейными моделями, но они обычно требуют больших выборок. Например, если имеются пять предикторов с пятью категориями, один предиктор с двумя категориями и зависимая переменная с двумя категориями, то таблица размерности 7, для которой нужно построить лог-линейную модель, будет иметь $5 \times 5 \times 5 \times 2 \times 2$ или 12500 ячеек. Необходимая для этого выборка будет огромной.

Что же делает CHAID, когда мы запускаем его? Вместо полных таблиц большой размерности CHAID строит двухвходовые таблицы сопряженности между каждым предиктором и зависимой переменной и выполняет тесты на значимость на основе критерия хи-квадрат. Сначала выполняются тесты для объединения категорий. По каждому предиктору CHAID берет пару категорий, сравнивает, различаются ли они по зависимой переменной, и объединяет их, если они не показывают этого различия (дают p -значение больше заданного уровня значимости для объединения).

В нашем случае по порядковому предиктору *Доход* CHAID сравнивает категорию *Менее 10 тыс. рублей* с категорией *От 10 до 25 тыс. рублей*, затем категорию *От 10 до 25 тыс. рублей* с категорией *От 26 до 40 тыс. рублей*, затем категорию *От 26 до 40 тыс. рублей* с категорией *Более 40 тыс. рублей*. Еще раз обратите внимание, что в порядковом предикторе несмежные категории (например, категория *менее 10 тыс. рублей* и категория *Более 40 тыс. рублей*) сравниваться и объединяться не могут. Допустим, категории предиктора *Доход Менее 10 тыс. рублей* и *От 10 до 25 тыс. рублей* значимо не различаются по кредитоспособности (в обеих категориях наблюдается высокая доля «плохих» заемщиков), имеют наибольшее p -значение. Тогда CHAID объединяет их и формирует новый набор категорий (объединенная категория *Менее 10 тыс. рублей/От 10 до 25 тыс. рублей*, категория *От 26 до 40 тыс. рублей*, категория *Более 40 тыс. рублей*) и снова начинает процесс сравнения. Процесс объединения категорий остановится, когда все оставшиеся категории предиктора будут различаться на заданном уровне значимости для объединения. Сформировав новый набор категорий по предиктору *Доход* алгоритм начинает аналогичным образом формировать набор категорий для порядкового предиктора *Возраст*, опять же сравнивая и объединяя только смежные категории. Затем алгоритм формирует набор категорий для номинального предиктора *Сфера занятости*. Здесь уже CHAID может сравнивать и объединять любые категории переменной. Затем переходит к предиктору *Пол*. Здесь категории предиктора *Пол* не

могут быть объединены, поскольку у этой переменной только два уровня. Процесс сравнения и объединения категорий по каждому предиктору показан на рисунке 2.2.

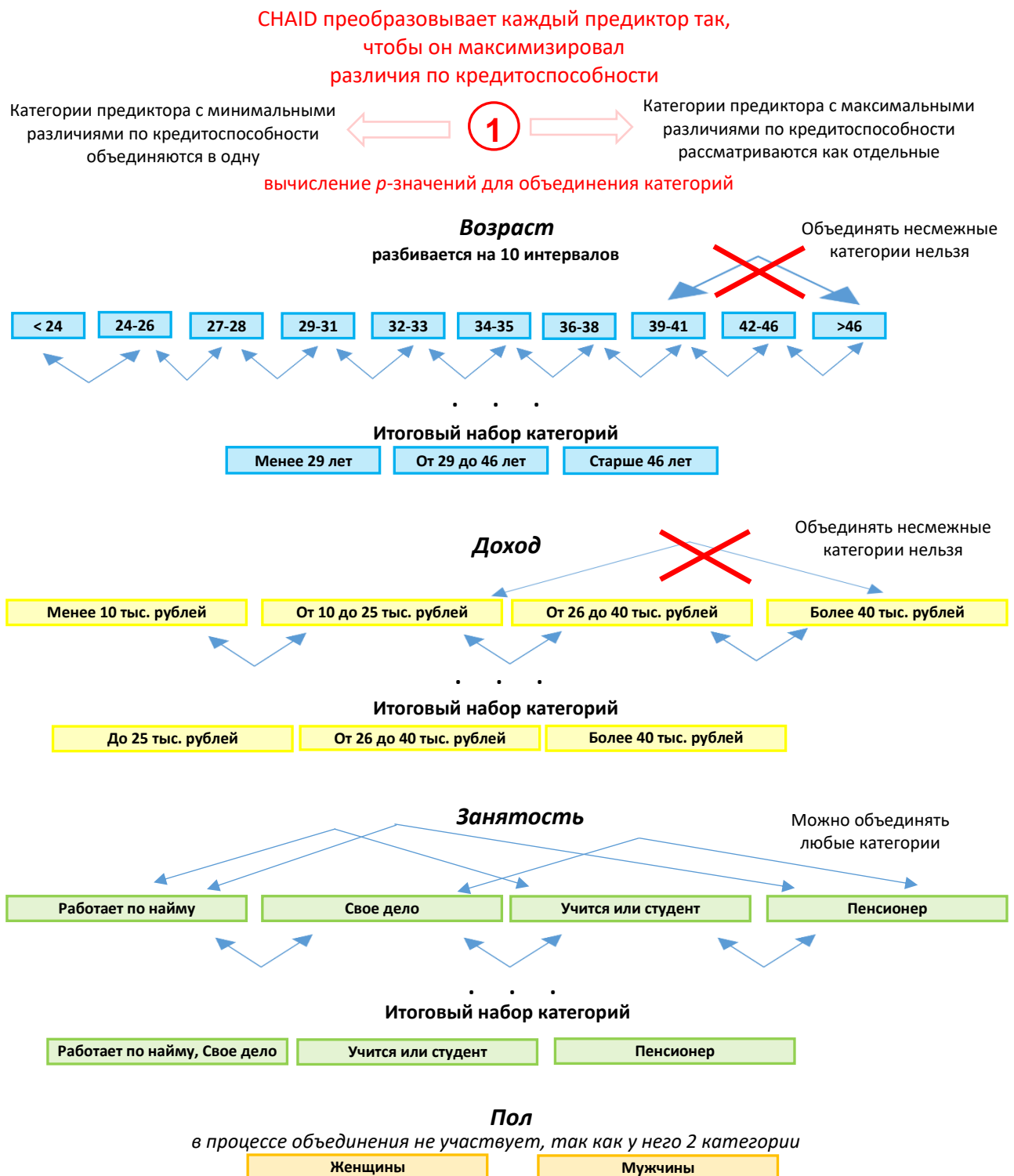


Рисунок 2.2 Объединение категорий предикторов

Завершив этап объединения, CHAID запускает тесты на значимость для разбиения узла (рисунок 2.3), в них участвуют преобразованные предикторы *Доход* (допустим, набор из 3 категорий: объединенная

категория *Менее 10 тыс. рублей*/От 10 до 25 тыс. рублей, категория От 26 до 40 тыс. рублей и категория *Более 40 тыс. рублей*), *Сфера занятости* (набор из 3 категорий: объединенная категория *Работает по найму/Свое дело*, категория *Учится или студент*, категория *Пенсионер*), *Возраст* (набор из 3 категорий: объединенная категория *Менее 29 лет*, объединенная категория *От 29 до 46 лет* и объединенная категория *Старше 46 лет*) и предиктор *Пол* (категория *Женщины* и категория *Мужчины*). При этом помним, что p -значения для предикторов корректируются с помощью поправки Бонферрони, чтобы учесть количество сравнений категорий по каждому предиктору.

CHAID выбирает предиктор, лучше всего
максимизирующий различия по
кредитоспособности, из набора преобразованных

2

вычисление скорректированных p -значений для разбиения узла

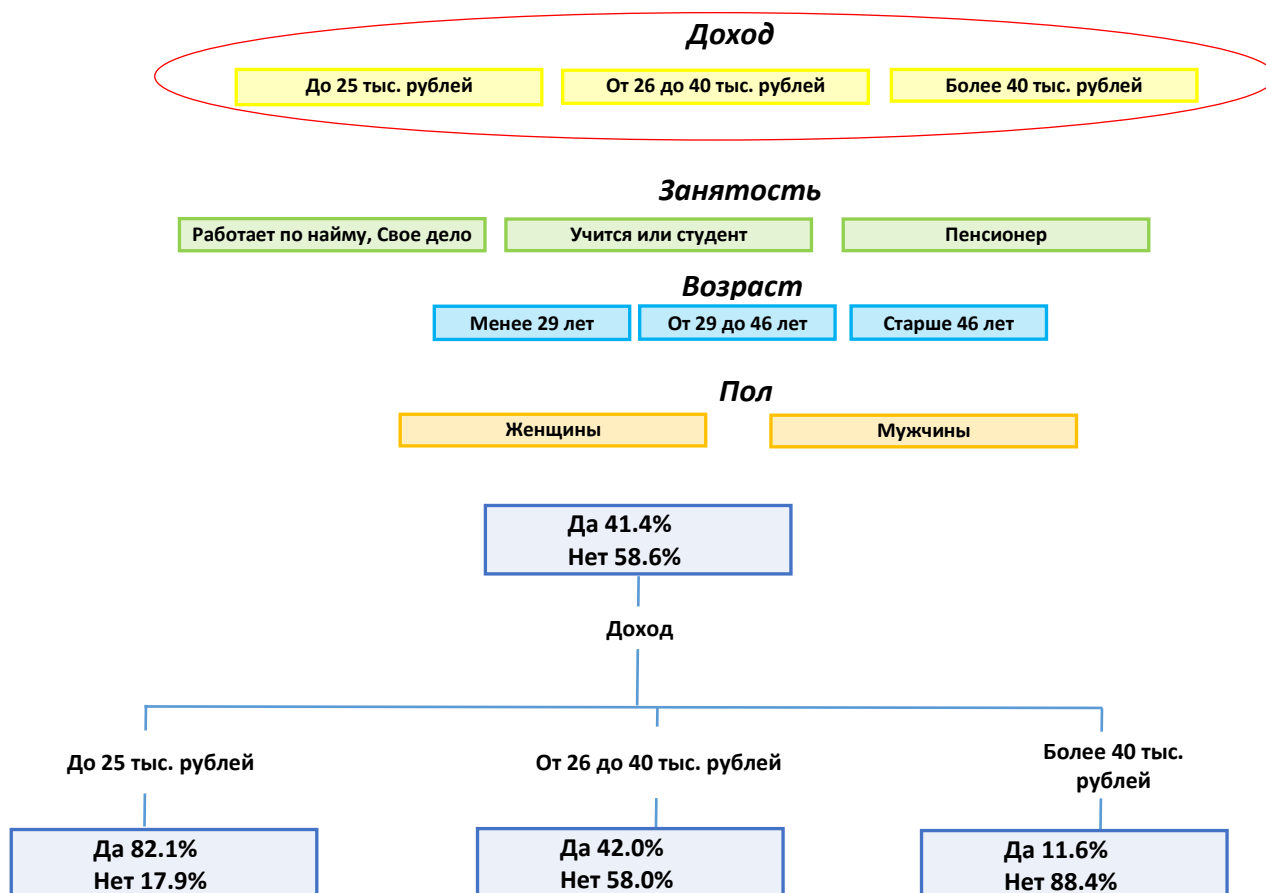


Рисунок 2.3 Выбор предиктора для разбиения узла

Например, лучшим предиктором объявлена переменная *Доход* (имеет наименьшее скорректированное p -значение, не превышающее заданный уровень значимости для разбиения узла). Тогда CHAID обращается к

первой новой группе (например, объединенной категории *Менее 10 тыс. рублей/От 10 до 25 тыс. рублей*) и снова повторяет вышеописанные шаги, ищет наименее различающиеся категории для объединения и выбирает наиболее значимый предиктор для разбиения. Предположим, что для рассматриваемой группы таким предиктором стал *Пол*. Тогда CHAID разделяет группу *Менее 10 тыс. рублей/От 10 до 25 тыс. рублей* по переменной *Пол*. Затем он исследует каждую из оставшихся групп, образованную переменной *Доход* (категории *От 26 до 40 тыс. рублей*, *Более 40 тыс. рублей*), снова по каждой группе проверяет категории предикторов на объединение и разбивает узел с помощью предиктора, который наиболее значимо связан с зависимой переменной для этой группы. Затем CHAID опускается на следующий уровень дерева и берет первую группу предиктора *Пол Женщины* внутри группы *Менее 10 тыс. рублей/От 10 до 25 тыс. рублей*, снова исследует категории и выясняет, есть ли среди предикторов значимо влияющие на зависимую переменную. Если таких предикторов для группы *Женщины* внутри группы *Менее 10 тыс. рублей/От 10 до 25 тыс. рублей* не оказывается или выполняется условие останова, то CHAID объявляет эту группу терминальным узлом и переходит к аналогичному исследованию группы *Мужчины* внутри группы *Менее 10 тыс. рублей/От 10 до 25 тыс. рублей*. Таким способом, уровень за уровнем, CHAID систематически разделяет данные на группы (называемые узлами), показывающие значимые различия по отношению к зависимой переменной. Результаты этого процесса представляются в форме дерева, в котором ветвление происходит по мере деления на группы. Взглянув на построенное дерево (рисунок 2.4), мы можем с уверенностью ответить на ряд вопросов. Какие из предикторов взаимосвязаны с переменной дефолта, помогают предсказать ее? Какие комбинации категорий этих предикторов дают наибольший процент попадания в интересующую категорию зависимой переменной? Они представляют собой целевые группы, на которых нужно сосредоточить внимание. В следующих разделах будет подробно рассказано, как строить и интерпретировать дерево решений CHAID в пакете R CHAID.

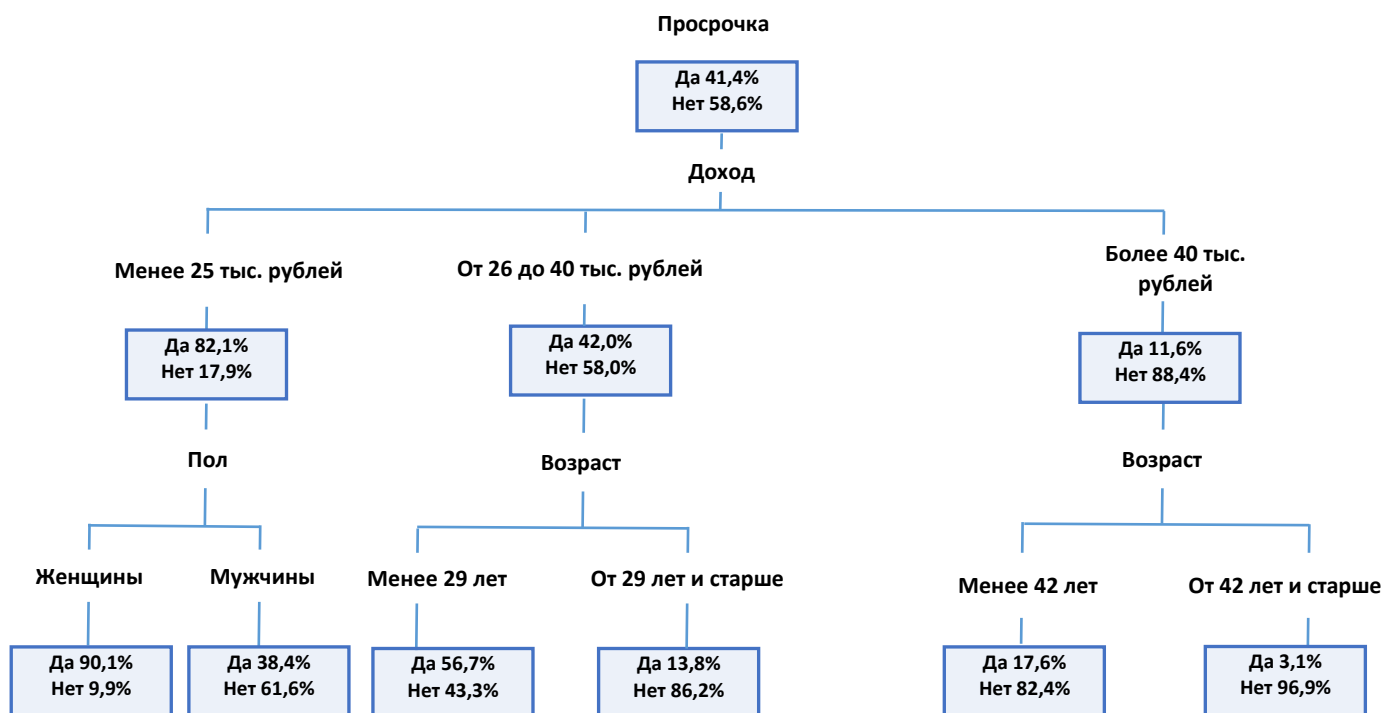


Рисунок 2.4 Итоговое дерево CHAID

Лекция 2.2 Предварительная подготовка данных перед построением модели дерева CHAID

2.2.1 Загрузка данных

Данные, которыми я воспользуюсь для построения дерева классификации CHAID, записаны в файле *Churn.csv*. Исходная выборка содержит записи о 4431 клиенте, классифицированном на два класса: 0 — оттока нет (2496 клиентов) и 1 — отток есть (1935 клиентов). По каждому наблюдению (клиенту) фиксируются следующие переменные (характеристики):

- порядковый предиктор *Длительность междугородних звонков в мин* [*longdist*];
- порядковый предиктор *Длительность местных звонков в мин* [*local*];
- номинальный предиктор *Наличие скидки на междугородние звонки* [*int_disc*];
- номинальный предиктор *Тип местных звонков* [*billtype*];
- номинальный предиктор *Способ оплаты* [*pay*];
- номинальный предиктор *Пол* [*gender*];
- номинальный предиктор *Семейное положение* [*marital*];

- количественный предиктор *Доход [income]*;
- порядковый предиктор *Возрастная категория [agecat]*;
- номинальная зависимая переменная *Наличие оттока [churn]*.

Необходимо разработать модель оттока, с помощью которой предполагается классифицировать новых клиентов на лояльных и склонных к уходу.

Запустим программу R. Для работы нам потребуются следующие пакеты:

- Hmisc;
- car;
- imputeMissings;
- lsr;
- CHAID;
- pROC;
- xlsx;
- stringr.

Давайте установим их с помощью функции `install.packages`. Если вы используете консольную версию R, потребуется еще задать постоянный CRAN-репозиторий, из которого будут устанавливаться пакеты.

```
# задаем постоянный CRAN репозиторий
cat(".Rprofile: Setting US repositoryn")
r = getOption("repos")
r["CRAN"] = "http://cran.us.r-project.org"
options(repos = r)
rm(r)

# устанавливаем пакеты
install.packages("dplyr")
install.packages("Hmisc")
install.packages("car")
install.packages("imputeMissings")
install.packages("lsr")
install.packages("CHAID", repos="http://R-Forge.R-project.org")
install.packages("pROC")
install.packages("xlsx")
install.packages("stringr")
```

Установив пакеты, потом их можно загрузить для работы с помощью функции `library`.

С помощью функции `read.csv2` запишем данные файла *Churn.csv* в объект **data** (его еще называют таблицей данных или датафреймом). Поскольку файл имеет формат CSV, в котором в качестве символа-разделителя используется точка с запятой, укажем значение параметра `sep=";"`.

```
# считываем CSV-файл в датафрейм data
data <- read.csv2("C:/Trees/Churn.csv", sep=";")
```

2.2.2 Фиксация пустых строковых значений как пропусков

Данные часто содержат пропуски. По умолчанию пропуски для векторов типов `integer` и `numeric` будут записаны как значения `NA` (not available – нет в наличии), а пропуски для векторов типа `factor` специально никак не помечаются и представлены как пустые строковые значения (обозначение `NA` не используется, потому что у фактора может быть действительный уровень `NA`). Кроме того, есть еще недопустимые значения – значения `NaN` (не являются числом – not a number), а также бесконечные значения – значения `Inf` и `-Inf`. Давайте взглянем на первые 10 наблюдений.

```
# смотрим первые 10 наблюдений
# датафрейма data
head(data, 10)
```

Сводка 2.1. Первые 10 наблюдений датафрейма data

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
1	<2	<8	Нет	Бюджетный	СС	Мужской	Женат	77680.0	<31	1
2		<8	Нет	Бесплатный	СС	Мужской	Женат	37111.5	<31	1
3	<2	<8	Нет		СС	Мужской	Женат	NA		1
4	<2	<8		Бесплатный	CH	Мужской	Одинокий	NA	<31	0
5	<2		Нет	Бесплатный	Auto		Одинокий	16829.6	<31	1
6		<8	Нет	Бесплатный		Женский	Одинокий	57272.7	<31	1
7		<8	Нет	Бюджетный		Мужской	Женат	NA	<31	1
8	<2	<8	Нет	Бесплатный	Auto	Мужской		NA	<31	1
9	<2	<8	Да		CH	Мужской	Одинокий	92167.3	<31	1
10	<2	<8	Нет	Бесплатный	СС	Мужской	Женат	37135.5	<31	1

Переменные `longdist`, `local`, `int_disc`, `billtype`, `pay`, `gender`, `marital`, `agecat` и `churn` представлены строковыми значениями. При этом переменные `longdist`, `local`, `int_disc`, `billtype`, `pay`, `gender`, `marital` и `agecat` в некоторых наблюдениях имеют пустые строковые значения. Переменная `income`, которая представлена числовыми значениями, в ряде наблюдений имеет значения `NA`. Пустые строковые значения и значения `NA` и есть наши фактические пропуски. Обратите внимание, пока мы не пометим пустые строковые значения как пропуски, программа не будет их учитывать при вычислениях.

Например, выведем количество пропусков по переменной `income`.

```
# выводим количество пропусков
# по переменной income
sum(is.na(data$income))
```

Сводка 2.2. Сумма пропусков по переменной tenure

```
[1] 4
```

Символ доллара `$` указывает на принадлежность переменной `income` к датафрейму `data`, то есть мы указываем программе, что нас интересует не

какая-то абстрактная переменная *income*, а конкретная переменная *income*, принадлежащая датафрейму **data**. Для вычислений сначала используем функцию `is.na`, которая определяет, является ли значение пропущенным или нет (возвращает логическое значение TRUE или логическое значение FALSE соответственно). Затем функция `sum` подсчитывает количество пропусков как сумму логических значений TRUE.

Теперь выведем количество пропусков по переменной *pay*, относительно которой мы можем сказать, что у нее есть пропуски.

```
# выводим количество пропусков
# по переменной pay
sum(is.na(data$pay))
```

Сводка 2.3. Сумма пропусков по переменной pay

```
[1] 0
```

Однако мы точно знаем, что у нее есть пропуски. Теперь пометим пустые строковые значения в датафрейме **data** как пропуски.

```
# помечаем пустые строковые
# значения как пропуски
is.na(data) <- data==''
```

Выведем первые 10 наблюдений.

```
# выводим первые 10 наблюдений
# датафрейма data
head(data, 10)
```

Сводка 2.4. Первые 10 наблюдений датафрейма data после того, как пустые строковые значения были помечены как пропуски

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
1	<2	<8	Нет	Бюджетный	СС	Мужской	Женат	77680.0	<31	1
2	<NA>	<8	Нет	Бесплатный	СС	Мужской	Женат	37111.5	<31	1
3	<2	<8	Нет	<NA>	СС	Мужской	Женат	NA	<NA>	1
4	<2	<8	<NA>	Бесплатный	CH	Мужской	Одинокий	NA	<31	0
5	<2	<NA>	Нет	Бесплатный	Auto	<NA>	Одинокий	16829.6	<31	1
6	<NA>	<8	Нет	Бесплатный	<NA>	Женский	Одинокий	57272.7	<31	1
7	<NA>	<8	Нет	Бюджетный	<NA>	Мужской	Женат	NA	<31	1
8	<2	<8	Нет	Бесплатный	Auto	Мужской	<NA>	NA	<31	1
9	<2	<8	Да	<NA>	CH	Мужской	Одинокий	92167.3	<31	1
10	<2	<8	Нет	Бесплатный	СС	Мужской	Женат	37135.5	<31	1

Пустые строковые значения теперь помечены как <NA>, символы <> используются для того, чтобы пропуски можно было отличить от возможного действительного уровня NA.

Теперь снова выведем количество пропусков по переменной *pay*.

```
# выводим количество пропусков
# по переменной pay
sum(is.na(data$pay))
```

Сводка 2.5. Сумма пропусков по переменной `pay` после того, как пустые строковые значения были помечены как пропуски

```
[1] 2
```

Еще более удобный вариант зафиксировать пустые строковые значения как пропуски – использование при загрузке данных значения параметра `na.strings=""`.

```
# фиксируем пустые строковые
# значения как пропуски
data <- read.csv2("C:/Trees/Churn.csv", sep=";", na.strings="")
```

2.2.3 Изменение типов переменных

Теперь посмотрим, как выглядят наши переменные.

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

Сводка 2.6. Типы переменных в датафрейме `data`

```
'data.frame':    4431 obs. of  10 variables:
 $ longdist: Factor w/ 5 levels "<2","15-20","2-8",...: 1 NA 1 1 1 NA NA 1 1 1 ...
 $ local   : Factor w/ 5 levels "<8","21-35","36-55",...: 1 1 1 1 NA 1 1 1 1 1 ...
 $ int_disc: Factor w/ 2 levels "Да","Нет": 2 2 2 NA 2 2 2 1 2 ...
 $ billtype: Factor w/ 2 levels "Бесплатный","Бюджетный": 2 1 NA 1 1 1 2 1 NA 1 ...
 $ pay      : Factor w/ 4 levels "Auto","CC","CD",...: 2 2 2 4 1 NA NA 1 4 2 ...
 $ gender   : Factor w/ 2 levels "Женский","Мужской": 2 2 2 2 NA 1 2 2 2 2 ...
 $ marital  : Factor w/ 2 levels "Женат","Одинокий": 1 1 1 2 2 2 1 NA 2 1 ...
 $ income   : num  77680 37112 NA NA 16830 ...
 $ agecat   : Factor w/ 5 levels "<31","31-45",...: 1 1 NA 1 1 1 1 1 1 1 ...
 $ churn    : int   1 1 1 0 1 1 1 1 1 1 ...
```

В сводке мы видим, что количественная переменная `income` правильно записана как вектор типа `numeric`, потому что ее значения являются числами с плавающей точкой. Номинальные переменные `int_disc`, `billtype`, `pay`, `gender`, `marital` правильно записаны как векторы типа `factor`, для представления их строковых значений будут использоваться неупорядоченные числовые коды.

Однако есть и проблемы. Номинальная зависимая переменная `churn` неправильно записана как вектор типа `integer`, поскольку она представлена целочисленными значениями вместо строковых, а для представления целочисленных значений R использует вектор типа `integer`. Если мы будем строить модель с такой зависимой переменной, будет возвращена ошибка, так как CHAID работает только с номинальной зависимой переменной, представленной в виде фактора, а в других пакетах R (например, в пакетах `gpart` и `randomForest`) вместо

задачи классификации будет решаться задача регрессии. Кроме того, порядковые переменные *longdist*, *local* и *agecat* определены как векторы типа **factor**, что соответствует номинальным переменным. Если мы будем строить модель, то эти порядковые переменные будут обрабатываться методом CHAID как номинальные. Алгоритм будет сравнивать и объединять *любые* имеющиеся категории предиктора (в случае отсутствия различий по зависимой переменной) вместо того, чтобы сравнивать и объединять только *смежные* категории. С помощью функции **as.factor** присвоим переменной *churn* правильный тип **factor**.

```
# преобразовываем переменную
# churn в вектор типа factor
data$churn <- as.factor(data$churn)
```

С помощью функции **ordered** преобразуем переменные *longdist*, *local* и *agecat* в упорядоченные факторы. Аргумент **labels** задает вектор символьных меток (категорий), выстроенных по порядку.

```
# преобразовываем переменные longdist, local,
# agecat в вектор типа ordered factor
data$longdist <- ordered(data$longdist,
                        levels = c("<2", "2-8", "9-14", "15-20", "21+"))
data$local <- ordered(data$local,
                    levels = c("<8", "8-20", "21-35", "36-55", "56+"))
data$agecat <- ordered(data$agecat,
                    levels = c("<31", "31-45", "46-58", "59-70", "71+"))
```

Для удобства с помощью функции **recode** пакета **dplyr** перекодируем уровни зависимой переменной *churn* 0 и 1 в *Остается* и *Уходит*.

```
# загружаем пакет dplyr
library(dplyr)
# с помощью функции recode пакета dplyr
# переименовываем категории переменной marital_status
data$churn <- recode(data$churn, "0"="Остается", "1"="Уходит")
```

Выполнив переименование уровней, отсоединяем пакет **dplyr**. В среде R необходимо взять за правило отсоединять пакет, если он в дальнейшем не потребуется или будет использована одноименная функция из другого пакета.

```
# отсоединяем пакет dplyr
detach("package:dplyr", unload=TRUE)
```

Теперь снова посмотрим типы переменных в нашем датафрейме.

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

Сводка 2.7. Типы переменных в датафрейме data (преобразованные переменные выделены желтым)

```
'data.frame':      4431 obs. of  10 variables:
 $ longdist: Ord.factor w/ 5 levels "<2"<"2-8"<"9-14"<...: 1 NA 1 1 1 NA NA 1 1 1 ...
 $ local   : Ord.factor w/ 5 levels "<8"<"8-20"<"21-35"<...: 1 1 1 1 NA 1 1 1 1 1 ...
 $ int_disc: Factor w/ 2 levels "Да","Нет": 2 2 2 NA 2 2 2 1 2 ...
 $ billtype: Factor w/ 2 levels "Бесплатный","Бюджетный": 2 1 NA 1 1 1 2 1 NA 1 ...
 $ pay      : Factor w/ 4 levels "Auto","СС","CD",...: 2 2 2 4 1 NA NA 1 4 2 ...
 $ gender   : Factor w/ 2 levels "Женский","Мужской": 2 2 2 2 NA 1 2 2 2 2 ...
 $ marital  : Factor w/ 2 levels "Женат","Одинокий": 1 1 1 2 2 2 1 NA 2 1 ...
 $ income   : num  77680 37112 NA NA 16830 ...
 $ agecat   : Ord.factor w/ 5 levels "<31"<"31-45"<...: 1 1 NA 1 1 1 1 1 1 1 ...
 $ churn    : Factor w/ 2 levels "Остается","Уходит": 2 2 2 1 2 2 2 2 2 2.....
```

Видим, что номинальная переменная *churn* теперь правильно записана как фактор. Порядковые переменные правильно записаны как упорядоченные факторы. Для преобразования типов данных руководствуйтесь следующей таблицей.

Тип преобразования	Функция
в вектор integer	as.integer
в вектор numeric	as.numeric
в вектор factor	as.factor
в вектор ordered factor	ordered
в вектор character	as.character
в вектор logical	as.logical

Рисунок 2.5 Функции R, выполняющие преобразования типов данных

2.2.4 Обработка дублирующихся наблюдений

Мы могли бы уже приступить к предварительной обработке данных, однако очень часто при подготовке выборки допускаются ошибки, в частности, в набор данных может попасть одно и то же наблюдение. Давайте убедимся, что наш набор не содержит дублирующихся наблюдений (строк). Для этого мы воспользуемся функцией **duplicated**, которая выведет нам все дублирующиеся наблюдения.

```
# смотрим дублирующиеся наблюдения
data[duplicated(data),]
```

Сводка 2.8. Список дублирующихся наблюдений датафрейма data

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
358	<2	<8	Нет	Бюджетный	СС	Женский*	_Женат	32118.4	71+	Уходит
461	<2	<8	Да	Бесплатный	СС	Женский*	_Одинокий	18831.1	46-58	Уходит
2060	21+	21-35	Нет	Бюджетный	СС	Мужской*	_Женат	84269.0	31-45	Остается
3161	9-14	56+	Нет	Бюджетный	СС	Мужской*	_Одинокий	54643.3	31-45	Остается
3834	2-8	21-35	Нет	Бесплатный	Auto	Женский	Же&нат	92353.3	46-58	Остается
4382	15-20	21-35	Нет	Бесплатный	СС	Женский*	_Женат	87404.6	46-58	Уходит
4431	21+	8-20	Нет	Бесплатный	СС	Женский*	_Одинокий	75639.8	31-45	Остается

В сводке 2.8 приводится список дублирующихся наблюдений. При этом обратите внимание, что для всех дублирующихся наблюдений характерно присутствие лишних символов в значениях переменных *gender* и *marital*.

Давайте убедимся в том, что первая строка, приведенная в сводке, действительно дублируется в нашем наборе. Выполняем отбор наблюдений по условиям – значениям переменных в первой строке.

```
# проверим, дублируется ли наблюдение
# по заданному набору условий
data[data$longdist == "<2" & data$local == "<8" & data$int_disc == "Нет" &
      data$billtype == "Бюджетный" & data$pay == "CC" & data$gender == "Женский&*" &
      data$marital == "_Женат" & data$income == 32118.4 & data$agecat == "71+" &
      data$churn == "Уходит", ]
```

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

Для отбора наблюдений по заданным условиям также можно было воспользоваться следующим программным кодом:

```
data[which(data[, 'longdist'] == "<2" & data[, 'local'] == "<8" &
           data[, 'int_disc'] == "Нет" & data[, 'billtype'] == "Бюджетный" &
           data[, 'pay'] == "CC" & data[, 'gender'] == "Женский&*" &
           data[, 'marital'] == "_Женат" & data[, 'income'] == 32118.4 &
           data[, 'agecat'] == "71+" &
           data[, 'churn'] == "Уходит"), ]
```

Сводка 2.9. Конкретное дублирующееся наблюдение

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
348	<2	<8	Нет	Бюджетный	CC	Женский&*	_Женат	32118.4	71+	Уходит
358	<2	<8	Нет	Бюджетный	CC	Женский&*	_Женат	32118.4	71+	Уходит

Из сводки видно, что действительно первая строка, приведенная в списке дублей, повторяется в нашем наборе данных два раза. Все значения предикторов совпадают, включая совпадение значений дохода с точностью до первого десятичного знака и совпадение лишних символов в строковых значениях переменных *gender* и *marital*.

Теперь с помощью функции `unique` оставим в нашем наборе только уникальные, недублирующиеся наблюдения.

```
# оставим только уникальные наблюдения
data <- unique(data)
# посмотрим, сколько наблюдений мы
# теперь будем использовать
nrow(data)
```

Сводка 2.10. Количество уникальных наблюдений датафрейма data

```
[1] 4424
```

В итоге мы будем работать с 4424 наблюдениями и можно приступать к обработке данных.

2.2.5 Вывод подробной информации о переменных

Для проведения предварительной обработки данных необходимо получить более развернутую информацию о переменных. В таких случаях можно воспользоваться функцией `describe` из пакета `Hmisc`, разработанным Фрэнком Харреллом, профессором биостатистики Вандербильтского университета, экспертом Центра оценки и исследований биопрепаратов Управления по санитарному надзору за качеством пищевых продуктов и медикаментов (CDER FDA).

```
# загружаем пакет Hmisc
library(Hmisc)
# с помощью функции describe пакета
# Hmisc выведем подробную информацию
# о переменных
describe(data)
```

Сводка 2.11. Статистики переменных для датафрейма `data`, вычисленные с помощью функции `describe` пакета `Hmisc` (фрагмент)

`data`

10 Variables 4424 Observations

`longdist`

n	missing	distinct
4421	3	5

Value	<2	2-8	9-14	15-20	21+
Frequency	736	867	993	1054	771
Proportion	0.166	0.196	0.225	0.238	0.174

`local`

n	missing	distinct
4423	1	5

Value	<8	8-20	21-35	36-55	56+
Frequency	738	763	1444	738	740
Proportion	0.167	0.173	0.326	0.167	0.167

`int_disc`

n	missing	distinct
4423	1	2

Value	Да	Нет
Frequency	1376	3047
Proportion	0.311	0.689

`billtype`

n	missing	distinct
4422	2	2

Value	Бесплатный	Бюджетный
Frequency	2181	2241
Proportion	0.493	0.507

`pay`

n	missing	distinct
4422	2	4

Value	Auto	CC	CD	CH
Frequency	889	2554	2	977
Proportion	0.201	0.578	0.000	0.221

В полученном отчете первой приводится информация о количестве переменных и общем количестве наблюдений. Для каждой категориальной переменной приводятся n – количество непропущенных наблюдений, **missing** – количество пропущенных значений, **distinct** – количество уникальных значений. Для каждой количественной переменной приводятся n – количество непропущенных наблюдений, **missing** – количество пропущенных значений, **distinct** – количество уникальных значений, **Info** – мера информативности, **Mean** – среднее значение, **Gmd** – средняя разность Джини, **.05** – 5%-ный квантиль, **.10** – 10%-ный квантиль, **.25** – 25%-ный квантиль, **.50** – 50%-ный квантиль, **.75** – 75%-ный квантиль, **.90** – 90%-ный квантиль, **.95** – 95%-ный квантиль. Давайте подробнее рассмотрим некоторые показатели.

Среднее значение, соответствующее показателю **Mean**, вычисляется как сумма всех значений, поделенная на их количество. Среднее значение еще можно вычислить с помощью базовой функции **mean**.

Показатель **Info** вычисляется как единица минус сумма кубов относительных частот уникальных значений, деленная на единицу минус квадрат обратной величины размера выборки.

$$Info = \frac{1 - \sum_{i=1}^k f_i^3}{1 - (n^{-1})^2}$$

где

f_i – частота i -той категории переменной;

n^{-1} – обратная величина размера выборки (единица, деленная на общее количество наблюдений).

Наименьшие значения информативности получают переменные, имеющие одно уникальное значение, а также бинарные переменные с сильно отличающимися относительными частотами.

Квантиль – это такое значение переменной, которое делит ее диапазон (упорядоченный ряд чисел) на две части с известными пропорциями в каждой из частей. Самым известным квантилем является 0.5-квантиль (говорят *квантиль уровня 0.5*) или медиана – значение, которое делит упорядоченный числовой ряд пополам, то есть ровно половина остальных значений больше него, а другая половина меньше его. 0.25-квантиль – это значение, ниже которого будет лежать 25% значений числового ряда, а выше – 75% значений числового ряда. Среди всех возможных квантилей обычно выделяют определенные семейства. Квантили одного семейства делят диапазон изменения признака на заданное число равнонаполненных частей. Семейство определяется тем, сколько частей получается. Наиболее популярными квантилями являются квартили, разбивающие диапазон изменения признака на 4

равнонаполненные части; децили – на 10 равнонаполненных частей; процентиля – на 100 частей.

Допустим, у нас есть числовой ряд

18 20 23 20 23 27 24 23 29

Давайте его упорядочим.

18 20 20 23 23 23 24 27 29

Теперь вычислим 0.5-квантиль или медиану. У нас числовой ряд состоит из 9 значений, поэтому $n=9$.

$$\begin{aligned}\text{Медиана} &= \left(\frac{n+1}{2} \right) - \text{ное значение} = \left(\frac{9+1}{2} \right) - \text{ное значение} = \\ &= \left(\frac{10}{2} \right) - \text{ное значение} = 5 - \text{ое значение}\end{aligned}$$

Теперь давайте автоматически вычислим медиану с помощью базовой функции `median`.

Создаем вектор `series`, представляющий собой вышеприведенный числовой ряд. При этом не будем упорядочивать значения, поскольку под капотом базовая функция `median` самостоятельно сделает это.

```
# создаем числовой ряд
series <- c(18, 20, 23, 20, 23, 27, 24, 23, 29)
# вычисляем медиану
median(series)
```

Сводка 2.12. Медиана или 0.5-квантиль, вычисленный с помощью базовой функции `median`

```
[1] 23
```

Кроме того, медиану можно автоматически вычислить с помощью базовой функции `quantile`.

```
# вычисляем медиану
quantile(series, 0.50)
```

Сводка 2.13. Медиана или 0.5-квантиль, вычисленный с помощью базовой функции `quantile`

```
50%
23
```

Теперь вычислим 0.25-квантиль или первый (нижний) квартиль.

$$\begin{aligned}
 0.25\text{-квантиль} &= \frac{1}{4}(n+1)\text{-ное значение} = \frac{1}{4}(9+1)\text{-ное значение} = \\
 &= \left(\frac{10}{4}\right)\text{-ное значение} = 2.5\text{-ое значение} = \frac{20+20}{2} \quad \leftarrow \text{Вычисляем среднее} \\
 &= \frac{40}{2} = 20 \quad \text{2-го и 3-го значений}
 \end{aligned}$$

Давайте автоматически вычислим 0.25-квантиль с помощью базовой функции `quantile`.

```
# вычисляем 0.25-квантиль
quantile(series, 0.25)
```

Сводка 2.14. 0.25-квантиль, вычисленный с помощью базовой функции `quantile`

```
25%
20
```

2.2.6 Нормализация строковых значений

Обратимся к переменным *gender* и *marital* в отчете, сгенерированном функцией `describe` пакета `Hmisc`. Видно, что значения этих переменных содержат лишние символы.

Сводка 2.15. Распределения значений переменных *gender* и *marital* (фрагмент отчета, сгенерированного функцией `describe` пакета `Hmisc`)

```
gender
  n missing distinct
4423      1         4

Value      Женский Женский&* Мужской Мужской&*
Frequency    2235         4    2182         2
Proportion    0.505    0.001    0.493    0.000
-----
marital
  n missing distinct
4423      1         5

Value      _Женат _Одинокий Же&нат Женат Одинокий
Frequency         3         3         1    2620    1796
Proportion    0.001    0.001    0.000    0.592    0.406
-----
```

Кроме того, распределения значений переменных можно вывести, воспользовавшись функциями `lapply` и `summary`. Функция `lapply` представляет набор данных как список или вектор и применяет заданную нами функцию (в данном случае функцию `summary`, вычисляющую частоты значений) к элементам вектора или списка, возвращая результат в виде списка.

```
# создаем вектор названий переменных
names <- c("gender", "marital")
# с помощью функции lapply применяем
# функцию summary к переменным, названия
# которых записаны в векторе names
lapply(data[,names], function(x) summary(x))
```

Сводка 2.16. Распределения значений переменных gender и marital, вычисленные с помощью функций lapply и summary

```
$gender
  Женский Женский&* Мужской Мужской&* NA's
    2235         4    2182         2     1

$marital
  _Женат _Одинокий Же&нат Женат Одинокий NA's
      3         3      1    2620    1796     1
```

Если нужно вывести только уровни переменных, можно воспользоваться функциями lapply и levels.

```
# с помощью функции lapply применяем
# функцию levels к переменным, названия
# которых записаны в векторе names
lapply(data[,names], function(x) levels(x))
```

Сводка 2.17. Уровни переменных gender и marital, вычисленные с помощью функций lapply и levels

```
$gender
[1] "Женский" "Женский&*" "Мужской" "Мужской&*"

$marital
[1] "_Женат" "_Одинокий" "Же&нат" "Женат" "Одинокий"
```

Давайте удалим ненужные символы &*, которыми заканчиваются несколько значений переменной gender, с помощью функции gsub. Первый аргумент функции – символ, который нужно удалить, второй аргумент – символ, на который нужно заменить, третий аргумент – строка или символьный вектор. Обратите внимание, что переменные-факторы нужно преобразовать в символьные векторы. Внешне факторы выглядят (и часто ведут себя) как символьные векторы, но на самом деле под капотом они используют целочисленные значения (коды), связанные с определенными строковыми значениями, поэтому мы не можем напрямую обработать их как строки.

```
# преобразовываем переменную gender из вектора типа factor
# в вектор типа character (нельзя напрямую изменить строковое
# значение фактора, потому что в факторе под капотом для строковых
# значений используются целочисленные коды, поэтому переводим в
# вектор типа character, у которого значения - строки)
data$gender <- as.character(data$gender)

# удаляем с помощью функции gsub символы & и *,
# первый аргумент - удаляемые символы, второй
# аргумент - символы, на которые нужно заменить,
# третий аргумент - вектор типа character или строка
data$gender <- gsub('&\\&\\*', "", data$gender)
```

Удалив лишние символы, преобразуем символьный вектор обратно в фактор.

```
# преобразовываем переменную gender обратно из вектора
# типа character в вектор типа factor
data$gender <- as.factor(data$gender)
```

Удалить лишние символы в значениях переменной `marital` нам поможет функция `str_replace_all` пакета `stringr`. Первый аргумент – строка или символьный вектор, второй аргумент – символ, который нужно удалить, третий аргумент – символ, на который нужно заменить. В данном случае для удаления лишних символов мы воспользуемся символьными классами POSIX. На рисунке 2.6 приводятся наиболее часто используемые символьные классы POSIX.

POSIX-класс	Эквивалент	Значение
<code>[:upper:]</code>	<code>[A-Z]</code>	Символы верхнего регистра
<code>[:lower:]</code>	<code>[a-z]</code>	Символы нижнего регистра
<code>[:alpha:]</code>	<code>[:upper:][:lower:]</code>	Буквы
<code>[:digit:]</code>	<code>[0-9]</code> , т. е. <code>\d</code>	Цифры
<code>[:alnum:]</code>	<code>[:alpha:][:digit:]</code>	Буквы и цифры
<code>[:punct:]</code>	<code>[-!"#\$%&'()*+,-./:;<=>?@[_`{ }~]</code>	Знаки пунктуации

Рисунок 2.6 Наиболее распространенные классы POSIX

Нам понадобится символьный класс `[:alnum:]`, который находит символы, являющиеся буквами и цифрами. Обратите внимание, что использование класса возможно лишь внутри квадратных скобок. Конструкция `^[[:alnum:]]` обозначает «найти все символы, не являющиеся буквами и цифрами». С помощью второго аргумента функции `str_replace_all` мы находим такие символы и удаляем.

```
# загружаем пакет stringr
library(stringr)

# преобразовываем переменную marital из вектора типа factor
# в вектор типа character
data$marital <- as.character(data$marital)

# удаляем с помощью функции str_replace_all пакета stringr
# все символы, не являющиеся буквами алфавита и числами
# первый аргумент – вектор типа character или строка,
# второй аргумент – символы, которые нужно удалить,
# третий аргумент – символы, на которые нужно заменить
data$marital <- str_replace_all(data$marital, "^[[:alnum:]]", "")

# преобразовываем переменную marital обратно из вектора
# типа character в вектор типа factor
data$marital <- as.factor(data$marital)
```

Снова выведем уровни переменных.

```
# снова выводим информацию об уровнях переменной
lapply(data[,names], function(x) levels(x))
```

Сводка 2.18. Уровни переменных `gender` и `marital` после обработки с помощью функций `gsub` и `str_replace_all`

```
$gender
[1] "Женский" "Мужской"

$marital
[1] "Женат"      "Одинокий"
```

Видим, что лишние символы удалены и можно продолжать работу.

2.2.7 Обработка редких категорий

Теперь обратимся к переменной `rau` в отчете, сгенерированном функцией `describe` пакета `Hmisc`. Мы видим, что категория `CD` встречается лишь 2 раза. Распределение значений этой переменной еще можно вывести с помощью функции `summary`.

```
# выводим информацию о распределении
# значений переменной rau
summary(data$pay)
```

Сводка 2.19. Распределение значений переменной `rau`

```
Auto    CC    CD    CH NA's
889 2554    2  977    2
```

Такие редкие категории, как категория `CD`, являются источником шума в данных, который негативно повлияет на качество модели. Кроме того, когда мы разобьем наш набор данных на обучающую и контрольную выборки, может оказаться, что данная категория отсутствует в обучающей выборке, но присутствует в контрольной выборке. Это вызовет проблемы при моделировании.

Регрессионная модель, встретив в новых данных наблюдение с неизвестной категорией предиктора, не сможет вычислить прогноз, потому что необходимый для прогноза регрессионный коэффициент по этой категории предиктора будет отсутствовать. Допустим у нас есть новые данные и одно из наблюдений имеет по переменной `rau` категорию `CD`, которой не было в обучающей выборке. Тогда функция `predict`, применяющая модель логистической регрессии к новым данным, вернет ошибку «у фактора `rau` есть новые уровни `CD`». Модель дерева проверит соответствие условиям, записанным в правилах классификации, и попытается выдать прогноз. Если соответствие не будет найдено, функция `predict` пакета `CHAID` или пакета `rpart` вернет ошибку «у фактора `rau` есть новые уровни `CD`». В случайном лесе функция `predict`

пакета `randomForest` вернет ошибку «new factor levels not present in the training data». Поэтому обработка редких категорий должна осуществляться на самых ранних этапах предварительной обработки.

В пакетах `ganger` и `h2o`, специально разработанных для работы с большими и высокоразмерными данными, редкие категории не представляют проблемы. В пакете `ganger` если категория является «новой» для модели, она просто записывается в соответствующий левый узел. В пакете `h2o` такая категория помечается как пропущенное значение и к ней применяются правила обработки пропусков.

Обработка редких категорий выполняется либо до разбиения на обучающую и контрольную выборки, либо после него в зависимости от причин, обусловивших появление таких категорий. Если переменная содержит 2-3 редких категорий небольшой частоты, скорее всего такие категории случайны и могли быть обусловлены ошибками ввода. В нашем примере у переменной `pay` есть категория `CC`, встречающаяся 2554 раза и категория `CD`, которая встречается 2 раза, поэтому скорее всего речь идет об ошибке ввода. В таком случае эти категории, как правило, объединяют с самой часто встречающейся категорией и это можно сделать как до разбиения на обучение и контроль, так и после него. Если переменная содержит множество категорий небольшой частоты, нам необходимо задать порог укрупнения – минимальное количество наблюдений в категории, ниже которого категория объявляется редкой. Поэтому для объективности решение о выборе такого порога должно приниматься уже после разбиения на обучающую и контрольную выборки. В противном случае вновь получится, что решение о выборе порога мы принимали с учетом информации «из будущего». Такие редкие категории часто объединяют в одну отдельную категорию, либо объединяют с уже существующими категориями по результатам, полученным с помощью метода CHAID.

Итак, объединим редкую категорию `CD` с самой часто встречающейся категорией `CC`. Для этого воспользуемся функцией `recode` пакета `car` и затем с помощью `summary` выведем информацию о распределении значений переменной `pay`.

```
# загружаем пакет car
library(car)
# с помощью функции recode пакета car
# объединяем редкую категорию CD с
# самой часто встречающейся
# категорией CC
data$pay <- recode(data$pay, "'CD'='CC'")
# снова выводим информацию о распределении
# значений переменной pay
summary(data$pay)
```

Сводка 2.20. Распределение значений переменной `pay` после укрупнения

```
Auto   CC   CH NA's
889 2556  977    2
```

Видим, что теперь категория *CD* отсутствует, она объединена с категорией *CC*.

Укрупнение редких категорий часто позволяет немного повысить качество модели. В ряде случаев более оптимальной стратегией может оказаться не объединение редко встречающихся категорий с наиболее часто встречающейся категорией, а выделение редких категорий в отдельную группу.

Завершаем работу с пакетом *Hmisc*, отсоединив его.

```
# отсоединяем пакет Hmisc  
detach("package:Hmisc", unload=TRUE)
```

2.2.8 Однократное случайное разбиение набора данных на обучающую и контрольную выборки для проверки модели

В прогножном моделировании нам важно построить модель на обучающих данных, а затем получить точные прогнозы для новых, еще не встречавшихся данных, состоящих из тех же самых предикторов, что и использованный нами обучающий набор. Если модель может выдавать точные прогнозы на ранее не встречавшихся данных, можно сказать, что модель обладает способностью *обобщать* результат на новые данные. Нам требуется построить модель с максимальной *обобщающей способностью* (*generalization*).

Обычно задача моделиера сводится к тому, чтобы модель давала точные прогнозы на обучающем наборе. Если обучающий набор и новые данные имеют много общего между собой, можно ожидать, что модель будет точно прогнозировать новые данные. Однако в ряде случаев на новых данных модель работает существенно хуже. Почему так происходит?

Проблема заключается в том, что на этапе подготовки данных часто отсутствует априорная информация о полезности тех или иных предикторов и избыточное включение предикторов, не несущих новой информации, ведет к тому, что модель становится слишком сложной. Она слишком точно подстраивается под особенности обучающего набора, улавливает не только фактические взаимосвязи, но и случайные возмущения обучающих данных. По сути такая модель восстанавливает не только искомую зависимость, но и выполняет подгонку конкретных наблюдений, фактически осуществляет аппроксимацию погрешностей собственных измерений. В итоге мы получаем модель, которая идеально работает на обучающем наборе, но плохо обобщает результат на новые данные. Такую ситуацию называют *переобучением* (*overfitting*). С другой стороны, включение недостаточного числа полезных признаков, наоборот, приводит к тому, что модель не может в достаточной мере уловить фактические зависимости и качество модели даже на обучающей

выборке остается довольно низким. Такую ситуацию называют *недообучением (underfitting)*.

Чтобы исключить подобные ситуации, необходимо проверить работу модели на контрольной выборке, то есть посмотреть, как модель будет предсказывать новые наблюдения, не участвовавшие в обучении. Обычно наиболее распространенными методами проверки являются однократное случайное разбиение набора данных на обучающую и контрольную выборки, многократное случайное разбиение набора данных на обучающую и контрольную выборки, перекрестная проверка, однократное случайное разбиение набора данных на обучающую, контрольную и тестовую выборки.

Мы начнем с однократного случайного разбиения на обучающую и контрольную выборки. При таком способе проверки модель строится на обучающей выборке, а ее качество проверяется на контрольной выборке (рисунок 2.7). Правила классификации, сформулированные на основе построенного дерева, затем применяются к данным, предназначенным для проверки модели. Обратите внимание, данный вид проверки с осторожностью должен применяться на малых выборках (менее 1000 наблюдений). Небольшие размеры обучающей выборки могут привести к построению неадекватной модели, т.к. может не оказаться достаточного количества наблюдений в той или иной категории, чтобы корректно построить дерево.



Рисунок 2.7 Схема работы метода *Разделение выборки на обучающую и контрольную*

Давайте случайным образом разобьем наш датафрейм `data` на обучающий датафрейм `development` и контрольный датафрейм `holdout`. При этом 70% наблюдений исходного набора попадут в обучающую выборку, а 30% наблюдений – в контрольную выборку. Для получения воспроизводимых результатов разбиения задаем стартовое значение генератора случайных чисел.

```
# разбиваем набор данных на случайную
# и контрольную выборки
set.seed(42)
```



```
data$random_number <- runif(nrow(data),0,1)
development <- data[ which(data$random_number > 0.3), ]
holdout <- data[ which(data$random_number <= 0.3), ]

data$random_number <- NULL
development$random_number <- NULL
holdout$random_number <- NULL
```

Давайте взглянем на получившиеся датафреймы.

```
# смотрим обучающий датафрейм
str(development)
```

Сводка 2.21. Обучающий датафрейм development

```
'data.frame':    3080 obs. of  10 variables:
 $ longdist: Ord.factor w/ 5 levels "<2"<"2-8"<"9-14"<...: 1 NA 1 1 NA NA 1 1 1 1 ...
 $ local   : Ord.factor w/ 5 levels "<8"<"8-20"<"21-35"<...: 1 1 1 NA 1 1 1 1 1 1 ...
 $ int_disc: Factor w/ 2 levels "Да","Нет": 2 2 NA 2 2 2 1 2 2 2 ...
 $ billtype: Factor w/ 2 levels "Бесплатный","Бюджетный": 2 1 1 1 1 2 NA 1 1 2 ...
 $ pay      : Factor w/ 3 levels "Auto","CC","CH": 2 2 3 1 NA NA 3 2 3 2 ...
 $ gender   : Factor w/ 2 levels "Женский","Мужской": 2 2 2 NA 1 2 2 2 2 1 ...
 $ marital  : Factor w/ 2 levels "Женат","Одинокий": 1 1 2 2 2 1 2 1 2 1 ...
 $ income   : num  77680 37112 NA 16830 57273 ...
 $ agecat   : Ord.factor w/ 5 levels "<31"<"31-45"<...: 1 1 1 1 1 1 1 1 1 1 ...
 $ churn    : Factor w/ 2 levels "Остается","Уходит": 2 2 1 2 2 2 2 2 1 2 ...
```

```
# смотрим контрольный датафрейм
str(holdout)
```

Сводка 2.22. Контрольный датафрейм holdout

```
'data.frame':    1344 obs. of  10 variables:
 $ longdist: Ord.factor w/ 5 levels "<2"<"2-8"<"9-14"<...: 1 1 1 1 1 1 1 1 1 1 ...
 $ local   : Ord.factor w/ 5 levels "<8"<"8-20"<"21-35"<...: 1 1 1 1 1 1 1 1 1 1 ...
 $ int_disc: Factor w/ 2 levels "Да","Нет": 2 2 2 2 2 2 2 2 2 2 ...
 $ billtype: Factor w/ 2 levels "Бесплатный","Бюджетный": NA 1 1 1 1 1 1 1 2 1 ...
 $ pay      : Factor w/ 3 levels "Auto","CC","CH": 2 1 2 1 3 1 2 1 2 1 ...
 $ gender   : Factor w/ 2 levels "Женский","Мужской": 2 2 2 1 1 2 1 2 2 2 ...
 $ marital  : Factor w/ 2 levels "Женат","Одинокий": 1 NA 2 2 1 2 2 2 1 2 ...
 $ income   : num  NA NA 24458 50969 25949 ...
 $ agecat   : Ord.factor w/ 5 levels "<31"<"31-45"<...: NA 1 1 1 1 1 2 3 3 3 ...
 $ churn    : Factor w/ 2 levels "Остается","Уходит": 2 2 1 2 2 2 2 1 2 1 ...
```

Мы могли бы уже начать строить модель, а потом проверять ее, но тут нужно ответить на вопрос, а достаточен ли объем обучающей/контрольной выборки для построения/проверки модели?

В практике банковского скоринга для ответа на поставленный вопрос часто используют правило «Number of Events Per Variable» (количество событий на одну переменную), сформулированное Фрэнком Харреллом. Для задачи классификации оно связывает минимальный объем выборки с количеством «событий» – наблюдений в миноритарной (наименьшей по размеру) категории зависимой переменной и количеством предикторов, поданным на вход модели. Согласно этому правилу,

необходимо взять количество наблюдений в обучающей выборке, относящихся к миноритарной категории зависимой переменной (в кредитном скоринге это «плохие» заемщики). Это число наблюдений нужно разделить на количество заданных предикторов. Для регрессионной модели на один предиктор должно приходиться не менее 20 событий, при построении дерева решений CHAID² на один предиктор должно приходиться не менее 50 событий, а для модели случайного леса, градиентного бустинга, SVM и нейронной сети на одну независимую переменную должно приходиться не менее 200 событий. Для задачи регрессии мы просто берем количество наблюдений и делим на количество предикторов и для регрессионной модели на одну независимую переменную должно приходиться не менее 20 наблюдений, для дерева решений CHAID (в тех случаях, когда реализация алгоритма позволяет решать задачу регрессии) на один предиктор должно приходиться не менее 50 наблюдений, для случайного леса и других сложных моделей на один предиктор должно приходиться не менее 200 наблюдений.

Если это правило выполняется, то объем выборки для обучения является достаточным. В противном случае необходимо либо увеличить объем выборки, либо сократить количество предикторов, подаваемых на вход модели. Затем вся та же самая процедура применяется к контрольной выборке, если правило выполняется, объем выборки для проверки достаточен.

Мы решаем задачу классификации и у нас есть общая выборка объемом 4424 клиента, классифицированных на два класса: класс *Остается* (2492 клиента) и класс *Уходит* (1932 клиента). Мы разбили выборку на обучающую и контрольную и получили следующее распределение классов в выборках.

```
# выводим информацию о распределении
# классов зависимой переменной churn
# в обучающей выборке
summary(development$churn)
```

Сводка 2.23. Распределение классов в обучающей выборке

Остается	Уходит
1746	1334

```
# выводим информацию о распределении
# классов зависимой переменной churn
# в контрольной выборке
summary(holdout$churn)
```

² По мнению Фрэнка Харрела для дерева решений CART правило NEPV невозможно сформулировать из-за высокой нестабильности метода и склонности к переобучению.

Сводка 2.24. Распределение классов в контрольной выборке

Остается	Уходит
746	598

Выясняем, достаточен ли объем выборки для обучения. У нас 1746 оставшихся клиентов, 1334 ушедших клиента и 9 независимых переменных. Миноритарный класс – класс *Уходит*. Проверая выполнение правила 50EPV, мы получаем $1334/9 = 148,2$. Наша выборка обеспечивает достаточное количество событий на одну переменную и мы можем использовать эту выборку для обучения.

Выясняем, достаточен ли объем выборки для проверки. У нас 746 оставшихся клиентов, 598 ушедших клиентов. Проверая выполнение правила 50EPV, мы получаем $598/9 = 66,4$. Наша выборка обеспечивает достаточное количество событий на одну переменную и мы можем использовать эту выборку для контроля.

2.2.9 Импутация пропусков

Выполнив разбиение набора данных на обучающую и контрольную выборки, можно приступить к импутации пропусков.

Давайте выведем сводку о количестве пропусков по каждой переменной в обучающей и контрольной выборках, воспользовавшись функцией `sapply`. Функция `sapply` представляет набор данных как список или вектор и применяет заданную нами функцию к каждому элементу списка или вектора, возвращая результат в виде вектора.

```
# выведем информацию о пропусках
# в обучающей выборке
sapply(development, function(x) sum(is.na(x)))
```

Сводка 2.25. Отчет по пропущенным значениям в датафрейме `development`, возвращенный функцией `sapply`

longdist	local	int_disc	billtype	pay	gender	marital	income	agecat
3	1	1	1	2	1	0	2	0
churn								
0								

```
# выведем информацию о пропусках
# в контрольной выборке
sapply(holdout, function(x) sum(is.na(x)))
```

Сводка 2.26. Отчет по пропущенным значениям в датафрейме `holdout`, возвращенный функцией `sapply`

```
longdist    local int_disc billtype    pay    gender    marital    income    agecat
0           0           0         1      0         0         1         2         1
churn
0
```

Как вариант, сводку о количестве пропусков можно получить с помощью функции `lapply`. Теперь результат будет возвращен в виде списка.

```
# выведем информацию о пропусках в обучающей выборке
# с помощью lapply, теперь результат будет
# возвращен в виде списка
lapply(development, function(x) sum(is.na(x)))
```

Сводка 2.27. Отчет по пропущенным значениям в датафрейме `development`, возвращенный функцией `lapply` (фрагмент)

```
$longdist
[1] 3
```

```
$local
[1] 1
```

```
# выведем информацию о пропусках в контрольной выборке
# с помощью lapply, теперь результат будет
# возвращен в виде списка
lapply(holdout, function(x) sum(is.na(x)))
```

Сводка 2.28. Отчет по пропущенным значениям в датафрейме `holdout`, возвращенный функцией `lapply` (фрагмент)

```
$longdist
[1] 0
```

```
$local
[1] 0
```

Видно, что пропуски есть не только в количественных, но и в категориальных переменных. И здесь возникает вопрос, как осуществлять импутацию пропущенных значений. Кроме того, читатель может спросить, почему мы приступаем к импутации пропусков после разбиения на обучающую и контрольную выборки, почему нельзя было обработать пропуски сразу на всем наборе?

Чаще всего пропуски переменных заменяются значениями вычисленных статистик: для количественных переменных используется среднее и медиана, для категориальных переменных – мода (наиболее часто встречающаяся категория).

Импутацию средним, медианой и модой необходимо выполнять после разбиения набора данных на обучающую и контрольную выборки. Если выполнить импутацию на всем наборе, а потом разбить его на

обучающую и контрольную выборки, получится, что для вычисления статистик, с помощью которых мы импутировали пропуски, мы использовали все наблюдения набора, часть из которых потом у нас вошла в контрольную выборку (по сути выборку новых данных). Однако мы должны смоделировать наиболее близкую к реальности ситуацию, у нас есть обучающая выборка, никаких новых данных еще нет, поэтому получается, что статистики для импутации, которые мы получили на всем наборе, пришли к нам частично из «будущего» (из новой, контрольной выборки, которой по факту еще нет), что неверно. Статистики, вычисленные на обучающей выборке, можно применить для импутации пропусков как в обучающей, так и в контрольной выборках, а можно отдельно вычислить статистики для обучающей выборки, отдельно вычислить статистики для контрольной выборки и выполнить импутацию. При этом первый способ импутации более предпочтителен. Помимо импутации средним значением, медианой и модой пропуски можно заменить значениями-константами (например, часто используется значение -999), а пропуски в категориальных переменных часто кодируют отдельной категорией для пропусков. Импутацию константами и отдельной категорией можно выполнять как до, так и после разбиения на обучение и контроль, потому что в рамках этой операции мы не делаем вычислений, охватывающих все наблюдения исходного набора.

Часто практикуется создание индикаторов пропусков, когда рядом с переменной, у которой пропущены значения, создается специальная переменная, принимающая значение 1, если значение исходной переменной пропущено, и 0, если значение не пропущено. Их можно также создавать как до, так и после разбиения на обучение и контроль, потому что мы не делаем вычислений, охватывающих все наблюдения исходного набора.

Давайте заменим пропуски в количественной переменной `income` средним значением.

Чтобы вычислить среднее значение, необходимо воспользоваться функцией `mean`.

```
# вычисляем среднее значение  
# переменной income  
mean(development$income, na.rm = TRUE)
```

Сводка 2.29. Среднее значение переменной `income`

```
[1] 50184.27
```

Выполняем импутацию пропусков переменной с помощью среднего значения.

```
# выполняем импутацию пропусков с помощью среднего, обратите
# внимание, среднее было вычислено отдельно для
# каждой выборки
development$income[is.na(development$income)] <- mean(development$income, na.rm=TRUE)
holdout$income[is.na(holdout$income)] <- mean(holdout$income, na.rm=TRUE)
```

Теперь выведем первые 10 наблюдений датафрейма **development**.

```
# выведем первые 10 наблюдений
# датафрейма development
head(development, 10)
```

Сводка 2.30. Первые 10 наблюдений после импутации количественных значений

	longdist	local	int_disc	billtype	pay	gender	marital	income	agecat	churn
1	<2	<8	Нет	Бюджетный	СС	Мужской	Женат	77680.00	<31	Уходит
2	<NA>	<8	Нет	Бесплатный	СС	Мужской	Женат	37111.50	<31	Уходит
4	<2	<8	<NA>	Бесплатный	CH	Мужской	Одинокий	50184.27	<31	Остается
5	<2	<NA>	Нет	Бесплатный	Auto	<NA>	Одинокий	16829.60	<31	Уходит
6	<NA>	<8	Нет	Бесплатный	<NA>	Женский	Одинокий	57272.70	<31	Уходит
7	<NA>	<8	Нет	Бюджетный	<NA>	Мужской	Женат	50184.27	<31	Уходит
9	<2	<8	Да	<NA>	CH	Мужской	Одинокий	92167.30	<31	Уходит
10	<2	<8	Нет	Бесплатный	СС	Мужской	Женат	37135.50	<31	Уходит
11	<2	<8	Нет	Бесплатный	CH	Мужской	Одинокий	18831.10	<31	Остается
12	<2	<8	Нет	Бюджетный	СС	Женский	Женат	19654.90	<31	Уходит

В сводке 2.30 видно, что импутация средними значениями выполнена успешно (импутированные значения выделены желтым и они совпадают со средним значением, приведенным в сводке 2.29).

Обратите внимание, что импутацию мы выполнили с помощью среднего значения, вычисленного отдельно для каждой выборки.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

Часто нужно вычислить средние значения по нескольким количественным переменным. Допустим, у нас есть датафрейм **data**, включающий количественные переменные *var1*, *var2* и *var3*.

Создаем вектор из названий этих количественных переменных.

```
names <- c("var1", "var2", "var3")
```

Затем необходимо воспользоваться функцией **sapply**, которая применит к каждой количественной переменной датафрейма **development** (имена переменных перечислены в векторе **names**) функцию **mean** и таким образом вычислит среднее значение по каждой количественной переменной. Для удобства можно округлить полученные средние значения до второго знака после десятичной знака с помощью функции **round**.

```
round(sapply(data[names], mean, na.rm = T), 2)
```

Если нужно заменить средние значения по нескольким количественным переменным, можно вновь создать вектор из названий нужных переменных, написать собственную функцию **imp_missing**, которая будет заменять пропуски средним значением, и применить ее с помощью **sapply**.

```
names <- c("var1", "var2", "var3")

imp_missing <- function(x) {
  x[is.na(x)] <- mean(x, na.rm=TRUE)
  x
}

data[names] <- sapply(data[names], imp_missing)
```

Кроме того, пропуски в нескольких количественных переменных можно импутировать еще более элегантным способом, не создавая собственную функцию и вектор, состоящий из названий предикторов. Для этого необходимо воспользоваться функциями `sapply`, `lapply`, `ifelse` и `is.na`.

```
data[sapply(data, is.numeric)] <- lapply(data[sapply(data,
  is.numeric)], function(x)
  ifelse(is.na(x),
    mean(x, na.rm = TRUE), x))
```

Функция `lapply` применяет функцию `ifelse` к количественным переменным датафрейма `development` (мы отобрали эти переменные с помощью функции `sapply`). Функция `ifelse()` воспринимает первый аргумент как условие, второй аргумент возвращается, если условие верно, а третий аргумент – если нет. В данном случае мы с помощью функции `is.na` проверяем, является ли значение переменной пропущенным, если это так, возвращаем среднее значение, если нет, возвращаем исходное значение.

Теперь приступим к импутации пропусков в категориальных переменных. Начнем с переменной *billtype* и рассмотрим различные способы импутации.

Как вариант, можно было создать индикатор пропусков. Функция `ifelse` проверяет, является ли значение переменной пропущенным, если это так, возвращает значение 1, если нет, возвращает значение 0.

```
# создаем индикатор пропусков, у функции ifelse первый аргумент -
# проверяемое условие (является ли наблюдение переменной billtype
# пропущенным), второй аргумент - значение, которое возвращается,
# если условие выполняется, третий аргумент - значение, которое
# возвращается, если условие не выполняется
development$billtype_ind <- ifelse(is.na(development$billtype), 1, 0)
```

Выведем первые 10 наблюдений новой переменной *billtype_ind*.

```
# выводим первые 10 наблюдений новой
# переменной billtype_ind
head(development[, "billtype_ind"], 10)
```

Сводка 2.31. Первые 10 наблюдений переменной `billtype_ind`

```
[1] 0 0 0 0 0 0 1 0 0 0
```

Однако в нашем случае у переменной *billtype* очень мало пропусков, поэтому особой пользы от индикатора пропусков не будет. Давайте удалим его.

```
# удаляем переменную billtype_ind
development$billtype_ind <- NULL
```

Пропуски можно записать в отдельную категорию *MISSING*. Воспользуемся функцией *recode* пакета *car* и создадим новую переменную *billtype2*, у которой пропуски будут записаны в отдельную категорию *MISSING*.

```
# с помощью функции recode пакета car запишем пропуски
# в отдельную категорию MISSING
development$billtype2 <- recode(development$billtype, "NA='MISSING'")
# посмотрим первые 10 наблюдений переменной billtype2
head(development[, "billtype2"], 10)
```

Сводка 2.32. Первые 10 наблюдений переменной *billtype2*

```
[1] Бюджетный Бесплатный Бесплатный Бесплатный Бесплатный Бюджетный MISSING
[8] Бесплатный Бесплатный Бюджетный
Levels: MISSING Бесплатный Бюджетный
```

Вновь удалим переменную.

```
# удаляем переменную billtype2
development$billtype2 <- NULL
```

В итоге остановимся на том, что заменим пропуск самой часто встречающейся категорией. Вместо того, чтобы сначала вычислить моду, а потом сделать импутацию с помощью нее, воспользуемся пакетом *imputeMissing*, который позволяет автоматизировать процесс импутации пропусков. Обратите внимание, что на данный момент пакет не поддерживает упорядоченные факторы, поэтому либо упорядоченные факторы необходимо перевести в обычные, либо изначально обрабатывать порядковые переменные как целочисленные векторы. Создаем вектор из названий порядковых переменных и с помощью функции *lapply* преобразуем эти переменные обратно в неупорядоченные факторы.

```
# с помощью функции lapply преобразовываем переменные
# longdist, local и agecat в неупорядоченные
# факторы
names <- c("longdist", "local", "agecat")
development[,names] <- lapply(development[,names], factor, ordered=FALSE)
holdout[,names] <- lapply(holdout[,names], factor, ordered=FALSE)
```

Во-первых, пакет *imputeMissing* удобен тем, что с помощью функции *compute* можно сразу вычислить для каждой количественной переменной – медиану, а для каждой категориальной переменной – моду.


```
# загружаем пакет imputeMissings
library(imputeMissings)
# с помощью функции compute пакета
# imputeMissings вычисляем моды и
# медианы для импутации
values <- compute(development)
values
```

Сводка 2.33. Медианы и моды для импутации пропусков, вычисленные на обучающей выборке

```
$longdist
[1] "15-20"

$local
[1] "21-35"

$int_disc
[1] "Нет"

$billtype
[1] "Бюджетный"

$pay
[1] "CC"

$gender
[1] "Мужской"

$marital
[1] "Женат"

$income
[1] 50290.7

$agecat
[1] "31-45"

$churn
[1] "Остается"
```

Во-вторых, если нужно получить сводку по количеству пропусков в каждой переменной, пакет `imputeMissings` предлагает воспользоваться функцией `colSums`.

```
# выводим сводку о количестве пропусков в
# каждой переменной с помощью функции
# colSums пакета imputeMissings
colSums(is.na(development))
```

Сводка 2.34. Сводка по пропускам в переменных обучающей выборки, полученная с помощью функции `colSums` пакета `imputeMissings`

```
longdist    local int_disc billtype    pay    gender marital    income    agecat
      3         1         1         1      2         1         0         0         0
churn
0
```

В-третьих, функция `impute` пакета `imputeMissings` позволяет автоматически выполнить импутацию данных. По умолчанию используется `method="median/mode"`, то есть пропущенные значения количественных переменных (в R им соответствуют целочисленные и числовые векторы) будут заменены медианой, а пропущенные значения категориальных переменных (в R им соответствуют символьные векторы и факторы) – модой. Если вместо значения `"median/mode"` будет выставлено значение `"randomForest"`, для импутации будет использован случайный лес (при этом импутируемые переменные не могут быть символьными векторами).

С помощью этой функции мы и выполним автоматизированную импутацию пропусков в категориальных переменных. Поскольку пропуски в нашей единственной количественной переменной уже заполнены, импутация медианами не будет выполнена.

Выполняем импутацию.

```
# выполняем автоматическую импутацию с помощью
# функции impute пакета imputeMissings
development <- impute(development, method = "median/mode")
holdout <- impute(holdout, method = "median/mode")
```

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

Если с помощью функции `impute` пакета `imputeMissings` необходимо вычислить статистики для импутации на обучающей выборке, а затем применить их к пропущенным наблюдениям обеих выборок, то программный код будет выглядеть так:

```
values <- compute(development)
development <- impute(development, object=values)
holdout <- impute(holdout, object=values)
```

Теперь, когда импутация выполнена, с помощью функции `ordered` преобразуем переменные *longdist*, *local* и *agecat* обратно в упорядоченные факторы.

```
# с помощью функции ordered преобразовываем переменные
# longdist, local и agecat обратно в упорядоченные
# факторы
development$longdist <- ordered(development$longdist,
                               levels = c("<2", "2-8", "9-14", "15-20", "21+"))
development$local <- ordered(development$local,
                             levels = c("<8", "8-20", "21-35", "36-55", "56+"))
development$agecat <- ordered(development$agecat,
                              levels = c("<31", "31-45", "46-58", "59-70", "71+"))
holdout$longdist <- ordered(holdout$longdist,
                            levels = c("<2", "2-8", "9-14", "15-20", "21+"))
holdout$local <- ordered(holdout$local,
                         levels = c("<8", "8-20", "21-35", "36-55", "56+"))
holdout$agecat <- ordered(holdout$agecat,
                          levels = c("<31", "31-45", "46-58", "59-70", "71+"))
```

Снова с помощью `sapply` выводим сводку о количестве пропусков по каждой переменной в обучающей и контрольной выборках.

```
# выводим информацию о пропусках в обучающей выборке
sapply(development, function(x) sum(is.na(x)))
```

Сводка 2.35. Отчет по пропущенным значениям в датафрейме `development` после импутации модами

```
longdist    local int_disc billtype    pay    gender    marital    income    agecat
0           0           0           0           0           0           0           0
churn
0
```

```
# выводим информацию о пропусках в контрольной выборке
sapply(holdout, function(x) sum(is.na(x)))
```

Сводка 2.36. Отчет по пропущенным значениям в датафрейме `holdout` после импутации модами

```
longdist    local int_disc billtype    pay    gender    marital    income    agecat
0           0           0           0           0           0           0           0
churn
0
```

2.2.10 Биннинг количественных переменных для получения порядковых предикторов

Напомню, что метод CHAID работает только с номинальными и порядковыми переменными. Необходимо представить нашу количественную переменную `income` в виде порядковой. Для этого нам необходимо выполнить биннинг.

Для количественных независимых переменных биннинг – это разбивка диапазона значений переменной на интервалы (бины). Например, есть переменная *Возраст* с диапазоном значений от 20 до 70 лет, можно разбить на интервалы: от 18 до 30 лет, от 31 года до 50 лет, от 51 года до 70 лет. В итоге получим категориальную переменную, в которой заданные нами интервалы являются категориями. Для категориальных независимых переменных биннинг – это переназначение (группировка) исходных категорий переменной. Например, есть переменная *Возраст* с категориями от 18 до 25 лет, от 26 до 35 лет, от 36 до 45 лет, от 46 до 55 лет, от 56 до 65 лет. Категории можно укрупнить, из пяти категорий сделать три: от 18 до 35 лет, от 36 до 55 лет, 56 лет и старше.

Основная причина проведения биннинга – это борьба с нелинейностью при построении скоринговых моделей на основе логистической регрессии. Часто взаимосвязь между непрерывной переменной и событием является нелинейной. Уравнение логистической регрессии, несмотря на то что ее выходное значение подвергается нелинейному преобразованию путем логита, все равно моделирует линейные зависимости между предикторами и зависимой переменной.

Для иллюстрации можно взять пример с нелинейной зависимостью между возрастом и событием (например, откликом). Допустим, рассчитанный регрессионный коэффициент в уравнении логистической регрессии получился отрицательным. Это значит, что вероятность

отклика с возрастом уменьшается. После проведенного биннинга, когда были выделены категории от 18 до 25 лет, от 26 до 35 лет, от 36 до 45 лет и старше 45 лет, оказалось, что зависимость между возрастом и событием нелинейная. Первая (молодые) и последняя (старший возраст) категории склонны к отклику, а промежуточные сегменты, наоборот, не склонны к отклику.

Биннинг удобен тем, что упрощает интерпретацию результатов логистической регрессии. В результате регрессионного оценивания категориям предиктора, полученным в ходе биннинга, присваиваются коэффициенты, который переводят в скоринговые баллы, характеризующие вероятности интересующего события. В зависимости от принадлежности к той или иной категории конкретной переменной клиент получает свой балл. Например, клиент в возрастной категории от 18 до 25 лет получает скоринговый балл 20, клиент в возрастной категории от 26 до 35 лет получает скоринговый балл 10 и т. д.

Однако у биннинга имеются и серьезные недостатки. Авторитетный статистик Фрэнк Харрелл приводит ряд причин, по которым не следует проводить биннинг количественных независимых переменных. Основной недостаток – это потеря прогнозной силы переменной в силу снижение ее информативности (вспомним, что наиболее полную информацию несет количественная шкала). Кроме того, в основе биннинга лежит некорректное предположение о том, что зависимость между предиктором и откликом внутри интервалов является монотонной (по мнению Харрелла, это предположение еще менее разумно, чем предположение о линейности). При разбиении всего диапазона значений переменной на интервалы первый и последний интервалы будут очень широкими, потому что плотность распределения в них низкая. После биннинга для вычисления p -значений и доверительных интервалов необходимо использовать сложное имитационное моделирование, потому что переменная после биннинга распределена совершенно ненормально, особенно если для выбора границ диапазонов использовалось значение зависимой переменной, то есть, по сути, информация «из будущего». Нельзя не отметить и очевидный субъективизм категоризации, выражающийся в том, что если нескольким исследователям предложить категоризировать переменную, они выберут разные границы интервалов. Стоит отметить, что в силу недостатков, изложенных ниже, биннинг как инструмент борьбы с нелинейностью используется все реже и уступает место преобразованиям на основе ограниченных кубических сплайнов (регрессионных сплайнов, кусочных кубических полиномов), логарифма, корней второй и третьей степени. В силу всего вышесказанного необходимость категоризации

количественных переменных перед выполнением CHAID является ахиллесовой пятой этого метода.

Самый простой вариант биннинга – разбить количественную переменную на определенное количество интервалов одинаковой ширины или одинакового объема.

Для этого воспользуемся базовой функцией `cut`. Функция `cut` имеет общий вид:

<code>x</code>	Задаёт числовой вектор, который нужно преобразовать в фактор путем разбиения на интервалы
<code>breaks</code>	Задаёт либо числовой вектор, состоящий из двух и более точек расщепления или конкретное число (больше или равно 2), определяющее количество интервалов. Когда задается конкретное число, создаются интервалы одинаковой ширины
<code>labels</code>	Задаёт метки для создаваемых уровней. По умолчанию метки создаются с помощью нотации интервала "(a,b]". Если задано значение FALSE, вместо фактора возвращаются простые целочисленные коды
<code>include.lowest</code>	Если задано значение TRUE, включает самое нижнее значение точек разбиения (при условии, что установлено значение параметра <code>right=TRUE</code>) или самое высокое значение точек разбиения (при условии, что установлено значение параметра <code>right=FALSE</code>)
<code>right</code>	Закрывает интервалы справа, если задано значение TRUE, либо закрывает интервалы слева, если задано значение FALSE
<code>dig.lab</code>	Задаёт целочисленное значение и используется, когда не заданы метки. Он определяет количество цифр, которое используется для отображения числовых значений точек разбиения. При отображении числовых значений точек разбиения в научном или экспоненциальном формате (используется для представления больших чисел) рекомендуется увеличить значение параметра.
<code>ordered_result</code>	Если задано значение TRUE, возвращает упорядоченный фактор

Выясним диапазон значений переменной *income*.

```
# выводим диапазон значений
# переменной income
range(development$income)
```

Сводка 2.37. Диапазон значений переменной *longdist*

```
[1] 110.28 99832.90
```

Разобьем количественную переменную *income* на 10 интервалов одинаковой ширины и создадим переменную *binned*.

```
# разбиваем переменную income на 10 интервалов
# одинаковой ширины
development$binned <- cut(x=development$income, breaks=seq(0, 100000, by=10000),
                          include.lowest=TRUE, right=TRUE,
                          ordered_result=TRUE, dig.lab = 6)
```

Взглянем на нашу новую переменную *binned*.

```
# смотрим распределение значений
# переменной binned
summary(development$binned)
```

Сводка 2.38. Порядковая переменная *binned*, полученная в результате биннинга переменной *income* с помощью базовой функции *cut*

[0,10000]	(10000,20000]	(20000,30000]	(30000,40000]	(40000,50000]	(50000,60000]
278	321	307	307	283	382
(60000,70000]	(70000,80000]	(80000,90000]	(90000,100000]		
319	268	308	307		

Мы получили переменную, состоящую из 10 интервалов примерно одинаковой ширины. Обратите внимание на квадратные и круглые скобки интервалов. Интервал закрывается либо слева, либо справа, то есть соответствующий конец включается в данный интервал. Согласно принятой в математике нотации интервалов круглая скобка означает, что соответствующий конец не включается (открыт), а квадратная – что включается (закрит). В данном случае интервалы открыты слева и закрыты справа, за исключением первого интервала, который закрыт слева и справа. Первый интервал $[0,10000]$ содержит наблюдения со значениями переменной от 0 до 10000 включительно, второй интервал $(10000,20000]$ содержит наблюдения, в которых значение переменной больше 10000, но при этом меньше или равно 20000, и так далее.

Недостаток этого варианта биннинга заключается в том, что мы получили интервалы одинаковой ширины, но при этом они сильно отличаются по размерам. Теперь попробуем разбить переменную *longdist* так, чтобы получить интервалы одинакового размера (квантили). Для этого воспользуемся функцией *quantileCut* пакета *lsr*. Вместо создания интервалов одинаковой ширины (то, что делает базовая функция *cut*), функция *quantileCut* использует базовую функцию *quantile* для разбиения переменной на интервалы неодинаковой ширины так, чтобы каждый интервал содержал примерно одинаковое количество наблюдений.

```
# загружаем пакет lsr
library(lsr)
# с помощью функции quantileCut пакета lsr разбиваем
# переменную income на 10 интервалов с одинаковым
# количеством наблюдений
development$binmed2 <- quantileCut(x=development$income, n=10,
                                include.lowest=TRUE, right=TRUE,
                                ordered_result=TRUE, dig.lab=6)
```

Взглянем на новую переменную *binmed2*.

```
# смотрим распределение значений
# переменной binmed2
summary(development$binmed2)
```

Сводка 2.39. Порядковая переменная *binmed2*, полученная в результате биннинга переменной *longdist* с помощью функции *quantileCut*

```
[10.5574,11254.7] (11254.7,20806.8] (20806.8,30901.3] (30901.3,40756] (40756,50290.7]
              308              308              310              306              341
(50290.7,58990.9] (58990.9,68767] (68767,79962.5] (79962.5,89931.5] (89931.5,99932.6]
              275              309              307              308              308
```

Теперь мы получили переменную, состоящую из 10 интервалов примерно одинакового размера. В первый интервал вошли наблюдения со значениями переменной от 10,5574 до 11254,7 включительно. Во второй интервал вошли все наблюдения, у которых значения больше 11254,7 и при этом они меньше или равны 20806,8.

Теперь вновь взглянем на типы переменных.

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

Сводка 2.40. Типы переменных в датафрейме *data* (новые переменные *binmed* и *binmed2* выделены желтым)

```
'data.frame':      3080 obs. of  12 variables:
 $ longdist: Ord.factor w/ 5 levels "<2"<"2-8"<"9-14"<...: 1 4 1 1 4 4 1 1 1 1 ...
 $ local   : Ord.factor w/ 5 levels "<8"<"8-20"<"21-35"<...: 1 1 1 3 1 1 1 1 1 1 ...
 $ int_disc: Factor w/ 2 levels "Да","Нет": 2 2 2 2 2 2 1 2 2 2 ...
 $ billtype: Factor w/ 2 levels "Бесплатный","Бюджетный": 2 1 1 1 1 2 2 1 1 2 ...
 $ pay      : Factor w/ 3 levels "Auto","CC","CH": 2 2 3 1 2 2 3 2 3 2 ...
 $ gender   : Factor w/ 2 levels "Женский","Мужской": 2 2 2 2 1 2 2 2 2 1 ...
 $ marital  : Factor w/ 2 levels "Женат","Одинокий": 1 1 2 2 2 1 2 1 2 1 ...
 $ income   : num  77680 37112 50184 16830 57273 ...
 $ agecat   : Ord.factor w/ 5 levels "<31"<"31-45"<...: 1 1 1 1 1 1 1 1 1 1 ...
 $ churn    : Factor w/ 2 levels "Остается","Уходит": 2 2 1 2 2 2 2 2 1 2 ...
 $ binmed   : Ord.factor w/ 10 levels "[0,10000]"<"(10000,20000]"<...: 8 4 6 2 6 6 ...
 $ binmed2  : Ord.factor w/ 10 levels "[10.5574,11254.7]"<...: 8 4 5 2 6 5 10 4 2 2 ...
```

Из сводки 2.40 видно, что переменные *binmed* и *binmed2* являются порядковыми, в среде R им соответствуют упорядоченные факторы (*ordered factors*).

Давайте удалим наши новые переменные.

```
# удаляем только что созданные переменные
development$binned <- NULL
development$binned2 <- NULL
```

Теперь применим немного модифицированные правила разбиения, найденные на обучающей выборке, к количественным переменным обучающей и контрольной выборок.

```
# выполняем биннинг с учетом полученных правил разбиения
development$income <- cut(x=development$income,
                          breaks=c(-Inf,11000,21000,31000,41000,
                                    50000,59000,69000,80000,
                                    90000,+Inf),
                          include.lowest=TRUE,
                          ordered_result=TRUE)

holdout$income <- cut(x=holdout$income,
                     breaks=c(-Inf,11000,21000,31000,41000,
                               50000,59000,69000,80000,
                               90000,+Inf),
                     include.lowest=TRUE,
                     ordered_result=TRUE)
```

Обратите внимание, что как и в случае с импутацией статистиками, нельзя создать с помощью биннинга новую переменную на общем наборе данных, а потом разбить набор на обучение и контроль и работать с такой переменной в соответствующей выборке, как с обычной исторической переменной. Это обусловлено тем, что для биннинга используется информация о распределении значений переменной по всему набору данных. В результате получится, что в контрольной выборке мы будем использовать переменную, категории которой были получены, исходя из информации всего набора данных. Здесь важно понять, что с помощью биннинга мы на обучающей выборке получаем правила дискретизации (для количественных переменных) и правила перегруппировки (для категориальных переменных), которые применяются к соответствующей переменной в обучающей и контрольной выборках.

Теперь у нас все готово для моделирования.

Лекция 2.3 Построение модели дерева CHAID и ее проверка

2.3.1 Построение модели и работа с диаграммой дерева

Загрузим установленный пакет CHAID с помощью функции `library`.

```
# загружаем пакет CHAID  
library(CH Aid)
```

Чтобы построить дерево CHAID, необходимо воспользоваться функцией `chaid`. Функция `chaid` имеет общий вид:

```
chaid(formula, control=chaid_control(), data)
```

где

formula	Задаёт формулу в формате <i>Зависимая переменная ~ Предиктор1 + Предиктор2 + Предиктор3 + др.</i>
control	Задаёт критерии роста дерева CHAID с помощью функции <code>chaid_control</code> : <ul style="list-style-type: none">• alpha2 – уровень значимости для объединения категорий, по умолчанию 0.05;• alpha3 – уровень значимости для разбиения уже сформированных объединённых категорий, по умолчанию равен -1, т.е. не используется (чтобы включить этот параметр, необходимо задать положительное число не больше 1);• alpha4 – уровень значимости для разбиения узла;• minsplit – минимальное число наблюдений в узле перед расщеплением, по умолчанию 20;• minbucket – минимальное абсолютное количество наблюдений в терминальном узле, по умолчанию 7;• minprob – минимальная относительная частота наблюдений в терминальном узле, по умолчанию 0.01;• stump – выполняется лишь разбиение корневого узла, по умолчанию не используется;• maxheight – максимальная высота дерева, по умолчанию равен -1, т.е. не используется. Минимальное количество наблюдений в каждом терминальном узле задают с помощью параметров minbucket и minprob . Если вы хотите, чтобы лишь параметр minbucket

	определял минимальный размер узла, задайте параметр minprob равным 1.
data	Задаёт таблицу данных для анализа. Если таблица данных названа data , можно просто указать data .

С помощью вспомогательной функции `chaid_control` задаём набор условий для построения дерева (значения параметров `minsplit` и `minbucket` увеличиваем для удобства отрисовки диаграммы дерева), передаём его параметру `control` основной функции `chaid` и строим модель дерева классификации по всем исходным предикторам.

```
# задаём набор условий для построения дерева CHAID
params <- chaid_control(minprob = 0.01, minsplit = 1000, minbucket = 500)
# строим модель дерева CHAID
chd <- chaid(churn ~ ., control = params, development)
```

Выведем диаграмму дерева. Увеличение значений параметров `minsplit` и `minbucket` приведет к построению менее ветвистого дерева.

```
# выводим диаграмму дерева
plot(chd)
```

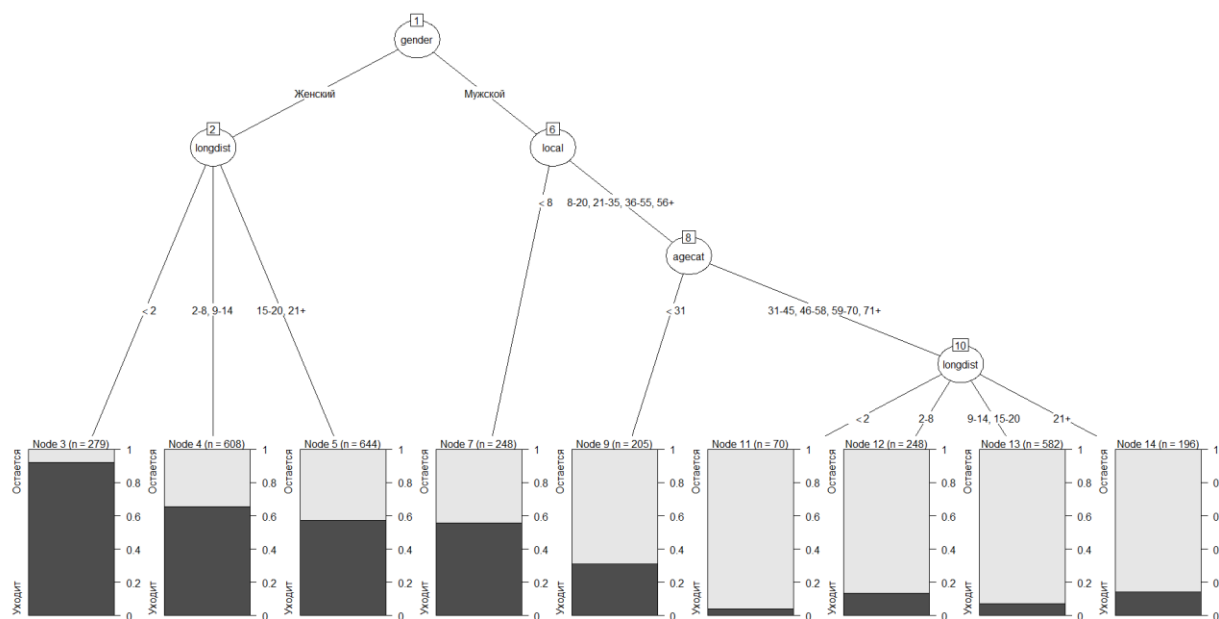


Рисунок 2.8 Диаграмма дерева классификации CHAID

Диаграмма дерева (рисунок 2.8) не всегда удобна в плане интерпретации по причине того, что определение одного узла часто накладывается на определение другого узла. Поэтому можно вывести диаграмму дерева в схематичном виде.

```
# выводим диаграмму дерева
# в схематичном виде
print(chd)
```

Сводка 2.41. Схематичное изображение дерева классификации CHAID

Model formula:

```
churn ~ longdist + local + int_disc + billtype + pay + gender +  
marital + income + agecat
```

Fitted party:

```
[1] root  
| [2] gender in Женский  
| | [3] longdist <2: Уходит (n = 279, err = 7.9%)  
| | [4] longdist in 2-8, 9-14: Уходит (n = 608, err = 34.4%)  
| | [5] longdist in 15-20, 21+: Уходит (n = 644, err = 42.7%)  
| [6] gender in Мужской  
| | [7] local <8: Уходит (n = 248, err = 44.4%)  
| | [8] local in 8-20, 21-35, 36-55, 56+  
| | | [9] agecat <31: Остается (n = 205, err = 31.2%)  
| | | [10] agecat in 31-45, 46-58, 59-70, 71+  
| | | | [11] longdist <2: Остается (n = 70, err = 4.3%)  
| | | | [12] longdist in 2-8: Остается (n = 248, err = 13.3%)  
| | | | [13] longdist in 9-14, 15-20: Остается (n = 582, err = 7.4%)  
| | | | [14] longdist in 21+: Остается (n = 196, err = 14.3%)
```

Number of inner nodes: 5

Number of terminal nodes: 9

В сводке 2.41 приводится уже более подробная информация о построенном дереве. **Number of inner nodes** показывает количество внутренних узлов (оно равно общему количеству узлов минус количество терминальных узлов). **Number of terminal nodes** показывает количество терминальных узлов.

Как и на диаграмме дерева, в сводке мы видим, что корневой узел (**root**) разбивается по предиктору *Пол* [*gender*] на узел 2 (женщины) и узел 6 (мужчины). Узел 2 (женщины) разбивается по предиктору *Длительность междугородних звонков в минутах* [*longdist*] на узлы 3 (меньше 2 минут), 4 (от 2 до 14 минут) и 5 (больше 14 минут), при этом все три узла становятся терминальными. Узел 6 (мужчины) разбивается по предиктору *Длительность местных звонков в мин* [*local*] на узел 7 (менее 8 минут), который становится терминальным, и узел 8 (от 8 минут и выше). Узел 8 разбивается в свою очередь по предиктору *Возрастная категория* [*agecat*] на узел 9 (моложе 31 года), который становится терминальным, и узел 10 (от 31 года и старше). Затем узел 10 (от 31 года и старше) разбивается по предиктору *Длительность междугородних звонков в минутах* [*longdist*] на узлы 11 (менее 2 минут), 12 (от 2 до 8 минут), 13 (от 9 до 20 минут) и 14 (от 21 минуты и выше). Для терминальных узлов выводится более подробная информация: спрогнозированная категория зависимой переменной (категория *Остается* или *Уходит*), количество наблюдений в узле (**n**), доля неправильно классифицированных наблюдений (**err**).

Например, рассмотрим терминальный узел 3. Речь идет о женщинах, тратящих на междугородние звонки менее 2 минут. Для этого узла

прогнозируется категория *Уходит*. Узел состоит из 279 наблюдений, при данном прогнозе 7.9% наблюдений классифицируются неверно.

Теперь внимательно рассмотрим терминальные узлы, по которым спрогнозирована категория *Уходит*. Можно выделить четыре таких узла (для удобства в сводке 2.29 они выделены желтым фоном):

- женщины, тратящие на междугородние звонки менее 2 минут (узел 3);
- женщины, тратящие на междугородние звонки от 2 до 14 минут (узел 4);
- женщины, тратящие на междугородние звонки больше 14 минут (узел 5);
- мужчины, тратящие на местные звонки менее 8 минут (узел 7).

2.3.2 Получение прогнозов модели

Пришло время взглянуть на прогнозы модели. Для получения прогнозов нужно воспользоваться функцией `predict`. Функция `predict` имеет общий вид:

```
predict(model, data, type)
```

где

model	Задаёт подогнанную модель, с помощью которой нужно вычислить прогнозы
data	Задаёт таблицу данных, для которой нужно вычислить прогнозы
type	Задаёт тип прогнозов: <ul style="list-style-type: none">• "prob" – будут вычислены вероятности классов зависимой переменной;• "response" – будут спрогнозированы значения (классы) зависимой переменной

Вычисление прогнозов начнем с вероятностей классов зависимой переменной `churn`. Давайте вычислим вероятности классов для наблюдений контрольной выборки, то есть для наблюдений, которые не участвовали в обучении модели. Чтобы вычислить вероятности, для аргумента `type` функции `predict` нужно задать значение "prob". Для удобства запишем полученные вероятности в объект `prob_hold` и выведем результаты для первых 10 наблюдений.

```
# выводим спрогнозированные вероятности отрицательного
# класса для первых 10 наблюдений
# контрольной выборки
prob_hold <- predict(chd, holdout, type="prob")
prob_hold[1:10]
```

Сводка 2.42. Спрогнозированные вероятности отрицательного класса зависимой переменной для первых 10 наблюдений контрольной выборки

```
[1] 0.44354839 0.44354839 0.44354839 0.07885305 0.07885305 0.44354839 0.07885305 0.44354839
[9] 0.44354839 0.44354839
```

В сводке 2.42 приводятся спрогнозированные вероятности для первых 10 наблюдений контрольной выборки. По умолчанию выводится вероятность отрицательного (первого) класса, которым является класс *Остается*.

Для удобства работы спрогнозированные вероятности можно округлить с помощью функции `round`. Функция `round` имеет общий вид

```
round(x, digits = 0)
```

где

x	Задаёт числовой вектор
digits	Задаёт количество десятичных знаков

Давайте округлим наши вероятности до второго десятичного знака.

```
# округляем спрогнозированные вероятности
# до второго десятичного знака
round(prob_hold[1:10], 2)
```

Сводка 2.43. Спрогнозированные вероятности отрицательного класса зависимой переменной для первых 10 наблюдений контрольной выборки, округленные до второго десятичного знака

```
[1] 0.44 0.44 0.44 0.08 0.08 0.44 0.08 0.44 0.44 0.44
```

Вероятности классов для конкретного наблюдения – это просто дробные доли классов зависимой переменной в том терминальном узле, в которое попало интересующее нас наблюдение. В ходе обучения (построения дерева) каждое наблюдение попадает в конкретный терминальный узел. Например, интересующее нас наблюдение попало в терминальный узел, где 44% наблюдений относится к классу *Остается*, а 56% наблюдений – к классу *Уходит*. Соответственно вероятности классов будут записаны в дробном виде как 0,44 и 0,56. Дерево запоминает правила разбиения, в результате которых был получен каждый терминальный узел и соответствующие ему вероятности классов. Давайте с помощью самостоятельно написанной функции `treetable` извлечем правила

разбиения, полученные нашей моделью. Дополнительно нам понадобится пакет **partykit**.

```
# извлекаем правила разбиения,  
# полученные нашей моделью  
library(partykit)  
treetable <- function(party_tree) {  
  
  df_list <- list()  
  var_names <- attr(party_tree$terms, "term.labels")  
  var_levels <- lapply(party_tree$data, levels)  
  
  walk_the_tree <- function(node, rule_branch = NULL) {  
    # проходим структуру разбиений дерева (рекурсивная функция)  
    # извлекаем правила для каждой ветви  
    if(missing(rule_branch)) {  
      rule_branch <- setNames(data.frame(t(replicate(length(var_names), NA))), var_names)  
      rule_branch <- cbind(rule_branch, nodeId = NA)  
      rule_branch <- cbind(rule_branch, predict = NA)  
    }  
    if(is.terminal(node)) {  
      rule_branch[["nodeId"]] <- node$id  
      rule_branch[["predict"]] <- predict_party(party_tree, node$id, type="prob")  
  
      df_list[[as.character(node$id)]] <- rule_branch  
    } else {  
      for(i in 1:length(node)) {  
        rule_branch1 <- rule_branch  
        val1 <- decision_rule(node,i)  
        rule_branch1[[names(val1)[1]]] <- val1  
        walk_the_tree(node[i], rule_branch1)  
      }  
    }  
  }  
  
  decision_rule <- function(node, i) {  
    # возвращаем правила разбиения в датафрейм вместе  
    # с названиями переменных и значениями  
    var_name <- var_names[node$split$varid[[1]]]  
    values_vec <- var_levels[[var_name]][ node$split$index == i]  
    values_txt <- paste(values_vec, collapse = ", ")  
    return( setNames(values_txt, var_name))  
  }  
  walk_the_tree(party_tree$node)  
  res_table <- Reduce(rbind, df_list)  
  return(res_table)  
}  
  
table <- treetable(chd)  
table
```

Сводка 2.44. Правила разбиения, в соответствии с которыми модель осуществляет прогнозы

	longdist	local	int_disc	billtype	pay	gender	marital	income
1	<2	<NA>	NA	NA	NA	Женский	NA	NA
2	2-8, 9-14	<NA>	NA	NA	NA	Женский	NA	NA
3	15-20, 21+	<NA>	NA	NA	NA	Женский	NA	NA
4	<NA>	<8	NA	NA	NA	Мужской	NA	NA
5	<NA> 8-20, 21-35, 36-55, 56+	NA	NA	NA	NA	Мужской	NA	NA
6	<2 8-20, 21-35, 36-55, 56+	NA	NA	NA	NA	Мужской	NA	NA
7	2-8 8-20, 21-35, 36-55, 56+	NA	NA	NA	NA	Мужской	NA	NA
8	9-14, 15-20 8-20, 21-35, 36-55, 56+	NA	NA	NA	NA	Мужской	NA	NA
9	21+ 8-20, 21-35, 36-55, 56+	NA	NA	NA	NA	Мужской	NA	NA

	agecat	nodeId	predict.Остается	predict.Уходит
1	<NA>	3	0.07885305	0.92114695
2	<NA>	4	0.34375000	0.65625000
3	<NA>	5	0.42701863	0.57298137
4	<NA>	7	0.44354839	0.55645161
5	<31	9	0.68780488	0.31219512
6	31-45, 46-58, 59-70, 71+	11	0.95714286	0.04285714
7	31-45, 46-58, 59-70, 71+	12	0.86693548	0.13306452
8	31-45, 46-58, 59-70, 71+	13	0.92611684	0.07388316
9	31-45, 46-58, 59-70, 71+	14	0.85714286	0.14285714

В итоге мы получаем 9 правил разбиения, соответствующих 9 терминальным узлам, а значит у нас будет всего 9 «комплектов» спрогнозированных вероятностей. На контрольной выборке или новых данных дерево проверяет каждое наблюдение на совпадение полученным правилам и присваивает соответствующие им вероятности классов.

При решении бизнес-задач все наблюдения делят на два класса: класс с отрицательными исходами (первый уровень зависимой переменной) и класс с положительными исходами (второй уровень зависимой переменной). Обычно положительный класс обозначает наступление какого-то важного для бизнеса события (оттока, отклика, дефолта) и является интересующим классом. В нашем примере важным событием является отток, поэтому положительным (интересующим) классом станет класс «Уходит» (соответствует ушедшим клиентам), а отрицательным классом – класс «Остается» (соответствует оставшимся клиентам).

Разбиение на два спрогнозированных класса получают с помощью варьирования порога отсечения – порогового значения спрогнозированной вероятности положительного класса, меняющегося в интервале 0 до 1. По умолчанию используется пороговое значение 0,5. Если вероятность положительного класса больше 0,5, то прогнозируется положительный класс, если она меньше этого порогового значения, прогнозируется отрицательный класс. Возьмем наше наблюдение, для которого были спрогнозированы вероятности классов 0,44 и 0,56. Нашим

положительным классом является класс *Уходит*. В данном случае вероятность класса *Уходит* равна 0,56 и превышает пороговое значение 0,5, поэтому прогнозируется класс *Уходит*.

Теперь давайте выведем спрогнозированные классы по первым 10 наблюдениям контрольной выборки. Поскольку вероятности – это числа с плавающей точкой, довольно редко бывает ситуация, когда они обе будут точно равны 0,5. Однако, если это произойдет, то прогноз будет осуществлен случайным образом. Поэтому для получения воспроизводимых результатов классификации рекомендуется задать стартовое значение генератора случайных чисел. Обратите внимание, чтобы вычислить классы, для аргумента `type` функции `predict` нужно выбрать значение `"response"`.

```
# задаем стартовое значение генератора
# случайных чисел для воспроизводимости
set.seed(42)

# выводим спрогнозированные классы
# для первых 10 наблюдений
# контрольной выборки
predvalue_hold <- predict(chd, holdout, type="response")
predvalue_hold[1:10]
```

Сводка 2.45. Спрогнозированные классы зависимой переменной для первых 10 наблюдений контрольной выборки

```
      3      8      14      18      22      25      35      37      38      43
Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
Levels: Остается Уходит
```

Видно, что для всех наблюдений, у которых вероятность класса *Уходит* была выше 0,5, предсказан класс *Уходит*.

Теперь выведем результаты классификации для контрольной выборки, передав функции `table` в качестве аргументов фактические значения зависимой переменной, записанные в `holdout$churn`, и спрогнозированные значения зависимой переменной, записанные в `predvalue_hold`.

```
# строим таблицу классификации
table(holdout$churn, predvalue_hold)
```

Сводка 2.46. Таблица классификации (для стандартного порога отсечения 0,5)

Спрогнозированные классы			
Фактические классы	→		
		Остается	Уходит
		Остается	Уходит
		Остается	Уходит
		464	282
		73	525

Неверно спрогнозированные наблюдения

Верно спрогнозированные наблюдения

В итоге получаем таблицу классификации. Строки таблицы – это фактические классы зависимой переменной, столбцы – спрогнозированные классы зависимой переменной.

На пересечении строки и столбца с одинаковым именем записывают количество правильно классифицированных наблюдений (для удобства выделены в таблице классификации зеленым цветом). В нашем случае на пересечении строки *Остается* и столбца *Остается* записано количество верно классифицированных оставшихся клиентов. На пересечении строки *Уходит* и столбца *Уходит* записано количество верно классифицированных ушедших клиентов.

На пересечении строки и столбца с разными именами записывают количество неправильно классифицированных наблюдений (для удобства выделены в таблице классификации красным цветом). В нашем случае на пересечении строки *Остается* и столбца *Уходит* записано количество неверно классифицированных оставшихся клиентов (они классифицированы как ушедшие). На пересечении строки *Уходит* и столбца *Остается* записано количество неверно классифицированных ушедших клиентов (они классифицированы как оставшиеся).

Таблица классификации позволяет получить информацию о правильности модели. *Правильность (ассигасу)* – это доля правильно классифицированных наблюдений от общего количества наблюдений (или просто доля правильных ответов). Правильность – один из показателей обобщающей способности для моделей классификации. Правильность классификации для класса *Остается* (оставшиеся клиенты) составляет 62,2%, или $(464/(464 + 282)) \cdot 100\%$. Мы берем количество верно спрогнозированных наблюдений класса *Остается* и делим на общее количество наблюдений класса *Остается*. Правильность классификации для категории *Уходит* (ушедшие клиенты) составляет 87,8%, или $(525/(525 + 73)) \cdot 100\%$. Мы берем количество верно спрогнозированных наблюдений класса *Уходит* и делим на общее количество наблюдений класса *Уходит*. Общая правильность классификации составляет 73,6%, или $(464+525)/(464+282+73+525)$. Мы берем общее количество верно спрогнозированных наблюдений и делим на общее количество наблюдений.

2.3.3 Работа с матрицей ошибок

На основе фактической и спрогнозированной принадлежности наблюдений к отрицательному или положительному классу возможны четыре типа случаев.

ТР (*True Positives*) – верно классифицированные положительные примеры, или истинно положительные случаи. Пример истинно положительного случая – ушедший клиент верно классифицирован как ушедший.

TN (*True Negatives*) – верно классифицированные отрицательные примеры, или истинно отрицательные случаи. Пример истинно отрицательного случая – оставшийся клиент верно классифицирован как оставшийся.

FN (*False Negatives*) – положительные примеры, неверно классифицированные как отрицательные (ошибка I рода). Это так называемый «ложный пропуск», когда интересующее нас событие ошибочно не обнаруживается (ложноотрицательные случаи). Пример ложноотрицательного случая – ушедший клиент ошибочно классифицирован как оставшийся.

FP (*False Positives*) – отрицательные примеры, неверно классифицированные как положительные (ошибка II рода). Это «ложная тревога», когда при отсутствии события ошибочно выносится решение о его присутствии (ложноположительные случаи).

Пример ложноположительного случая – оставшийся клиент ошибочно классифицирован как ушедший.

Эти четыре типа случаев образуют следующую матрицу ошибок (рис. 2.9).

фактический отрицательный класс	TN	FP
фактический положительный класс	FN	TP
	спрогнозированный отрицательный класс	спрогнозированный положительный класс

Рисунок 2.9 Матрица ошибок

Теперь построим матрицу ошибок для наших результатов классификации (рис. 2.10).

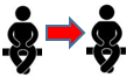



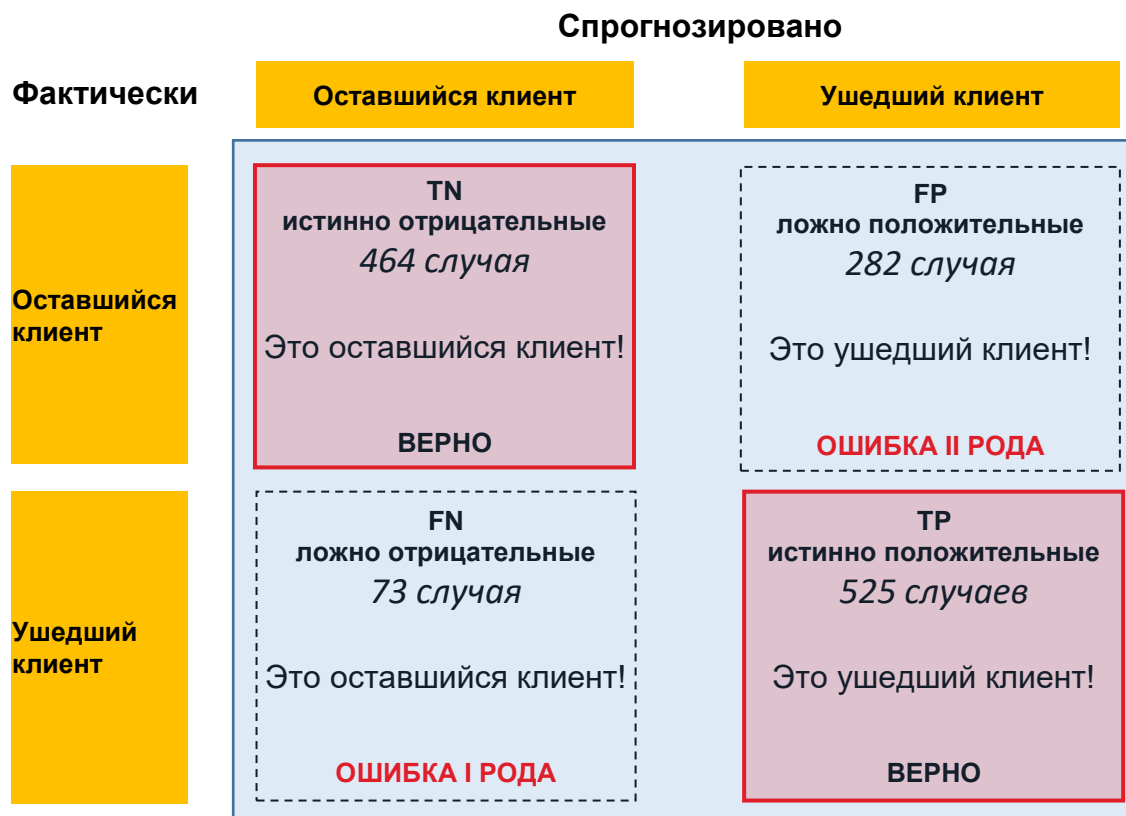
		Спрогнозировано	
Фактически		Оставшийся клиент	Ушедший клиент
	Оставшийся клиент	<p>TN истинно отрицательные 464 случая</p> <p>Это оставшийся клиент!</p>  <p>ВЕРНО</p>	<p>FP ложно положительные 282 случая</p> <p>Это ушедший клиент!</p>  <p>ОШИБКА II РОДА</p>
	Ушедший клиент	<p>FN ложно отрицательные 73 случая</p> <p>Это оставшийся клиент!</p>  <p>ОШИБКА I РОДА</p>	<p>TP истинно положительные 525 случаев</p> <p>Это ушедший клиент!</p>  <p>ВЕРНО</p>

Рисунок 2.10 Четыре типа случаев для нашей таблицы классификации

Мы уже знакомы с одним из способов подытожить результаты классификации –вычислением *правильности* (*accuracy*), которую можно выразить в виде следующей формулы:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} = \frac{464 + 525}{464 + 282 + 73 + 525} = 73,6$$



$$\text{Правильность} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Рисунок 2.11 Правильность классификации

Правильность – это количество правильно классифицированных случаев (TP+TN, показано на рисунке 2.11 в виде областей красного цвета), поделенное на общее количество случаев (TP+TN+FP+FN, показано на рисунке 2.11 в виде области синего цвета).

Следует помнить, что у правильности есть серьезный недостаток, она не может служить достоверной метрикой качества при работе с несбалансированными наборами данных. Представьте, вам требуется выяснить отклик клиентов. У вас есть набор данных, в котором 13411 наблюдений соответствуют ситуации «не откликнулся» и 1812 наблюдений – «откликнулся». Другими словами, 88% примеров относятся к классу «не откликнулся». Такие наборы данных, в которых один класс встречается гораздо чаще, чем остальные, часто называют *несбалансированными наборами данных (imbalanced datasets)* или *наборами данных с несбалансированными классами (datasets with imbalanced classes)*. Теперь предположим, что вы построили модель дерева и получили следующую таблицу классификации.

	Не откликнулся	Откликнулся
Не откликнулся	13411	0
Откликнулся	1812	0

В итоге вы получили правильность, равную 88%, или $(13411+0)/(13411+0+1812+0)$, просто всегда прогнозируя класс «не откликнулся». Правильность по классу *Не откликнулся* у нас будет равна 100%, а правильность по классу *Откликнулся* будет равна 0%. Таким образом, получаем высокое значение правильности при нулевом качестве прогнозирования класса *Откликнулся*.

Кроме правильности есть еще три показателя, с помощью которых можно подытожить информацию матрицы ошибок и тем самым оценить качество классификации – чувствительность (Sensitivity), специфичность (Specificity) и 1 – специфичность ($1 - \text{Specificity}$).

Чувствительность – это количество истинно положительных случаев (True Positive Rate), поделенное на общее количество положительных случаев в выборке. Чувствительность еще называют *полнотой* (*recall*). Она измеряется по формуле:

$$Se = TPR = \frac{TP}{TP + FN} = \frac{525}{525 + 73} = 0,88$$

В нашем примере чувствительность – это способность модели правильно определять ушедших клиентов. Чувствительность – это правильность классификации для класса *Уходит*. Модель с высокой чувствительностью максимизирует долю правильно классифицированных ушедших клиентов. Чувствительность минимизирует вероятность совершения ошибки I рода, при этом увеличивая вероятность совершения ошибки II рода. Повышая чувствительность, мы минимизируем риск классифицировать ушедшего клиента как оставшегося, но при этом увеличиваем риск классифицировать оставшегося клиента как ушедшего. Образно говоря, увеличивая чувствительность, повышаем «пессимизм» модели. Наша модель демонстрирует высокую чувствительность.

		Спрогнозировано	
Фактически		Оставшийся клиент	Ушедший клиент
	Оставшийся клиент	TN истинно отрицательные 464 случая Это оставшийся клиент! ВЕРНО	FP ложно положительные 282 случая Это ушедший клиент! ОШИБКА II РОДА
	Ушедший клиент	FN ложно отрицательные 73 случая Это оставшийся клиент! ОШИБКА I РОДА	TP истинно положительные 525 случаев Это ушедший клиент! ВЕРНО

$$\text{Чувствительность} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Рисунок 2.12 Чувствительность

Специфичность – это количество истинно отрицательных случаев (True Negatives Rate), поделенное на общее количество отрицательных случаев в выборке. Она измеряется по формуле:

$$Sp = TNR = \frac{TN}{TN + FP} = \frac{464}{464 + 282} = 0,62$$

В нашем примере специфичность – это способность модели правильно определять оставшихся клиентов. Специфичность – это правильность классификации для класса *Остается*. Модель с высокой специфичностью максимизирует долю правильно классифицированных оставшихся клиентов. Специфичность минимизирует вероятность совершения ошибки II рода, при этом увеличивая вероятность совершения ошибки I рода. Повышая специфичность, мы минимизируем риск классифицировать оставшегося клиента как ушедшего, но при этом увеличиваем риск классифицировать ушедшего клиента как оставшегося. Образно говоря, увеличивая специфичность, повышаем «оптимизм» модели. Наша модель показывает высокий уровень специфичности.

		Спрогнозировано	
Фактически		Оставшийся клиент	Ушедший клиент
Оставшийся клиент		TN истинно отрицательные <i>464 случая</i> Это оставшийся клиент! ВЕРНО	FP ложно положительные <i>282 случая</i> Это ушедший клиент! ОШИБКА II РОДА
		FN ложно отрицательные <i>73 случая</i> Это оставшийся клиент! ОШИБКА I РОДА	TP истинно положительные <i>525 случаев</i> Это ушедший клиент! ВЕРНО

$$\text{Специфичность} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Рисунок 2.13 Специфичность

1 – специфичность (единица минус специфичность) – это количество ложно положительных случаев (False Positives Rate), поделенное на общее количество отрицательных случаев в выборке и вычисляется по формуле:

$$FPR = 1 - Sp = \frac{FP}{FP + TN} = 1 - 0,62 = \frac{282}{282 + 464} = 0,38$$

В нашем примере 1 – специфичность характеризует уровень «ложных срабатываний» модели, когда оставшийся клиент классифицируется как ушедший.

		Спрогнозировано	
Фактически		Оставшийся клиент	Ушедший клиент
Оставшийся клиент		TN истинно отрицательные 464 случая Это оставшийся клиент! ВЕРНО	FP ложно положительные 282 случая Это ушедший клиент! ОШИБКА II РОДА
		FN ложно отрицательные 73 случая Это оставшийся клиент! ОШИБКА I РОДА	TP истинно положительные 525 случаев Это ушедший клиент! ВЕРНО

$$1 - \text{специфичность} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Рисунок 2.14 1 – специфичность

Меняя пороговое значение спрогнозированной вероятности положительного класса, мы будем получать разные результаты классификации, а соответственно разные значения вышеприведенных показателей.

Давайте снизим пороговое значение до 0,3. Теперь класс *Уходит* прогнозируется, когда вероятность класса *Уходит* превышает пороговое значение 0,3.

```
# снижаем пороговое значение спрогнозированной
# вероятности положительного класса до 0.3
table(holdout$churn == "Уходит", predict(chd,
                                         holdout, type="prob"), ["Уходит"] >= 0.30)
```

Сводка 2.47. Таблица классификации
(для порога отсечения 0,3)

	FALSE	TRUE
FALSE	413	333
TRUE	49	549

Снизив пороговое значение, повышаем чувствительность – способность модели правильно классифицировать ушедших клиентов. При этом специфичность (способность модели правильно классифицировать

оставшихся клиентов) у нас снизится. Давайте проверим это утверждение.

Вычисляем чувствительность и специфичность.

$$Se = TPR = \frac{TP}{TP + FN} = \frac{549}{549 + 49} = 0,92$$
$$Sp = TNR = \frac{TN}{TN + FP} = \frac{413}{413 + 333} = 0,55$$

Действительно по сравнению с исходной моделью, использующей порог отсечения 0,5, чувствительность повысилась (с 0,88 до 0,92), а специфичность снизилась (с 0,62 до 0,55).

А сейчас повысим пороговое значение до 0,7. Теперь класс *Уходит* прогнозируется, когда вероятность класса *Уходит* превышает пороговое значение 0,7.

```
# повышаем пороговое значение спрогнозированной  
# вероятности положительного класса до 0.7  
table(holdout$churn == "Уходит", predict(chd,  
                                         holdout, type="prob"), "Уходит") >= 0.70)
```

Сводка 2.48. Таблица классификации
(для порога отсечения 0,7)

	FALSE	TRUE
FALSE	729	17
TRUE	492	106

Увеличив пороговое значение, снижаем чувствительность – способность модели правильно классифицировать ушедших клиентов. При этом специфичность (способность модели правильно классифицировать оставшихся клиентов) у нас повысится.

Вычисляем чувствительность и специфичность.

$$Se = TPR = \frac{TP}{TP + FN} = \frac{106}{106 + 492} = 0,18$$
$$Sp = TNR = \frac{TN}{TN + FP} = \frac{729}{729 + 17} = 0,98$$

Действительно по сравнению с исходной моделью, использующей порог отсечения 0,5, чувствительность снизилась (с 0,88 до 0,18), а специфичность повысилась (с 0,62 до 0,98).

Оптимальная модель должна обладать 100%-ной чувствительностью и 100%-ной специфичностью, но добиться этого невозможно. Повышая чувствительность, неизбежно снижаем специфичность и, наоборот, понижая чувствительность, неизбежно повышаем специфичность. Поэтому на практике строится ROC-кривая – кривая соотношений истинно положительных случаев (чувствительности) и ложно

положительных случаев (1 – специфичности) для различных порогов отсечения и выбирается такой порог отсечения, который дает оптимальное соотношение чувствительности и 1 – специфичности. Понятие «оптимальности» зависит от того, какая задача стоит перед моделером. Можно найти порог, при котором максимизируем долю правильно классифицированных оставшихся клиентов (специфичность), максимизируем долю правильно классифицированных ушедших клиентов (чувствительность) или достигаем баланса между специфичностью и чувствительностью. Давайте подробнее поговорим о ROC-кривой.

2.3.4 Знакомство с ROC-кривой и AUC

ROC-кривая (англ. receiver operating characteristic или рабочая характеристика приёмника) характеризует способность бинарного классификатора отличать отрицательный класс от положительного при разных порогах отсечения. Более строго, ROC-кривая – это кривая соотношений истинно положительных случаев (чувствительности) и ложно положительных случаев (1 – специфичности) для *различных* пороговых значений спрогнозированной вероятности интересующего класса. Используя ROC-кривую, мы не привязаны к конкретному пороговому значению, как в случае с правильностью.

Сам термин «receiver operating characteristic» пришел из теории обработки сигналов времен Второй мировой войны. После атаки на Перл Харбор в 1941 году, когда самолеты японцев были сначала ошибочно приняты за стаю перелетных птиц, а потом за грузовой конвой транспортных самолетов, и перед инженерами-электротехниками и инженерами по радиолокации была поставлена задача увеличить точность распознавания вражеских объектов по радиолокационному сигналу.

Допустим, у нас есть 20 наблюдений, из которых 12 наблюдений относятся к отрицательному классу, а 8 наблюдений – к положительному классу. С помощью бинарного классификатора мы получили следующие спрогнозированные вероятности положительного класса:

№	фактический класс	спрогнозированная вероятность положительного класса
1	N	0,18
2	N	0,24
3	N	0,32
4	N	0,33
5	N	0,4
6	N	0,53
7	N	0,58
8	N	0,59
9	N	0,6
10	N	0,7
11	N	0,75
12	N	0,85
13	P	0,52
14	P	0,72
15	P	0,73
16	P	0,79
17	P	0,82
18	P	0,88
19	P	0,9
20	P	0,92

Рисунок 2.15 Исходные спрогнозированные вероятности положительного класса

Построение ROC-кривой происходит следующим образом.

1. Сначала сортируем все наблюдения в порядке убывания спрогнозированной вероятности положительного класса.
2. Затем создаем график, у которого значения оси абсцисс будут значениями $1 - \text{специфичности}$ (цена деления оси задается значением $1/\text{neg}$), а значения оси ординат будут значениями чувствительности (цена деления оси задается значением $1/\text{pos}$). При этом **pos** – это количество наблюдений положительного класса, а **neg** – количество наблюдений отрицательного класса.
3. На графике задаем точку с координатами (0, 0) и для каждого отсортированного наблюдения x :
 - если x принадлежит положительному классу, двигаемся на $1/\text{pos}$ вверх;
 - если x принадлежит отрицательному классу, двигаемся на $1/\text{neg}$ вправо.

Итак, давайте отсортируем данные в порядке убывания спрогнозированных вероятностей положительного класса.

№	фактический класс	спрогнозированная вероятность положительного класса
20	P	0,92
19	P	0,9
18	P	0,88
12	N	0,85
17	P	0,82
16	P	0,79
11	N	0,75
15	P	0,73
14	P	0,72
10	N	0,7
9	N	0,6
8	N	0,59
7	N	0,58
6	N	0,53
13	P	0,52
5	N	0,4
4	N	0,33
3	N	0,32
2	N	0,24
1	N	0,18

Рисунок 2.16 Отсортированные спрогнозированные вероятности положительного класса

Теперь строим ROC-кривую. Цена деления оси ординат (оси чувствительности) будет равна $1/8$, поскольку у нас 8 наблюдений положительного класса. Цена деления оси абсцисс (оси 1 – специфичности) будет равна $1/12$, поскольку у нас 12 наблюдений отрицательного класса. Первое наблюдение принадлежит положительному классу, значит двигаемся на $1/8$ вверх. Второе наблюдение тоже принадлежит положительному классу, значит снова двигаемся на $1/8$ вверх. Третье наблюдение вновь принадлежит положительному классу, значит опять двигаемся на $1/8$ вверх. Четвертое наблюдение принадлежит отрицательному классу, двигаемся на $1/12$ вправо и так далее.

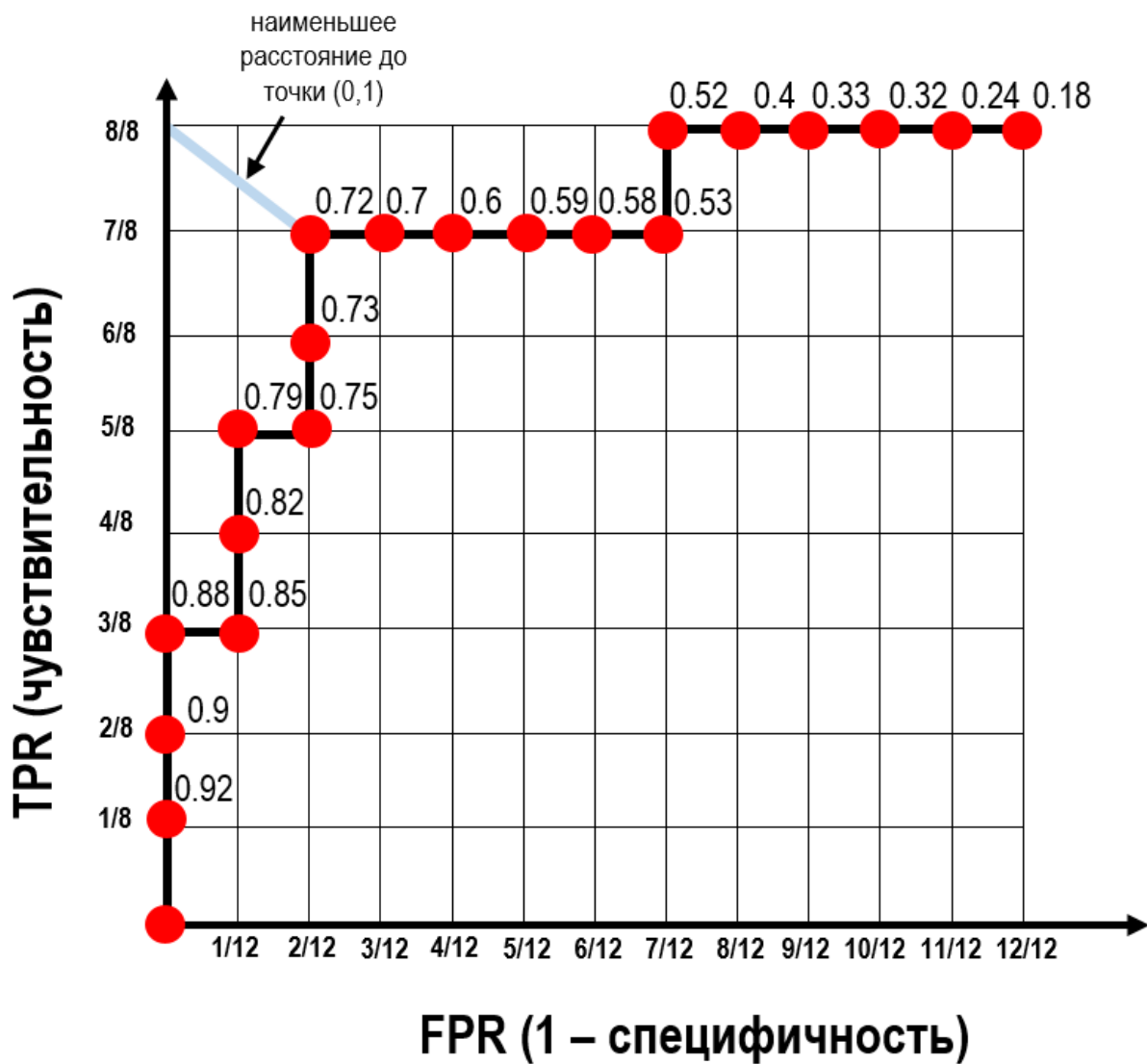


Рисунок 2.17 Построение ROC-кривой вручную

По мере построения ROC-кривой для каждого значения вероятности положительного класса записываем соответствующие ей пары значений 1 – специфичности и чувствительности (координаты).

№	фактический класс	спрогнозированная вероятность положительного класса	1 – специфичность или FPR (при данном пороге вероятности из 12 отрицательных наблюдений n наблюдений будут неверно классифицированы как положительные)	Чувствительность или TPR (при данном пороге вероятности из 8 положительных наблюдений n наблюдений будут верно классифицированы как положительные)
20	P	0,92	0	1/8
19	P	0,9	0	2/8
18	P	0,88	0	3/8
12	N	0,85	1/12	3/8
17	P	0,82	1/12	4/8
16	P	0,79	1/12	5/8
11	N	0,75	2/12	5/8
15	P	0,73	2/12	6/8
14	P	0,72	2/12	7/8
10	N	0,7	3/12	7/8
9	N	0,6	4/12	7/8
8	N	0,59	5/12	7/8
7	N	0,58	6/12	7/8
6	N	0,53	7/12	7/8
13	P	0,52	7/12	8/8
5	N	0,4	8/12	8/8
4	N	0,33	9/12	8/8
3	N	0,32	10/12	8/8
2	N	0,24	11/12	8/8
1	N	0,18	12/12	8/8

Рисунок 2.18 Отсортированные спрогнозированные вероятности положительного класса и соответствующие значения FPR и TPR

Значение вероятности положительного класса, при котором ROC-кривая находится на минимальном расстоянии от верхнего левого угла – точки с координатами (0,1), дает наибольшую правильность классификации. В данном случае таким значением будет значение 0,72. Из таблицы видно, что при пороге 0,72 мы правильно классифицируем 83,3% отрицательных (10 из 12 отрицательных, потому что согласно столбцу «FPR» только 2 из 12 отрицательных будут неверно классифицированы) и 87,5% (7 из 8 положительных согласно столбцу «TPR»). Таким образом, мы правильно классифицируем 17 человек из 20, т.е. правильность составляет 85%. ROC-кривая позволила нам найти такой порог отсечения, при котором чувствительность и 1 – специфичность практически сбалансированы. При идеальной классификации график ROC-кривой проходит через верхний левый угол. В этом случае доля истинно положительных

примеров составляет 100%, а доля ложно положительных примеров равна 0%. Поэтому чем ближе кривая к верхнему левому углу, тем выше дискриминирующая способность модели.

Визуальное сравнение двух и более ROC-кривых не всегда позволяет выявить наиболее эффективную модель. Для сравнения двух и более ROC-кривых сравниваются площади под кривыми. Площадь под кривой или AUC (Area Under Curve) меняется от 0,5 до 1. Чем больше значение AUC, тем выше качество модели. Наряду с правильностью AUC – один из важнейших показателей обобщающей способности для моделей бинарной классификации. Обычно считают, что значение AUC от 0,9 до 1 соответствует отличной дискриминирующей способности модели, от 0,8–0,9 – очень хорошей, 0,7–0,8 – хорошей, 0,6–0,7 – средней, 0,5–0,6 – неудовлетворительной. AUC можно вычислить с помощью численного метода трапеций, когда площадь под ROC-кривой аппроксимируется суммой площадей трапеций под кривой. Однако есть и другие способы вычислить AUC.

AUC классификатора C – это вероятность того, что классификатор C присвоит случайно отобранному положительному примеру более высокий ранг, чем случайно отобранному отрицательному примеру. Таким образом, $AUC(C) = P[C(x^+) > C(x^-)]$.

Давайте проверим это. Сначала создаем вектор из наших 20 фактических значений (классов) зависимой переменной.

```
# создаем вектор фактических значений (классов)
# зависимой переменной
cls = c('P', 'P', 'P', 'N', 'P', 'P', 'N', 'P', 'P', 'N', 'N', 'N',
        'N', 'N', 'N', 'P', 'N', 'N', 'N', 'N')
```

Создаем вектор спрогнозированных вероятностей положительного класса.

```
# создаем вектор спрогнозированных вероятностей
# положительного класса
score = c(0.92, 0.9, 0.88, 0.85, 0.82, 0.79, 0.75, 0.73, 0.72, 0.7, 0.6,
          0.59, 0.58, 0.53, 0.52, 0.4, 0.33, 0.32, 0.24, 0.18)
```

Записываем положительные и отрицательные примеры.

```
# записываем положительные
# и отрицательные примеры
pos = score[cls == 'P']
neg = score[cls == 'N']
```

Извлекаем случайным образом положительные и отрицательные примеры и вычисляем долю случаев, когда положительные примеры получили более высокий ранг, чем отрицательные примеры.

```
# задаем стартовое значение генератора
# случайных чисел для воспроизводимости
set.seed(14)

# извлекаем случайным образом положительные и
# отрицательные примеры и вычисляем долю случаев,
# когда положительные примеры получили более
# высокий ранг, чем отрицательные примеры
p = replicate(200000, sample(pos, size=1) > sample(neg, size=1))
mean(p)
```

Сводка 2.49. Значение AUC, вычисленное как доля случаев, когда положительный пример был проранжирован выше, чем отрицательный

```
[1] 0.864645
```

Обратите внимание, что AUC не зависит от преобразования спрогнозированных вероятностей (например, возведения в квадрат или деления на 2), поскольку зависит не от самих спрогнозированных вероятностей, а от меток классов наблюдений при упорядочивании по этим вероятностям. Давайте убедимся в этом.

Сначала разделим все вероятности на 2.

```
# разделим вероятности на 2
score_divided_by_2 = c(0.92, 0.9, 0.88, 0.85, 0.82, 0.79, 0.75, 0.73, 0.72, 0.7, 0.6,
                        0.59, 0.58, 0.53, 0.52, 0.4, 0.33, 0.32, 0.24, 0.18)/2

score_divided_by_2
```

Сводка 2.50. Вероятности положительного класса, разделенные на 2

```
[1] 0.460 0.450 0.440 0.425 0.410 0.395 0.375 0.365 0.360 0.350 0.300
[12] 0.295 0.290 0.265 0.260 0.200 0.165 0.160 0.120 0.090
```

Вычисляем AUC.

```
# снова вычисляем AUC как долю случаев, когда
# случайно отобранный положительный объект
# будет проранжирован выше, чем случайно
# отобранный отрицательный
pos = score_divided_by_2[cls == 'P']
neg = score_divided_by_2[cls == 'N']

set.seed(14)
p = replicate(200000, sample(pos, size=1) > sample(neg, size=1))
mean(p)
```

Сводка 2.51. Значение AUC: вероятности положительного класса поделены на 2

```
[1] 0.864645
```

Получаем то же самое значение AUC.

Теперь возведем вероятности в квадрат.

```
# возведем вероятности в квадрат
score_squared = c(0.92, 0.9, 0.88, 0.85, 0.82, 0.79, 0.75, 0.73, 0.72, 0.7, 0.6,
                  0.59, 0.58, 0.53, 0.52, 0.4, 0.33, 0.32, 0.24, 0.18)^2

score_squared
```

Сводка 2.52. Вероятности положительного класса, возведенные в квадрат

```
[1] 0.8464 0.8100 0.7744 0.7225 0.6724 0.6241 0.5625 0.5329 0.5184 0.4900 0.3600
[12] 0.3481 0.3364 0.2809 0.2704 0.1600 0.1089 0.1024 0.0576 0.0324
```

Вычисляем AUC.

```
# снова вычисляем AUC как долю случаев, когда
# случайно отобранный положительный объект
# будет проранжирован выше, чем случайно
# отобранный отрицательный
pos = score_squared[cls == 'P']
neg = score_squared[cls == 'N']

set.seed(14)
p = replicate(200000, sample(pos, size=1) > sample(neg, size=1))
mean(p)
```

Сводка 2.53. Значение AUC: вероятности положительного класса возведены в квадрат

```
[1] 0.864645
```

Вновь получаем то же самое значение.

А теперь для проверки вычислим AUC автоматически с помощью функции гос пакета `rROC`. Функция гос пакета `rROC` имеет общий вид:

```
roc(response, predictor, percent=FALSE, na.rm=TRUE
     auc=TRUE, plot=FALSE, ci=FALSE)
```

где

<code>response</code>	Задаёт вектор фактических ответов (вектор типа <code>factor</code> , <code>numeric</code> или <code>character</code>), обычно закодированных как 0 и 1
<code>predictor</code>	Задаёт числовой вектор спрогнозированных вероятностей интересующего класса зависимой переменной. Этот вектор должен быть той же длины, что и вектор фактических ответов
<code>percent</code>	Настраивает вывод значений чувствительности, специфичности и AUC. Если задано значение <code>FALSE</code> (по умолчанию), значения выводятся в виде дробных долей. Если задано значение <code>TRUE</code> , значения выводятся в процентном виде
<code>na.rm</code>	Если задано значение <code>TRUE</code> , значения <code>NA</code> удаляются (игнорируются при вычислении ROC-кривой)

auc	Задаёт вычисление значения AUC. Если задано значение TRUE (по умолчанию), AUC вычисляется. Если задано значение FALSE, AUC не вычисляется
plot	Задаёт построение ROC-кривой. Если задано значение FALSE (по умолчанию), график ROC-кривой не строится. Если задано значение TRUE, будет построен график ROC-кривой
ci	Задаёт вычисление доверительного интервала AUC. Если задано значение FALSE (по умолчанию), доверительный интервал AUC не вычисляется. Если задано значение TRUE, будет вычислен доверительный интервал AUC.

Мы передаём функции `roc` в качестве аргументов фактические значения зависимой переменной и вектор спрогнозированных вероятностей положительного класса.

```
# загружаем пакет pROC
library(pROC)
# автоматически вычисляем AUC
roc <- roc(cls, score)
```

Итак выводим информацию о значении AUC.

```
# выводим информацию о значении AUC
roc
```

Сводка 2.54. Значение AUC, вычисленное с помощью функции `roc` пакета `pROC`

```
Call:
roc.default(response = cls, predictor = score)

Data: score in 12 controls (cls N) < 8 cases (cls P).
Area under the curve: 0.8646
```

В результате получаем такое же значение AUC. Кроме того, есть ещё один способ вычислить AUC, не прибегая к методу трапеций. В рамках этого способа AUC вычисляется по формуле:

$$AUC = \frac{1}{PN} \sum_{j=1}^N (s_j - j) = \frac{1}{PN} \sum_{j=1}^N \sum_{t=1}^{s_j-j} 1$$

где:

P – количество наблюдений положительного класса;

N – количество наблюдений отрицательного класса.

s_j – ранг j -го наблюдения с отрицательным классом в последовательности, отсортированной по мере убывания вероятности положительного класса;
 $s_j - j$ – количество наблюдений положительного класса, лежащих выше j -го наблюдения отрицательного класса.

Итак, для каждого наблюдения с отрицательным классом в отсортированном наборе вычисляем количество наблюдений с положительным классом, которые имеют большее значение вероятности. Идем снизу вверх и фиксируем.

№	фактический класс	спрогнозированная вероятность положительного класса	$s_j - j$
20	P	0.92	
19	P	0.9	
18	P	0.88	
12	N	0.85	3
17	P	0.82	
16	P	0.79	
11	N	0.75	5
15	P	0.73	
14	P	0.72	
10	N	0.7	7
9	N	0.6	7
8	N	0.59	7
7	N	0.58	7
6	N	0.53	7
13	P	0.52	
5	N	0.4	8
4	N	0.33	8
3	N	0.32	8
2	N	0.24	8
1	N	0.18	8

Рисунок 2.19 Отсортированные спрогнозированные вероятности положительного класса (подсчет количества наблюдений положительного класса, лежащих выше соответствующего наблюдения отрицательного класса)

Берем наблюдение отрицательного класса 1. Подсчитываем количество наблюдений положительного класса, лежащих выше его. Такими наблюдениями будут наблюдения 13, 14, 15, 16, 17, 18, 19 и 20. Всего 8 наблюдений. Записываем это число напротив наблюдения 1. Аналогичный процесс повторяем для каждого наблюдения отрицательного класса. Затем суммируем полученные значения и сумму делим на произведение количества положительных примеров и количества отрицательных примеров.

$$AUC = \frac{1}{8 \times 12} (8 + 8 + 8 + 8 + 8 + 7 + 7 + 7 + 7 + 7 + 5 + 3) =$$

$$= \frac{83}{96} = 0.865$$

А теперь с помощью `plot.roc` пакета `pROC` построим саму ROC-кривую и сравним с той, что строили вручную.

Функция `plot.roc` пакета `pROC` имеет общий вид:

```
plot.roc(x, add=FALSE, axes=TRUE, legacy.axes=FALSE,
         print.auc=FALSE, percent=FALSE)
```

где

<code>x</code>	Задаёт объект, содержащий ROC-кривую, или вектор фактических ответов, если нет объекта с ROC-кривой
<code>add</code>	Задаёт построение дополнительной ROC-кривой на графике. Если задано значение <code>TRUE</code> , ROC-кривая будет добавлена на существующий график. Если задано значение <code>FALSE</code> , будет построен новый график ROC-кривой
<code>axis</code>	Задаёт вывод осей. Если задано значение <code>FALSE</code> (по умолчанию), будут выведены оси чувствительности и специфичности. Если задано значение <code>TRUE</code> , оси не выводятся
<code>legacy.axes</code>	Задаёт показатель, который будет отложен по оси абсцисс графика. По умолчанию задано значение <code>FALSE</code> и по оси абсцисс откладываются значения специфичности в убывающем порядке. Если задано значение <code>TRUE</code> , по оси абсцисс откладываются значения 1 – специфичности в возрастающем порядке
<code>print.auc</code>	Задаёт печать AUC на графике ROC-кривой
<code>percent</code>	Задаёт печать AUC на графике ROC-кривой в процентном виде
<code>color</code>	Задаёт цвет ROC-кривой

```
# автоматически строим ROC-кривую
plot.roc(x=roc, legacy.axes=TRUE)
```

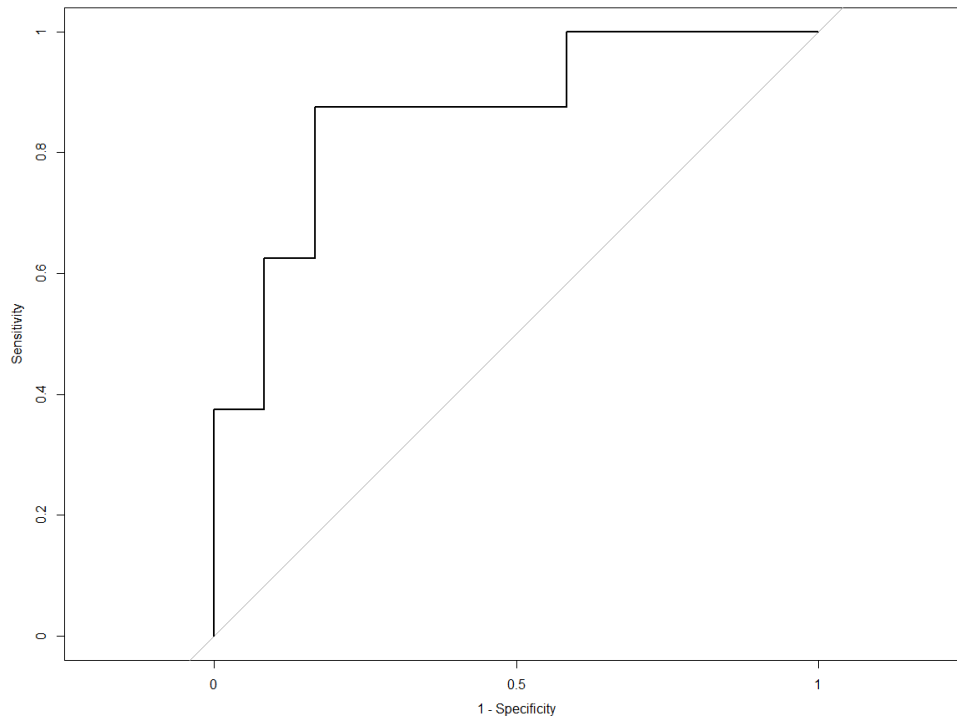


Рисунок 2.20 ROC-кривая, построенная автоматически с помощью функции `plot.roc` пакета `rROC`

Автоматически построенная ROC-кривая идентична той ROC-кривой, которую мы строили вручную. Обратите внимание на появившуюся диагональную линию. Она соответствует бесполезному классификатору, выдающему случайные прогнозы, и дана для удобства визуального анализа ROC-кривой. Чем ближе ROC-кривая к верхнему левому углу и соответственно чем больше она удалена от диагональной линии, тем модель эффективнее.

На практике бывают ситуации, когда при упорядочении наблюдений по убыванию вероятности положительного класса два наблюдения, принадлежащие разным классам, получают одинаковые вероятности.

№	фактический класс	спрогнозированная вероятность положительного класса
20	P	0.92
19	P	0.9
18	P	0.88
12	N	0.85
17	P	0.82
16	P	0.79
11	N	0.75
15	P	0.75
14	P	0.72
10	N	0.7
9	N	0.6
8	N	0.59
7	N	0.58
6	N	0.53
13	P	0.52
5	N	0.4
4	N	0.33
3	N	0.32
2	N	0.24
1	N	0.18

Рисунок 2.21 Отсортированные спрогнозированные вероятности положительного класса, двум наблюдениям разных классов присвоены одинаковые вероятности

Для таких наблюдений построение ROC-кривой осуществляется иначе. Вообще говоря, здесь могут быть две стратегии. Первая стратегия, которую называют «пессимистичной», заключается в том, чтобы поместить в начало такой последовательности сначала отрицательный пример, а затем положительный (двигаемся вправо и затем вверх, получаем нижний L-образный сегмент). Вторая стратегия, которую называют «оптимистичной», заключается в том, чтобы поместить в начало такой последовательности сначала положительный пример, а затем отрицательный (двигаемся вверх и затем вправо, получаем верхний L-образный сегмент). Компромиссная стратегия, применяющаяся на практике, заключается в усреднении пессимистичного и оптимистичного сегментов. Усреднением будет диагональ, проведенная в прямоугольнике, образованном этими двумя наблюдениями.

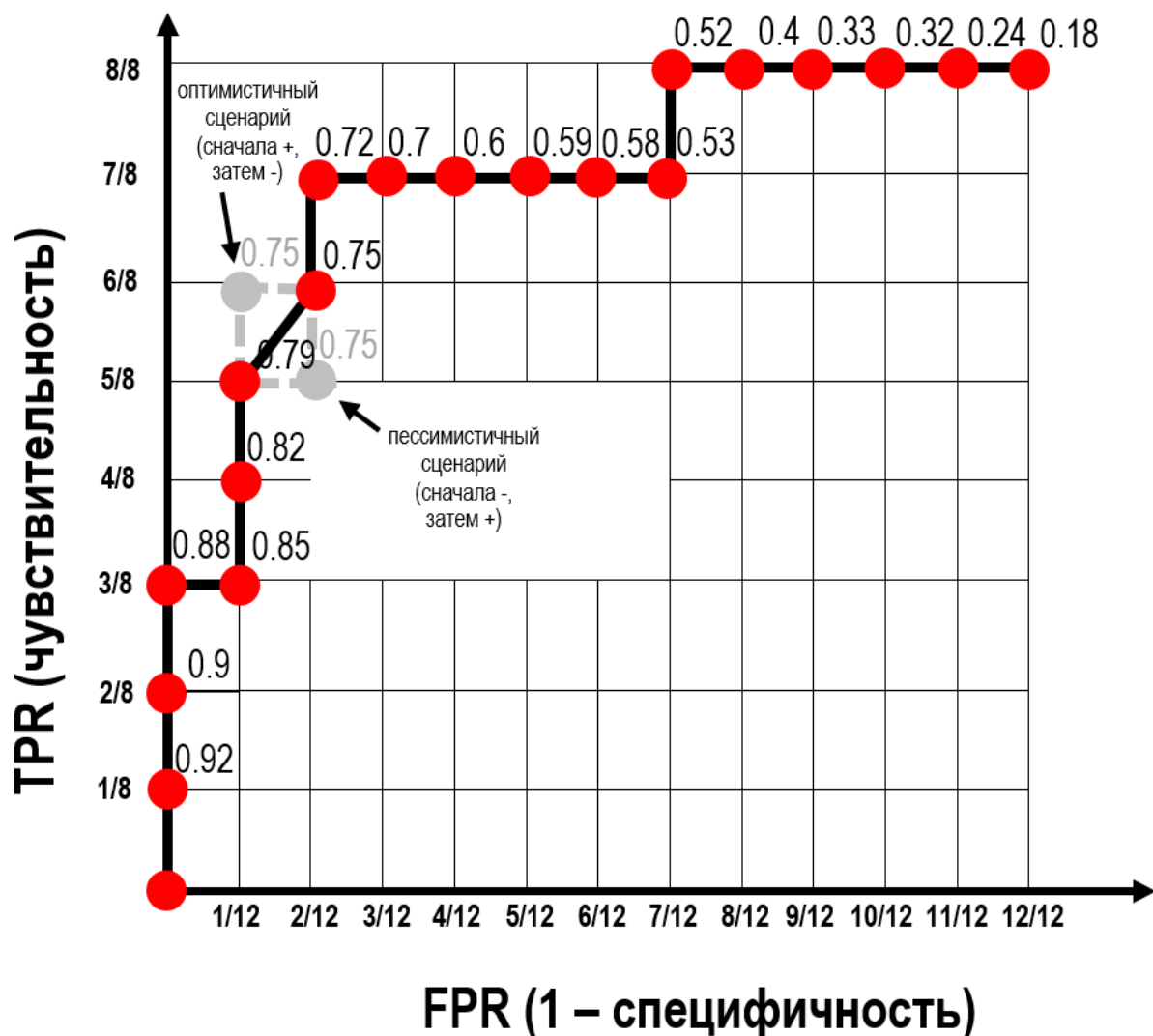


Рисунок 2.22 Построение ROC-кривой вручную в том случае, когда двум наблюдениям, относящимся к разным классам, присвоены одинаковые вероятности

Обратите внимание, что в отличие от правильности оценка AUC устойчива к дисбалансу классов, потому что она не использует информацию об исходных распределениях классов и их размерах, для нее важен лишь порядок ранжирования объектов.

2.3.5 Построение ROC-кривой и вычисление AUC для модели

Теперь с помощью функции `roc` пакета `rROC` вычислим AUC модели дерева CHAID для контрольной выборки. В качестве аргументов передаем фактические значения зависимой переменной `churn` в контрольном датафрейме `holdout` и спрогнозированные вероятности положительного класса этой же переменной в этом же датафрейме (им соответствует второй столбец в объекте `prob_hold`).

```
# выводим AUC нашей модели дерева
# для контрольной выборки
roc_hold <- roc(holdout$churn, prob_hold[,2])
roc_hold
```

Сводка 2.55. Значение AUC для модели дерева CHAID на контрольной выборке, вычисленное с помощью функции roc пакета rROC

```
Call:
roc.default(response = holdout$churn, predictor = prob_hold[, 2])

Data: prob_hold[, 2] in 746 controls (holdout$churn 0стается) < 598 cases (holdout$churn
Уходит).
Area under the curve: 0.7875
```

Построим график ROC-кривой модели CHAID для контрольной выборки.

```
# строим ROC-кривую нашей модели дерева
# для контрольной выборки
roc_hold <- roc(holdout$churn, prob_hold[,2])
roc_hold
plot.roc(roc_hold)
```

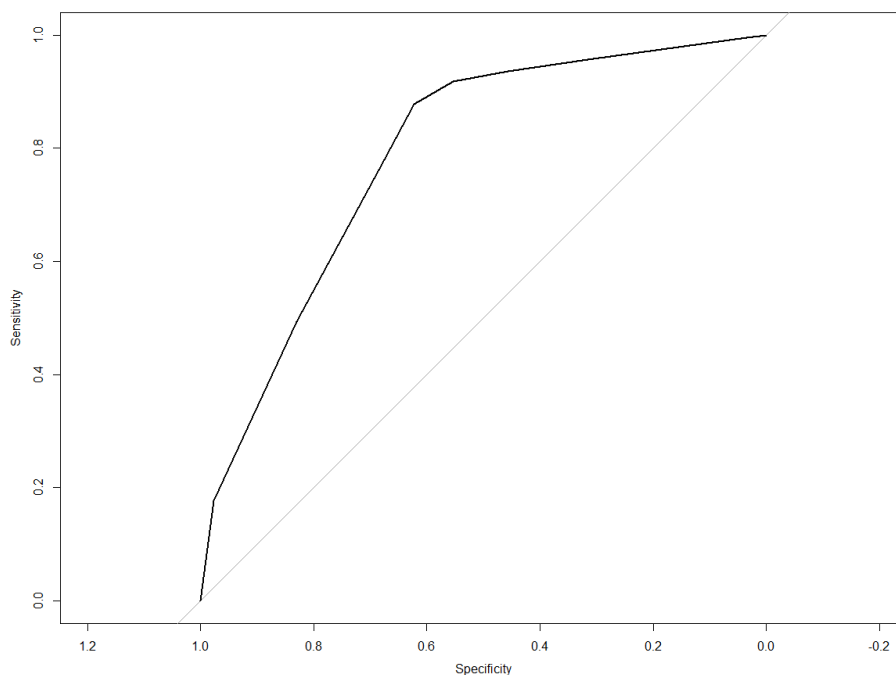


Рисунок 2.23 ROC-кривая модели CHAID для контрольной выборки, построенная с помощью функции roc пакета rROC

2.3.6 Вычисление интервальных оценок AUC для модели

Итак, мы построили модель, вычислили AUC, но здесь есть важный момент: мы построили ее на основе выборки, которую случайным образом извлекаем из генеральной совокупности – совокупности всех

объектов (единиц), охватываемых исследованием. Допустим, мы сформировали выборку, исследуем доход клиента и выяснили, что среднее значение дохода равно 83000. Это тоже модель, только она намного проще нашей модели дерева. А какими будут средние значения дохода, если мы извлечем повторные выборки и построим по ним модели? Мы повторно извлекли выборку и получили среднее значение дохода 84500. Из-за случайного отбора выборки будут отличаться друг от друга и мы будем получать разные средние значения дохода, но какими именно будут эти значения мы просто не знаем. Кроме того, мы не знаем, насколько мы можем доверять им. Да, различия обнаружены, но интуитивно понятно, что такой результат можно получить случайно, даже если в действительности (в генеральной совокупности) различий нет или, наоборот, когда эти различия существуют. На помощь нам приходит доверительный интервал.

Доверительным называют интервал, который накрывает истинное значение неизвестного параметра (μ) с заданной доверительной вероятностью. Доверительная вероятность (или уровень доверия) – это частота или доля всех возможных доверительных интервалов, которые накрывают истинное значение оцениваемого параметра. Например, если мы извлечем N случайных выборок одинакового размера из генеральной совокупности (при N , стремящимся к бесконечности) с аналогичным построением доверительного интервала, доля интервалов, накрывающих истинное значение неизвестного параметра (μ), будет совпадать с доверительной вероятностью.

Доверительную вероятность обозначают следующим образом: $(1-\alpha) \times 100\%$, где α – уровень значимости доверительного интервала. Таким образом, если мы зададим уровень значимости доверительного интервала 0.05, мы получим 95%-ную доверительную вероятность, т.е. $(1-0.05) \times 100\%=95\%$. Обычно устанавливают 95%-ную доверительную вероятность или 99%-ную доверительную вероятность. Для 95%-ной доверительной вероятности 95% доверительных интервалов будут накрывать истинное значение параметра, а 5% не будут накрывать его. Для 99%-ной доверительной вероятности 99% доверительных интервалов будут накрывать истинное значение параметра, а 1% не будут накрывать его.

Существуют два подхода к построению доверительных интервалов: точный и приближенный. Использование точного подхода требует выполнения ряда строгих предположений о модели и виде распределения данных, поэтому применяется редко. Приближенный подход основан на аппроксимации распределения исследуемого параметра. Он включает в себя асимптотический метод и бутстреп-метод.

Начнем с асимптотического метода. Идея асимптотического метода заключается в том, что границы доверительного интервала мы строим,

используя центральную предельную теорему. Согласно этой теореме, если из генеральной совокупности, имеющей распределение со средним μ и стандартным отклонением σ , многократно извлекать случайные выборки размером N , то при большом N (не менее 100 наблюдений) распределение выборочных средних будет стремиться к нормальному распределению со средним $\mu_{\bar{X}}$, равным среднему генеральной совокупности μ , и стандартным отклонением $\sigma_{\bar{X}}$, равным стандартному отклонению генеральной совокупности σ , поделенному на квадратный корень от объема выборки \sqrt{n} .

Итак, у нас есть \bar{X} , n , но мы не знаем μ . При этом нужно получить надежную оценку μ . Что делать? Начать с того, чтобы посмотреть на поведение \bar{X} . В этом нам поможет приведенная выше центральная предельная теорема.

Что из себя представляет \bar{X} ? Это выборочная статистика, это случайная величина, она может принимать *любое* значение, поскольку извлекаемые выборки случайны. Взглянем на рисунок ниже. Это распределение выборочных средних значений \bar{X} , где значения \bar{X} взяты из выборок размера n . Размер выборки одинаков для каждого \bar{X} , но наблюдения, по которым было вычислено \bar{X} , могут отличаться из-за случайного характера выборок. При этом помним, что согласно центральной предельной теореме распределение выборочных средних подчиняется нормальному распределению.

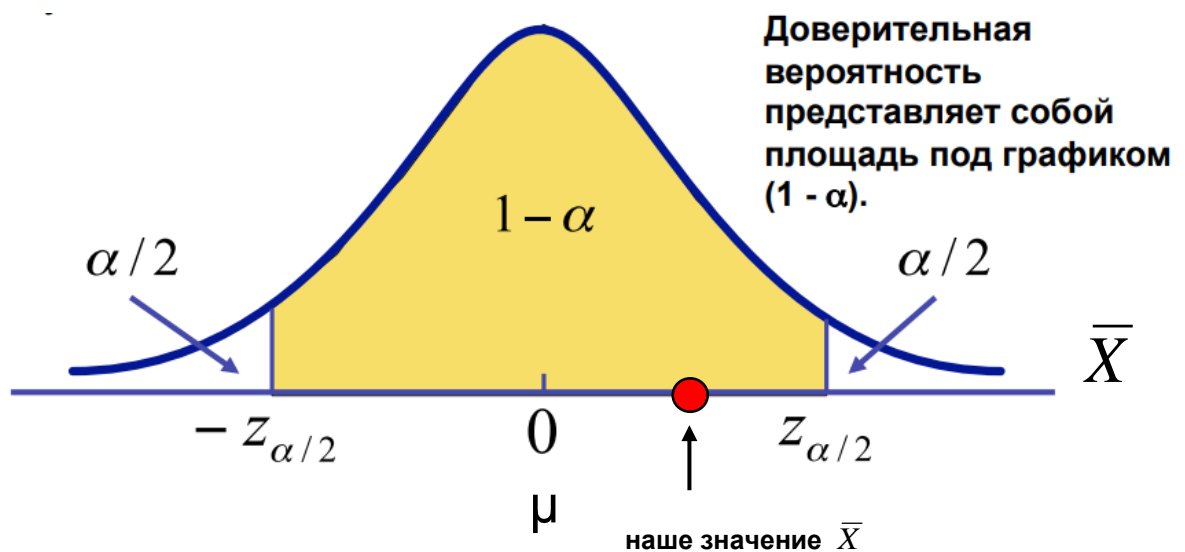


Рисунок 2.24. Кривая нормального распределения и доверительная вероятность

Площадь, которая ограничена кривой нормального распределения и границами интервала, отложенными по оси абсцисс, будет как раз

представлять собой доверительную вероятность $1-\alpha$. Площади, ограниченные кривой и хвостами распределения, выходящими за пределы доверительного интервала, будут равны $\alpha/2$.

К сожалению, нам редко бывает известен параметр генеральной совокупности μ (в нашем случае речь идет об истинном среднем значении дохода). На рисунке выше оно просто дано как опорная точка. Учитывая возможное расстояние между неизвестным параметром μ и нашим выборочным средним значением \bar{X} (см. рисунок), рискованно говорить, что конкретное значение \bar{X} равно μ . Правильнее сказать так:

наша точечная оценка	\pm	некоторый предел погрешности	накрывает	параметр генеральной совокупности μ
----------------------------	-------	------------------------------------	-----------	--

Что взять в качестве «некоторого предела погрешности»? Сначала нужно вычислить z-значение для соответствующей доверительной вероятности. Допустим, мы хотим вычислить z-значение для 95%-ной доверительной вероятности. Нас будет интересовать половина площади симметричной фигуры, ограниченной концами доверительного интервала и кривой нормального распределения, которая равна $0,95/2=0,475$.

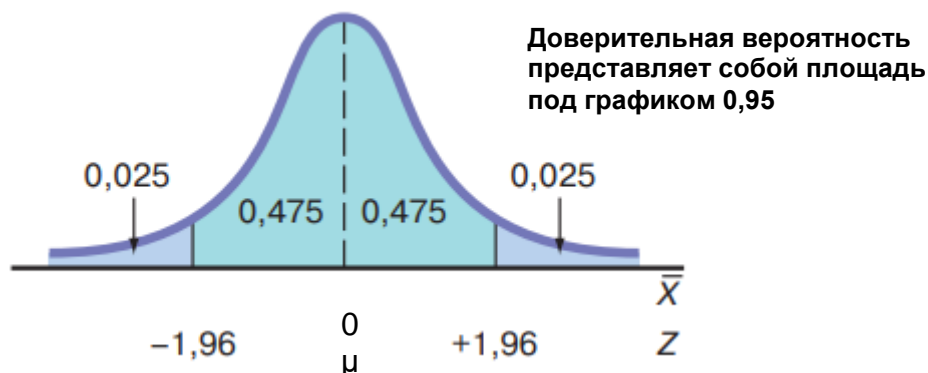


Рисунок 2.25. Кривая нормального распределения для определения критического значения Z, соответствующего 95%-ной доверительной вероятности

По таблице z-значений находим соответствующее z-значение для 0,475. Мы находим z-значение 1,96 на пересечении строки 1,9 и столбца 0,06. Таким образом, 95% площади под кривой нормального распределения будут лежать в пределах 1,96 стандартных отклонений от среднего.

x	0	1	2	3	4	5	6	7	8	9
1,8	4641	4648	4656	4664	4671	4678	4686	4692	4699	4706
1,9	4713	4719	4726	4732	4738	4744	4750	4756	4761	4767

Рисунок 2.26. Таблица z-значений для определения критического значения Z, соответствующего 95%-ной доверительной вероятности

После того, как z-значение вычислено, нужно учесть изменчивость данных и размер выборки, для этого умножаем z-значение на стандартное отклонение $\sigma_{\bar{x}}$, поделенное на квадратный корень от размера выборки \sqrt{n} . Результат деления стандартного отклонения на квадратный корень от размера выборки еще называют стандартной ошибкой (SE).

Таким образом, нашу формулировку

$$\begin{array}{ccccc} \text{наша} & & \text{некоторое} & & \text{параметр} \\ \text{точечная} & & \text{расстояние от} & & \text{генеральной} \\ \text{оценка} & \pm & \text{точечной оценки} & \text{накрывает} & \text{совокупности} \\ & & & & \mu \end{array}$$

можно свести к формуле:

$$\bar{X} \pm z_{\alpha/2} \times \frac{\sigma_{\bar{x}}}{\sqrt{n}}$$

Это и будет формулой доверительного интервала. Интервал, у которого левая граница – это выборочное среднее *минус* количество стандартных ошибок (z-значение), правая граница – это выборочное среднее *плюс* количество стандартных ошибок (z-значение), накрывает неизвестный параметр генеральной совокупности μ с определенной доверительной вероятностью.

Важной характеристикой доверительного интервала является его ширина. Ширина доверительного интервала позволяет судить о степени неопределенности истинного значения параметра. Чем шире интервал, тем больше неопределенность. Ширина доверительного интервала зависит от доверительной вероятности. Чем больше доверительная вероятность, тем шире будет доверительный интервал. Это обусловлено тем, что более высокой доверительной вероятности будет соответствовать большее z-значение и мы будем использовать большее количество стандартных отклонений.

На рисунке ниже показано, как увеличивается ширина доверительного интервала при возрастании z-значения и доверительной вероятности.

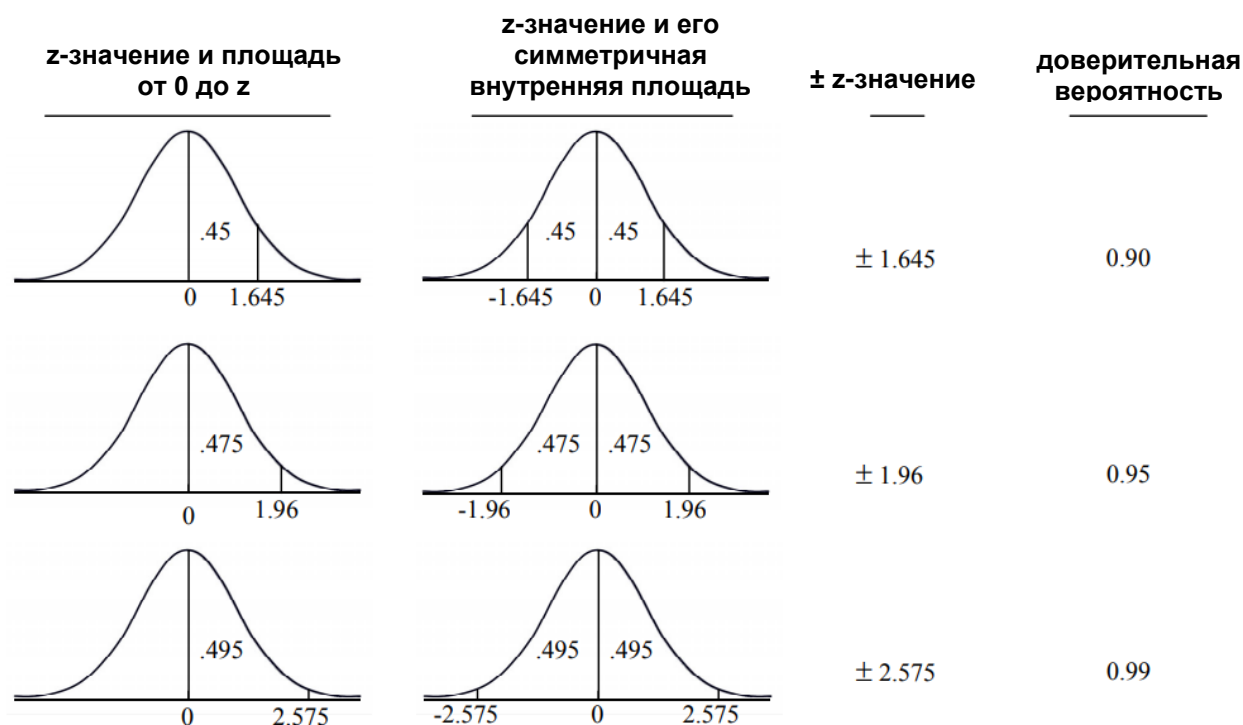


Рисунок 2.27. Ширина доверительного интервала и доверительная вероятность

Видно, что для вычисления 90%-ного доверительного интервала используется z-значение 1,645, для вычисления 95%-ного доверительного интервала – z-значение 1,96, а для вычисления 99%-ного доверительного интервала – z-значение 2,575.

С увеличением объема выборки ширина доверительного интервала, наоборот, будет уменьшаться. Именно поэтому разработка прогнозных моделей должна осуществляться на основе больших массивов данных.

Допустим, мы хотим построить 90%-ный доверительный интервал для среднего значения дохода (доход выражен в тысячах рублей).

Сгенерируем 50 случайных чисел в диапазоне от 45 до 100, которые будут нашими значениями дохода.

```
# генерируем 50 случайных чисел
# в диапазоне от 45 до 100
set.seed(42)
income <- runif(50, min=45, max=100)
income
```

Сводка 2.56. Значения дохода

```
[1] 95.31433 96.53915 60.73767 90.67462 80.29600 73.55028 85.51236 52.40666 81.13458 83.77856
[11] 70.17580 84.55117 96.40697 59.04859 70.42611 96.70080 98.80245 51.46180 71.12484 75.81830
[21] 94.72173 52.62906 99.38905 97.06675 49.53407 73.28165 66.46119 94.81560 69.58333 90.98023
[31] 85.56776 89.60803 66.34596 82.68434 45.21716 90.81038 45.40338 56.42124 94.86308 78.64783
[41] 65.87576 68.96744 47.05871 98.54470 68.74632 97.66671 93.82652 80.19883 98.40316 79.03610
```

Вычислим среднее значение дохода.

```
# вычисляем среднее значение дохода
mean <- mean(income)
mean
```

Сводка 2.57. Средний доход

```
[1] 77.93634
```

Запишем информацию о размере выборки в объект N.

```
# записываем информацию
# о размере выборки
N <- 50
```

Теперь умножаем z-значение на стандартное отклонение σ , поделенное на корень от общего количества наблюдений \sqrt{n} , чтобы вычислить предел погрешности. Обратите внимание, для 90%-ной доверительной вероятности z-значение равно 1,645. Если бы мы строили 95%-ный доверительный интервал, нам нужно было взять z-значение 1,96.

```
# вычисляем предел погрешности
егг <- 1.645*(sd(income)/sqrt(N))
```

Вычисляем нижнюю границу 90%-ного доверительного интервала.

```
# вычисляем нижнюю границу 90%-ного
# доверительного интервала
mean-егг
```

Сводка 2.58. Нижняя граница 90%-ного доверительного интервала

```
[1] 74.05113
```

Вычисляем верхнюю границу 90%-ного доверительного интервала.

```
# вычисляем верхнюю границу 90%-ного
# доверительного интервала
mean+егг
```

Сводка 2.59. Верхняя граница 90%-ного доверительного интервала

```
[1] 81.82156
```

Можно сделать вывод, что наш интервал 74,05 – 81,82 покрывает истинное значение дохода с доверительной вероятностью, приблизительно равной 90%, при условии асимптотической нормальности. При этом 90%-ная доверительная вероятность означает, что если мы повторно сформируем из генеральной совокупности одинаковые по объёму выборки и для каждой выборки вычислим доверительный интервал, то 90% доверительных интервалов будут покрывать истинное значение дохода, а 10% доверительных интервалов не будут покрывать его. Рисунок ниже поясняет этот вывод.

Очень часто встречается неправильная интерпретация доверительного интервала. Например, часто пишут, что для 95%-ного доверительного интервала существует вероятность 95%, что параметр генеральной совокупности попадает в него (или существует вероятность 95%, что интервал накрывает параметр генеральной совокупности). Однако интерпретацию 95%-ного доверительного интервала неверно сводить к такому вероятностному суждению. После извлечения выборки и вычисления интервала данный интервал либо охватывает значение параметра, либо нет, и это уже не является вопросом вероятности. Вероятность 95% относится не к определенному расчетному интервалу, а к надежности процедуры оценки. Нас просто интересует, сколько интервалов из построенных будут накрывать истинное значение параметра. Статистик Ежи Нейман, разработавший метод доверительных интервалов, по этому поводу писал в своей оригинальной статье: «Следует заметить, что в приведенном выше описании вероятностное суждение относится к проблемам оценки, которые будут заботить статистика в будущем. На самом деле я уже неоднократно заявлял, что частота правильных результатов будет стремиться к α . Возьмем ситуацию, когда выборка уже извлечена и получены результаты [конкретные границы]. Можем ли мы сказать, что в данном конкретном случае вероятность истинного значения [попадающего в эти границы] равна α ? Очевидно, что ответ будет отрицательным. Параметр – это неизвестная константа и нельзя вынести никакого вероятностного суждения, касающегося его значения».

Доверительный интервал AUC можно также вычислить с помощью бутстреп-метода. Ключевая идея бутстрепа заключается в получении большого количества случайных наборов данных из исследуемой выборки. В нашем случае мы получили оценку AUC. Как убедиться в том, что она является надежной? Основная идея такова: хорошо бы повторить наш эксперимент много раз и посмотреть на распределение результатов. Однако откуда взять столько выборок? Способ есть. Вместо того, чтобы делать повторные эксперименты, мы на основе одной имеющейся выборки генерируем множество псевдовыборок того же размера и будем считать, что они – результат повторения нашего эксперимента.

Допустим у нас есть исходная выборка, состоящая из 10 наблюдений [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Из этой выборки объемом 10 наблюдений мы случайным образом выбираем наблюдения с возвращением 10 раз. Поскольку отбор с возвращением, то одно и то же наблюдение может быть выбрано несколько раз, а некоторые наблюдения вообще не будут выбраны. Мы получаем бутстреп-выборку, которая имеет такой же размер, что и исходная выборка. Возможная бутстреп-выборка может выглядеть как [10, 9, 7, 8, 1, 3, 9, 10, 10, 7]. Считаем интересующую нас

оценку среднего дохода по этой новой выборке и запоминаем результат. Процедуру генерирования бутстреп-выборки и вычисления по ней оценки среднего дохода повторяем много раз. На практике обычно используется 10000 бутстреп-выборок. Меньшее количество дает более низкое качество оценки. Большее количество увеличивает время вычислений.

Для наглядности описанный алгоритм бутстреп-выборок приводится на рисунке 2.24.

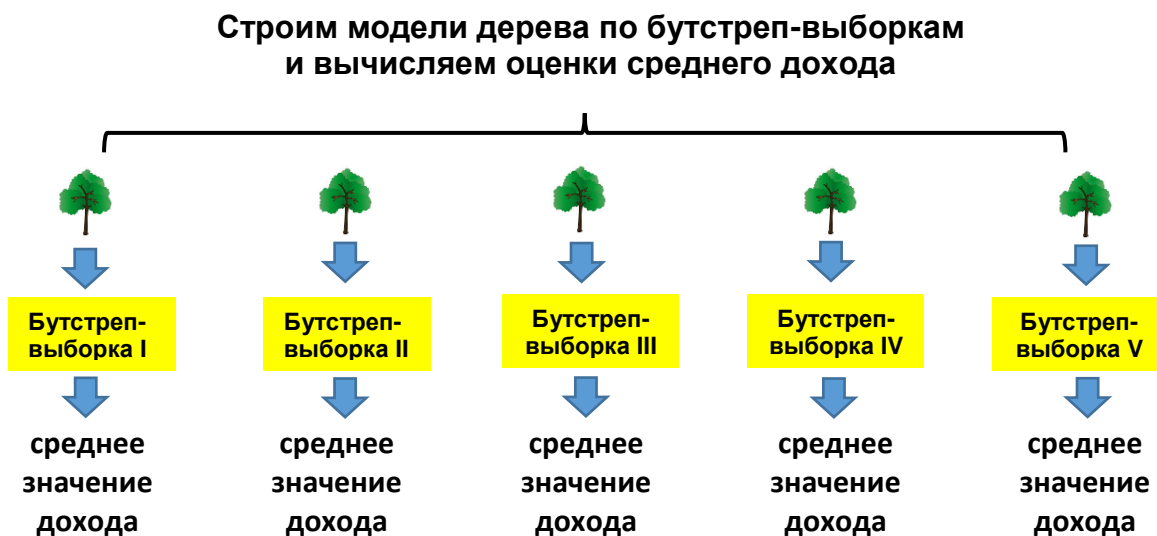


Рисунок 2.28 Механизм работы бутстреп-выборок

Получив множество оценок среднего дохода, можно построить эмпирическую функцию распределения оценок, далее мы берем 2.5 и 97.5 процентиля и получаем 95%-ный доверительный интервал.

Давайте построим 95%-ный доверительный интервал для нашего игрушечного примера, воспользовавшись бутстреп-методом. Сначала необходимо получить распределение средних значений дохода, вычисленных по бутстреп-выборкам (в данном случае используется 5000 бутстреп-выборках).

```
# генерируем данные
set.seed(42)
income <- runif(50, min=45, max=100)
# задаем функцию, вычисляющую среднее
m <- function(x) mean(x)
# создаем вектор, в который будем
# записывать средние значения,
# вычисляемые по бутстреп-выборкам
boot <- numeric(5000)
# создаем 5000 бутстреп-выборок и вычисляем
# 5000 средних значений дохода
for (i in 1:5000) boot[i] <- m(sample(income, replace=T))
```

Получив распределение средних значений дохода, смотрим значения, соответствующие квантилям 0.025 и 0.975 для получения нижней и верхней границ 95%-ного доверительного интервала.

Сводка 2.60. Нижняя граница 95%-ного доверительного интервала, вычисленного с помощью бутстреп-метода

```
2.5%
73.28814
```

Сводка 2.61. Верхняя граница 95%-ного доверительного интервала, вычисленного с помощью бутстреп-метода

```
97.5%
82.35124
```

Можно вычислить доверительный интервал несколько иным способом.

```
# генерируем данные
set.seed(42)
income <- runif(50, min=45, max=100)
# задаем количество бутстреп-выборок (5000)
B = 5000
# задаем размер бутстреп-выборки,
# тот же, что и размер исходной
# (50 наблюдений)
n = 50
# создаем матрицу, у которой строки - бутстреп-выборки (5000),
# а столбцы - наблюдения, отобранные в бутстреп-выборку с
# возвращением (50)
boot.samples = matrix(sample(income, size = B * n, replace = TRUE), B, n)
# применяем с помощью функции apply нашу функцию mean к каждой
# строке матрицы boot.samples (1 означает строки, 2 - столбцы)
boot.statistics = apply(boot.samples, 1, function(x) mean(x))
# вычисляем нижнюю границу 95%-ного
# доверительного интервала
```



```
quantile(boot,0.025)
# вычисляем верхнюю границу 95%-ного
# доверительного интервала
quantile(boot,0.975)
```

Сводка 2.62. Нижняя граница 95%-ного доверительного интервала, вычисленного с помощью бутстреп-метода

```
2.5%
73.2881386375835
```

Сводка 2.63. Верхняя граница 95%-ного доверительного интервала, вычисленного с помощью бутстреп-метода

```
97.5%
82.3512427501578
```

От доверительного интервала среднего перейдем к доверительному интервалу AUC. В пакете **rROC** реализованы асимптотический и бутстреп-метод вычисления доверительного интервала AUC, по умолчанию для оценки доверительного интервала используется асимптотический метод. В рамках асимптотического метода для вычисления доверительного интервала AUC используют следующую формулу:

$$AUC \pm z \times SE(AUC)$$

где:

z – z -значение для соответствующей доверительной вероятности (95%-ной доверительной вероятности соответствует z -значение 1,96);

$SE(AUC)$ – стандартная ошибка AUC.

Для расчета стандартной ошибки AUC обычно используют формулу, предложенную Хенли и Макнейлом.

$$SE(AUC) = \sqrt{\frac{AUC(1 - AUC) + (N_+ - 1)(Q_1 - AUC^2) + (N_- - 1)(Q_2 - AUC^2)}{N_+ N_-}}$$

где

N_- – количество наблюдений в отрицательном классе;

N_+ – количество наблюдений в положительном классе;

$$Q_1 = \frac{AUC}{2 - AUC};$$

$$Q_2 = \frac{2AUC^2}{1 + AUC}.$$

Кроме того, для вычисления стандартной ошибки AUC используется метод Делонга.

Допустим, чем выше спрогнозированный балл, тем выше принадлежность к положительному классу. Пусть $\hat{\theta}$ – это вычисленное значение AUC. $X_i, i=1,2,...,N_+$ и $Y_j, j=1,2,...,N_-$ – это спрогнозированные баллы для наблюдений положительного и отрицательного классов соответственно.

Для каждого наблюдения положительного класса i определяем

$$V_{10}(X_i) = \frac{1}{N_-} \sum_{j=1}^{N_-} \varphi(X_i, Y_j)$$

и для каждого наблюдения отрицательного класса j вычисляем

$$V_{01}(Y_j) = \frac{1}{N_+} \sum_{i=1}^{N_+} \varphi(X_i, Y_j)$$

где

$$\varphi(X_i, Y_j) = \begin{cases} 1 & Y < X \\ \frac{1}{2} & Y = X \\ 0 & Y > X \end{cases}$$

Затем вычисляем

$$S_{10} = \frac{1}{N_+ - 1} \sum_{i=1}^{N_+} \{V_{10}(X_i) - \hat{\theta}\}^2$$

и

$$S_{01} = \frac{1}{N_- - 1} \sum_{j=1}^{N_-} \{V_{01}(Y_j) - \hat{\theta}\}^2$$

В итоге стандартная ошибка AUC вычисляется по формуле:

$$SE(AUC) = \sqrt{\frac{1}{N_+} S_{10} + \frac{1}{N_-} S_{01}}$$

Давайте вычислим доверительный интервал AUC нашей модели для контрольной выборки.

```
# вычисляем 95%-ный доверительный интервал AUC
# по асимптотическому методу Делонга (по умолчанию)
# для модели дерева на контрольной выборке
roc_hold <- roc(holdout$churn, prob_hold[,2], ci=TRUE)
roc_hold
```

Сводка 2.64. 95%-ный доверительный интервал AUC для модели дерева CHAID на контрольной выборке: асимптотический метод

```
Call:
roc.default(response = holdout$churn, predictor = prob_hold[, 2], ci = TRUE)

Data: prob_hold[, 2] in 746 controls (holdout$churn 0 остается) < 598 cases (holdout$churn
Уходит).
Area under the curve: 0.7875
95% CI: 0.7637-0.8112 (DeLong)
```

По умолчанию в пакете **rROC** вычисляется 95%-ный доверительный интервал AUC и используется асимптотический метод Делонга.

В нашем случае 95%-ный доверительный интервал AUC, вычисленный по асимптотическому методу, показывает, что доверительный интервал AUC 0.76-0.81 накрывает истинное значение AUC с доверительной вероятностью, приблизительно равной 95%, при условии асимптотической нормальности. При этом 95%-ная доверительная вероятность означает, что если мы повторно сформируем из генеральной совокупности одинаковые по объёму выборки и для каждой выборки вычислим доверительный интервал, то 95% доверительных интервалов будут накрывать истинное значение AUC, а 5% доверительных интервалов не будут накрывать его.

Теперь вычислим 95%-ный доверительный интервал AUC с помощью бутстрепа (по умолчанию генерируются 2000 бутстреп-выборок). Поскольку бутстреп использует рандомизацию, для получения воспроизводимых результатов зададим стартовое значение генератора случайных чисел.

```
# вычисляем 95%-ный доверительный интервал AUC
# по бутстреп-методу (по умолчанию 2000 бутстреп-
# выборок) для модели дерева на контрольной выборке
roc_hold <- roc(holdout$churn, prob_hold[,2], ci=TRUE)
roc_hold
set.seed(14)
ci.auc(roc_hold, method="bootstrap")
```

Сводка 2.65. 95%-ный доверительный интервал AUC для модели дерева CHAID на контрольной выборке: бутстреп-метод (2000 бутстреп-выборок)

```
95% CI: 0.763-0.8111 (2000 stratified bootstrap replicates)
```

На практике выбранное по умолчанию количество бутстреп-выборок (2000) часто дает оптимистичные оценки доверительных интервалов. С помощью параметра **boot.n** можно задать нужное количество бутстреп-выборок. Большее количество бутстреп-выборок позволяет получить более точные оценки доверительных интервалов, но занимает большее

время вычислений. Как правило, при использовании большего количества бутстреп-выборок доверительные интервалы становятся более широкими. Обычно задают около 10000 бутстреп-выборок, в нашем случае мы для экономии времени вычислений зададим 5000 бутстреп-выборок.

```
set.seed(14)
ci.auc(roc_hold, method="bootstrap", boot.n=5000)
```

Сводка 2.66. 95%-ный доверительный интервал AUC для модели дерева CHAID на контрольной выборке: бутстреп-метод (5000 бутстреп-выборок)

95% CI: 0.7632-0.811 (5000 stratified bootstrap replicates)

Обратите внимание, что по умолчанию используется стратифицированный бутстреп. Если простой бутстреп генерирует выборки с учетом равной вероятности появления каждого объекта, то стратифицированный бутстреп учитывает соотношение частот между относительно гомогенными группами (стратами), на которые могут быть разделены выборочные объекты. Чтобы вычислить доверительный интервал с помощью нестратифицированного бутстрепа, необходимо задать параметр `boot.stratified` и установить значение `FALSE`.