

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (ПГУ)**

С. В. Рындина

**БИЗНЕС-АНАЛИТИКА
НА ОСНОВЕ БОЛЬШИХ ДАННЫХ:
ОБУЧЕНИЕ С УЧИТЕЛЕМ
НА ЯЗЫКАХ PYTHON И R**

Учебно-методическое пособие

Пенза
Издательство ПГУ
2020

УДК 004.6 (075)
Р95

Р е ц е н з е н т

кандидат технических наук, доцент
А. А. Масленников

Рындина, С. В.

Р95 Бизнес-аналитика на основе больших данных: обучение с учителем на языках Python и R : учеб.-метод. пособие / С. В. Рындина. – Пенза : Изд-во ПГУ, 2020. – 80 с.

Рассмотрены возможности языков программирования Python и R для реализации методов обучения с учителем: деревьев решений, метода kNN (метода поиска ближайших соседей), логистической регрессии. Приведены задания для лабораторных работ по методам обучения с учителем.

Материал пособия соответствует программе дисциплины «Бизнес-аналитика на основе больших данных».

Издание подготовлено на кафедре «Цифровая экономика» Пензенского государственного университета и предназначено для обучающихся по направлению подготовки 38.03.05 «Бизнес-информатика» при изучении дисциплины «Бизнес-прогнозирование» и написании выпускной работы.

УДК 004.6 (075)

Содержание

Введение	5
1. Обучение с учителем	6
1.1. Дерево решений	6
1.2. Метод ближайших соседей	17
1.3. Линейные модели классификации и регрессии.....	19
2. Обучение с учителем на языке R, RStudio	24
2.1. Деревья решений.....	24
2.1.1. Набор данных оператора связи (дерево классификации).....	31
2.1.2. Набор данных <i>медицинское страхование</i> (дерево регрессии).....	33
2.2. Метод ближайших соседей	34
2.2.1. Набор данных оператора связи (классификация).....	37
2.2.2. Набор данных <i>медицинское</i> <i>страхование</i> (регрессия).....	38
2.3. Линейная регрессия	39
2.3.1. Набор данных <i>медицинское страхование</i> (парная линейная регрессия)	41
2.3.2. Набор данных <i>медицинское страхование</i> (множественная линейная регрессия).....	42
2.4. Линейный классификатор (логистическая регрессия).....	46
3. Обучение с учителем на языке Python, Anaconda, Jupyter Notebook	49
3.1. Деревья решений.....	49
3.1.1. Набор данных оператора связи (дерево классификации).....	51
3.1.2. Набор данных <i>медицинское страхование</i> (дерево регрессии).....	58

3.2. Метод ближайших соседей	60
3.2.1. Набор данных оператора связи (классификация)	62
3.2.2. Набор данных <i>медицинское</i> <i>страхование</i> (регрессия)	64
3.3. Линейная регрессия	65
3.4. Линейный классификатор (логистическая регрессия)	69
4. Лабораторные работы	74
4.1. Лабораторная работа «Деревья решений»	74
4.2. Лабораторная работа «Метод ближайших соседей»	75
4.3. Лабораторная работа «Линейная регрессия»	76
4.4. Лабораторная работа «Линейный классификатор (логистическая регрессия)»	77
Библиографический список	78

Введение

В соответствии с учебным планом студенты третьего курса направления подготовки 38.03.05 «Бизнес-информатика» изучают дисциплину «Бизнес-аналитика на основе больших данных».

Обучение с учителем – методы выявления в размеченных данных зависимостей, которые позволяют на основании признаков объекта предсказать (спрогнозировать) значение метки (ответа, результата).

Возможность использовать выявленные зависимости для предсказания значений меток для новых объектов, для которых известны описания признаков, но неизвестны метки, т.е. уже на неразмеченных данных, является чрезвычайно важной для повышения отдачи от бизнеса: настройки цифровых продуктов, оптимизации бизнес-процессов и т.п.

Целью учебно-методического пособия является рассмотрение методов обучения с учителем и их реализация на двух языках R и Python. Языки программирования R и Python – основные инструменты для Data science (науки о данных). Программирование на этих языках для реализации алгоритмов анализа данных – фундамент для аналитиков данных, специалистов в области Data science, специалистов в области технологий Big Data.

В пособии использованы материалы:

Открытый курс по машинному обучению от Open Data Science [1, 2].

Открытый курс «Введение в Data Science и машинное обучение» от Анатолия Карпова [3].

Книга «Прогнозное моделирование в IBM SPSS Statistics, R и Python: метод деревьев решений и случайный лес» – автор А. В. Груздев [4].

Книга «Введение в статистическое обучение» – авторы: Е. Джеймс, Д. Уиттон, Т. Хасты, Р. В. Тибширани [5].

Книга «Построение систем машинного обучения» – автор Луис Педро Коэльо [6].

Книга «Python и машинное обучение» – автор Себасьян Рашка [7].

Книга «Классификация, регрессия и другие алгоритмы Data Mining» – авторы: В. К. Шитиков, С. Э. Мастицкий [8].

1. Обучение с учителем

Обучение с учителем (Supervised Learning) – группа методов анализа данных для восстановления зависимостей на основе обучающей выборки. Обучающая выборка, каждый элемент которой – это вектор независимых переменных (признаков, атрибутов и т.п.), описывающих объект, и целевое значение для объекта – зависимая переменная (результат, отклик), и есть «учитель».

Обучение с учителем позволяет на основе размеченных (маркированных) тренировочных данных построить модель, которая будет делать прогнозы для новых объектов (для новых прецедентов, случаев) на основе известных о них неразмеченных данных (рис. 1.1).



Рис. 1.1. Обучение с учителем

Основные группы методов обучения с учителем – это классификация, когда для тренировочных данных об объектах известна метка принадлежности к какому-либо классу: для заемщиков известно, был ли возвращен кредит; для абонентов сотового оператора известно, ушел или остался; для подписчиков – перешел ли на более дорогой тариф, на более бюджетный или остался на текущем и т.п., и регрессия, когда для тренировочных данных известна выходная (результатирующая, целевая) числовая величина (целого или вещественного типа): объем потребляемого абонентом трафика, продолжительность жизненного цикла клиента и т.п.

1.1. Дерево решений

Материалы: [1; 3, тема 2: Решающие деревья; 4, с. 14–22; 5, с. 328–363; 8, тема 6.4].

Деревья решений используются для моделирования взаимосвязи зависимой переменной от нескольких независимых переменных

(предикторов). Если зависимая переменная (отклик, результат) является меткой класса объекта, т.е. категориальной переменной (например, ушел абонент от сотового оператора или остался), то построенное дерево будет деревом классификации. Если зависимая переменная – количественная (например, чек клиента), то построенное разбиение (сегментация) будет деревом регрессии.

При построении дерева, начиная с корневого узла, определяются правила разбиения или сегментации. Для построения правила выбирается независимая переменная, которая позволяет разделить наблюдения на сегменты, максимально отличающиеся по значению зависимой переменной. На ветвях, отходящих от корневого узла, пишутся правила (статистики), на основании которых произошло разбиение наблюдений, для получившихся узлов определяется среднее значение количественной переменной (дерево регрессии) или вероятность отнесения к определенной категории (дерево классификации).

Рассмотрим процедуру построения дерева решений для классификации. Дерево классификации используется для предсказания отклика y , который является категориальной переменной (меткой класса).

1. Среди предикторов $X_1, X_2, \dots, X_k$ для правила разбиения выбирается X_p , значения которого позволяют разделить наблюдения по отклику y наилучшим образом (чтобы каждое из двух подмножеств было однородным по значению y внутри подмножества и различным по отношению друг к другу). Для категориальной переменной X_p в качестве условия разделения выбирается равенство одному из возможных значений X_p в выборке. Для количественной переменной условием является неравенство $X_p \leq l$, где l – некоторое значение на интервале от минимального значения X_p до максимального значения в выборке наблюдений.

2. Наблюдения в левом узле такие, для которых X_p условие – истина (true), а в правом – все остальные, т.е. наблюдения, для которых условие разбиения ложно (false).

3. Для каждого из получившихся при разбиении узлов вычисляются процентные доли значений зависимой переменной и выносятся решение: к какой категории будет принадлежать попавшее в нее наблюдение.

4. Если в узле достигнута однородность (т.е. все наблюдения, оказавшиеся в узле, из одного класса), то для этого узла процедура закончена.

5. Если в узле оказалось множество наблюдений из разных классов, то процесс бинарного разбиения для этого узла может быть продолжен с пункта 1 (в дальнейшем разбиении для текущего узла учитываются только наблюдения, попавшие в этот узел).

В качестве статистического критерия, оценивающего качество разбиения, используется индекс Джини или альтернативный вариант – энтропия.

Индекс Джини (**Gini impurity**):

$$I_G = 1 - \sum_{i=1}^m p_i^2, \quad (1.1)$$

где p_i^2 – частоты представителей разных классов в узле дерева; m – число классов для отклика y .

Энтропия Шеннона:

$$S = - \sum_{i=1}^m p_i \log_2 p_i, \quad (1.2)$$

где p_i – частоты представителей разных классов в узле дерева; m – число классов для отклика y .

Рассмотрим процедуру построения дерева решений для регрессии. Дерево регрессии используется для предсказания отклика y , который является количественной переменной. В этом случае исходное множество разбивается на сегменты, которым соответствует некоторое среднее значение отклика.

1. Среди предикторов $X_1, X_2, \dots, X_k$ рассматриваются различные варианты X_p для расщепления дерева. Для категориальной переменной X_p в качестве условия разделения выбирается равенство одному из возможных значений X_p в выборке. Для количественной переменной условием является неравенство $X_p \leq l$, где l – некоторое значение на интервале от минимального значения X_p до максимального значения в выборке наблюдений.

2. Наблюдения в левом узле те, для которых X_p условие – истина (true), а в правом все остальные, т.е. наблюдения, для которых условие разбиения ложно (false).

3. Для каждого из получившихся при разбиении узлов вычисляются средние значения отклика y .

4. В качестве признака для расщепления дерева выбирается тот предиктор, для которого разбиение дает лучшее значение MSE (mean squared error):

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad (1.3)$$

где y_i – отклик для наблюдения с номером i ; $\hat{y}_i$ – значение отклика, рассчитанное по модели.

Так как при расщеплении по значениям предиктора X_p получили две области $R_1(p, l)$ (условие $X_p \leq l$ – истина) и $R_2(p, l)$ (условие $X_p \leq l$ – ложь), то

$$MSE = \frac{1}{n_1} \sum_{i: x_i \in R_1} (y_i - \hat{y}_{R_1})^2 + \frac{1}{n_2} \sum_{i: x_i \in R_2} (y_i - \hat{y}_{R_2})^2, \quad (1.4)$$

где n_1 – количество наблюдений в области $R_1(p, l)$; $\hat{y}_{R_1}$ – среднее значение отклика в области $R_1(p, l)$; n_2 – количество наблюдений в области $R_2(p, l)$; $\hat{y}_{R_2}$ – среднее значение отклика в области $R_2(p, l)$.

5. В качестве предиктора для правила разбиения из возможных вариантов выбирается тот предиктор, для которого значение MSE – наименьшее. При необходимости процесс повторяется для новых узлов.

Деревья решений (если не ограничивать глубину и не вводить иные ограничения) могут давать излишне детализированную картину, когда в листьях сосредоточено мало наблюдений. Для дерева регрессии MSE для какого-либо узла равно нулю, если все наблюдения имеют одно и то же значение отклика. Если в выборке все значения отклика различны, то количество листьев в таком дереве совпадет с числом наблюдений (переобучение). Для дерева классификации также возможна ситуация, когда в конечных узлах (листьях) оказывается слишком мало наблюдений по сравнению с исходным объемом выборки.

Такая ситуация называется переобучением. Дерево решений строится с целью определять по значениям предикатов прогноз от-

клика для нового наблюдения. Переобученное дерево дает минимальную ошибку на обучающей (тренировочной) выборке и очень часто ошибается на проверочной (тестовой). Соответственно, прогноз на переобученном дереве проигрывает прогнозу на основе дерева, в котором решена проблема переобучения тем или иным способом: за счет ограничений на глубину дерева, на количество наблюдений в узле, для которого возможно расщепление и т.п., либо строится максимально возможное дерево, а затем на основании определенных критериев отсекаются лишние ветки (убирается не-существенная детализация).

Рассмотрим имитированные данные продаж магазина (табл. 1.1), где *gender* (пол) – категориальная переменная, принимающая два возможных значения: ‘f’ – женский и ‘m’ – мужской; *age* (возраст) – количественная переменная; *totalsum* (сумма покупки) – количественная переменная; *discount* (скидка) – категориальная переменная, принимающая два возможных значения: ‘y’ – приобретались товары по акции, ‘n’ – товары приобретены за полную стоимость.

Таблица 1.1

Набор имитированных данных для магазина

<i>N</i>	<i>gender</i>	<i>age</i>	<i>totalsum</i>	<i>discount</i>
1	f	32	25412	y
2	f	25	31258	y
3	f	47	17896	n
4	f	29	62300	y
5	f	38	27366	y
6	m	49	45985	n
7	m	32	52122	n
8	m	51	31645	n
9	m	28	12489	y
10	m	35	23567	n

Визуализируем выборку (рис. 1.2), где 0 – числовая метка для значения переменной *gender* – ‘f’, а 1 – соответственно числовая метка значения ‘m’. Размер маркера на графике отвечает переменной *discount*: круг большего размера соответствует покупке товаров за полную стоимость, меньшего – покупке товаров со скидкой.

Построим дерево решений для переменной *discount* (рис. 1.3).

В начальном узле (корне) 10 наблюдений: пять, соответствующих любителям акций, и пять, соответствующих оплатившим полную стоимость товаров. Энтропия рассчитана по формуле (1.2):

$$S = -\frac{5}{10} \log_2 \frac{5}{10} - \frac{5}{10} \log_2 \frac{5}{10} = 1.$$

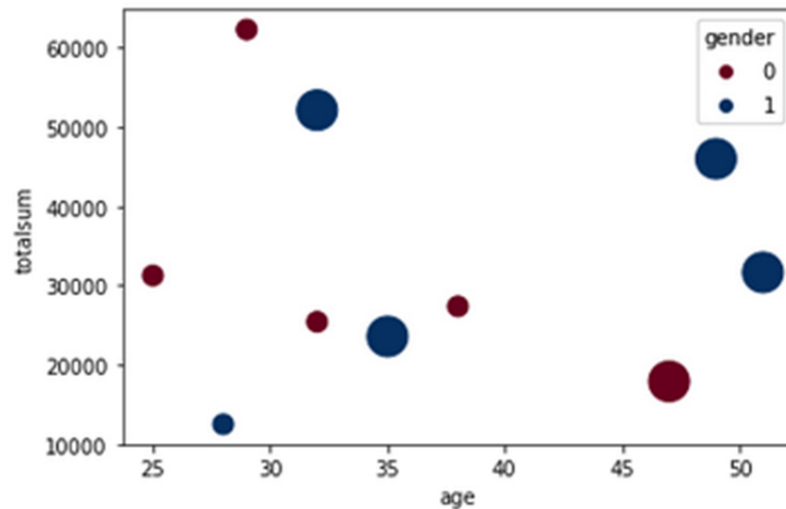


Рис. 1.2. Зависимость суммы покупки от возраста (с учетом пола и восприимчивости к акциям)

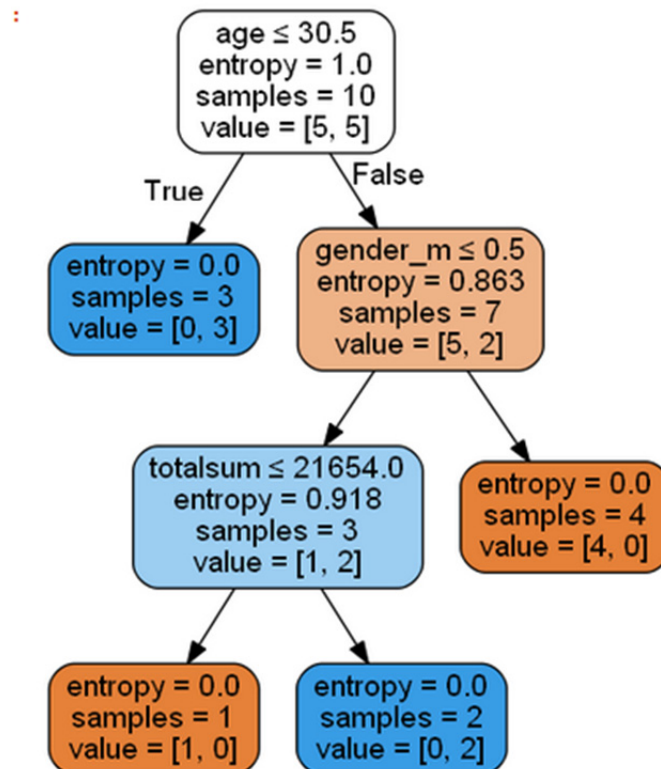


Рис. 1.3. Дерево решений для набора данных (табл. 1.1): отклик – discount

Разбиение осуществляется по количественному показателю age (значение для разбиения – 30.5). Левому узлу соответствует условие=true, правому – условие=false. Все наблюдения первого узла соответствуют классу: discount='y' (или числовой метке 1). Для такого узла энтропия равна 0. В правом узле – семь наблюдений, из которых пять относятся к отклику discount='n' (или числовой метке 0), а два наблюдения – к классу discount='y' (или числовой метке 1). Визуализация разбиения показана на рис. 1.4. Энтропия рассчитана по формуле (1.2)

$$S = -\frac{5}{7}\log_2 \frac{5}{7} - \frac{2}{7}\log_2 \frac{2}{7} = 0.863.$$

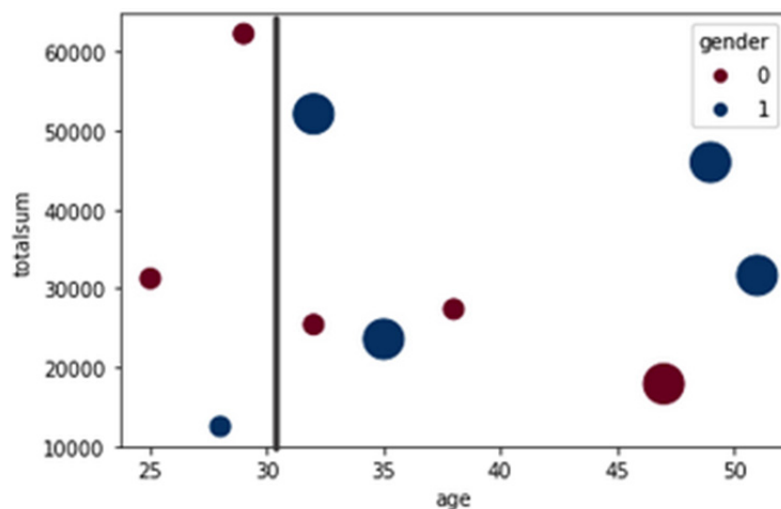


Рис. 1.4. Первое расщепление дерева решений по переменной age

В правом узле возможно дальнейшее разбиение по предиктору gender, который является категориальной переменной с двумя значениями: 0 – ‘f’ и 1 – ‘m’. Условие $gender_m \leq 0.5$ соответствует классу с меткой 0, т.е. женщинам. В правый узел попали четверо мужчин, и все они покупают товары за полную стоимость. Энтропия этого узла равна нулю.

В левый узел попали женщины, среди которых одна склонна покупать за полную стоимость, а две другие делают покупки по акциям. Энтропия рассчитана по формуле (1.2)

$$S = -\frac{1}{3}\log_2 \frac{1}{3} - \frac{2}{3}\log_2 \frac{2}{3} = 0.918.$$

В этом узле возможно дальнейшее разбиение, и оно проведено по предиктору totalsum, который является количественной перемен-

ной (значение для разбиения – 21654). В листьях, исходящих из этого узла, энтропия равна нулю, наблюдения однородны по значению отклика (рис. 1.5).

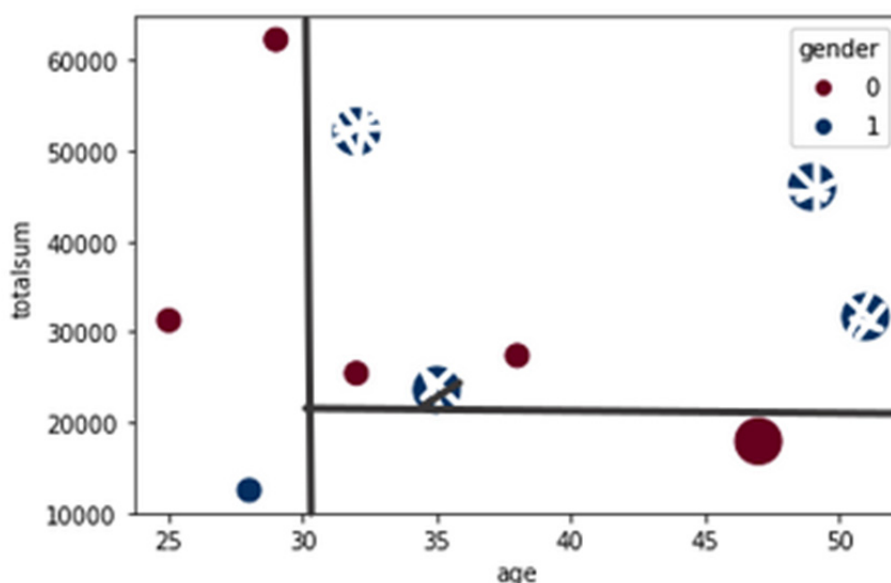


Рис. 1.5. Третье расщепление дерева решений по переменной totalsum

Выводы, которые можно сделать на основании этого дерева: покупатели до 30 лет более склонны к покупкам по акциям; мужчины старше 30 лет покупают товары за полную стоимость.

На тех же данных построим дерево регрессии. На роль отклика выберем переменную totalsum (рис 1.6).

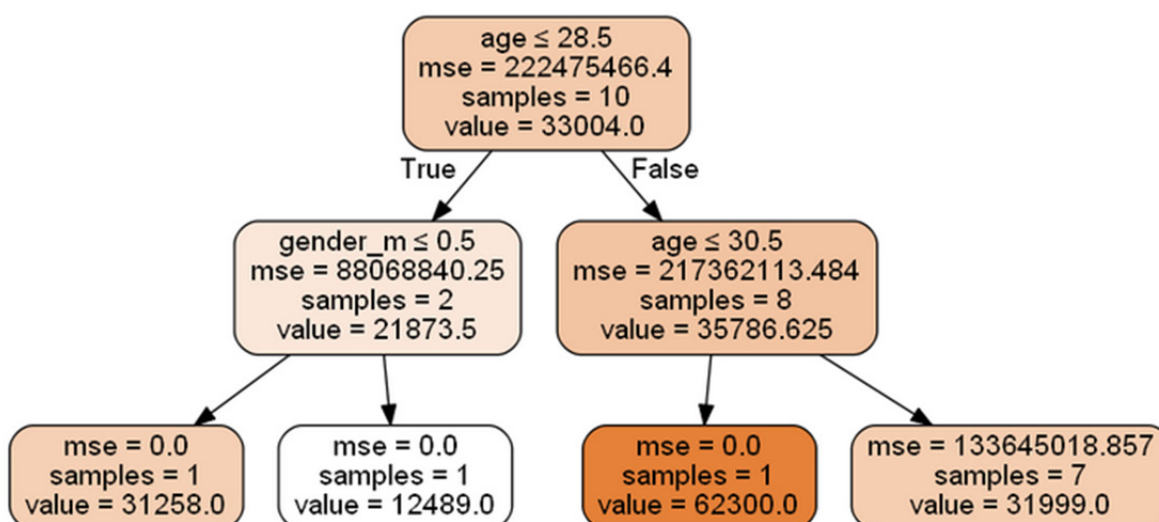


Рис. 1.6. Дерево решений для набора данных (табл. 1.1): отклик – totalsum

В начальном узле среднее значение стоимости покупки равно 33004. Значение MSE, рассчитанное по формуле (1.3), составляет 222475466.4, так как для всех наблюдений модельное значение – это среднее значение стоимости покупки. В начальном узле расщепление происходит по количественному показателю age (значение для разбиения – 28.5). Левому узлу соответствует условие=true, правому – условие=false (рис. 1.7).

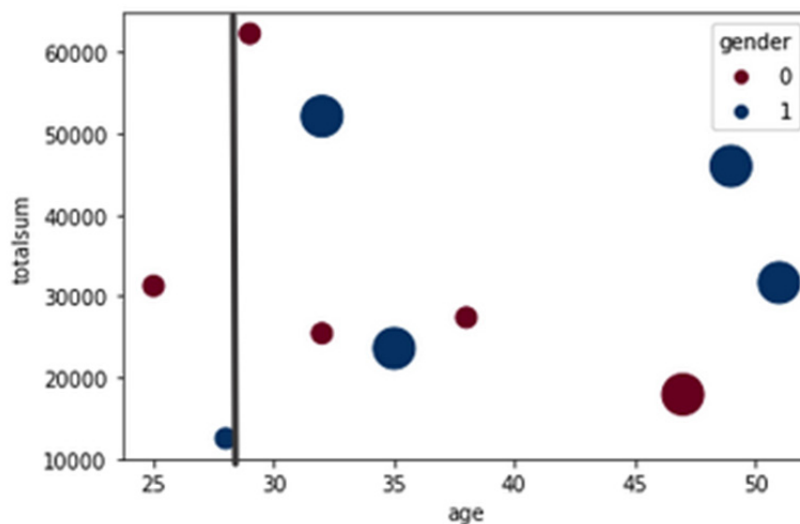


Рис. 1.7. Первое расщепление дерева решений по переменной age

Второе расщепление для левого узла, в который попало всего два наблюдения, будет по категориальной переменной gender. Значение $gender_m$ для левого узла равно нулю и отвечает метке класса 'f' – женщина, а значение $gender_m$ для правого узла равно единице и отвечает метке класса 'm' – мужчина. Так как в каждом из конечных узлов оказалось по одному наблюдению, то среднее значение отклика totalsum для этих узлов совпадает с наблюдаемым значением и MSE принимает значение, равное нулю (наименьшее из возможных).

В правом узле происходит повторное разбиение по количественному показателю age (значение для разбиения – 30.5). Это разбиение изолирует в левом узле покупателя с суммой покупки 62300. В правом узле размещено семь наблюдений, для которых среднее значение стоимости покупки (средний чек) – 31999, а значение MSE, рассчитанное по формуле (1.3), составляет 133645018.857.

Дать экономическую интерпретацию такому дереву несколько сложнее, каких-то явно различимых по признакам и представительных групп наблюдений дерево регрессии не выявило.

Изменение выборки на одно наблюдение (например, исключим покупателя под номером четыре) приведет к совершенно другому разбиению (рис. 1.8 и 1.9).

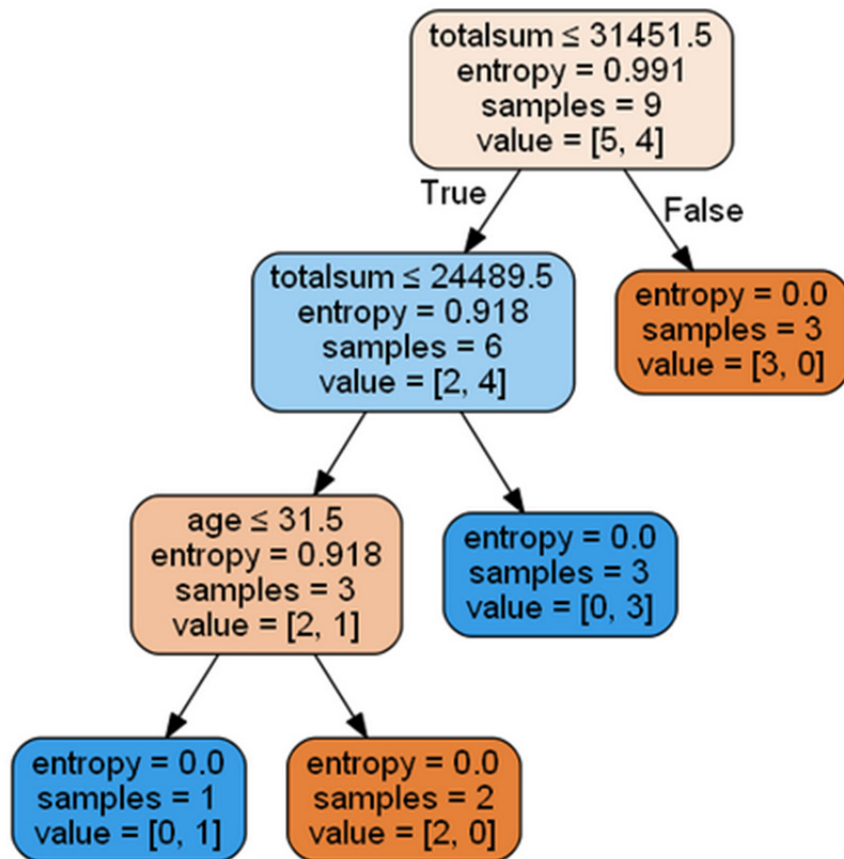


Рис. 1.8. Дерево решений для набора данных (табл. 1.1) с исключенным четвертым наблюдением (отклик – discount)

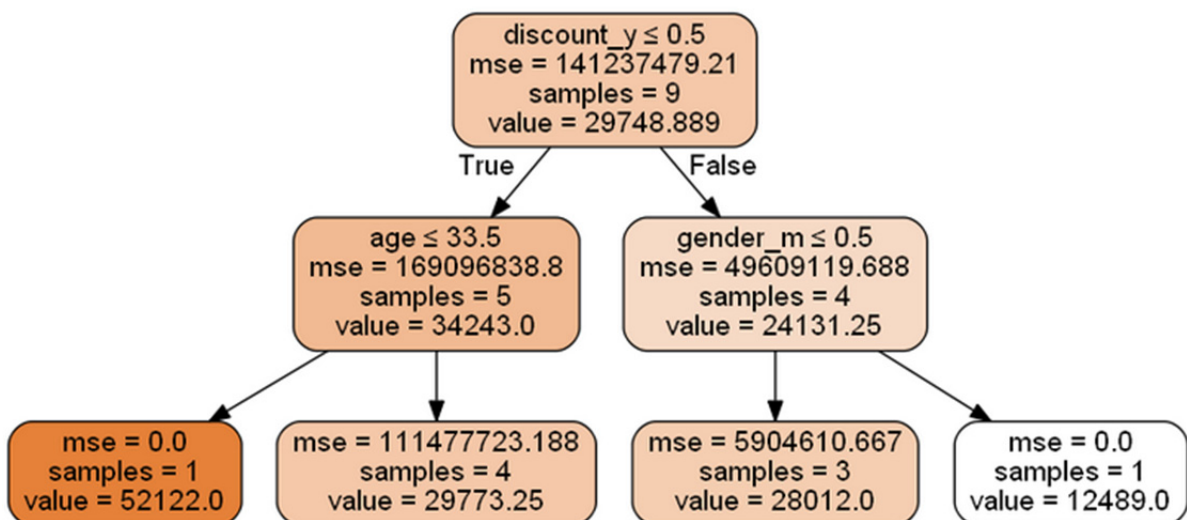


Рис. 1.9. Дерево решений для набора данных (табл. 1.1) с исключенным четвертым наблюдением (отклик – totalsum)

Это дерево имеет другие предикаты для решающего правила, иные листья. Конечно, столь явные отличия возникли из-за малого объема обучающей выборки, для которой исключенное наблюдение – это 10 % обучающих примеров. Однако этот пример показывает, что метод деревьев решений дает нестабильные результаты при небольших изменениях обучающего набора данных.

При построении одиночных деревьев на этапе выбора предикативной переменной для решающего правила происходит сравнение различных вариантов разбиений и на основании критерия качества выбирается лучший вариант. Кроме значительного объема вычислений при большом числе предикатов это также дает преимущество оказаться выбранным для правила расщепления переменным с большим числом возможных значений: между количественной и категориальной переменными выбор смещается в пользу количественной.

Оценить качество классификации можно с помощью таблицы сопряженности. Каждый столбец этой таблицы соответствует фактическим классам из тестового набора, а каждая строка – классам, предсказанным классификатором (табл. 1.2).

Таблица 1.2

**Анализ соответствия предсказанных
и фактических классов размеченных данных**

	$y = 1$	$y = 0$	
$\hat{y}=1$	True Positive (TP)	False Positive (FP)	TP + FP
$\hat{y}=0$	False Negative (FN)	True Negative (TN)	FN + TN
	TP + FN	FP + TN	TP + FN + FP + TN

Очевидно, что в случае двух классов возможны ошибки двух видов: False Positive (FP) – неверно распознан объект класса с меткой 0 и False Negative (FN) – неверно распознан объект класса с меткой 1.

Метрики, оценивающие качество проведенной кластеризации, следующие [12].

Accuracy – это метрика, которая показывает долю правильных ответов модели. Ее значение равно отношению числа правильных ответов, которые дала модель, к числу всех объектов. Эта метрика зависит от соотношения классов объектов в выборке и используется нечасто.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (1.5)$$

Для оценки качества распознавания каждого класса вводятся метрики *precision* (точность) и *recall* (полнота).

Precision (точность) показывает долю объектов, названных классификатором положительными и при этом действительно являющимися положительными:

$$Precision = \frac{TP}{TP + FP}. \quad (1.6)$$

Recall (полнота) показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм:

$$Recall = \frac{TP}{TP + FN}. \quad (1.7)$$

F-мера – среднее гармоническое *precision* и *recall*. *F-мера* достигает максимума при полноте и точности равными единице и близка к нулю, если один из аргументов близок к нулю:

$$F = \frac{recision \cdot recall}{recision + recall}. \quad (1.8)$$

1.2. Метод ближайших соседей

Материалы: [1; 5, с. 52–55, с. 182–186; 6, с. 60–62; 8, тема 7.1].

Метод ближайших соседей kNN (k-Nearest Neighbors) – это ленивый классификатор. В процессе обучения классификатор только хранит обучающие данные, никакая модель не строится, параметры модели не определяются. Фактически работа метода начинается в момент поступления в классификатор новых немаркированных данных.

Метод ближайших соседей, как и дерево решений, можно использовать и для задачи классификации, и для задачи регрессии.

Суть метода:

- вычислить расстояние от нового объекта до каждого объекта обучающей выборки;
- определить k объектов, до которых расстояние минимально;
- для нового объекта метка – это значение класса, который встречается среди k объектов чаще всего в задаче классификации (1.9) или среднее значение отклика (целевого признака) в задаче регрессии (1.10).

Определение метки класса в окрестности k отобранных объектов:

$$\max \frac{1}{k} \sum_{i \in N_0} I(y_i = j), \quad (1.9)$$

где N_0 – совокупность k ближайших объектов; j – одно из возможных значений категориальной переменной y (отклика).

Определение среднего значения целевого признака в окрестности k отобранных объектов

$$\frac{1}{k} \sum_{i \in N_0} y_i, \quad (1.10)$$

где N_0 – совокупность k ближайших объектов.

Иногда для определения класса используется не простое большинство, как в (1.9), а учитывается близость объектов из совокупности N_0 к тестовому объекту (1.11): чем ближе объект к тестовому, тем больше вес:

$$\max \frac{1}{k} \sum_{i \in N_0} \rho_i I(y_i = j), \quad (1.11)$$

где N_0 – совокупность k ближайших объектов; j – одно из возможных значений категориальной переменной y (отклика); ρ_i – вес объекта i , определяемый на основе расстояния от i -го объекта до тестового.

Если $k = 1$, то для наблюдений из обучающей выборки результатом применения метода будет само наблюдение (точнее его значение отклика). Так как наблюдение содержится в обучающем множестве, то расстояние от предъявляемого наблюдения до него самого равно нулю. Затруднение в случае классификации будет только в случае, если в обучающей выборке есть более одного наблюдения с совпадающими значениями предикторов и различными значениями отклика.

По умолчанию в программных реализациях метода значение k часто принимается равным пяти.

У метода ближайших соседей есть определенные особенности, которые в условиях работы с реальными данными сильно влияют на результат и требуют шагов по нивелированию влияния:

– для большого набора данных определение расстояния от тестового объекта до всех объектов обучающей выборки – ресурсозатратный этап;

– категориальные переменные часто оказывают меньшее влияние на результаты метода по сравнению с количественными переменными (имеет значение также размерность): вклад предиктора пол (значения: 0 – женский, 1 – мужской) в расстояние между объектами будет меньше, чем предиктора заработная плата (интервал значений от 10 тысяч до 150 тысяч), влияние предиктора может еще возрасти, если единица измерения предиктора будет не тысяча рублей, а рубль.

1.3. Линейные модели классификации и регрессии

Материалы: [2; 5, с. 71–117, с. 143–154; 6, с. 168–174; 7, с. 260–278; 8, глава 4; 9, с. 419–433].

Если отклик – это вещественнозначная непрерывная переменная, то обобщение классификаторов на случай, когда отклик имеет бесконечное число возможных значений, а не ограничен дискретным набором меток классов, дает не всегда приемлемые для прогнозирования результаты. Также есть проблема с зашумленностью данных: на отклик влияют не только те предикторы, которые регистрировались в наблюдениях, но и те, для которых регистрация не была проведена или просто невозможна. Разумно считать, что модель должна улавливать приближенную связь (общий тренд) между откликом и предикторами.

Самая простая модель регрессии – полином наименьшей степени с одним предиктором: одномерная линейная регрессия, которая моделирует связь между зависимой переменной (количественным откликом) y и единственной объясняющей переменной (количественным предиктором) x :

$$y \approx \beta_0 + \beta_1 x, \quad (1.12)$$

где β_0 – свободный член (определяет пересечение с осью абсцисс линии регрессии); β_1 – коэффициент при независимой переменной (определяет угол наклона линии регрессии).

Интерпретация β_0 – это значение зависимой переменной при условии равенства нулю предиктора (если такое значение предиктора имеет смысл), β_0 может также интерпретироваться как случайная часть влияния отсутствующих в модели предикторов. Интерпретация β_1 – это среднее изменение отклика при увеличении пре-

диктора на одну единицу. При этом единицы связаны с измерением отклика и предиктора. Если отклик – это объем реализации в тысячах штук, а предиктор – цена товара в тысячах рублей, то β_1 показывает, на сколько тысяч изменится объем продаж при увеличении цены товара на тысячу рублей.

Знак равенства в формуле (1.12) возникает при включении случайной поправки ε , отражающей статистический характер связи между y и x , который моделируется линейной функцией.

На модель линейной регрессии накладываются ограничения:

- математическое ожидание случайных ошибок равно нулю;
- дисперсия случайных ошибок одинакова и конечна, это свойство называется гомоскедастичностью (гомо – одинаковый, скедастичность – разброс);
- случайные ошибки не коррелированы, т.е. каждая ошибка конкретного наблюдения распределена независимо от других.

Параметры модели β_0 и β_1 неизвестны, и с помощью обучающей выборки на основе одного из методов оценки параметров получают их оценки $\hat{\beta}_0$ и $\hat{\beta}_1$, модельное значение (предсказанное значение) зависимой переменной также обозначается $\hat{y}$ с помощью диакритического знака '^'.

Тогда модельное уравнение регрессии можно представить в виде:

$$\hat{y} \approx \hat{\beta}_0 + \hat{\beta}_1 x. \quad (1.13)$$

Отклонение модельного значения $\hat{y}$ от наблюдаемого y называется остатком:

$$e_i = y_i - \hat{y}_i, \quad (1.14)$$

где i – номер наблюдения в обучающей выборке.

Чтобы избежать переобучения, действуем согласно эвристическому правилу: количество параметров модели, подлежащих оценке, должно быть значительно меньше объема выборки.

Для оценки параметров модели очень часто используется метод наименьших квадратов (МНК), но существуют и альтернативные методы, например, метод максимального правдоподобия, метод инструментальных переменных.

Суть МНК заключается в минимизации суммы квадратов отклонений модельных значений отклика от наблюдаемых значений для обучающей выборки:

$$\sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x)^2 \rightarrow \min. \quad (1.15)$$

С формулой (1.15) без направления оптимизации уже знакомы – это средневзвешенная квадратичная ошибка MSE (1.3), определяющая качество расщепления в деревьях регрессии. Это неудивительно, многие методы, решающие задачу регрессии, используют именно сумму квадратов остатков (отклонений моделируемых откликов от наблюдаемых) для контроля над качеством построения модели по алгоритму обучения.

Для решения оптимизационной задачи необходимо вычислить производные по параметрам модели, приравнять их к нулю и решить полученные уравнения относительно параметров $\hat{\beta}_0$ и $\hat{\beta}_1$:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}, \quad (1.16)$$

где $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ – выборочные средние.

Для оценки качества модели регрессии рассматривается коэффициент детерминации R^2 – доля изменения зависимой переменной, объясненная предиктором:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{\beta}_0 - \hat{\beta}_1 x)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} = \frac{\sum_{i=1}^n (y_i - e_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}. \quad (1.17)$$

Чем ближе это значение к единице, тем лучше модель.

Однако обычно опираются не на числовое значение характеристики модели, а тестируют статистические гипотезы. Рассматриваются исходная гипотеза H_0 и альтернативная H_1 . Для выбора предпочтительной гипотезы на основе характеристик модели, построенной на основе обучающей выборки, вычисляется некоторый статистический критерий, для числового значения которого определяется p-value – вероятность, что отклоняемая гипотеза H_0 верна. Если p-value оказывается ниже некоторого порогового значения (чаще всего 0.05 или 0.01), отклонив гипотезу H_0 в пользу H_1 , можно предполагать, что ошибка такого решения менее 5 % (менее 1 %), но это не дает гарантий, что наш частный случай – то самое редкое исключение.

В случае с коэффициентом детерминации рассматривается гипотеза H_0 : его значение в модели, построенной на основе обучающей выборки, говорит об отсутствии влияния предиктора x на отклик y . Альтернативная гипотеза: есть статистически значимая связь между предиктором и откликом.

Множественная регрессия позволяет построить модель зависимости количественного отклика от нескольких предикторов. Однако в этом случае возникает проблема с оценкой полезности каждого включенного предиктора для предсказания отклика. Если модель оценивается как статистически незначимая (для коэффициента детерминации при тестировании не отклоняется гипотеза H_0), то вопрос со всеми предикторами можно до некоторой степени считать решенным – модель статистически незначима и вряд ли может использоваться для предсказаний.

Если гипотезу H_0 отклонили, то это не гарантия, что все предикторы улучшают качество модели.

Один из механизмов оценки тесноты связи предикторов и отклика – корреляционная матрица, которая состоит из значений коэффициентов корреляции между переменными, включенными в модель:

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}. \quad (1.18)$$

Чем ближе к единице значение модуля коэффициента корреляции, тем теснее между признаками линейная связь (тем ближе она к функциональной). Знак коэффициента корреляции указывает на направление связи, что происходит с откликом при увеличении предиктора. Если коэффициент корреляции положителен, то в ответ на увеличение предиктора отклик увеличивается; отрицателен – в ответ на увеличение предиктора отклик уменьшается.

При статистической незначимости модели линейной регрессии можно делать заключения об отсутствии между откликом и предикторами линейных связей, но не об отсутствии связей в принципе (связи могут иметь нелинейный характер). И даже при наличии нелинейных связей между предикторами и откликом уравнение линейной регрессии может показать неплохое качество (многие нелинейные связи хорошо аппроксимируются линейными зависимостями).

Как и в предыдущих разделах возникает вопрос: как адаптировать метод для решения задачи классификации, когда отклик – это категориальная переменная? В простейшем случае отклик принимает два возможных значения, которые можно закодировать метками 1 и 0.

Для решения задачи классификации с помощью регрессии рассмотрим логистическую регрессию, которая является линейным классификатором. Ценность логистической регрессии в том, что она не просто предсказывает, к какому классу относится предъявленный объект, но и прогнозирует вероятность уверенности в отнесении объекта к этому классу.

Основная идея линейного классификатора заключается в том, что признаковое пространство может быть разделено гиперплоскостью на два полупространства, в каждом из которых прогнозируется одно из двух значений целевого класса. Если это можно сделать без ошибок, то обучающая выборка называется *линейно разделимой*.

Если для моделирования отклика использовать линейную регрессионную модель, то столкнемся с ситуацией, когда предсказания будут выходить за пределы интервала (0, 1) – предсказанное значение может быть как меньше наблюдаемого, т.е. меньше 0, так и больше наблюдаемого, т.е. больше 1. Поэтому в модели используют преобразование линейной модели, ограничивающее отклик интервалом от нуля до единицы:

$$p(X) \approx \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}. \quad (1.19)$$

Величина:

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X} \quad (1.20)$$

называется риском события (или шансом). Так, например, если в модели оттока абонентов оператора сотовой связи (0 – остался, 1 – ушел) $p(X) = 0.2$, то для сегмента клиентов с такой вероятностью ухода уйдет один из пяти (риск ухода для такого сегмента – 1/4).

Оценка параметров модели (1.19) проводится по методу максимального правдоподобия.

Логистическая регрессия может быть расширена на случай, когда отклик имеет более двух уровней, но чаще в таком случае используют дискриминантный анализ.

2. Обучение с учителем на языке R, RStudio

2.1. Деревья решений

В R для построения деревьев решений необходимо импортировать пакет `rpart`, подключить его в текущей сессии и воспользоваться функцией `rpart()` (аргументы функции – в табл. 2.1, параметры метода – в табл. 2.2, результат – в табл. 2.3).

```
install.packages('rpart', dependencies=TRUE)
library(rpart)
tree <- rpart(y ~ X1+X2, method="class")
```

Таблица 2.1

Аргументы `rpart()`

formula	Формула: имя_переменной_отклика~имя_предиктора1+имя_предиктора2 Если все переменные, кроме отклика, являются предикторами в модели, то используется сокращенная запись ~. имя_переменной_отклика ~ .
data	Фрейм данных, столбцы которого используются в формуле. Необязательный аргумент
method	Для деревьев регрессии используется значение <code>method='anova'</code> . Для деревьев классификации используется значение <code>method='class'</code>
control	Список параметров, управляющих подробностями алгоритма. По умолчанию: <code>minsplit = 20</code> , <code>minbucket = round(minsplit/3)</code> , <code>cp = 0.01</code> , <code>maxcompete = 4</code> , <code>maxsurrogate = 5</code> , <code>usesurrogate = 2</code> , <code>xval = 10</code> , <code>surrogatestyle = 0</code> , <code>maxdepth = 30</code> , ... (см. табл. 2.2)

Таблица 2.2

Параметры `rpart.control()`

minsplit	Минимальное число наблюдений, которое должно существовать в узле для того, чтобы была предпринята попытка разделения
minbucket	Минимальное количество наблюдений в любом терминальном узле (листе). Если не указано явно числовое значение, то рассчитывается через <code>minsplit</code> по формуле <code>minbucket = round(minsplit/3)</code> . Если указано значение <code>minbucket</code> , а значение <code>minsplit</code> нет, то оно восстанавливается через <code>minbucket</code> по приведенной формуле

ср	Параметр сложности. Основная роль этого параметра заключается в экономии вычислительного времени путем отсеечения расщеплений, которые, очевидно, не стоят того. По существу, пользователь сообщает программе, что любое разделение, которое не улучшает подгонку по ср, скорее всего, будет обрезано перекрестной проверкой и поэтому программе не нужно его преследовать
maxdepth	Максимальная глубина любого узла конечного дерева, причем корневой узел будет считаться глубиной 0.

```
# Выведем краткую сводку по дереву для gpart()
tree
# Выведем подробную сводку результата для gpart()
summary(tree)
```

Таблица 2.3

Сводка по результатам gpart()

ср	Изменение сложности модели с ростом дерева
nsplit	Номер разбиения
rel error	Относительная ошибка классификации: доля наблюдений, неверно классифицированных моделью, деленная на базовую ошибку классификации (долю миноритарной категории зависимой переменной в выборке). При таком масштабировании в начальном узле относительная ошибка классификации равна 1. Для дерева регрессии это значение – доля изменчивости отклика, необъясненная моделью
xerror	Кросс-валидационная ошибка классификации (относительная). Например, для деревьев классификации это оценка доли неверно классифицированных наблюдений на новых данных. Также для получения абсолютной кросс-валидационной ошибки нужно умножить полученное значение на базовую ошибку классификации. Для дерева регрессии – это сумма квадратов остатков (квадратов разностей между наблюдаемым значением отклика и предсказанным) для наблюдений, не участвовавших в построении дерева
xstd	Стандартное отклонение для кросс-валидационной ошибки
Variable importance	Важность переменных (упорядочены по убыванию этого параметра)

left son (obs) right son (obs)	Номер узла, соответствующего левому и правому узлу-потомку, а также число наблюдений, попавших в каждый узел-потомок
Для дерева классификации	
predicted class	Показывает прогноз категории зависимой переменной для узла
expected loss	Доля неверно классифицированных наблюдений в узле
P(node)	Доля наблюдений исходной выборки попавших в узел
class counts	Распределение наблюдений по категориям
probabilities	Распределение вероятностей по категориям
Для дерева регрессии	
mean	Среднее значение отклика для наблюдений узла
MSE	Средняя квадратическая ошибка, значение находится по формуле (1.3)

В R для визуализации деревьев решений необходимо импортировать пакет `rpart.plot`, подключить его в текущей сессии и воспользоваться функцией `rpart.plot()`.

```
install.packages('rpart.plot', dependencies=TRUE)
library(rpart.plot)
rpart.plot(tree)
```

В результате будет визуализация для краткой сводки по дереву. Построим дерево классификации и дерево регрессии для набора из табл. 1.1.

Подключим предварительно загруженные пакеты и импортируем набор данных табл. 1.1 из файла `shop.csv`.

```
library(rpart)
library(rpart.plot)
df_shop<-read.table("shop.csv", header=T, sep=";")
#Построим дерево классификации для отклика discount
shop_tree_cl <- rpart( discount ~ ., method="class", data=df_shop, minsplit = 2)
#Выводим краткую сводку по построенному дереву
shop_tree_cl
```

В построенном дереве было проведено два разбиения (рис. 2.1).

В начальном узле с номером 1 все 10 исходных наблюдений, пять наблюдений относятся к минорной категории (так как в переменной `discount` доля наблюдений каждого значения одинакова – 50 %, то минорная категория может быть любая из двух, но здесь это значение `discount='y'`), минорная категория определяется как неверно классифицированная, соответственно, предсказанная для узла категория `discount='n'`.

```

n= 10

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

1) root 10 5 n (0.5000000 0.5000000)
2) totalsum>=31451.5 4 0 n (1.0000000 0.0000000) *
3) totalsum< 31451.5 6 1 y (0.1666667 0.8333333)
   6) age>=42.5 1 0 n (1.0000000 0.0000000) *
   7) age< 42.5 5 0 y (0.0000000 1.0000000) *

```

Рис. 2.1. Краткая сводка по дереву классификации

Разбиение осуществляется по переменной `totalsum` (значение разбиения – 31451.5). Левый узел – четыре наблюдения, для которых $\text{totalsum} \geq 31415.5$, все эти наблюдения соответствуют отклику `discount='n'`, поэтому левый узел терминальный, число неверно классифицированных – 0, предсказанная категория – 'n'. Правый узел – шесть наблюдений, для которых $\text{totalsum} < 31415.5$, неверно классифицированных наблюдений – 1, предсказанная категория – 'y'. Вероятность для 'n' в этом узле – 1/6 или 0.1667, для 'y' – 5/6 или 0.8333. Этот узел не терминальный, для него разбиение продолжается по предиктору `age` (значение для разбиения – 42.5). В левом узле – одно наблюдение, неверно классифицированных – нет, предсказанная категория 'n' – узел терминальный. В правом узле – пять наблюдений, неверно классифицированных нет, предсказанная категория 'y' – узел терминальный. Всего терминальных узлов – три (на один больше, чем разбиений).

#Визуализируем дерево классификации
`rpart.plot(shop_tree_cl)`

На рис. 2.2 показан результат выполнения кода. На дереве отображаются вероятность категории 'y' и доля исходной выборки, попавшая в этот узел (рис. 2.3).

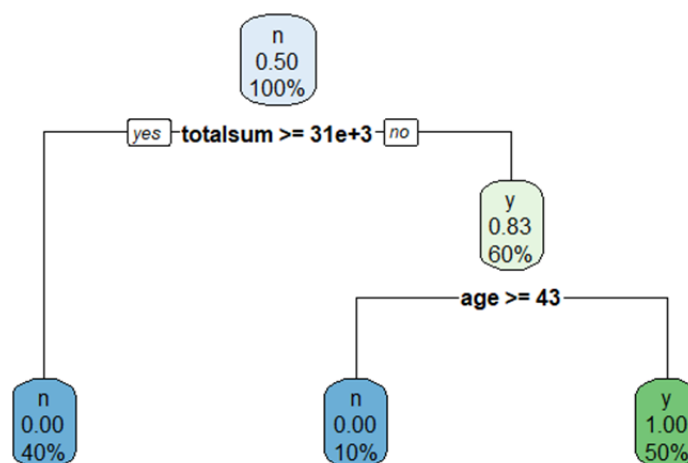


Рис. 2.2. Визуализация дерева классификации

```
#Выведем подробную сводку по построенному дереву
summary(shop_tree_cl)
```

```

      CP nsplit rel error xerror      xstd
1 0.80      0      1.0      2.0 0.0000000
2 0.20      1      0.2      0.6 0.2898275
3 0.01      2      0.0      0.6 0.2898275

Variable importance
  age totalsum  gender
    44      44     11

Node number 1: 10 observations,      complexity param=0.8
predicted class=n expected loss=0.5 P(node) =1
  class counts:      5      5
  probabilities: 0.500 0.500
left son=2 (4 obs) right son=3 (6 obs)
Primary splits:
  totalsum < 31451.5 to the right, improve=3.333333, (0 missing)
  age < 42.5 to the right, improve=2.142857, (0 missing)
  gender splits as RL, improve=0.200000, (0 missing)
Surrogate splits:
  age < 48 to the right, agree=0.8, adj=0.50, (0 split)
  gender splits as RL, agree=0.7, adj=0.25, (0 split)

Node number 2: 4 observations
predicted class=n expected loss=0 P(node) =0.4
  class counts:      4      0
  probabilities: 1.000 0.000

Node number 3: 6 observations,      complexity param=0.2
predicted class=y expected loss=0.1666667 P(node) =0.6
  class counts:      1      5
  probabilities: 0.167 0.833
left son=6 (1 obs) right son=7 (5 obs)
Primary splits:
  age < 42.5 to the right, improve=1.6666670, (0 missing)
  totalsum < 20731.5 to the left, improve=0.6666667, (0 missing)
  gender splits as LR, improve=0.1666667, (0 missing)

Node number 6: 1 observations
predicted class=n expected loss=0 P(node) =0.1
  class counts:      1      0
  probabilities: 1.000 0.000

Node number 7: 5 observations
predicted class=y expected loss=0 P(node) =0.5
  class counts:      0      5
  probabilities: 0.000 1.000
```

Рис. 2.3. Подробная сводка по дереву классификации

Построим дерево регрессии для набора данных из табл. 1.1 (переменная отклика totalsum).

```
shop_tree_rg <- rpart(totalsum ~ ., method="anova", data=df_shop, minsplit = 3)
shop_tree_rg
```

Краткая сводка по построенному дереву регрессии (рис. 2.4). В начальном узле с номером 1 все 10 исходных наблюдений, показатель deviance – это значение MSE для узла, умноженное на число наблюдений, или иначе просто сумма квадратов отклонений значений отклика в наблюдениях узла от средних значений (формула (1.3)). Значение yval – среднее значение переменной отклика для наблюдений узла. Узел 2 – левый узел разбиения по переменной age (условие разбиения $age < 28.5$), в узле – два наблюдения, для которых $MSE = 176137700/2$, а среднее значение отклика – 21873.5. Узел 3 – правый узел разбиения по переменной age (условие разбиения $age \geq 28.5$), в узле – восемь наблюдений, для которых $MSE = 1738897000/8$, а среднее значение отклика – 35786.62. В узле 3 происходит разбиение по переменной age (узлы 6 и 7). В узле 6 – разбиение по переменной gender (узлы 12 и 13). В узле 13 – расщепление по переменной age (узлы 26 и 27).

```
n= 10

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

1) root 10 2224755000 33004.00
 2) age< 28.5 2 176137700 21873.50 *
 3) age>=28.5 8 1738897000 35786.62
   6) age>=30.5 7 935515100 31999.00
    12) gender=f 3 49996420 23558.00 *
    13) gender=m 4 511453700 38329.75
       26) age>=33.5 3 257818800 33732.33 *
       27) age< 33.5 1 0 52122.00 *
  7) age< 30.5 1 0 62300.00 *
```

Рис. 2.4. Краткая сводка по дереву регрессии

```
#Визуализируем дерево регрессии
rpart.plot(shop_tree_rg)
```

На рис. 2.5 показан результат выполнения кода. На дереве отображаются среднее значение отклика totalsum для наблюдений узла (т.е. предсказание) и доля исходной выборки, попавшая в этот узел.

```
#Выведем подробную сводку по построенному дереву
summary(shop_tree_rg)
```

На рис. 2.6 показано начало сводки по результатам построения дерева регрессии.

Для получения предсказаний по модели используется функция `predict(model, data)`, где фрейм данных `data` должен иметь аналогичную структуру (названия столбцов, тип данных), что и обучающие данные, а в качестве `model` используется построенное дерево.

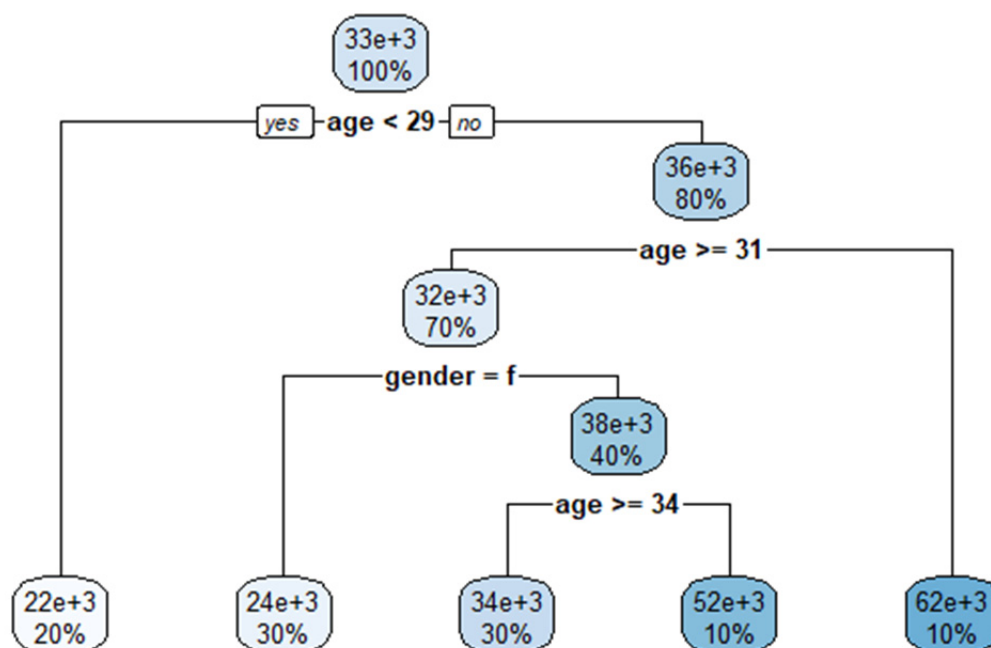


Рис. 2.5. Визуализация дерева регрессии

```

      CP nsplit rel error   xerror   xstd
1 0.2501628      0 1.0000000 1.234568 0.4457160
2 0.1681377      2 0.4996743 2.110798 0.4623229
3 0.1140058      3 0.3315367 2.588530 0.6162292
4 0.0100000      4 0.2175309 2.483997 0.6110758

Variable importance
age gender
 80    20

Node number 1: 10 observations,      complexity param=0.2501628
mean=33004, MSE=2.224755e+08
left son=2 (2 obs) right son=3 (8 obs)
Primary splits:
  age      < 28.5 to the left, improve=0.1392154000, (0 missing)
  discount splits as RL,      improve=0.0069001810, (0 missing)
  gender   splits as LR,      improve=0.0001116427, (0 missing)

Node number 2: 2 observations
mean=21873.5, MSE=8.806884e+07

Node number 3: 8 observations,      complexity param=0.2501628
mean=35786.62, MSE=2.173621e+08
left son=6 (7 obs) right son=7 (1 obs)
Primary splits:
  age      < 30.5 to the right, improve=0.46200660, (0 missing)
  gender   splits as LR,      improve=0.02975443, (0 missing)
  discount splits as LR,      improve=0.01827042, (0 missing)

```

Рис. 2.6. Подробная сводка по дереву регрессии (начало)

```
#Для данных обучающей выборки ля построим прогноз
predict(shop_tree_cl, df_shop)
```

2.1.1. Набор данных оператора связи (дерево классификации)

Набор данных [4, с. 319–320] можно скачать из репозитория <https://github.com/Smeilz/ML-Class/tree/master/R/Trees> имя файла Churn_binned.csv.

Описание переменных набора:

- longdist: длительность междугородних звонков в минуту (порядковый предиктор);
- local: длительность местных звонков в минуту (порядковый предиктор);
- int_disc: скидка за междугородние звонки (номинальный предиктор);
- billtype: тип местных звонков (номинальный предиктор);
- pay: способ оплаты (номинальный предиктор);
- gender: пол (номинальный предиктор);
- marital: семейное положение (номинальный предиктор);
- incomecat: категория дохода (порядковый предиктор);
- agecat: возрастная категория (порядковый предиктор);
- churn: наличие оттока (номинальный предиктор).

Выполним построение дерева классификации для переменной отклика churn – отток абонентов.

```
#Подключим библиотеки
library(rpart)
library(rpart.plot)
#Импортируем данные
df_churn<-read.table("Churn_binned.csv", header=T, sep=";")
#Выполним преобразование для порядковых предикторов
#упорядочим уровни
ord_longdist<- c("<2", "2--8", "9--14", "15-20", "21+")
df_churn$longdist <- ordered(df_churn$longdist, levels = ord_longdist)
ord_local<- c("<8", "8--20", "21-35", "36-55", "56+")
df_churn$local <- ordered(df_churn$local, levels = ord_local)
ord_incomecat<- c("<15", "15-35", "36-50", "51-65", "66+")
df_churn$incomecat <- ordered(df_churn$incomecat, levels = ord_incomecat)
ord_agecat<- c("<31", "31-45", "46-58", "59-70", "71+")
df_churn$agecat <- ordered(df_churn$agecat, levels = ord_agecat)
#Построим дерево классификации (переменная отклика churn)
churn_tree_cl <- rpart(churn ~ .,method="class", data=df_churn)
#Выведем краткую сводку по дереву
churn_tree_cl
```

На рис. 2.7 можно увидеть краткую сводку по построенному дереву классификации и определить, какими значениями предикторов обладают абоненты, которые ушли от оператора. Например,

в узле 13 – женщины в возрастной группе до 45 лет, с длительностью междугородних звонков меньше двух минут имеют вероятность ухода в 0.8586.

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

1) root 4431 1935 Остается (0.5633040 0.4366960)
2) gender=Мужской 2187 441 Остается (0.7983539 0.2016461)
4) local=8--20,21-35,36-55,56+ 1839 243 Остается (0.8678630 0.1321370) *
5) local=<8 348 150 Уходит (0.4310345 0.5689655)
10) longdist=2--8,9--14,15-20,21+ 123 18 Остается (0.8536585 0.1463415) *
11) longdist=<2 225 45 Уходит (0.2000000 0.8000000) *
3) gender=Женский 2244 750 Уходит (0.3342246 0.6657754)
6) agecat=<31,31-45 1049 482 Уходит (0.4594852 0.5405148)
12) longdist=2--8,9--14,15-20,21+ 858 403 Остается (0.5303030 0.4696970)
24) incomecat=36-50,51-65,66+ 520 199 Остается (0.6173077 0.3826923) *
25) incomecat=<15,15-35 338 134 Уходит (0.3964497 0.6035503) *
13) longdist=<2 191 27 Уходит (0.1413613 0.8586387) *
7) agecat=46-58,59-70,71+ 1195 268 Уходит (0.2242678 0.7757322) *
```

Рис. 2.7. Краткая сводка по дереву классификации

На новых данных (для клиентской базы активных абонентов) с использованием построенной модели можно предсказать вероятность их ухода и использовать предсказания для сегментации клиентов в программах лояльности: более сильные воздействия применять к сегментам, максимально склонным к уходу.

На рис. 2.8 показано визуальное представление построенного дерева решений (параметр sex настраивает размер шрифта для текста на графике).

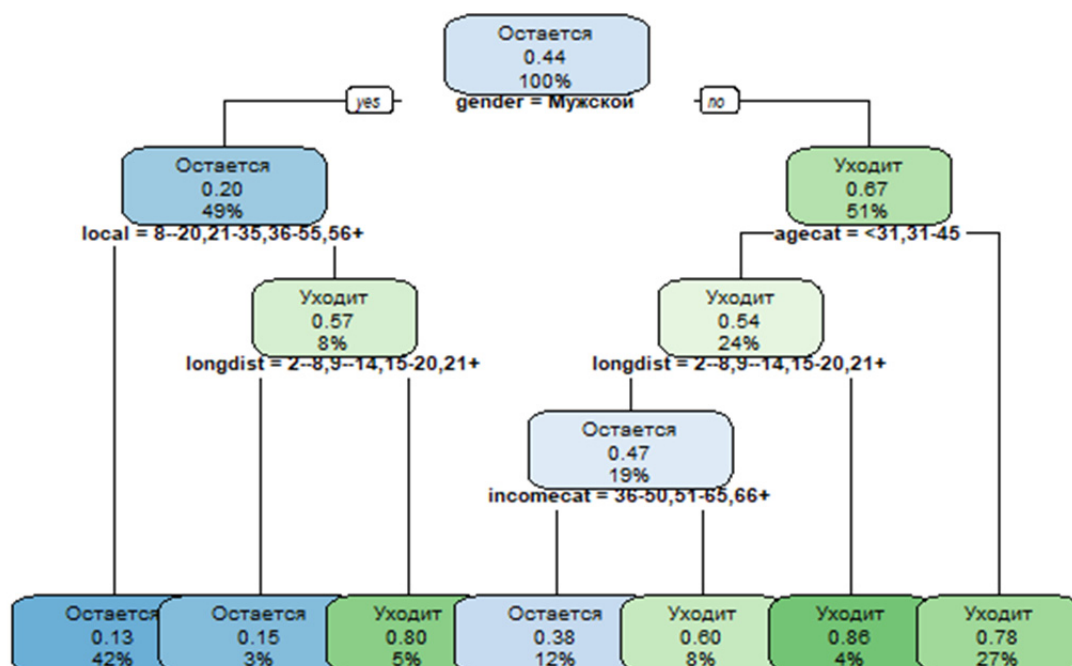


Рис. 2.8. Визуализация дерева классификации

```
rpart.plot(churn_tree_cl, cex =.6)
```

Для построения дерева решений на данных оператора сотовой связи можно использовать также набор <https://www.kaggle.com/becksddef/churn-in-telecoms-dataset>.

2.1.2. Набор данных *медицинское страхование* (дерево регрессии)

Данные, которые мы используем, – расходы на медицинское обслуживание тех, кто имеет медицинскую страховку: URL: <https://www.kaggle.com/mirichoi0218/insurance>

Описание переменных набора:

- age: возраст основного бенефициара;
- sex: пол застрахованного;
- bmi: индекс массы тела;
- children: число детей, охваченных медицинским страхованием / число иждивенцев;
- smoker: курит ли застрахованный;
- region: жилой район получателя в США, Северо-Восток, Юго-Восток, Юго-Запад, Северо-Запад;
- charges: индивидуальные медицинские расходы, оплачиваемые страховкой.

Выполним построение дерева регрессии для переменной отклика charges – медицинские расходы.

```
#Подключим библиотеки
library(rpart)
library(rpart.plot)
#Импортируем данные
df_insurance<-read.table("insurance.csv", header=T, sep=",")
#Построим дерево регрессии (переменная отклика charges)
charges_tree_rg <- rpart(charges ~ .,method="anova", data=df_insurance)
#Выведем краткую сводку по дереву
charges_tree_rg
```

На рис. 2.9 представлена краткая сводка по построенному дереву регрессии. Как и ожидалось, самые большие расходы, оплачиваемые страховкой, у курящих страхователей с индексом массы тела больше 30 (избыточной массой тела) – терминальный узел 7. Самые небольшие медицинские расходы у некурящих в возрастной группе меньше 42.5 лет – терминальный узел 4.

На рис. 2.10 – визуальное представление построенного дерева решений

```
rpart.plot(charges_tree_rg, cex =1)
```

```

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

1) root 1338 196074200000 13270.420
  2) smoker=no 1064 38188720000 8434.268
    4) age< 42.5 596 13198540000 5398.850 *
    5) age>=42.5 468 12505450000 12299.890 *
  3) smoker=yes 274 36365600000 32050.230
    6) bmi< 30.01 130 3286655000 21369.220 *
    7) bmi>=30.01 144 4859010000 41692.810 *

```

Рис. 2.9. Краткая сводка по дереву регрессии

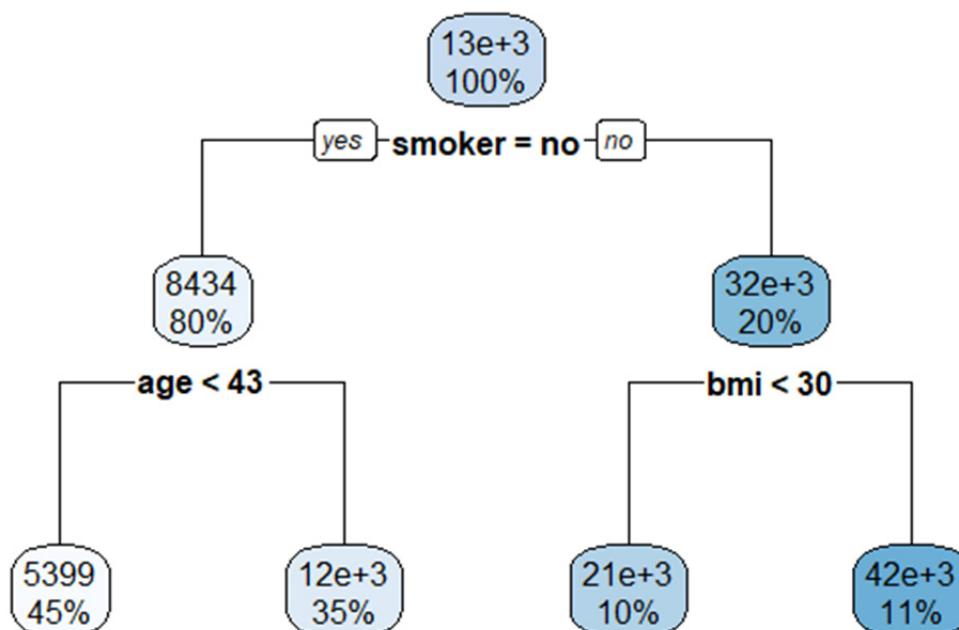


Рис. 2.10. Визуализация дерева регрессии

2.2. Метод ближайших соседей

В R для реализации метода ближайших соседей можно воспользоваться функцией `knn()` из библиотеки `class` (аргументы функции – в табл. 2.4).

#Подключим библиотеку

```
library(class)
```

```
fit <- knn(train, test, cl, k = 1, l = 0, prob = TRUE)
```

Таблица 2.4

Аргументы функции `knn()`

train	Матрица или фрейм данных для предикторов обучающей выборки. Важно, чтобы все предикторы имели тип <code>numeric</code>
test	Матрица или фрейм данных для предикторов тестовой выборки. Важно, чтобы все предикторы имели тип <code>numeric</code>

cl	Вектор меток или значений отклика для обучающей выборки
k	Количество соседей для предсказания значения отклика наблюдения тестовой выборки
prob	Значение по умолчанию False, если True выводит долю голосов за победивший класс

Для разбиения исходной выборки на тренировочную (обучающую) и тестовую воспользуемся функцией `createDataPartition()` из библиотеки `caret` (аргументы функции – в табл. 2.5).

```
#Инсталлируем библиотеку
install.packages('caret', dependency=TRUE)
#Подключим библиотеку
library(caret)
#Запишем в переменную tr.index случайно отобранные индексы отклика
tr.index <- createDataPartition(y, p = 0.8, list = FALSE)
```

Таблица 2.5

Аргументы функции `createDataPartition()`

y	Вектор откликов по всему набору данных
p	Доля наблюдений для обучающей выборки
list	Результаты должны быть представлены списком (True) или матрицей (False)

Рассмотрим работу функции на наборе данных (см. табл. 1.1). Исходные наблюдения были случайно разделены на обучающую выборку (восемь наблюдений) и тестовую (два наблюдения: 1 и 10). Тестовые наблюдения выделены на рис. 2.11 в рамку. Размер маркера на графике отвечает значению переменной `discount` (если `discount='y'`, то размер большой, иначе – маленький).

Для применения метода ближайших соседей к задаче классификации в качестве предикторов выбраны: `totalsum` и `age`, отклик – `discount`. На рис. 2.11 видно, что если $k = 1$, то ближайшие наблюдения (отмечены стрелкой на графике) для первого тестового наблюдения – это девятое в исходном наборе, и для этих наблюдений значения отклика `discount` совпадают, а для второго тестового наблюдения – это пятое наблюдение в исходном наборе, и значения отклика `discount` для них различны. Таким образом, в результате применения `knn()` к так разделенным наблюдениям одно из двух тестовых наблюдений будет классифицировано верно, а другое – ошибочно.

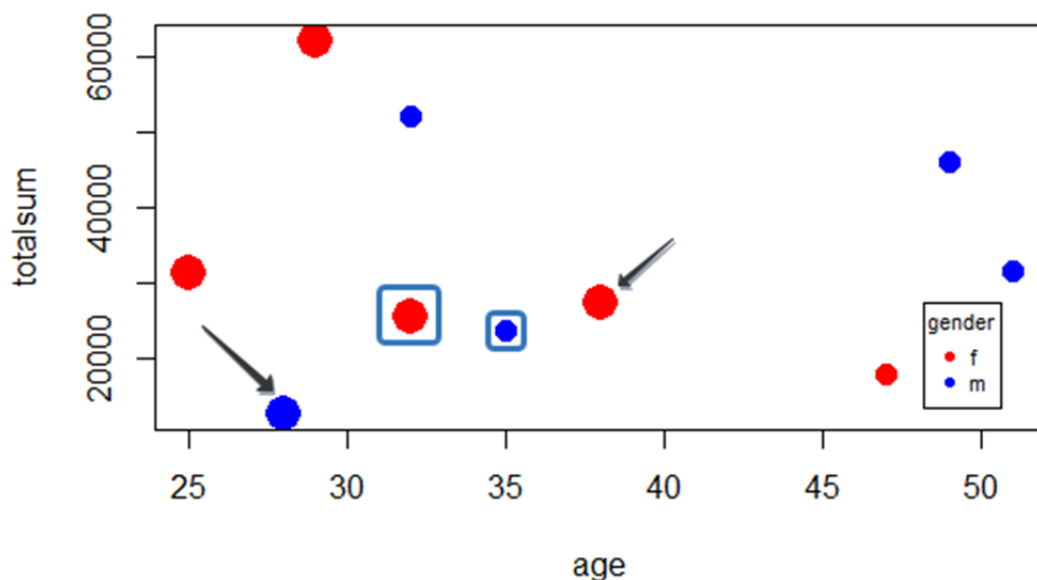


Рис. 2.11. Метод ближайших соседей (классификация)

Для применения метода ближайших соседей к задаче регрессии в качестве предикторов выбраны: discount и age, отклик – totalsum. Исходные наблюдения были случайно разделены на обучающую выборку (восемь наблюдений) и тестовую (два наблюдения: 4 и 6). Тестовые наблюдения выделены на рис. 2.12 в рамку.

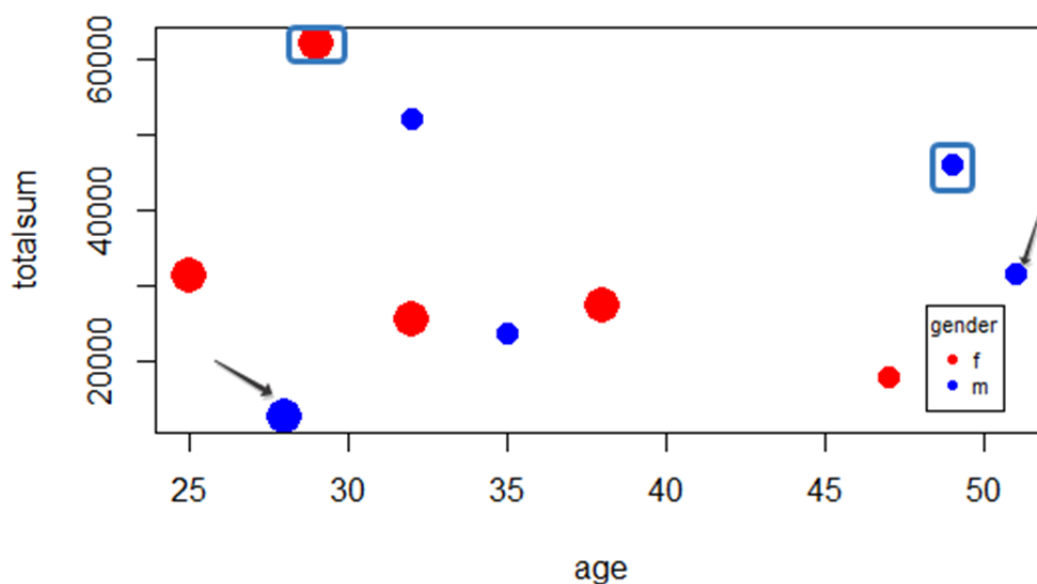


Рис. 2.12. Метод ближайших соседей (регрессия)

На рис. 2.12 видно, что если $k = 1$, то ближайшие наблюдения (отмечены стрелкой на графике) для первого тестового наблюдения – это девятое в исходном наборе, и для этих наблюдений значения отклика totalsum отличаются кардинально 62300 (наблюдаемое значение отклика для тестового примера) и 12489 (будет предсказано по

ближайшему соседу), а для второго тестового наблюдения – это восьмое наблюдение в исходном наборе, и значения отклика totalsum для этих наблюдений отличаются не так существенно: 45985 (наблюдаемое значение отклика для второго тестового примера) и 31645 (будет предсказано по ближайшему соседу).

Иллюстративный пример демонстрирует, что метод ближайших соседей не всегда дает удовлетворительные результаты (особенно на малых выборках и при количестве соседей меньше пяти).

2.2.1. Набор данных оператора связи (классификация)

Данные, которые используем, – отток клиентов оператора связи (см. 2.1.1).

Переменная отклика churn – отток абонентов. Остальные переменные – предикторы.

```
#Подключим библиотеки
library(caret)
library(class)
#Импортируем данные
df_churn<-read.table("Churn_binned.csv", header=T, sep=";")
#Выполним преобразование для порядковых предикторов
#упорядочим уровни
ord_longdist <- c("<2", "2--8", "9--14", "15-20", "21+")
df_churn$longdist <- ordered(df_churn$longdist, levels = ord_longdist)
ord_local <- c("<8", "8--20", "21-35", "36-55", "56+")
df_churn$local <- ordered(df_churn$local, levels = ord_local)
ord_incomecat <- c("<15", "15-35", "36-50", "51-65", "66+")
df_churn$incomecat <- ordered(df_churn$incomecat, levels = ord_incomecat)
ord_agecat <- c("<31", "31-45", "46-58", "59-70", "71+")
df_churn$agecat <- ordered(df_churn$agecat, levels = ord_agecat)
#Для девяти предикторов выполним преобразование типа на numeric
for(i in c(1:9)) df_churn[,i] <- as.numeric(df_churn[,i])
#Разделим исходные данные на тестовые и тренировочные
tr.index = createDataPartition(df_churn$churn, p = 0.8, list = FALSE)
train_churn <- df_churn[tr.index,-10]
test_churn <- df_churn[-tr.index,-10]
cl <- df_churn[tr.index,10]
test_target <- df_churn[-tr.index,10]
#Выполним предсказание отклика churn для тестовых данных
knn_churn<-knn(train_churn, test_churn, cl, k=5)
knn_churn
```

Для 886 наблюдений тестовой выборки будет выведен вектор предсказанных значений отклика churn (рис. 2.13).

```

[1] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[10] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[19] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[28] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[37] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[46] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[55] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[64] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[73] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[82] Уходит Уходит Остается Уходит Уходит Уходит Уходит Уходит Уходит
[91] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит
[100] Уходит Уходит Уходит Уходит Уходит Уходит Уходит Уходит Остается
[109] Уходит Остается Уходит Остается Остается Уходит Уходит Уходит Уходит
[118] Уходит Уходит Остается Остается Остается Остается Остается Остается Остается
[127] Остается Остается Остается Остается Уходит Остается Уходит Остается Остается
[136] Остается Остается Остается Остается Остается Остается Остается Остается Остается

```

Рис. 2.13. Предсказанные значения отклика churn для тестовой выборки (фрагмент)

```

#Оценим частоту совпадений предсказанного отклика
#c наблюдаемым значением для тестовых данных
tb <- table(knn_churn,test_target)
tb

```

На рис. 2.14 можно увидеть таблицу сопряженности по предсказанному отклику для тестовых данных.

```

               test_target
knn_churn      Остается Уходит
Остается         420     117
Уходит           79     270

```

Рис. 2.14. Таблица сопряженности для наблюдаемых и предсказанных значений оттока для тестовой выборки

Как видим, метод ближайших соседей в этих данных лучше справляется с предсказанием для абонентов, которые остаются, чем для абонентов, которые уходят.

2.2.2. Набор данных *медицинское страхование* (регрессия)

Данные, которые мы используем, – расходы на медицинское обслуживание тех, кто имеет медицинскую страховку (см. 2.1.2).

Переменная отклика charges – медицинские расходы, оплачиваемые страховкой. Остальные переменные – предикторы.

```

#Подключим библиотеки
library(caret)
library(class)
#Импортируем данные
df_insurance<-read.table("insurance.csv", header=T, sep=",")
#Для шести предикторов выполним преобразование типа на numeric

```

```

for(i in c(1:6)) df_insurance[,i] <- as.numeric(df_insurance[,i])
#Разделим исходные данные на тестовые и тренировочные
tr.index = createDataPartition(df_insurance$charges, p = 0.8, list = FALSE)
train_insurance <- df_insurance[tr.index,-7]
test_insurance <- df_insurance[-tr.index,-7]
cl <- df_insurance[tr.index,7]
test_target <- df_insurance[-tr.index,7]
#Выполним предсказание отклика charges для тестовых данных
knn_insurance<-knn(train_insurance, test_insurance, cl, k=5)
#Оценим частоту совпадений предсказанного отклика с наблюдаемым
#для тестовых данных
test_insurance['charges'] <- test_target
test_insurance['knn_pred'] <- knn_insurance

```

На рис. 2.15 для тестовых данных представлена таблица для сравнения предсказанного и наблюдаемого значений медицинских расходов. В строках – номер наблюдения исходной выборки, попавший в тестовую.

	age	sex	bmi	children	smoker	region	charges	knn_pred
7	46	1	33.440	1	1	3	8240.590	7345.084
13	23	2	34.400	0	1	4	1826.843	1986.9334
15	27	2	42.130	0	2	3	39611.758	3201.24515
20	30	2	35.300	0	2	4	36837.467	38711
24	34	1	31.920	1	2	1	37701.877	43943.8761
27	63	1	23.085	0	1	1	14451.835	14349.8544
28	55	1	32.775	2	1	2	12268.632	12643.3778
33	19	1	28.600	5	1	4	4687.797	22493.65964

Рис. 2.15. Тестовая выборка, дополненная предсказанными значениями медицинских расходов

2.3. Линейная регрессия

В R для построения модели линейной регрессии можно воспользоваться функцией `lm()` из библиотеки `stats`, которая по умолчанию подключена в среде RStudio (аргументы функции – в табл. 2.6).

```
fit <- lm(formula, data, na.action)
```

Функции, позволяющие вывести дополнительные результаты по линейной модели, показаны в табл. 2.7, аргументом является переменная, в которую записана модель.

Таблица 2.6

Аргументы функции `lm()`

formula	$y \sim x_1 + x_2 + x_3$ Если по умолчанию в модель линейной регрессии должны войти все предикторы, можно записать $y \sim$. Знак '+' используется для разделения предикторов, знак '-' используется для исключения предикторов из модели, если нужно исключить свободный член, то -1
data	Матрица или фрейм данных предикторов и отклика
na.action	Действие в случае наличия в данных NA. По умолчанию такие наблюдения игнорируются

Таблица 2.7

Функции, которые можно использовать с аргументом `model=lm()`

summary()	Показывает детальную информацию по модели
coefficients()	Выводит оцененные параметры модели: свободный член и коэффициенты при предикторах
confint()	Вычисляет доверительные интервалы по параметрам модели (по умолчанию – 95 %)
fitted()	Выводит предсказанные значения отклика для обучающей выборки
residuals()	Выводит остатки для построенной модели
anova()	Создает таблицу дисперсионного анализа для построенной модели (если в качестве аргументов передается более одной модели – таблицу дисперсионного анализа сравнения моделей)
predict()	Строит предсказания для новых данных, кроме построенной модели, в качестве аргумента передается новый фрейм данных. Если передается обучающая выборка, то результат совпадает с функцией fitted()

Для оценки тесноты взаимосвязи между переменными используется функция `cor()`:

```
cor(x, method = c("pearson", "kendall", "spearman"))
```

Аргументы функции `cor()`: x – числовой вектор, матрица, фрейм данных (т.е. факторы нужно преобразовать с помощью функции `as.numeric()`), аргумент `method` по умолчанию имеет значение "pearson" (формула (1.18)).

2.3.1. Набор данных *медицинское страхование* (парная линейная регрессия)

Данные, которые мы используем, – расходы на медицинское обслуживание тех, кто имеет медицинскую страховку (см. 2.1.2).

Переменная отклика *charges* – медицинские расходы, оплачиваемые страховкой. Остальные переменные – предикторы. В качестве единственного предиктора рассмотрим показатель *bmi* – индекс массы тела.

```
#Импортируем данные
df_insurance<-read.table("insurance.csv", header=T, sep=",")
#Запишем в новый фрейм данных переменные charges и bmi
df_insurance1<-df_insurance[c(3,7)]
#Построим модель линейной регрессии по обучающей выборке
insurance_lm<-lm(charges~., data=df_insurance1)
#Выведем сводку по построенной модели
summary(insurance_lm)
#Выведем предсказания отклика на обучающей выборке
fitted(insurance_lm)
```

На рис. 2.16 представлена сводка по построенной модели линейной регрессии.

```
> summary(insurance_lm)
```

Call:
lm(formula = charges ~ ., data = df_insurance1)

Residuals:

Min	1Q	Median	3Q	Max
-20956	-8118	-3757	4722	49442

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1192.94	1664.80	0.717	0.474
bmi	393.87	53.25	7.397	2.46e-13 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11870 on 1336 degrees of freedom
Multiple R-squared: 0.03934, Adjusted R-squared: 0.03862
F-statistic: 54.71 on 1 and 1336 DF, p-value: 2.459e-13

Рис. 2.16. Сводка по модели парной линейной регрессии

Уравнение линейной регрессии согласно сводке имеет вид:

$$\widehat{charges} = 1192.94 + 393.87bmi. \quad (2.1)$$

То есть увеличение индекса массы тела на одну единицу в среднем увеличивает расходы на 393.87 денежных единиц. И хотя

$R^2 = 0.03934$, т.е. примерно 4 % изменения медицинских расходов связаны с предиктором bmi, тем не менее значение p-value говорит, что уравнение статистически значимо (вероятность верности гипотезы H_0 , что уравнение статистически незначимо 2.459×10^{-13}). Это связано с большим объемом обучающей выборки, когда даже в сильно зашумленных данных обнаруживаются не самые тесные связи.

#Построим линию регрессии на графике.

```
plot(df_insurance1)
```

```
abline(1192.94, 393.87, col='red')
```

На рис. 2.17 представлен результат выполнения кода по построению графиков.

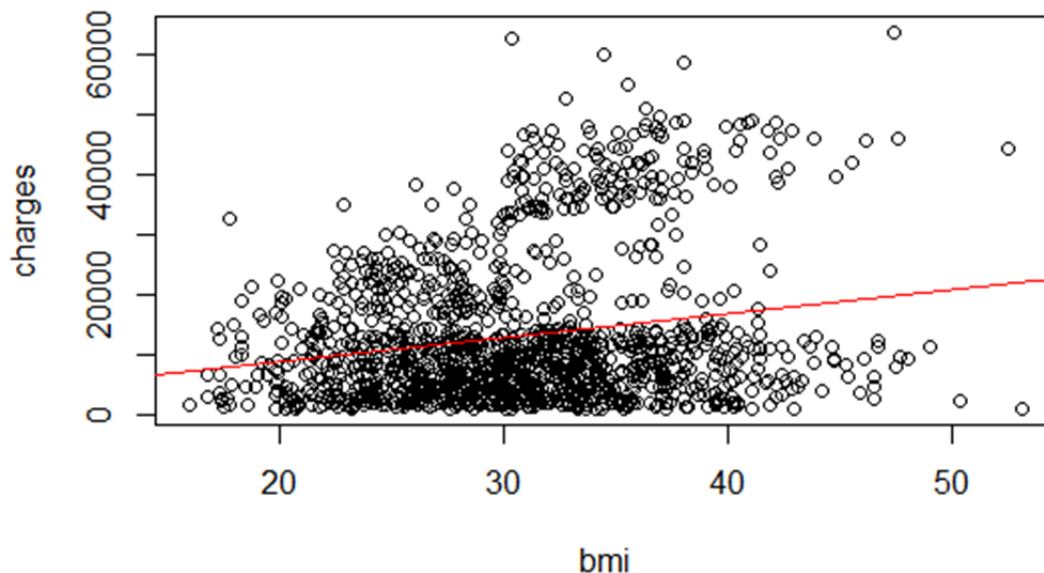


Рис. 2.17. Визуализация построенной регрессии

2.3.2. Набор данных *медицинское страхование* (множественная линейная регрессия)

Данные, которые мы используем, – расходы на медицинское обслуживание тех, кто имеет медицинскую страховку (см. 2.1.2).

Переменная отклика charges – медицинские расходы, оплачиваемые страховкой. Остальные переменные – предикторы.

#Импортируем данные

```
df_insurance<-read.table("insurance.csv", header=T, sep=",")
```

#Построим модель линейной регрессии по обучающей выборке

```
insurance_lm<-lm(charges~., data=df_insurance)
```

#Выведем сводку по построенной модели

```
summary(insurance_lm)
```

```
#Выведем предсказания отклика на обучающей выборке  
fitted(insurance_lm)
```

На рис. 2.18 представлена сводка по построенной модели линейной регрессии.

```
> summary(insurance_lm)
```

Call:
lm(formula = charges ~ ., data = df_insurance)

Residuals:

Min	1Q	Median	3Q	Max
-11304.9	-2848.1	-982.1	1393.9	29992.8

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-11938.5	987.8	-12.086	< 2e-16	***
age	256.9	11.9	21.587	< 2e-16	***
sexmale	-131.3	332.9	-0.394	0.693348	
bmi	339.2	28.6	11.860	< 2e-16	***
children	475.5	137.8	3.451	0.000577	***
smokeryes	23848.5	413.1	57.723	< 2e-16	***
regionnorthwest	-353.0	476.3	-0.741	0.458769	
regionsoutheast	-1035.0	478.7	-2.162	0.030782	*
regionsouthwest	-960.0	477.9	-2.009	0.044765	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6062 on 1329 degrees of freedom
Multiple R-squared: 0.7509, Adjusted R-squared: 0.7494
F-statistic: 500.8 on 8 and 1329 DF, p-value: < 2.2e-16

Рис. 2.18. Сводка по модели множественной линейной регрессии

Уравнение линейной регрессии содержит предикторы, которых не было в обучающей выборке: `sexmale`, `smokeryes`, `regionnorthwest`, `regionsouthwest`, `regionsoutheast` – это фиктивные (*dummy*) переменные, созданные на основе предикторов-факторов. Фактор с двумя уровнями значений заменялся на одну фиктивную переменную (в название переменной добавляется, какое значение берется за базовое), если у фактора k -уровней, то создается $k - 1$ фиктивная переменная.

Можно выполнить подобное преобразование самостоятельно, чтобы под рукой была матрица из преобразованных предикторов:

```
#Преобразуем факторные переменные с двумя уровнями в dummy  
df_insurance$sex<-ifelse(df_insurance$sex=='female', 0, 1)  
df_insurance$smoker<-ifelse(df_insurance$smoker=='no', 0, 1)  
#Создаем три новые dummy переменные для трех уровней  
#переменной region  
df_insurance$region_nw<-ifelse(df_insurance$region=='northwest', 1, 0)
```

```
df_insurance$region_sw<-ifelse(df_insurance$region=='southwest', 1, 0)
df_insurance$region_se<-ifelse(df_insurance$region=='southeast', 1, 0)
#Удаляем переменную region из обучающей выборки
df_insurance$region<-NULL
#Построим модель линейной регрессии по преобразованным данным
insurance_lm<-lm(charges~., data=df_insurance)
summary(insurance_lm)
```

Сводка по модели для преобразованных данных (рис. 2.19) совпадает со сводкой, когда преобразование в фиктивные переменные было проведено внутри модели (см. рис. 2.18).

```
> summary(insurance_lm)
```

Call:
lm(formula = charges ~ ., data = df_insurance)

Residuals:

	Min	1Q	Median	3Q	Max
	-11304.9	-2848.1	-982.1	1393.9	29992.8

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	-11938.5	987.8	-12.086	< 2e-16	***
age	256.9	11.9	21.587	< 2e-16	***
sex	-131.3	332.9	-0.394	0.693348	
bmi	339.2	28.6	11.860	< 2e-16	***
children	475.5	137.8	3.451	0.000577	***
smoker	23848.5	413.1	57.723	< 2e-16	***
region_nw	-353.0	476.3	-0.741	0.458769	
region_sw	-960.0	477.9	-2.009	0.044765	*
region_se	-1035.0	478.7	-2.162	0.030782	*

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6062 on 1329 degrees of freedom
Multiple R-squared: 0.7509, Adjusted R-squared: 0.7494
F-statistic: 500.8 on 8 and 1329 DF, p-value: < 2.2e-16

Рис. 2.19. Сводка по модели множественной линейной регрессии

Как видим, теперь $R^2 = 0.7509$, т.е. примерно 75 % изменения медицинских расходов связаны с включенными предикторами. Правда, не все они одинаково полезны для предсказания отклика. Предиктор sex – статистически незначимый и на расходы не влияет. Интерпретация влияния региона: возможно влияние (не самое существенное) оказывают северное и южное направления, а вот разбиение на четыре градации было лишним. Построим модель, исключив переменную sex и объединив северные регионы в один уровень, а южные – в другой.

```
#Преобразуем факторные переменные с двумя уровнями в dummy
df_insurance$smoker<-ifelse(df_insurance$smoker=='no', 0, 1)
#Используем оператор логического или '|' (логическое и '&', логическое не '!')
df_insurance$region_s<-
ifelse(df_insurance$region=='southwest' | df_insurance$region=='southeast', 1, 0)
#Исключим из набора данных переменные sex и region
df_insurance$region<-NULL
df_insurance$sex<-NULL
#Построим модель линейной регрессии по преобразованным данным
insurance_lm<-lm(charges~., data=df_insurance)
summary(insurance_lm)
```

Сводка по модели для преобразованных данных (рис. 2.20) показывает, что выполненные преобразования пошли модели на пользу: все предикторы, включенные в модель, имеют статистически значимое влияние на результат. Показатель Adjusted R^2 (скорректированный коэффициент детерминации) вырос. Его рост свидетельствует, что исключенные предикторы только мешали.

```
> summary(insurance_lm)

Call:
lm(formula = charges ~ ., data = df_insurance)

Residuals:
    Min       1Q   Median       3Q      Max
-11342.9  -2843.3   -978.6   1359.8  29893.1

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -12144.70     940.46  -12.914  < 2e-16 ***
age           257.06       11.88   21.639  < 2e-16 ***
bmi           337.86       28.13   12.012  < 2e-16 ***
children      472.20      137.55    3.433 0.000615 ***
smoker       23839.60     410.65   58.053  < 2e-16 ***
region_s      -820.40     341.16   -2.405 0.016318 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6057 on 1332 degrees of freedom
Multiple R-squared:  0.7508,    Adjusted R-squared:  0.7498
F-statistic: 802.5 on 5 and 1332 DF,  p-value: < 2.2e-16
```

Рис. 2.20. Сводка по модели множественной линейной регрессии

В библиотеке `gvlma` есть одноименная функция, которая проверяет, удовлетворяют ли обучающие данные исходным предпосылкам модели. Это сводный параметр, для которого тестируется гипотеза H_0 – все предпосылки выполняются, если p -value меньше, чем 0.05,

то гипотеза отклоняется и это означает, что какие-то предпосылки нарушены и необходимо проверять их по отдельности, чтобы выяснить суть нарушения и применить какой-то прием, метод, который позволит улучшить качество модели.

```
install.packages('gvlma', dependencies=TRUE)
```

```
library(gvlma)
```

```
gvlm<-gvlma(insurance_lm)
```

2.4. Линейный классификатор (логистическая регрессия)

В R для построения модели логистической регрессии можно воспользоваться функцией `glm()` из библиотеки `stats`, которая по умолчанию подключена в среде RStudio (аргументы функции – в табл. 2.8).

```
fit <- glm(formula, data, family=binomial(link='logit'))
```

Таблица 2.8

Аргументы функции `glm()`

formula	$y \sim x_1 + x_2 + x_3$ Если по умолчанию в модель линейной регрессии должны войти все предикторы, можно записать $y \sim$. Знак '+' используется для разделения предикторов, знак '-' используется для исключения предикторов из модели, если нужно исключить свободный член, то -1
data	Матрица или фрейм данных предикторов и отклика
family	Для логистической регрессии этот параметр задается вероятностным распределением <code>binomial()</code> и связью <code>link='logit'</code>

Данные, которые используем для построения логистической регрессии, – отток клиентов оператора связи (см. 2.1.1).

Переменная отклика `churn` – отток абонентов. Остальные переменные – предикторы.

```
#Импортируем данные
```

```
df_churn<-read.table("Churn_binned.csv", header=T, sep=";")
```

```
#Выполним преобразование для порядковых предикторов
```

```
#упорядочим уровни
```

```
ord_longdist <- c("<2", "2--8", "9--14", "15-20", "21+")
```

```
df_churn$longdist <- ordered(df_churn$longdist, levels = ord_longdist)
```

```
ord_local <- c("<8", "8--20", "21-35", "36-55", "56+")
```

```
df_churn$local <- ordered(df_churn$local, levels = ord_local)
```

```
ord_incomecat <- c("<15", "15-35", "36-50", "51-65", "66+")
df_churn$incomecat <- ordered(df_churn$incomecat, levels = ord_incomecat)
ord_agecat <- c("<31", "31-45", "46-58", "59-70", "71+")
df_churn$agecat <- ordered(df_churn$agecat, levels = ord_agecat)
#Для девяти предикторов выполним преобразование типа на numeric
for(i in c(1:9)) df_churn[,i] <- as.numeric(df_churn[,i])
#Построим логистическую регрессию по всем предикторам
churn_glm<-glm(churn~., data=df_churn, family=binomial(link='logit'))
summary(churn_glm)
```

Сводка по модели логистической регрессии показана на рис. 2.21.

```
> summary(churn_glm)

Call:
glm(formula = churn ~ ., family = binomial(link = "logit"), data = df_churn)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0768  -0.7349  -0.4501   0.8834   2.4816

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  4.76724    0.31174  15.292 < 2e-16 ***
longdist    -0.33982    0.02888 -11.768 < 2e-16 ***
local       -0.26673    0.02957  -9.022 < 2e-16 ***
int_disc     0.07461    0.07764   0.961  0.3365
billtype    -0.07714    0.07205  -1.071  0.2843
pay          0.11540    0.05539   2.083  0.0372 *
gender      -2.31400    0.07699 -30.055 < 2e-16 ***
marital      0.01929    0.07335   0.263  0.7925
incomecat   -0.16094    0.02776  -5.798 6.72e-09 ***
agecat       0.13002    0.02689   4.835 1.33e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6071.5  on 4430  degrees of freedom
Residual deviance: 4659.2  on 4421  degrees of freedom
AIC: 4679.2
```

Рис. 2.21. Сводка по модели логистической регрессии

Есть предикторы, которые можно исключить из модели. Построим логистическую регрессию только по тем переменным, значимость которых подтверждена на уровне более 99 %.

```
churn_glm1<-glm(churn~longdist+local+gender+incomecat+agecat,
                data=df_churn, family=binomial(link='logit'))
summary(churn_glm1)
```

Сводка по модели логистической регрессии с меньшим числом предикторов изображена на рис. 2.22.

```

> summary(churn_glm1)

Call:
glm(formula = churn ~ longdist + local + gender + incomecat +
     agecat, family = binomial(link = "logit"), data = df_churn)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0426  -0.7392  -0.4555   0.8859   2.4689

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)   5.02856    0.20888  24.074 < 2e-16 ***
longdist     -0.34007    0.02879 -11.814 < 2e-16 ***
local        -0.26577    0.02952  -9.004 < 2e-16 ***
gender       -2.30605    0.07677 -30.038 < 2e-16 ***
incomecat    -0.16054    0.02773  -5.790 7.04e-09 ***
agecat        0.12792    0.02680   4.773 1.81e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6071.5  on 4430  degrees of freedom
Residual deviance: 4665.8  on 4425  degrees of freedom
AIC: 4677.8

```

Рис. 2.22. Сводка по модели логистической регрессии

Проведем дисперсионный анализ для проверки гипотезы: исключенные переменные не влияли на качество предсказаний оттока. Это возможно, так как модель `churn_glm1` является вложенной в модель `churn_glm` (если пересечение предикторов двух моделей не совпадает с предикторами одной из них, то проводить такой анализ нельзя, такие модели несравнимы).

```
anova(churn_glm, churn_glm1, test='Chisq')
```

На рис. 2.23 результаты сравнения двух моделей и значение статистики для теста хи-квадрат, большее 0.05, подтверждают корректность использования для предсказаний модели с меньшим числом предикторов.

```

> anova(churn_glm, churn_glm1, test='Chisq')
Analysis of Deviance Table

Model 1: churn ~ longdist + local + int_disc + billtype + pay + gender +
  marital + incomecat + agecat
Model 2: churn ~ longdist + local + gender + incomecat + agecat
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1      4421      4659.2
2      4425      4665.8 -4   -6.5543   0.1614

```

Рис. 2.23. Дисперсионный анализ для сравнения качества двух моделей логистической регрессии

3. Обучение с учителем на языке Python, Anaconda, Jupyter Notebook

Для выполнения анализа с помощью языка Python установим Anaconda с официального сайта: URL: <https://www.anaconda.com/distribution/>. На момент написания пособия актуальна версия Anaconda для Python 3.7 [12].

В Python большинство методов машинного обучения находятся в библиотеке `scikit-learn` [13], которая импортируется как `sklearn`.

Два основных метода, которые используются в классах библиотеки `scikit-learn`, реализующих различные алгоритмы обучения:

- `fit(features, labels)` – этап обучения (`features` – это предикторы, признаки обучающего пространства; `labels` – отклик, метки наблюдений);
- `predict(features)` – вызывается только после `fit()` и возвращает предсказание для входных данных предикторов (`features`).

3.1. Деревья решений

В Python для построения деревьев необходимо импортировать из класса `sklearn.tree` (библиотека `scikit-learn`) класс `DecisionTreeClassifier` для построения дерева классификации и класс `DecisionTreeRegressor` для построения дерева регрессии (параметры класса – в табл. 3.1, методы класса – в табл. 3.2).

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
```

Таблица 3.1

Параметры классов `DecisionTreeClassifier()` и `DecisionTreeRegressor()`

criterion	Функция для измерения качества разделения. Для <code>DecisionTreeClassifier()</code> возможные значения – “gini”, “entropy”. Для <code>DecisionTreeRegressor()</code> возможные значения – “mse”, “friedman_mse”, “mae”
splitter	Стратегия, используемая для выбора разделения на каждом узле. Возможные значения “best” (выбирается наилучшее расщепление), “random” (выбирается наилучшее случайное расщепление)

max_depth	Максимальная глубина дерева. Если не задана, то расщепление продолжается до глубины, на которой все терминальные узлы имеют энтропию, равную нулю (состоят из наблюдений одного значения отклика), либо пока в терминальных узлах не окажется min_samples_split наблюдений
min_samples_split	Минимальное количество наблюдений, необходимое для разделения внутреннего узла. Значение по умолчанию равно двум
min_samples_leaf	Минимальное количество наблюдений, требуемое для размещения в конечном узле. Точка разделения на любой глубине будет рассматриваться только в том случае, если она выходит на наименьшее min_samples_leaf количество наблюдений в каждом из узлов (левом и правом), получающихся при расщеплении. Значение по умолчанию равно двум
max_features	Максимальное число предикторов, по которым ищется лучшее разбиение в дереве (это нужно потому, что при большом количестве признаков будет "дорого" искать лучшее (по критерию типа прироста информации) разбиение среди всех признаков)

Таблица 3.2

Методы классов DecisionTreeClassifier() и DecisionTreeRegressor()

fit(X, y)	Выполняет построение дерева решений по предикторам фрейма данных X и отклику y
predict(X)	Прогнозирует значение отклика для каждого наблюдения данных X

Для построения дерева используется функция export_graphviz() из библиотеки scikit-learn. Ее параметры в табл. 3.3.

Повторим реализацию построение деревьев решений на языке Python на тех же данных, что и на языке R.

Таблица 3.3

Параметры функции export_graphviz()

decision_tree	Дерево решений, которое будет экспортировано в GraphViz
out_file	Дескриптор или имя выходного файла
max_depth	Максимальная глубина представления. Если нет, то дерево строится полностью

feature_names	Задаёт названия предикторов
class_names	Задаёт имена классов зависимой переменной
label	Значение по умолчанию False, если True выводит мажоритарный класс для наблюдений узла в дереве классификации и среднее значение отклика для наблюдений узла в дереве регрессии
leaves_parallel	Значение по умолчанию False, если True рисует в нижней части дерева все терминальные узлы

3.1.1. Набор данных оператора связи (дерево классификации)

Код notebook для подключения библиотек – рис. 3.1.

```
# Импортируем библиотеки
import numpy as np
import pandas as pd
%matplotlib inline
import seaborn as sns
from matplotlib import pyplot as plt
```

Рис. 3.1. Подключение библиотек

Файл с данными оператора связи можно скачать из репозитория <https://github.com/Smeilz/ML-Class/tree/master/R/Trees> имя файла Churn_binned.csv. Файл с данными Churn_binned.csv должен находиться в той же директории, что и папка .anaconda.

Импортируем данные (рис. 3.2).

```
churn_data = pd.read_csv('Churn_binned.csv', ';')
churn_data.head()
```

Рис. 3.2. Импорт данных

При загрузке данных возвращается сообщение об ошибке: UnicodeDecodeError: 'utf-8' codec can't decode byte 0xcd in position 0: invalid continuation byte.

Это сообщение информирует, что загружаемый файл не имеет кодировку 'utf-8' и не может быть считан. Для изменения кодировки можно воспользоваться open source текстовым редактором Notepad++ (или другим редактором, который имеет необходимую функцию конвертации в кодировку UTF-8). Дистрибутив Notepad++ можно скачать по ссылке URL: <https://notepad-plus-plus.org/downloads/>

Воспользуемся Notepad++ также для замены знака ‘<’ на слово ‘меньше’ (Search>Replace). В дальнейшем будут даны пояснения, почему это было необходимо.

На рис. 3.3 для выделенного текста файла изменяется кодировка (Encoding>Convert to UTF-8). Затем файл сохраняется под именем Churn_binned.csv.

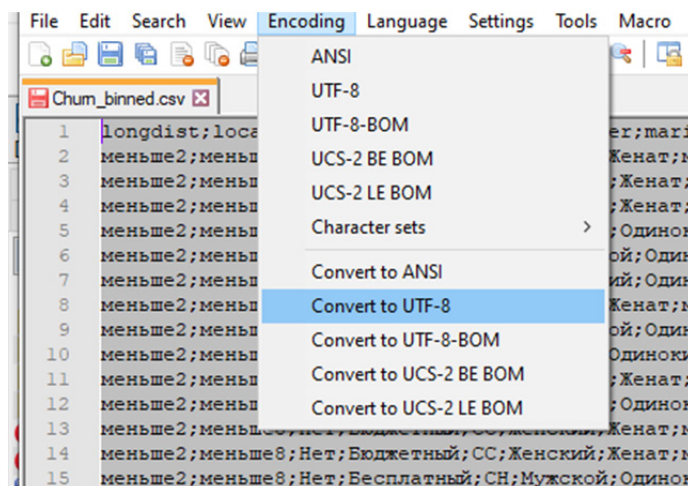


Рис. 3.3. Изменение кодировки средствами Notepad++

Теперь выполним код для импорта данных повторно (рис. 3.4).

```
In [35]: churn_data = pd.read_csv('Churn_binned.csv', ';')
churn_data.head()
```

Out[35]:

	longdist	local	int_disc	billtype	pay	gender	marital	incomecat	agecat	churn
0	меньше2	меньше8	Нет	Бюджетный	СС	Мужской	Женат	меньше15	меньше31	Уходит
1	меньше2	меньше8	Нет	Бесплатный	СС	Мужской	Женат	меньше15	меньше31	Уходит
2	меньше2	меньше8	Нет	Бесплатный	СС	Мужской	Женат	меньше15	меньше31	Уходит
3	меньше2	меньше8	Нет	Бесплатный	СН	Мужской	Одинокий	меньше15	меньше31	Остается
4	меньше2	меньше8	Нет	Бесплатный	Auto	Мужской	Одинокий	меньше15	меньше31	Уходит

Рис. 3.4. Импорт данных

Создадим новые фреймы данных: X – для предикторов; y – для отклика, и выведем информацию о предикторах (рис. 3.5).

Все предикторы набора данных, характеризующих абонентов, являются категориальными (номинальными или порядковыми). Для построения дерева необходимо, чтобы текстовые значения были заменены числовыми метками. В библиотеке pandas [14] есть метод `factorize()`, который позволяет преобразовать столбец с текстовыми данными в числовой массив. Для номинальной переменной этот метод подходит. Однако для порядковой он приводит к утрате информации об упорядочении уровней (так как метки присваиваются

в лексикографическом упорядочении, что не всегда соответствует реальному следованию уровней). Есть возможность преобразовать категориальную текстовую переменную в числовой формат с помощью словаря и метода `map()` (рис. 3.6).

```
In [36]: #Сохраняем в новый фрейм данных столбцы предикторов
X=churn_data.drop(['churn'], axis=1)
#Сохраняем в переменную y отклик
y=churn_data.churn
#Информация о структуре X
X.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4431 entries, 0 to 4430
Data columns (total 9 columns):
longdist      4431 non-null object
local         4431 non-null object
int_disc      4431 non-null object
billtype      4431 non-null object
pay           4431 non-null object
gender        4431 non-null object
marital       4431 non-null object
incomecat     4431 non-null object
agecat        4431 non-null object
dtypes: object(9)
memory usage: 311.7+ KB
```

Рис. 3.5. Преобразование данных

```
In [2]: shop1_data = pd.read_csv('shop1.csv', ';')
shop1_data.head()
```

Out[2]:

	gender	age	totalsum	discount
0	f	32	25412	y
1	f	25	31258	y
2	f	47	17896	n
3	f	29	62300	y
4	f	38	27366	y

```
In [3]: shop1_data['gender']=shop1_data['gender'].map({'m': 0, 'f': 1})
shop1_data.head()
```

Out[3]:

	gender	age	totalsum	discount
0	1	32	25412	y
1	1	25	31258	y
2	1	47	17896	n
3	1	29	62300	y
4	1	38	27366	y

Рис. 3.6. Преобразование категориальной переменной gender

Преобразуя так категориальную переменную, мы не всегда даем верный сигнал, что предиктор – фактор и его промежуточные

значения бессмысленны, а разница в уровнях не количественная, а качественная.

В библиотеке `pandas` есть метод `get_dummies()`, который позволяет преобразовать категориальную переменную в фиктивные (или манекены, заместители). Эти переменные принимают только два значения 0/1 и описывают каждый уровень категориальной переменной отдельно. Параметры метода `get_dummies()` даны в табл. 3.4.

Таблица 3.4

Параметры функции `get_dummies()`

<code>data</code>	Данные для преобразования
<code>dummy_na</code>	Значение по умолчанию <code>False</code> , если <code>True</code> не игнорирует <code>NaN</code> , а определяет еще одну переменную для пропущенных значений
<code>drop_first</code>	Значение по умолчанию <code>False</code> , если <code>True</code> , то получаем $k - 1$ фиктивную переменную для k уровней категориального признака

Если не определить параметр `drop_first=True`, то преобразование для категориальной переменной создаст столько столбцов, сколько категорий (рис. 3.7). Очевидно, что `gender_f` не содержит новой информации, так как если `gender_m=0`, то очевидно, что пол респондента – женский. Следовательно, уровень категориальной переменной с номером k определяется через равенство нулю одновременно всех $k - 1$ фиктивных переменных, определяющих остальные уровни. Поэтому при преобразовании категориальной переменной в фиктивные в дальнейшем будем использовать параметр `drop_first=True`.

Выполним преобразование предикторов (рис. 3.8).

Построим дерево классификации (рис. 3.9). В выводе есть сводка по значениям параметров построенного дерева.

В библиотеке `scikit-learn` сводку по дереву можно получить в графическом формате. Однако возможности, реализованные в самой библиотеке, не всегда дают стабильный результат. Поэтому установим дополнительно библиотеку `pydotplus`.

Инсталляцию выполним с помощью `Anaconda Prompt` (входит в дистрибутив `Anaconda`):

код после приглашения командной строки:

```
conda install -c conda-forge pydotplus
```

После успешной загрузки всех необходимых обновлений подтверждаем в командной строке: установить библиотеку (ответом на вопрос о согласии на установку ‘y’ – да).

```
In [4]: shop1_data = pd.read_csv('shop1.csv', ';')
shop1_data.head()
```

```
Out[4]:
```

	gender	age	totalsum	discount
0	f	32	25412	y
1	f	25	31258	y
2	f	47	17896	n
3	f	29	62300	y
4	f	38	27366	y

```
In [5]: shop1_data = pd.get_dummies(shop1_data)
shop1_data.head()
```

```
Out[5]:
```

	age	totalsum	gender_f	gender_m	discount_n	discount_y
0	32	25412	1	0	0	1
1	25	31258	1	0	0	1
2	47	17896	1	0	1	0
3	29	62300	1	0	0	1
4	38	27366	1	0	0	1

Рис. 3.7. Преобразование категориальных переменных gender и discount

```
In [37]: #Преобразуем категориальные переменные в dummy
X=pd.get_dummies(X, drop_first=True)
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4431 entries, 0 to 4430
Data columns (total 22 columns):
longdist_2--8      4431 non-null uint8
longdist_21+      4431 non-null uint8
longdist_9--14    4431 non-null uint8
longdist_меньше2  4431 non-null uint8
local_36-55       4431 non-null uint8
local_56+         4431 non-null uint8
local_8--20       4431 non-null uint8
local_меньше8     4431 non-null uint8
int_disc_Нет      4431 non-null uint8
billtype_Бюджетный 4431 non-null uint8
pay_CC            4431 non-null uint8
pay_CN            4431 non-null uint8
gender_Мужской    4431 non-null uint8
marital_Одинокий  4431 non-null uint8
incomecat_36-50   4431 non-null uint8
incomecat_51-65   4431 non-null uint8
incomecat_66+     4431 non-null uint8
incomecat_меньше15 4431 non-null uint8
agecat_46-58      4431 non-null uint8
agecat_59-70      4431 non-null uint8
agecat_71+        4431 non-null uint8
agecat_меньше31   4431 non-null uint8
dtypes: uint8(22)
memory usage: 95.3 KB
```

Рис. 3.8. Преобразование категориальных переменных в фиктивные

```

In [47]: #Импортируем класс DecisionTreeClassifier из библиотеки sklearn
from sklearn.tree import DecisionTreeClassifier
#Создаем экземпляр класса DecisionTreeClassifier() задав параметры модели
clf=DecisionTreeClassifier(criterion='entropy', max_depth=5)
#Строим дерево
clf.fit(X,y)

Out[47]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort=False,
                                random_state=None, splitter='best')

```

Рис. 3.9. Построение дерева классификации

Далее в Jupyter Notebook выполняем код (рис. 3.10), подключающий библиотеку pydotplus, а также загружаем необходимые классы и методы из встроенных в Anaconda библиотек.

```

In [39]: import pydotplus

In [40]: from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz

```

Рис. 3.10. Импорт библиотек, методов и классов

Выполняем визуализацию построенного дерева (рис. 3.11) с помощью функции `export_graphviz()` (параметры функции описаны в табл. 3.3).

Так как в функции `export_graphviz()` была определена максимальная глубина визуализации, равная трем, а построенное дерево имеет глубину пять, то отсутствующие в визуализации узлы отображены серыми прямоугольниками без числовых параметров узла. По умолчанию отобразились бы все ветви и все узлы построенного дерева.

Для фиктивных переменных значение меньше 0.5 означает 0 для левой ветви True (истина) и 1 – для правой ветви False (ложь). Так, левая часть дерева от корневого узла соответствует абонентам женщинам, правая – абонентам мужчинам.

При изменении кодировки в Notepad++ также была проведена замена знака '<' на слово 'меньше' (Search>Replace). Это связано с визуализацией дерева. Знаки неравенства зарезервированы для правил разбиения, и если такой знак встречается в названии столбца, то это становится препятствием для корректного построения дерева и

выводится сообщение об ошибке. Изначально в названии столбцов знак ‘<’ отсутствовал. Однако после преобразования категориальных переменных в фиктивные он бы там появился, так как название столбца для фиктивной переменной – это объединение первоначального названия переменной и названия уровня переменной, для которого строится фиктивная переменная. Чтобы избежать этой коллизии и на последнем этапе не столкнуться с необходимостью переименования столбцов, были предприняты шаги по избавлению от знака неравенства в уровнях категориальных переменных.

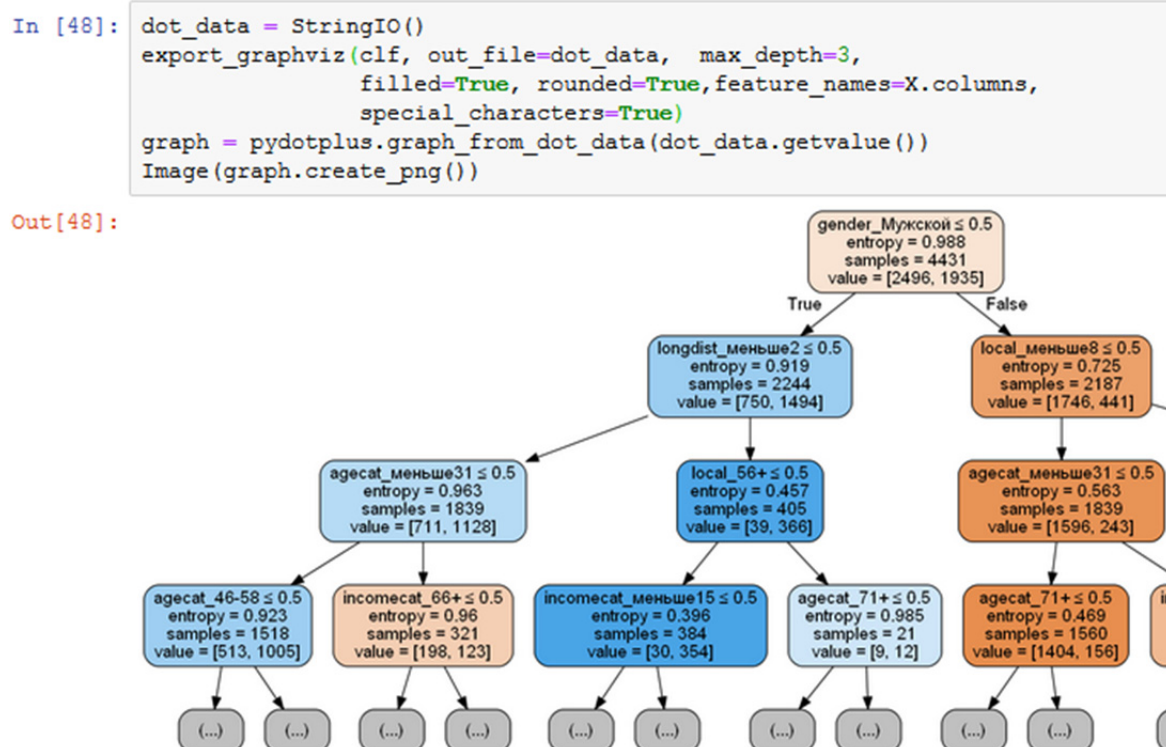


Рис. 3.11. Визуализация дерева классификации (фрагмент)

Возвращение предсказаний показано на рис. 3.12. Для получения предсказаний по новым данным вместо X нужно использовать фрейм предикторов с тем же числом столбцов и их названиями.

```
In [49]: pred = clf.predict(X)
pred
```

Out[49]: array(['Уходит', 'Уходит', 'Уходит', ..., 'Уходит', 'Уходит', 'Уходит'],
dtype=object)

Рис. 3.12. Предсказания с использованием дерева классификации (для исходных значений предикторов)

3.1.2. Набор данных *медицинское страхование* (дерево регрессии)

Второй набор данных – набор данных медицинское страхование (см. 2.2.2). Файл с данными `insurance.csv` должен находиться в той же директории, что и папка `.anaconda`.

Код notebook для подключения библиотек на рис. 3.13.

```
# Импортируем библиотеки
import numpy as np
import pandas as pd
%matplotlib inline
import seaborn as sns
from matplotlib import pyplot as plt
```

Рис. 3.13. Подключение библиотек

Загрузка набора данных показана на рис. 3.14.

In [51]:	<pre>insurance_data = pd.read_csv('insurance.csv', ',') insurance_data.head()</pre>																																																						
Out[51]:	<table><tr><th></th><th>age</th><th>sex</th><th>bmi</th><th>children</th><th>smoker</th><th>region</th><th>charges</th></tr><tr><td>0</td><td>19</td><td>female</td><td>27.900</td><td>0</td><td>yes</td><td>southwest</td><td>16884.92400</td></tr><tr><td>1</td><td>18</td><td>male</td><td>33.770</td><td>1</td><td>no</td><td>southeast</td><td>1725.55230</td></tr><tr><td>2</td><td>28</td><td>male</td><td>33.000</td><td>3</td><td>no</td><td>southeast</td><td>4449.46200</td></tr><tr><td>3</td><td>33</td><td>male</td><td>22.705</td><td>0</td><td>no</td><td>northwest</td><td>21984.47061</td></tr><tr><td>4</td><td>32</td><td>male</td><td>28.880</td><td>0</td><td>no</td><td>northwest</td><td>3866.85520</td></tr></table>								age	sex	bmi	children	smoker	region	charges	0	19	female	27.900	0	yes	southwest	16884.92400	1	18	male	33.770	1	no	southeast	1725.55230	2	28	male	33.000	3	no	southeast	4449.46200	3	33	male	22.705	0	no	northwest	21984.47061	4	32	male	28.880	0	no	northwest	3866.85520
	age	sex	bmi	children	smoker	region	charges																																																
0	19	female	27.900	0	yes	southwest	16884.92400																																																
1	18	male	33.770	1	no	southeast	1725.55230																																																
2	28	male	33.000	3	no	southeast	4449.46200																																																
3	33	male	22.705	0	no	northwest	21984.47061																																																
4	32	male	28.880	0	no	northwest	3866.85520																																																

Рис. 3.14. Загрузка данных для анализа

Создадим новые фреймы данных: X – для предикторов, y – для отклика, и выведем информацию о предикторах (рис. 3.15).

Выполним преобразование предикторов (рис. 3.16).

Построим дерево регрессии (рис. 3.17). В выводе есть сводка по значениям параметров построенного дерева.

Выполняем код (рис. 3.18), подключающий библиотеку `pydot-plus`, а также загружаем необходимые классы и методы из встроенных в Anaconda библиотек.

Выполняем визуализацию построенного дерева (рис. 3.19) с помощью функции `export_graphviz()` (параметры функции описаны в табл. 3.3).

```
In [52]: #Сохраняем в новый фрейм данных столбцы предикторов
X=insurance_data.drop(['charges'], axis=1)
#Сохраняем в переменную y отклик
y=insurance_data.charges
#Информация о структуре X
X.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 6 columns):
age          1338 non-null int64
sex          1338 non-null object
bmi          1338 non-null float64
children     1338 non-null int64
smoker       1338 non-null object
region       1338 non-null object
dtypes: float64(1), int64(2), object(3)
memory usage: 62.8+ KB
```

Рис. 3.15. Преобразование данных

```
In [53]: #Преобразуем категориальные переменные в dummy
X=pd.get_dummies(X, drop_first=True)
X.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 8 columns):
age          1338 non-null int64
bmi          1338 non-null float64
children     1338 non-null int64
sex_male     1338 non-null uint8
smoker_yes   1338 non-null uint8
region_northwest 1338 non-null uint8
region_southeast 1338 non-null uint8
region_southwest 1338 non-null uint8
dtypes: float64(1), int64(2), uint8(5)
memory usage: 38.0 KB
```

Рис. 3.16. Преобразование категориальных переменных в фиктивные

```
In [55]: #Импортируем класс DecisionTreeRegressor из библиотеки sklearn
from sklearn.tree import DecisionTreeRegressor
#Создаем экземпляр класса DecisionTreeClassifier() задав параметры модели
reg=DecisionTreeRegressor(criterion='mse', max_depth=5)
#Строим дерево
reg.fit(X,y)

Out[55]: DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
                                max_leaf_nodes=None, min_impurity_decrease=0.0,
                                min_impurity_split=None, min_samples_leaf=1,
                                min_samples_split=2, min_weight_fraction_leaf=0.0,
                                presort=False, random_state=None, splitter='best')
```

Рис. 3.17. Построение дерева регрессии

Возвращение предсказаний представлено на рис. 3.20. Для получения предсказаний по новым данным вместо X нужно использовать фрейм предикторов с тем же числом столбцов и их названиями.

```
In [56]: import pydotplus
```

```
In [57]: from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
```

Рис. 3.18. Импорт библиотек, методов и классов

```
In [58]: dot_data = StringIO()
export_graphviz(reg, out_file=dot_data, max_depth=3,
               filled=True, rounded=True, feature_names=X.columns,
               special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out [58]:

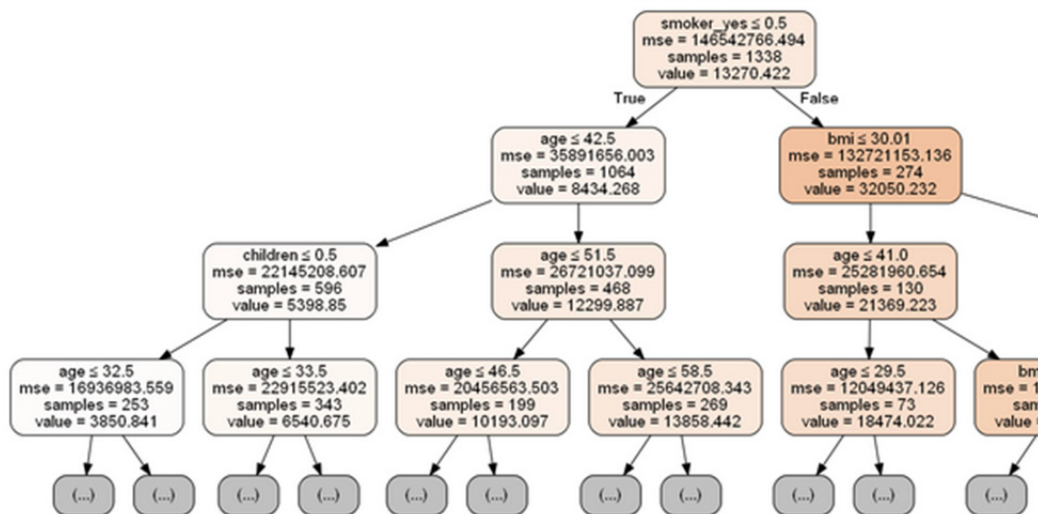


Рис. 3.19. Визуализация дерева регрессии (фрагмент)

```
In [59]: pred = reg.predict(X)
pred
```

```
Out[59]: array([16674.48305882, 6449.22794132, 6449.22794132, ...,
                2585.69774966, 2585.69774966, 26760.986315 ])
```

Рис. 3.20. Предсказания с использованием дерева регрессии (для исходных значений предикторов)

3.2. Метод ближайших соседей

В Python метод ближайших соседей реализуется в классах `KNeighborsClassifier` (для классификации) и `KNeighborsRegressor` (для регрессии), которые импортируются из класса `sklearn.neighbors` (библиотека `scikit-learn`) (параметры класса – в табл. 3.5, методы класса – в табл. 3.6).

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
```

Таблица 3.5

Параметры классов KNeighborsClassifier() и KNeighborsRegressor()

n_neighbors	Количество соседей (по умолчанию равно 5)
weightsstr	Весовая функция, используемая в прогнозировании. Возможное значение: <ul style="list-style-type: none"> – ‘uniform’ равномерные веса. Все точки в каждом районе взвешиваются одинаково; – ‘distance’: весовые точки, обратные их расстоянию. В этом случае более близкие соседи точки запроса будут иметь большее влияние, чем у соседей, которые находятся дальше; – [callable] определяемая пользователем функция, которая принимает массив расстояний и возвращает массив той же формы, содержащий весовые коэффициенты
algorithm	Алгоритм, используемый для вычисления ближайших соседей

Таблица 3.6

Методы классов KNeighborsClassifier() и KNeighborsRegressor()

fit(X_train, y_train)	Передаёт в классификатор обучающие данные: предикторы – фрейм данных X_train и отклик y_train
predict(X_test)	Прогнозирует значение отклика для каждого наблюдения данных X_test
predict_proba(X_test)	Оценки вероятности для тестовых данных

В Python для разбиения исходного набора данных на тестовые и тренировочные необходимо импортировать из класса sklearn.model_selection (библиотека scikit-learn) класс train_test_split (параметры класса в табл. 3.7).

```
from sklearn.model_selection import train_test_split
```

Таблица 3.7

Параметры класса train_test_split ()

df	Фрейм данных для разбиения
test_size	Доля наблюдений для тестовой выборки (по умолчанию 0.25)
shuffle	True – нужно перемешать данные перед разбиением, False – нет

Повторим реализацию метода ближайших соседей на языке Python на тех же данных, что и на языке R (см. 2.2).

3.2.1. Набор данных оператора связи (классификация)

Файл с данными Churn_binned.csv должен находиться в той же директории, что и папка .anaconda.

Код notebook для подключения библиотеки и загрузки данных – на рис. 3.21.

```
# Импортируем библиотеки
import pandas as pd

churn_data = pd.read_csv('Churn_binned.csv', ';')
churn_data.head()
```

Рис. 3.21. Подключение библиотеки и импорт данных

Создадим новые фреймы данных: X – для предикторов, y – для отклика, и преобразуем категориальные переменные в фиктивные (рис. 3.22).

```
#Сохраняем в новый фрейм данных столбцы предикторов
X=churn_data.drop(['churn'], axis=1)
#Сохраняем в переменную y отклик
y=churn_data.churn

#Преобразуем категориальные переменные в dummy
X=pd.get_dummies(X, drop_first=True)
```

Рис. 3.22. Преобразование данных

Выполним разбиение данных на тренировочную и тестовую части (рис. 3.23). Доля тестовых данных – 30 %.

```
#Проведем разбиение исходных наблюдений предикторов и отклика
#на обучающую выборку и тестовую
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Рис. 3.23. Разбиение данных

Передадим в классификатор тренировочные данные (рис. 3.24).

```
In [23]: knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)

Out[23]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=10, p=2,
weights='uniform')
```

Рис. 3.24. Создание объекта класса KNeighborsClassifier

Найдем предсказания оттока для тестового набора данных методом ближайших соседей и сформируем таблицу X_test1, включающую предикторы, отклик и предсказание (рис. 3.25).

```
knn_pred = knn.predict(X_test)
X_test1 = X_test
X_test1['knn'] = knn_pred
X_test1['churn'] = y_test
X_test1.head()
```

Рис. 3.25. Формирование набора данных с предсказаниями и откликом на основе тестовых данных

Фрагмент таблицы с предсказаниями на тестовых данных изображен на рис. 3.26.

incomecat_36-50	incomecat_51-65	incomecat_66+	incomecat_меньше15	agecat_46-58	agecat_59-70	agecat_71+	agecat_меньше31	knn	churn
0	0	0	0	0	1	0	0	Уходит	Уходит
1	0	0	0	0	0	1	0	Остается	Остается
0	0	0	1	0	1	0	0	Остается	Остается
0	0	0	1	1	0	0	0	Остается	Уходит
0	0	0	1	0	0	1	0	Остается	Остается

Рис. 3.26. Тестовые данные с предсказанным значением оттока по методу kNN (фрагмент)

С помощью класса accuracy_score() из класса sklearn.metrics можно вычислить долю верно сделанных предсказаний (рис. 3.27).

```
In [26]: from sklearn.metrics import accuracy_score
accuracy_score(y_test, knn_pred)

Out[26]: 0.749624060150376
```

Рис. 3.27. Доля верно сделанных предсказаний на тестовых данных

На рис. 3.28 представлена таблица сопряженности по предсказанному отклику для тестовых данных.

```
In [28]: X_test1.groupby(['knn', 'churn'])['knn'].count().unstack()
Out[28]:
```

	churn	Остается	Уходит
knn			
Остается		637	225
Уходит		108	360

Рис. 3.28. Таблица сопряженности для наблюдаемых и предсказанных значений оттока для тестовой выборки

3.2.2. Набор данных *медицинское страхование* (регрессия)

Второй набор данных – набор данных медицинское страхование (см. разд. 2.2.2). Файл с данными `insurance.csv` должен находиться в той же директории, что и папка `.anaconda`.

Код notebook для подключения библиотеки и загрузки данных – на рис. 3.29.

```
# Импортируем библиотеки
import pandas as pd

insurance_data = pd.read_csv('insurance.csv', ',')
insurance_data.head()
```

Рис. 3.29. Подключение библиотеки и импорт данных

Создадим новые фреймы данных: `X` – для предикторов, `y` – для отклика, и преобразуем категориальные переменные в фиктивные (рис. 3.30).

```
#Сохраняем в новый фрейм данных столбцы предикторов
X=insurance_data.drop(['charges'], axis=1)
#Сохраняем в переменную y отклик
y=insurance_data.charges

#Преобразуем категориальные переменные в dummy
X=pd.get_dummies(X, drop_first=True)
```

Рис. 3.30. Преобразование данных

Выполним разбиение данных на тренировочную и тестовую части (рис. 3.31). Доля тестовых данных – 30 %.

Передадим в классификатор тренировочные данные (рис. 3.32).

```

#Проведем разбиение исходных наблюдений предикторов и отклика
#на обучающую выборку и тестовую
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.neighbors import KNeighborsRegressor

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

```

Рис. 3.31. Разбиение данных

```

In [41]: knn = KNeighborsRegressor(n_neighbors=10)
         knn.fit(X_train, y_train)

Out[41]: KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                             weights='uniform')

```

Рис. 3.32. Создание объекта класса KNeighborsRegressor

Найдем предсказания медицинских расходов для тестового набора данных методом ближайших соседей и сформируем таблицу X_test1, включающую предикторы, отклик и предсказание (рис. 3.33).

```

knn_pred = knn.predict(X_test)
X_test1 = X_test
X_test1['knn'] = knn_pred
X_test1['churn'] = y_test
X_test1.head()

```

Рис. 3.33. Формирование набора данных с предсказаниями и откликом на основе тестовых данных

Фрагмент таблицы с предсказаниями на тестовых данных – на рис. 3.34.

	age	bmi	children	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest	knn	churn
1190	31	32.775	2	0	0	1	0	0	6503.317445	5327.40025
588	61	35.910	0	0	0	0	0	0	25186.624901	13635.63790
700	21	34.870	0	0	0	0	1	0	7890.210160	2020.55230
632	29	35.530	0	0	0	0	1	0	13425.698545	3366.66970
288	59	36.765	1	0	1	0	0	0	21111.149846	47896.79135

Рис. 3.34. Тестовые данные с предсказанными значениями медицинских расходов по методу kNN (фрагмент)

3.3. Линейная регрессия

В Python модель линейной регрессии реализуется в классе LinearRegression, который импортируется из класса sklearn.linear_model (библиотека scikit-learn) (методы класса – в табл. 3.8).

```
from sklearn.linear_model import LinearRegression
```

Аргумент класса `LinearRegression()`: `fit_intercept=True` (модель включает свободный член), `fit_intercept=False` (свободный член отсутствует).

Таблица 3.8

Методы класса `LinearRegression()`

<code>fit(X, y)</code>	Создает объект класса <code>LinearRegression()</code> на основе обучающих данных: предикторы – фрейм данных <code>X</code> и отклик <code>y</code>
<code>predict(X_test)</code>	Прогнозирует значение отклика для каждого наблюдения данных <code>X_test</code>
<code>score(X,y)</code>	Возвращает значение коэффициента детерминации
<code>intercept_</code>	Возвращает значение свободного члена
<code>coef_</code>	Возвращает значение коэффициентов

Повторим построение модели линейной регрессии на языке Python на тех же данных, что и на языке R (см. разд. 2.3).

Набор данных медицинское страхование, код notebook для подключения библиотеки и загрузки данных показаны на рис. 3.35.

```
# Импортируем библиотеки
import pandas as pd

insurance_data = pd.read_csv('insurance.csv', ',')
insurance_data.head()
```

Рис. 3.35. Подключение библиотеки и импорт данных

Создадим новые фреймы данных: `X` – для предикторов, `y` – для отклика, и преобразуем категориальные переменные в фиктивные (рис. 3.36).

```
#Сохраняем в новый фрейм данных столбцы предикторов
X=insurance_data.drop(['charges'], axis=1)
#Сохраняем в переменную y отклик
y=insurance_data.charges

#Преобразуем категориальные переменные в dummy
X=pd.get_dummies(X, drop_first=True)
```

Рис. 3.36. Преобразование данных

Импортируем класс `LinearRegression`, создадим экземпляр класса, передав обучающие данные (рис. 3.37). Выведем оценки параметров модели линейной регрессии (рис. 3.38). Получим значение коэффициента детерминации для модели линейной регрессии (рис. 3.39). Изменим вывод оценок параметров модели, связав с предикторами модели (рис. 3.40).

```
In [6]: from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X, y)

Out[6]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Рис. 3.37. Построение модели линейной регрессии на обучающей выборке

```
In [30]: lr.intercept_

Out[30]: -11938.538576167157

In [31]: lr.coef_

Out[31]: array([ 256.85635254,  339.19345361,  475.50054515, -131.3143594 ,
                23848.53454191, -352.96389942, -1035.02204939, -960.0509913 ])
```

Рис. 3.38. Оценки параметров линейной регрессии

```
In [32]: lr.score(X, y)

Out[32]: 0.7509130345985205
```

Рис. 3.39. Значение коэффициента детерминации

```
In [33]: klr=pd.DataFrame()
klr['name']=X.columns
klr['coef']=lr.coef_
klr

Out[33]:
```

	name	coef
0	age	256.856353
1	bmi	339.193454
2	children	475.500545
3	sex_male	-131.314359
4	smoker_yes	23848.534542
5	region_northwest	-352.963899
6	region_southeast	-1035.022049
7	region_southwest	-960.050991

Рис. 3.40. Вывод коэффициентов совместно с предикторами

Предсказанные значения отклика по предикторам обучающей выборки представлены на рис. 3.41. Воспользуемся возможностями библиотеки statsmodels, чтобы вывести результаты по линейной модели в привычном для языка R формате (рис. 3.42). Построение модели парной линейной регрессии по переменным charges (отклик) и bmi (предиктор) и вывод сводки по модели – на рис. 3.43.

```
In [46]: lr.predict(X)
Out[46]: array([25293.7130284 , 3448.60283431, 6706.9884907 , ...,
               4149.13248568, 1246.58493898, 37085.62326757])
```

Рис. 3.41. Предсказания отклика по обучающей выборке

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
|
```

Рис. 3.42. Импорт классов из библиотеки statsmodels

```
In [22]: results = smf.ols('charges ~ bmi', data=insurance_data).fit()
         results.summary()
```

```
Out[22]: OLS Regression Results
```

Dep. Variable:	charges	R-squared:	0.039
Model:	OLS	Adj. R-squared:	0.039
Method:	Least Squares	F-statistic:	54.71
Date:	Mon, 23 Mar 2020	Prob (F-statistic):	2.46e-13
Time:	00:00:24	Log-Likelihood:	-14451.
No. Observations:	1338	AIC:	2.891e+04
Df Residuals:	1336	BIC:	2.892e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
Intercept	1192.9372	1664.802	0.717	0.474	-2072.974	4458.849
bmi	393.8730	53.251	7.397	0.000	289.409	498.337

Omnibus:	261.030	Durbin-Watson:	1.983
Prob(Omnibus):	0.000	Jarque-Bera (JB):	431.091
Skew:	1.297	Prob(JB):	2.45e-94
Kurtosis:	4.004	Cond. No.	160.

Рис. 3.43. Сводка по модели парной линейной регрессии

Для включения в формулу дополнительных предикторов выведем названия столбцов (рис. 3.44). Построение модели множественной линейной регрессии для отклика charges представлено на рис. 3.45.

```
In [24]: insurance_data = pd.get_dummies(insurance_data, drop_first=True)
insurance_data.columns

Out[24]: Index(['age', 'bmi', 'children', 'charges', 'sex_male', 'smoker_yes',
               'region_northwest', 'region_southeast', 'region_southwest'],
              dtype='object')
```

Рис. 3.44. Вывод названий столбцов

```
In [34]: results = smf.ols('charges ~ age +bmi + children + smoker_yes',
                           data=insurance_data).fit()
results.summary()
```

Out[34]: OLS Regression Results

Dep. Variable:	charges	R-squared:	0.750
Model:	OLS	Adj. R-squared:	0.749
Method:	Least Squares	F-statistic:	998.1
Date:	Mon, 23 Mar 2020	Prob (F-statistic):	0.00
Time:	00:24:52	Log-Likelihood:	-13551.
No. Observations:	1338	AIC:	2.711e+04
Df Residuals:	1333	BIC:	2.714e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-1.21e+04	941.984	-12.848	0.000	-1.4e+04	-1.03e+04
age	257.8495	11.896	21.675	0.000	234.512	281.187
bmi	321.8514	27.378	11.756	0.000	268.143	375.559
children	473.5023	137.792	3.436	0.001	203.190	743.814
smoker_yes	2.381e+04	411.220	57.904	0.000	2.3e+04	2.46e+04

Рис. 3.45. Сводка по результатам построения множественной регрессии

3.4. Линейный классификатор (логистическая регрессия)

В Python модель логистической регрессии реализуется в классе LogisticRegression, который импортируется из класса sklearn.linear_model (библиотека scikit-learn) (методы класса – в табл. 3.9).

```
from sklearn.linear_model import LogisticRegression
```

Методы класса **LogisticRegression()**

fit(X, y)	Создает объект класса LogisticRegression() на основе обучающих данных: предикторы – фрейм данных X и отклик y
predict(X_test)	Прогнозирует значение отклика для каждого наблюдения данных X_test
predict_proba(X_test)	Прогнозирует вероятности возможных значений отклика для каждого наблюдения данных X_test
score(X,y)	Возвращает значение коэффициента детерминации
intercept_	Возвращает значение свободного члена
coef_	Возвращает значение коэффициентов

Повторим построение модели логистической регрессии на языке Python на тех же данных, что и на языке R (см. разд. 2.4).

Файл с данными Churn_binned.csv должен находиться в той же директории, что и папка .anaconda.

Код notebook для подключения библиотеки и загрузки данных – на рис. 3.46.

```
# Импортируем библиотеки
import pandas as pd

churn_data = pd.read_csv('Churn_binned.csv', ';')
churn_data.head()
```

Рис. 3.46. Подключение библиотеки и импорт данных

Создадим новые фреймы данных: X – для предикторов, y – для отклика, и преобразуем категориальные переменные в фиктивные (рис. 3.47).

```
# Сохраняем в новый фрейм данных столбцы предикторов
X=churn_data.drop(['churn'], axis=1)
# Сохраняем в переменную y отклик
y=churn_data.churn

# Преобразуем категориальные переменные в dummy
X=pd.get_dummies(X, drop_first=True)
```

Рис. 3.47. Преобразование данных

Импортируем класс LogisticRegression, создадим экземпляр класса, передав обучающие данные (рис. 3.48).

```
In [50]: from sklearn.linear_model import LogisticRegression
logr = LogisticRegression()
logr.fit(X, y)
```

Рис. 3.48. Построение модели логистической регрессии на обучающей выборке

Выведем оценки параметров модели логистической регрессии (рис. 3.49, 3.50).

```
In [62]: logr.coef_
Out[62]: array([[ 0.37915008,  0.19986719,  0.28137112,  1.68389109,  0.01431584,
                 -0.10461522, -0.00429745,  0.86452055,  0.09741697, -0.06750605,
                 0.10423842,  0.19861089, -2.38805753,  0.02683807, -0.46210836,
                 -0.47632068, -0.57195411, -0.10726861,  0.66431155,  0.71854556,
                 0.51366725,  0.29362521]])
```

Рис. 3.49. Оценки параметров логистической регрессии (при предикторах)

```
In [65]: logr.intercept_
Out[65]: array([0.02755696])
```

Рис. 3.50. Оценка свободного члена логистической регрессии

Получим значение коэффициента детерминации для модели логистической регрессии (рис. 3.51).

```
In [51]: logr.score(X, y)
Out[51]: 0.7661927330173776
```

Рис. 3.51. Значение коэффициента детерминации

Предварительно преобразуем матрицу коэффициентов логистической регрессии в вектор (рис. 3.52). Изменим вывод оценок параметров модели, связав с предикторами модели (рис. 3.53).

```
In [63]: import numpy as np
np.array(logr.coef_).flatten()
Out[63]: array([ 0.37915008,  0.19986719,  0.28137112,  1.68389109,  0.01431584,
                 -0.10461522, -0.00429745,  0.86452055,  0.09741697, -0.06750605,
                 0.10423842,  0.19861089, -2.38805753,  0.02683807, -0.46210836,
                 -0.47632068, -0.57195411, -0.10726861,  0.66431155,  0.71854556,
                 0.51366725,  0.29362521])
```

Рис. 3.52. Преобразование матрицы коэффициентов регрессии в вектор

```
In [64]: klogr=pd.DataFrame()
klogr['name']=X.columns
klogr['coef']=np.array(logr.coef_).flatten()
klogr
```

Out[64]:

	name	coef
0	longdist_2--8	0.379150
1	longdist_21+	0.199867
2	longdist_9--14	0.281371
3	longdist_меньше2	1.683891
4	local_36-55	0.014316
5	local_56+	-0.104615
6	local_8--20	-0.004297
7	local_меньше8	0.864521
8	int_disc_Нет	0.097417
9	billtype_Бюджетный	-0.067506
10	pay_CC	0.104238
11	pay_CH	0.198611
12	gender_Мужской	-2.388058
13	marital_Одинокий	0.026838
14	incomecat_36-50	-0.462108

Рис. 3.53. Вывод коэффициентов регрессии совместно с предикторами (фрагмент)

Предсказанные значения отклика и вероятности отнесения отклика к одному из классов по предикторам обучающей выборки представлены на рис. 3.54.

```
In [66]: logr.predict(X)
```

```
Out[66]: array(['Уходит', 'Уходит', 'Уходит', ..., 'Уходит', 'Уходит', 'Уходит']
dtype=object)
```

```
In [67]: logr.predict_proba(X)
```

```
Out[67]: array([[0.37556469, 0.62443531],
[0.35987123, 0.64012877],
[0.35987123, 0.64012877],
...,
[0.38897563, 0.61102437],
[0.38897563, 0.61102437],
[0.38897563, 0.61102437]])
```

Рис. 3.54. Предсказания отклика по обучающей выборке

Воспользоваться возможностями библиотеки `statsmodels`, чтобы вывести результаты по логистической регрессии в привычном для языка R формате, предлагается самостоятельно.

Загрузка необходимых методов:

```
import statsmodels.api as sm
from statsmodels.formula.api import logit
```

Отклик должен быть числовым вектором, как и все предикторы. В названии столбцов не должно быть знаков сравнения, препинания и т.п. Поэтому лучше использовать не `pd.get_dummies()`, а метод `map(dictionary)`, где `dictionary` — словарь (например, `dictionary={'yes': 1, 'no': 0}`).

4. Лабораторные работы

4.1. Лабораторная работа «Деревья решений»

Цель работы: научиться проводить предобработку данных, строить деревья регрессии и классификации и интерпретировать терминальные узлы дерева с точки зрения бизнес-контекста.

Формируемые знания, умения и навыки: изучить, как реализуются деревья решений на языках R и Python, получить навыки выделения бизнес-контекста из результатов построения деревьев решений.

Необходимо:

1. С сайта <https://www.kaggle.com/> импортировать один из наборов.
2. Используя возможности языков R и Python, провести классификацию данных с помощью деревьев решений, выдвинуть гипотезы о бизнес-контексте результатов анализа.

Контрольные вопросы и задания

R

1. Какие аргументы у функции `rpart()`? Какие значения могут принимать эти аргументы? Какие параметры у `rpart.control()`? В чем отличие использования функции для построения деревьев регрессии и классификации?
2. Как получить краткую сводку по дереву решений? Как получить подробную сводку по дереву решений?
3. Какие преобразования исходных данных выполнялись и почему?
4. Какие функции использовались для визуализации полученных результатов?

Python

5. Какие параметры и методы у класса `DecisionTreeClassifier` и класса `DecisionTreeRegressor`? Как получить доступ к данным о прогнозах классов для наблюдений?
6. Какие преобразования исходных данных выполнялись и почему?
7. Какие функции использовались для визуализации полученных результатов?

Бизнес-контекст

Опишите выбранный набор данных: контекст данных, какие переменные есть в наборе, как происходит разбиение пространства признаков на классы, как можно использовать в бизнесе результаты построения деревьев решений.

4.2. Лабораторная работа «Метод ближайших соседей»

Цель работы: научиться импортировать данные, проводить предобработку данных, разбивать данные на обучающую выборку и тестовую, применять к данным метод ближайших соседей.

Формируемые знания, умения и навыки: изучить, как реализуются метод ближайших соседей на языках R и Python.

Необходимо:

1. С сайта <https://www.kaggle.com/> импортировать один из наборов, включающий столбцы с числовыми значениями.
2. Используя возможности языков R и Python, разбить данные на тренировочные и тестовые и выполнить предсказание отклика на тестовых данных по методу ближайших соседей.

Контрольные вопросы и задания

R

1. Какие аргументы у функции `knn()`? Какие значения могут принимать эти аргументы? В каком формате возвращается результат?
2. Какие преобразования исходных данных выполнялись и почему?
3. Какие функции использовались для разбиения исходных данных на тренировочные данные и тестовые данные?

Python

4. Какие параметры и методы у классов `KNeighborsClassifier` и `KNeighborsRegressor`?
5. Какие преобразования исходных данных выполнялись и почему?
6. Какой класс используется для разбиения исходных данных на тренировочные и тестовые и какие параметры у класса?

Бизнес-контекст

Как можно использовать в бизнесе результаты метода ближайших соседей?

4.3. Лабораторная работа «Линейная регрессия»

Цель работы: научиться импортировать данные, проводить предобработку данных, строить модель линейной регрессии и интерпретировать полученные результаты с точки зрения бизнес-контекста.

Формируемые знания, умения и навыки: изучить, как реализуется построение линейной регрессии на языках R и Python, получить навыки выделения бизнес-контекста из результатов построения линейной регрессии.

Необходимо:

1. С сайта <https://www.kaggle.com/> импортировать один из наборов, содержащий транзакции.

2. Используя возможности языков R и Python, провести построение линейной регрессии, выдвинуть гипотезы о бизнес-контексте результатов анализа.

Контрольные вопросы и задания

R

1. Какие аргументы у функции `lm()`? Какие значения могут принимать эти аргументы? В каком формате возвращается результат? Какие функции можно использовать дополнительно для вывода расширенной сводки по результатам модели и оценок ее качества?

2. Какие преобразования исходных данных выполнялись и почему?

Python

3. Какие параметры и методы у класса `LinearRegression`?

4. Какие преобразования исходных данных выполнялись и почему?

5. Какие возможности предоставляет библиотека `statsmodels`, чтобы вывести результаты по линейной модели в привычном для языка R формате?

Бизнес-контекст

Опишите выбранный набор данных: контекст данных, какие данные есть в наборе, как можно использовать в бизнесе полученные результаты построения линейной регрессии.

4.4. Лабораторная работа «Линейный классификатор (логистическая регрессия)»

Цель работы: научиться импортировать данные, проводить предобработку данных, строить модель логистической регрессии и интерпретировать полученные результаты с точки зрения бизнес-контекста.

Формируемые знания, умения и навыки: изучить, как реализуется построение логистической регрессии на языках R и Python, получить навыки выделения бизнес-контекста из результатов построения логистической регрессии.

Необходимо:

1. С сайта <https://www.kaggle.com/> импортировать один из наборов.
2. Используя возможности языков R и Python, провести построение логистической регрессии, выдвинуть гипотезы о бизнес-контексте результатов анализа.

Контрольные вопросы и задания

R

1. Какие аргументы у функции `glm()`? Какие значения могут принимать эти аргументы? В каком формате возвращается результат? Какие функции можно использовать дополнительно для вывода расширенной сводки по результатам модели и оценок ее качества?
2. Какие преобразования исходных данных выполнялись и почему?

Python

3. Какие параметры и методы у класса `LogisticRegression`?
4. Какие преобразования исходных данных выполнялись и почему?
5. Какие возможности предоставляет библиотека `statsmodels`, чтобы вывести результаты по логит модели в привычном для языка R формате?

Бизнес-контекст

Опишите выбранный набор данных: контекст данных, какие данные есть в наборе, как можно использовать в бизнесе полученные результаты построения логистической регрессии.

Библиографический список

1. Открытый курс машинного обучения. Тема 3: Классификация, деревья решений и метод ближайших соседей / Open Data Science. – URL: <https://habr.com/ru/company/ods/blog/322534/>
2. Открытый курс машинного обучения. Тема 4: Линейные модели классификации и регрессии / Open Data Science – URL: <https://habr.com/ru/company/ods/blog/323890/#lineyny-klassifikator/>
3. Карпов, А. Введение в Data Science и машинное обучение / А. Карпов / Stepik – URL: <https://stepik.org/course/4852>
4. Груздев, А. В. Прогнозное моделирование в IBM SPSS Statistics, R и Python: метод деревьев решений и случайный лес : руководство / А. В. Груздев. – Москва : ДМК Пресс, 2018. – 642 с. – URL: <https://e.lanbook.com/book/123700> (дата обращения: 09.03.2020).
5. Введение в статистическое обучение с примерами на языке R / Г. Джеймс, Д. Уиттон, Т. Хасты, Р. Тибширани ; пер. с англ. С. Э. Мастицкого. – Москва : ДМК Пресс, 2017. – 456 с. – URL: <https://e.lanbook.com/book/93580> (дата обращения: 09.03.2020).
6. Коэльо, Л. П. Построение систем машинного обучения на языке Python / Л. П. Коэльо, В. Ричарт ; пер. с англ. А. А. Слинкина. – 2-е изд. – Москва : ДМК Пресс, 2016. – 302 с. – URL: <https://e.lanbook.com/book/82818> (дата обращения: 09.03.2020).
7. Рашка, С. Python и машинное обучение: крайне необходимое пособие по новейшей предсказательной аналитике, обязательное для более глубокого понимания методологии машинного обучения : руководство / С. Рашка ; пер. с англ. А. В. Логунова. – Москва : ДМК Пресс, 2017. – 418 с. – URL: <https://e.lanbook.com/book/100905> (дата обращения: 08.03.2020).
8. Шитиков, В. К. Классификация, регрессия и другие алгоритмы Data Mining с использованием R / В. К. Шитиков, С. Э. Мастицкий. – URL: <https://ranalytics.github.io/data-mining/index.html>
9. Кабаков, Р. И. R в действии. Анализ и визуализация данных в программе R : руководство / Р. И. Кабаков ; пер. с англ. П. А. Волковой. – Москва : ДМК Пресс, 2014. – 588 с. – URL: <https://e.lanbook.com/book/58703> (дата обращения: 17.03.2020).
10. Форман, Д. Много цифр. Анализ больших данных с помощью Excel / Д. Форман ; пер. с англ. А. Соколовой. – Москва : Аль-

пина Паблишер, 2016. – 461 с. – URL: <https://new.znanium.com/catalog/product/551044> (дата обращения: 22.03.2020).

11. Груздев, А. В. Изучаем Pandas / А. В. Груздев, М. Хейдт ; пер. с англ. А. В. Груздева. – 2-е изд., испр. и доп. – Москва : ДМК Пресс, 2019. – 700 с. – URL: <https://e.lanbook.com/book/131693> (дата обращения: 09.03.2020).

12. Официальный сайт Anaconda. – URL: <https://www.anaconda.com/>

13. Описание библиотеки scikit-learn. – URL: <https://scikit-learn.org/stable/index.html>

14. Описание библиотеки pandas. – URL: <https://pandas.pydata.org/>

15. Рындина, С. В. Бизнес-аналитика на основе больших данных: обучение без учителя на языках Python и R : учеб.-метод. пособие / С. В. Рындина. – Пенза : Изд-во ПГУ, 2020. – 76 с.

Учебное издание

Рындина Светлана Валентиновна

**Бизнес-аналитика на основе больших данных:
обучение с учителем на языках Python и R**

Редактор *Н. А. Сидельникова*
Технический редактор *Д. В. Тарасов*
Компьютерная верстка *Д. В. Тарасова*

Подписано в печать 19.06.2020.
Формат 60×84¹/₁₆. Усл. печ. л. 4,65.
Заказ № 144. Тираж 3.

Издательство ПГУ.
440026, Пенза, Красная, 40.
Тел./факс: (8412) 56-47-33; e-mail: iic@pnzgu.ru