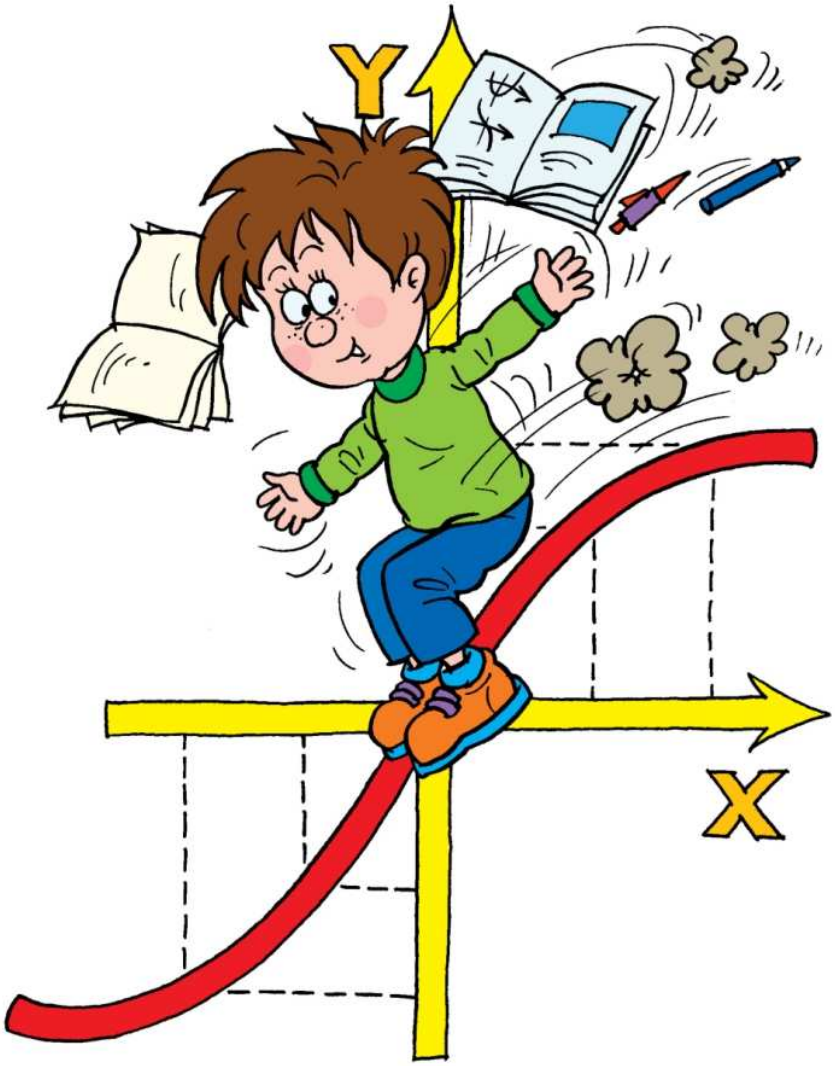
В.К. Шитиков, С.Э. Мастицкий

Классификация, регрессия и другие алгоритмы Data Mining с использованием R



Тольятти, Лондон - 2017

Шитиков В.К., Мастицкий С.Э. (2017) Классификация, регрессия и другие алгоритмы Data Mining с использованием R. 351 с.

Описана широкая совокупность методов построения статистических моделей классификации и регрессии для откликов, измеренных в альтернативной, категориальной и метрической шкалах. Подробно рассматриваются деревья решений, машины опорных векторов с различными разделяющими поверхностями, нелинейные формы дискриминантного анализа, искусственные нейронные сети и т.д. Показана технология применения таких методов бутстреп-агрегирования деревьев решений, как бэггинг, случайный лес и бустинг. Представлены различные методы построения ансамблей моделей для коллективного прогнозирования. Особое внимание уделяется сравнительной оценке эффективности и поиску оптимальных областей значений гиперпараметров моделей с использованием пакета `caret` для статистической среды R. Рассматриваются такие алгоритмы Data Mining, как генерация ассоциативных правил и анализ последовательностей. Отдельные главы посвящены методам многомерной ординации данных и различным алгоритмам кластерного анализа.

Описание методов статистического анализа сопровождается многочисленными примерами из различных областей на основе общедоступных исходных данных. Представлены несложные скрипты на языке R, дающие возможность читателю легко воспроизвести все расчеты.

Книга может быть использована в качестве учебного пособия по статистическим методам для студентов и аспирантов высших учебных заведений.

Контактная информация:
В.К. Шитиков
Институт экологии Волжского бассейна РАН
г. Тольятти, ул. Комзина, 10
Самарская обл., 445003, Россия
E-mail: stok1946@gmail.com
rtutorialsbook@gmail.com

Иллюстрация на обложке худ. А. Банных (г. Владимир)

СОДЕРЖАНИЕ

1. РЕАЛИЗАЦИЯ МОДЕЛЕЙ DATA MINING В СРЕДЕ R (ВМЕСТО ПРЕДИСЛОВИЯ)

1.1. Data Mining как направление анализа данных

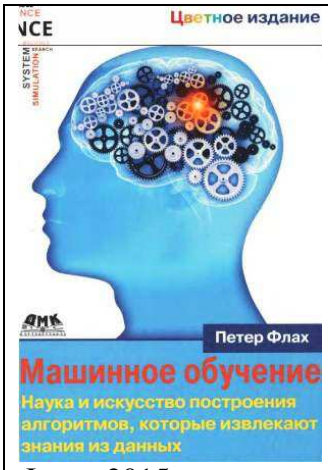
> «*Наше видение природы претерпевает радикальные изменения в сторону множественности, темпоральности и сложности.*»
>
> (Пригожин, Стенгерс, 2005, с. 11)

> «*Любое существо, наделенное интеллектом, решает в своей жизни, как правило, только три задачи: а) запоминание по ассоциациям; б) распознавание образов; в) задачи оптимизации и принятия решений.*»
>
> (Хокинс, Блейксли, 2007)

От статистического анализа разового эксперимента к Data Mining

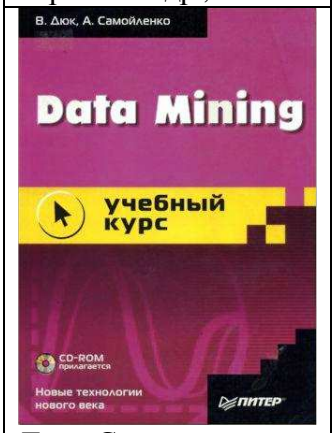
Экспериментальные данные, представленные в компьютерном формате в виде взаимосвязанных таблиц, нуждаются в таких процедурах обработки анализа и интерпретации, которые, во-первых, делают очевидными потенциально возможные закономерности и связи между отдельными компонентами и, во-вторых, дают возможность предсказать новые факты. В узком плане речь может идти об оценке значения целевого признака y (отклика) для любого объекта a по его описанию x – набору независимых переменных (предикторов). Однако в более широком смысле затрагиваются традиционно ключевые вопросы многомерного анализа систем:

- Можно ли считать идентичными анализируемые объекты и за счет каких признаков можно объяснить их возможные отличия?
- Как можно объединить отобранные объекты в группы?
- Существует ли пространственная или временная изменчивость описанных объектов и каковы ее структурные особенности?
- Изменение каких признаков приводит к систематическим причинно-следственным изменениям других?
- Как можно осуществить прогноз состояния или поведения анализируемого объекта?

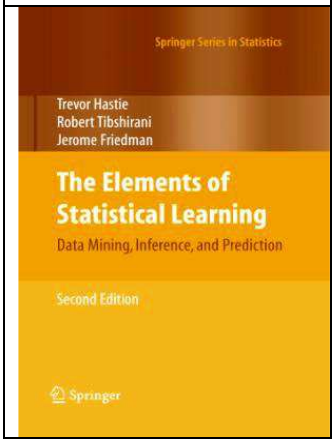


Флах, 2015

Барсегян и др., 2009



Дюк, Самойленко, 2001



До начала 90-х годов основной практикой научных исследований была оценка по Р. Фишеру отдельных "взаимодействий в разовом эксперименте" и, казалось, не было особой нужды кардинально менять ситуацию в этой области. Однако внедрение современных информационных технологий обрушило на людей колоссальные объемы разнородных данных в самых различных областях. Возник вопрос, что делать с этой информацией, поскольку без возможности эффективной обработки ее осмысление оказалось невозможным (Дюк, Самойленко, 2001; Барсегян и др., 2009).

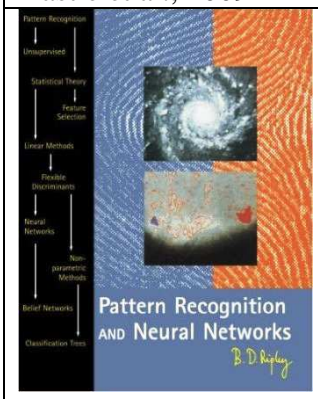
Время "лоскутных исследований" стремительно проходит, уступая место комплексному (сейчас говорят "системному") подходу к описанию процессов и явлений. Поэтому актуальной стратегией современной прикладной статистики является обработка массивов постоянно пополняемых и расширяемых данных с целью создания адекватных многофункциональных моделей изучаемых систем. Создались предпосылки для построения адаптируемых моделей, шаг за шагом улучшающихся по мере поступления новых экспериментальных данных или расширения "сферы влияния" модели.

Современные информационные технологии предполагают размещение таблиц в сконцентрированном виде в хранилищах данных (Барсегян и др., 2009). В этих системах разрозненная информация представляется в виде многомерного куба, которым можно легко манипулировать, извлекая срезами нужную информацию. Для проверки сложных гипотез и решения стратегических проблем используется аппарат извлечения знаний из обширных баз данных (knowledge discovery in databases), основой которого является интеллектуальный анализ данных – Data Mining.

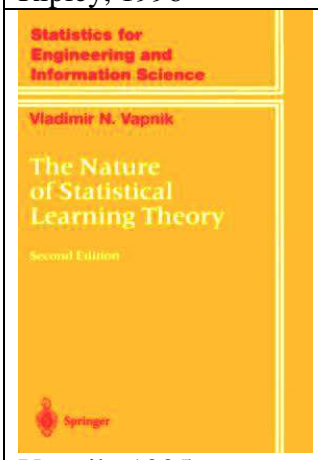
Data Mining – это метафора от горной добычи: разработки пород, извлечение чего-то ценного, откапывание драгоценных крупиц ценных веществ в большом количестве сырого материала, но, соответственно, в применении к данным. По содержанию этот термин достаточно точно определяет Г. Пиатецкий-Шапиро (Piatetsky-Shapiro): «*это технология, которая предназначена для поиска в больших объемах данных неочевидных, объективных и полезных на практике закономерностей*».

К настоящему времени термин Data Mining

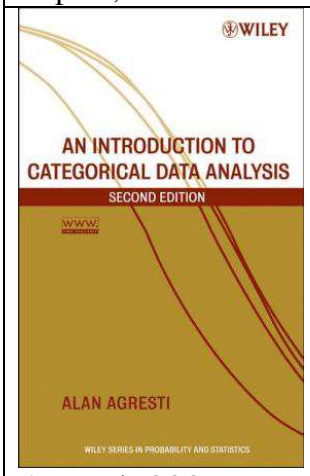
Hastie et al., 2009



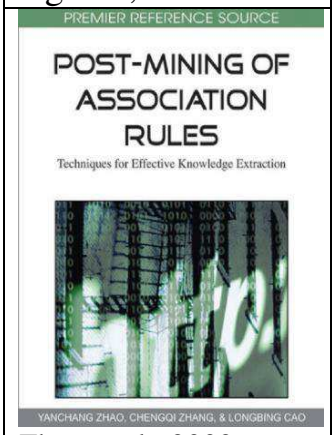
Ripley, 1996



Vapnik, 1995



Agresti, 2007



Zhao et al., 2009

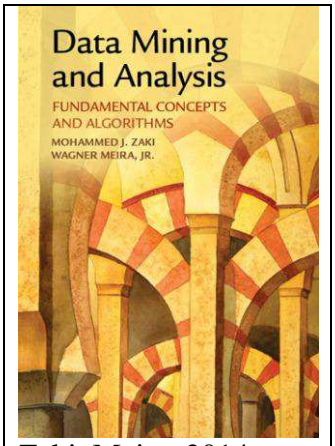
оформился как собирательное название целой совокупности методов при решении конкретных задач глубокого анализа данных с целью выявления скрытых закономерностей. Ряд направлений имеет достаточно специфический и иногда локальный характер: обработка текстов (Text Mining – Sakurai, 2012), анализ социальных сетей и функции работы с графами (Zafarani et al., 2015), решение проблем работы с большими массивами данных (Leskovec et al., 2014), выделение последовательностей термов (Wang, Yang, 2005) и т.д. В настоящей книге эти алгоритмы подробно рассматриваться не будут.

Извлечение научных гипотез (data mining) и их последующий анализ (data analysis) – два комплексных неразрывно связанных процесса. Они протекают по стандартной схеме установления физических законов: сбор экспериментальных данных, организация их в виде таблиц и поиск такого способа обработки, который позволил бы обнаружить в исходных данных новые знания об анализируемом процессе. При этом должно быть ясное понимание того, что эти знания, как всегда для любого сложного явления, остаются в какой-то степени приближенными: чем глубже анализируется реальная сложная система, тем менее определенными становятся наши суждения о ее поведении.

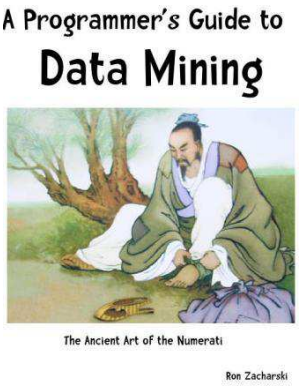
Принципиальная множественность моделей окружающего мира

Сложность системы и точность, с которой её можно анализировать, связаны обратной зависимостью: «*исследователь постоянно находится между Сциллой усложненности и Харибдой недостоверности. С одной стороны, построенная им модель должна быть простой в математическом отношении, чтобы её можно было исследовать имеющимися средствами. С другой стороны, в результате всех упрощений она не должна утратить и "рациональное зерно", существо проблемы*» (Самарский, 1979, с. 28).

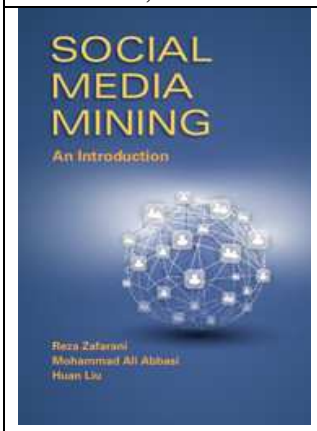
Любая сложная система ведет себя *контринтуитивно*, т.е. она реагирует на воздействие совсем иным образом, чем это нами интуитивно ожидалось (Форрестер, 1977; Розенберг, 2013). При этом декларируемая нами точность любого математического описания системы – это не абсолютный вердикт, а только принятое нами соглашение о способе отождествления



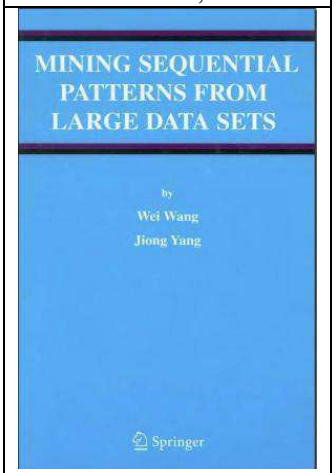
Zaki, Meira, 2014



Zacharski, 2015



Zafarani et al., 2015



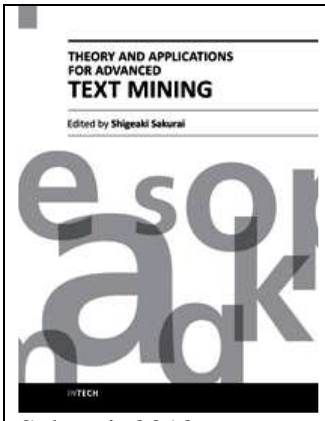
Wang, Yang, 2005

модели и реального мира.

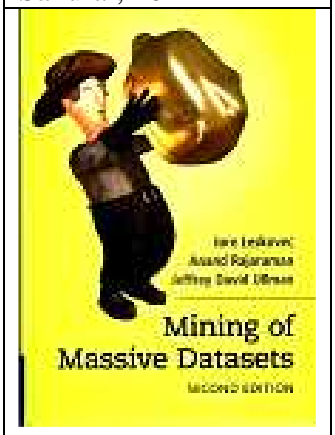
В современных исследованиях обычно анализируются результаты пассивных наблюдений, поскольку поставить управляемый рандомизированный эксперимент часто попросту невозможно. Т.к. мы точно не знаем, какие параметры определяют наш процесс, то для интерпретации данных крайне важно иметь по возможности полный список конкурирующих гипотез. Тогда итогом статистического анализа является уже не одна неоспоримая истина, а набор полезных (оптимальных в частном смысле) моделей исследуемого явления, которые могут быть сопоставлены между собой, а в дальнейшем уточнены, дополнены или заменены на лучшие. «*Процедура проверки значимости нулевой гипотезы, основанная на значении* P_{val}*, – квинтэссенция традиционной* (*ортодоксальной*) *статистической практики и одновременно – ее величайшее недоразумение и заблуждение… Преодоление порогового (критического) уровня* $P_{val} < 0{,}05$ *всего лишь в одной выборке часто необоснованно считается достаточным для вывода о статистической значимости наблюдаемого эффекта* (*или даже его "достоверности*")» (Хромов-Борисов, 2011). Вероятно, при всей ее категоричности, эта точка зрения вполне заслуживает всяческого внимания.

Сразу стоит также подчеркнуть, что идеального метода, который одинаково хорошо покажет себя для любых целей и на любом наборе данных, попросту не существует. Поэтому нельзя игнорировать принцип *множественности моделей* В.В. Налимова (1971): для объяснения и предсказания структуры и (или) поведения сложной системы возможно построение нескольких моделей, имеющих одинаковое право на существование. Мультимодельный вывод (Anderson, 2008) предполагает также оценку параметров (их ошибок и доверительных интервалов) на основе не единственной модели, а их ряда. И здесь важны тщательная диагностика и широкая верификация построенных моделей, которые мы будем подробно обсуждать в главе 2.

Наконец, следует помнить о том, что любой статистический метод будет хорош настолько, насколько качественными являются входные данные для обучения модели (англ. *"garbage in – garbage out!*" или "хлам на входе – хлам на выходе"). Без затраты усилий на



Sakurai, 2012



Leskovec et al., 2014

подготовку обучающей выборки (фильтрация, трансформация, удаление пропущенных значений, создание производных предикторов и т.д.) и понимания моделируемого процесса чудес не случается.

Нарастающая множественность алгоритмов построения моделей

Одна из ключевых проблем, с которой исследователь сталкивается при разработке статистической модели изучаемого явления, заключается в выборе оптимального для конкретного случая алгоритма извлечения закономерностей. За несколько последних десятилетий было разработано огромное множество методов для решения задач классификации и регрессии, что, безусловно, существенно затрудняет этот выбор.

Попытки ранжирования статистических методов по их эффективности предпринимались неоднократно. Например, одна из недавно опубликованных статей (Fernandez-Delgado et al., 2014) так и называлась «*Do We Need Hundreds of Classifiers to Solve Real World Classification Problems*?» («*Нужны ли нам сотни классификаторов для решения практических проблем классификации?*»).

В этом обстоятельном исследовании была изучена эффективность работы 179 методов классификации из 17 "семейств" на 121 наборе данных. Читатель может рассматривать эту интересную статью даже просто как справочник по методам распознавания.

В проведенном исследовании для каждого метода классификации оценивалась общая верность предсказаний (overall accuracy) и другие показатели эффективности моделей. Авторы предложили ограничиться следующими четырьмя семействами методов, обладающими наибольшей точностью прогноза (перечислены в порядке убывания эффективности):

1. "Случайный лес" (Random Forest);
2. Машины опорных векторов (Support Vector Machines);
3. Искусственные нейронные сети (Artificial Neural Networks);
4. Бустинговые ансамбли моделей (Boosting Ensembles).

Отметим, что перечисленные методы практически непригодны для интерпретации механизмов прогнозируемого явления, что вызвало ряд критических замечаний.

Хотя приведенный выше список касается моделей-классификаторов, можно ожидать, что перечисленные методы будут также хорошо работать и для задач регрессии (т.е. для предсказания количественного отклика). Тем не менее, если количество предикторов невелико и в них отражается реально существующая закономерность, не следует забывать, что хорошие результаты

можно получить и с использованием традиционных методов регрессии. Интересное обсуждение того, что простые методы (на примере классификаторов) при решении практических задач часто превосходят более сложные алгоритмы, можно найти также в широко цитируемой работе Д. Хэнда (Hand, 2006).

Часто выбор того или иного метода обусловлен предыдущим опытом и уровнем осведомленности исследователя. Так, в определенных областях может существовать своего рода "традиция" по использованию тех или иных методов для решения конкретного круга задач. В силу естественной ограниченности своей специализации исследователь может также просто не знать о существовании методов, которые являются более подходящими для его ситуации. Можно столкнуться и с такими случаями, когда некий разработчик программного обеспечения утверждает, что его новый алгоритм "не имеет аналогов", превосходя все другие доступные решения. Поэтому важнейшей задачей аналитиков является репрезентативное тестирование и тщательная сравнительная диагностика широкого множества моделей-претендентов. Наличие подобной информации будет особенно полезным при работе над новыми проектами/данными, когда предыдущий опыт, который мог бы подсказать, с чего стоит начинать, отсутствует.

Типы и характеристики групп моделей Data Mining

Предварительно, не вторгаясь в терминологические детали, отметим, что все три дисциплины – Data Mining, машинное обучение (Machine Learning) и статистический анализ – работают на одном и том же предметном поле и используют фактически одни и те же алгоритмы. Общие для них черты и темы гораздо обширней, чем различия, которые носят характер того или иного "уклона": Data Mining включает в сферу своей компетенции практически необходимые приемы работы с большими массивами данных, сетями и проч., машинное обучение склонно к красивой фразеологии по поводу искусственного интеллекта, а статистический анализ ищет обоснование своих процедур в теоретико-вероятностном подходе. В рамках нашего изложения эти дисциплины рассматриваются как синонимы.

Представленное выше многообразие методов построения моделей порождает разнообразие подходов к их группировке.

По способам решения задачи разделяют на "обучение с учителем" и "обучение без учителя". Формирование решающих правил без "учителя" объединяет алгоритмы, выявляющие скрытые закономерности без каких-либо предварительных знаний об анализируемых данных. В этом случае результаты наблюдений обычно геометрически интерпретируются как существенно "размытые" сгущения точек (объектов) в многомерном пространстве признаков.

Важнейшими методами поиска закономерностей без учителя являются *кластеризация* и *ординация*. Кластеризация исходит из принципа "дискретности" (или разделяемости) и пытается найти оптимальное разбиение исходной совокупности на отдельные группы (классы) однородных объектов

таким образом, чтобы различия между группами были максимально возможными. Ординация основывается на принципе континуальности (непрерывности) и ищет упорядоченную последовательность проекций изучаемых объектов на главные оси пространства, с которыми потенциально может быть связана интерпретация научных гипотез.

Алгоритмы индуктивного распознавания с обучением (Ripley, 1996; Vapnik, 1995; Agresti, 2007; Hastie et al., 2007; Zaki M., Meira, 2014) предполагают наличие априори заданной выборки прецедентов, позволяющей построить модели статистической связи $x \to y$, где

- $y \in Y$, $\mathbf{Y}$ – наблюдаемые реализации отклика (responses) или моделируемой случайной величины;
- $x \in \mathbf{X}$, $\mathbf{X}$ – множество переменных (predictors, independent variables, features), с помощью которых предполагается объяснить изменчивость переменной y.

Большинство моделей с учителем устроено таким образом, что их можно записать в виде $y = f(\mathbf{x}, \boldsymbol{\beta}) + \boldsymbol{\varepsilon}$, где f – математическая функция, выбранная из некоторого произвольного семейства, $\boldsymbol{\beta}$ – вектор параметров этой функции, а $\boldsymbol{\varepsilon}$ – ошибки модели, которые обычно сгенерированы несмещенным, некоррелированным случайным процессом. В ходе построения модели по фиксированным выборочным значениям y минимизируют некоторую функцию остатков $Q(y, \boldsymbol{\beta})$ и находят $\hat{\boldsymbol{\beta}}$ – вектор с оптимальными оценками параметров модели.

Варьируя вид функций $f()$ и $Q()$, можно получать разные модели, из которых предпочтение отдается наиболее эффективным – т.е. моделям, которые дают несмещенные, точные и надежные прогнозы отклика y. Под *смещением* (bias) понимается различие между рассчитанным по модели прогнозом (например, среднее из оценок всех возможных выборок, которые могут быть взяты из совокупности) и истинным неизвестным значением моделируемой переменной. *Точность* (accuracy) – различие между оценками отклика, основанного на выборочных данных и истинным значением реализации y. *Надежность* (precision) – различие между оценкой y по разовой выборке и средним из прогнозов по всему множеству выборок, которые могут быть взяты из той же совокупности.

Выбор подходящего семейства моделей зависит от типа обрабатываемых данных (бинарные, категориальные, порядковые или метрические) и закона их распределения. В первую очередь это, разумеется, относится к форме представления отклика $\mathbf{Y}$: если он представляет собой категориальную величину, то решается задача классификации, которая может иметь весьма специфические алгоритмы построения моделей и критерии их оптимизации. В случае метрических значений $\mathbf{Y}$ решается задача регрессии, а выбор способа моделирования порядковых или счетных данных часто оказывается неоднозначным. Тип переменных, составляющих матрицу предикторов, тоже в ряде случаев может оказать влияние на степень адекватности полученных результатов.

К типу задач обучения с учителем могут также относиться задачи выявления логических закономерностей и построения совокупности диагностических правил ("дистилляция эталонов") для последующего распознавания.

Некоторую систематичность в типизацию моделей классификации и регрессии может внести их связь с тремя основными парадигмами машинного обучения: геометрической, вероятностной и логической (Дюк, Самойленко, 2001; Барсегян и др., 2009). Однако группировка по чисто внутренним математическим аспектам достижения оптимального решения, будь то применение переборных процессов, линейной алгебры или статистической теории, на наш взгляд, мало обоснована.

Наконец, традиционным является деление методов Data Mining на описательные и прогнозирующие. Описательные методы должны приводить к *объяснению* или улучшению *понимания* данных. Ключевой момент в таких моделях – легкость и прозрачность восприятия исследователем предполагаемой связи между наблюдаемыми переменными. Используемые при этом методы могут относиться как к чисто статистическим (дескриптивный анализ, корреляционный и регрессионный анализ, факторный анализ, дисперсионный анализ, компонентный анализ, дискриминантный анализ, анализ временных рядов), так и к "кибернетическим" (ассоциативные правила, нечеткая логика, деревья решений, сети Кохонена, генетические алгоритмы).

При *прогнозировании* не ставится цель вскрыть структуру внутренних взаимосвязей между предикторами или сделать сравнительную оценку силы их влияния на отклик. Основная задача – предсказать величину целевого признака y на основании наблюдаемой вариации значений признаков x_1, ..., x_n. Если моделируемый процесс имеет объективно сложный характер (например, временные ряды с нестационарным трендом), то такие модели могут иметь большое (до десятков тысяч) или неопределенное число параметров. Для их построения используются следующие принципы самоорганизации: *а*) определяется класс моделей, в рамках которого возможен последовательный переход от простейших вариантов к наиболее сложным (т.е. индуктивный процесс) и *б*) задается механизм эволюции моделей и критерии их отбора, благодаря которым каждая отдельная "мутация" оценивается с точки зрения полезности для улучшения качества конечного результата.

Обеспечивая точность предсказания, модели прогнозирования в большинстве случаев существенно теряют в интерпретируемости. Часто бывает затруднительным сделать предметный анализ нескольких десятков коэффициентов моделей МГУА или нейронных сетей. Очевидна стремительно набирающая темп современная тенденция на создание ансамблей из сотен моделей, осуществляющих коллективное предсказание (bagging, random forest, boosting), когда традиционные регрессионные методы,

такие как анализ ковариаций, стандартных отклонений, доверительных интервалов коэффициентов и проч. вообще не имеют смысла.

Природа многомерного отклика и его моделирование

При изучении современных информационных, биологических и социально-экономических систем характерна оценка взаимных зависимостей между комплексами многомерных переменных, и тогда задачей построения статистической модели является объяснение изменчивости многомерного отклика **Y**. Часто такой отклик может быть представлен в форме некоторой относительно замкнутой системы элементов *N*, относящихся к множеству $S = \{y\}$ различных типов этих элементов (Арапов и др., 1975; Шитиков и др., 2012, глава 5). В разных предметных областях это может быть каталог продаваемых автомобилей, категории обеспеченности населения, словарь встретившихся словоформ, список экологических видов, обнаруженных в водоеме, перечень кандидатов, за которых голосовали на выборах и т.д. С помощью классификатора *S* объекты разбиваются на подпопуляции (классы), т. е. каждой рубрике $y \in S$ соответствует подмножество $N(y)$, определяющее частоту всех вхождений объектов этого типа в *N*. Закон *Ципфа–Парето*, описывающий характер таких распределений, играет практически ту же универсальную роль, что и закон Гаусса в обычных стохастических процессах с конечной дисперсией (Кудрин, 2002) – подробности см. в главе 9.

Композиции объектов $N(y)$, наблюдаемые в различных условиях (точках пространства, подразделениях, временных срезах), составляют матрицу **Y** случайных наблюдений многомерного отклика. Как всегда, задачей статистического моделирования является оценка зависимости Y от набора экзогенных независимых переменных **X**. Построить такую модель $\mathbf{X} \rightarrow \mathbf{Y}$ можно с применением двух подходов. Первый основан на свертке многомерного отклика к одномерному с помощью какой-нибудь функции (получив энтропию Шеннона, индекс типа Доу-Джонса, общую сумму стоимости товаров или нечто похожее), после чего используются обычные модели регрессии. Разумеется, в результате этого оказывается потерянной значительная часть важнейшей информации о структуре объекта. Другой вариант – использовать модели с многомерным откликом, которые обычно представляются в виде канонических матричных уравнений.

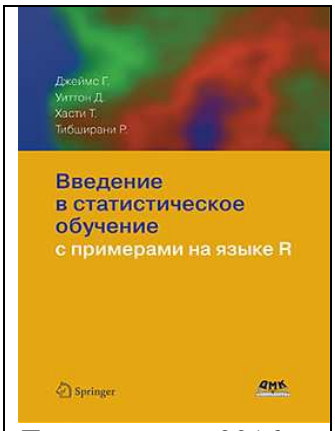
Если модель обычной регрессии оценивает условную вероятность математического ожидания *y*, то в случае многомерного отклика **Y** анализируется влияние предикторов на структурную изменчивость корреляционной матрицы (или иной матрицы дистанций). К настоящему времени с использованием такого подхода разработаны методы непараметрического дисперсионного анализа (McArdle, Anderson, 2001), анализа избыточности и канонического корреспондентного анализа (Legendre, Legendre, 2012), построения деревьев с многомерным откликом (De'Ath, 2002) и др. В экологических исследованиях процедуры оптимального проецирования многомерных данных по осям главных

компонент и одновременной оценки коэффициентов линейных канонических моделей объединяются под названием *прямая ординация*.

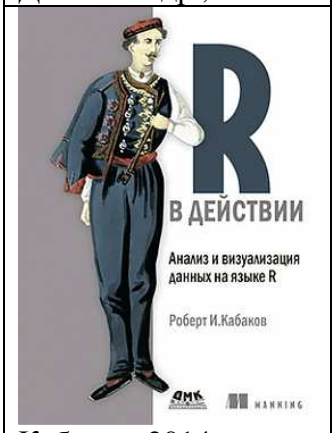
1.2. Статистическая среда R и ее использование в Data Mining

> *«Программное обеспечение системы R, которое постоянно совершенствуется совместными усилиями сотен энтузиастов и находится в открытом доступе в хранилище CRAN, – это выдающийся пример духа сотрудничества и кооперации в статистической науке.»*
>
> (Myers, Patil, 2012)

Необходимым условием современного статистического анализа данных является эффективное использование компьютерных программ, от функциональной полноты и алгоритмической продуманности которых зависит итоговая интерпретация результатов исследования и надежность выводов. И здесь есть прямой смысл обратиться к свободно распространяемой системе R, которая является наиболее полной, надежной и динамично развивающейся статистической средой, объединяющей язык программирования высокого уровня и мощные библиотеки программных модулей для вычислительной и графической обработки данных.



Джеймс и др., 2016



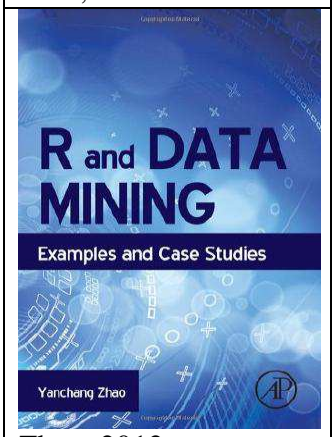
Кабаков, 2014

Сегодня статистическая среда R (http://www.r-project.org/) является безусловным лидером среди некоммерческих систем статистического анализа и постепенно становится незаменимой при проведении научно-технических расчетов в большинстве западных университетских центров и многих ведущих фирмах. Расширение библиотек программных модулей за счет усилий множества разработчиков привело к возникновению распределенной системы хранения и распространения пакетов R, то есть "CRAN" (от «Comprehensive R Archive Network»; http://cran.r-project.org), которая обладает также развитой системой информационной поддержки.

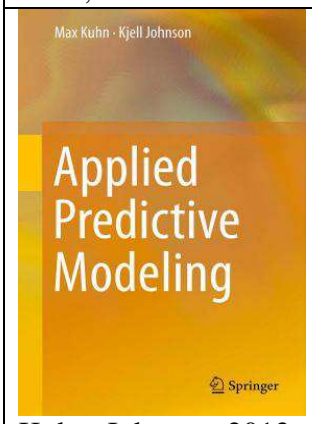
Всемерная поддержка научным сообществом данного проекта и широкое преподавание статистики на базе R обусловили то, что приведение скриптов на этом языке постепенно становится общепризнанным мировым "стандартом" как в журнальных публикациях, так и при неформальном общении ученых всего мира.

Значительный раздел среды R посвящен машинному обучению и собственно статистической обработке

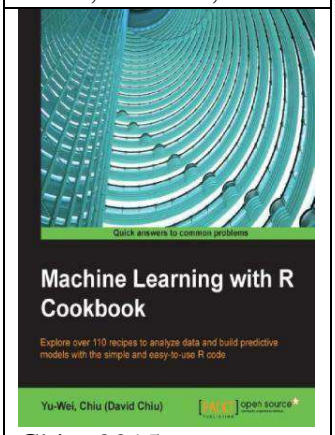
Люк, 2016



Zhao, 2012



Kuhn, Johnson, 2013



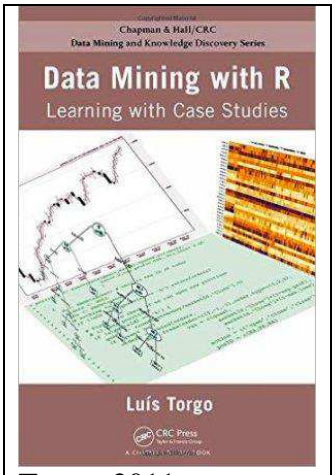
Chiu, 2015

массивов данных. Хорошим путеводителем к поиску того, какие ресурсы, пакеты и библиотеки могут быть использованы для решения тех или иных задач пользователя, является страница CRAN Task Views (http://cran.r-project.org/web/views). Там представлено, например, описание следующих коллекций пакетов по тематике Data Mining:

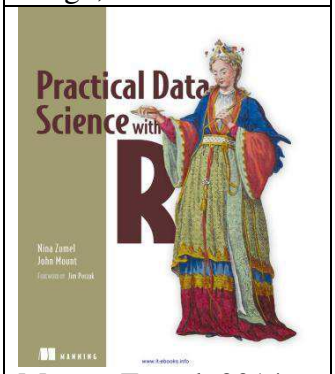
- Machine Learning and Statistical Learning (статистическое обучение);
- Cluster Analysis and Finite Mixture Models (кластерный анализ и модели с ограниченным числом латентных переменных);
- Time Series Analysis (анализ временных рядов);
- Multivariate Statistics (многомерная статистика);
- Analysis of Spatial Data (анализ пространственных данных).

Для реализации этих методов разработаны десятки "пакетов" (т.е. тематически обобщенных наборов функций), из которых мы рекомендуем обратить внимание на следующие, поскольку они будут фигурировать в скриптах последующих глав книги:

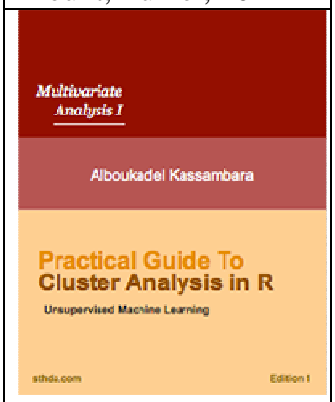
- `mgcv`, `nlme`, `car`, `Hims` – обобщенные линейные и аддитивные модели, модели со смешанными эффектами и проч. функции;
- `lars`, `glmnet` – модели с регуляризацией (гребневая, лассо);
- `pls`, `ppls` – регрессия на главные компоненты, метод регуляризованных частных наименьших квадратов;
- `data.tree`, `rpart`, `party`, `mvpart` – построение деревьев решений;
- `randomForest`, `gbm`, `bst`, `Boruta` – бэггинг, случайные леса, бустинг, ранжирование предикторов;
- `MASS`, `klaR`, `kknn`, `binda` – дискриминантный анализ, kNN-регрессия;
- `e1071`, `kernlab` – метод опорных векторов с разными ядрами;
- `nnet`, `neuralnet`, `kohonen` – нейронные сети;
- `VGAM`, `mlogit` – мультиноминальный логит;
- `pscl`, `MASS` – модели счетных данных;
- `vegan`, `labdsv`, `FactoMineR` – многомерные методы и ординация;
- `outliers` – анализ выбросов в данных;



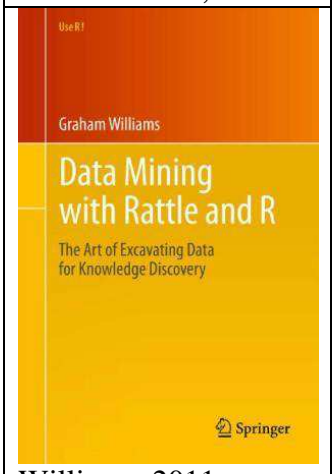
Torgo, 2011



Mount, Zumel, 2014



Kassambara, 2017



Williams, 2011

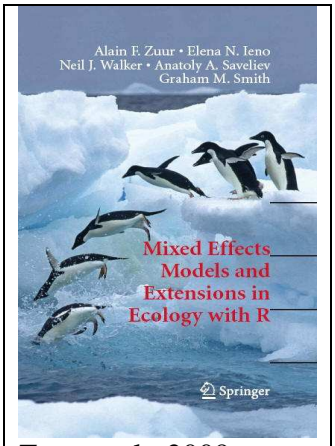
- `pROC` – анализ ROC-кривых;
- `boot`, `cvTools` – бутстреп, перекрестная проверка;
- `ForecastCombinations`, `caretEnsemble` – построение коллективных прогнозов;
- `cluster, pvclust dendextend` – кластерный анализ;
- `arules` – выделение ассоциативных правил;
- `TraMineR` – поиск закономерных последовательностей объектов или событий.

Наряду с характерным для R прогрессирующим "размножением" пакетов, наблюдается, к счастью, и своего рода обратная тенденция: а) создание комплексных пакетов с графическим интерфейсом в стиле кнопочного меню R Commander и б) интеграция функций из разных пакетов в рамках некоторой единой надстройки.

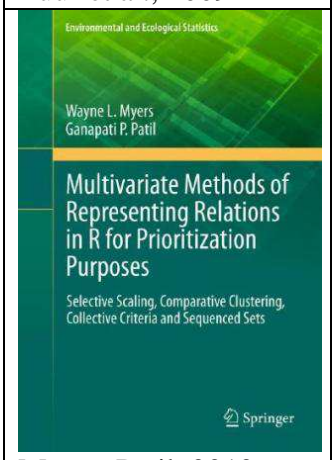
Прекрасным примером приложения с графическим интерфейсом и открытым исходным кодом, основанном на использовании ресурсов R, является Rattle (Williams, 2009). Для его инсталляции достаточно обычным образом установить пакет `rattle`. В состав этого пакета входят функции, которые выполняют трансформацию и визуализацию данных, кластеризацию, построение моделей опорных векторов, деревьев решений, бустинг-моделей и случайных лесов, нейронных сетей, выделение ассоциативных правил и т.д., а также развитые средства отображения и представления полученных результатов.

Эффективным средством интеграции десятков функций из различных пакетов для построении моделей классификации и регрессии является `caret`. Этот пакет автоматически осуществляет загрузку соответствующих функций, учитывает имеющиеся синтаксические различия между ними и выполняет комплексное тестирование качества моделей. Пакет `caret` особенно эффективен для оптимизации настраиваемых коэффициентов и параметров многих из этих моделей с использованием интеллектуальных подходов и алгоритмов ресэмплинга.

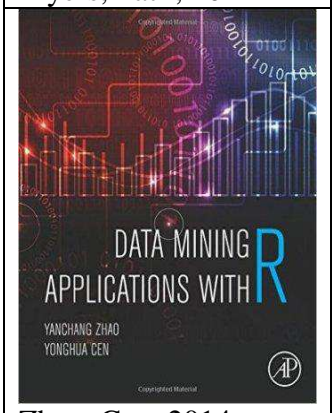
Все перечисленные выше пакеты подробно документированы и снабжены иллюстративным материалом. Кроме этого, в среде Интернет имеется большое количество различных методических пособий и консультационных пунктов по проблемам использования алгоритмов Data Mining в R. Например, Wikibook Data Mining Algorithms In R стремится интегрировать три блока



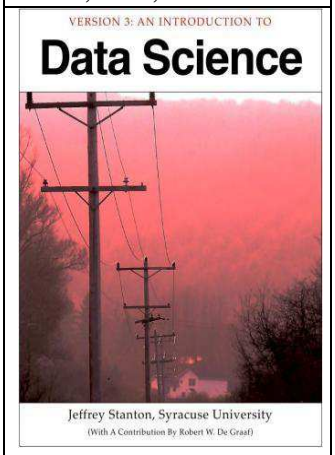
Zuur et al., 2009



Myers, Patil, 2012



Zhao, Cen, 2014



Stanton, 2013

информации для каждого метода: математическое описание и обоснование, детали выполнения, и примеры использования. Справочник по школе данных School of Data Handbook – хороший пример демонстрации веб-интерфейса связи с серверами, на которых формируются массивы информации, интересный тем, кто хочет узнать больше о процедурах обработки данных.

Для уверенного пользования языком R и быстрой ориентировки в богатом наборе функций этой статистической среды можно порекомендовать всегда иметь под рукой справочные карты, составленные разными авторами: «R Reference Card» (перевод на русский язык см. на странице http://r-analytics.blogspot.ru/p/blog-page_06.html), «R Reference Card by Jonathan Baron», «Time series Reference Card» и «Regression Reference Card» (https://cran.r-project.org/other-docs.html).

Я. Джао (Zhao, 2013) составил прекрасную карту ссылок «Reference Card for Data Mining», которая обеспечивает всестороннюю индексацию пакетов и функций R в соответствии с их категориями и функциональными возможностями. Ее последняя версия доступна по адресу http://www.rdatamining.com/docs, а перевод на русский язык с незначительными изменениями представлен нами в Приложении. В этом справочнике автор приводит не только способы реализации традиционных методов Data Mining и Text Mining, но и функции для доступа к серверам с исходной информацией через различные протоколы обмена, преобразования массивов данных, организации параллельных вычислений, формирования итоговых отчетов в разных форматах и т.д. Приведены функции доступа к библиотеке алгоритмов машинного обучения Weka (Waikato Environment for Knowledge Analysis) и интерфейс с другими языками программирования.

В карте даны также ссылки, касающиеся свободно распространяемого набора утилит и библиотек (Hadoop MapReduce, Apache Spark и т.д.) для организации распределённых вычислений в задачах обработки данных очень большого объема на компьютерах, объединенных в вычислительные «кластеры».

Список только признанных фундаментальных монографий типа «Используем R!» (Use R!) насчитывает уже несколько сотен томов. В многочисленных фундаментальных руководствах детально описаны функции и пакеты R, использующие широкий набор статистических методов, таких как классические линейные и нелинейные регрессионные модели, различные формы дисперсионного анализа, алгоритмы иерархической кластеризации, анализ временных рядов, деревья классификации и регрессии (Faraway, 2006; Fox, 2008; Zuur et al., 2009; Myers, Patil, 2012; Mount, Zumel, 2014, 2016; Kuhn, Johnson, 2013; Stanton, 2013; Кабаков, 2014; Chiu, 2015, Kassambara, 2017). Отдельно можно отметить библиографические источники, в названии которых имеется термин "Data Mining": Torgo (2011), Zhao (2012), Zhao, Cen (2014). Большинство из этих книг с разной степенью легальности доступно для скачивания в Интернете (список 9 свободно распространяемых пособий приводится, в частности, на www.kdnuggets.com).

1.3. О чем эта книга и чего в ней нет

Выполняя работу над нижеизложенным текстом, мы поставили перед собой несколько целей:

Шипунов и др., 2014

Мастицкий, Шитиков, 2015

В очередной раз привлечь внимание русскоязычных читателей к статистической среде R

Как показала практика, исследователи на постсоветском пространстве, связанные с анализом данных, ограничиваются в своей работе, как правило, средствами Microsoft Excel или предпочитают использовать недостаточно гибкую программу Statistica, стоимость одной лицензии которой составляет более 85 тыс. руб. Мы им искренне сочувствуем: они лишаются и потраченных денег и удовольствия работать в по-настоящему интеллектуальной и богатой методами среде.

До недавнего времени основным препятствием для русскоязычных пользователей при освоении R являлось, безусловно, то, что почти вся документация по этой системе существовала на английском языке. Однако в последние годы эта проблема потеряла актуальность в связи с появлением весьма полных пособий как отечественных авторов (Зарядов, 2010; Шипунов и др., 2012; Мастицкий, Шитиков, 2015), так и переводов процитированных выше лучших зарубежных монографий. Именно поэтому в настоящем пособии не приводится

Шитиков, Розенберг, 2014

Шитиков, 2016

Храмов, 2016

Мастицкий, 2016

развернутого описания языка R и его стандартных функций, поскольку все это достаточно полно представлено в доступных литературных источниках.

Дать краткое, но внятное описание сущности используемых статистических методов

Было бы бессмысленной тратой сил писать очередное руководство по математическим основам, например, классификации и регрессии. Ответы на все вопросы, характерные для традиционной прикладной статистики, даны в десятках тысяч прекрасно изданных книг (Кендалл, Стьюарт, 1973, 1976; Бокс, Дженкинс, 1974; Дрейпер, Смит, 1986, 1987; Ллойд, Ледерман, 1989, 1990; Айвазян, Мхитарян, 1998; Everitt, Howell, 2005; Legendre, Legendre, 2012 и многие другие).

В то же время, в русскоязычных изданиях достаточно скупо представлены современные методы Data Mining и машинного обучения (Вапник, Червоненкис, 1974; Вапник, 1984; Горбань и др., 1998; Загоруйко, 1999; Чубукова, 2006; Зайцев, 2009; Джеймс и др., 2016). И тем более трудно найти наглядные примеры реализации таких алгоритмов поиска скрытых закономерностей в данных, как выделение ассоциативных правил, анализ последовательностей, построение деревьев классификации и регрессии, случайные леса, бустинг, нейросетевые технологии, метод опорных векторов, генетический алгоритм и др.

Мы по мере сил постарались заполнить этот пробел. Использование перечисленных методов показано в сравнении с детально описанными классическими процедурами построения статистических моделей, таких как логистическая регрессия и дискриминантный анализ (Классификация…, 1980; Афифи, Эйзен, 1982; Айвазян и др., 1989; Ким и др., 1989).

Показать, какие из статистических методов с высокой вероятностью хорошо сработают в большинстве ситуаций

Вряд ли можно составить единый рейтинг эффективности методов обработки данных, неизменный на все случаи жизни. Много зависит от объема и характера распределения данных, цели исследования и проч. Однако нельзя не учитывать многочисленные примеры закономерностей, прослеживаемых на практике.

Использованные в нашей книге примеры наборов данных малочисленны и тривиальны, чтобы делать сколько-нибудь обоснованные выводы. Зато мы подробно останавливаемся на процедурах селекции информативного набора предикторов, оптимизации параметров моделей, верификации и сравнения их адекватности. Все это позволяют выполнить функции пакета `caret`, демонстрация возможностей которого красной нитью проходит через многие последующие главы.

Большие надежды мы возлагали на коллективы моделей, эффективность прогноза которых часто оказывается лучше индивидуальной модели (Розенберг и др., 1993), но желаемого результата не получили.

Предоставить набор скриптов, позволяющих читателю воспроизвести представленные расчеты, либо использовать их на собственных примерах

Освоение статистической среды R и реализованных в ней методов легче всего выполнять в режиме "обучения по прецедентам", т.е. на примерах. Поэтому при рассмотрении тех или иных задач обычно принято приводить как отдельные фрагменты скриптов, так и законченные функции.

Мы хотели сделать свое изложение, по возможности, разноплановым и иллюстрированным. Например, убедительным доказательством наличия закономерности в данных часто может служить, их качественная графическая иллюстрация (Зиновьев, 2000; Maindonald, Braun, 2010). Поэтому, где было возможно, мы старались включить в наши скрипты примеры визуализации данных с использованием пакета `qqplot2` (Мастицкий, 2016) или других пакетов, поддерживающих эту графическую систему.

В одной из дискуссий в Интернете промелькнула мысль «Data Mining – это обработка данных без применения статистики». Видимо тут имелось в виду скупое употребление псевдо-теоретической фразеологии вроде «истинных параметров некой мифической случайной величины или теоретической функции регрессии», которые необходимо оценить. Избегнуть подобных оборотов постарались и мы, чем рискуем навлечь недовольство ортодоксальных статистиков.

К счастью, большинство описанных ниже методов не апеллируют к вероятностной природе обрабатываемых данных и их использование не всегда связано с (безосновательными) надеждами на закон нормального распределения. Однако это нисколько не мешает легко сгенерировать с их помощью продуктивные научные гипотезы, которые могут оказаться нетривиальными, практически полезными и удобными для интерпретации. В тех же случаях, когда проверка статистической значимости необходима, мы (следуя разработчикам пакетов R) активно использовали методы

ресэмплинга: бутстреп, рандомизацию и перекрестную проверку (Эфрон, 1988; Edgington, 1995; Davison, Hinkley, 2006; Шитиков, Розенберг, 2014).

Эффективными средствами получения оценок параметров и их апостериорных распределений является байесовский подход и итерационные методы Монте-Карло, использующие цепи Маркова (MCMC – Monte Carlo Markov chain). Их разработка иногда трактуется как наиболее существенный прорыв в статистике за последние несколько десятков лет. Однако мы не стали касаться этой важной темы, требующей подробного и обоснованного обсуждения.

В настоящее пособие включена совокупность методических сообщений, опубликованных авторами за последнее время в блоге «R: Анализ и визуализация данных» (http://r-analytics.blogspot.com). Нам показалась целесообразной идея представить для удобства читателей весь этот несколько разобщенный материал в концентрированной форме, а также расширить некоторые разделы для полноты изложения.

Прокомментируем кратко содержание глав книги:

Поскольку большинство алгоритмов машинного обучения используют одинаковые процедуры тестирования и сходный набор критериев оценки эффективности моделей, мы постарались обобщить их описание в *главе 2*. Там же мы приводим общую постановку задачи кластеризации и проецирования многомерных данных по осям главных координат сокращенного пространства. Эта глава играет роль своего рода "введения в тему" для всех последующих глав.

Глава 3 содержит подробное описание функций пакета `caret`, а разделы *главы 4* показывают, как с его помощью выполнить построение и тестирование различных моделей регрессии.

Бинарные матрицы являются весьма распространенной формой представления исходных данных. *В главе 5* мы не только рассматриваем применимость для них традиционных методов моделирования, но и даем описание алгоритмов, использующих логические процедуры генерации непротиворечивых высказываний и ассоциативных правил. Там же на основе пакета `TraMineR` мы представляем инструментарий для анализа последовательностей знаков или событий.

Главы 6 и 7 объединяют описание и особенности реализации различных алгоритмов классификации на два и более класса. Как и в главе 4, особое внимание уделено процедурам тестирования и подстройки гиперпараметров моделей.

Глава 8 посвящена сразу трем проблемам: методам классификации порядковых данных, созданию комплексных прогнозов на основе "коллектива" моделей и построению различных версий моделей для счетных данных по материалам сайта http://www.highstat.com и книги А. Зуура с коллегами (Zuur et al., 2009).

Глава 9 описывает различные алгоритмы непрямой и прямой ординации многомерных наблюдений. Предлагается процедура построения кумулятивной кривой распределения чувствительности видов (SSD).

Глава 10 (за исключением раздела о самоорганизующихся картах Кохонена) представляет собой изложение материалов сайта http://www.sthda.com/ и книги А. Кассамбары (Kassambara, 2017) по кластерному анализу в среде R.

Демонстрации возможностей пакета `rattle` посвящена целиком глава 11, основанная на публикациях Г. Вильямса (Williams, 2009, 2011).

Как видно из рубрикатора глав, в книге довольно много заимствований из различных источников и, вероятно, она имела бы трудности при проверке в системе «Антиплагиат». Однако все в науке в той или иной степени плагиат, поскольку подавляющее большинство исследований лишь опровергает или подтверждает уже высказанные гипотезы. Вспомним, например, историю об Амбруазе Паре, который многое переписал у Везалия, а потом придумал блестящую формулу: «Я просто зажег свою свечку от соседней горящей свечи».

По вполне понятным причинам мы не смогли охватить в книге весь перечень методов построения статистических моделей и соответствующих пакетов. Например, в спектральном анализе и хемометрике (https://ru.wikipedia.org) чрезвычайно популярны методы, основанные на технике условного математического ожидания, и, наряду с методом частных наименьших квадратов (PLS) и регрессией на главные компоненты (PCR), рекомендуется использовать параллельный факторный анализ (PARAFAC), анализ Тюкера (Tucker) и др.

Мы также совсем не затронули такие обширные разделы Data Mining, как анализ временных рядов и моделирование пространственно-распределенных данных. Мы сожалеем, что уделили столь важным темам недостаточное внимание, но это вряд ли можно считать существенным недостатком книги, поскольку детальный разговор планируется впереди, в том числе на страницах блога http://r-analytics.blogspot.com.

В заключение нельзя не высказать благодарность нашим коллегам, которые помогали нам советами, а также родным и близким, самоотверженно поддержавшим нас в этом предприятии.

2. СТАТИСТИЧЕСКИЕ МОДЕЛИ: КРИТЕРИИ И МЕТОДЫ ИХ ОЦЕНИВАНИЯ

2.1. Основные шаги построения и верификации моделей

Построение и последующая проверка работоспособности полученных моделей представляет собой сложный и итеративный процесс, по итогам которого достигается приемлемый уровень уверенности исследователя в том, что результаты, получаемые с помощью итоговой модели, окажутся практически полезными. При этом выделяются следующие стандартные шаги (Kuhn, Johnson, 2013), которые мы будем подробно обсуждать далее:

1. Разведочный анализ данных (exploratory data analysis), главная цель которого – изучение статистических свойств имеющихся в наличии выборок (распределение переменных, наличие выбросов, необходимость трансформации и др.) и выявление характера взаимосвязей между откликом и предикторами (Mount, Zumel, 2014).

2. Выбор методов построения моделей и спецификация систематической части последних. Например, модели типа "доза-эффект" в биологии на одном и том же исходном материале могут быть построены с использованием самых различных функций: логистической, экспоненциальной, Вейбулла, Гомперца, Михаелиса-Ментен, Брейна-Кузенса и т.д. (Шитиков, 2016).

3. Оценка параметров моделей и их *диагностика* (Мастицкий, Шитиков, 2014). Диагностика и оценка валидности (model validation) включает в себя ряд стандартных процедур. Так, в случае с классическими регрессионными моделями, подгоняемыми по методу наименьших квадратов, выполняются: а) проверка статистической значимости модели в целом и анализ неопределенности оцененных коэффициентов; б) проверка допущений в отношении остатков модели; в) обнаружение необычных наблюдений и выбросов; г) построение графиков, позволяющих оценить соответствие модели структуре анализируемых данных.

4. Анализ вклада отдельных предикторов и селекция оптимальной комбинации из них (model selection). Оценка качества каждой модели-претендента (model evaluation) по совокупности объективных критериев эффективности, включая тестирование на порции "свежих" данных, не участвовавших в процессе оценивания коэффициентов.

5. Ранжирование нескольких альтернативных моделей и, при необходимости, подстройка их важнейших параметров (model tuning).

Рассматриваемые в этом разделе параметрические регрессионные модели в классическом представлении являются аппроксимацией математического ожидания отклика Y по обучающей выборке с помощью неизвестной функции регрессии $f(\ldots)$:

$$E(Y|x_1, x_2, \ldots, x_m) = f(\boldsymbol{\beta}, x_1, x_2, \ldots, x_m) + \boldsymbol{\varepsilon},$$

где остатки $\boldsymbol{\varepsilon}$ отражают ошибку модели, т.е. необъяснимую случайную вариацию наблюдаемых значений зависимой переменной относительно ожидаемого среднего значения.

С практической точки зрения, тестирование таких моделей ставит своей задачей выявление следующих основных проблем их возможного использования:

- *Смещение* (bias), или *систематическая ошибка* модели (сдвиг предсказываемых значений на некоторую трудно объяснимую величину);
- Высокая *случайная дисперсия* прогноза, определяемая чаще всего излишней чувствительностью модели к небольшим изменениям в распределении обучающих данных;
- *Неадекватность* – тенденция модели не отражать основных закономерностей генеральной совокупности данных и основываться на случайных флуктуациях обучающей выборки;
- *Переусложнение* (overfitting) модели, которое «так же вредно, как и ее недоусложнение» (Ивахненко, 1982).

Действительно, для любого проверочного наблюдения $\boldsymbol{x}_0$ математическое ожидание *среднеквадратичной ошибки* его прогноза можно разложить на сумму трех величин: дисперсии $f(x_0)$, квадрата смещения $f(x_0)$ и дисперсии остатков $\boldsymbol{\varepsilon}$ (подробнее см. James et al., 2013):

$$E[y_0 - f(\boldsymbol{x}_0)]^2 = Var[f(\boldsymbol{x}_0)] + [Bias(f(\boldsymbol{x}_0))]^2 + Var(\boldsymbol{\varepsilon}),$$

где *Bias* означает смещение, а *Var* – дисперсию. Здесь мы предположили, что неизвестная истинная функция $f(\ldots)$ была получена на большом числе обучающих выборок, а отклонения y_0 вычислялись по каждой из множества моделей с последующим усреднением результатов.

Из приведенного уравнения следует, что для минимизации ожидаемой ошибки прогноза мы должны подобрать такую модель, для которой одновременно достигаются низкое смещение и низкая дисперсия. Обратите внимание, что дисперсия никогда не может быть ниже некоторого уровня *неустранимой ошибки* $Var(\boldsymbol{\varepsilon})$ (irreducible error).

Выше мы упомянули также феномен переусложнения модели, при котором наблюдается низкая дисперсия прогноза на обучающей совокупности, но часто получаются непредсказуемые результаты при тестировании блоков "свежих" данных, не участвовавших в построении модели.

В качестве простого примера используем данные по электрическому сопротивлению (Ом) мякоти фруктов киви в зависимости от процентного содержания в ней сока. Таблица с этими данными (`fruitohms`) входит в состав пакета DAAG, который является приложением к книге Maindonald (2010). В рассматриваемом примере у нас есть лишь один предиктор – содержание сока в мякоти фруктов `juice`, и было бы вполне логичным на первоначальном этапе рассмотреть простую линейную регрессию.

Для визуализации данных воспользуемся одним из лучших графических пакетов для R – `ggplot2`, который позволяет строить как

всевозможные простые диаграммы рассеяния, так и гораздо более сложные графики, включающие, например, двумерные диаграммы распределения плотности (Мастицкий, 2016). Легко показать линию регрессии с ее доверительными интервалами, которые накрывают менее половины наблюдений (рис. 2.1):

```
# Загрузка и визуализация данных в виде диаграммы рассеяния:
library(DAAG)
data("fruitohms")
library(ggplot2)
ggplot(fruitohms, aes(x = juice, y = ohms)) + geom_point() +
    stat_smooth(method = "lm") +
    xlab("Содержание сока, %") +
    ylab("Сопротивление, Ом")
```

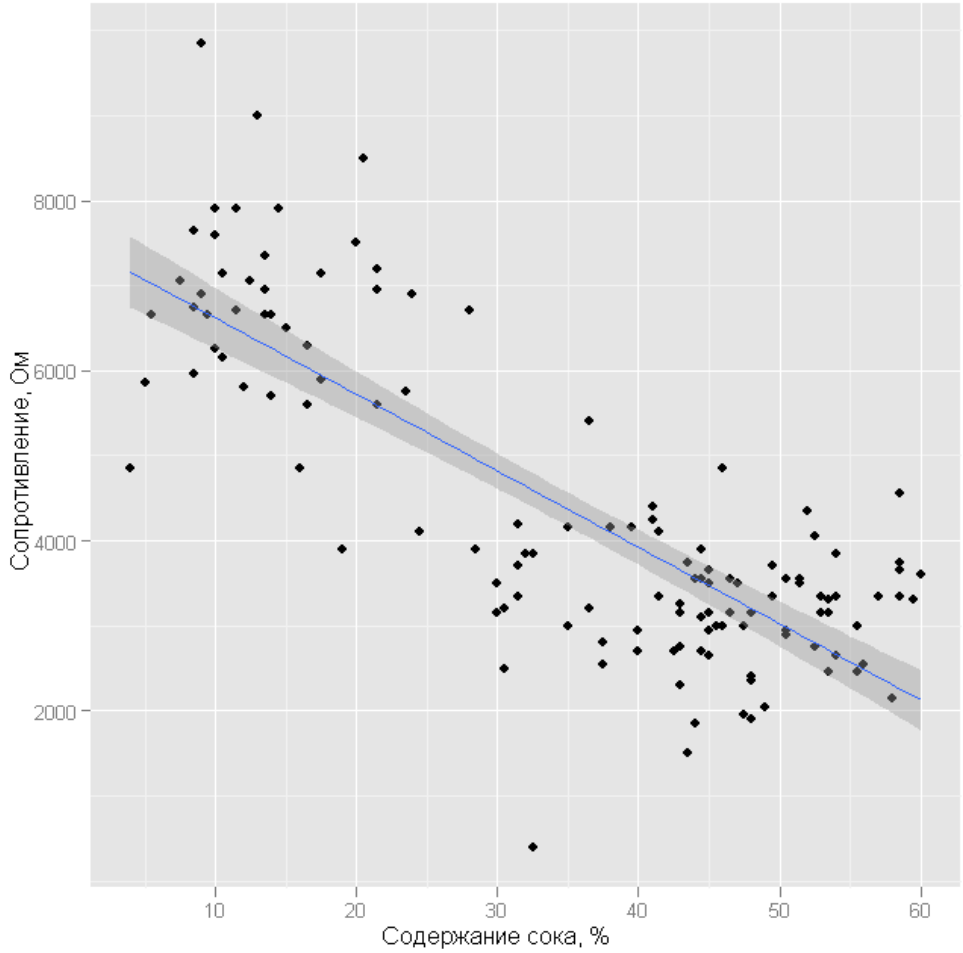


Рис. 2.1. График линейной зависимости электрического сопротивления мякоти плодов киви от содержания сока

Все критерии оценки качества классических моделей регрессии `f` так или иначе основаны на анализе остатков (residuals), т.е. разностей между прогнозируемыми `f(x[i,])` и наблюдаемыми `y[i]` значениями, где **`x`** – матрица независимых переменных. Базовыми критериями являются сумма квадратов остатков `RSS` (residual sum of squares), корень из среднеквадратичной ошибки `RMSE` (root mean square error) и стандартное отклонение остатков `RSE` (residual standard error). Для унификации

обозначений в листингах кода перейдем от предметных обозначений переменных к их формальным эквивалентам (`y` и `x` соответственно). Преобразуем заодно омы в килоомы, чтобы избежать слишком больших значений:

```
x <- fruitohms$juice
y <- fruitohms$ohms/1000
#  Строим модель с одним предиктором
n <- dim(as.matrix(x))[1]; m <- dim(as.matrix(x))[2]
M_reg <- lm(y ~ x); pred <- predict(M_reg)
RSS <- sum((y - pred) * (y - pred))
RMSE <- sqrt(RSS/n)
RSE <- sqrt(RSS/(n - m - 1))
c(RSS, RMSE, RSE)
```

```
[1] 158.706891   1.113507   1.122309
```

Мы не можем сказать с определенностью, малы или велики эти ошибки, поскольку все зависит от шкалы измерения `y`, и поэтому необходимо определиться с набором "эталонов" для сравнения. Тогда, выбрав некоторый подходящий критерий качества, мы сможем оценить, насколько эффективность тестируемой модели по этому критерию отличается от эталона. Такими эталонными моделями являются (Mount, Zumel, 2014):

1. *Нулевая модель*, определяющая нижнюю границу качества. Если тестируемая модель значимо не превосходит по своей эффективности нулевую, то результат моделирования можно трактовать как неудачу. Обычно нуль-модель строится в соответствии с двумя принципами: а) она является независимой и не усматривает какой-либо связи между переменными и откликом, б) эквивалентна константе и дает на выходе один и тот же результат для всех возможных входов. Как правило, это среднее значение отклика для располагаемой выборки, либо наиболее популярная категория при классификации.
2. *Модель с минимальным уровнем байесовской ошибки* (Bayes rate model) – самая лучшая модель для данных, имеющихся под рукой. Она, как правило, основывается на всех имеющихся переменных (т.е. является максимально "насыщенной" – saturated model) и ее ошибка определяется только набором наблюдений с разными значениями отклика y при одних и тех же x_1, ..., x_n. Если тестируемая модель по критерию эффективности значимо лучше нуль-модели и приближается к максимально насыщенной, то процесс селекции моделей можно считать завершенным.
3. *Модель с одной наиболее информативной переменной*. Если в процессе селекции более сложные модели по своей эффективности не превосходят модель с одной переменной, то включение дополнительных переменных вряд ли имеет смысл.

Традиционными оценками качества аппроксимации данных являются коэффициент детерминации `Rsquared`, дисперсионное отношение Фишера `F` и соответствующее ему p-значение. Отметим, что F-критерий интерпретируется как мера превышения точности предсказания отклика `y` построенной модели над нуль-моделью:

```
Rsquared <- 1 - RSS/sum((mean(y) - y)^2)
F <- (sum((mean(y) - pred)^2)/m)/(RSS/(n - m - 1))
p <- pf(q = F, df1 = m, df2 = (n - m - 1), lower.tail = FALSE)
c(Rsquared, F, p)
```

```
[1] 6.387089e-01 2.227492e+02 1.234110e-29
```

Разумеется, все эти величины можно получить и с помощью стандартной функции `summary()`, но нам показалось интересным показать всю "кухню" их расчетов.

```
summary(M_reg)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.519404   0.233779   32.16   <2e-16 ***
x           -0.089877   0.006022  -14.93   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.122 on 126 degrees of freedom
Multiple R-squared: 0.6387,     Adjusted R-squared: 0.6358
F-statistic: 222.7 on 1 and 126 DF,  p-value: < 2.2e-16
```

```
anova(M_reg)
```

```
Analysis of Variance Table
Response: y
           Df Sum Sq Mean Sq F value    Pr(>F)
x           1 280.57  280.57  222.75 < 2.2e-16 ***
Residuals 126 158.71    1.26
```

Мы убедились в том, что линейная модель со статистически значимыми коэффициентами по всем формальным критериям вполне адекватна полученным данным. Для проверки условия однородности дисперсии остатков в пакете `car` имеется функция `ncvTest()` (от "non-constant variance test" – "тест на непостоянную дисперсию"), которая позволяет проверить нулевую гипотезу о том, что дисперсия остатков никак не связана с предсказанными моделью значениями.

```
ncvTest(M_reg)
```

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 4.647123    Df = 1     p = 0.03110564
```

Те же данные мы можем использовать для построения обобщенной линейной модели (general linear model, GLM), задав в качестве аргумента `family` функции `glm()` гауссовый характер распределения данных `"gaussian"`. Вместо минимизации суммы квадратов отклонений эта модель ищет экстремум логарифма функции наибольшего правдоподобия (log maximum likelihood), вид которой зависит от заданного распределения.

В общем случае, значение функции правдоподобия численно равно вероятности того, что модель правильно предсказывает любое предъявленное ей наблюдение из заданной выборки. Логарифм функции правдоподобия `LL` будет всегда отрицательным, и, поскольку log(1) = 0, то для оптимальной

модели, прогнозирующей наименее ошибочное значение отклика, значение LL будет стремиться к 0. Часто для оценки расхождений между наблюдаемыми и прогнозируемыми данными вместо LL используют остаточный *девианс*[1] (deviance), который определен как D = -2(LL - S), где S – правдоподобие "насыщенной модели" с минимально возможным уровнем неустранимой ошибки. Обычно принимают S = 0, поскольку наилучшая модель выполняет, как правило, верное предсказание.

Чем меньше выборочный остаточный девианс D, тем лучше построенная модель. Аналогично можно рассчитать логарифм правдоподобия и девианс для нулевой модели D.null. Тогда адекватность модели определяется соотношением девианса остатков D и нуль-девианса D.null путем вычисления псевдо-коэффициента детерминации PRsquare. Девианс-анализ является обобщением техники дисперсионного анализа и статистическую значимость разности двух значений девианса (D.null – D) можно оценить по критерию χ^2:

```
M_glm <- glm(y ~ x)
lgLik <- logLik(M_glm)
D.null <- M_glm$null.deviance
D <- M_glm$deviance
df <- with(M_glm, df.null - df.residual)
p <- pchisq(D.null - D, df, lower.tail = FALSE)
PRsquare = 1 - D/D.null
c(lgLik, D, D.null, p, PRsquare)
```

```
[1] -1.9538e+02  1.5870e+02  4.3927e+02  5.6410e-63  6.3870e-01
```

Те же величины можно получить с использованием функции summary():

```
summary(M_glm)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.519404   0.233779   32.16   <2e-16 ***
x           -0.089877   0.006022  -14.93   <2e-16 ***
---
    Null deviance: 439.28  on 127  degrees of freedom
Residual deviance: 158.71  on 126  degrees of freedom
AIC: 396.77
```

```
with(M_glm, null.deviance - deviance)
```

```
 [1]  280.57
```

```
anova(M_glm, glm(x ~ 1), test = "Chisq")
```

```
Analysis of Deviance Table
     Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                   127     439.28
x     1   280.57       126     158.71 < 2.2e-16 ***
```

Нетрудно заметить, что классическая и обобщенная линейные модели в случае нормального распределения дают идентичные результаты и отличаются лишь по характеру используемой терминологии.

---

[1] Этот термин в русскоязычной литературе пока не устоялся: используются также *девиация* (Ллойд, Ледерман, 1990) и *аномалия* (International Statistical Institute)

Рассмотрим теперь обоснованность выбора систематической части модели. Поверим утверждениям, что наиболее совершенным инструментом экспертизы нелинейности пока остается глаз человека, и отметим на рис. 2.1 некоторую асимметрию распределения точек относительно линии регрессии: в середине шкалы *x* модель имеет тенденцию к завышению значений *y*, тогда как в области низких и высоких значений *x* наблюдается обратная тенденция.

Прекрасным способом оценить возможную нелинейность зависимости является использование кривых сглаживания, рассчитанных при помощи моделей локальной регрессии с использованием функции `loess()` или сплайнов. Вместо обычного облака рассеяния точек покажем двумерную гистограмму `hexbin` (или "сотовую карту"), в которой данные распределены по ячейкам и число наблюдений в каждой ячейке представлено соответствующим оттенком цвета (рис. 2.2):

```
library(hexbin)
ggplot(fruitohms, aes(x=juice, y=ohms)) +
    geom_hex(binwidth=c(3, 500)) +
    geom_smooth(color="red", se=F) +
    xlab("Содержание сока, %") +
    ylab("Сопротивление, Ом")
```

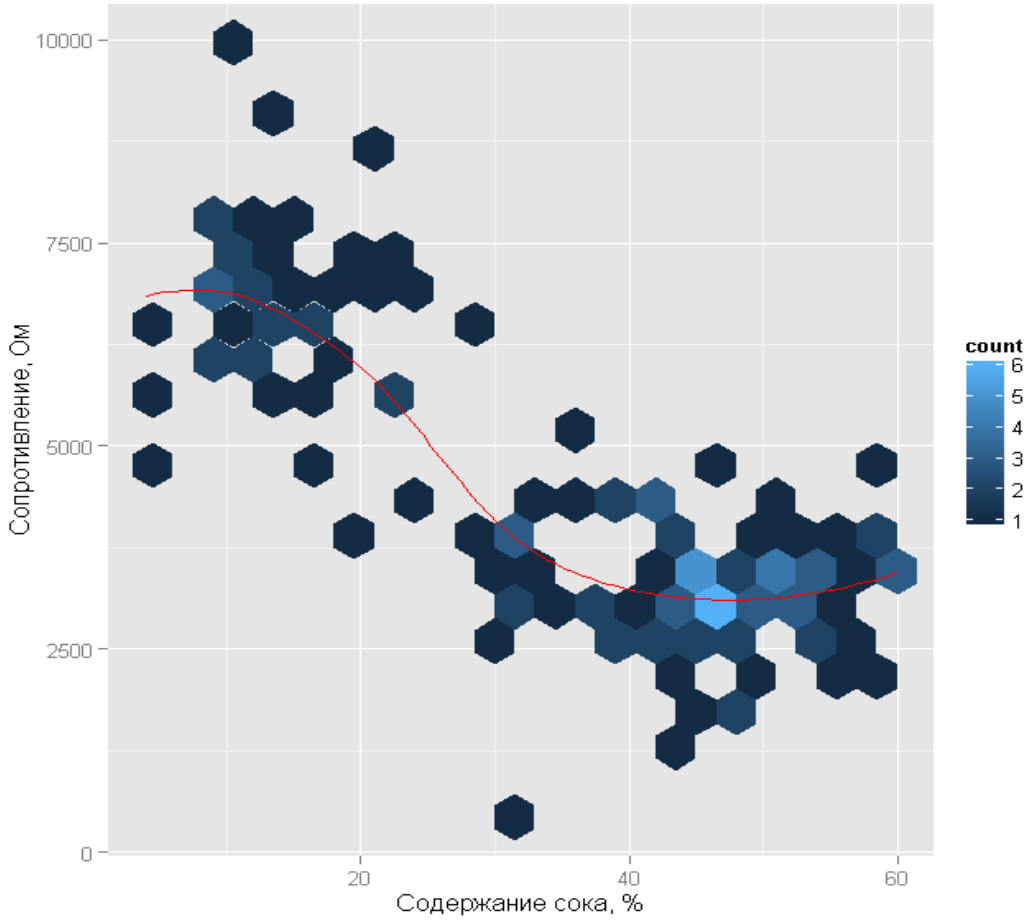


Рис. 2.2. Кривая сглаживания, описывающая зависимость электрического сопротивления мякоти плодов киви от содержания сока

Поскольку функциональная форма истинной модели нам неизвестна, сделаем предположение, что удовлетворительная аппроксимация данных может быть выполнена полиномиальной зависимостью. Отметим, что при увеличении степени *m* полинома любые критерии эффективности, основанные на остатках (`RSE`, `Rsquared`, `F`) будут монотонно улучшаться, поскольку каждая новая модель будет полнее отражать характер зависимости, имеющий место в обучающей выборке (рис. 2.3).

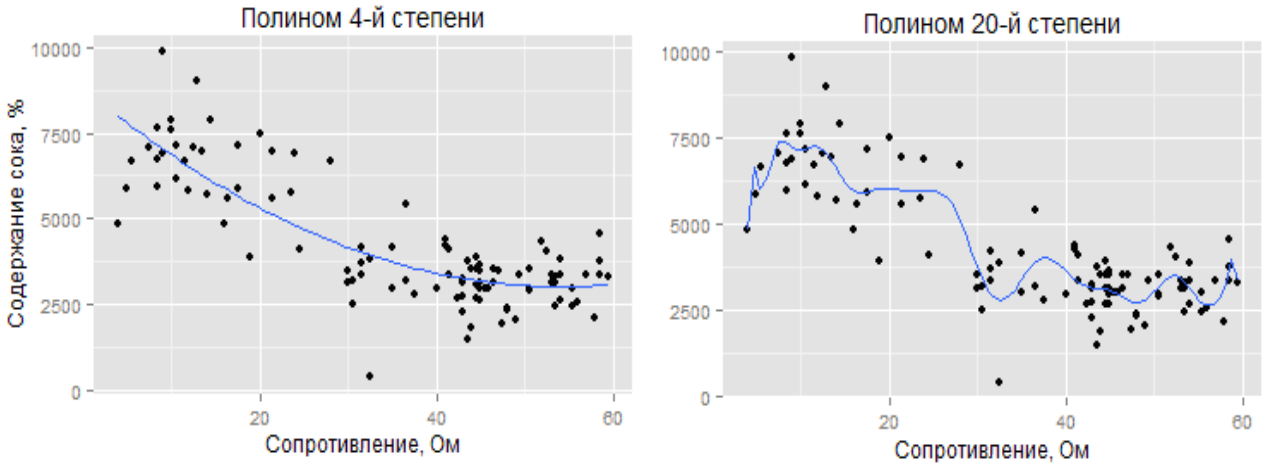


Рис. 2.3. Аппроксимация данных полиномами разных степеней

Однако решение с минимально возможной ошибкой на обучающей выборке оказывается неоправданно сложным – аппроксимирующая функция старательно учитывает все случайные флуктуации измерений и не в состоянии отследить основную форму тренда. Такая ситуация называется переобучением модели и выражается она в том, что на новых независимых данных такие модели могут вести себя довольно непредсказуемым образом. Теряется обобщающая способность решения (model generalization), связанная как со способностью модели описать наиболее характерные закономерности изучаемого явления, так и с универсальной применимостью прогнозов к широкому множеству новых примеров. Поэтому построение модели оптимальной сложности, в которой сбалансированы точность и устойчивость прогноза, является фундаментальной задачей совершенствования методов моделирования.

Одним из путей поиска наилучшей аппроксимирующей функции является использование метода *регуляризации*, когда задача минимизации ошибки решается на основе критериев, налагающих "штраф" за увеличение сложности модели. Если `D` – выборочный остаточный девианс, `k` – число свободных параметров модели, а `n` – объем обучающей выборки, то можно определить семейство следующих весьма популярных критериев, которые обобщены под названием *информационных* (Burnham, Anderson, 2002):

- классический критерий Акаике `AIC = D + 2*k;`
- байесовский критерий Шварца `BIC = D + k*ln(n);`
- скорректированный критерий Акаике AIC_c= `AIC+2k(k+1)/(n-k-1).`

При построении параметрических моделей с использованием функций R рассчитать эти критерии можно различными способами:

```
k <- extractAIC(M_glm)[1]; AIC <- extractAIC(M_glm)[2]
AIC <- AIC(M_glm); AICc <- AIC(M_glm) + 2*k*(k+1)/(n-k-1)
BIC <- BIC(M_glm); BIC <- AIC(M_glm, k=log(n))
c(AIC, AICc, BIC)
```

```
[1] 396.7719 396.8679 405.3280
```

Оптимальной считается такая модель, которой соответствуют субэкстремальные значения критериев качества (например, минимум AIC). Рассмотрим пример использования информационных критериев для выбора оптимального числа параметров полиномиальной регрессии (рис. 2.4):

```
# Построение моделей со степенью полинома от 1 до 7
max.poly <- 7
# Создание пустой таблицы для хранения значений AIC и BIC,
# рассчитанных для всех моделей, и ее заполнение
AIC.BIC <- data.frame(criterion = c(rep("AIC", max.poly),
rep("BIC", max.poly)), value = numeric(max.poly*2),
degree = rep(1:max.poly, times = 2))
for(i in 1:max.poly)  {
     AIC.BIC[i, 2] <- AIC(lm(y ~ poly(x, i)))
     AIC.BIC[I + max.poly, 2] <- BIC(lm(y ~ poly(x, i)))
}
# График AIC и BIC для разных степеней полинома
qplot(degree, value, data = AIC.BIC,
      geom = "line", linetype = criterion) +
      xlab("Степень полинома") + ylab("Значение критерия")
```

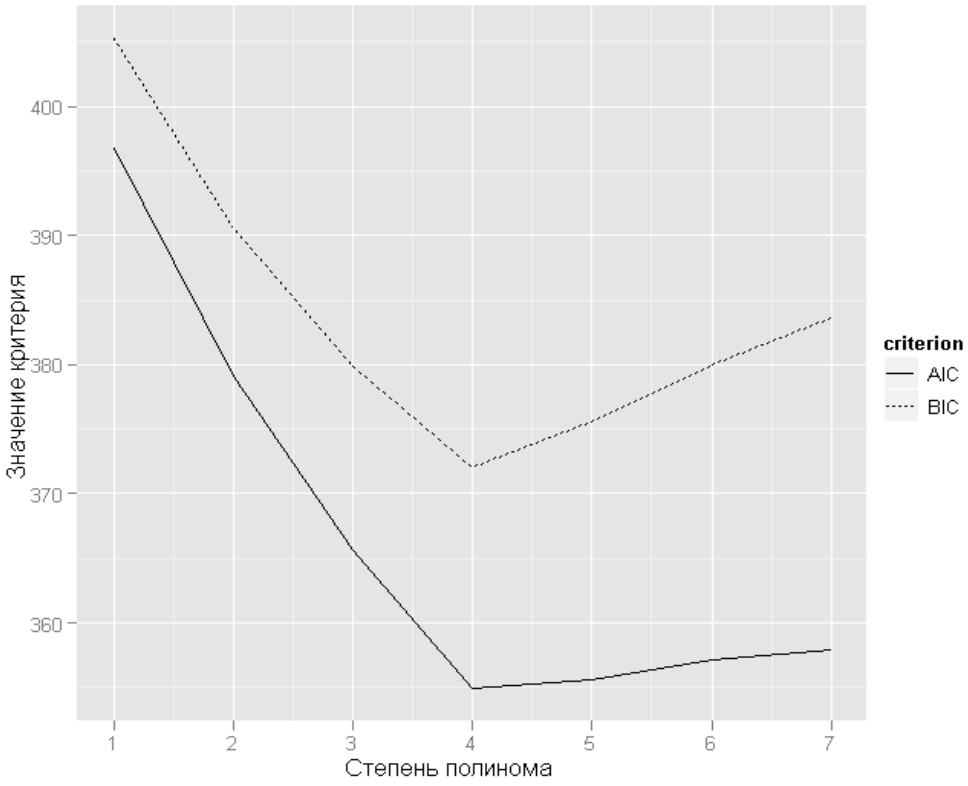


Рис. 2.4. Поиск оптимальной степени полинома с использованием информационных критериев

Минимальные значения информационных критериев соответствуют выводу, что наилучшая модель – полином 4-й степени.

2.2. Использование алгоритмов ресэмплинга для тестирования моделей и оптимизации их параметров

Фундаментальным для статистики является рассуждение о соотношении свойств случайных выборок и генеральной совокупности, из которых они были извлечены. Любой эксперимент в принципе ограничен некоторым (чаще всего, небольшим) множеством наблюдений, причем никакие сверхинтеллектуальные методы обработки не являются панацеей от влияния неучтенных факторов или систематических погрешностей. Один из возможных путей надежного оценивания свойств данных заключается в использовании компьютерно-интенсивных (computer-intensive) технологий, объединенных общим термином "*численный ресэмплинг*" (англ. resampling) или, как их иногда называют в русскоязычной литературе, "методов генерации повторных выборок" (Эфрон, 1988; Edgington, 1995; Davison, Hinkley, 2006). Ключевая особенность этой технологии заключается в том, что повторные выборки при классическом сценарии извлекаются из генеральной совокупности, а псевдоповторные выборки при выполнении ресэмплинга – из самой эмпирической выборки (хорошо известна фраза «*The population is to the sample as the sample is to the bootstrap samples*» из работы Fox, 2002).

Ресэмплинг объединяет четыре разных подхода к обработке данных, отличающихся по алгоритму, но близких по сути: перекрестная проверка (cross-validation, CV), бутстреп (bootstrap), рандомизация, или перестановочный тест (permutation), и метод "складного ножа" (jackknife). Ниже рассматриваются два из них: перекрестная проверка для оптимизации параметров модели и бутстреп для оценивания доверительных интервалов критериев эффективности моделей.

Минимизация ошибок на обучающем множестве, которое не бывает ни идеальным, ни бесконечно большим, неизбежно приводит к моделям, смещенным относительно истинной функции процесса, порождающего наблюдаемые данные. Преодолеть это смещение можно только с использованием "принципа внешнего дополнения", т.е. блоков "свежих" данных, не участвовавших в построении модели.

При отсутствии дополнительных блоков данных, специально предназначенных для тестирования, вполне естественно выглядит идея провести случайное разбиение исходной выборки на обучающее (train sample) и проверочное (test sample) подмножества, после чего оценить параметры модели только на обучающей выборке, тогда как оценку погрешности прогноза отклика $\hat{y}_i$ осуществить для значений из проверочной совокупности. Если подобные шаги выполняются многократно и каждое из

наблюдений используется поочередно то для обучения, то для контроля, то такая процедура эмпирического оценивания моделей, построенных по прецедентам, называется *перекрестной проверкой*, или "кросс-проверкой".

Стандартной считается методика $r \times k$-кратной кросс-проверки ($r \times k$-fold cross-validation), когда исходная выборка случайным образом r раз разбивается на k блоков (folds) равной (или почти равной) длины. При реализации каждой повторности r (replication) один из блоков по очереди становится проверочной совокупностью, а все остальные блоки – одной большой обучающей выборкой (рис. 2.5):

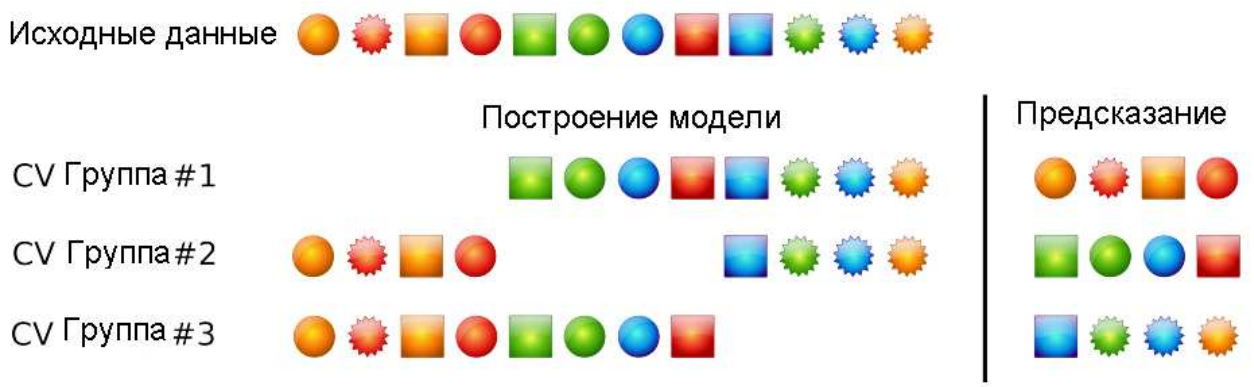


Рис. 2.5. Пример 1×3-кратной кросс-проверки

При этом генерируется $r \times n$ значений отклика $\hat{y}$, и *ошибкой перекрестной проверки* при использовании модели M на исходной выборке X^n называется средняя по всем k разбиениям величина ошибки на контрольных подмножествах: $$s_{CV} = \sqrt{\frac{1}{r \times n} \sum_{i=1}^{r \times n} (y_i - \hat{y}_i)^2} .$$

Выполнить разбиение исходной выборки y на k блоков можно с использованием различных функций R, например, из пакета `caret`:

```
createDataPartition(y, p = 0.5)
createFolds (y, k = 10)
createMultiFolds (y, k = 10, times = r)
```

Частным случаем является "скользящий контроль", или перекрестная проверка с последовательным исключением одного наблюдения (leave-one-out CV, LOOCV), т.е. $k = n$. При этом строится n моделей по $(n - 1)$ выборочным значениям, а исключенное наблюдение каждый раз используется для расчета ошибки прогноза. В. Н. Вапник (1984) теоретически обосновал применение скользящего контроля и показал, что если исходные выборки независимы, то средняя ошибка перекрестной проверки даёт *несмещённую* оценку ошибки модели. Это выгодно отличает её от средней ошибки на обучающей выборке, которая при переусложнении модели может оказаться оптимистично заниженной.

В разделе 2.1 мы рассматривали пример выбора оптимального числа параметров полиномиальной регрессии с использованием информационных критериев. Попробуем достичь той же цели, выполнив перекрестную проверку моделей в режиме "leave-one-out" с использованием самостоятельно оформленной функции:

```
# Функция скользящего контроля для модели y~poly(x, degree)
crossvalidate <- function(x, y, degree) {
   preds <- numeric(length(x))
   for(i in 1:length(x)) {
      x.in <- x[-i]; x.out <- x[i]
      y.in <- y[-i]; y.out <- x[i]
      m <- lm(y.in ~ poly(x.in, degree=degree) )
      new <- data.frame(x.in = seq(-3, 3, by=0.1))
      preds[i]<- predict(m, newdata=data.frame(x.in=x.out))
   }
  # Тестовая статистика - сумма квадратов отклонений:
  return(sum((y-preds)^2))
}
# Заполнение таблицы результатами кросс-проверки
# и сохранение квадрата ошибки в таблице "a"
a <- data.frame(cross=numeric(max.poly))
for(i in 1:max.poly)
{
  a[i,1] <- crossvalidate(x, y, degree=i)
}
# График суммы квадратов ошибки при кросс-проверке
qplot(1:max.poly,cross, data=a, geom=c("line"))+
xlab("Степень полинома ") + ylab("Квадратичная ошибка")
```

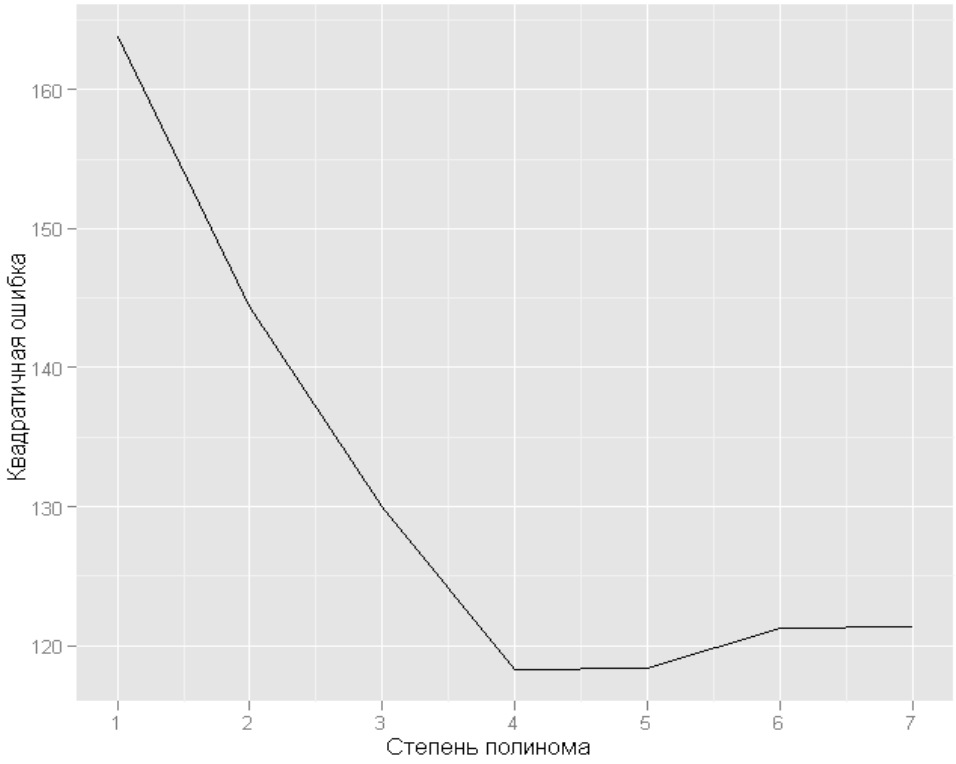


Рис. 2.6. Поиск степени функции полиномиальной регрессии с использованием перекрестной проверки

Как видно из графика на рис. 2.6, по мере усложнения модели от линейной регрессии с одним предиктором до полинома 7-й степени ошибка предсказаний на проверочном множестве (test error) сначала снижается, а затем после достижения некоторого минимума начинает возрастать. Такая *U*-образная форма кривой ошибки является универсальной и справедлива практически для любых алгоритмов и методов создания предсказательных моделей. Повторный рост ошибки на проверочных данных является сигналом того, что модель стала переобученной. Результаты, представленные на рис. 2.4 и 2.6, говорят о том, что в нашем конкретном случае оба метода одинаково выбирают в качестве наилучшей модели полином 4-й степени.

Проверим, является ли статистически значимым превышение качества полиномиальной модели над моделью с одним предиктором:

```
(M_poly <- glm(y ~ poly(x, 4)))

Coefficients:
(Intercept)  poly(x,4)1  poly(x,4)2  poly(x,4)3  poly(x,4)4
      4.360     -16.750       4.751       3.946      -3.371
Degrees of Freedom: 127 Total (i.e. Null);  123 Residual
Null Deviance:      439.3
Residual Deviance: 109.2        AIC: 354.9

anova(M_glm, M_poly, test ="Chisq")

Analysis of Deviance Table
Model 1: y ~ x
Model 2: y ~ poly(x, 4)
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1       126     158.71
2       123     109.21  3   49.501 4.743e-12 ***
```

Имеет место статистически значимое снижение ошибки модели.

В среде R перекрестную проверку модели можно выполнить не только на основе самостоятельно оформленных процедур, но и с использованием нескольких общедоступных функций из известных пакетов:

- `CVlm(df, form.lm, m=3)` из пакета `DAAG`, где `df` – исходная таблица с данными, `form.lm` – формула линейной модели, `m` – число блоков (сюда подставляется значение *k*);
- `cv.glm(df, glmfit, K = n)` из пакета `boot`, где `df` – исходная таблица с данными, `glmfit` – объект обобщенной линейной модели типа `glm`, `K` – число блоков (т.е. по умолчанию выполняется скользящий контроль);
- `cvLm(lmfit, folds = cvFolds(n, K, R), cost)` из пакета `cvTools`, где `lmfit` – объект обобщенной линейной модели `lm`, `K` – число блоков, `R` – число повторностей, `cost` – наименование одного из пяти разновидностей используемых критериев качества;
- `train(form, df, method, trainControl, …)` из пакета `caret`, где `df` – исходная таблица с данными, `form, method` – формула и метод построения модели, `trainControl` – объект, в котором прописываются все необходимые параметры перекрестной проверки.

Бутстреп-процедура состоит в том, чтобы случайным образом многократно извлекать повторные выборки из эмпирического распределения. Конкретно, если мы имеем исходную выборку из n членов $x_1, x_2, \ldots, x_{n-1}, x_n$, то с помощью датчика случайных чисел, равномерно распределенных на интервале [1, n], можем "вытянуть" произвольный элемент x_k, который снова вернем в исходную выборку для возможного повторного извлечения. Такая процедура повторяется n раз и образуется бутстреп-выборка, в которой одни элементы могут повторяться два или более раз, тогда как другие элементы – отсутствовать. Например, при $n = 6$ одна из таких бутстреп-комбинаций имеет вид $x_4, x_2, x_2, x_1, x_4, x_5$.

В R бутстреп легко реализуется с помощью функции `sample(..., replace=T)`, генерирующей любые случайные выборки с "возвращением". Вероятность того, что конкретное наблюдение не войдет в бутстреп-выборку размера n, равна $(1 - 1/n)n$ и стремится к $1/e = 0.368$ при $n \to \infty$:

```
Nboot = 1000; n=100
mean(replicate(Nboot, length(unique(sample(n, replace=TR)))))
```

```
[1] 63.544
```

Таким способом можно сформировать любое, сколь угодно большое число бутстреп-выборок (обычно 500-1000), каждая из которых содержит около 2/3 уникальных значений эмпирической совокупности.

В результате легкой модификации частотного распределения реализаций исходных данных можно ожидать, что каждая следующая генерируемая псевдовыборка будет обладать значением оцениваемого параметра, немного отличающимся от вычисленного для первоначальной совокупности. На основе разброса значений анализируемого показателя, полученного в ходе такого процесса имитации, можно построить, например, доверительные интервалы оцениваемого параметра.

В предыдущем разделе мы рассматривали использование информационных критериев для выбора оптимальной степени полинома. Осуществим оценку квантильных интервалов байесовского критерия `BIC` для каждой из модели-претендента:

```
set.seed(123)
Nboot = 1000
BootMat <- sapply(1:max.poly, function(k)
   replicate(Nboot, {
       ind <- sample(n, replace=T)
       BIC(glm(y[ind]~poly(x[ind],k)))
     }  )
)
apply(BootMat, 2, mean)
```

```
[1] 401.079 385.967 374.095 366.126 366.585 370.824 373.144
```

Искомые интервалы покажем на графике (рис. 2.7), подключив функцию `stat_summary()` из пакета `ggplot2`:

```
library(reshape) # для функции melt()
BootMat.df <- data.frame(melt(BootMat))
```

```
ggplot(data = BootMat.df, aes(X2, value)) +
   geom_jitter(position = position_jitter(width = 0.1),
               alpha = 0.2) +
# Добавляем средние значения в виде точек красного цвета:
  stat_summary(fun.y = mean, geom = "point", color = "red",
                size = 5)  +
# Добавляем отрезки, символизирующие 0.25 и 0.75 квантили:
  stat_summary(fun.y = mean,
     fun.ymin = function(x){quantile(x, p = 0.25)},
     fun.ymax = function(x){quantile(x, p = 0.75)},
     geom = "errorbar", color = "magenta", width = 0.5,
             size =1.5) +
     xlab("Степень полинома") +
     ylab("Информационный критерий Байеса")
```

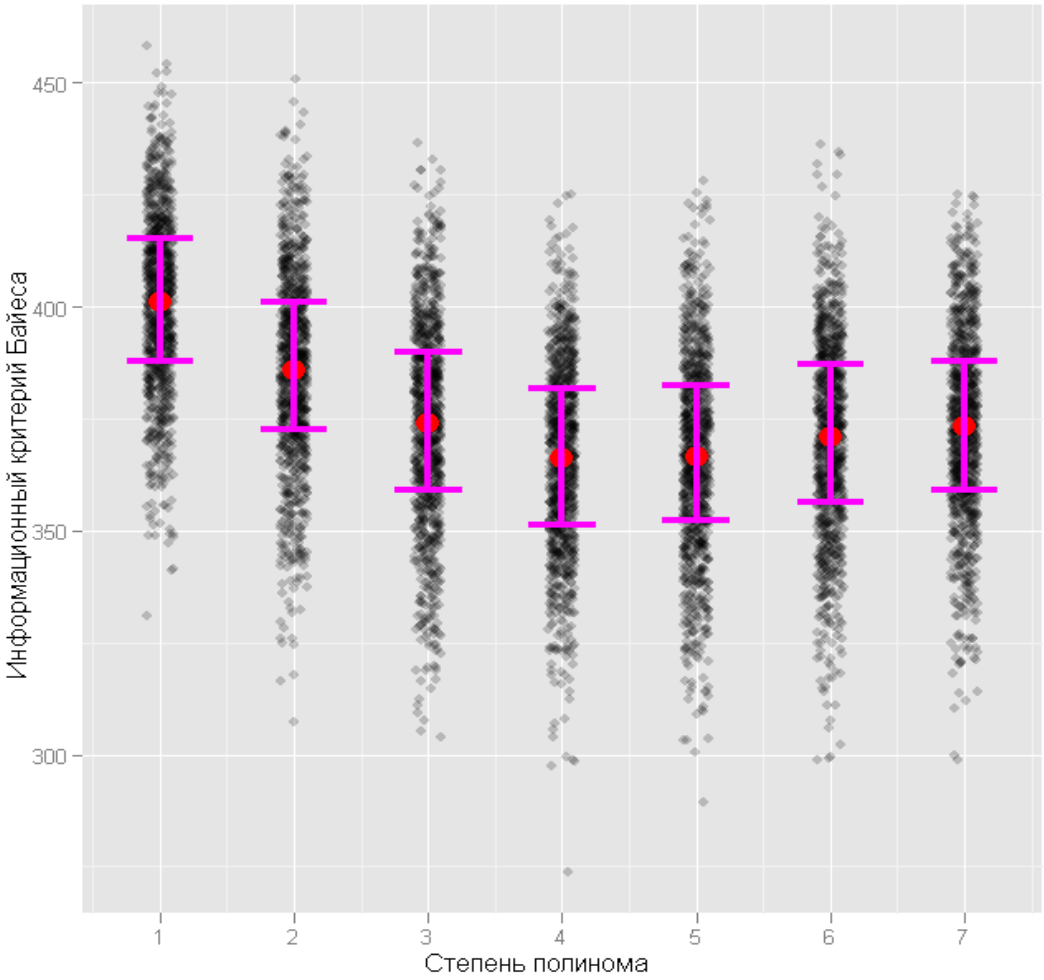


Рис. 2.7. Доверительные интервалы байесовского информационного критерия для разных степеней полинома

Для оценки 95%-х доверительных интервалов методом процентилей достаточно установить значения `p = c(0.025, 0.975)` (мы это не сделали исключительно из эстетических соображений). Из результатов бутстрепа можно сделать содержательные выводы, например: если доверительные интервалы смежных степеней полинома будут пересекаться, то уменьшение `BIC` статистически незначимо и вполне можно ограничиться более простой моделью. Подробнее о различных возможностях ресэмплинга рассказано в книге (Шитиков, Розенберг, 2014).

2.3. Модели для предсказания класса объектов

Классификация – наиболее часто встречающаяся задача машинного обучения, и заключается в построении моделей, выполняющих отнесение интересующего нас объекта к одному из нескольких известных классов. Существуют сотни методов классификации (см. Fernandez-Delgado et al., 2014), которые можно использовать для предсказания значения отклика с двумя и более классами. Возникает вопрос: отвечает ли такое множество потребностям реально решаемых задач?

Попробуем выделить основные характерные черты, отличающие эти методы. Во-первых, многое зависит от того, что является поставленной целью исследования: *объяснение* внутренних механизмов изучаемых процессов или только *прогнозирование* отклика. Если ставится задача "вскрытия" структуры взаимосвязей между независимыми переменными и откликом, то создаваемая модель должна в *явном* виде отображать их в виде наглядной схемы, либо осуществлять сравнительную оценку силы влияния отдельных переменных. Примерами хорошо интерпретируемых моделей классификации являются деревья решений, логистическая регрессия и модели дискриминации.

Если же основной задачей является достижение высокой общей точности предсказаний (overall accuracy) значения целевого признака y для объекта $\boldsymbol{a}$, то представление модели в явном виде не требуется. Изучаемый процесс, который часто имеет объективно сложный характер, представляется в виде "черного ящика", а решающие процедуры могут иметь большое (до десятков тысяч) или неопределенное число трудно интерпретируемых параметров. Эффективными методами прогнозирования классов являются случайные леса, бустинг, бэггинг, искусственные нейронные сети, машины опорных векторов, групповой учет аргументов МГУА и др.

Во-вторых, некоторую систематичность в типизацию моделей классификации может внести их связь с тремя основными парадигмами машинного обучения: геометрической, вероятностной и логической. Обычно множество объектов имеет некую *геометрическую* структуру: каждый из них, описанный числовыми признаками, можно рассматриваться как точка в многомерной системе координат. Геометрическая модель разделения на классы строится в пространстве признаков с применением таких геометрических понятий, как прямые, плоскости (в общем виде "гиперплоскости") и криволинейные поверхности. Примеры моделей, реализующих геометрическую парадигму: логистическая регрессия, метод опорных векторов и дискриминантный анализ. Другим важным геометрическим понятием является функция расстояния между объектами, которая приводит к классификатору по ближайшим соседям.

Вероятностный подход заключается в предположении о существовании некоего случайного процесса, который порождает значения целевых переменных, подчиняющиеся вполне определенному, но неизвестному нам распределению вероятностей. Примером модели вероятностного характера является байесовский классификатор,

формирующий решающее правило по принципу апостериорного максимума. Модели *логического* типа по своей природе наиболее алгоритмичны, поскольку легко выражаются на языке правил, понятных человеку, таких как: *if* <условие> = 1 *then Y* = <категория класса> . Примером таких моделей являются деревья классификации.

Некоторые авторы (Mount, Zumel, 2014, p. 91) подчеркивают различие терминов "предсказание" (prediction) и "прогнозирование" (forecasting). Предсказание лишь озвучивает результат (например, «Завтра будет дождь»), а при прогнозировании итог связывается с вероятностью события («Завтра с вероятностью 80% будет дождь»). Мы считаем, что на практике трудно провести между этими терминами четкую границу. К тому же, часто эта разница в совершенно не принципиальна – главное понимать контекст задачи.

Наконец, третьим основанием для группировки методов является природа наблюдаемых признаков, которые можно разделить на четыре типа: бинарные (0/1), категориальные, счетные и метрические. Имеются определенные нюансы при использовании перечисленных типов признаков в качестве предикторов, которые оговариваются нами ниже в рекомендациях по применению каждого метода моделирования. Например, бинарное пространство переменных некорректно использовать для линейного дискриминантного анализа. Однако принципиально важное значение имеет, к какому типу признаков относится отклик: задача классификации предполагает, что он измерен в бинарных, категориальных или, отчасти, порядковых шкалах.

Бинарный классификатор формирует некоторое диагностическое правило и оценивает, к какому из двух возможных классов следует отнести изучаемый объект (согласно медицинской терминологии условно назовем эти классы "норма" или "патология"). Группы точек "патология/норма" в заданном пространстве предикторов, как правило, статистически неразделимы: например, повышение температуры тела до 37.5^{o} часто свидетельствует о заболевании, хотя не всегда болезнь может сопровождаться высокой температурой. Поэтому при тестировании модели вероятны ошибочные ситуации, такие как пропуск положительного (патологического) заключения FN или его "гипердиагностика" FP, т.е. отнесение нормального состояния к патологическому.

Результаты теста на некоторой контрольной выборке можно представить обычной таблицей сопряженности, которую часто называют *матрицей неточностей* (confusion matrix):

Истинное состояние тест-объектов	Результаты теста	
	Предсказана патология (1)	Предсказана норма (0)
Патология (1)	Истинно положительные TP (True positives)	Ложноотрицательные FN (False negatives)
Норма (0)	Ложноположительные FP (False positives)	Истинно отрицательные TN (True negatives)

В этих обозначениях объективная ценность рассматриваемого бинарного классификатора определяется следующими показателями:

- *Чувствительность* (sensitivity) `SE` = Err_{II}= `TP / (TP + FN)`, определяющая насколько хорош тест для выявления патологических экземпляров;
- *Специфичность* (specificity) `SP` = Err_I = `FP / (FP + TN)`, показывающая эффективность теста для правильной диагностики отклонений от нормального состояния;
- *Точность* (accuracy) `AC = (TP + TN) / (TP + FP + FN + TN)`, определяющая общую вероятность теста давать правильные результаты.

По аналогии с классической проверкой статистических гипотез специфичность Err_I определяет ошибку I рода и, соответственно, вероятность нулевой гипотезы, тогда как чувствительность Err_{II} – мощность теста. Точность является, безусловно, наиболее широко известной мерой производительности классификатора, которая становится катастрофически некорректной в случае несбалансированных частот классов. Если, например, число пациентов, заболевших лихорадкой, составляет менее 1% от числа обследованных, то полный пропуск патологии даст вполне приличный результат тестирования 99%.

Рассмотрим популярный пример выделения спама (spam – от слияния двух слов spiced и ham, – или *пряная ветчина*, как образец некачественного пищевого продукта) в электронных письмах в зависимости от встречаемости тех или иных слов (всего 58 частотных показателей). Выборка по спаму представлена в обширной коллекции наборов данных Центра машинного обучения и интеллектуальных систем Калифорнийского университета (UCI Machine Learning Repository – http://archive.ics.uci.edu/ml/) и после некоторой предварительной обработки используется для иллюстрации в книге Mount, Zumel (2014). Скачаем этот файл с сайта ее авторов и разделим исходные данные в соотношении 10:1 на обучающую и проверочную выборки:

```
# spamD <- read.table(
#'https://raw.github.com/WinVector/zmPDSwR/master/Spambase/spamD.tsv',
#            header=T, sep='\t')
spamD <- read.table('spamD.tsv', header=T, sep='\t')
dim(spamD)
```

```
[1] 4601   59
```

```
spamTrain <- subset(spamD, spamD$rgroup>=10)
spamTest <- subset(spamD, spamD$rgroup<10)
c(nrow(spamTrain), nrow(spamTest))
```

```
[1]   4143  458
```

```
# Составляем список переменных и объект типа формула
spamVars <- setdiff(colnames(spamD), list('rgroup','spam'))
spamFormula <- as.formula(paste('spam=="spam"',
   paste(spamVars, collapse=' + '), sep=' ~ '))
spamModel <- glm(spamFormula, family=binomial(link='logit'),
   data=spamTrain)
```

```
# Добавляем столбец со значениями вероятности спама
spamTrain$pred <- predict(spamModel, newdata=spamTrain,
  type='response')
spamTest$pred  <- predict(spamModel,newdata=spamTest,
  type='response')
```

Компоненты матрицы неточностей и перечисленные показатели легко получить с использованием обычной функции `table(...)`– например, так:

```
#  На обучающей выборке
(см.train <- table(Факт = spamTrain$spam,
                   Прогноз = spamTrain$pred > 0.5))
```

```
          Прогноз
Факт       FALSE TRUE
  non-spam  2396  114
  spam       178 1455
```

```
#  На проверочной выборке:
(см <- table(Факт = spamTest$spam,
             Прогноз = spamTest$pred > 0.5))
```

```
          Прогноз
Факт       FALSE TRUE
  non-spam   264   14
  spam        22  158
```

```
c(Точность <- (см[1, 1] + см[2, 2])/sum(см),
  Чувствительность <- см[1, 1]/(см[1, 1] + см[2, 1]),
  Специфичность    <- см[2, 2]/(см[2, 2] + см[1, 2]))
```

```
[1] 0.9213974  0.9230769  0.9186047
```

Иногда предпочтительнее использовать функцию `confusionMatrix(y, pred)` из пакета `caret`:

```
library(caret)
pred <- ifelse(spamTest$pred > 0.5, "spam", "non-spam")
confusionMatrix(spamTest$spam, pred)
```

```
Confusion Matrix and Statistics
          Reference
Prediction non-spam spam
  non-spam      264   14
  spam           22  158
                Accuracy : 0.9214
                  95% CI : (0.8928, 0.9443)
    No Information Rate : 0.6245
    P-Value [Acc > NIR] : <2e-16
                   Kappa : 0.834
 Mcnemar's Test P-Value : 0.2433
            Sensitivity : 0.9231
            Specificity : 0.9186
         Pos Pred Value : 0.9496
         Neg Pred Value : 0.8778
             Prevalence : 0.6245
         Detection Rate : 0.5764
   Detection Prevalence : 0.6070
       'Positive' Class : non-spam
```

Выбрать другой класс в качестве положительного исхода можно, задав аргумент `positive = "spam"`. Функция предоставляет пользователю такие статистики, как доверительные интервалы и *p*-значение для точности,

результаты теста χ^2 по Мак-Немару, вероятностный индекс κ (каппа) Дж. Коэна, а также еще шесть других критериев оценки эффективности классификатора, интересных, по всей вероятности, ограниченному кругу специалистов:

- *прогностическая ценность* (prevalence)
  `PV = (TP + FN)/(TP + FP + FN + TN);`
- *положительная прогностическая ценность* (вероятность фактической патологии при положительном диагнозе)
  `PPV = SE * PV/(SE*PV + (1- SP)*(1 - PV));`
- *отрицательная прогностическая ценность* (вероятность отсутствия патологии при негативном результате теста)
  `NPV = SP*(1 - PV)/(PV*(1 - SE) + SP*(1 - PV));`
- *частота выявления* (detection rate)
  `DR = TP/( TP + FP + FN + TN);`
- *частота распространения* (detection prevalence)
  `DP = (TP + FP)/(TP + FP + FN + TN);`
- *сбалансированная точность* (balanced accuracy) `BAC = (SE + SP)^2.`

Эффективность классификатора может также оцениваться с использованием информационных критериев – *энтропии* $E = -\Sigma\, p_i \log_2 p_i$, где p_i – вероятности каждого возможного исхода, и условной энтропии (conditional entropy). Эти меры могут быть рассчитаны с использованием функций:

```
entropy <- function(x) { xpos <- x[x>0]
   scaled <- xpos/sum(xpos) ; sum(-scaled*log(scaled,2))
}
print(entropy(table(spamTest$spam)))
```

```
[1] 0.9667165
```

```
conditionalEntropy <- function(t) {
(sum(t[,1])*entropy(t[,1]) + sum(t[,2])*entropy(t[,2]))/sum(t)
}
print(conditionalEntropy(cM))
```

```
[1] 0.3971897
```

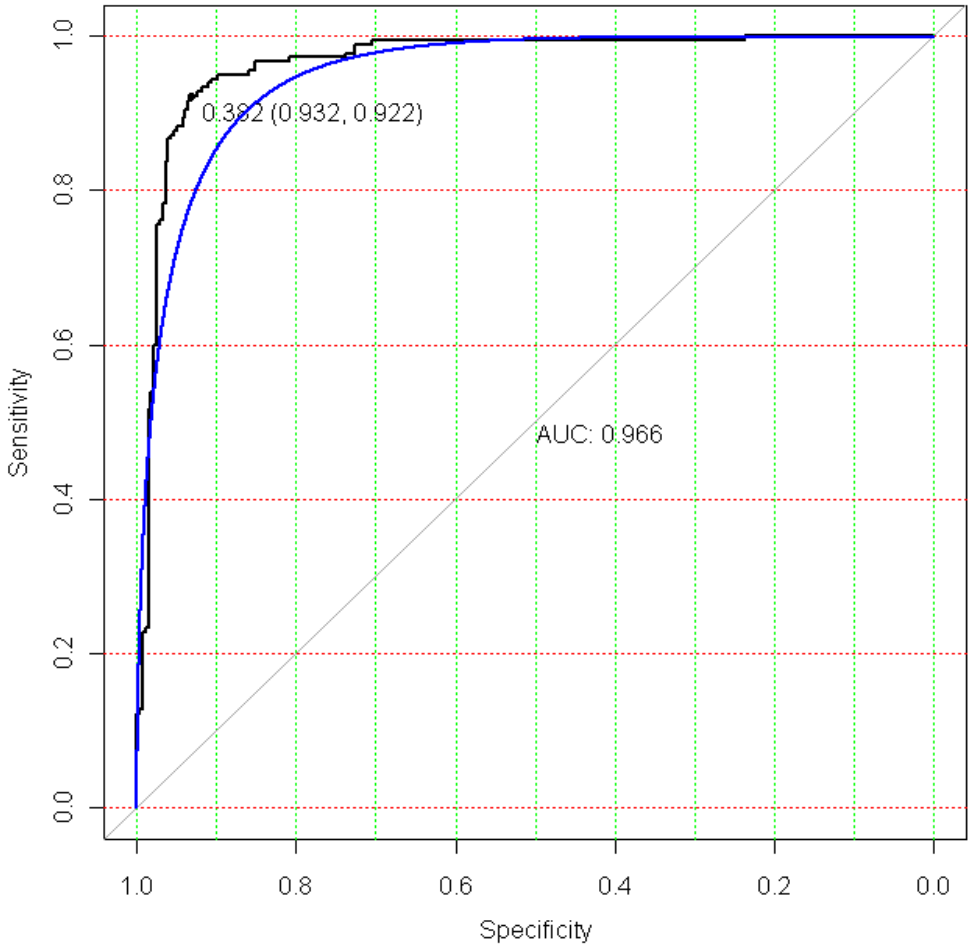
Исходная энтропия `E = entropy(table(y))` определяет среднее количество информации, измеряемой в битах, которую мы приобретаем, если извлечь из выборки очередной экземпляр того или иного класса. Условная энтропия `conditionalEntropy(table(y, pred))` показывает, насколько эта мера информации уменьшается из-за ошибок предсказания для различных категорий.

Общепринятым графоаналитическим методом оценки качества теста и интерпретации перечисленных показателей является *ROC-анализ* (от "Receiver Operator Characteristic" – функциональная характеристика приемника), название которого взято из методологии оценки качества сигнала при радиолокации.

ROC-кривая получается следующим образом (Goddard, Hinberg, 1989). Пусть мы имеем выборку значений независимого количественного

показателя, который варьирует от x_{min} до x_{max}, и сопряженного с ним бинарного отклика (1 – патология, 0 – норма). Любое произвольное значение x на этом диапазоне может считаться классификационным *порогом*, или *точкой отсечения* (cutt-off value), делящим вектор ***y*** на два подмножества, и для этого разбиения можно рассчитать значения чувствительности `SE` и специфичности `SP`. Если выполнить сканирование всех возможных значений $x_{max} \geq x \geq x_{min}$, то можно построить график зависимости, где по оси Y откладывается `SE`, а по оси X – `(1 - SP)`. Реализация этой процедуры в R может привести к ступенчатой или сглаженной кривой следующего вида (рис. 2.8):

```
# ROC кривая
library(pROC)
m_ROC.roc <- roc(spamTest$spam, spamTest$pred)
plot(m_ROC.roc, grid.col=c("green", "red"), grid=c(0.1, 0.2),
     print.auc=TRUE, print.thres=TRUE)
plot(smooth(m_ROC.roc), col="blue", add = T, print.auc=F)
```



.

Рис. 2.8. ROC-кривая для оценки вероятности спама

В случае идеального классификатора ROC-кривая проходит вблизи верхнего левого угла, где доля истинно-положительных случаев равна 1, а доля ложно-положительных примеров равна нулю. Поэтому, чем ближе кривая к верхнему левому углу, тем выше предсказательная способность

модели. Наоборот, главная диагональная линия соответствует "бесполезному" классификатору, который "угадывает" классовую принадлежность случайным образом. Следовательно, близость ROC-кривой к диагонали говорит о низкой эффективности построенной модели.

Для нахождения оптимального порога, соответствующего наиболее безошибочному классификатору, через крайнюю точку ROC-кривой проводят линию максимальной точности, параллельную главной диагонали. На приведенном графике такая точка, соответствующая значению x = 0.382, имеет наилучшую комбинацию значений чувствительности `SE` = 0.932 и специфичности `SP` = 0.922. Обратите внимание, что в качестве значений x фигурирует оценка вероятности отнесения к спаму и, видимо, мы совершенно напрасно принимали ранее в качестве порога величину 0.5.

Полезным показателем является численная оценка площади под ROC-кривыми AUC (Area Under Curve). Практически она изменяется от 0.5 ("бесполезный" классификатор) до 1.0 ("идеальная" модель). Показатель AUC предназначен исключительно для сравнительного анализа нескольких моделей, поэтому связывать его величину с прогностической силой можно только с большими допущениями.

В нашем примере в качестве классификатора писем со спамом мы использовали модель логистической регрессии, полагая, что бинарный отклик имеет биномиальное распределение. Напомним, что в случае обобщенных линейных моделей GLM вместо минимизации суммы квадратов отклонений ищется экстремум логарифма функции максимального правдоподобия (log maximum likelihood), вид которой зависит от характера распределения данных. В нашем случае логарифм функции правдоподобия `LL` численно равен сумме логарифмов вероятностей классов, которые модель правильно предсказывает для каждого наблюдения:

```
(LL <- logLik(spamModel))
'log Lik.' -807.0323 (df=58)
```

Как и в случае гауссова распределения (см. раздел 2.1), оценка адекватности биномиальной модели осуществляется с использованием девианса `D = -2(LL - S)`, где `S = 0` – правдоподобие "насыщенной модели" с минимальным уровнем байесовской ошибки. Исходя из априорной вероятности одного из классов, можно рассчитать логарифм правдоподобия и девианс для нулевой модели `D.null`. Эффективность классификатора определяется соотношением девианса остатков `D` и нуль-девианса `D.null`, что соответствует псевдо-коэффициенту детерминации `Rsquared` модели. Статистическую значимость разности девиансов (`D.null - D`) можно оценить по критерию χ^2:

```
df <- with(spamModel, df.null - df.residual)
c(D.null <- spamModel$null.deviance,
  D <- spamModel$deviance,
  Rsquared = 1-D/D.null,
  pchisq(D.null - D, df, lower.tail = FALSE))
```

```
[1] 5556.36  1614.065  0.7095104  0.000000
```

Мы получили модель, вполне адекватную по отношению к имеющимся данным. Разумеется, все эти вычисления могут быть выполнены с использованием базовых функций `summary()` и `anova()`:

```
summary(spamModel)
Null_Model <- glm(spam ~ 1,family=binomial(link='logit'),
   data=spamTrain)
anova(spamModel, Null_Model , test ="Chisq")
```

Мы не станем приводить здесь длинные протоколы с результатами этих процедур, включающие статистический анализ 58 коэффициентов модели. Отложим также для специального раздела обсуждение возможных путей решения проблемы поиска оптимального состава предикторов.

Вероятностные модели логита, как и иные детерминированные классификаторы, не только выполняют предсказание класса каждого тестируемого объекта, но и возвращают оцененную вероятность такой принадлежности. При этом весьма полезно проанализировать график плотности распределения вероятностей обоих классов, особенно при подборе оптимальных пороговых значений классификатора. Следующая команда R обеспечивает построение такого графика:

```
ggplot (data=spamTest) +
        geom_density (aes (x=pred, color=spam, linetype=spam))
```

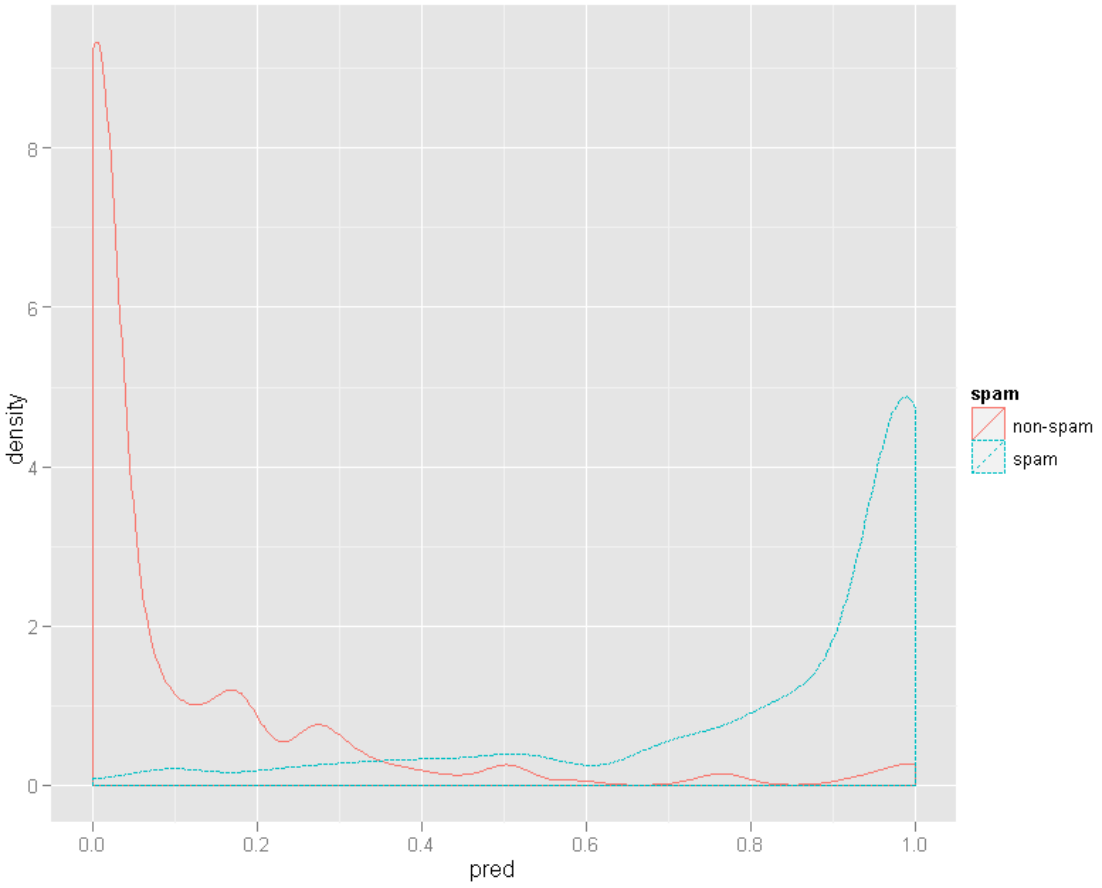


Рис. 2.9. Кривые плотности апостериорной вероятности принадлежности объектов к двум классам: электронным письмам с наличием спама и без него

2.4. Проецирование многомерных данных на плоскости

Многомерный анализ (Кендалл, Стьюарт, 1976) представляет собой часть статистики, которая выполняет обработку и интерпретацию результатов, полученных на основе наблюдений одновременно нескольких взаимосвязанных случайных переменных, каждая из которых представляется одинаково важной, по крайней мере, первоначально. В случае, если данные измерены в метрических шкалах и предполагаются линейные отношения между переменными, то в анализе применяются многомерные статистические методы, основанные на операциях матричной алгебры.

Во многих задачах обработки многомерных наблюдений исследователя интересует, в первую очередь, возможность снижения размерности, т.е. способ выделения небольшого набора исходных признаков или их линейных комбинаций, которые в наибольшей степени объясняют изменчивость наблюдаемых объектов. Это обусловлено следующими тремя причинами: а) возможностью наглядного представления (визуализация и ординация); б) стремлению к лаконизму исследуемых моделей и в) необходимостью сжатия объемов информации (Айвазян и др., 1989).

Принцип *ординации* наблюдений (нем. Ordnung) заключается в использовании различных методов оптимального целенаправленного проецирования (projecting pursuit) облака точек из многомерного пространства в пространство малой размерности (с 2 или 3 осями координат). Для этого используются различные методы или их модификации, но в целом сущность ординации заключается в представлении исходной матрицы данных **Y** в виде совокупности p латентных переменных **F**:

$$Y_1, Y_2, \ldots, Y_m \Rightarrow F_1, F_2, \ldots, F_p,$$

которые и являются осями ординационной диаграммы (двумерной при $p = 2$ или трехмерной в случае с тремя такими латентными переменными). Выбор осей ординации осуществляется с использованием принципа оптимальности: т.к. стремления достичь минимума потерь содержательной информации, имеющейся в исходных данных.

Как правило, первая ось F_1 проводится через центр сгущения значений y_{ij} и совпадает с направлением наибольшей по длине оси эллипсоида рассеяния. Вторая ось F_2, также проходит через центр распределения, но проводится перпендикулярно к первой и совпадает по направлению со второй из главных полуосей эллипсоида рассеяния. Эта операция обеспечивает формирование "картинки" данных с минимально возможными искажениями, а новые обобщающие переменные F_1 и F_2 становятся ортогональными (т.е. взаимно некоррелированными).

Большинство методов снижения размерности основано на анализе собственных значений и собственных векторов, который является важнейшим разделом линейной (матричной) алгебры. В кратком изложении суть преобразований $\mathbf{Y} \Rightarrow \mathbf{F}$ заключается в следующем:

1. Нахождение собственных значений и собственных векторов осуществляется в ходе математической операции линейного преобразования квадратной симметричной матрицы **C** размерностью *m*. Это может быть любая матрица дистанций, но чаще всего используется дисперсионно-ковариационная матрица $\mathbf{C} = \mathbf{Y}'\mathbf{Y}/(n - 1)$ стандартизованных исходных данных.

2. Собственными значениями квадратной матрицы **C** называются такие значения λ_k, при которых система *m* уравнений вида $(\mathbf{C} - \lambda_k \mathbf{I})\mathbf{u}_k = \mathbf{0}$ имеет нетривиальное решение. Здесь $\mathbf{u}_k$ – собственные векторы матрицы **C**, соответствующие λ_k, $k = 1, 2, \ldots, m$, **I** – единичная матрица.

3. Матрица из *k* собственных векторов $\mathbf{U}_k$ представляет собой веса для пересчета из исходного в редуцированное информационное пространство, т.е. матрицы переменных **Y** и **F** связаны соотношением $\mathbf{F}_{n\times k} = \mathbf{Y}_{n\times m}\,\mathbf{U}_{m\times k}$.

4. Каждое собственное значение λ_k соответствует величине дисперсии, объясняемой на *k*-м уровне, т.е. сумма всех собственных значений будет равняться сумме дисперсий всех исходных переменных.

5. Ряд собственных значений обычно ранжируется от самого большого до минимального: первое собственное значение λ_1, объясняющее наибольшую долю вариации данных, часто называют "доминирующим" или "ведущим". Значения λ_1 и λ_2 определяют меру значимости осей F_1 и F_2 ординационной диаграммы, т.е. концентрацию исходных точек вдоль каждой оси и, в итоге, ее выраженность.

6. Собственные значения λ_k находят в ходе итерационной процедуры, и точность их вычисления зависит от степени обусловленности исходной матрицы **C**.

Классическим методом снижения размерности данных является анализ главных компонент (PCA, Principal Components Analysis), который широко используется в различных областях науки и техники и детально описан в многочисленных руководствах (ter Braak, 1983; Айвазян и др., 1989; Джонгман и др., 1999; Legendre, Legendre, 2012).

Снижение размерности исходного пространства методом PCA можно представить как последовательный, итеративный процесс, который можно оборвать на любом шаге *u*. Вследствие ортогональности системы координат главных компонент (т.е. фактически, их взаимной независимости) нет необходимости перестраивать матрицы счетов (scores) **F** и нагрузок (loadings) **U** при изменении числа компонент: последующие столбцы или строки просто прибавляются или отбрасываются.

Используем в качестве примера идентификацию типа оконных стекол (Glass Identification) по выборке, представленный в коллекции баз данных UCI Machine Learning Repository. Существует два основных способа производства листового стекла: горизонтальный на расплаве олова (флеш-стекло) и по принципу вертикального вытягивания. Термически полированное флеш-стекло отличается идеально глянцевой поверхностью,

высокой светопропускающей способностью и великолепными оптическими свойствами, исключающими искажение изображения.

В ходе криминалистической экспертизы мельчайших осколков стекла можно определить их оптическую рефракцию (RI) и сделать химический анализ содержания окислов основных элементов. Ставится вопрос, можно ли по этим данным выполнить прогноз типа производства стекла Class. Исходные данные представлены в файле Glass.txt, и можно предварительно рассмотреть описательные статистики отдельных переменных:

```
DGlass <- read.table(file = "Glass.txt", sep = ",",
                     header = TRUE, row.names = 1)
print(t(apply(DGlass[, -10], 2, function (x) {
       c(Минимум = min(x), Максимум = max(x),
       Среднее = mean(x), Отклонение = sd(x),
       Корреляция = cor(x, DGlass$Class)) # признаков с Class
})), 3)
```

```
   Минимум Максимум Среднее Отклонение Корреляция
RI    1.51     1.53  1.5186    0.00305    -0.0601
Na   10.73    14.86 13.2017    0.58830     0.0186
Mg    0.00     4.49  3.2949    0.88778    -0.1462
Al    0.29     2.12  1.2817    0.32374     0.1998
Si   69.81    74.45 72.5869    0.64117    -0.0785
K     0.00     1.10  0.4775    0.21873     0.0386
Ca    7.08    16.19  8.9247    1.37263     0.0446
Ba    0.00     3.15  0.0298    0.25353     0.0312
Fe    0.00     0.37  0.0676    0.09951     0.0532
```

В среде R расчет главных компонент может быть осуществлен с использованием функций princomp() или prcomp(), но мы будем ориентироваться на многообразие возможностей пакета vegan() и его функцию rda(). Протокол анализа ограничим четырьмя главными компонентами, объясняющими 98% общей вариации данных:

```
library(vegan)
Y <- as.data.frame(DGlass[,1:9])
mod.pca <- rda(Y ~ 1)
summary(mod.pca)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4
Eigenvalue             2.6056 0.5828 0.21129 0.18895
Proportion Explained   0.7126 0.1594 0.05779 0.05168
Cumulative Proportion  0.7126 0.8720 0.92984 0.98152
Loads
          PC1       PC2       PC3        PC4
RI -0.006663  0.003040 -0.001587 -0.0007297
Na  0.461583  1.180084  0.833131  0.0499341
Mg  2.037591  0.659746 -0.616211 -0.5223737
Al  0.240546 -0.337483 -0.107770  0.6588300
Si  0.691702 -1.339222  0.458279 -0.4986449
K   0.265988 -0.302647 -0.111021  0.2512089
Ca -3.512043  0.228783 -0.158561 -0.3491594
Ba -0.190199 -0.009844 -0.267021  0.3372595
Fe -0.039086 -0.017137 -0.046526  0.0299297
```

Рассмотрим, какую конфигурацию имеет распределение наблюдений в пространстве первых двух главных компонент PC1 и PC2. Наибольший интерес для нас представляет взаимное расположение сгущений точек, принадлежащих стеклам двух типов: Class = {1, 3} – оконное и автомобильное флеш-стекло, и Class = 2 – тянутое стекло. Выполним построение ординационной диаграммы с использованием пакета ggplot2:

```
F <- as.factor(ifelse(DGlass$Class == 2, 2, 1))
pca.scores <- as.data.frame(summary(mod.pca)$sites[,1:2])
pca.scores <- cbind(pca.scores, F)
# Составляем таблицу для "каркаса" точек на графике
l <- lapply(unique(pca.scores$F), function(c)
        { f <- subset(pca.scores, F == c); f[chull(f),  ]})
hull <- do.call(rbind, l)
# Включаем в названия осей доли объясненной дисперсии
axX <- paste("PC1 (",
   as.integer(100*mod.pca$CA$eig[1]/sum(mod.pca$CA$eig)),"%)")
axY <- paste("PC2 (",
   as.integer(100*mod.pca$CA$eig[2]/sum(mod.pca$CA$eig)),"%)")
# Выводим ординационную диаграмму
ggplot() +
  geom_polygon(data=hull,aes(x=PC1,y=PC2, fill=F),
            alpha=0.4, linetype=0) +
  geom_point(data=pca.scores,aes(x=PC1,y=PC2,shape=F,
            colour=F),size=3) +
  scale_colour_manual( values = c('purple', 'blue'))+
  xlab(axX) + ylab(axY) + coord_equal() + theme_bw()
```

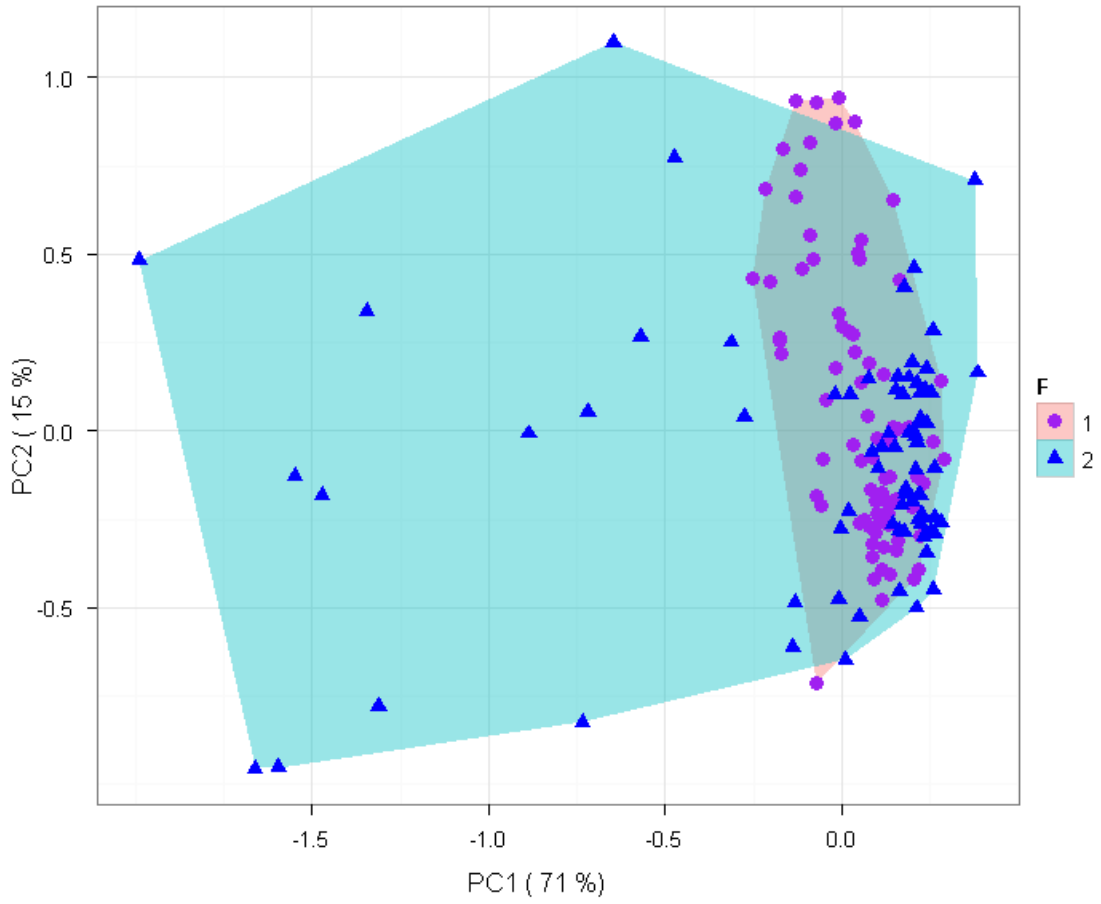


Рис. 2.10. Ординационная диаграмма, построенная методом PCA

Заметим, что для выделения наблюдений каждого класса на диаграммах обычно применяются четыре вспомогательных графических объекта: а) "каркас" (hull) или контур, проводимый через крайние точки, как мы это выполнили на рис. 2.10; б) эллипс, ограничивающий, например, 95% наблюдений; в) точка центроида или "центра тяжести" облака точек; г) "паук" (spider), т.е. набор линий, соединяющих каждую точку с центроидом своего класса.

По существу расчетов можно отметить следующее:

- первая главная компонента объясняет 71% совокупной дисперсии в данных, а F_1 и F_2 совместно – 87%, что является вполне обнадеживающим результатом;
- ось PC1 в значительной мере определяется содержанием магния и кальция, а ось PC2 – концентрациями натрия и кремния;
- диаграмма на рис. 2.10 показывает, что химический состав флеш-стекла в целом не отличается от тянутого, но для него не характерно повышенное содержание кальция (точки левее PC1 = -0.5).

Итак, ординация показала, что облака точек обоих классов являются плохо разделяемыми. Обратим внимание, что в расчетах не использовался такой ключевой показатель, как тип производства стекла, известный нам по примерам обучающей выборки.

2.5. Многомерный статистический анализ данных

Взаимосвязь между двумя комплексами переменных может быть установлена в ходе канонического анализа (от греческого κανων). В математике, канонической формой называется самая простая и универсальная форма, к который определенные функции, отношения, или выражения могут быть приведены без потери общности. Наиболее распространенными формами канонических моделей, основанных на приведении к собственным значениям и собственным векторам, являются анализ избыточности RDA (redundancy analysis) и линейный дискриминантный анализ LDA (linear discriminant analysis).

Анализ избыточности (Джонгман и др., 1999; Legendre, Legendre, 2012) является непосредственным расширением идей множественной регрессии на моделирование данных с многомерным откликом. Анализ асимметричен: если $\mathbf{Y}(n\times m)$ – таблица зависимых и $\mathbf{X}(n\times q)$ – таблица объясняющих переменных, то чаще всего $m <> q$. RDA выполняется в три этапа:

1. Оцениваются коэффициенты **B** модели множественной линейной регрессии **Y** на **X** и формируется матрица прогнозируемых значений $\hat{\mathbf{Y}}$:

$$\hat{\mathbf{Y}} = \mathbf{X}\,\mathbf{B} = \mathbf{X}[\mathbf{X}`\mathbf{X}]^{-1}\mathbf{X}`\mathbf{Y};$$

2. Вычисляется матрица канонических коэффициентов $\mathbf{S} = \mathbf{B}\mathbf{U}_c$ размерностью $m\times q$, которая рассчитывается на основе собственных векторов $\mathbf{U}_c$, взвешивающих влияние объясняющих переменных (constrained eigenvalues).

3. Рассчитывается ковариационная матрица $\mathbf{C}_{y|x}$ на основе значений $\hat{\mathbf{Y}}$, выполняется анализ главных компонент:

$$(\mathbf{C}_{y|x} - \lambda_k \mathbf{I})\mathbf{u}_k = \mathbf{0}, \text{ где } \mathbf{C}_{y|x} = \mathbf{C}_{YX}\mathbf{C}_{XX}^{-1}\mathbf{C}^{T}_{YX},$$

и обычным путем вычисляется матрица нагрузок на основе *k* собственных векторов, не связанных с объясняющими переменными **X** (unconstrained eigenvalues).

Для примера, описанного в разделе 2.4, матрица **X** сводится к вектору типа стекла `F`, и анализ избыточности приводит к следующим результатам:

```
F <- as.factor(ifelse(DGlass$Class == 2, 2, 1))
Y <- as.data.frame(DGlass[, 1:9])
mod.rda <- rda(Y ~ F)
summary(mod.rda)
```

```
Importance of components:
                          RDA1    PC1    PC2     PC3     PC4
Eigenvalue            0.11833 2.5441 0.5584 0.21014 0.15944
Proportion Explained  0.03236 0.6958 0.1527 0.05747 0.04361
Cumulative Proportion 0.03236 0.7282 0.8809 0.93837 0.98197
Species scores
          RDA1       PC1        PC2       PC3       PC4
RI  6.164e-05  0.006766 -0.0027906  0.001443  0.001003
Na -2.177e-01 -0.423708 -1.1896878 -0.802298 -0.127952
Mg -7.082e-01 -1.943017 -0.5819804  0.515740  0.600069
Al  3.060e-01 -0.296512  0.2471444  0.226655 -0.564066
Si  2.683e-02 -0.715098  1.3444594 -0.549095  0.330432
K   1.054e-01 -0.289194  0.2676415  0.153595 -0.215380
Ca  3.605e-01  3.501644 -0.1493831  0.091594  0.335991
Ba  4.946e-02  0.183784 -0.0005734  0.318738 -0.313081
Fe  2.934e-02  0.034801  0.0124987  0.051718 -0.014555
Centroids for factor constraints
        RDA1 PC1 PC2 PC3 PC4
cen1 -0.3612   0   0   0   0   0
cen2  0.4134   0   0   0   0   0
```

После включения в анализ объясняющего фактора «Тип производства стекла» появляется связанная с ним ось новой переменной RDA1, а к матрице нагрузок на главные компоненты добавляется столбец соответствующих канонических коэффициентов. Собственное значение, определяющее RDA1, невелико и составляет только 3.2% от общей вариации данных.

Сформируем ординационную диаграмму в координатах {`RDA1, PC1`}, на которой обозначим координаты центроидов и выделим 95% доверительные области в виде эллисов (рис. 2.11):

```
library(ggplot2)
rda.scores <- as.data.frame(scores(mod.rda, display="sites",
                                   scales = 3))
rda.scores <- cbind(F,rda.scores)
centroids <- aggregate(cbind(RDA1, PC1)~F,rda.scores, mean)
# функция для std.err
f <- function(z)sd(z)/sqrt(length(z))
se <- aggregate(cbind(RDA1,PC1) ~ F, rda.scores, f)
names(se) <- c("F", "RDA1.se", "PC1.se")
# объединяем координаты центроидов и стандартные ошибки
centroids <- merge(centroids, se, by = "F")
```

```
#  Формируем диаграмму
ggplot() +  geom_point(data = rda.scores,
    aes(x=RDA1, y=PC1, shape=F, colour=F), size=2) +
  xlim(c(-2, 5)) + ylim(c(-0.75, 1)) +
  scale_colour_manual(values = c('purple', 'blue')) +
  stat_ellipse(data=rda.scores, aes(x=RDA1, y=PC1, fill=F),
                    geom="polygon", level=0.8, alpha=0.2) +
  geom_point(data=centroids, aes(x=RDA1, y=PC1, colour=F),
             shape=1, size=4) + theme_bw()
```

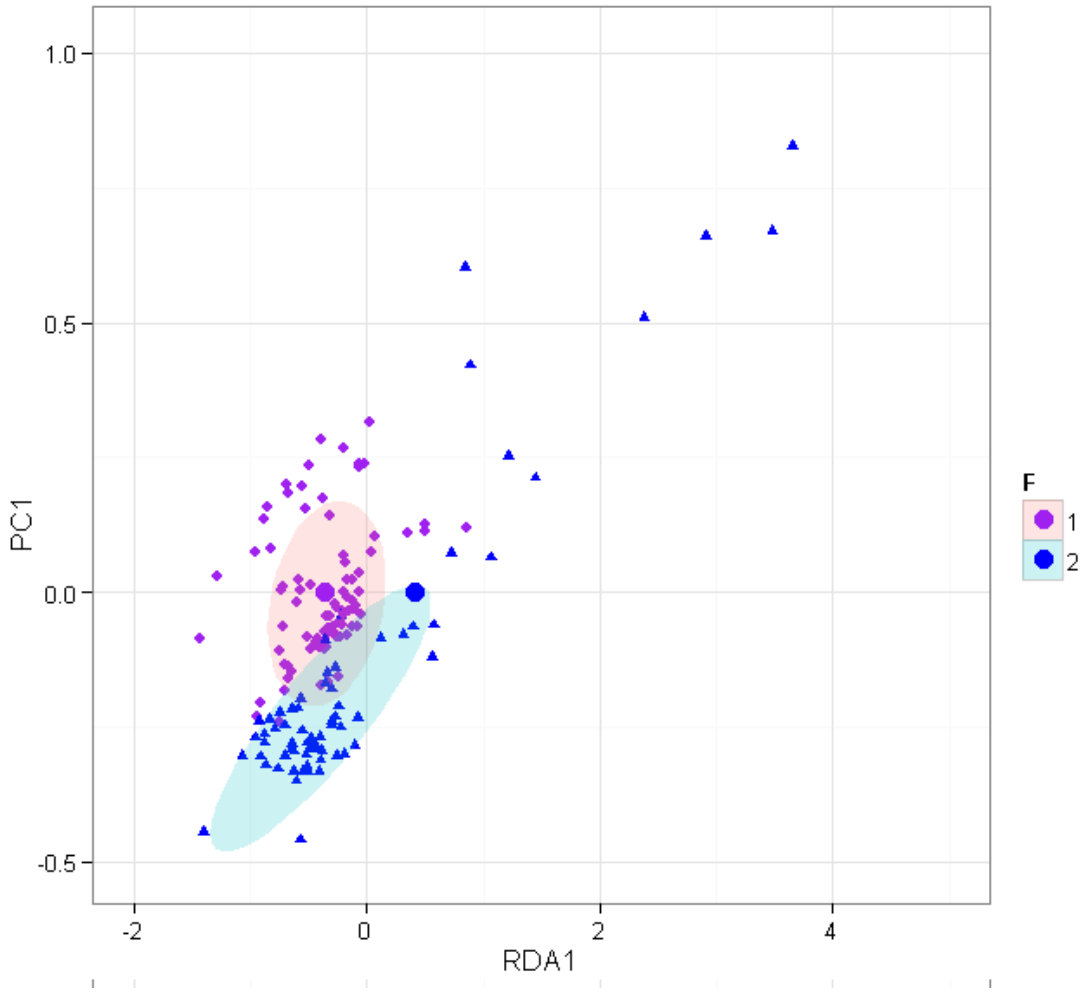


Рис. 2.11. Ординационная диаграмма, построенная методом RDA (не показаны 7 точек за пределами верхнего правого угла графика)

Несмотря на небольшую долю дисперсии, объясняемой фактором способа производства стекла F, координаты центроидов, определяющих центры тяжести сравниваемых групп, стали располагаться на некотором расстоянии друг от друга на рис. 2.11, что позволяет мысленно провести разделяющую линию между ними.

В арсенал многомерных процедур входят не только задачи снижения размерности или распознавания групп, но и методы, предназначенные для традиционной проверки статистических гипотез. Многомерный дисперсионный анализ является обобщением обычного одномерного дисперсионного анализа и предназначен для выявления различий между

группами по совокупности средних значений комплекса признаков. Нулевая гипотеза заключается в предположении о равенстве векторов средних значений для сравниваемых групп наблюдений.

Для формирования модели многомерного дисперсионного анализа можно воспользоваться функциями `aov()`, `lm()` либо `manova()`, которые приводят к идентичным результатам. Получить традиционную дисперсионную таблицу можно с использованием функции `anova()`:

```
mod.an <- manova(as.matrix(Y) ~ F)
anova(mod.an)
```

```
Analysis of Variance Table
             Df  Pillai approx F num Df den Df    Pr(>F)
(Intercept)   1 1.00000 65907164      9    153 < 2.2e-16 ***
F             1 0.27971        7      9    153 6.106e-08 ***
Residuals   161
```

Теснота связи между химическим составом стекла `Y` и способом его изготовления `F` была оценена с использованием статистики Пиллая (Pillai's trace).

Поскольку мы имеем лишь две группы наблюдений, то статистическая проверка гипотезы о равенстве средних двух векторов $H_o: \overline{\mathbf{X}}_1 = \overline{\mathbf{X}}_2$ может быть осуществлена также с использованием статистики Хотеллинга, которая является многомерным аналогом *t*-критерия Стьюдента:

```
library(Hotelling)
split.data <- split(Y, F)
(summ.hot <- hotelling.test(split.data[[1]], split.data[[2]],
                  perm = T, B = 1000))
```

```
Test stat:  6.6015
Permuation P-value:  0
Number of permutations : 1000
```

В выполненном перестановочном тесте ни одна из 1000 статистик Хотеллинга, полученных при случайном перемешивании данных, не превысила тестируемое значение, что свидетельствует о высокой статистической значимости разделяющего фактора.

2.6. Методы кластеризации

В разделах 2.4-2.5 мы рассмотрели две процедуры: *необъясненная* (непрямая) ординация наблюдений, при которой конфигурация точек определяется случайной вариацией данных и не связывается с какими-либо внешними причинами, и *прямая* ординация с использованием объясняющих переменных (в рассмотренном примере – способ изготовления стекла `F`). Ранжирование, непрямая ординация и кластеризация представляют собой совокупность методов обучения без учителя, основанных, как правило, на анализе расстояний между всеми возможными парами объектов в пространстве наблюдаемых независимых признаков. Такой подход, основанный на минимально возможном искажении исходной взаимной

упорядоченности точек, обеспечивает наглядное графическое представление геометрической метафоры исследуемых объектов.

Под *кластеризацией* (от англ. cluster – гроздь, скопление) понимается задача разбиения всей исходной совокупности элементов на отдельные группы однородных объектов, сходных между собой, но имеющих отчетливые отличия этих групп друг от друга. Пусть $d(x_i, x_j)$ – некоторая мера близости между каждой парой классифицируемых объектов i и j. В качестве таковой может использоваться любая полезная функция: евклидово или манхэттенское расстояние, коэффициент корреляции Пирсона, расстояние χ^2, коэффициенты сходства Жаккара, Съеренсена, Ренконена и многие другие.

Наиболее часто применяется агломеративный иерархический алгоритм "дендрограмма", отдельные версии которого отличаются правилами вычисления расстояния между кластерами. Дендрограммы просты в интерпретации и предоставляют полную информацию о соподчиненности всех классифицируемых объектов. Иногда они просто встраиваются в другие, более общие графики (например, "тепловые карты"), расширяя их понимание и возможность интерпретации.

Рассмотрим в качестве примера набор данных `USArrests` о криминогенной обстановке по штатам США (из пакета `cluster`). С помощью тепловой карты, представленной на рис. 2.12, можно: а) разбить 50 штатов США на группы по криминальной напряженности, б) установить характер взаимосвязи между числом арестованных за убийство (`Murder`), изнасилование (`Rape`), разбой (`Assault`), а также долей городских жителей (`UrbanPop`) и, наконец, в) убедиться в том, что число изнасилований в Аляске много меньше, чем в Род-Айленде.

```
library(cluster)
data("USArrests")
x = as.matrix(USArrests)
rc <- rainbow(nrow(x), start = 0, end = .3)
cc <- rainbow(ncol(x), start = 0, end = .3)
hv <- heatmap(x,  scale = "col",  RowSideColors = rc,
           ColSideColors = cc, margins = c(10,10),
           cexCol = 1.5, cexRow = 1)
```

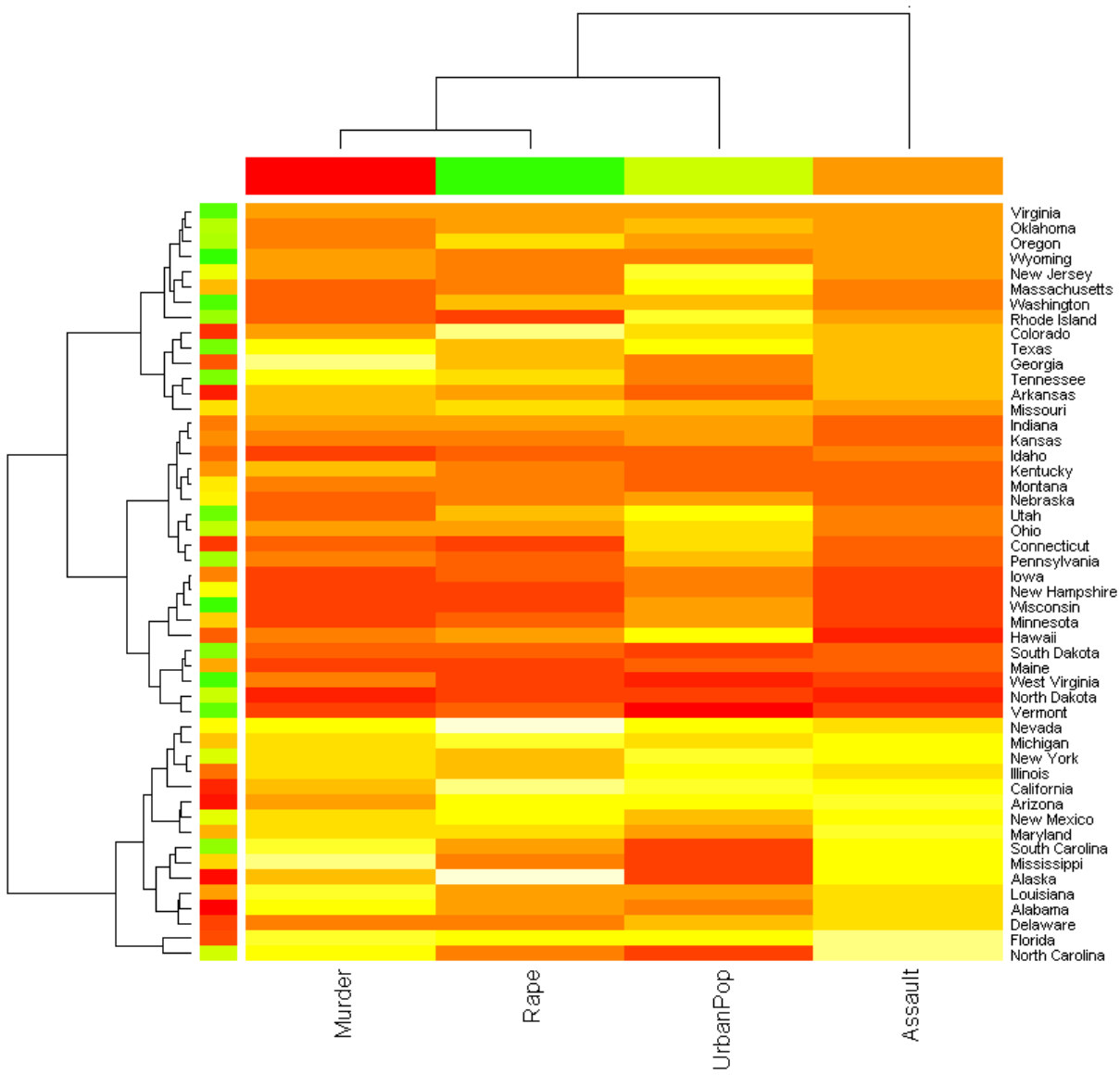


Рис. 2.12. Пример тепловой карты с дендрограммами

Однако существенным недостатком иерархических методов является их детерминированность: объединение классов на более высоком уровне полностью определяется результатами агломерации на нижних уровнях. Для оценки того, какой из вариантов кластеризации в наибольшей степени отражает близость объектов в исходном признаковом пространстве, могут быть использованы кофенетическая корреляция (cophenetic correlation) или различные ранговые индексы.

Кроме иерархических методов классификации большое распространение также получили различные итерационные процедуры, которые пытаются найти наилучшее разбиение, ориентируясь на заданный критерий оптимизации, не строя при этом полного дерева. Наиболее популярный неиерархический алгоритм – метод *k* средних Мак-Кина, в котором сам пользователь должен задать искомое число конечных кластеров, обозначаемое как "*k*". Его главным преимуществом является возможность

обрабатывать очень большие массивы данных, поскольку нет необходимости хранить в памяти компьютера всю матрицу расстояний целиком.

Выполним разбиение 50 штатов на 5 классов по степени их криминализации:

```
df.stand <- as.data.frame(scale(USArrests))
(clus <- kmeans(df.stand, centers = 5))
```

```
K-means clustering with 5 clusters of sizes 13, 11, 12, 7, 7
Cluster means:
      Murder    Assault    UrbanPop        Rape
1 -0.2162425 -0.2611064 -0.04793489 -0.06172647
2 -1.1034717 -1.1654231 -0.99194587 -1.04874074
3  0.7298036  1.1188219  0.75717991  1.32135653
4  1.5803956  0.9662584 -0.77751086  0.04844071
5 -0.6958674 -0.5679476  1.12728218 -0.55096728
Within cluster sum of squares by cluster:
[1] 10.860162  8.499862 18.257332  6.128432  5.244931
 (between_SS / total_SS =  75.0 %)
```

Мы видим, что к каждому из пяти кластеров относится от 7 до 13 штатов, причем наибольший криминальный риск имеет место в 3 и 4 группах.

Любой алгоритм кластеризации может считаться результативным, если выполняется *гипотеза компактности* (Загоруйко, 1999), т.е. можно найти такое разбиение объектов на группы, что расстояния между объектами из одной группы (intra-cluster distances) будут меньше некоторого значения $\varepsilon > 0$, а между объектами из разных групп (cross-cluster distance) – больше ε. Можно сформировать таблицу всех этих расстояний как показано ниже:

```
library('reshape2')
n <- dim(df.stand)[[1]]
euc.dist <- as.matrix(dist(df.stand))
dist = melt(euc.dist)
df.stand$cluster <- clus$cluster
pairs <- data.frame(dist = dist,
  ca = as.vector(outer(1:n, 1:n, function(a, b)
            df.stand[a, 'cluster'])),
  cb = as.vector(outer(1:n, 1:n, function(a, b)
            df.stand[b, 'cluster'])))
dcast(pairs, ca ~ cb, value.var='dist.value', mean)
```

```
  ca        1        2        3        4        5
1  1 1.173101 2.145333 2.691664 2.572693 1.794814
2  2 2.145333 1.106637 4.360496 3.787671 2.488403
3  3 2.691664 4.360496 1.566414 2.564101 3.195211
4  4 2.572693 3.787671 2.564101 1.159667 3.581986
5  5 1.794814 2.488403 3.195211 3.581986 1.100954
```

В полученной таблице по главной диагонали приведены средние внутрикластерные расстояния, которые очевидно меньше, чем межкластерные расстояния (недиагональные элементы таблицы).

Поскольку объекты таблицы `USArrests` многомерны, то для вывода ординационной диаграммы выполним свертку информации по 4 имеющимся показателям к двум главным компонентам. После этого можно осуществить визуализацию групп (см. рис. 2.13) и "на глаз" оценить качество кластеризации.

```
c.pca <- prcomp(USArrests, center = TRUE, scale = TRUE)
d <- data.frame(x=c.pca$x[, 1], y=c.pca$x[, 2])
d$cluster <- clus$cluster
library('ggplot2'); library('grDevices')
h <- do.call(rbind, lapply(unique(clus$cluster),
function(c) { f <- subset(d,cluster==c); f[chull(f),]}))
ggplot() + geom_text(data=d, aes(label=cluster, x=x, y=y,
                                 color=cluster), size=3) +
   geom_polygon(data=h, aes(x=x, y=y, group=cluster,
   fill=as.factor(cluster)), alpha=0.4, linetype=0) +
   theme(legend.position = "none")
```

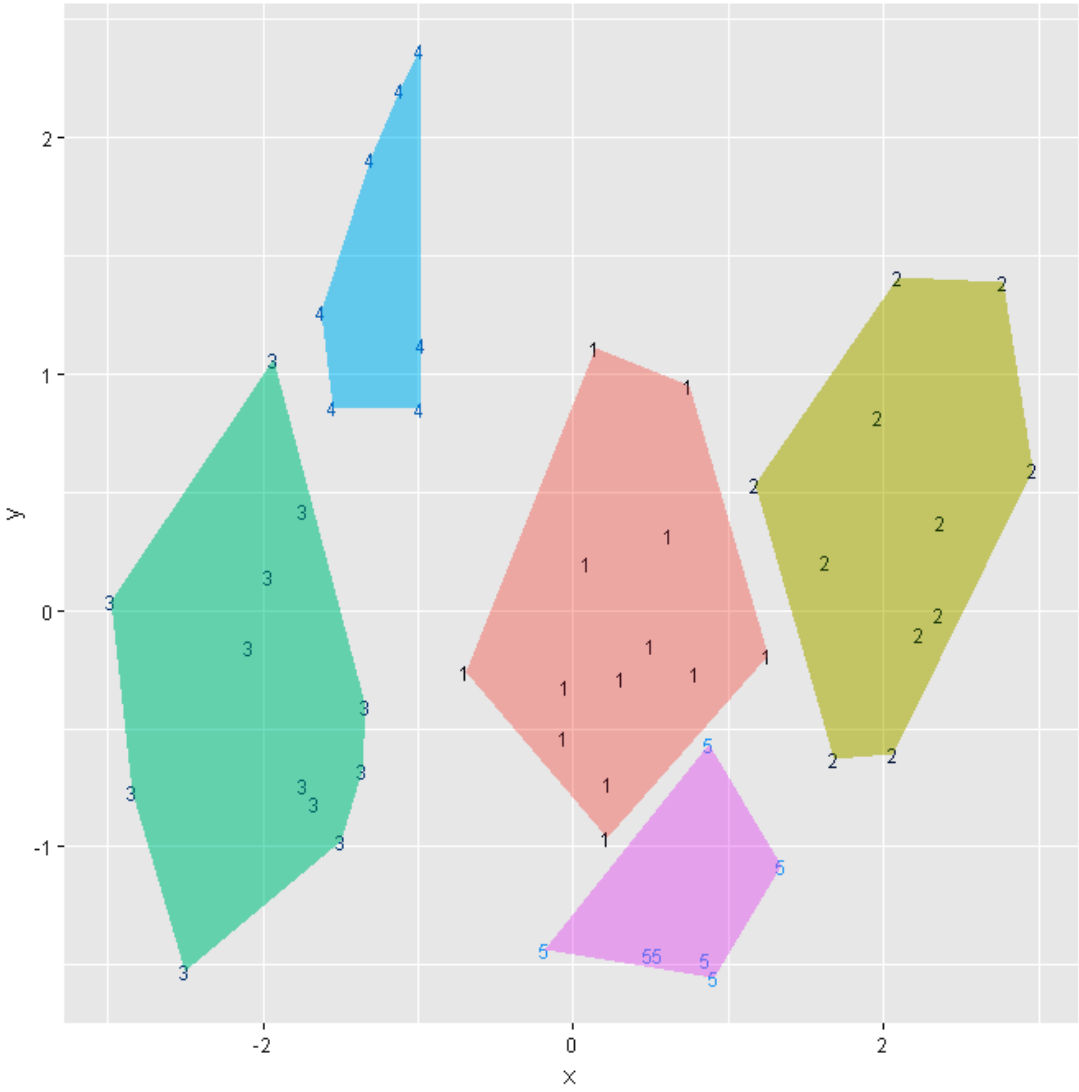


Рис. 2.13. Пример кластеризации методом *k* средних

Другим современным подходом к кластеризации объектов являются алгоритмы типа нечетких *C*-средних и Гюстафсона–Кесселя (Барсегян и др., 2004), которые ищут кластеры в пространстве нечетких множеств в форме эллипсоидов, что делает эти методы более гибкими при решении различных практических задач. В литературе также описывается множество других методов кластеризации, не использующих матрицы сходств и основанных на оценивании функций плотности статистического распределения, эвристических алгоритмах перебора, идеях математического программирования. Каждый из них имеет специфическую идеологическую основу и подходы к построению критериев качества моделей, которые мы рассмотрим далее в главе 10.

3. ПАКЕТ CARET - ИНСТРУМЕНТ ПОСТРОЕНИЯ СТАТИСТИЧЕСКИХ МОДЕЛЕЙ В R

3.1. Универсальный интерфейс доступа к функциям машинного обучения в пакете caret

Использование сложных алгоритмов построения статистических моделей становится все более распространенной тенденцией в самых разных областях – от академических исследований до всевозможных бизнес-приложений. Среда статистических вычислений R отличается особенно богатым набором пакетов для создания различных моделей классификации и регрессии. Однако такое высокое разнообразие реализованных алгоритмов создает и ряд некоторых проблем.

Для аналитика становится затруднительным выбрать конкретный алгоритм машинного обучения, составить план оптимизации параметров, учесть синтаксические нюансы каждой функции и отследить особенности ее выполнения. Кроме того, процедуры, позволяющие реализовать полный цикл разработки предсказательных моделей, часто "разбросаны" по разным пакетам, что требует времени для поиска необходимых компонент и их освоения. Возникла потребность как-то интегрировать разные функции и методы в рамках некоторой единой надстройки, что позволило бы обобщить характерные для всех моделей процедуры вычислений.

Выполняя эту задачу, д-р М. Кун с сотрудниками предприняли попытку разработать универсальный интерфейс, предоставляющий доступ к основным алгоритмам машинного обучения, реализованным в R и других специализированных статистических системах (например, Weka). Разработчики поставили перед собой следующие задачи:

- учет имеющихся синтаксических различий между используемыми функциями R при построении моделей классификации и регрессии и тестировании их возможностей для прогнозирования;
- развитие ряда полуавтоматических, интеллектуальных подходов и критериев для оптимизации настраиваемых коэффициентов и параметров многих из этих моделей с использованием алгоритмов ресэмплинга;
- создание пакета с постоянно расширяющимся набором методов;
- реализация параллельных вычислений при подгонке моделей.

Результатом этой работы стал пакет `caret` (сокращение от **C**lassication **a**nd **Re**gression **T**raining), который сегодня стал одним из наиболее популярных инструментов среди пользователей R, занимающихся разработкой предсказательных моделей. Основные возможности пакета `caret` достаточно полно представлены в публикациях разработчиков (Kuhn, 2008, 2012, 2013; Kuhn, Johnson, 2013).

На момент написания этой книги пакет `caret` включал унифицированный интерфейс со 147 функциями из 27 пакетов, которые автоматически подгружаются по мере необходимости (при условии, что они

установлены в системе). Наиболее распространенная технология работы с функциями пакета связана с автоматизированным нахождением оптимальных значений гиперпараметров моделей (tuning parameters), которые обычно невозможно вычислить аналитически, и построена по следующей формальной схеме:

```
1.  Определение наборов данных и их предобработка (при необходимости)
2.  Определение спецификации параметров модели
3.  Цикл для каждого параметра модели:
4.    | Цикл для каждой итерации ресэмплинга:
5.    |   |  Выделение тестовой выборки
6.    |   |  Подгонка модели по объектам обучающей выборки
7.    |   |  Прогнозирование отклика для тестовых объектов
8.    | end
9.    | Вычисление показателей средней эффективности прогноза
10. end
11. Установление оптимальных параметров модели
12. Построение итоговой модели по всей выборке с использованием оптимальных
    параметров
```

Кроме возможностей для настройки параметров, пакет `caret` содержит набор различных функций, способствующих реализации *полного цикла* разработки предсказательных моделей, начиная от сервисных процедур подготовки исходных данных до детальной оценки важности переменных, включенных в итоговую модель. Перечислим основные этапы построения моделей и функции пакета, реализующие их выполнение:

1. *Разбиение* исходных данных *на обучающую и контрольную выборки*. Формирование индексов любой такой последовательности для вектора `x` случайным образом в заданном соотношении `p` осуществляется функцией `createDataPartition(x, p=3/4, list = FALSE)`, что, в целом, эквивалентно использованию функции `sample(length(x), size = 0.75*length(x))`. С помощью аналогичных функций `createResample()`, `createFolds()` и `createMultiFolds()` можно создать бутстреп-выборки или произвольные разбиения на части в любом количестве повторностей.

2. *Разведочный анализ* исходных переменных. Графический анализ характера распределения данных и их взаимной зависимости, наличия выбросов и других аномальных ситуаций может быть выполнен с помощью функции `featurePlot()`. Функция `nearZeroVar()` позволяет исключить предикторы, дисперсия значений которых близка к нулю, а `findCorrelation()` позволяет выявить переменные, которые в значительной мере коррелируют с другими признаками.

3. *Предварительная обработка* исходной выборки ("выравнивание" дисперсий, приведение к нормальному распределению, сглаживание выбросов, заполнение пропущенных значений и проч., что часто является эффективной мерой улучшения структуры данных). Функция `preProcess()` и метод `predict.preProcess()` могут выполнить большое количество

различных процедур трансформации данных и других операций предобработки.

4. *Обучение моделей*, нахождение оптимальных гиперпараметров и оценка точности предсказания. Последние две задачи реализуются с использованием разнообразных методов ресэмплинга. Все эти процедуры выполняются функцией `train()`, а `predict.train()` позволяет получать предсказания значений переменной-отклика на основе выбранной оптимальной модели.

5. *Оценка "важности"* предикторов (variable importance) и селекция оптимального их набора. Функция `varImp()` рассчитывает для каждой переменной количественные показатели, отражающих их вклад в получение точных предсказаний в рамках построенной модели. Отбор информативного комплекса переменных может быть осуществлен также с помощью функции `rfe()`, выполняющей рекурсивное исключение переменных, или функции `gafs ()`, в которой реализован генетический алгоритм.

В последующих разделах будут рассмотрены подробности использования как перечисленных, так и других функций, входящих в пакет `caret`.

3.2. Обнаружение и удаление "ненужных" предикторов

Разведочный анализ исходных данных играет очень важную роль в процессе создании эффективных предсказательных моделей. Главная его цель – понимание свойств имеющихся в наличии переменных, таких как закономерность распределения, наличие выбросов или эффекта очень низкой дисперсии, характер взаимоотношений между откликом и предикторами, оценка уровня мультиколлинеарности и др. Поскольку многие алгоритмы могут быть чувствительными к наличию предикторов, которые не несут в себе никакой или почти никакой информации, то уже на предварительном этапе некоторые из них разумно идентифицировать как "ненужные" и в дальнейшем исключить из рассмотрения.

В качестве примера используем набор данных `GermanCredit`, входящий в состав пакета `caret`. Он содержит информацию по 1000 клиентам одного из немецких банков, (подробное описание см. на сайте UCI Machine Learning Repository). Каждый клиент описан в пространстве 61 признака, которые могут использоваться для предсказания класса кредитоспособности: отклика `Class`, принимающего два значения `Good` (хороший) и `Bad` (плохой).

Предположим, что мы имеем дело с экстремальным случаем, когда некоторый предиктор представлен только одним уникальным значением (например, у всех клиентов банка значения этой переменной равны 1). При таком сценарии дисперсия предиктора равна нулю и он бесполезен для предсказания интересующего нас отклика. В других случаях дисперсия может быть отличной от 0, но все же недостаточно высокой для того, чтобы сделать соответствующую переменную полезной для предсказания отклика. Подобные предикторы с околонулевой дисперсией (near-zero variance) рекомендуется удалять из дальнейшего анализа (Kuhn, Johnson, 2013).

Но как обнаружить такие предикторы? Одним из свойств вариационного ряда является доля уникальных значений по сравнению с общим числом наблюдений:

```
library(caret)
data(GermanCredit)
(u <-unique(GermanCredit$ResidenceDuration))
# Доля уникальных значений:
length(u)/nrow(GermanCredit)
```

```
[1] 4 2 3 1
[1] 0.004
```

Как видим, 1000 клиентов представлены только четырьмя уникальными значениями, доля которых составляет лишь 0.4% от их общего числа.

Однако сама по себе эта доля ни о чем не говорит, поскольку подавляющее большинство признаков в таблице `GermanCredit` являются индикаторными переменными, обозначающими наличие или отсутствие того или иного свойства у клиента, т.е. представлены только значениям 1 и 0. Важной является не только низкая доля уникальных значений, но еще и относительная частота этих значений. Поэтому рекомендуется (Kuhn, Johnson, 2013) рассчитывать отношение частоты наиболее часто встречающегося значения к частоте второго по встречаемости значения. Высокое отношение будет указывать на явный дисбаланс в частотах уникальных значений и, как следствие, на низкую дисперсию:

```
(t<- sort(table(GermanCredit$ResidenceDuration),
                  decreasing = TRUE))
t[1]/t[2]
```

```
  4   2   3   1
413 308 149 130
[1] 1.340909
```

На практике предлагается придерживаться следующих эмпирических правил для заключения о том, что некоторый предиктор обладает околонулевой дисперсией:

- доля его уникальных значений от общего числа наблюдений составляет не более 10%;
- отношение частот первых двух наиболее обычных его значений превышает 20.

В состав пакета `caret` входит функция `nearZeroVar()`, которая позволяет автоматически обнаружить предикторы, удовлетворяющие этим двум условиям:

```
# Создадим копию данных без столбца с откликом Class:
gcred = GermanCredit[, -10]
# Функция nearZeroVar() возвращает вектор номеров переменных,
# обладающих околонулевой дисперсией:
(nz = nearZeroVar(gcred))
print("Имена этих переменных:"); names(gcred)[nz]
# Удаляем предикторы с околонулевой дисперсией:
gcred.clean = gcred[, -nz]
```

```
 [1]  9 14 15 23 24 26 27 29 33 44 46 53 58
 [1]  "Имена этих переменных:"
 [1] "ForeignWorker"                     "CreditHistory.NoCredit.AllPaid"
 [3] "CreditHistory.ThisBank.AllPaid"    "Purpose.DomesticAppliance"
 [5] "Purpose.Repairs"                   "Purpose.Vacation"
 [7] "Purpose.Retraining"                "Purpose.Other"
 [9] "SavingsAccountBonds.gt.1000"       "Personal.Female.Single"
[11] "OtherDebtorsGuarantors.CoApplicant" "OtherInstallmentPlans.Stores"
[13] "Job.UnemployedUnskilled"
```

Описанная процедура вызывает, вероятно, некоторые сомнения у читателя, знакомого с основами метрологии. Во-первых, сама по себе близость дисперсии к нулевому значению ни о чем не говорит, поскольку все зависит от шкалы измерений (для субмолекулярных конструкций изменения размера в миллимикроны могут быть существенными, в то время, как для астрономических наблюдений ошибка в несколько километров является ничтожной). Поэтому дисперсия измерений считается недопустимо малой, если она не превосходит оценки погрешности измерений (или ошибки воспроизведения опыта). Во-вторых, следует уточнить, что эта процедура является эвристикой, полезной лишь для наблюдений, измеренных в порядковых или счетных шкалах с небольшим интервалом значений.

Другая проблема использования таких классических методов, как линейная регрессия или логистическая регрессия заключается в *мультиколлинеарности*. Наличие нескольких высоко коррелирующих друг с другом предикторов может привести к созданию неустойчивых решений или вообще сделать построение модели невозможным. Поскольку такие переменные несут, по сути, одинаковую информацию, то удаление части из них не приведет к заметному снижению качества модели.

Выберем наиболее коррелирующие пары переменных из таблицы `GermanCredit`. Составим для этого специальную функцию (А. Шипунов):

```
# Наибольшие значения треугольной матрицы
top.mat <- function(X, level=0.45, N=12, values=TRUE) {
X.nam <- row.names(X)
X.tri <- as.vector(lower.tri(X))
X.rep.g <- rep(X.nam, length(X.nam))[X.tri]
X.rep.e <- rep(X.nam, each=length(X.nam))[X.tri]
X.vec <- as.vector(X)[X.tri]
X.df <- data.frame(Var1=X.rep.g, Var2=X.rep.e, Value=X.vec)
{if (values)
    {X.df <- X.df[abs(X.df$Value) >= level, ]
    X.df <- X.df[order(-abs(X.df$Value)), ]}
else
    {X.df <- X.df[order(-abs(X.df$Value)), ]
    X.df <- X.df[1:N, ]}}
row.names(X.df) <- seq(1, along=X.df$Value)
return(X.df)
}
top.mat(cor(gcred.clean))
```

```
                                Var1                                Var2      Value
1            OtherInstallmentPlans.None          OtherInstallmentPlans.Bank -0.8405461
2                      Housing.ForFree                    Property.Unknown  0.7798526
3                  Personal.Male.Single          Personal.Female.NotSingle -0.7380357
4                           Housing.Own                       Housing.Rent -0.7359677
5  OtherDebtorsGuarantors.Guarantor       OtherDebtorsGuarantors.None -0.7314079
6                 CreditHistory.Critical           CreditHistory.PaidDuly -0.6836174
```

```
 7                      Job.SkilledEmployee          Job.UnskilledResident -0.6524383
 8                                   Amount                       Duration  0.6249842
 9             SavingsAccountBonds.Unknown     SavingsAccountBonds.lt.100 -0.5832811
 10                        Housing.ForFree                   Housing.Own -0.5484452
 11 Job.Management.SelfEmp.HighlyQualified          Job.SkilledEmployee -0.5438517
 12                  CreditHistory.PaidDuly          NumberExistingCredits -0.5403545
```

В состав пакета caret входит функция `findCorrelation()`, которая, как следует из ее названия, находит предикторы, чей уровень корреляции с другими предикторами в среднем превышает некоторый заданный пользователем порог (аргумент `cutoff`):

```
# Функция findCorrelation() возвращает вектор
# номеров переменных с высокой корреляцией:
(highCor = findCorrelation(cor(gcred.clean), cutoff = 0.75))
print("Имена этих переменных:")
names( gcred.clean)[highCor]
# Удаляем эти переменные:
gcred.clean =  gcred.clean[, -highCor]
```

```
 [1] 40 42
[1] Имена этих переменных:
[1] "Property.Unknown"
[2] "OtherInstallmentPlans.None"
```

Наконец, специальная функция `findLinearCombos()` предназначена для нахождения и исключения переменных, связанных линейными зависимостями. Если в выборке, например, есть переменная, которая может быть выражена через сумму нескольких других переменных, то такой предиктор можно рассматривать как малоинформативный (несущий дублирующую информацию) и его можно исключить из построения модели. Результатом функции `findLinearCombos()` является список из двух элементов: `$linearCombos` – список найденных линейных комбинаций и `$remove` – вектор индексов переменных, которые можно выразить через линейную комбинацию остальных переменных.

```
(linCombo <- findLinearCombos(gcred.clean))
# Удаляем эти переменные:
gcred.clean =  gcred.clean[, -linCombo$remove]
dim(gcred.clean)
```

```
$linearCombos[[1]]
[1] 30  9 10 11 12 26 27 28 29
$linearCombos[[2]]
[1] 34  9 10 11 12 31 32 33
$linearCombos[[3]]
[1] 43  9 10 11 12 41 42
$remove
[1] 30 34 43
```

В результате всех этих операций число предикторов сократилось с 61 до 43.

Отметим, что очистка исходных данных от "ненужных" предикторов с использованием представленных функций может показаться излишне радикальной. Во-первых, например, непонятно, почему рекомендуется удалить признак `OtherInstallmentPlans.None`, а не составляющий с ним корреляционную пару `OtherInstallmentPlans.Bank`. Во-вторых, разработаны более продвинутые методы минимизации "корреляционных плеяд":

использование фактора инфляции дисперсии *VIF* или различные последовательные алгоритмы селекции. Наконец, для некоторых алгоритмов вышеописанные проблемы не несут реальной угрозы: например, деревья принятия решений нечувствительны к признакам с околонулевой дисперсией, а регрессия на главные компоненты способна преодолеть эффект мультиколлинеарности. Однако в условиях очень большого числа переменных подобные "простые" методы редукции могут оказаться неожиданно эффективными. Кроме того, при большом числе предикторов удаление "ненужных" переменных может ускорить вычисления.

3.3. Предварительная обработка: преобразование и групповая трансформация переменных

Необходимость преобразования исходных значений предикторов может быть вызвана разными причинами. Например, некоторые статистические методы требуют, чтобы все предикторы измерялись в одинаковых единицах. В других случаях качество модели может в значительной мере зависеть от характера распределения данных или наличия выбросов. Большинство наиболее распространенных способов преобразования количественных предикторов может быть реализовано функцией `preProcess()` из пакета `caret`. Ее основными аргументами являются следующие:

```
preProcess(x, method = c("center", "scale"), na.remove = TRUE,
                              verbose = FALSE),
```

где

- `x` – таблица или матрица с исходными данными (все переменные должны быть количественными);
- `method` – список с названиями методов трансформации;
- `verbose` – флаг для указания необходимости выводить протокол обработки.

Методы предобработки можно условно разделить на три группы:

- преобразование отдельных предикторов: "`center`", "`scale`", "`range`","`expoTrans`", "`BoxCox`", "`YeoJohnson`";
- групповая трансформация подмножества переменных: "`spatialSign`", "`pca`" и "`ica`";
- заполнение пропущенных значений: "`knnImpute`", "`bagImpute`", "`medianImpute`".

Для приведения всех переменных к одинаковым единицам измерения служит *стандартизация*, являющаяся самой распространенной операцией предобработки (и потому задается в `preProcess()` по умолчанию). Она заключается в вычитании из исходного значения x_i некоторой переменной X соответствующего среднего значения $\bar{x}$ ("центрирование", или "`center`") и последующего деления полученной разницы на стандартное отклонение этой переменной σ_x ("нормализация", или "`scale`"):

$$x'_i = (x_i - \bar{x})/\sigma_x.$$

В результате стандартизации все количественные переменные будут иметь среднее значение, равное 0, и стандартное отклонение, равное 1.

Например, в таблице `GermanCredit` содержатся переменные, измеренные в разных шкалах: размер кредита `Amount` измеряется в немецких марках, возраст клиента `Age` – в годах, и т.д. Как следствие, размах значений переменных также существенно разнится: индикаторные переменные по определению варьируют от 0 до 1, тогда как размер кредита изменяется от 250 до 18424 марок. Выполним стандартизацию трех метрических переменных:

```
data(GermanCredit)
TransPred <- c("Duration","Amount","Age")
preVar <- preProcess(GermanCredit[, TransPred])
TransVar = predict(preVar, GermanCredit[, TransPred])
print("До преобразования:")
summary(GermanCredit[,TransPred])
print("После преобразования:")
summary(TransVar)
# при необходимости, обновление переменных
GermanCredit[,TransPred] <- TransVar
```

```
[1] "До преобразования:"
    Duration        Amount           Age       
 Min.   : 4.0   Min.   :  250   Min.   :19.00  
 1st Qu.:12.0   1st Qu.: 1366   1st Qu.:27.00  
 Median :18.0   Median : 2320   Median :33.00  
 Mean   :20.9   Mean   : 3271   Mean   :35.55  
 3rd Qu.:24.0   3rd Qu.: 3972   3rd Qu.:42.00  
 Max.   :72.0   Max.   :18424   Max.   :75.00  

[1] "После преобразования:"
    Duration             Amount              Age          
 Min.   :-1.402e+00   Min.   :-1.070e+00   Min.   :-1.455e+00  
 1st Qu.:-7.383e-01   1st Qu.:-6.751e-01   1st Qu.:-7.513e-01  
 Median :-2.407e-01   Median :-3.372e-01   Median :-2.238e-01  
 Mean   : 1.260e-16   Mean   : 6.628e-17   Mean   : 5.330e-17  
 3rd Qu.: 2.568e-01   3rd Qu.: 2.483e-01   3rd Qu.: 5.674e-01  
 Max.   : 4.237e+00   Max.   : 5.368e+00   Max.   : 3.468e+00  
```

Опция `method` функции `preProcess()` может принимать и другие значения. В частности, значение `"range"` приводит значения переменных к диапазону [0, 1], а `"expoTrans"` выполняет вычисление экспоненциальной функции.

Некоторые из переменных имеют также явно выраженные асимметричные распределения (например, `Amount` и `Age`), что может представлять проблему для классических статистических методов. Часто решить эту проблему позволяют такие простые преобразования исходных значений, как извлечение квадратного корня, логарифмирование или расчет обратных значений. Если истинное нормализующее преобразование неизвестно, лучшим считается преобразование Бокса-Кокса (Box-Cox transformation – Box, Cox, 1964), которое позволяет найти оптимальное решение, в первую очередь, для нормализации дисперсии.

Универсальное семейство преобразований Бокса-Кокса (БК) случайной величины x зависит от значения параметра λ:

- при $\lambda \neq 0$ мы имеем дело со степенным преобразованием вида $y(\lambda) = \frac{x^{\lambda} - 1}{\lambda}$, где показатель степени может принимать любые положительные или отрицательные значения;
- в частных случаях после подстановки параметра λ в основную формулу будем иметь: $y = 1/x$ при $\lambda = -1$; $y = 1/\sqrt{x}$ при $\lambda = -0.5$; $y = \sqrt{x}$ при $\lambda = 0.5$; $y = x^2$ при $\lambda = 2$ и т.д.;
- при $\lambda = 0$ используется логарифмическое преобразование $y(\lambda) = \ln(x)$ (поскольку деление на нуль приводит к неопределенности).

Таким образом, большинство "простых формул" представляют собой лишь частный случай преобразования БК.

Применим к метрическим показателям таблицы БК-преобразование

```
preBox <- preProcess(GermanCredit[, TransPred],
                     method="BoxCox")
```

```
Pre-processing: Box-Cox transformation
Lambda estimates for Box-Cox transformation:
0.1, -0.1, -0.7
```

Для предиктора `Amount` мы получили расчетное значение $\lambda = -0.1$. Результат преобразования с этим параметром представим графически (рис. 3.1):

```
BoxVar <- predict(preBox,GermanCredit[,TransPred])
y <- factor(GermanCredit$Class)
trellis.par.set(theme = col.whitebg(), warn = FALSE)
featurePlot(GermanCredit$Amount, y, "density",
            labels = c("Amount",""))
featurePlot(BoxVar[,2], y, "density",
            labels = c("Box-Amount",""))
```

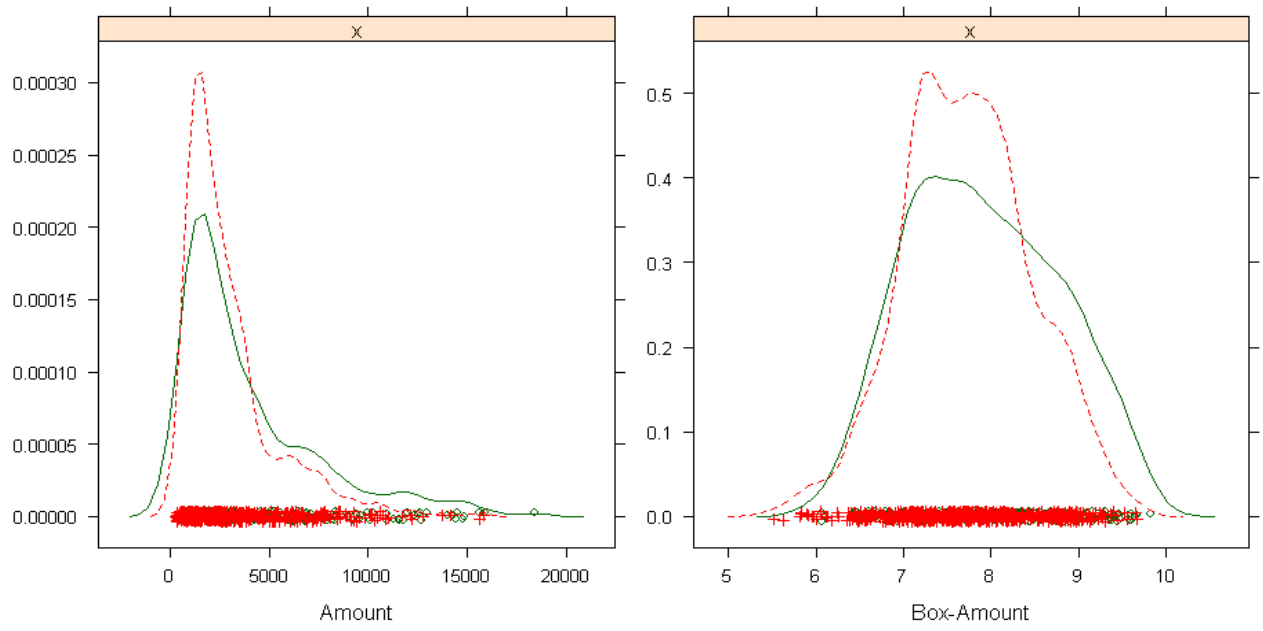


Рис. 3.1. Результат преобразования Бокса-Кокса для размера кредита в немецком банке

При значении параметра `method="YeoJohnson"` выполняется трансформация Йео-Джонсона (Yeo-Johnson), которая весьма похожа на БК-преобразование, но учитывает в расчетах нулевые и отрицательные значения обрабатываемых переменных. Другой функцией, также позволяющей оценить оптимальное значение λ для набора исходных данных, является функция `BoxCoxTrans()` из пакета `caret`.

В ряде случаев возникает необходимость одновременного преобразования целой группы предикторов или сразу всех предикторов, входящих в тот или иной набор данных. Как правило, такая необходимость возникает при наличии многомерных выбросов, а также когда исследователь желает снизить размерность исходной задачи, например, за счет удаления некоторого количества высоко коррелирующих предикторов.

Если ожидается, что тот или иной метод построения предсказательной модели чувствителен к наличию многомерных выбросов, исходные данные можно преобразовать при помощи метода пространственных признаков (spatial sign – Serneels et al., 2006). С математической точки зрения, этот метод проецирует значения предикторов на поверхность многомерной сферы, в результате чего отдельные наблюдения становятся нормированными относительно центра этой сферы: $X'_{ij} = X_{ij} / \sum_{i=1}^{m} X_{ij}^2$, где m – это число преобразуемых предикторов.

Поскольку знаменатель выражает квадрат расстояния до центра распределения предикторов, перед применением этого метода важно выполнить стандартизацию всех предикторов:

```
preProcess(x, method = c("center", "scale", "spatialSign"))
```

чем будет достигнут примерно одинаковый их вклад в величину квадрата расстояния. Кроме того, важно уточнить, что поскольку это преобразование применяется одновременно к группе предикторов, то последующее изменение их состава может исказить смысл проведенной трансформации.

Как было показано в разделе 2.4, одним из наиболее популярных методов снижения размерности исходного набора данных является анализ главных компонент PCA (principle components analysis). Этот метод позволяет сформировать ортогональную систему координат, обеспечивающую оптимальное расположение точек объектов относительно осей главных компонент нового редуцированного пространства.

Пересчет из исходной системы координат в p-мерное ($m > p$) пространство главных компонент осуществляется с использованием линейных ортогональных преобразований переменных X_i: $T_k = \sum_{i=1}^{m} P_{ik}(X_i - \mu_i)$, где P_{ik} – нагрузки (компоненты собственного вектора, соответствующего k-му собственному значению), $k = 1, 2, \ldots, p$. Основной критерий качества такого преобразования – достаточно высокая доля общей вариации исходных данных, объясняемая p первыми собственными значениями (например, 80% или 95%).

Для реализации PCA-процедуры с помощью `preProcess()` необходимо определить параметр `thresh` – кумулятивную долю дисперсии исходных данных, которая должна содержаться в главных компонентах (по умолчанию `thresh = 0.95`) или `pcaComp` – максимальное число главных компонент (по умолчанию этот параметр имеет значение `NULL`, т.е. он выключен, и оптимальное число главных компонент выбирается на основе `thresh`).

Построим две модели логистической регрессии на основе "очищенных"

```
y <- factor(GermanCredit$Class)
gcred <- GermanCredit[, -10]
nz <- nearZeroVar(gcred)
gcred.clean <- gcred[, -nz]
highCor <- findCorrelation(cor(gcred.clean), cutoff = 0.75)
gcred.clean <-  gcred.clean[, -highCor]
linCombo <- findLinearCombos(gcred.clean)
gcred.clean <-  gcred.clean[, -linCombo$remove]
dim(gcred.clean)
```

(см. раздел 3.2) данных таблицы `GermanCredit` и после PCA-преобразования:

```
[1] 1000   43
```

Определим оптимальную размерность пространства главных компонент по критерию Кайзера-Гуттмана, который рекомендует оставить только те главные компоненты, собственные значения которых превышают их среднее

```
library(vegan)
mod.pca <- rda(gcred.clean, scale=TRUE)
ev <- mod.pca$CA$eig
# Иллюстрация Критерия Кайзера-Гуттмана
barplot(ev, col="bisque", las=2)
abline(h=mean(ev), col="red")
legend("topright", "Средние собственных значений",
              lwd=1, col=2, bty="n")
c(length(ev[ev > mean(ev)]), sum(ev[ev > mean(ev)])/sum(ev))
```

(рис. 3.2):

```
[1] 19.0000000  0.7267951
```

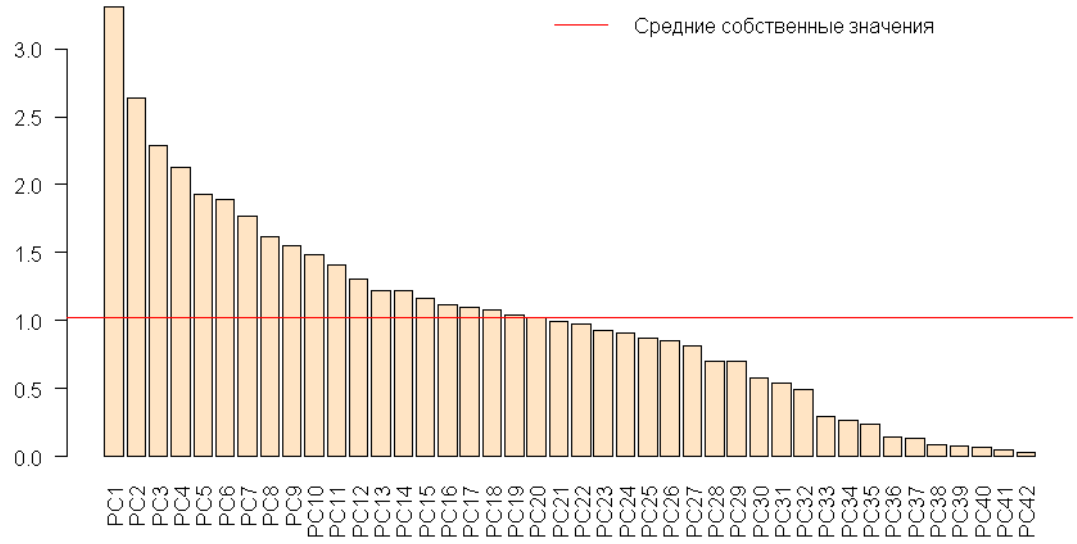


Рис. 3.2. Оценка числа главных компонент по критерию Кайзера-Гуттмана

Сформируем новую таблицу предикторов из 19 главных компонент, которые объясняют 72.7% общей дисперсии:

```
prePCA <- preProcess(gcred.clean,
          method= c("center", "scale", "pca"), pcaComp=19)
gcred.pca <- predict(prePCA,gcred.clean)
```

Попробуем сравнить точность прогноза двух моделей логистической регрессии с использованием функции `train()`, которая будет предметом нашего подробного рассмотрения ниже. Для тестирования моделей будем многократно (`times = 100`) случайным образом делить всю выборку на обучающую (800 объектов или 80%) и контрольную (200 объектов или 20%), для чего с помощью функции `createDataPartition()` создадим соответствующую "заготовку" `train.index`. Метод тестирования `method="LGOCV"` (многократное разбиение на обучающую и контрольную выборки) и `train.index` определим в специальном объекте `trControl`. На функцию `train()` подадим данные для построения модели, тип модели и условия тестирования:

```
train.index <- createDataPartition(y, p = .8, times = 100)
trControl = trainControl(method="LGOCV", index = train.index)
print("Модель на основе исходного набора из 43 предикторов")
modSource <- train(gcred.clean, y, "glm",
        family=binomial, trControl = trControl)
print("Модель на основе 19 главных компонент")
(modPCA <- train(gcred.pca, y, "glm",
        family=binomial, trControl = trControl))
```

```
[1] Модель на основе исходного набора из 43 предикторов'
Resampling: Repeated Train/Test Splits (100 reps, 0.75%)
Resampling results
  Accuracy  Kappa  Accuracy SD  Kappa SD
  0.753     0.37   0.0251       0.0603

[1] Модель на основе 19 главных компонент
Resampling results
  Accuracy  Kappa  Accuracy SD  Kappa SD
  0.749     0.362  0.0224       0.0554
```

Нельзя сказать, что преобразовав переменные, мы получили более точную модель, но зато обошлись значительно меньшим числом переменных.

При значении параметра `method="ica"` функции `preProcess()` используется другой метод снижения размерности пространства переменных – анализ независимых компонент ICA (Independent Component Analysis), который выполняет ту же задачу, что и PCA, однако основан на несколько иных математических концепциях и процедурах. В частности, цель ICA состоит в нахождении таких новых компонент (нового пространства латентных переменных), которые взаимно независимы в полном статистическом смысле.

Заметим в заключение, что общим недостатком любых преобразований является потеря возможности количественно интерпретировать роль отдельных предикторов, поскольку они больше не будут выражаться в

исходных единицах измерения. Это, разумеется, не является большой проблемой, если модель разрабатывается исключительно для прогнозирования и не предназначена для объяснения влияния тех или иных переменных на прогнозируемый отклик. Однако важной составляющей моделирования является оценка вклада, или "важности", каждого предиктора при получении предсказаний на основе той или иной модели.

Эти количественные показатели важности переменных могут быть рассчитаны функцией `varImp()`. Смысл оценивающих метрик варьирует в зависимости от конкретного алгоритма: например, для моделей классификации можно использовать площадь под ROC-кривой (см. раздел 2.3) и оценивать этот показатель путем добавления или исключения каждого предиктора модели (рис. 3.3):

```
plot(varImp(modSource, scale = FALSE))
```

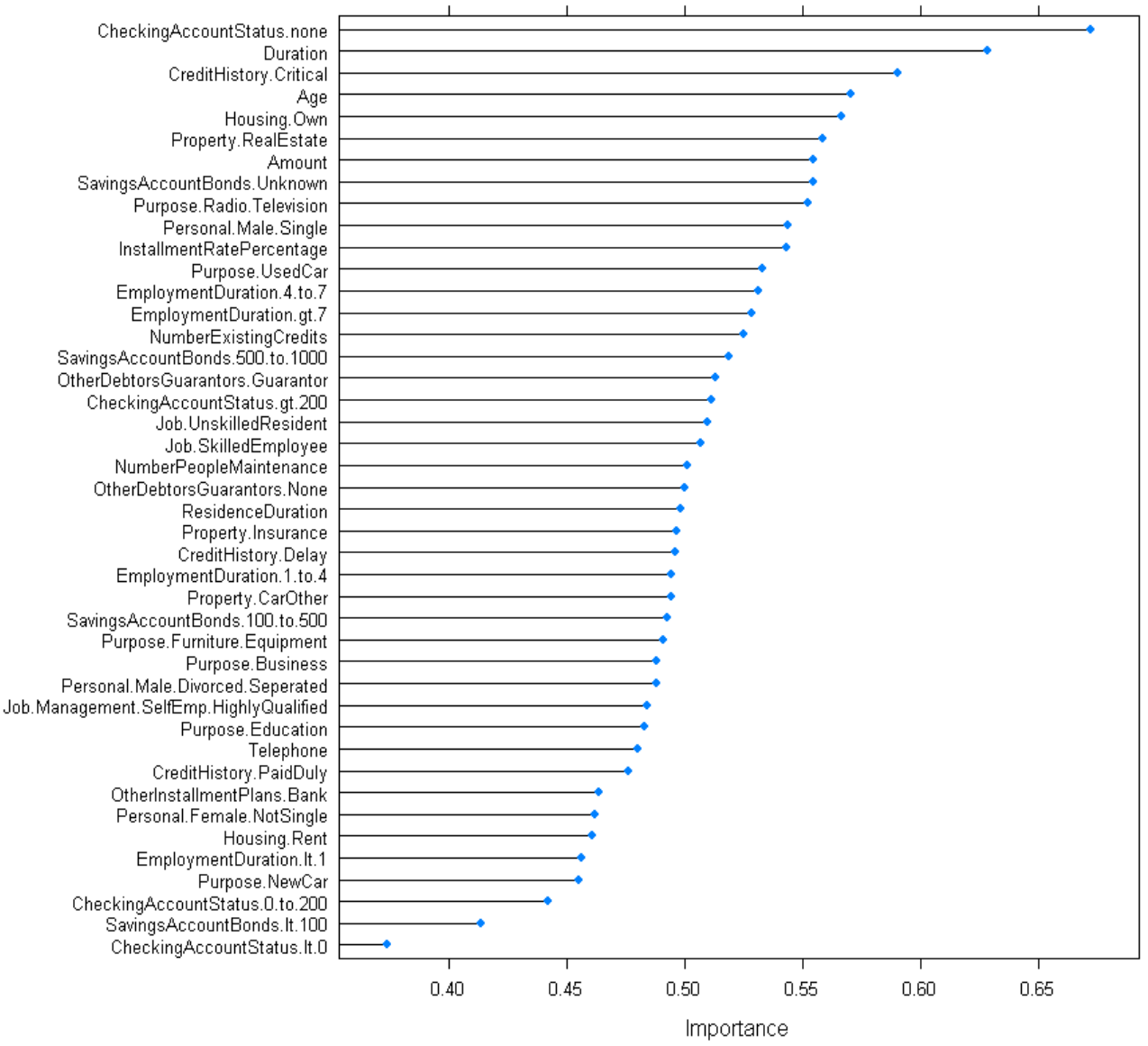


Рис. 3.3. Важность отдельных показателей клиентов банка для оценки их кредитоспособности

Разумеется, если мы будем в этом же ключе анализировать модель на основе главных компонент, то столкнемся с системой трудно интерпретируемых показателей.

3.4. Заполнение пропущенных значений в данных

К сожалению, на практике в ходе сбора данных далеко не всегда удается полностью укомплектовать исходные таблицы. Пропуски отдельных значений являются повсеместным явлением и поэтому, прежде чем начать применять статистические методы, обрабатываемые данные следует привести к "каноническому" виду. Для этого необходимо либо удалить фрагменты объектов с недостающими элементами, либо заменить имеющиеся пропуски на некоторые разумные значения.

Проблема "борьбы с пропусками" столь же сложна, как и сама статистика, поскольку в этой области существует впечатляющее множество подходов. В русскоязычных книгах по использованию R (Кабаков, 2014; Мастицкий, Шитиков, 2015) бегло представлены только некоторые функции пакета `mice`, который, несмотря на свою "продвинутость", мало удобен для практической работы с данными умеренного и большого объема. Хорошей альтернативой являются методы `"knnImpute"`, `"bagImpute"` и `"medianImpute"` функции `preProcess()` из пакета `caret`, которую мы рассмотрели в разделе 3.3 как инструмент для трансформации данных.

Используем в качестве примера для дальнейших упражнений таблицу `algae`, включенную в пакет `DMwR` и содержащую данные гидробиологических исследований обилия водорослей в различных реках. Каждое из 200 наблюдений содержит информацию о 18 переменных, в том числе:

- три номинальных переменных, описывающих размеры `size = c("large", "medium", "small")` и скорость течения реки `speed = c("high", "low", "medium")`, а также время года `season = c("autumn", "spring", "summer", "winter")`, сопряженное с моментом взятия проб;
- 8 переменных, составляющих комплекс наблюдаемых гидрохимических показателей: максимальное значение pH `mxPH` (1), минимальное содержание кислорода `mnO2` (2), хлориды `Cl` (10), нитраты `NO3` (2), ионы аммония `NH4` (2), орто-фосфаты `oPO4` (2), общий минеральный фосфор `PO4` (2) и количество хлорофилла *a* `Chla` (12) (в скобках приведено число пропущенных значений);
- средняя численность каждой из 7 групп водорослей `a1-a7` (видовой состав не идентифицировался).

Читатель может самостоятельно воспользоваться функциями описательной статистики `summary()` или `describe()` из пакета `Hmisc`, а мы постараемся поддержать добрую традицию и привести парочку примеров диаграмм, построенных с использованием пакета `ggplot2` (рис. 3.4):

```
library(DMwR)
library(ggplot2)
summary(algae) # вывод не приводится
```

```
ggplot(data=algae[!is.na(algae$mnO2),], aes(speed , mnO2)) +
       geom_violin(aes(fill = speed), trim = FALSE,
       alpha = 0.3) +
       geom_boxplot(aes(fill = speed), width = 0.2,
       outlier.colour = NA) +
       theme(legend.position = "NA")
```

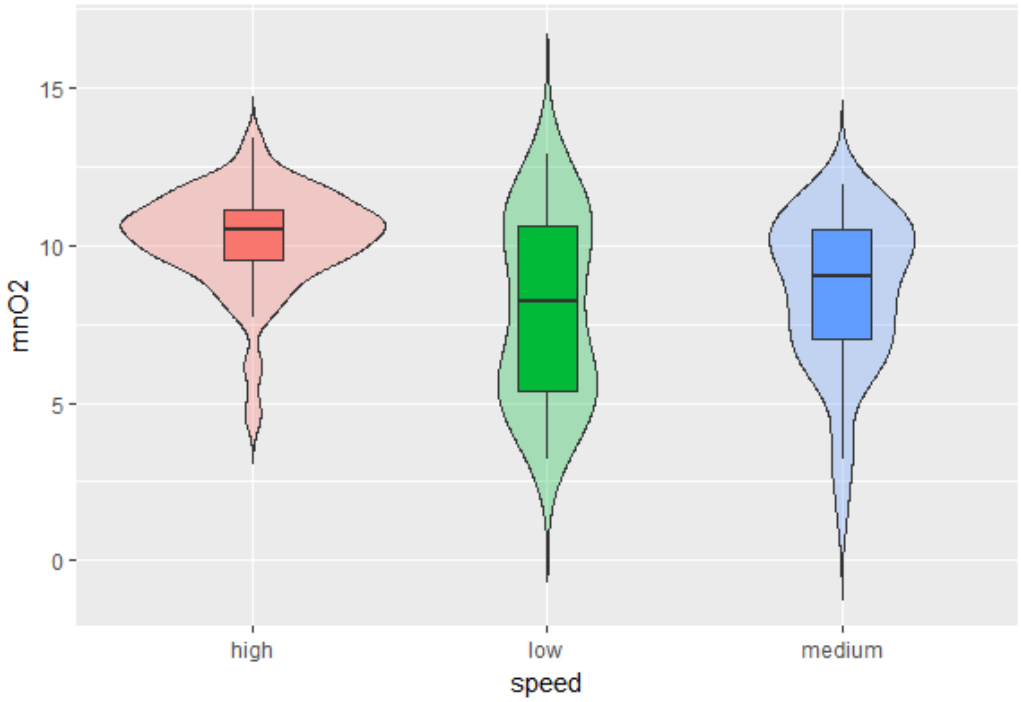


Рис. 3.4. Распределения значений содержания кислорода в воде рек с разной скоростью течения

На рис. 3.4 мы получили так называемую "*скрипичную диаграмму*" (violin plot), которая объединяет в себе идеи диаграмм размахов и кривых распределения вероятности. Суть достаточно проста: продольные края "ящиков с усами" (для сравнения приведены тоже) замещаются кривыми плотности вероятности. В итоге, например, легко можно выяснить не только тот факт, что в потоках с быстрым течением (`high`) содержание кислорода выше, но и ознакомиться с характером распределения соответствующих значений.

Другой пример – категоризованные графики, удобные для визуализации данных, разбитых на отдельные подмножества (категории), каждое из которых отображается в отдельной диаграмме подходящего типа. Такие диаграммы, или "панели" (от англ. panels, facets или multiples), определенным образом упорядочиваются и размещаются на одной странице. Из графиков, представленных на рис. 3.5, легко увидеть, что численность водорослей группы `a1` падает с увеличением концентрации фосфатов.

```
qplot(PO4, a1, data = algae[!is.na(algae$PO4), ]) +
      facet_grid(facets = ~ season)+
        geom_smooth(color="red", se = FALSE)
```

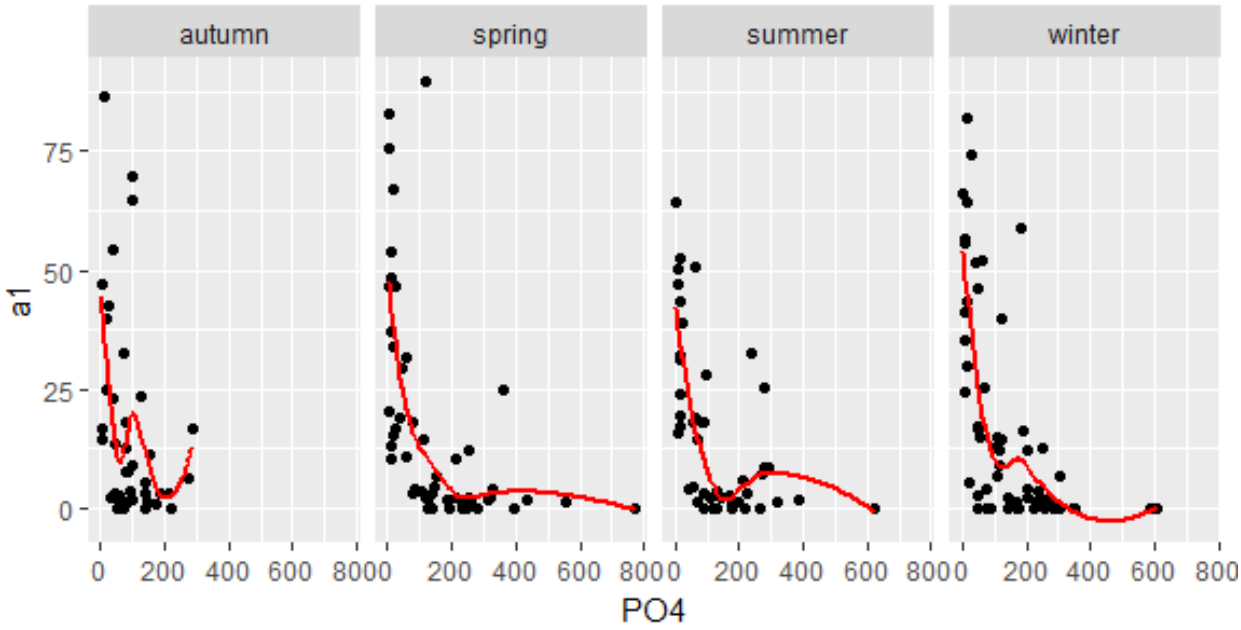


Рис. 3.5. Графики изменения обилия водорослей в зависимости от содержания минерального фосфора в разное время года

Однако в контексте темы этого раздела важно обратить внимание на то, что мы все время старались блокировать появление пропущенных значений: `algae[!is.na(algae$PO4), ]`. Если в обрабатываемой таблице обнаружены недостающие данные, то в общих чертах можно избрать одну из следующих возможных стратегий:

- удалить строки с неопределенностями;
- заполнить неизвестные значения выборочными статистиками соответствующей переменной (среднее, медиана и т.д.), полагая, что взаимосвязь между переменными в имеющемся наборе данных отсутствует (это соответствует известному "наивному" подходу);
- заполнить неизвестные значения с учетом корреляции между переменными или меры близости между наблюдениями;
- постараться обходить эту неприятную ситуацию, используя, например, формальный параметр `na.rm` некоторых функций.

Последняя альтернатива является наиболее ограничивающей, поскольку она далеко не во всех случаях позволяет осуществить необходимый анализ. В свою очередь, удаление целых строк данных слишком радикально и может привести к серьезным потерям важной информации:

```
# Число строк с пропущенными значениями
nrow(algae[!complete.cases(algae),])
# Их удаление
algae <- na.omit(algae)
```

```
[1] 16
```

Можно удалить не все строки, а только те, в которых число пропущенных значений превышает, например, 20% от общего числа переменных, для чего существует специальная функция из пакета DMwR:

```
data(algae)
manyNAs(algae, 0.2)
algae <- algae[-manyNAs(algae, 0.2), ]
```

```
[1]  62 199
```

В результате мы удалили только две строки (62-ю и 199-ю), где число пропущенных значений больше одного. Обратите внимание, что выполняя команду data(algae), мы каждый раз обновляем в памяти содержимое этого набора данных.

Если мы готовы принять гипотезу о том, что зависимостей между переменными нет, то простым и часто весьма эффективным способом заполнения пропусков является использование средних значений. В том случае, если есть сомнения в нормальности распределения данных, предпочтительнее использовать медиану. Покажем, как это можно сделать с использованием функции preProcess() из пакета caret:

```
data(algae)
ind <- apply(algae, 1, function(x) sum(is.na(x))) > 0
algae[ind, 4:11]
```

	mxPH	mnO2	Cl	NO3	NH4	oPO4	PO4	Chla
28	6.80	11.1	9.000	0.630	20	4.000	NA	2.70
38	8.00	NA	1.450	0.810	10	2.500	3.000	0.30
48	NA	12.6	9.000	0.230	10	5.000	6.000	1.10
55	6.60	10.8	NA	3.245	10	1.000	6.500	NA
56	5.60	11.8	NA	2.220	5	1.000	1.000	NA
57	5.70	10.8	NA	2.550	10	1.000	4.000	NA
58	6.60	9.5	NA	1.320	20	1.000	6.000	NA
59	6.60	10.8	NA	2.640	10	2.000	11.000	NA
60	6.60	11.3	NA	4.170	10	1.000	6.000	NA
61	6.50	10.4	NA	5.970	10	2.000	14.000	NA
62	6.40	NA	NA	NA	NA	NA	14.000	NA
63	7.83	11.7	4.083	1.328	18	3.333	6.667	NA
116	9.70	10.8	0.222	0.406	10	22.444	10.111	NA
161	9.00	5.8	NA	0.900	142	102.000	186.000	68.05
184	8.00	10.9	9.055	0.825	40	21.083	56.091	NA
199	8.00	7.6	NA	NA	NA	NA	NA	NA

```
library(caret)
pPmI <- preProcess(algae[, 4:11], method = 'medianImpute')
algae[, 4:11] <- predict(pPmI, algae[, 4:11])
(Imp.Med <- algae[ind, 4:11])
```

	mxPH	mnO2	Cl	NO3	NH4	oPO4	PO4	Chla
28	6.80	11.1	9.000	0.630	20.0000	4.000	**103.2855**	2.700
38	8.00	**9.8**	1.450	0.810	10.0000	2.500	3.0000	0.300
48	**8.06**	12.6	9.000	0.230	10.0000	5.000	6.0000	1.100
55	6.60	10.8	**32.730**	3.245	10.0000	1.000	6.5000	**5.475**
56	5.60	11.8	**32.730**	2.220	5.0000	1.000	1.0000	**5.475**
57	5.70	10.8	**32.730**	2.550	10.0000	1.000	4.0000	**5.475**
58	6.60	9.5	**32.730**	1.320	20.0000	1.000	6.0000	**5.475**
59	6.60	10.8	**32.730**	2.640	10.0000	2.000	11.0000	**5.475**
60	6.60	11.3	**32.730**	4.170	10.0000	1.000	6.0000	**5.475**

```
61  6.50 10.4 32.730 5.970  10.0000   2.000  14.0000  5.475
62  6.40  9.8 32.730 2.675 103.1665  40.150  14.0000  5.475
63  7.83 11.7  4.083 1.328  18.0000   3.333   6.6670  5.475
116 9.70 10.8  0.222 0.406  10.0000  22.444  10.1110  5.475
161 9.00  5.8 32.730 0.900 142.0000 102.000 186.0000 68.050
184 8.00 10.9  9.055 0.825  40.0000  21.083  56.0910  5.475
199 8.00  7.6 32.730 2.675 103.1665  40.150 103.2855  5.475
```

Альтернативой "наивному" подходу является учет структуры связей между переменными. Например, можно воспользоваться тем, что между двумя формами фосфора `oPO4` и `PO4` существует тесная корреляционная связь. Тогда, например, можно избавиться от некоторых неопределенностей в показателе `PO4`, вычислив его пропущенные значения по уравнению простой регрессии:

```
data(algae)
lm(PO4 ~ oPO4, data = algae)
```

```
Coefficients:
(Intercept)         oPO4
     42.897        1.293
```

```
# Функция  вывода значений PO4 в зависимости от oPO4
fillPO4 <- function(oP) {if (is.na(oP)) return(NA)
      else return(42.897 + 1.293 * oP)
}
# Восстановление пропущенных значений PO4
algae[is.na(algae$PO4), 'PO4'] <-
   sapply(algae[is.na(algae$PO4), 'oPO4'], fillPO4)
   algae[ind, 10]
```

```
[1]  48.069   3.000   6.000   6.500   1.000   4.000   6.000
11.000   6.000
[10]  14.000  14.000   6.667  10.111 186.000  56.091      NA
```

Одно из пропущенных значений удалось восстановить.

Разумеется, легко придти к мысли не утруждать себя перебором всех возможных корреляций, а учесть все связи одновременно и целиком. Использование метода `"bagImpute"` осуществляет для каждой из имеющихся переменных построение множественной бутстреп-агрегированной модели, или бэггинг-модели (bagging), на основе деревьев регрессии, используя все остальные переменные в качестве предикторов. Этот метод мудр и точен, но требует значительных затрат времени на вычисление, особенно при работе с данными большого объема:

```
data(algae)
pPbI <- preProcess(algae[, 4:11], method = 'bagImpute')
algae[, 4:11] <- predict(pPbI, algae[, 4:11])
(Imp.Bag <- algae[ind, 4:11])
```

Наконец, третий метод функции `preProcess()` для заполнения пропусков – `"knnImpute"` – основан на простейшем, но чрезвычайно эффективном алгоритме *k* ближайших соседей (*k*-nearest neighbours) или kNN. В основе метода kNN лежит гипотеза о том, что тестируемый объект *d* будет

иметь примерно тот же набор признаков, что и обучающие объекты в локальной области его ближайшего окружения (рис. 3.6).

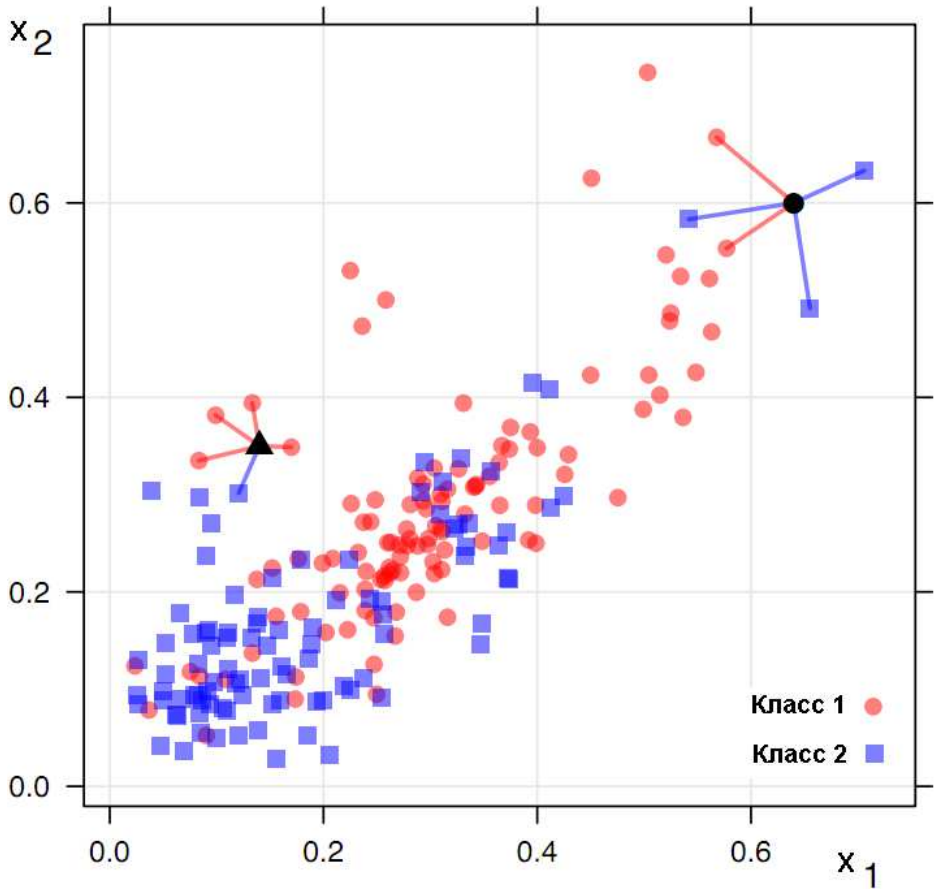


Рис. 3.6. Интерпретация метода *k* ближайших соседей

Если речь идет о классификации, то неизвестный класс объекта определяется голосованием *k* его ближайших соседей (на рис. 3.6 $k = 5$). kNN-регрессия оценивает значение неизвестной координаты *Y*, усредняя известные ее величины для тех же *k* соседних точек.

Одна из важных проблем kNN – выбор метрики, на основе которой оценивается близость объектов. Наиболее общей формулой для подсчета расстояния в *m*-мерном пространстве между объектами $\mathbf{X_1}$ и $\mathbf{X_2}$ является мера Минковского:

$$D_S(\mathbf{X_1}, \mathbf{X_2}) = [\Sigma |x_{1i} - x_{2i}|^p]^{1/r},$$

где *i* изменяется от 1 до *m*, а *r* и *p* – задаваемые исследователем параметры, с помощью которых можно осуществить нелинейное масштабирование расстояний между объектами. Мера расстояния по Евклиду получается, если принять в метрике Минковского $r = p = 2$, и является, по-видимому, наиболее общей мерой расстояния, знакомой всем со школы по теореме Пифагора. При $r = p = 1$ имеем "манхэттенское расстояние" (или "расстояние городских кварталов"), не столь контрастно оценивающее большие разности координат *x*.

Вторая проблема метода kNN заключается в решении вопроса о том, на мнение какого числа соседей *k* нам целесообразно положиться? В свое время

мы обсудим этот вопрос детально, а сейчас будем ориентироваться на значение $k = 5$, используемое по умолчанию:

```
data(algae)
pPkI <- preProcess(algae[, 4:11], method = 'knnImpute')
alg.stand <- predict(pPkI, algae[, 4:11])
```

Получив в результате применения `predict()` матрицу переменных с пропущенными значениями, заполненными этим методом, мы с удивлением обнаруживаем, что данные оказались стандартизованными (т.е. центрированными и нормированными на стандартное отклонение). Но а как иначе можно было вычислить меру близости по переменным, измеренным в разных шкалах? Пришлось для возвращения в исходное состояние применить обратную операцию:

```
m <- pPkI$mean
sd <- pPkI$std
algae[, 4:11] <- t(apply(alg.stand, 1,
                         function (r) m + r * sd))
(Imp.Knn <- algae[ind, 4:11])
```

Наконец, зададимся следующим закономерным вопросом: а какой метод лучше? Обычно эта проблема не имеет теоретического решения, и исследователь полагается на собственную интуицию и опыт. Но мы можем оценить, насколько расходятся между собой результаты, полученные каждым способом заполнения. Для этого сформируем блок данных из 3 * 16 = 48 строк исходной таблицы с пропусками, заполненными тремя методами (`"Med"`, `"Bag"`, `"Knn"`), и выполним снижение размерности пространства переменных методом главных компонент в двумерное (см. раздел 2.4). Посмотрим, как "лягут карты" на плоскости:

```
ImpVal <- rbind(Imp.Med, Imp.Knn)
ImpVal <- rbind(ImpVal, Imp.Bag)
Imp.Metod <- as.factor(c(rep("Med", 16), rep("Knn", 16),
rep("Bag", 16)))
library(RANN)
library(vegan)
Imp.M <- rda(ImpVal ~ Imp.Metod, ImpVal)
plot(Imp.M, display = "sites", type = "p")
ordihull(Imp.M, Imp.Metod, draw = "polygon", alpha = 67,
         lty = 2, col = c(1, 2, 3), label = TRUE)
```

На рис. 3.7 мы выделили контуром (hull), проведенным через крайние точки, области каждого из трех блоков данных и поместили метку метода в центры тяжести полученных многоугольников. Понятно, что медианное заполнение характеризуется меньшей вариацией результатов, поскольку игнорирует специфичность свойств каждого объекта. Оба других метода, учитывающих внутреннюю структуру данных, дали приблизительно похожие результаты.

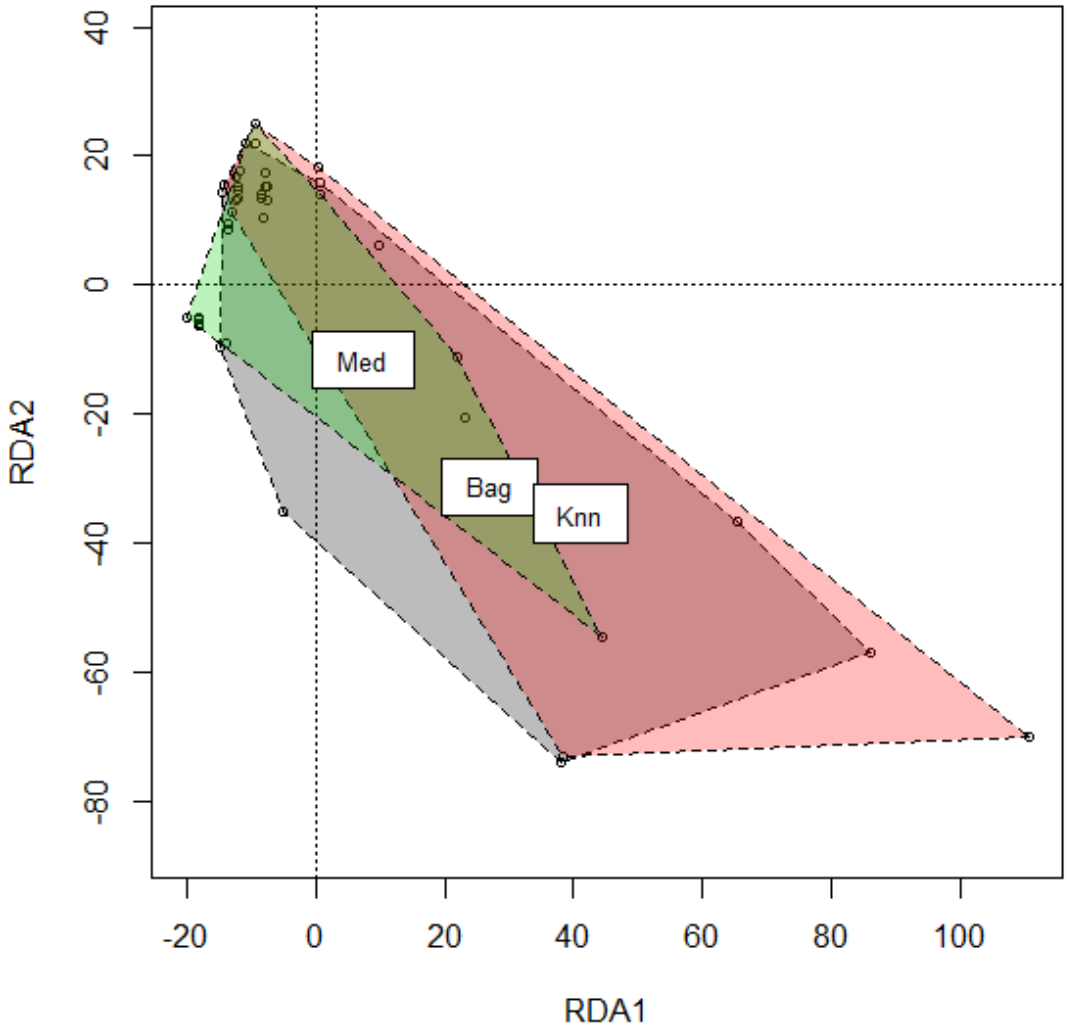


Рис. 3.7. Ординационная диаграмма блоков данных таблицы `algae` с пропущенными значениями, заполненными разными способами

3.5. Функция `train()` из пакета `caret`

Как подчеркивалось ранее в разделе 3.1, пакет `caret` был разработан как эффективная надстройка, позволяющая унифицировать и интегрировать использование множества различных функций и методов построения моделей классификации и регрессии, представленных в других пакетах R. При этом происходит всестороннее тестирование и оптимизация настраиваемых коэффициентов и гиперпараметров моделей. Эта разработанная единая технология реализована в функции `train()`, использующей полуавтоматические интеллектуальные подходы и универсальные критерии качества с применением алгоритмов ресэмплинга.

Перед выполнением подгонки модели с помощью функции `train()` необходимо задать соответствующий алгоритм и всю совокупность условий процесса оптимизации параметров модели, для чего при помощи функции

trainControl() создается специальный объект. Вызов этой функции со значениями, принимаемыми по умолчанию, имеет следующий вид:

```
trainControl(method = "boot",
   number = ifelse(grepl("cv", method), 10, 25), p = 0.75,
   repeats = ifelse(grepl("cv", method), 1, number),
   search = "grid", initialWindow = NULL, horizon = 1,
   fixedWindow = TRUE, verboseIter = FALSE, returnData = TRUE,
   returnResamp = "final", savePredictions = FALSE,
   classProbs = FALSE, summaryFunction = defaultSummary,
   selectionFunction = "best", seeds = NA,
   preProcOptions = list(thresh = 0.95, ICAcomp = 3, k = 5),
   sampling = NULL, index = NULL, indexOut = NULL,
   timingSamps = 0, predictionBounds = rep(FALSE, 2),
   adaptive = list(min = 5, alpha = 0.05, method = "gls",
   complete = TRUE), trim = FALSE, allowParallel = TRUE)
```

Остановимся на описании наиболее важных аргументов этой функции:

- method – метод ресэмплинга: "boot", "boot632", "cv", "repeatedcv", "LOOCV", "LGOCV" (для повторяющихся разбиений на обучающую и проверочную выборки), "none" (исследуется только модель на обучающей выборке), "oob" (для таких алгоритмов, как случайный лес, бэггинг деревьев и др.), "adaptive_cv", "adaptive_boot" или "adaptive_LGOCV";
- number – задает число итераций ресэмплинга, в частности, число блоков (folds) при перекрестной проверке;
- repeats – число повторностей (только для *k*-кратной перекрестной проверки);
- p – доля обучающей выборки от общего объема при выполнении перекрестной проверки;
- verboseIter – TRUE означает, что в ходе вычислений caret будет показывать, на каком этапе они находятся (это удобно для оценки оставшегося времени вычислений, которое часто бывает очень большим);
- search – способ перебора параметров модели (по сетке – "grid", или случайным назначением – "random");
- returnResamp и savePredictions – определяют условия сохранения результатов выполнения ресэмплинга и прогнозируемых значений, т.е. "none", только для итоговой модели "final" или все "all";
- classProbs – TRUE означает, что в процессе вычислений алгоритм будет сохранять данные о вероятностях попадания объекта в каждый класс, а не только конечные метки класса;
- summaryFunction – определяет функцию, которая вычисляет метрику качества модели при ресэмплинге;
- selectionFunction – определяет функцию выбора оптимального значения настраиваемого параметра;
- preProcOptions – список опций, который передается функции предобработки данных preProcess().

Например, при создании объекта ctrl

```
ctrl <- trainControl(method = "repeatedcv",
                     number = 10, repeats = 10)
```

параметры перекрестной проверки будут иметь следующий смысл:

- method = 'repeatedcv' означает, что будет использоваться повторная перекрестная проверка (также возможна перекрестная проверка, leave-one-out проверка, и т.д.);
- number = 10 означает, что в процессе перекрестной проверки выборку надо разбивать на 10 равных частей;
- repeats = 10 означает, что повторная перекрестная проверка должна быть запущена 10 раз.

Теперь перейдем непосредственно к описанию функции train(), которая имеет следующий формат вызова:

```
train(x, y, method = "rf", preProcess = NULL, weights = NULL,
  metric = ifelse(is.factor(y), "Accuracy", "RMSE"), maximize =
  ifelse(metric %in% c("RMSE", "logLoss"), FALSE, TRUE),
  trControl = trainControl(), tuneGrid = NULL, tuneLength = 3)
```

Исходные данные задаются, как всегда, либо матрицей предикторов x и вектором отклика y, либо объектом formula с указанием таблицы данных data. Следующий аргумент – method – это, в сущности, название модели классификации или регрессии, которую необходимо построить и протестировать. Если выполнить команду names(getModelInfo()), то можно увидеть список из 233 доступных методов (это количество постоянно расширяется). Тот же список, но с различными возможностями поиска и сортировки можно посмотреть по следующим адресам в Интернете:

http://topepo.github.io/caret/modelList.html или
http://topepo.github.io/caret/bytag.html

Можно просмотреть названия всех моделей, которые имеют, например, отношение к линейной регрессии:

```
library(caret)
ls(getModelInfo(model = "lm"))
```

```
 [1] "bayesglm"  "elm"       "glm"       "glmboost"   "glmnet"
 [6]"glmStepAIC" "lm"        "lmStepAIC" "plsRglm"    "rlm"
```

С каждым методом связан набор гиперпараметров, подлежащих оптимизации. Можно убедиться в том, что простая линейная регрессия (method = "lm") не имеет параметров для настройки:

```
modelLookup("lm")
```

```
  model parameter     label forReg forClass probModel
1    lm parameter parameter   TRUE    FALSE     FALSE
```

В свою очередь, деревья решений rpart, которые мы рассмотрим позднее, имеют один параметр для настройки – Complexity Parameter (аббревиатура – cp):

```
modelLookup("rpart")
```

```
  model parameter                label forReg forClass probModel
1 rpart        cp Complexity Parameter   TRUE     TRUE      TRUE
```

Из сообщений функции `modelLookup()` можно также увидеть, что линейная регрессия не используется для классификации (`forClass= FALSE`), тогда как `rpart` (от "Recursive Partitioning and Regression Trees") можно применять как для построения деревьев регрессии, так и классификации. В последнем случае модель осуществляет не только предсказание класса, но и оценивает апостериорные вероятности (`probModel = TRUE`).

Метод перекрестной проверки, заданный объектом `trControl = trainControl()`, обеспечивает сканирование настраиваемых параметров и оценку эффективности модели на каждой итерации по определенным критериям качества. При построении каждой частной модели предварительно осуществляется предобработка данных с использованием методов, перечисленных в `preProcess` (и с учетом опций `preProcOptions` объекта `trControl`).

По умолчанию аргумент `metric` использует в качестве критерия качества точность предсказания (`"Accuracy"`) в случае классификации (см. пример, рассмотренный в разделе 3.3) и корень из среднеквадратичного отклонения прогнозируемых значений от наблюдаемых (`"RMSE"`) для регрессии. Логический аргумент `maximize` уточняет, должен ли этот критерий быть максимизирован или минимизирован. Другие значения `metric` вместе с различными значениями аргументов `summaryFunction` и `selectionFunction` объекта `trControl` обеспечивают широкие возможности пользовательского назначения метрик для оценки эффективности моделей.

Количество перебираемых значений настраиваемого параметра задается аргументом `tuneLength`. Например, чтобы задать 30 повторов оценки параметра *Cp* модели `rpart` вместо исходных трех, необходимо указать `tuneLength=30`. Другой вариант – определить последовательность этих значений в списке, задающем сетку: например, `tuneGrid = expand.grid(.cp = 0.5^(1:10))`, если заранее известен диапазон, к которому принадлежит оптимизируемое значение.

После завершения перебора всех положенных комбинаций параметров модели создается объект класса `train`, соответствующие элементы которого можно извлечь с помощью суффикса `$`:

```
  ls(mytrain)
 [1]"bestTune"     "call"          "coefnames"     "control"
 [5]"dots"         "finalModel"    "maximize"      "method"
 [9]"metric"       "modelInfo"     "modelType"     "perfNames"
[13]"pred"         "preProcess"    "resample"      "resampledCM"
[17]"results"      "terms"         "times"         "trainingData"
[21]"xlevels"      "yLimits"
```

Приведем краткий пример на тему подбора полиномиальной регрессии для описания зависимости электрического сопротивления (Ом) мякоти фруктов киви от процентного содержания в ней сока, который подробно разбирался нами в разделе 2.1. Позже (раздел 2.2) мы показали, как найти

оптимальную степень полинома $d = 4$ с использованием написанной нами функции скользящего контроля.

К сожалению, функция `train()` селекцию предикторов для метода `"lm"` не выполняет и оптимальную степень полинома не настраивает. Но мы можем выполнить любую перекрестную проверку для оценки динамики изменения критериев качества модели – среднеквадратичной ошибки `RMSE` и среднего коэффициента детерминации `RSquared` (рис. 3.8):

```
library(DAAG)
data("fruitohms")
set.seed(123)
max.poly <- 7
degree <- 1:max.poly
RSquared <- rep(0,max.poly)
RMSE <- rep(0,max.poly)
# Выполним 10-кратную кросс-проверку с 10 повторностями
fitControl <- trainControl(method = "repeatedcv",
          number = 10,repeats = 10)
# Тестируем модель для различных степеней
for ( d in degree)  {
  f <- bquote(juice ~ poly(ohms, .(d)))
  LinearRegressor <- train(as.formula(f),
     data = fruitohms,
     method = "lm",trControl =fitControl)
     RSquared[d] <- LinearRegressor$results$Rsquared
     RMSE[d]<- LinearRegressor$results$RMSE
}
library(ggplot2)
Degree.RegParams = data.frame(degree,RSquared,RMSE)
ggplot(aes(x = degree,y = RSquared),
       data = Degree.RegParams) + geom_line()
ggplot(aes(x = degree,y = RMSE),
         data = Degree.RegParams) + geom_line()
```

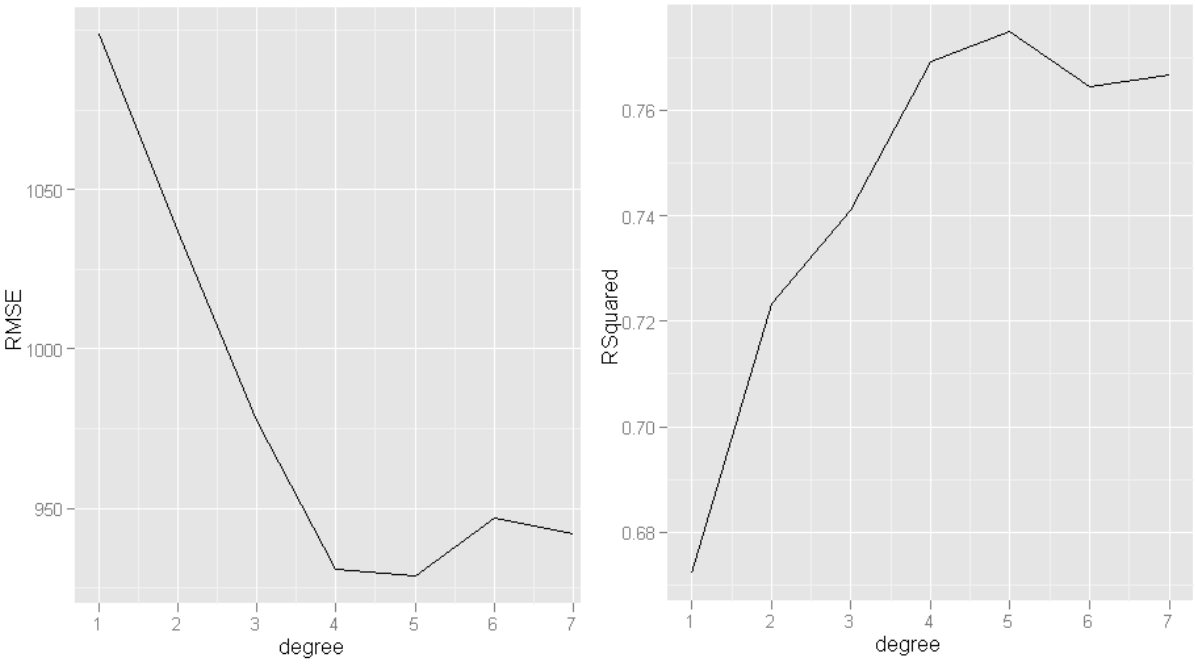


Рис. 3.8. Поиск степени функции полиномиальной регрессии с использованием функции `train()`

В отличие от ранее проведенных расчетов, минимум ошибки и максимум коэффициента детерминации имеют место при `d = 5`.

Если не задавать непосредственно объект `trControl`, то по умолчанию вместо кросс-проверки функция `train()` осуществляет бутстреппинг критериев качества подгонки `RMSE` и `RSquared`:

```
Poly5 <- train(ohms ~ poly(juice,5), data = fruitohms,
      method = "lm")
summary(Poly5$finalModel)
summary(lm(ohms ~ poly(juice, 5), data = fruitohms))
```

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       4360.0       83.2  52.406  < 2e-16 ***
poly(juice, 5)1 -16750.2      941.3 -17.796  < 2e-16 ***
poly(juice, 5)2   4750.6      941.3   5.047 1.59e-06 ***
poly(juice, 5)3   3945.6      941.3   4.192 5.27e-05 ***
poly(juice, 5)4  -3371.2      941.3  -3.582 0.000492 ***
poly(juice, 5)5   1057.3      941.3   1.123 0.263509
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 941.3 on 122 degrees of freedom
Multiple R-squared: 0.7539,     Adjusted R-squared: 0.7439
F-statistic: 74.76 on 5 and 122 DF,  p-value: < 2.2e-16
```

```
Poly5
```

```
Resampling: Bootstrap (25 reps)
     RMSE  Rsquared  RMSE SD  Rsquared SD
     966   0.734     141      0.0569
```

Мы получили в точности те же коэффициенты модели, что и при использовании базовой функции `lm()`, но для критериев `RMSE` и `RSquared` были найдены стандартные ошибки, позволяющие рассчитать доверительные интервалы этих статистик. Оценка проводилась по результатам 25 бутстреп-итераций, выполняемых функцией `train()` по умолчанию. Обратите внимание, что несмещенное бутстреп-значение коэффициента детерминации (0.734) несколько меньше, чем рассчитанное функцией `summary()` для финальной модели (0.754).

В заключении приведем сокращенную таблицу моделей, доступных для реализации в `train()`, с указанием наименований их параметров. С нашей точки зрения, таблица полезна также тем, что является своеобразным путеводителем по пакетам R и реализованным в них статистическим методам.

Список методов, моделей и их параметров, оптимизируемых с использованием функции `train()`

Модели	**Значение** `method`	**Пакет**	**Оптимизируемые параметры**
Деревья на основе рекурсивного деления (recursive partitioning)	`rpart`	`rpart`	`maxdepth`
	`ctree`	`party`	`mincriterion`
Бустинг деревьев (boosted trees)	`gbm`	`gbm`	`interaction.depth, n.trees, shrinkage`
	`blackboost`	`mboost`	`maxdepth, mstop`
	`ada`	`ada`	`maxdepth, iter, nu`
Другие модели бустинга (other boosted models)	`glmboost`	`mboost`	`mstop`
	`gamboost`	`mboost`	`mstop`
	`logitboost`	`caTools`	`nIter`
Случайный лес (random forest)	`rf`	`randomForest`	`mtry`
	`cforest`	`party`	`mtry`
Бэггинг-деревья (bagged trees)	`treebag`	`ipred`	
Нейронные сети (neural networks)	`nnet`	`nnet`	`decay, size`
Частные наименьшие квадраты (partial least squares)	`pls`	`pls, caret`	`ncomp`
Машина опорных векторов с RBF ядром (support vector machines RBF kernel)	`svmRadial`	`kernlab`	`sigma, C`
Машина опорных векторов с полиномиальным ядром (support vector machines polynomial kernel)	`svmPoly`	`kernlab`	`scale, degree, C`
Гауссовы процессы с RBF ядром (Gaussian processes with RBF kernel)	`gausssprRadial`	`kernlab`	`sigma`
Гауссовы процессы с полиномиальным ядром (Gaussian processes with polynomial kernel)	`gausssprPoly`	`kernlab`	`scale, degree`
Линейные модели наименьших квадратов (linear least squares)	`lm`	`stats`	
Многомерные адаптивные регрессионные сплайны (multivariate adaptive regression splines MARS)	`earth, mars`	`earth`	`degree, nprune`
Бэггинг-сплайны MARS (bagged MARS)	`bagEarth`	`caret, earth`	`degree, nprune`
Эластичные сети (elastic net)	`enet`	`elasticnet`	`lambda, fraction`
Лассо (the lasso)	`lasso`	`elasticnet`	`fraction`
Машины релевантных векторов с RBF ядром (relevance vector machines RBF kernel)	`rvmRadial`	`kernlab`	`sigma`

Машины релевантных векторов с полиномиальным ядром (relevance vector machines polynomial kernel	`rvmPoly`	`kernlab`	`scale, degree`
Линейный дискриминантный анализ (linear discriminant analysis)	`lda`	`MASS`	
Пошаговый диагональный дискриминантный анализ (stepwise diagonal discriminant analysis)	`sddaLDA, sddaQDA`	`SDDA`	
Логистическая регрессия для двух или более классов (logistic/multinomial regression)	`multinom`	`nnet`	`decay`
Регуляризованный дискриминантный анализ (Regularized discriminant analysis)	`rda`	`klaR`	`lambda, gamma`
Гибкий дискриминантный анализ (Flexible discriminant analysis FDA)	`fda`	`mda, earth`	`degree, nprune`
FDA на основе бэггинга (bagged FDA)	`bagFDA`	`caret, earth`	`degree, nprune`
Машины опорных векторов на основе метода наименьших квадратов с RBF ядром (least squares support vector machines RBF kernel)	`lssvmRadial`	`kernlab`	`sigma`
Метод *k*-ближайших соседей (*k* nearest neighbors)	`knn3`	`caret`	`k`
Разделение по центроидам (nearest shrunken centroids)	`pam`	`pamr`	`threshold`
Наивный байесовский классификатор (naive Bayes)	`nb`	`klaR`	`usekernel`
Обобщенный метод частных наименьших квадратов (Generalized partial least squares)	`gpls`	`gpls`	`K.prov`
Сети с квантованием обучающего вектора (learned vector quantization)	`lvq`	`class`	`k`

В последующих разделах мы приведем описание большинства перечисленных методов и продемонстрируем процесс оптимизации их параметров с помощью функции `train()`.

4. ПОСТРОЕНИЕ РЕГРЕССИОННЫХ МОДЕЛЕЙ РАЗЛИЧНОГО ТИПА

4.1. Селекция оптимального набора предикторов линейной модели

Модель множественной линейной регрессии является, безусловно, весьма полезной и широко применяемой для прогнозирования количественного отклика. Считается, что наиболее эффективный путь улучшения качества регрессии – исключение незначимых коэффициентов, или, выражаясь точнее, отбор *информативного комплекса* $\mathbf{S}_q$ из q переменных ($q < m$). Причины, по которым стоит проводить селекцию "оптимального подмножества" предикторов, Дж. Фаравэй (Faraway, 2006) видит в следующем:

1. Принцип бритвы Оккама утверждает, что из нескольких вероятных объяснений явления лучшим является самое простое. Компактная модель, из которой удалены избыточные предикторы, лучше объясняет имеющиеся данные.

2. Ненужные предикторы добавляют шум к оценке влияния других интересующих нас факторов. Иначе степени свободы (часто ограниченные) будут тратиться впустую.

3. При наличии коллинеарности некоторые переменные будут "пытаться сделать одну и ту же работу" (т.е., повторно объяснять вариацию значений зависимой переменной).

4. Если модель используется для прогнозирования, то можно сэкономить время и/или деньги, не измеряя избыточные переменные.

Алгоритмы выбора оптимального подмножества $\mathbf{S}_q$ обычно основаны на последовательном "переборном" процессе, при котором многократно создаются модели с различными наборами предикторов, лучшая из которых определяется по некоторому критерию эффективности (ошибка или точность модели, R^2, AIC и проч.). Рассмотрим технику применения и оценим полученные результаты с использованием трех таких методов: пошаговый (forward/backward) метод селекции, рекурсивное исключение и генетический алгоритм, представленные в пакете `caret`.

В качестве примера рассмотрим построение регрессионных моделей, прогнозирующих обилие водорослей группы `a1` в зависимости от гидрохимических показателей воды и условий отбора проб в различных водотоках (см. подробное описание таблицы переменных в разделе 3.4):

```
library(DMwR)
data(algae)
library(caret)
# Заполним пропуски в данных на основе алгоритма бэггинга
pPbI <- preProcess(algae[, 4:11], method='bagImpute')
algae[,4:11] <-  predict(pPbI, algae[,4:11])
# Сохраним таблицу для использования в дальнейшем
save(algae, file="algae.RData")
```

Важные предварительные выводы можно сделать, сформировав корреляционную матрицу предикторов (рис. 4.1):

```
Mcor <-cor(algae[,4:11])
library(corrplot)
corrplot(Mcor, method="color", addCoef.col="darkgreen",
         addgrid.col = "gray33", tl.col = "black")
```

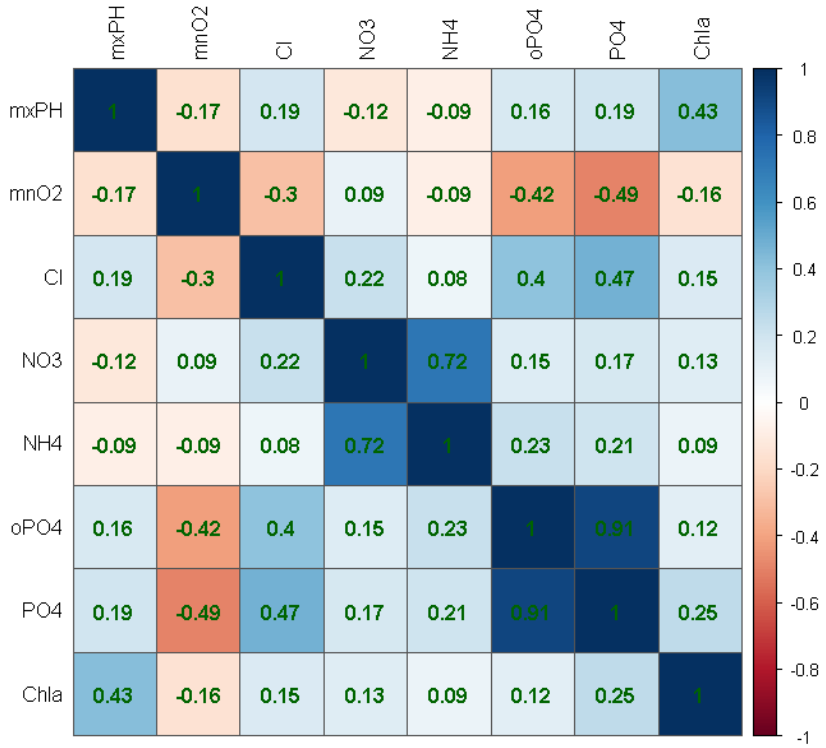


Рис. 4.1. Корреляционная матрица показателей качества воды в реках

Очевидно, что между предикторами существуют корреляционные связи умеренной силы, а коллинеарность, в целом, выражена слабо.

Полная регрессионная модель и пошаговая процедура

Построим сначала линейную модель на основе полного набора переменных:

```
lm.a1 <- lm(a1 ~ ., data = algae[, 1:12])
summary(lm.a1)
```

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)  41.2439725 23.9452133   1.722  0.08667 .
seasonspring  3.5927401  4.1371119   0.868  0.38630
seasonsummer  0.3093462  3.9942858   0.077  0.93835
seasonwinter  3.5401487  3.8521670   0.919  0.35930
sizemedium    3.6080130  3.7240594   0.969  0.33390
sizesmall     9.8601114  4.1192992   2.394  0.01769 *
speedlow      3.2806448  4.6757407   0.702  0.48380
speedmedium  -0.2221952  3.2139617  -0.069  0.94496
mxPH         -3.3723566  2.6959345  -1.251  0.21256
mnO2          1.0421490  0.7045909   1.479  0.14083
Cl           -0.0377924  0.0337294  -1.120  0.26398
```

```
NO3          -1.5215484  0.5501278  -2.766  0.00626 **
NH4           0.0016244  0.0010029   1.620  0.10702
oPO4         -0.0007233  0.0391875  -0.018  0.98529
PO4          -0.0559043  0.0300688  -1.859  0.06459 .
Chla         -0.0629710  0.0781788  -0.805  0.42159
---
Signif. codes:  0 C***T 0.001 C**T 0.01 C*T 0.05 C.T 0.1 C T 1

Residual standard error: 17.64 on 184 degrees of freedom
Multiple R-squared: 0.3686,     Adjusted R-squared: 0.3172
F-statistic: 7.162 on 15 and 184 DF,  p-value: 2.94e-12
```

Отметим, что при всей внушительной адекватности модели в целом (по *F*-критерию), почти все коэффициенты оцениваются как статистически незначимые. Выполним стандартную пошаговую процедуру включений с исключениями "слабых" предикторов, используя функцию `step()`:

```
lm_step.a1 <- step(lm.a1, trace = 0)
summary(lm_step.a1)
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 55.7204080 20.5365454   2.713  0.00727 **
sizemedium   3.3401764  3.3721698   0.991  0.32316
sizesmall   10.9864235  3.7636536   2.919  0.00393 **
mxPH        -3.7829230  2.4260402  -1.559  0.12056
NO3         -1.5247600  0.4931425  -3.092  0.00228 **
NH4          0.0014816  0.0009336   1.587  0.11416
PO4         -0.0691868  0.0102496  -6.750 1.68e-10 ***
---
Residual standard error: 17.49 on 193 degrees of freedom
Multiple R-squared: 0.3494,     Adjusted R-squared: 0.3292
F-statistic: 17.27 on 6 and 193 DF,  p-value: 5.882e-16
```

Доля значимых предикторов стала существенно выше. Проверим также, можно ли считать статистически значимой некоторое увеличение ошибки модели:

```
anova(lm.a1, lm_step.a1)
```

```
Analysis of Variance Table
  Res.Df   RSS Df Sum of Sq      F Pr(>F)
1    184 57263
2    193 59006 -9   -1743.4 0.6224  0.777
```

Выполним тестирование обоих моделей функцией `train()` из пакета `caret` (см. раздел 3.5) с использованием 10-кратной перекрестной проверки:

```
lm.a1.cv <- train(a1 ~ ., data = algae[, 1:12], method = 'lm',
                  trControl = trainControl(method = "cv"))
```

```
Resampling: Cross-Validation (10 fold)
  RMSE     Rsquared   RMSE SD   Rsquared SD
  19.0968  0.2955792  4.273678  0.1369611
```

```
lm_step.a1.cv <- train(a1 ~ size + mxPH + mnO2 + NO3 + NH4 +
          PO4, data = algae[, 1:12], method = 'lm',
              trControl = trainControl(method = "cv"))
```

```
Resampling: Cross-Validation (10 fold)
  RMSE      Rsquared   RMSE SD   Rsquared SD
  17.82252  0.3376223  3.372695  0.1312212
```

Преимущества модели, полученной с использованием пошаговой процедуры отбора предикторов, вполне очевидны.

Рекурсивное исключение переменных

Алгоритм рекурсивного исключения (RFE – Recursive Feature Elimination) выполняется следующим образом. Вначале строится модель по всем предикторам, которые ранжируются по их важности. Далее рассматривается последовательность подмножеств **S** ($\mathbf{S}_1 > \mathbf{S}_2$, и т.д.) из переменных наивысшего ранга. На каждой итерации подмножества $\mathbf{S}_i$ ранги предикторов пересматриваются, а модели пересчитываются. Итоговая модель основывается на подмножестве $\mathbf{S}_i$, обеспечивающем оптимум заданному критерию качества.

Функция `rfe()` из пакета `caret` включает алгоритм RFE в процедуру ресэмплинга, и тогда цикл рекурсивного исключения приобретает следующий вид:

```
1  Цикл для каждой итерации ресэмплинга
2  | Данные разделяются на обучающие и проверочные в зависимости,
2  |          например, от количества блоков, участвующих в перекрестной проверке
3  | Построение модели на обучающем множестве с использованием всех предикторов
4  | Предсказание на проверочной выборке
5  | Вычисление важности предикторов и их ранжирование
6  | Цикл для каждого подмножества Si, i = 1…S
7  | | Заполнение Si наиболее важными переменными
8  | | Построение модели на обучающем множестве с использованием всех
   | | предикторов Si
9  | | Предсказание на проверочной выборке
10 | | Повторное вычисление рангов важности для каждого предиктора
11 | Конец цикла
12 Конец цикла
13 Вычисление профилей эффективности для каждого Si с использованием
                                                          проверочной выборки
14 Определение оптимального числа предикторов
15 Уточнение итогового списка наилучших предикторов
16 Построение итоговой модели на основе оптимального Si с использованием
                                                       исходной обучающей выборки
```

Синтаксис параметров функций `rfe()` и `rfeControl()` в целом похож на таковой у функций `train()` и `trainControl()` (см. раздел 3.5). Отметим, что функция `rfe()` не осуществляет оптимизацию модели, включающей номинальные предикторы, поэтому их необходимо преобразовать в индикаторные переменные с использованием функции `model.matrix()`. После этого в новой таблице `x` вместо каждого из факторов образуется несколько бинарных (0/1) столбцов с наименованиями, соответствующими уровням этого фактора:

```
x <- model.matrix(a1 ~ ., data = algae[, 1:12])[, -1]
set.seed(10)
ctrl <- rfeControl(functions = lmFuncs, method = "cv",
       verbose = FALSE, returnResamp = "final")
lmProfileF <- rfe(as.data.frame(x), algae$a1,
       sizes = 1:10, rfeControl = ctrl)
```

```
Recursive feature selection
Outer resampling method: Cross-Validated (10 fold)
Resampling performance over subset size:
 Variables  RMSE Rsquared RMSESD RsquaredSD Selected
         1 21.99   0.1375  6.209    0.06023
         2 20.59   0.2093  6.056    0.14480
         3 19.41   0.2973  6.393    0.17497
         4 19.50   0.2956  6.460    0.16444
         5 18.55   0.2949  4.154    0.14745
         6 18.52   0.3071  4.250    0.12319
         7 18.29   0.2863  3.320    0.10532
         8 18.18   0.3004  3.386    0.10780
         9 17.91   0.3168  3.410    0.11256        *
        10 17.93   0.3160  3.378    0.11117
        15 18.06   0.3073  3.166    0.10928
```

```
ggplot(lmProfileF, metric = "Rsquared")
predictors(lmProfileF)
summary(lmProfileF$fit)
```

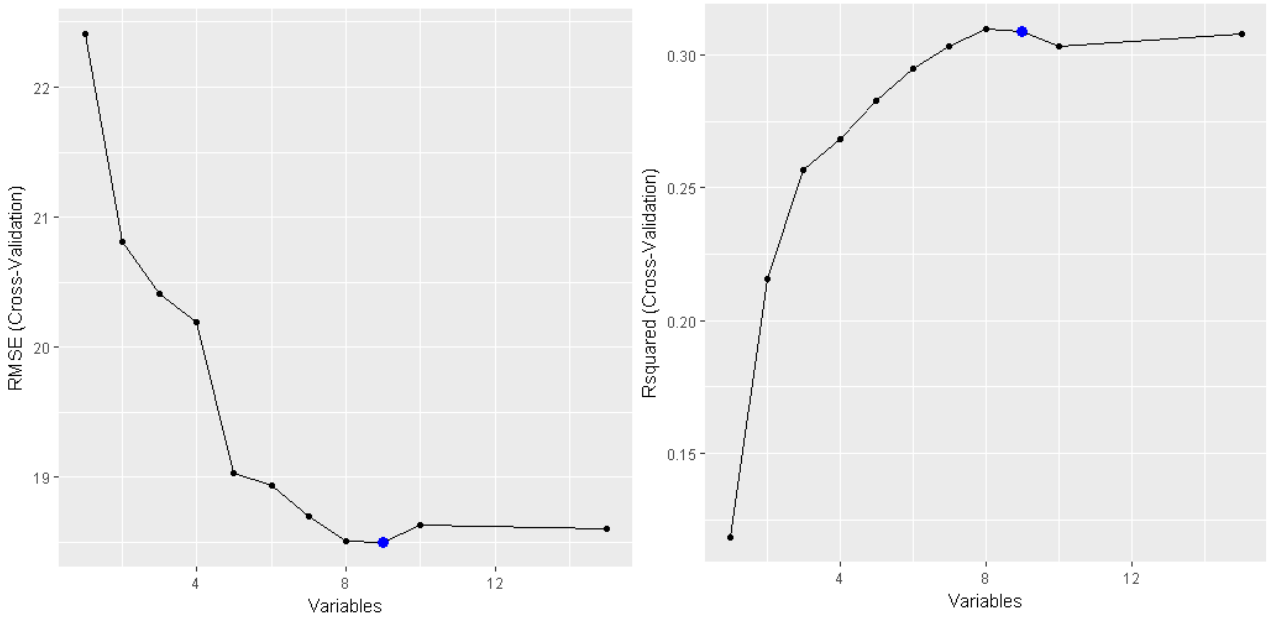


Рис. 4.2. Изменение корня из среднеквадратичной ошибки и коэффициента детерминации в зависимости от числа предикторов (по результатам перекрестной проверки)

```
[1] "NO3"   "sizesmall"     "PO4"   "NH4"   "mnO2"   "mxPH"
[7] "Cl"    "seasonwinter"  "sizemedium"

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)  49.8149919 21.4587614   2.321  0.02132 *
NO3          -1.6308558  0.5347832  -3.050  0.00262 **
sizesmall     9.7013042  3.8299749   2.533  0.01212 *
PO4          -0.0567740  0.0123847  -4.584 8.25e-06 ***
NH4           0.0016048  0.0009745   1.647  0.10124
mnO2          0.8186113  0.6360488   1.287  0.19965
mxPH         -3.9777321  2.4624003  -1.615  0.10789
seasonwinter  1.8598837  2.7183942   0.684  0.49469
```

```
sizemedium    3.2637287  3.3692541   0.969  0.33394
Cl           -0.0354895  0.0324232  -1.095  0.27509
---
Signif. codes:  0 C***T 0.001 C**T 0.01 C*T 0.05 C.T 0.1 C T 1

Residual standard error: 17.44 on 190 degrees of freedom
Multiple R-squared:  0.3627,    Adjusted R-squared:  0.3325
F-statistic: 12.02 on 9 and 190 DF,  p-value: 6.177e-15
```

Нельзя утверждать, что нам удалось достигнуть серьезных успехов, применив метод RFE: пошаговая модель `lm_step.a1` была компактнее и чуть лучше (по результатам перекрестной проверки), хотя и уступала модели RFE по величине ошибки на обучающей выборке.

Генетический алгоритм

Генетический алгоритм, позаимствованный у природных аналогов и разработанный Дж. Холландом (Holland, 1975), отличается от большинства иных процедур селекции тем, что поиск оптимального решения развивается не сам по себе, а с учетом предыдущего опыта. Смысл его заключается в следующем:

- формируемое решение кодируется как вектор x^0, который называется хромосомой и соответствует битовой маске, т.е. двоичному представлению набора исходных переменных;
- инициализируется исходная "популяция" $\Pi^0(x_1^0 \ldots x_\lambda^0)$ потенциальных решений, состоящая из некоторого количества хромосом λ;
- каждой хромосоме в популяции присваиваются две оценки: значение эффективности $\mu(x_i^0)$ в соответствии с заданной функцией оптимальности и вероятность воспроизведения $P(x_i^0)$, которая зависит от "перспективности" этой хромосомы;
- в соответствии с вероятностями воспроизведения хромосомы производят потомков, используя операции кроссинговера (обмена фрагментами между хромосомами) и мутации (замена значения бита 0/1 на противоположный) – см. рис. 4.3;
- в ходе репродуцирования создается новая популяция хромосом, причем с большей вероятностью воспроизводятся наиболее перспективные фрагменты "генетического кода";
- формирование новой популяции многократно повторяется и осуществляется поиск субоптимальных моделей;
- процесс останавливается, если получено удовлетворительное решение, либо исчерпано все отведенное на эволюцию время.

Синтаксис пары функций `gafs()` и `gafsControl()`, реализующих генетический алгоритм в пакете `caret`, в целом аналогичен таковому у функций `rfe()` и `rfeControl()`, но требует задания дополнительных аргументов, регулирующих скорость эволюции. Похоже, что здесь преобразовывать факторы в индикаторные переменные не требуется (эта проблема нами не исследовалась):

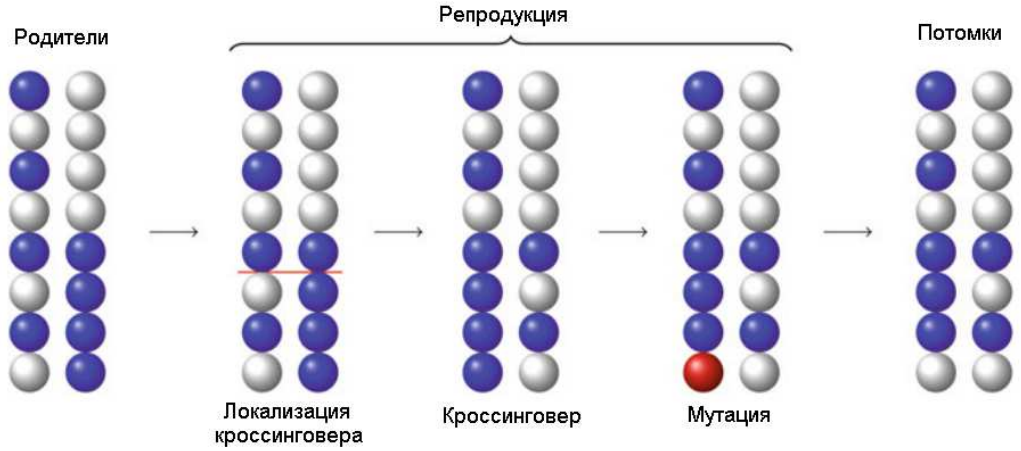


Рис. 4.3. Схема кроссинговера и мутации во время применения генетического алгоритма

```
set.seed(10)
ctrl <- gafsControl(functions = rfGA, method = "cv",
       verbose = FALSE, returnResamp = "final")
lmProfGafs <- gafs(algae[, 1:11], algae$a1,
    iters = 10, # 10 генераций
    gafsControl = ctrl)
lmProfGafs
```

```
Genetic Algorithm Feature Selection
Maximum generations: 10
Population per generation: 50
Crossover probability: 0.8
Mutation probability: 0.1
Elitism: 0
Internal performance values: RMSE, Rsquared
Subset selection driven to minimize internal RMSE
External performance values: RMSE, Rsquared
Best iteration chose by minimizing external RMSE
External resampling method: Cross-Validated (10 fold)
During resampling:
  * the top 5 selected variables (out of a possible 11):
    Cl (100%), PO4 (100%), size (100%), mxPH (50%), oPO4 (40%)
  * on average, 5.2 variables were selected (min = 3, max = 7)
In the final search using the entire training set:
   * 4 features selected at iteration 1 including:
     mxPH, Cl, NO3, PO4
   * external performance at this iteration is
       RMSE    Rsquared
     15.381       0.519
```

```
lm_gafs.a1 <- lm(a1 ~ size + mxPH + Cl + NO3 + PO4,
                 data = algae[, 1:12])
train(a1 ~ size + mxPH + Cl + NO3 + PO4,
      data = algae[, 1:12], method = 'lm',
trControl = trainControl(method = "cv"))
```

```
   RMSE      Rsquared   RMSE SD   Rsquared SD
  17.72904  0.3430528  2.838075  0.09572028
```

Несмотря на большие затраты вычислительных ресурсов (поиск решения продолжался более 30 мин.), была получена комбинация предикторов, превосходящая по критериям RMSE и Rsquared все три предыдущие модели.

Тестирование моделей с использованием дополнительного набора данных

Возникает естественный вопрос: а какой из четырех полученных моделей следует все же отдать предпочтение при прогнозировании? Хорошую возможность ответить на него предоставляет нам Л. Торго (Torgo, 2011), подготовивший на сайте своей книги (http://www.dcc.fc.up.pt) специально предназначенный для этого набор данных из 140 наблюдений (см. файл Eval.txt с предикторами и Sols.txt со значениями отклика). Пропущенные значения заполним с использованием алгоритма бэггинга:

```
Eval <- read.table('Eval.txt', header=F, dec='.',
   col.names=c('season','size','speed','mxPH','mnO2','Cl',
  'NO3','NH4','oPO4','PO4','Chla'), na.strings=c('XXXXXXX'))
Sols <- read.table('Sols.txt', header=F, dec='.',
          col.names=c('a1','a2','a3','a4','a5','a6','a7'),
          na.strings=c('XXXXXXX'))
ImpEval <- preProcess(Eval[, 4:11], method='bagImpute')
Eval[,4:11] <-  predict(ImpEval, Eval[,4:11])
# Сохраним данные для дальнейшего использования
save(Eval,Sols, file="algae_test.RData")
y <- Sols$a1
EvalF <- as.data.frame(model.matrix(y ~ .,Eval)[,-1])
```

Выполним прогноз для набора предикторов тестовой выборки и оценим точность каждой модели по трем показателям: среднему абсолютному отклонению (MAE), корню из среднеквадратичного отклонения (RSME) и квадрату коэффициента детерминации Rsq = 1 - NSME, где NSME – относительная ошибка, равная отношению средних квадратов отклонений от регрессии и от общего среднего:

```
# Функция, выводящая вектор критериев
ModCrit <- function (pred, fact) {
  mae <- mean(abs(pred-fact))
  rmse <- sqrt(mean((pred-fact)^2))
  Rsq <- 1-sum((fact-pred)^2)/sum((mean(fact)-fact)^2)
  c(MAE=mae, RSME=rmse,  Rsq = Rsq )
}
y <- Sols$a1
EvalF <- as.data.frame(model.matrix(y ~ .,Eval)[,-1])
Result <- rbind(
   lm_full = ModCrit(predict(lm.a1,Eval),Sols[,1]),
   lm_step = ModCrit(predict(lm_step.a1,Eval),Sols[,1]),
   lm_rfe = ModCrit(predict(lmProfileF$fit,EvalF),Sols[,1]),
   lm_gafs = ModCrit(predict(lm_gafs.a1,Eval),Sols[,1]))
Result
```

```
             MAE     RSME      Rsq
lm_full 12.64484 16.80890 0.3268512
lm_step 12.47868 16.62016 0.3418829
lm_rfe  12.38730 16.52863 0.3491115
lm_gafs 12.74591 17.19192 0.2958234
```

Вероятно, нет смысла делать серьезные выводы из того, что рейтинг лучших моделей при перекрестной проверке и при независимом тестировании на объектах, не участвовавших в построении моделей, оказался столь несовпадающим. Во-первых, разброс критериев точности моделей находится в пределах доверительных интервалов, поэтому их ранжирование, строго говоря, можно трактовать как обусловленное случайными причинами. Во-вторых, многое зависит от того, например, насколько неоднородны были между собой обе выборки. И, наконец, напомним, что одним из основных принципов моделирования сложных систем является *принцип множественности моделей*, сформулированный В. В. Налимовым и заключающийся в возможности представления одной и той же системы множеством различных моделей в зависимости от целей исследования.

4.2. Регуляризация, частные наименьшие квадраты и kNN-регрессия

Регрессия по методу "лассо"

Регрессия по методу наименьших квадратов (МНК) часто может стать неустойчивой, то есть сильно зависящей от обучающих данных, что обычно является проявлением тенденции к переобучению. Избежать такого переобучения помогает *регуляризация* – общий метод, заключающийся в наложении дополнительных ограничений на искомые параметры, которые могут предотвратить излишнюю сложность модели. Смысл процедуры заключается в "стягивании" вектора коэффициентов β в ходе их настройки таким образом, чтобы они в среднем оказались несколько меньше по абсолютной величине, чем это было бы при оптимизации по МНК.

Метод регрессии "*лассо*" (LASSO, Least Absolute Shrinkage and Selection Operator) заключается во введении дополнительного слагаемого регуляризации в функционал оптимизации модели, что часто позволяет получать более устойчивое решение. Условие минимизации квадратов ошибки при оценке параметров β выражается следующей формулой:

$$\beta = \arg\min[\sum_{i=1}^{n}(y_i - \sum_{j=1}^{m}\beta_j x_{ij})^2 + \lambda|\boldsymbol{\beta}|],$$

где λ – параметр регуляризации, имеющий смысл штрафа за сложность.

При этом достигается некоторый компромисс между ошибкой регрессии и размерностью используемого признакового пространства, выраженного суммой абсолютных значений коэффициентов $|\beta|$. В ходе минимизации некоторые коэффициенты становятся равными нулю, что, собственно, и определяет отбор информативных признаков.

При значении параметра регуляризации $\lambda = 0$, лассо-регрессия сводится к обычному методу наименьших квадратов, а при увеличении λ формируемая модель становится все более "лаконичной", пока не превратится в нуль-модель. Оптимальная величина λ находится с использованием перекрестной проверки, т.е. ей соответствует минимальная ошибка прогноза $\hat{y}_i$ на наблюдениях, не участвовавших в построении самой модели.

Обобщением регрессии с регуляризацией можно считать модель "*эластичных сетей*" (elastic net – Zou, Hastie, 2005). Эта модель устанавливает сразу два типа штрафных параметров λ_1 и λ_2, объединяет гребневую регрессию (при $\lambda_2 = 0$) и регрессию "лассо" (при $\lambda_1 = 0$):

$$\hat{\beta} = \arg\min[\sum_{i=1}^{n}(y_i - \sum_{j=1}^{m}\beta_j x_{ij})^2 + \lambda_1(\boldsymbol{\beta})^2 + \lambda_2 |\boldsymbol{\beta}|].$$

Продолжим рассмотрение примеров построения регрессионных моделей, прогнозирующих обилие водорослей группы `a1` в зависимости от гидрохимических показателей воды и условий отбора проб в различных водотоках (см. подробное описание таблицы переменных в разделе 3.4).

Поскольку даже ориентировочная величина параметра регуляризации нам неизвестна, то на первом этапе с использованием функции `glmnet()` из одноименного пакета проведем анализ изменения значений коэффициентов в зависимости от λ в ее широком диапазоне от 0.1 до 1000:

```
load(file="algae.RData") # Загрузка таблицы algae - раздел 4.1
grid=10^seq(10,-2,length=100)
library( glmnet)
lasso.a1 <- glmnet(x,algae$a1, alpha=1, lambda=grid)
plot(lasso.a1, xvar =  "lambda", label = TRUE, lwd=2)
```

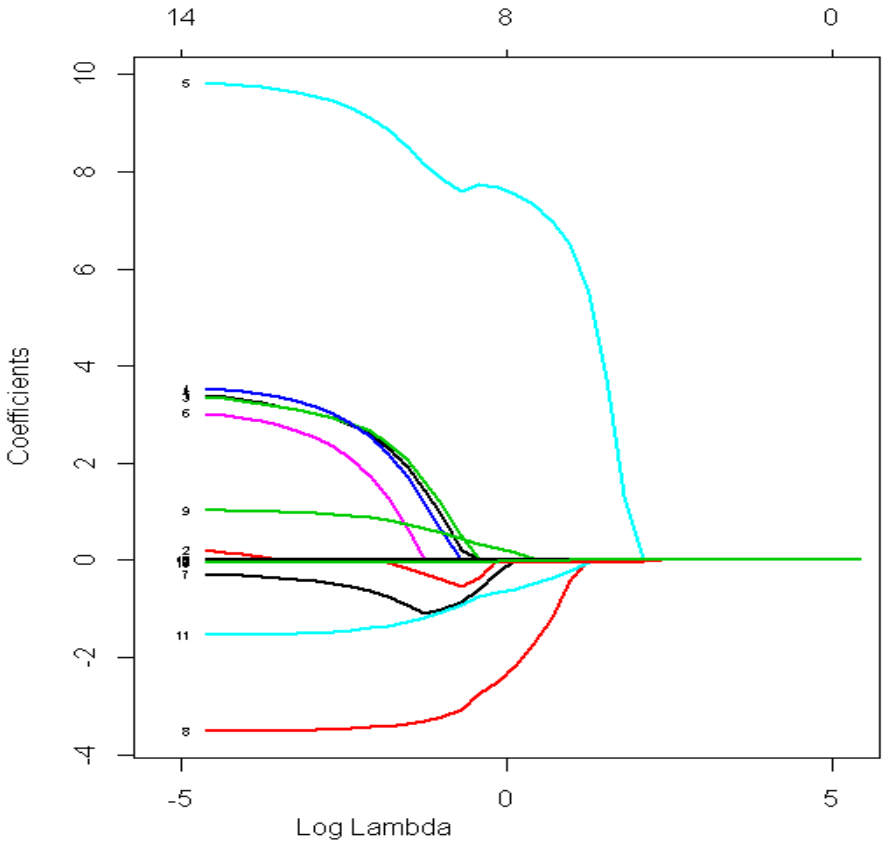


Рис. 4.4. Зависимость коэффициентов регрессионной модели от значения параметра регуляризации

Функция `train()` для метода `glmnet` выполняет оптимизацию сразу двух моделей регуляризации: при `alpha` = 1 подгоняется модель по методу

лассо, а при alpha = 0 – гребневая регрессия. Примем лассо-модель, и выполним тонкую настройку значения lambda в интервале от 0.5 до 4.5:

```
x <- model.matrix(a1 ~ .,data=algae[,1:12])[,-1]
grid.train = seq(0.5,4.5,length=15)
lasso.a1.train <- train(as.data.frame(x), algae$a1,
    method='glmnet',
    tuneGrid = expand.grid(.lambda = grid.train, .alpha = 1),
    trControl = trainControl(method = "cv"))
```

```
Resampling results across tuning parameters:
  lambda     RMSE      Rsquared   RMSE SD   Rsquared SD
  0.5000000  17.86789  0.3122892  3.148867  0.11004143
  0.7857143  17.70163  0.3226125  3.119873  0.10317362
  1.0714286  17.60777  0.3300379  3.064511  0.09865072
  1.3571429  17.58081  0.3339868  3.010830  0.09807188
  1.6428571  17.58690  0.3368441  2.974649  0.09840106
  1.9285714  17.61382  0.3394520  2.945534  0.09889217
  2.2142857  17.66452  0.3408752  2.914110  0.09916679
  2.5000000  17.73367  0.3414334  2.892287  0.09909826
  2.7857143  17.81998  0.3409785  2.889505  0.09957978
…
  4.5000000  18.49634  0.3249896  2.953191  0.10310683
Tuning parameter 'alpha' was held constant at a value of 1
RMSE was used to select optimal model using smallest value.
The final values used for model were alpha = 1 and lambda =
1.357143.
```

Выведем протокол со значениями коэффициентов модели:

```
coef(lasso.a1.train$finalModel,
                  lasso.a1.train$bestTune$lambda)
```

```
16 x 1 sparse Matrix of class "dgCMatrix"
                        1
(Intercept)  38.37192105
seasonspring  .
seasonsummer  .
seasonwinter  .
sizemedium    .
sizesmall     7.42569282
speedlow      .
speedmedium   .
mxPH         -1.78557347
mnO2          0.11517464
Cl           -0.03960609
NO3          -0.51584616
NH4           .
oPO4          .
PO4          -0.05193156
Chla         -0.02286621
```

Коэффициенты регрессии, отличные от 0, составляют информативный набор предикторов.

Метод частных наименьших квадратов (PLS)

В разделе 3.3 мы показали как на основе исходных переменных с помощью функции preProcess() можно сформировать новое пространство главных компонент и построить модель классификации, прогнозирующую

уровень доверия банка к клиенту. Напомним основные шаги построения модели регрессии на главные компоненты (PCR):

- стандартизация матрицы исходных предикторов **X** и отклика **Y**;
- выбор числа собственных значений p и формирование матрицы главных компонент (часто называемой таблицей *счетов* – scores)

$$\mathbf{T}_{n\times r} = \mathbf{X}_{n\times m}\,\mathbf{P}^{\mathrm{T}}{}_{m\times p}, \qquad \text{где } \mathbf{P} \text{ – матрица } \textit{нагрузок} \text{ (loadings);}$$

- оценка вектора коэффициентов **A** множественной регрессии на главные компоненты: $\mathbf{Y} = \mathbf{TA}$, $\mathbf{A}_r = (\mathbf{T}^{\mathrm{T}}\mathbf{T})^{-1}\mathbf{T}^{\mathrm{T}}\mathbf{Y}$;
- пересчет коэффициентов регрессии на главные компоненты обратно в коэффициенты регрессии для исходных предикторов: $\mathbf{B}_m = \mathbf{PA}$.

Однако такой двухступенчатый подход (сжатие информативного пространства и последующая регрессия) – не самый обоснованный способ построения предсказательных моделей, поскольку PCA-преобразование данных не всегда приводит к новым предикторам, наилучшим образом объясняющим отклик.

Метод *частных наименьших квадратов* PLS (Partial Least Squares, или Projection into Latent Structure) также использует разложение исходных предикторов по осям главных компонент, но дополнительно выделяет подмножество латентных переменных, в пространстве которых связь между зависимой переменной и предикторами достигает максимального значения.

В случае одномерного отклика сначала оценивается корреляционная связь между предикторами **X** и **Y**, осуществляется сингулярное разложение матрицы $\mathbf{X}^{\mathrm{T}}\mathbf{Y}$, и формируется вектор **W** первого направления PLS (direction), при вычислении которого особое внимание уделяется переменным, которые наиболее тесно связаны с откликом. Исходные переменные **X** ортогонально проецируются на ось, задаваемую вектором **W**, а затем последовательно рассчитываются значения **T** счетов и нагрузок **P**.

Для нахождения второго направления PLS вычисляются остатки, которые остались необъясненными первым PLS-направлением. Оценивается направление новой оси наибольшей корреляции и оцениваются значения счетов с использованием ортогонализированных остатков. Такой итеративный подход можно применить p раз пока модель не достигнет оптимальной сложности.

Реализация PLS-регрессии в среде R представлена в пакете `pls`, который включает в себя большое количество функций для построения регрессионных моделей, создания диагностических графиков и извлечения информации из результатов вычислений:

```
library(pls)
M.pls <- plsr(algae$a1 ~ x, scale = TRUE,
              Validation = "CV", method = "oscorespls")
summary(M.pls)
```

```
Data:   X dimension: 200 15
        Y dimension: 200 1
Fit method: oscorespls
Number of components considered: 15
```

```
TRAINING: % variance explained
        1 comps 2 comps 3 comps 4 comps 5 comps ... 13 comps 14 comps 15 comps
X         22.12   30.35   36.23   43.13   50.55        94.76    97.95   100.00
algae$a1  33.43   35.03   36.21   36.70   36.81        36.93    36.93    36.93
```

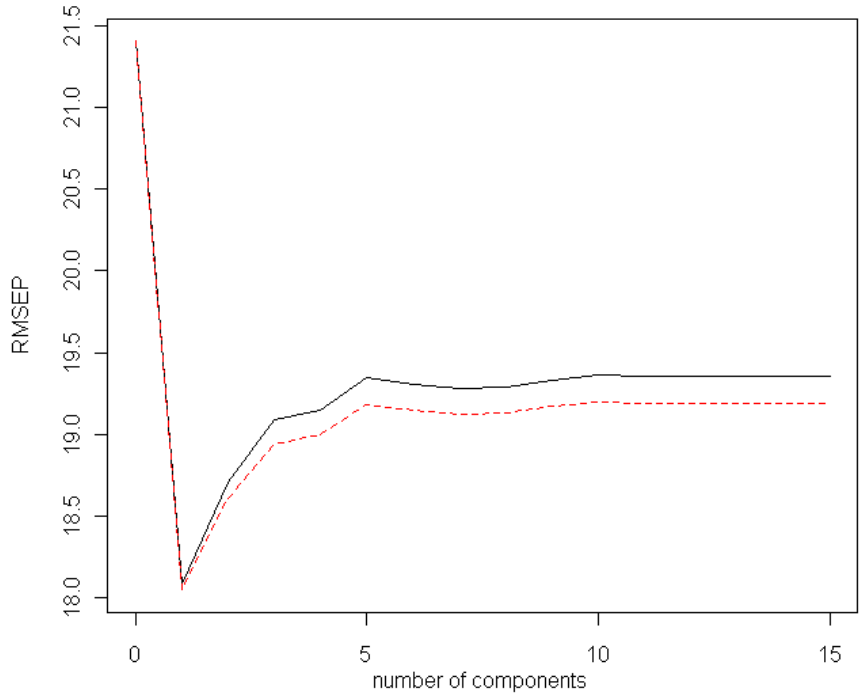


Рис. 4.5. Зависимость ошибки перекрестной проверки от числа направлений PLS

Оптимальная модель в нашем случае основывается только на 1-м направлении PLS, поскольку при включении новых осей проецирования ошибка перекрестной проверки монотонно возрастает – см. рис. 4.5. Заметим, что накопленная доля объясненной дисперсии для матрицы предикторов X равномерно возрастает при увеличении числа компонент, тогда как почти вся вариация отклика `algae$a1` сконцентрирована вдоль главного направления.

Выполним подбор размерности пространства латентных переменных с использованием функции `train()`. По умолчанию диапазон значений оптимизируемого параметра `ncomp` принимает значения от 1 до `tuneLength`:

```
set.seed(100)
ctrl <- trainControl(method = "cv", number = 10)
(plsTune.a1 <- train(x, algae$a1, method = "pls",
                     tuneLength = 14, trControl = ctrl,
                     preProc = c("center", "scale")))
```

```
Resampling: Cross-Validation (10 fold)
  ncomp  RMSE  Rsquared  RMSE SD  Rsquared SD
  1      17.8  0.349     3.97     0.167
  2      18.6  0.299     4.01     0.151
  3      18.9  0.287     4.27     0.158
. . .
  13     18.8  0.3       4.29     0.163
  14     18.8  0.3       4.29     0.163
RMSE was used to select optimal model using smallest value.
The final value used for the model was ncomp = 1.
```

Выполним для сравнения подбор числа компонент для модели регрессии на главные компоненты PCR:

```
set.seed(100)
ctrl <- trainControl(method = "cv", number = 10)
(pcrTune.a1 <- train(x, algae$a1, method = "pcr",
         tuneLength = 14, trControl = ctrl,
         preProc = c("center", "scale")))
```

```
  ncomp  RMSE  Rsquared  RMSE SD  Rsquared SD
  1      17.5  0.358     3.7      0.173
  2      17.6  0.352     3.59     0.169
  3      17.6  0.355     3.68     0.165
. . .
  13     18.4  0.321     4.34     0.162
  14     18.5  0.324     4.63     0.157
The final value used for the model was ncomp = 1.
```

Никаких преимуществ метода PLS по сравнению с PCR на нашем примере обнаружено не было; более того, ошибка в целом модели на главные компоненты оказалась несколько ниже.

Регрессия по методу k ближайших соседей

Ключевая идея, лежащая в основе метода *k* ближайших соседей, как уже обсуждалось в разделе 3.4, состоит в формулировке модели в терминах евклидовых расстояний в исходном многомерном пространстве признаков. Задача заключается в том, чтобы для каждой тестируемой точки $\boldsymbol{x}_0$ найти такую δ-окрестность (многомерный эллипсоид), чтобы в ней поместилось ровно *k* точек с известными значениями *y*. Тогда прогноз $f(\boldsymbol{x}_0)$ можно получить, усредняя значения отклика всех обучающих наблюдений из δ.

Функция `train()` оптимизирует число соседей `k`, используя другую функцию – `knnreg()` из пакета `caret` (рис. 4.6):

```
knnTune.a1 <- train(x, algae$a1, method = "knn",
         preProc = c("center", "scale"),
         trControl = ctrl, tuneGrid = data.frame(.k = 4:25))
plot( knnTune.a1)
```

```
Resampling results across tuning parameters:
  k   RMSE  Rsquared  RMSE SD  Rsquared SD
  4   18.5  0.307     3.25     0.157
  5   18.2  0.322     3.64     0.179
  6   18.2  0.313     3.57     0.176
  7   17.6  0.343     3.74     0.166
. . .
  18  17.1  0.395     3.59     0.115
  19  16.9  0.41      3.69     0.116
  20  16.9  0.413     3.74     0.121
  21  16.8  0.42      3.52     0.119
  22  16.9  0.418     3.48     0.113
  23  17    0.409     3.62     0.117
  24  17.1  0.407     3.54     0.118
  25  17.2  0.397     3.57     0.104

RMSE was used to select optimal model using smallest value.
The final value used for the model was k = 21.
```

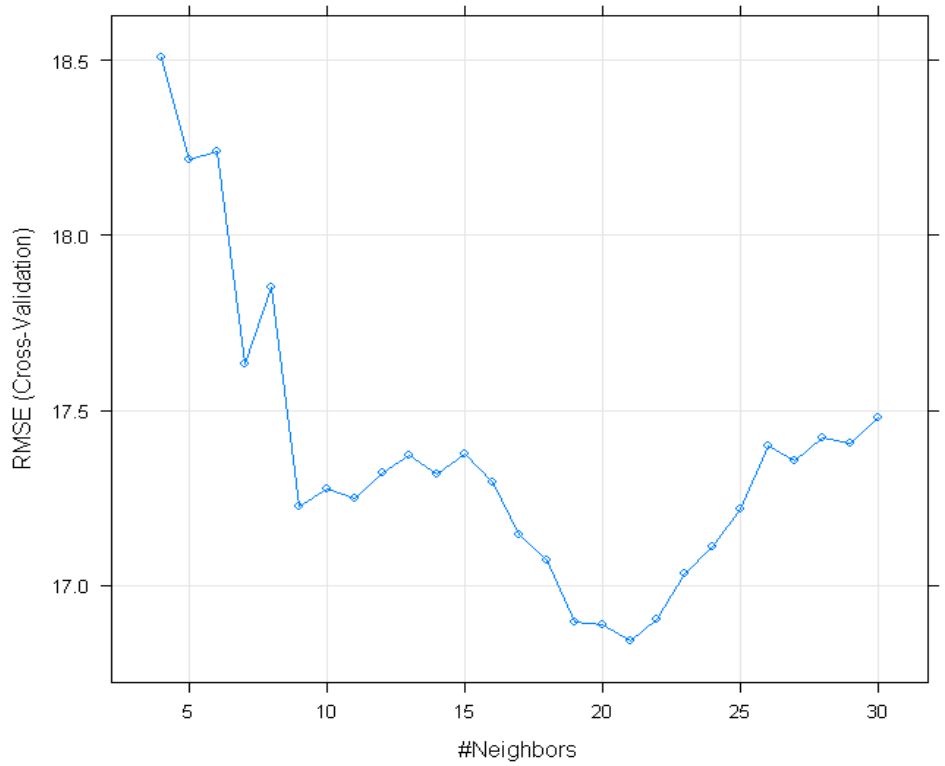


Рис. 4.6. Зависимость ошибки перекрестной проверки
от числа ближайших соседей

Тестирование моделей с использованием дополнительного набора данных

Используем для прогноза набор данных (Torgo, 2011) из 140 наблюдений, который мы уже применяли в предыдущем разделе. Данные, подготовленные для тестирования с восстановленными пропущенными значениями – таблицы `Eval` с предикторами и `Sols` со значениями отклика – мы сохранили в файле `algae_test.RData` (см раздел 4.1).

Выполним прогноз для набора предикторов проверочной выборки и оценим точность каждой модели по трем показателям: среднему абсолютному отклонению (`MAE`), корню из среднеквадратичного отклонения (`RSME`) и квадрату коэффициента детерминации `Rsq` = 1 - `NSME`, где `NSME` – относительная ошибка, равная отношению среднего квадрата отклонений от модельных значений и от общего среднего:

```
load(file="algae_test.RData") # Загрузка таблиц Eval, Sols
y <- Sols$a1
EvalF <- as.data.frame(model.matrix(y ~ .,Eval)[,-1])
# Функция, выводящая вектор критериев
ModCrit <- function (pred, fact) {
  mae <- mean(abs(pred-fact))
  rmse <- sqrt(mean((pred-fact)^2))
  Rsq <- 1-sum((fact-pred)^2)/sum((mean(fact)-fact)^2)
  c(MAE=mae, RSME=rmse,  Rsq = Rsq ) }
Result <- rbind(
  lasso = ModCrit(predict(lasso.a1.train,EvalF),Sols[,1]),
 pls_1c = ModCrit(predict(plsTune.a1, EvalF),Sols[,1]),
 pcr_1c = ModCrit(predict(pcrTune.a1, EvalF),Sols[,1]),
 kNN_21 = ModCrit(predict(knnTune.a1, EvalF),Sols[,1]))
Result
```

```
             MAE     RSME      Rsq
lasso  12.74513 17.13765 0.3002625
pls_1c 12.76695 17.27928 0.2886493
pcr_1c 12.85117 17.29990 0.2869504
kNN_21 12.21037 16.30965 0.3662442
```

Отметим, что регрессия на главные компоненты и PLS в условиях слабой мультиколлинеарности данных не приносит никаких ощутимых преимуществ по сравнению с классическими линейными моделями раздела 4.1. Модель по методу лассо, отлично зарекомендовавшая себя при перекрестной проверке, сработала хуже на свежих данных, что связано, видимо, с дрейфом параметра регуляризации λ от своего оптимального значения. И, наконец, пока лучшей из всех исследованных оказалась методически простейшая модель регрессии по 21 ближайшему соседу. В то же время эта непараметрическая модель имеет важнейший недостаток – она не предоставляет никакой информации для графической или содержательной интерпретации.

4.3. Построение деревьев регрессии

Деревья решений (Breiman at al., 1984; Quinlan, 1986) осуществляют разбиение пространства объектов в соответствии с некоторым набором *правил разбиения* (splitting rule). Эти правила являются логическими утверждениями в отношении той или иной переменной и могут быть истинными или ложными. Ключевыми здесь являются три обстоятельства: а) правила позволяют реализовать последовательную дихотомическую сегментацию данных, б) два объекта считаются похожими, если они оказываются в одном и том же сегменте разбиения, в) на каждом шаге разбиения увеличивается количество информации относительно исследуемой переменной (отклика).

Деревья классификации и регрессии являются одним из наиболее популярных методов решения многих практических задач, что обусловлено следующими причинами:

1. Деревья решений позволяют получать очень *легко интерпретируемые модели*, представляющие собой набор правил вида "если..., то...". Интерпретация облегчается, в том числе, за счет возможности представить эти правила в виде наглядной *древовидной структуры*.
2. В силу своего устройства деревья решений позволяют работать с *переменными любого типа* без необходимости какой-либо предварительной подготовки этих переменных для ввода в модель (например, логарифмирование, преобразование категориальных переменных в индикаторные и т.п.).
3. Исследователю *нет необходимости в явном виде задавать форму взаимосвязи* между откликом и предикторами, как это, например, происходит в случае с обычными регрессионными моделями. Это оказывается особенно полезным при работе с данными большого объема, о свойствах которых мало что известно.

4. Деревья решений, по сути, *автоматически выполняют отбор информативных предикторов* и учитывают возможные взаимодействия между ними. Это, в частности, делает деревья решений полезным инструментом разведочного анализа данных.
5. Деревья решений *можно эффективно применять к данным с пропущенными значениями*, что очень полезно при решении практических задач, где наличие пропущенных значений – это, скорее, правило, чем исключение.
6. Деревья решений одинаково хорошо применимы *как к количественным, так и к качественным зависимым переменным*.

К недостаткам этого класса моделей иногда относят *нестабильность* и *невысокую точность предсказаний*, что, как будет показано ниже, не всегда подтверждается. По своей сути, деревья используют "наивный подход" (naive approach) в том смысле, что они исходят из предположения о взаимной независимости признаков. Поэтому модели регрессионных деревьев статистически наиболее работоспособны, когда комплекс анализируемых переменных является не слишком мультиколлинеарным или имеется регулярная внутренняя множественная альтернатива в исходной комбинации признаков.

Алгоритм CART (Classification and Regression Tree) рекурсивно делит исходный набор данных на подмножества, которые становятся все более и более гомогенными относительно определенных признаков, в результате чего формируется древовидная иерархическая структура. Деление осуществляется на основе традиционных логических правил в виде ЕСЛИ (А) ТО (В), где А – некоторое логическое условие, а В – процедура деления подмножества на две части, для одной из которых условие А истинно, а для другой – ложно. Примеры условий: $X_i==F$, $X_i <= V$; $X_i >= V$ и др., где X_i – один из предикторов исходной таблицы, F – выбранное значение категориальной переменной, V – специально подобранное опорное значение (порог).

На первой итерации корневой узел дерева связывается с наиболее оптимальным условным суждением, и все множество объектов делится на две группы. От каждого последующего узла-родителя к узлам-потомкам также может отходить по две ветви, в свою очередь связанные с граничными значениями других наиболее подходящих переменных и определяющие правила дальнейшего разделения (splitting criterion). Конечными узлами дерева являются "листья", соответствующие найденным решениям и объединяющие все разделенные на группы объекты обучающей выборки. Общее правило выбора опорного значения для каждого узла построенного дерева можно сформулировать следующим образом: «выбранный признак должен разбить множество $\mathbf{X^*}$ так, чтобы получаемые в итоге подмножества $\mathbf{X^*_k}$, $k = 1, 2, \ldots, p$, состояли из объектов, принадлежащих к одному классу, или были максимально приближены к этому».

Описанный процесс относится к так называемым "жадным" алгоритмам, стремящимся, не считаясь ни с чем, построить максимально

"кустистое" дерево (также "глубокое дерево", deep tree). Естественно, чем обширнее и кустистее дерево, тем лучше будут результаты его тестирования на обучающей выборке, но не столь успешными – на проверочной выборке. Поэтому построенная модель должна быть еще и оптимальной по размерам, т.е. содержать информацию, улучшающую качество распознавания, и игнорировать ту информацию, которая его не улучшает. Для этого обычно проводят "обрезание" дерева (tree pruning) − отсечение ветвей там, где эта процедура не приводит к серьезному возрастанию ошибки.

Невозможно подобрать объективный внутренний критерий, приводящий к хорошему компромиссу между безошибочностью и компактностью, поэтому стандартный механизм оптимизации деревьев основан на перекрестной проверке (Loh, Shih, 1997). Для этого обучающая выборка разделяется, например, на 10 равных частей: 9 частей используется для построения дерева, а оставшаяся часть играет роль проверочной совокупности. После многократного повторения этой процедуры из некоторого набора деревьев-претендентов, у которых имеется практически допустимый разброс критериев качества модели, выбирается дерево, показавшее наилучший результат при перекрестной проверке.

Построение деревьев на основе рекурсивного разбиения

В общем случае может быть использовано несколько алгоритмов построения деревьев на основе различных схем и критериев оптимизации. Функция `rpart()` из одноименного пакета выполняет рекурсивный выбор для каждого следующего узла таких разделяющих значений, которые приводят к минимальной сумме квадратов внутригрупповых отклонений D_t для всех t узлов дерева. Для оценки качества построенного дерева **T** в ходе его оптимизации используется следующая совокупность критериев:

- штраф за сложность модели (cost complexity), включающий штрафной множитель λ за каждую неотсечённую ветвь $CC(\mathbf{T}) = \sum_t D_t + \lambda t$;
- девианс D_0 для нулевого дерева (т.е. оценка изменчивости в исходных данных);
- относительный штраф за сложность модели $C_p = \lambda / D_0$;
- относительная ошибка обучения для дерева из t узлов $REL_{er} = \sum_t D_t / D_0$;
- ошибка перекрестной проверки (CV_{er}) с разбиением на k блоков (обычно $k = 10$), также отнесенная к девиансу нуль-дерева D_0; CV_{er}, как правило, больше, чем REL_{er};
- стандартное отклонение (SE) ошибки перекрестной проверки.

Лучшим считается дерево, состоящее из такого количества ветвей t, для которого сумма ($CV_{er} + SE$) является минимальной.

В качестве примера рассмотрим построение дерева CART, прогнозирующего обилие водорослей группы `a1` в зависимости от гидрохимических показателей воды и условий отбора проб в различных водотоках (см. разделы 3.4 и 4.1-4.2). Используем сначала пакет `rpart`, для

работы с которым обычно применяется двухшаговая процедура: функция `rpart()` устанавливает связи между зависимой и независимыми переменными и формирует бинарное дерево, а функция `prun()` выполняет обрезание лишних ветвей.

```
load(file="algae.RData") # Загрузка таблицы algae - раздел 4.1
(rt.a1 <- rpart::rpart(a1 ~ ., data = algae[, 1:12]))
```

```
n= 200
node), split, n, deviance, yval
      * denotes terminal node
  1) root 200 90694.880 16.923500
    2) PO4>=43.818 148 31359.210  8.918919
      4) Cl>=7.8065 141 21678.580  7.439716
        8) oPO4>=51.118 85  3455.770  3.801176 *
        9) oPO4< 51.118 56 15389.430 12.962500
         18) mnO2>=10.05 24  1248.673  6.716667 *
         19) mnO2< 10.05 32 12502.320 17.646880
           38) NO3>=3.1875 9   257.080  7.866667 *
           39) NO3< 3.1875 23 11047.500 21.473910
             78) mnO2< 8 13  2919.549 13.807690 *
             79) mnO2>=8 10  6370.704 31.440000 *
      5) Cl< 7.8065 7  3157.769 38.714290 *
    3) PO4< 43.818 52 22863.170 39.705770
      6) mxPH< 7.87 28 11636.710 32.875000
       12) oPO4>=3.1665 14  1408.304 23.978570 *
       13) oPO4< 3.1665 14  8012.309 41.771430 *
      7) mxPH>=7.87 24  8395.785 47.675000
       14) PO4>=15.177 12  3047.517 38.183330 *
       15) PO4< 15.177 12  3186.067 57.166670 *
```

Приведенной командой мы построили полное дерево без обрезания ветвей, состоящее из 9 узлов и 10 листьев, обозначенных в приведенном протоколе разбиения символом *. В каждой строке представлены по порядку: условие разделения, число наблюдений, соответствующих этому условию, девианс (в данном случае – это эквивалент суммы квадратов отклонений от группового среднего) и среднее значение отклика для выделенной ветви. Например, перед первой итерацией общее множество из 200 наблюдений имеет среднее значение m =16.92 при девиансе D = 90694. При `PO4>=43.8` это множество делится на две части: `2)` 148 наблюдений (m =8.92, D = 31359) и `3)` 52 наблюдения с высоким уровнем обилия водорослей (m =39.7, D = 22863). Дальнейшие разбиения каждой из этих двух частей аналогичны.

Разумеется, лучший вариант – представить дерево графически. Популярны три варианта визуализации с использованием различных функций: `plot()`, `prettyTree()` из пакета `DMwR` и `prp()` из чрезвычайно продвинутого пакета `rpart.plot`:

```
DMwR::prettyTree(rt.a1)
```

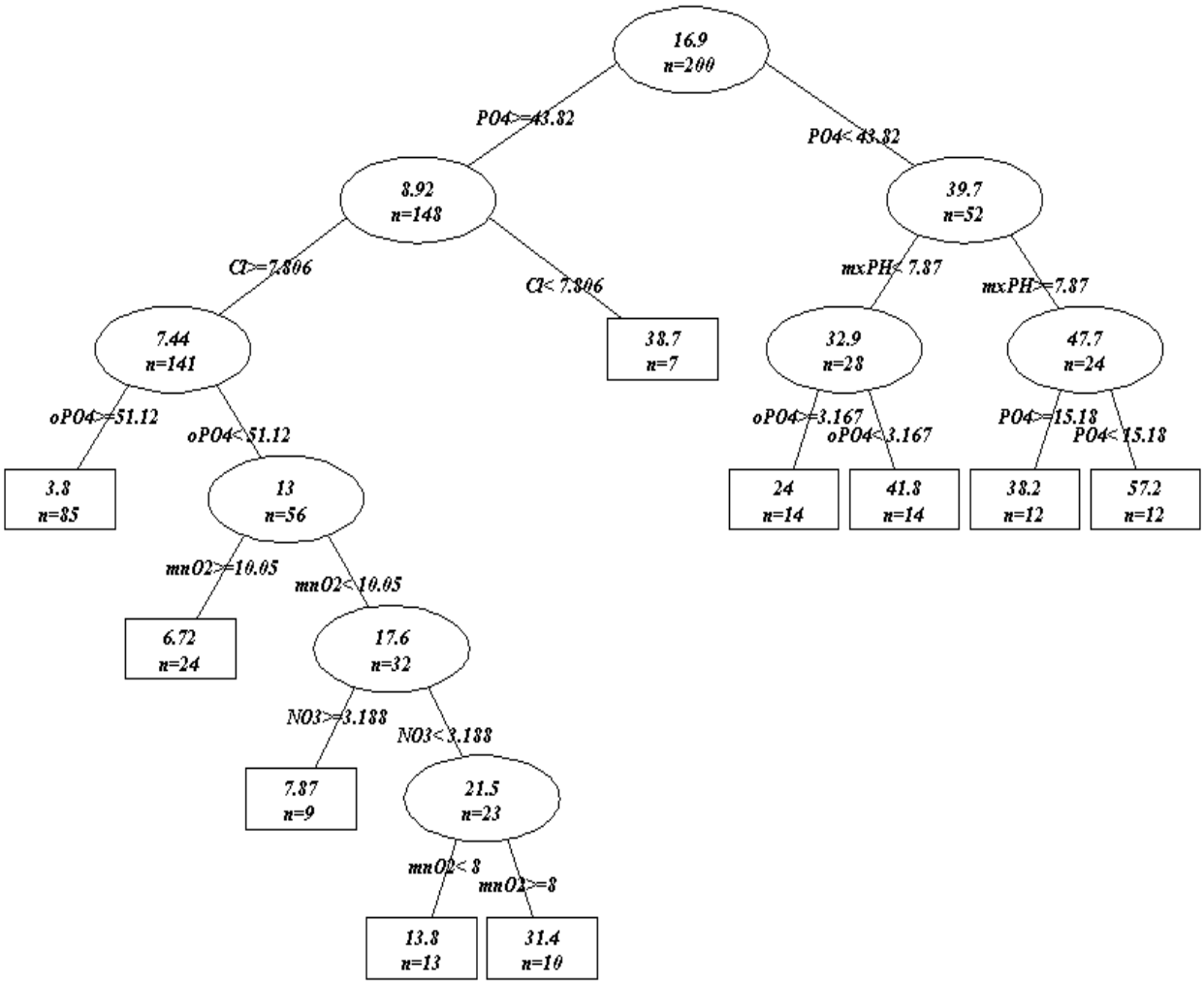


Рис. 4.7. Дерево `rpart` без обрезания ветвей

Полезно также проследить изменение перечисленных выше статистических критериев по мере выращивания дерева:

```
printcp(rt.a1)
```

```
Variables actually used in tree construction:
[1] Cl   mnO2 mxPH NO3  oPO4 PO4
Root node error: 90695/200 = 453.47
n= 200
        CP nsplit rel error  xerror    xstd
1 0.402145      0   1.00000 1.01569 0.13062
2 0.071921      1   0.59785 0.71689 0.11912
3 0.031241      2   0.52593 0.71641 0.11979
4 0.031211      3   0.49469 0.74315 0.12153
5 0.024435      4   0.46348 0.73140 0.12122
6 0.023840      5   0.43905 0.74064 0.11900
7 0.018065      6   0.41521 0.73495 0.11532
8 0.016291      7   0.39714 0.74622 0.11461
9 0.010000      9   0.36456 0.75839 0.11467
```

Функция `rpart()` и другие функции из пакета `rpart` имеют собственные возможности выполнить перекрестную проверку и оценить ее ошибку при различных значениях штрафа за сложность модели `cp`:

```
#   Снижаем порог штрафа за сложность с шагом .005
rtp.a1 <- rpart(a1 ~ ., data=algae[, 1:12],
   control=rpart.control(cp=.005))
#  График зависимости относительных ошибок от числа узлов
plotcp(rtp.a1)
with(rtp.a1, {lines(cptable[,2]+1, cptable[,3],
   type="b", col="red")
   legend("topright", c("Ошибка обучения",
   "Ошибка крос-проверки (CV)", "min(CV ошибка)+SE"),
   lty=c(1,1,2), col=c("red","black","black"), bty="n") })
```

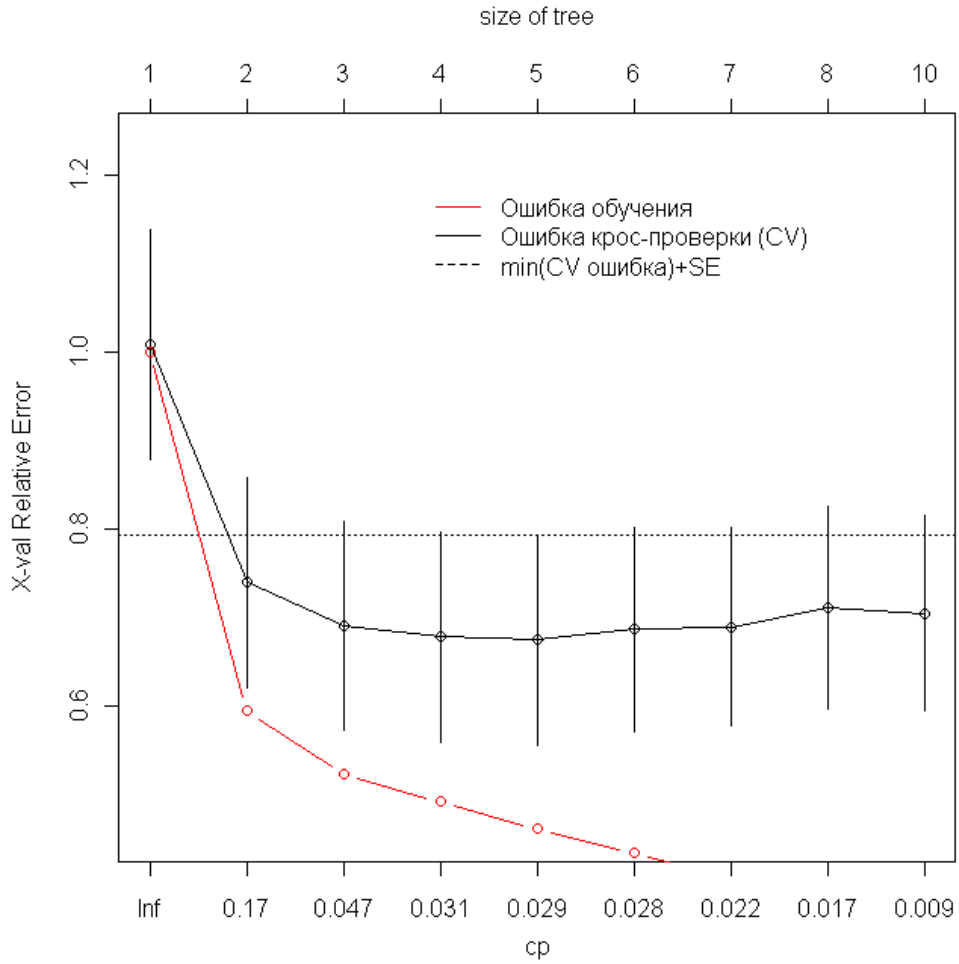


Рис. 4.8. Зависимость относительной ошибки перекрестной проверки от штрафа за сложность модели `cp`

На рис. 4.8 видно, что минимум относительной ошибки при перекрестной проверке приходится на значение `cp=0.029`. Выполним обрезку дерева при этом значении:

```
rtp.a1 <- prune(rtp.a1, cp=0.029)
prettyTree(rtp.a1)
```

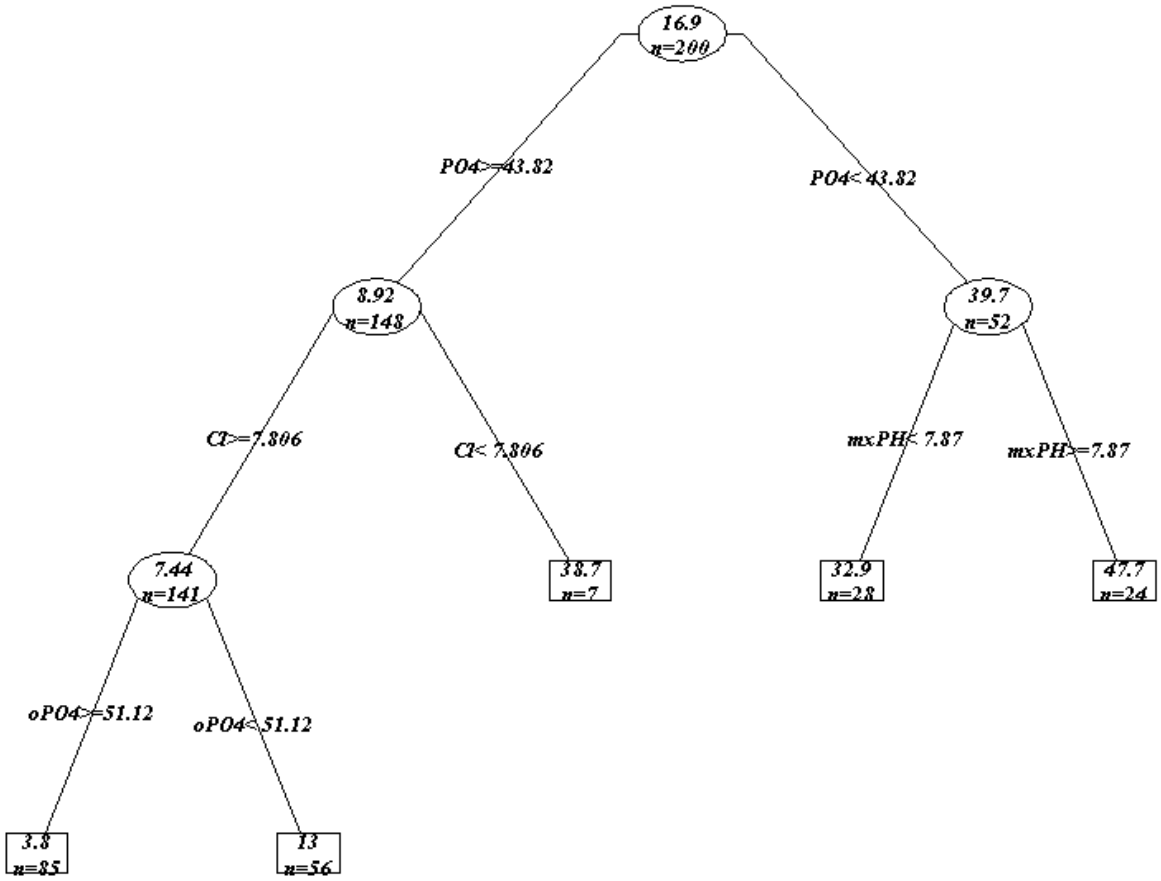


Рис. 4.9. Дерево `rpart` с обрезанием ветвей при `cp=0.029`

Выполним теперь дополнительную оптимизацию параметра `cp` с использованием функции `train()` из пакета `caret` (см раздел 3.5). Будем тестировать деревья регрессии при 30 значениях критерия `cp`, для каждого из которых выполним 10-кратную перекрестную проверку с 3 повторностями:

```
library(caret)
cvCtrl <- trainControl(method = "repeatedcv", repeats = 3)
(rt.a1.train <- train(a1 ~ ., data = algae[, 1:12],
   method = "rpart", tuneLength = 30, trControl = cvCtrl))
plot(rt.a1.train)
```

```
200 samples
Resampling: Cross-Validation (10 fold, repeated 3 times)
  cp      RMSE  Rsquared  RMSE SD  Rsquared SD
  0       15.8  0.45      3.19     0.18
  0.0139  15.8  0.446     3.23     0.189
  0.0279  15.6  0.459     3.5      0.208
  0.0418  15.1  0.474     3.64     0.216
  0.0557  15.2  0.468     3.79     0.216
  0.0697  15.4  0.454     4.01     0.234
  0.0836  15.7  0.451     4.18     0.242
  0.0976  16.2  0.428     4.08     0.214
…
  0.362   16.4  0.411     3.84     0.213
  0.376   17    0.388     4.04     0.203
  0.39    17.5  0.374     3.84     0.201
  0.404   19.4  0.236     2.93     0.166
```

```
RMSE was used to select optimal model using smallest value.
The final value used for the model was cp = 0.0418.
```

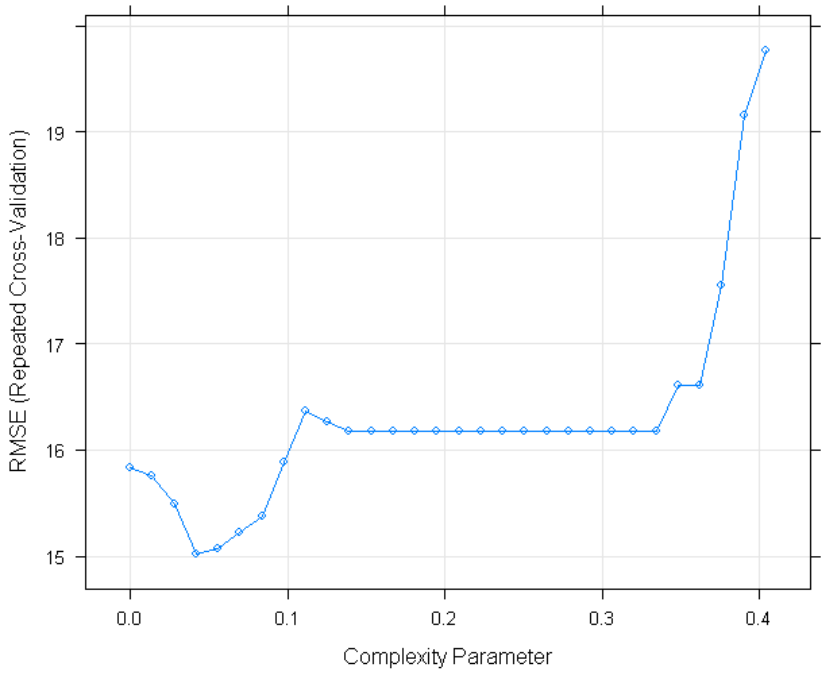


Рис. 4.10. Оценка параметра `cp` с использованием функции `train()`

```
rtt.a1 <- rt.a1.train$finalModel
prettyTree(rtt.a1)
```

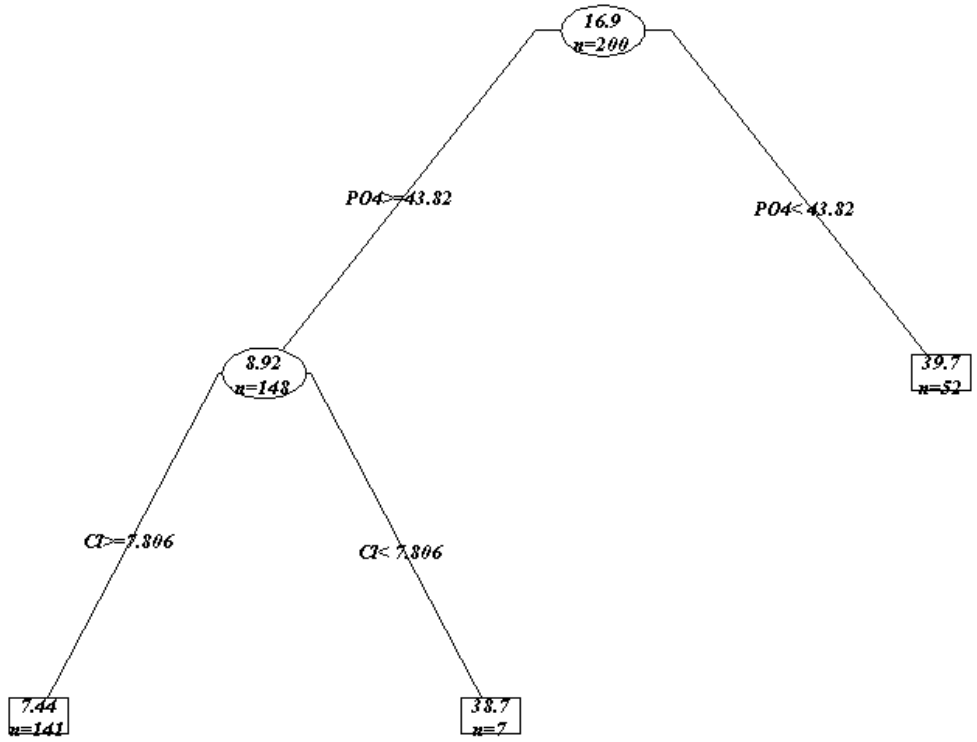


Рис. 4.11. Дерево `rpart` с обрезанием ветвей при `cp=0.0418`

При `cp = 0.0418` было получено существенно урезанное дерево, которое, правда, значительно потеряло в своей объясняющей ценности.

Построение деревьев с использованием алгортма условного вывода

Обратимся теперь к принципиально другим методам рекурсивного разделения при построении деревьев, представленным в пакете `party`. Стандартный механизм проверки статистического гипотез, который предотвращает переусложнение модели, реализован в функции `ctree()`, использующей метод построения деревьев на основе "условного вывода" (conditional inference). Алгоритм принимает во внимание характер распределения отдельных переменных и осуществляет на каждом шаге рекурсивного разделения данных несмещенный выбор влияющих ковариат, используя формальный тест на основе статистического критерия $c(\mathbf{t}_j, \mu_j, \Sigma_j)$, $j = 1, ..., m$, где μ, Σ – соответственно среднее и ковариация (Hothorn et al., 2006). Оценка статистической значимости c-критерия выполняется на основе перестановочного теста, в результате чего формируются компактные деревья, не требующие процедуры обрезания.

```
library(party)  # Построение дерева методом "условного вывода"
(ctree.a1 <- ctree(a1 ~ ., data = algae[, 1:12]))
plot(ctree.a1)
```

```
     Conditional inference tree with 4 terminal nodes
Number of observations:  200
1) PO4 <= 43.5; criterion = 1, statistic = 47.106
  2)*  weights = 52
1) PO4 > 43.5
  3) oPO4 <= 51.111; criterion = 0.985, statistic = 10.234
    4) size == {small}; criterion = 0.993, statistic = 14.65
      5)*  weights = 14
    4) size == {large, medium}
      6)*  weights = 49
  3) oPO4 > 51.111
    7)*  weights = 85
```

Оптимизацию параметра `mincriterion` выполним с использованием функции `train()` при тех же условиях перекрестной проверки:

```
ctree.a1.train <- train(a1 ~ ., data = algae[, 1:12],
    method = "ctree", tuneLength = 10, trControl = cvCtrl)
ctreet.a1 <- ctree.a1.train$finalModel
plot(ctreet.a1)
```

```
Resampling: Cross-Validation (10 fold, repeated 3 times)
mincriterion  RMSE  Rsquared  RMSE SD  Rsquared SD
  0.01          16.9  0.423     4.58     0.193
  0.119         16.8  0.432     4.42     0.19
  0.228         16.5  0.436     4.58     0.194
  0.337         16.4  0.45      4.63     0.197
  0.446         16.2  0.462     4.21     0.18
  0.554         16.1  0.464     4.25     0.18
  0.663         15.6  0.487     4.16     0.186
  0.772         15.6  0.485     4.17     0.174
  0.881         15.6  0.487     4.36     0.182
  0.99          16    0.443     4.65     0.157

RMSE was used to select optimal model using smallest value.
The final value used for the model was mincriterion = 0.881.
```

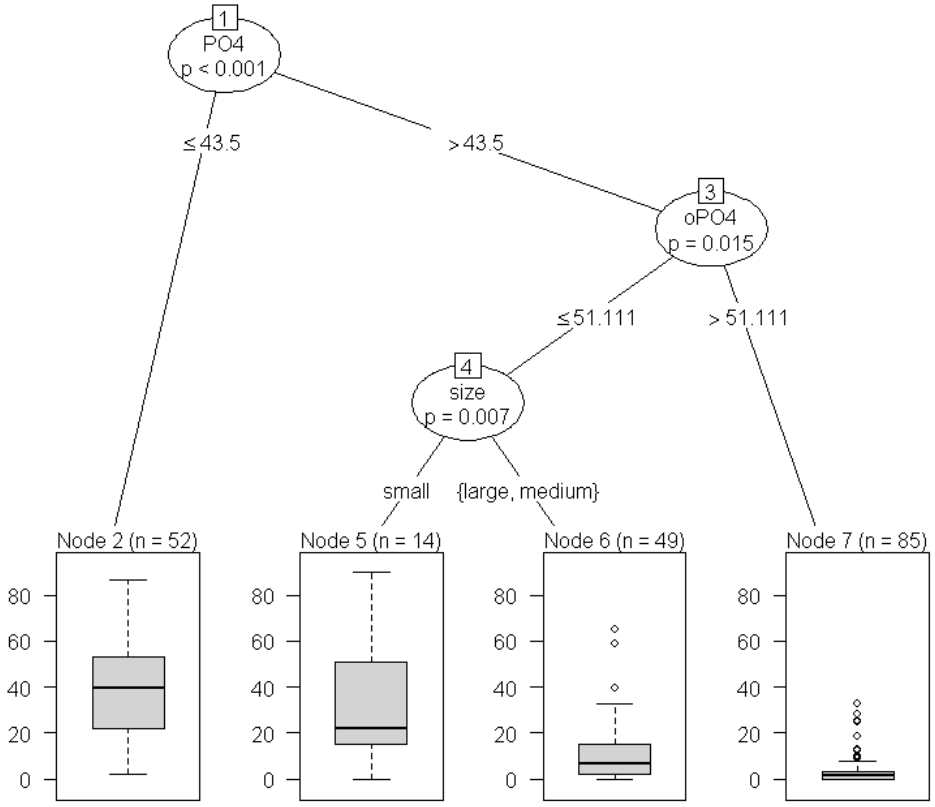


Рис. 4.12. Дерево `cpart` без оптимизации параметра `mincriterion`

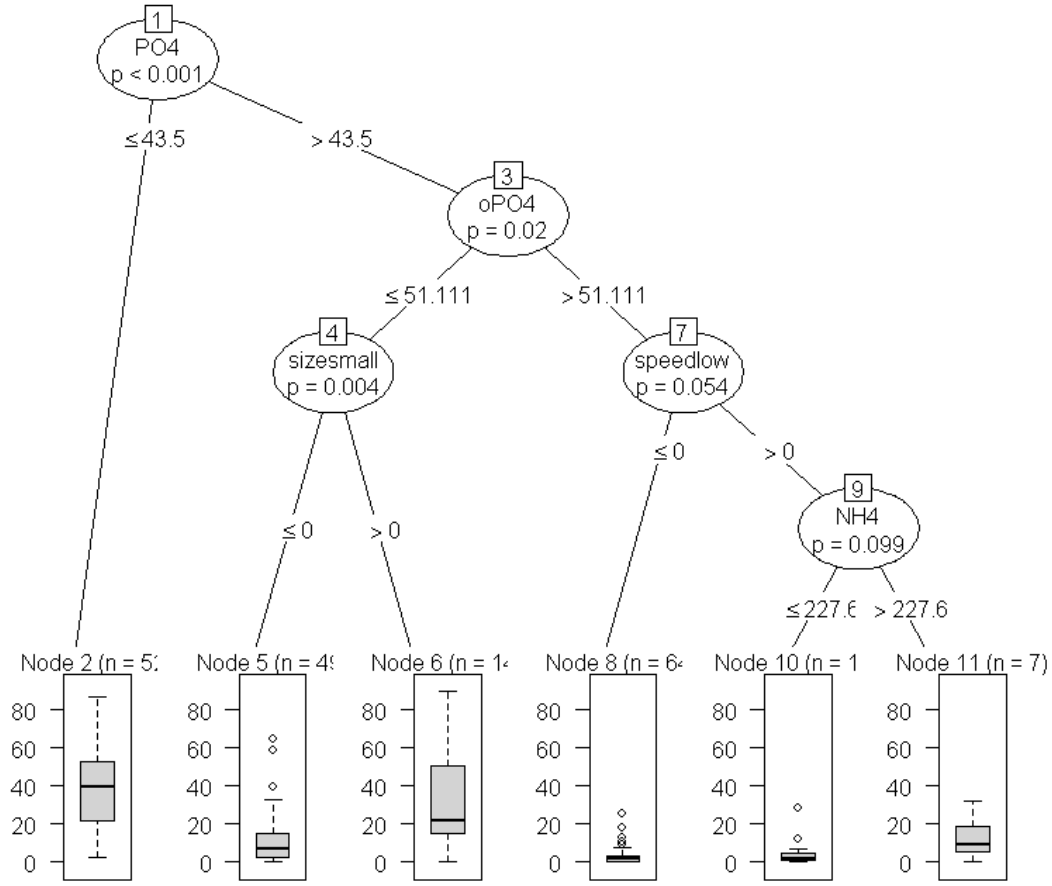


Рис. 4.13. Дерево `cpart` после оптимизации параметра `mincriterion`

Здесь имел место обратный процесс: число узлов дерева было предложено увеличить с 7 до 11. Обратим также внимание на то, что в дереве появились категориальные переменные (размер и скорость течения реки), которые были проигнорированы `rpart`-деревьями.

Тестирование моделей с использованием дополнительного набора данных

Используем для прогноза набор данных (Torgo, 2011) из 140 наблюдений, который мы уже применяли в предыдущем разделе. Данные с восстановленными пропущенными значениями мы сохранили ранее в файле `algae_test.RData` (см раздел 4.1).

Оценим точность каждой модели на этом наборе данных по трем показателям: среднему абсолютному отклонению (`MAE`), корню из среднеквадратичного отклонения (`RSME`) и квадрату коэффициента детерминации `Rsq = 1 - NSME`, где `NSME` – относительная ошибка, равная отношению средних квадратов отклонений от модельных значений и от общего среднего:

```
load(file="algae_test.RData") # Загрузка таблиц Eval,Sols
# Функция, выводящая вектор критериев
ModCrit <- function (pred, fact) {
  mae <- mean(abs(pred-fact))
  rmse <- sqrt(mean((pred-fact)^2))
  Rsq <- 1-sum((fact-pred)^2)/sum((mean(fact)-fact)^2)
  c(MAE=mae, RSME=rmse, Rsq=Rsq)
}
Result <- rbind(
 rpart_prune = ModCrit(predict(rtp.a1,Eval),Sols[,1]),
 rpart_train = ModCrit(predict(rt.a1.train,Eval),Sols[,1]),
 ctree_party = ModCrit(predict(ctree.a1,Eval),Sols[,1]),
 ctree_train = ModCrit(predict(ctree.a1.train,Eval),Sols[,1])
)
Result
```

```
                  MAE     RSME      Rsq
rpart_prune 11.16546 16.09485 0.3828278
rpart_train 10.72834 15.36578 0.4374751
ctree_party 11.32286 16.53470 0.3486336
ctree_train 11.25551 16.40532 0.3587876
```

Можно с разумной осторожностью сделать вывод о том, что деревья, построенные `rpart()`, немного точнее, чем деревья условного вывода `ctree()`. При этом все деревья решений оказались существенно эффективней для прогнозирования свежих данных, чем все ранее построенные модели.

4.4. Ансамбли моделей: бэггинг, случайные леса, бустинг

В статистике хорошо известно интуитивное соображение, согласно которому усреднение результатов наблюдений может дать более устойчивую и надежную оценку, поскольку ослабляется влияние случайных флуктуаций в отдельном измерении. На аналогичной идее было основано развитие алгоритмов комбинирования моделей, в результате чего построение их

ансамблей оказалось одним из самых мощных методов машинного обучения, нередко превосходящим по качеству предсказаний другие методы.

Одним из решений, обеспечивающих необходимое разнообразие моделей, является их повторное обучение на выборках, случайно выбранных из генеральной совокупности, либо иных подмножествах данных, сконструированных из уже имеющихся (рис. 4.14). Для получения устойчивого прогноза частные предсказания этих моделей тем или иным образом комбинируют, например, с помощью простого усреднения или голосования (возможно, взвешенного).

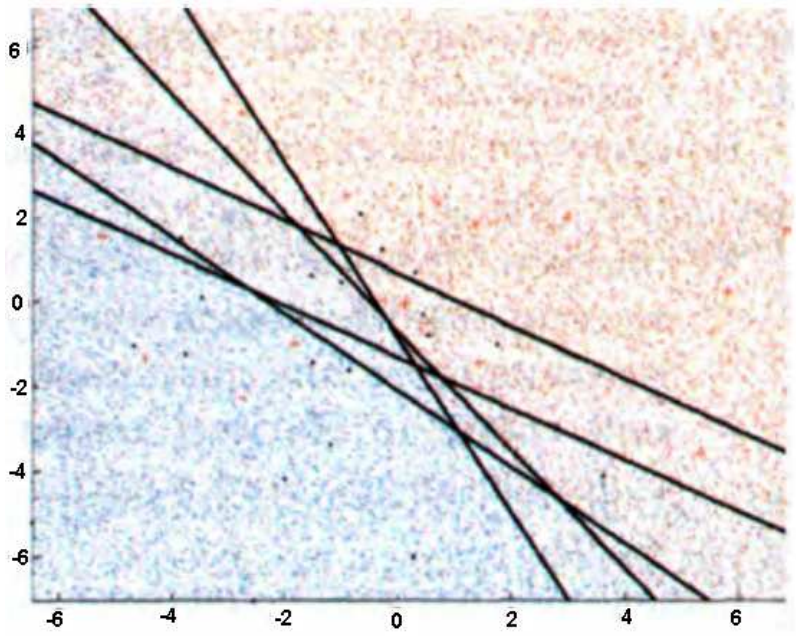


Рис. 4.14. Ансамбль из пяти линейных классификаторов: каждый сегмент пространства объектов отличается средними вероятностями предсказания классов (подробности см. Флах, 2015, с. 344)

В разделе 2.2 был описан бутстреп – процедура генерации повторных случайных выборок из исходного набора данных. Бутстреп-выборки производятся равномерно и с возвращением, поэтому некоторые исходные примеры будут отсутствовать, а другие – дублироваться: в среднем одна такая выборка содержит около 2/3 уникальных исходных наблюдений.

Бэггинг и случайные леса

Бутстреп при формировании ансамбля моделей оказался особенно полезен в сочетании с древовидными структурами, которые очень чувствительны к небольшому изменению обучающих данных. Описанные в предыдущем разделе деревья решений обычно имеют низкое смещение, но страдают от *высокой дисперсии*. Это означает, что если мы случайным образом разобьем обучающие данные на две части и построим дерево решений на основе каждой из них, то полученные результаты могут оказаться довольно разными.

Подобно тому, как усреднение нескольких наблюдений снижает оценку дисперсии данных, так и разумным способом снижения дисперсии прогноза

является извлечение большого количества порций данных из генеральной совокупности, построение предсказательной модели по каждой обучающей выборке и усреднение полученных предсказаний. Если вместо отдельных обучающих выборок (которых нам, как правило, всегда не хватает) выполнить бутстреп и на основе сгенерированных псевдо-выборок построить B деревьев регрессии, то средний коллективный прогноз

$$\hat{f}_{bag}(x) = \{f^1(x) + f^2(x) + ... + f^B(x)\}/B$$

будет обладать более низкой дисперсией. Эта процедура и называется *бэггингом* (bagging, сокр. от **b**ootstrap **agg**regat**ing**). Как нами будет показано в последующих главах, бэггинг можно проводить не только в отношении деревьев регрессии, но и иных моделей: опорных векторов, нейронных сетей, линейных дискриминантов, байесовских вероятностей и др.

Метод *случайного леса* (Random Forest) представляет собой дальнейшее улучшение бэггинга деревьев решений, которое заключается в *устранении корреляции* между деревьями. Как и в случае с бэггингом, мы строим несколько сотен деревьев решений по обучающим бутстреп-выборкам. Однако на каждой итерации построения дерева случайным образом выбирается m из p подлежащих рассмотрению предикторов и разбиение разрешается выполнять только по одному из этих m переменных.

Смысл этой процедуры, оказавшейся весьма эффективной для повышения качества получаемых решений, заключается в том, что с вероятностью $(p - m)/p$ блокируется какой-нибудь потенциально доминирующий предиктор, стремящийся войти в каждое дерево. Если доминирование таких предикторов разрешить, то все деревья в итоге будут очень похожи друг на друга, а получаемые на их основе предсказания будут сильно коррелировать и снижение дисперсии будет не столь очевидным. Благодаря блокированию доминантов, другие предикторы получат свой шанс, и вариация деревьев возрастает.

Выбор малого значения m при построении случайного леса обычно будет полезным при наличии большого числа коррелирующих предикторов. Естественно, если случайный лес строится с использованием $m = p$, то вся процедура сводится к простому бэггингу.

Применим методы бэггинга и случайного леса к прогнозированию данных по обилию водорослей в реках разного типа (см. три предыдущих раздела). Поскольку бэггинг – это просто частный случай метода случайного леса, то мы можем использовать одну и ту же функцию `randomForest()` пакета `randomForest` для R. Бэггинг выполняется, если задать параметр `mtry = ncol(x)`:

```
load(file="algae.RData") # Загрузка таблицы algae - раздел 4.1
x <- as.data.frame(model.matrix(a1~.,data=algae[,1:12])[,-1])
library(randomForest)
randomForest(x, algae$a1, mtry = ncol(x))
```

```
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 15
```

```
          Mean of squared residuals: 271.6539
                    % Var explained: 40.09
```

Как видно из полученных результатов, прогнозирование выполнялось по 500 деревьям, в которых было использовано только 40% исходных переменных. Оценить эффективность этой модели при перекрестной проверке можно с использованием функции `train()` из пакета `caret`:

```
bag.a1 <- train(x, algae$a1,
     preProc = c('center', 'scale'),
     method = 'rf', trControl = trainControl(method = "cv"),
     tuneGrid = expand.grid(.mtry = ncol(x)))
```

```
Resampling: Cross-Validated (10 fold)
  RMSE      Rsquared   RMSE SD   Rsquared SD
  16.27656  0.4503684  4.495015  0.1934045
```

Модель случайного леса можно построить этой же процедурой, задав последовательность значений `mtry` для оптимизации:

```
ranfor.a1 <- train(x, algae$a1,
     preProc = c('center', 'scale'),
     method = 'rf', trControl = trainControl(method = "cv"),
     tuneGrid = expand.grid(.mtry = 2:10),
           importance = TRUE)
```

```
Resampling results across tuning parameters:
  mtry  RMSE      Rsquared   RMSE SD   Rsquared SD
   2    15.58681  0.4973347  3.339887  0.2115185
   3    15.64973  0.4893305  3.613366  0.2112102
   4    15.87789  0.4819333  3.467073  0.2047955
   5    15.92607  0.4760140  3.492468  0.2048518
   6    15.94487  0.4722841  3.618424  0.2049526
   7    16.09182  0.4666949  3.623971  0.2080536
   8    16.12491  0.4651339  3.684320  0.2122767
   9    16.41238  0.4516508  3.713966  0.2038540
  10    16.30965  0.4522984  3.696520  0.2039949
RMSE was used to select optimal model using smallest value.
The final value used for the model was mtry = 2.
```

Заметим, что бутстреп дает хорошую возможность провести специальную процедуру перекрестной проверки, называемую "тестом по наблюдениям, не попавшим в сумку" (out–of–bag observations). Поскольку ключевая идея бэггинга состоит в многократном построении моделей по наблюдениям из бутстреп-выборок, то каждое конкретное дерево строится на основе примерно двух третей всех наблюдений. Остальная треть наблюдений не используется в обучении, но вполне может быть использована для независимого тестирования: ошибка на таких оставшихся данных (out–of–bag error) является состоятельной оценкой ошибки на контрольной выборке (Джеймс и др., 2016).

Основным преимуществом деревьев решений является привлекательная и легко интерпретируемая итоговая диаграмма вроде тех, которые показаны в разделе 4.3. Хотя набор полученных в результате бэггинга деревьев гораздо сложнее интерпретировать, чем отдельно стоящее дерево, можно получить целых два обобщенных показателя важности каждого предиктора. Их графики легко построить при помощи функции `varImpPlot()`:

```
varImpPlot(ranfor.a1$finalModel)
```

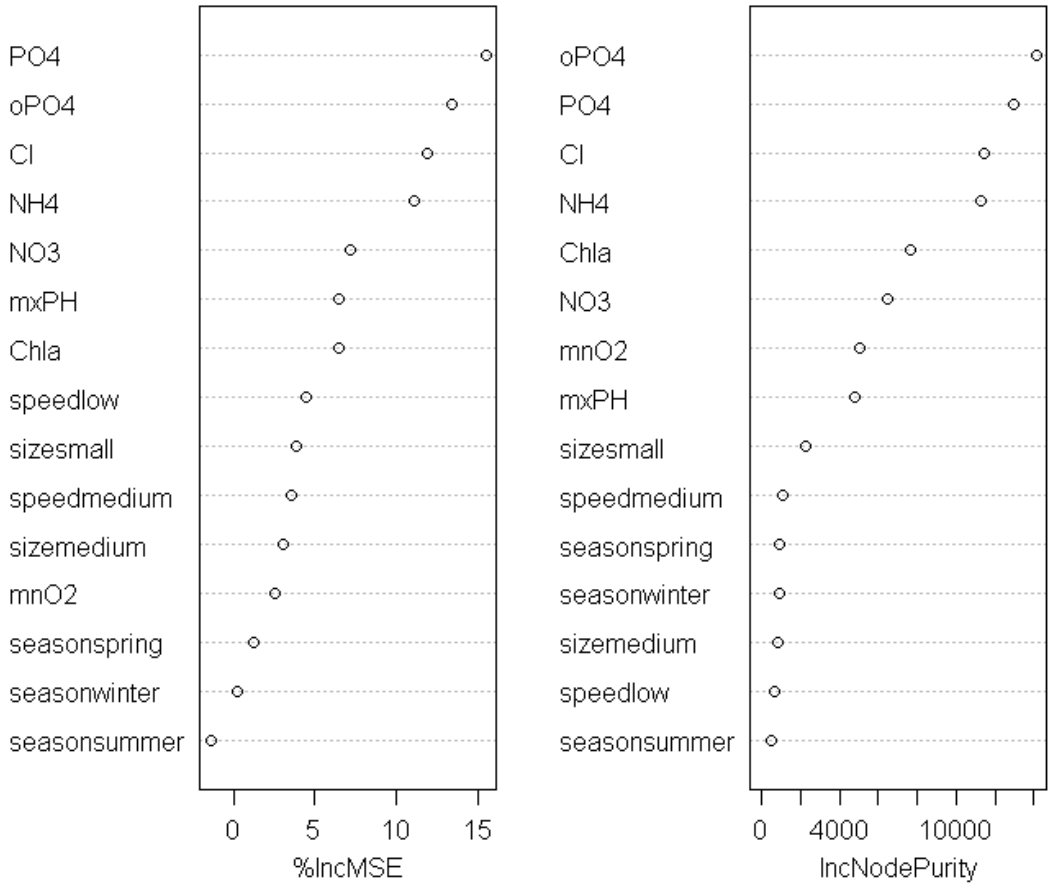


Рис. 4.15. Показатели важности отдельных переменных для модели случайного леса

На рис. 4.15 приведены два показателя важности: `%IncMSE` основан на среднем снижении точности предсказания на оставшихся данных, а `IncNodePurity` – мера среднего увеличения "чистоты узла" дерева (node purity) в результате разбиения данных по соответствующей переменной. В случае деревьев регрессии чистота узла выражается через ошибку RSS.

Количество деревьев `B` не является критическим параметром при использовании бэггинга: очень большое значение `B` не приведет к переобучению. На практике обычно используется значение `B`, достаточно большое для стабилизации ошибки: в частности, как следует из графика на рис. 4.16, величина `B` = 100 уже обеспечивает хорошее качество предсказаний в нашем примере (по умолчанию, `B` = 500).

```
plot(ranfor.a1$finalModel, col="blue", lwd=2)
plot(bag.a1$finalModel, col="green", lwd=2, add=TRUE)
legend("topright",c("Bagging", "RandomForrest"),
          col=c("green","blue"), lwd=2)
```

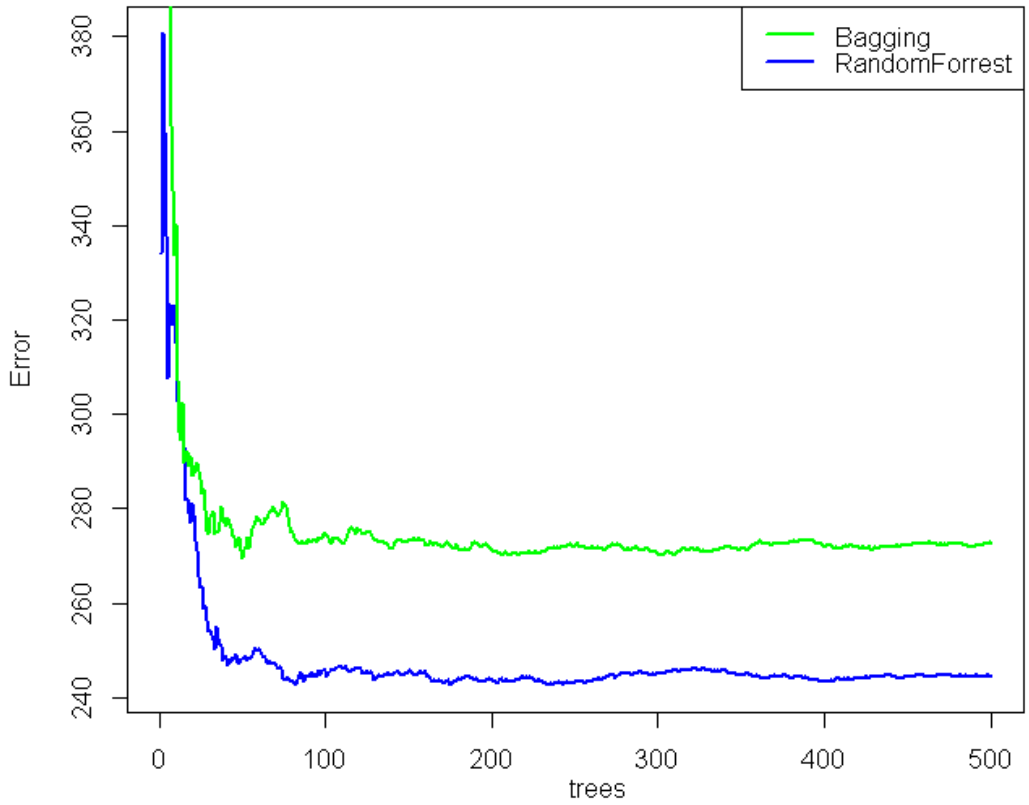


Рис. 4.16. Зависимость ошибки на обучающей выборке от числа агрегируемых деревьев при бэггинге и использовании алгоритма "случайный лес"

Бустинг

Другим методом улучшения предсказаний является *бустинг* (*boosting*), идея которого заключается в итеративном процессе последовательного построения частных моделей. Каждая новая модель обучается с использованием информации об ошибках, сделанных на предыдущем этапе, а результирующая функция представляет собой линейную комбинацию всего ансамбля моделей с учетом минимизации любой штрафной функции. Подобно бэггингу, бустинг является общим подходом, который можно применять ко многим статистическим методам регрессии и классификации. Здесь мы ограничимся обсуждением градиентного бустинга в контексте деревьев регрессии.

Бутстреп-выборки в ходе реализации бустинга не создаются, но вместо этого каждое дерево строится по набору данных $\{X, r\}$, который на каждом шаге модифицируется определенным образом. На первой итерации по значениям исходных предикторов строится дерево $f^1(x)$ и находится вектор остатков r_1. На последующем этапе новое регрессионное дерево $f^2(x)$ строится уже не по обучающим данным X, а по остаткам r_1 предыдущей модели. Линейная комбинация прогноза по построенным деревьям дает нам новые остатки $r_2 \leftarrow r_1 + \lambda f^2(x)$, и этот итерационный процесс повторяется B раз. Благодаря построению неглубоких деревьев по остаткам, прогноз отклика медленно улучшается в областях, где одиночное дерево работает не очень

хорошо. Такие деревья могут быть довольно небольшими, лишь с несколькими конечными узлами. Параметр сжатия λ регулирует скорость этого процесса, позволяя создавать комбинации деревьев более сложной формы для "атаки" остатков. Итоговая модель бустинга представляет собой ансамбль $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$.

В среде R для построения бустинг-моделей на основе деревьев решений можно использовать функцию gbm() из пакета gbm (Generalized Boosted Models). Процесс моделирования проходит под управлением трех гиперпараметров:

1. Число деревьев *B* (формальный параметр n.tree). В отличие от бэггинга, бустинг может, хотя и медленно, приводить к переобучению при чрезмерно большом *B*.

2. Параметр сжатия λ (shrinkage), который корректирует величину вклада каждого дополнительного дерева и контролирует скорость, с которой происходит обучение модели при реализации бустинга. Типичные значения λ варьируют от 0.01 до 0.001, и их оптимальный выбор зависит от решаемой проблемы. Для достижения хорошего качества предсказаний очень низкие значения λ требуют очень большого значения *B*.

3. Число внутренних узлов *d* (interaction.depth) в каждом дереве, которое контролирует сложность получаемого в результате бустинга ансамбля моделей. По своей сути, параметр *d* отражает глубину взаимодействий между предикторами в итоговой модели. Если эти взаимодействия не слишком выражены, то хорошо работает *d* = 1, и тогда дополнительные деревья представляют собой просто "пни" (stump), т.е. содержат только один внутренний узел. В таком случае получаемый в результате бустинга ансамбль становится аддитивной моделью, поскольку каждый ее член представлен только одной переменной.

Тип решаемой задачи регулируется параметром distribution, который определяет оптимизируемую функцию:

- для решения задач регрессии задается значение "gaussian" – квадратичный штраф, или "laplace" – штраф по абсолютной величине отклонения;
- для задач бинарной классификации используют значение "bernoulli" – функция кросс-энтропии, или "adaboost" – экспоненциальный штраф.

Используем значение shrinkage = 0.001, установленное функцией gbm() по умолчанию. Функция summary() в отношение этого метода выводит список предикторов и соответствующие им значения показателя важности:

```
library(gbm)
set.seed(1)
xd <- cbind(a1 = algae$a1, x)
boost.a1=gbm(a1 ~ ., data= xd, distribution="gaussian",
n.trees=1000,interaction.depth=3)
summary(boost.a1)
```

```
                      var     rel.inf
oPO4                 oPO4 28.53237588
NH4                   NH4 24.31701220
PO4                   PO4 20.50620263
Cl                     Cl 14.71265676
Chla                 Chla  4.23461990
mxPH                 mxPH  2.74122308
NO3                   NO3  2.06415040
mnO2                 mnO2  1.63774553
sizesmall       sizesmall  0.76292416
speedmedium   speedmedium  0.21372651
seasonwinter seasonwinter  0.12754643
speedlow         speedlow  0.06571911
sizemedium     sizemedium  0.05146608
seasonspring seasonspring  0.02160894
seasonsummer seasonsummer  0.01102240
```

Можно рассчитать среднюю ошибку модели на обучающей выборке, которая существенно меньше, чем при бэггинге:

```
pred=predict(boost.a1, x, n.trees=1000)
mean((pred-algae$a1)^2)
```

```
[1] 233.277
```

Выполним оптимизацию параметров построения градиентного бустинга с использованием функции `train()`. Как скаано выше, таких параметров три:

```
modelLookup("gbm")
```

```
  model         parameter                label forReg forClass probModel
1   gbm           n.trees # Boosting Iterations   TRUE     TRUE      TRUE
2   gbm interaction.depth        Max Tree Depth   TRUE     TRUE      TRUE
3   gbm         shrinkage             Shrinkage   TRUE     TRUE      TRUE
```

Принимая во внимание, что параметры `shrinkage` и `n.trees` связаны обратно пропорциональной зависимостью, уменьшим число деревьев до 50, одновременно увеличив значение `shrinkage` по сравнению с применяемыми выше:

```
(gbmFit.a1 <- train(a1 ~ ., data= xd,
    method = "gbm", trControl = trainControl(method = "cv"),
    tuneGrid = expand.grid(.shrinkage = c(0.1,0.05,0.02),
            .interaction.depth=2:5, .n.trees = 50),
    verbose = FALSE))
```

```
Stochastic Gradient Boosting
Resampling: Cross-Validated (10 fold)
  shrinkage  interaction.depth  RMSE      Rsquared   RMSE SD   Rsquared SD
  0.02       2                 16.18758  0.4938156  3.630732  0.2227428
  0.02       3                 16.19763  0.4939358  3.654470  0.2161953
  0.02       4                 16.06474  0.5008917  3.650950  0.2000208
  0.02       5                 16.07771  0.5014302  3.616061  0.2158463
  0.05       2                 15.73279  0.4760551  3.542693  0.2187849
  0.05       3                 15.55629  0.4834203  3.316275  0.1938207
  0.05       4                 15.71659  0.4751684  3.516850  0.2060737
  0.05       5                 15.49480  0.4835010  3.291277  0.1913610
  0.10       2                 16.42262  0.4231575  2.858400  0.1649736
  0.10       3                 16.07729  0.4551994  3.395341  0.1835292
  0.10       4                 15.94851  0.4657523  3.362024  0.2158680
  0.10       5                 16.48367  0.4290084  2.774404  0.1566670
 Tuning parameter 'n.trees' was held constant at a value of 50
RMSE was used to select optimal model using smallest value.
The final values used for the model were n.trees = 50,
interaction.depth = 5 and shrinkage = 0.05.
```

Бустинг деревьев регрессии может быть реализован также с использованием другого метода: с помощью функции `bstTree()` из пакета `bst`:

```
modelLookup("bstTree")
```

```
  1 bstTree     mstop # Boosting Iterations   TRUE     TRUE     FALSE
  2 bstTree  maxdepth        Max Tree Depth   TRUE     TRUE     FALSE
  3 bstTree        nu             Shrinkage   TRUE     TRUE     FALSE
```

Параметры, оптимизируемые методом `bstTree`, имеют несколько отличающиеся названия, но фактически эквивалентный содержательный смысл. Выполним их настройку с использованием параметров, заданных по умолчанию:

```
library(bst)
boostFit.a1 <- train(a1 ~ ., data= xd,
      method='bstTree', trControl=trainControl(method = "cv"),
      preProc=c('center','scale'))
plot(boostFit.a1)
```

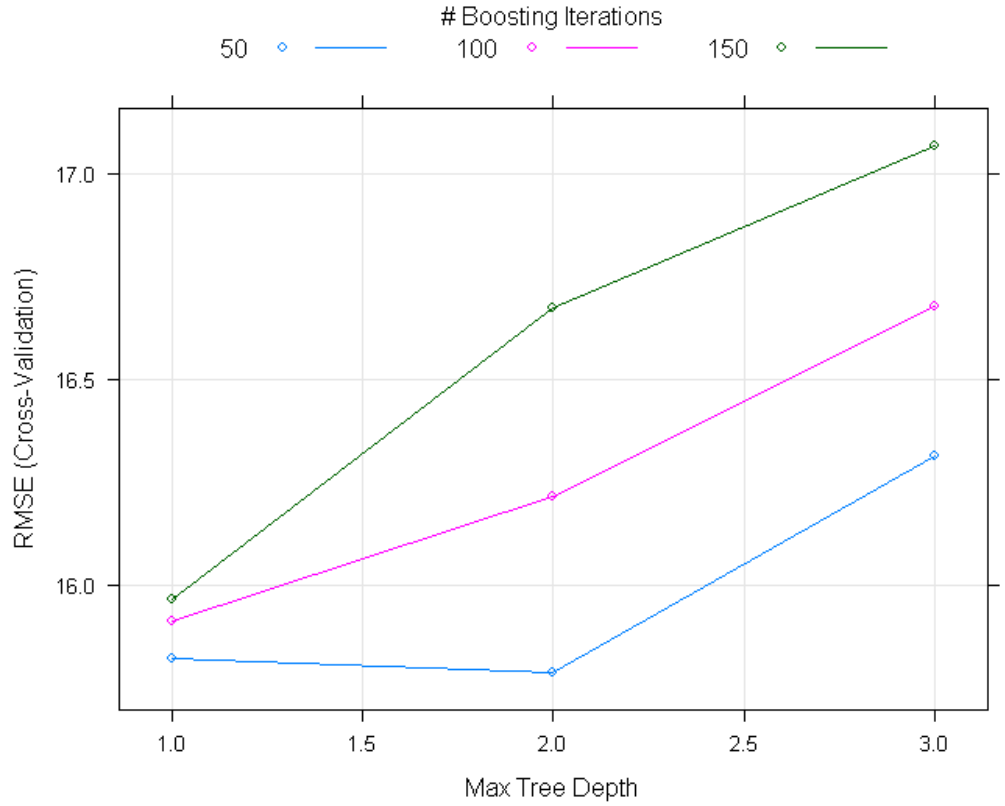


Рис. 4.17. Зависимость ошибки от числа агрегируемых деревьев при бустинге (по результатам перекрестной проверки)

```
Boosted Tree
Resampling: Cross-Validated (10 fold)
  maxdepth  mstop  RMSE      Rsquared   RMSE SD   Rsquared SD
  1          50    15.82132  0.4788595  3.528825  0.1782804
  1         100    15.91057  0.4709657  3.386138  0.1670161
  1         150    15.96351  0.4666395  3.205183  0.1540219
  2          50    15.78701  0.4803771  3.561134  0.1740701
  2         100    16.21444  0.4538259  3.317425  0.1518355
  2         150    16.67177  0.4263850  3.165604  0.1436462
```

```
  3          50    16.31444  0.4480353  3.605141  0.1762307
  3         100    16.67521  0.4249761  3.483872  0.1652263
  3         150    17.06778  0.4033277  3.421522  0.1589667

Tuning parameter 'nu' was held constant at a value of 0.1
RMSE was used to select optimal model using smallest value.
The final values used for the model were mstop = 50,
maxdepth = 2 and nu = 0.1.
```

Тестирование моделей с использованием дополнительного набора данных

Используем для прогноза набор данных (Torgo, 2011) из 140 наблюдений, который мы уже применяли в предыдущем разделе. Данные, с восстановленными пропущенными значениями мы сохранили ранее в файле `algae_test.RData` (см раздел 4.1).

Выполним прогноз для набора предикторов тестовой выборки и оценим точность каждой модели по трем показателям: среднему абсолютному отклонению (`MAE`), корню из среднеквадратичного отклонения (`RSME`) и квадрату коэффициента детерминации `Rsq = 1 - NSME`, где `NSME` – относительная ошибка, равная отношению среднему квадрату отклонений от модельных значений и от общего среднего:

```
load(file="algae_test.RData") # Загрузка таблиц Eval, Sols
y <- Sols$a1
EvalF <- as.data.frame(model.matrix(y ~ .,Eval)[,-1])
# Функция, выводящая вектор критериев
ModCrit <- function (pred, fact) {
  mae <- mean(abs(pred-fact))
  rmse <- sqrt(mean((pred-fact)^2))
  Rsq <- 1-sum((fact-pred)^2)/sum((mean(fact)-fact)^2)
  c(MAE=mae, RSME=rmse, Rsq=Rsq)
}
Result <- rbind(
   bagging = ModCrit(predict(bag.a1, EvalF), Sols[,1]),
   ranfor = ModCrit(predict(ranfor.a1, EvalF), Sols[,1]),
   bst.gbm = ModCrit(predict(gbmFit.a1, EvalF), Sols[,1]),
   bst.bst = ModCrit(predict(boostFit.a1, EvalF), Sols[,1]))
Result
```

```
               MAE     RSME       Rsq
bagging 10.101308 14.53114 0.4969259
ranfor   9.968411 14.22920 0.5176151
bst.gbm 10.034274 14.34638 0.5096377
bst.bst  9.971036 14.38706 0.5068524
```

4.5. Сравнение построенных моделей и оценка информативности предикторов

Разделы 4.1-4.4 содержат подробную информацию о результатах тестирования различных типов моделей регрессии в идентичных условиях и на одном и том же примере, обобщенную в файле `Models.txt`. Сравнительная точность прогноза, оцениваемая по квадрату коэффициента

детерминации Rsquared при 10-кратной перекрестной проверке (ось Y) и на контрольной выборке из 140 наблюдений (ось X), представлена на рис. 4.18:

```
Models <- read.delim('Models.txt',header=T)
plot(Models$Rsq,Models$Rsquared,pch=CIRCLE<-16,
     col=8-Models$col, cex=2.5,
     xlab="Rsquared на дополнительной выборке",
     ylab="Rsquared при кросс-проверке")
text(Models$Rsq,Models$Rsquared,rownames(Models),
     pos=4, font=4,cex=0.8)
legend('bottomright',c('Бэггинг/бустинг','Деревья',
     'Регрессия kNN','PLS/PCR','Лассо','Линейные модели'),
     col=2:7, pch=CIRCLE<-16, cex=1)
```

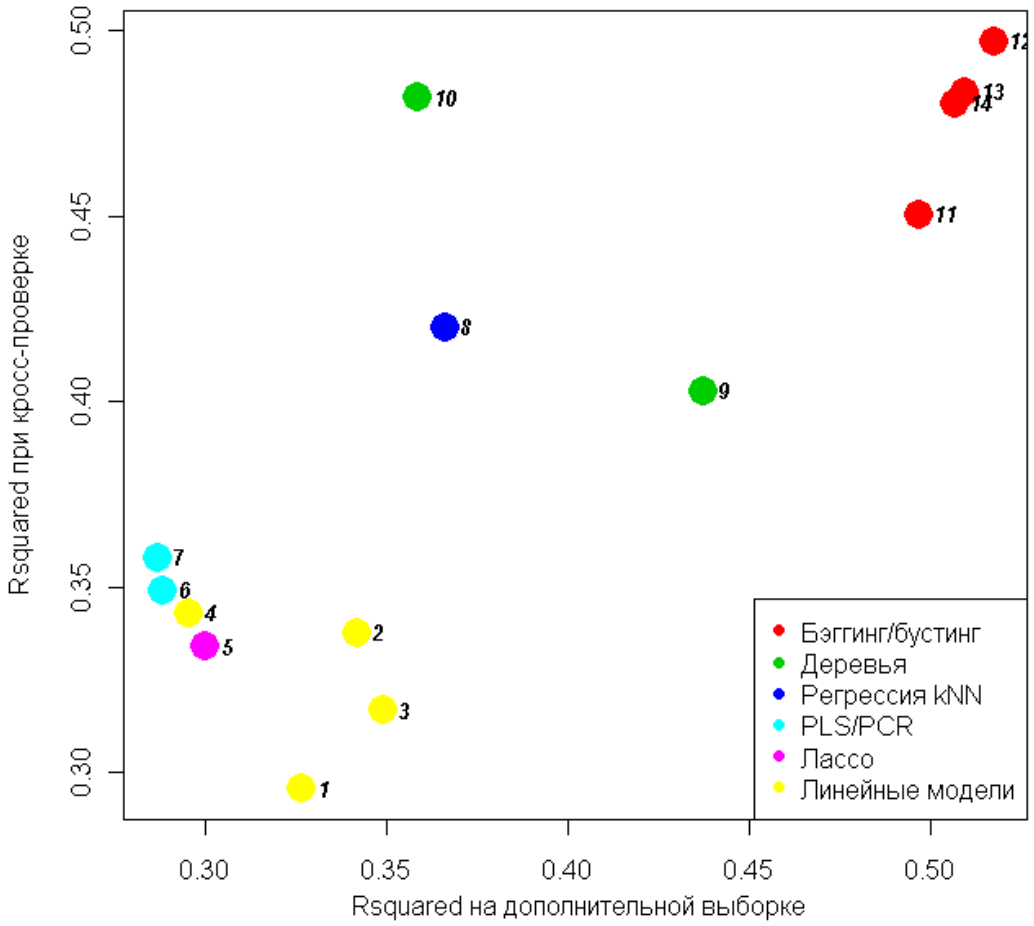


Рис. 4.18. Результаты тестирования точности моделей регрессии различного класса при кросс-проверке и на внешнем дополнении

Хотя использованный тестовый пример представляется вполне типичным, отсутствие повторностей нашего вычислительного эксперимента не дает нам права делать далеко идущие выводы. Однако некоторые достоинства и недостатки отдельных типов прогнозирующих моделей проявились достаточно четко.

Бесспорными лидерами по точности прогноза явились модели случайного леса (*12*), бустинга (*13-14*) и бэггинга (*11*), основанные на ансамблях деревьев решений. Неплохо себя проявили также одиночные

деревья и регрессия *k* ближайших соседей. Модели, основанные на обобщенных характеристиках обучающей выборки или ее преобразованиях, такие как регрессия на главные компоненты (*7*), PLS, лассо, дерево условного вывода (*10*), показали сравнительно неплохие результаты при перекрестной проверке, но оказались не столь хорошими "предсказателями" на "свежих" данных, возможно, не столь похожих на обучающие.

Метод случайного леса не только позволяет построить превосходные модели прогнозирования, но и выполнить такую работу, как селекция набора всех информативных признаков (finding all relevant variables). В общем случае эта проблема решается с использованием трех возможных подходов (https://habrahabr.ru/post/264915/):

- методы фильтрации (filter methods), которые рассматривают каждую переменную независимо и, в некоторой степени, изолированно, оценивая ее по тому или иному показателю (информационные или статистические критерии, минимальная избыточность при максимальной релевантности mRmR и др.);
- адаптационные методы (wrapper methods), осуществляющие направленный перебор разных подмножеств признаков и оценка их по заданному критерию (в разделе 4.1 рассматривались три таких алгоритма: пошаговый, генетический и RFE);
- встроенные методы (embedded methods), когда отбор признаков производится неотделимо от процесса обучения модели (основным алгоритмом является регуляризация – см. метод лассо в разделе 4.2).

В адаптационных методах требуется регрессионная модель (или классификатор), которая используется как черный ящик, возвращая признаки, ранжированные по какому-нибудь удобному критерию – см. рис. 4.15. По практическим соображениям эта модель должна быть в вычислительном отношении быстрой, эффективной и простой, а также мало зависящей от параметров пользователя. Пакет `Boruta` (бог леса в славянской мифологии), включающий одноименную функцию, реализует адаптационный алгоритмдля модели случайного леса (Kursa, Rudnicki, 2010).

Как и функция `varImp()` из пакета `caret`, функция `Boruta()` оценивает меру информативности каждой переменной в виде дополнительной ошибки регрессии, вызванной исключением этой переменной из модели. Среднее μ этой дополнительной ошибки и его стандартное отклонение σ рассчитываются по всем деревьям в лесу, которые используют оцениваемый признак для прогнозирования. Оценка $Z = \mu/\sigma$ может непосредственно использоваться для ранжирования признаков, однако она не является мерой статистической значимости, поскольку не распределена нормально.

Для того, чтобы оценить, является ли ценность признака существенной, а не обусловленной случайными флуктуациями (т.е. проверить гипотезу H_0: $Z = 0$), алгоритм `Boruta` использует внешнее дополнение, полученное в ходе рандомизации. Исходная таблица переменных расширяется таким образом, что в пару каждому предиктору создается соответствующий

"теневой" (shadow) признак, вектор которого получен случайным перемешиванием значений основного признака между строками. Для таких признаков корреляция с откликом отсутствует. Далее запускается алгоритм множественного построения моделей с использованием этой вдвое расширенной таблицы и вычисляется ценность всех признаков *Z*.

Информативная ценность теневого признака может отличаться от нуля только из-за случайных флуктуаций, поэтому множество значений *Z* теневых признаков служит эталоном того, чтобы решить, какие признаки действительно информативны. Для этого вычисляется "теневой порог" MZSA (maximum *Z* score among shadow attributes) и признаки, для которых Z > MZSA объявляются *значимо важными* (important), в то время как остальные – *незначимыми* (unimportant). Поскольку `Boruta` – высокозатратный с вычислительной точки зрения алгоритм и не всегда удается в условиях сокращенного числа итераций Random Forrest достичь полной ясности, некоторые признаки могут быть обозначены как *неопределенные* (tentative).

Используем метод `Boruta` для оценки важности предикторов, определяющих обилие водорослей в реках разного типа.

```
load(file="algae.RData") # Загрузка таблицы algae - раздел 4.1
library(Boruta)
algae.mod <- as.data.frame(model.matrix(a1 ~ .,
                     data=algae[,1:12])[,-1])
algae.mod <- cbind(algae.mod, a1=algae$a1)
algae.Boruta <- Boruta(a1 ~ ., data = algae.mod,
             doTrace = 2, ntree = 500)
getConfirmedFormula(algae.Boruta)
```

```
Boruta performed 99 iterations in 1.376562 mins.
 8 attributes confirmed important: Chla, Cl, mxPH, NH4, NO3,
                       Sizesmall, oPO4, PO4, ;
 5 attributes confirmed unimportant: seasonspring,
           seasonsummer, seasonwinter, sizemedium, speedlow;
 2 tentative attributes left: mnO2, speedmedium;
a1 ~ sizesmall + mxPH + Cl + NO3 + NH4 + oPO4 + PO4 + Chla
```

Таким образом, в результате 99 итераций создания моделей Random Forrest по 500 деревьев в каждой статистически значимыми были признаны 8 переменных, которые были включены в объект `formula`. Статистические показатели важности признаков можно получить в виде таблицы:

```
attStats(algae.Boruta)
```

```
                meanImp   medianImp     minImp     maxImp   normHits  decision
seasonspring  0.3855680  0.09735517 -0.8022415  2.1263346 0.01010101  Rejected
seasonsummer -0.3086411  0.05693780 -2.1205250  0.9013821 0.00000000  Rejected
seasonwinter -0.4294279 -0.45213668 -1.8695717  1.7123137 0.01010101  Rejected
sizemedium    1.0810475  0.92457355 -0.5714529  3.0446745 0.04040404  Rejected
sizesmall     3.4921206  3.52485556  0.5738518  5.9942249 0.73737374 Confirmed
speedlow      1.5411720  1.51556358 -0.8321455  3.6901298 0.16161616  Rejected
speedmedium   1.9939529  2.00780207 -0.4444101  3.6953856 0.42424242 Tentative
mxPH          4.6936750  4.65861782  2.6808697  6.7698339 0.90909091 Confirmed
mnO2          2.2344718  2.34487402 -0.3452873  4.7319744 0.39393939 Tentative
Cl           12.2215939 12.31395995 10.0974482 15.2918406 1.00000000 Confirmed
NO3           4.5193051  4.55940735  1.8457780  7.5044228 0.90909091 Confirmed
NH4          11.4733743 11.46492878  9.3601708 13.2503691 1.00000000 Confirmed
```

```
oPO4        14.0490915 13.97411465 12.2834284 16.0710432 1.00000000 Confirmed
PO4         16.0725740 15.97896519 14.2029194 18.3010180 1.00000000 Confirmed
Chla         7.5156806  7.57219524  5.4705756  9.8583923 1.00000000 Confirmed
```

Разумеется, более наглядно результаты выглядят на диаграмме (рис. 4.19). К сожалению, разработчики пакета сделали трудночитаемым перечень предикторов по оси X, и нам пришлось немного повозиться, чтобы исправить этот недостаток:

```
plot(algae.Boruta, xlab = "", xaxt = "n")
lz<-lapply(1:ncol(algae.Boruta$ImpHistory),function(i)
   algae.Boruta$ImpHistory[is.finite(algae.Boruta$ImpHistory[,i])
                           ,i])
names(lz) <- colnames(algae.Boruta$ImpHistory)
Labels <- sort(sapply(lz,median))
axis(side = 1,las=2,labels = names(Labels),
     at = 1:ncol(algae.Boruta$ImpHistory), cex.axis = 0.7)
```

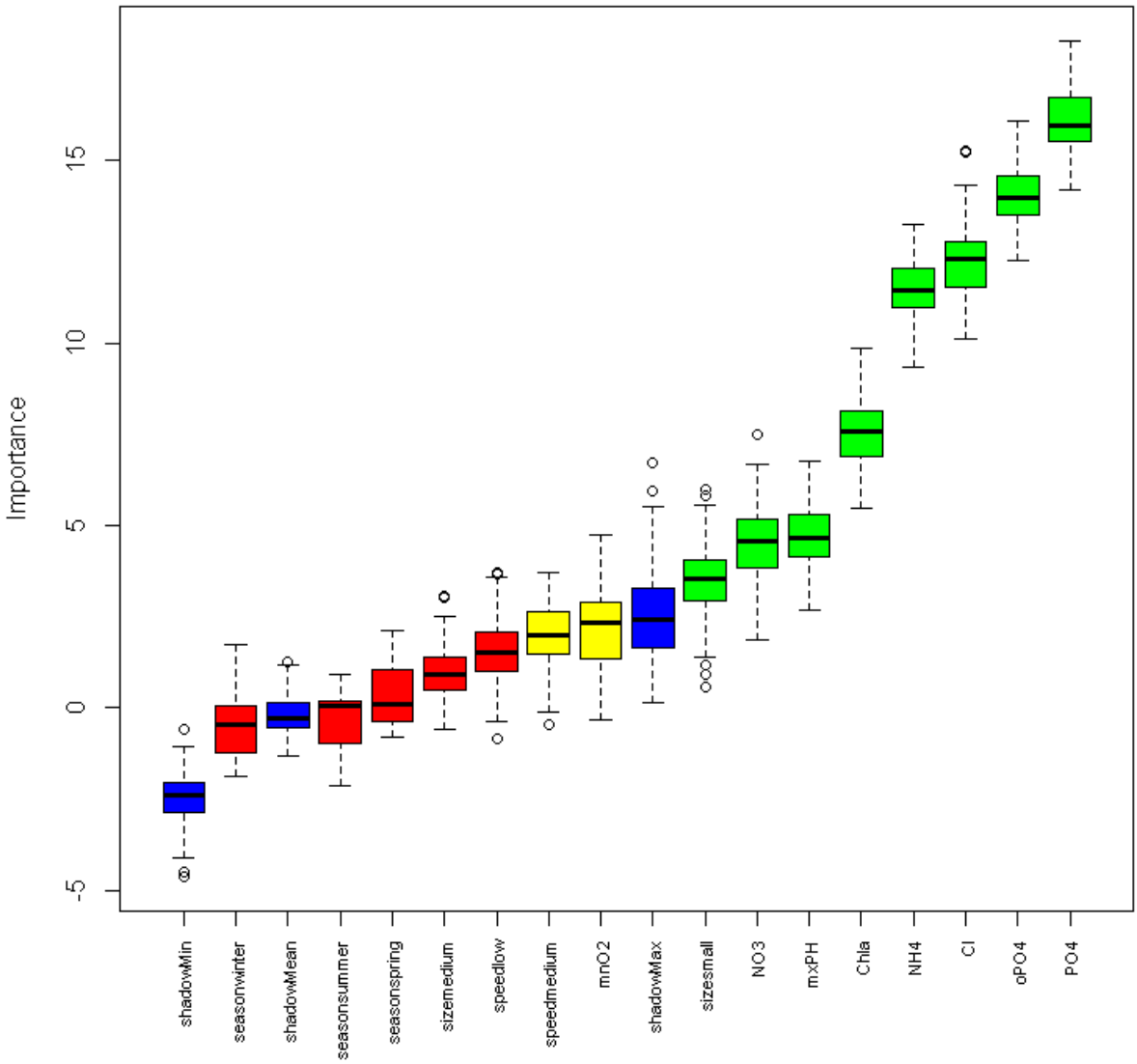


Рис. 4.19. Ранжирование предикторов с использованием алгоритма `Boruta`; синим цветом показана важность для значений теневых признаков

Представленный ранжированный перечень предикторов достаточно близок (хотя и в разной степени) наборам информативных переменных,

сформированным другими методами селекции в разделах выше. Однако оценки значимости придают алгоритму `Boruta` несомненный авторитет.

4.6. Деревья регрессии с многомерным откликом

В разделе 4.3 мы рассмотрели деревья CART, прогнозирующие конкретное значение одной зависимой переменной. Развитием этих идей являются деревья многомерной классификации и регрессии (MRT, Multivariate Regression Trees – De'Ath, 2002). Если в случае обычной регрессии строится модель, прогнозирующая значения одномерного вектора, то многомерный отклик задается в виде двумерной таблицы, содержащей несколько столбцов наблюдаемых признаков. Ставится задача оценить, какие предикторы и в какой степени влияют на совокупную изменчивость количественных соотношений между отдельными компонентами отклика.

Как и в одномерном случае, деревья MRT формируются в результате рекурсивной процедуры разделения строк таблицы данных на гомогенные подмножества, которая реализуется с использованием набора внешних количественных или категориальных независимых переменных **X**. "Листьями" полученного дерева являются кластеры объектов, скомпонованные таким образом, чтобы минимизировать различия между точками в многомерном пространстве в пределах каждой совокупности.

Искомым критерием, минимизирующим внутригрупповые различия, может быть, например, сумма квадратов отклонений $SS_D = \sum_{ij} (y_{ij} - \bar{y}_j)^2$, где y_{ij} – значение показателя отклика j для наблюдения i; $\bar{y}_j$ – средние значения этого показателя для формируемого кластера, куда включается i-е наблюдение. Геометрически SS_D можно представить как сумму евклидовых расстояний от объединяемых объектов до центра их группировки.

Процедура многомерной классификации состоит из последовательности шагов, на каждом из которых синхронно выполняются следующие действия: (а) бинарное разбиение объектов на группы, обусловленное значением одной из независимых переменных, и (б) перекрестная проверка полученных результатов.

Рассмотрим построение дерева MRT на знакомом нам по предыдущим разделам примере анализа обилия водорослей в зависимости от гидрохимических показателей воды и условий отбора проб в водотоках. Для переменных `a1-a7`, описывающих численности 7 групп водорослей, выполним преобразование Бокса-Кокса и шкалирование, что даст нам возможность соизмерить между собой значения различных показателей и корректно вычислить статистики SS_D:

```
load(file="algae.RData") # Загрузка таблицы algae - раздел 4.1
library(caret)
Transal = preProcess(algae[,12:18],
            method = c("BoxCox", "scale"))
Species = predict(Transal, algae[,12:18])
```

Построение дерева MRT будем осуществлять с использованием функции `mvpart()` из пакета `mvpart`. В левой части формулы, задающей структуру модели, представим матрицу из трансформированных численностей семи групп водорослей `Species`, а в правой части – независимые гидрохимические показатели рек `algae[, 1:11]`:

```
library(mvpart)
spe.mvpart <- mvpart(data.matrix(Species) ~ ., algae[,1:11],
     xv="pick", xval=nrow(Species), xvmult = 1,
         margin=0.08, which=4, bars = TRUE)
```

Здесь параметры `xval` и `xvmult` соответствуют числу блоков и повторностей перекрестной проверки, т.е. при `xval = nrow(Species)` осуществляется скользящий контроль. Аргументы `margin` (ширина полей), `which` (расположение текста) и `bars` (вывод столбиковых диаграмм) задают атрибуты визуализации дерева. Важный параметр `xv` определяет условие выбора числа листьев дерева: при значении `"1se"` оно определяется автоматически по результатам перекрестной проверки, а при `"pick"` размеры дерева можно выбрать интерактивно, выполнив щелчок мышью по следующему графику:

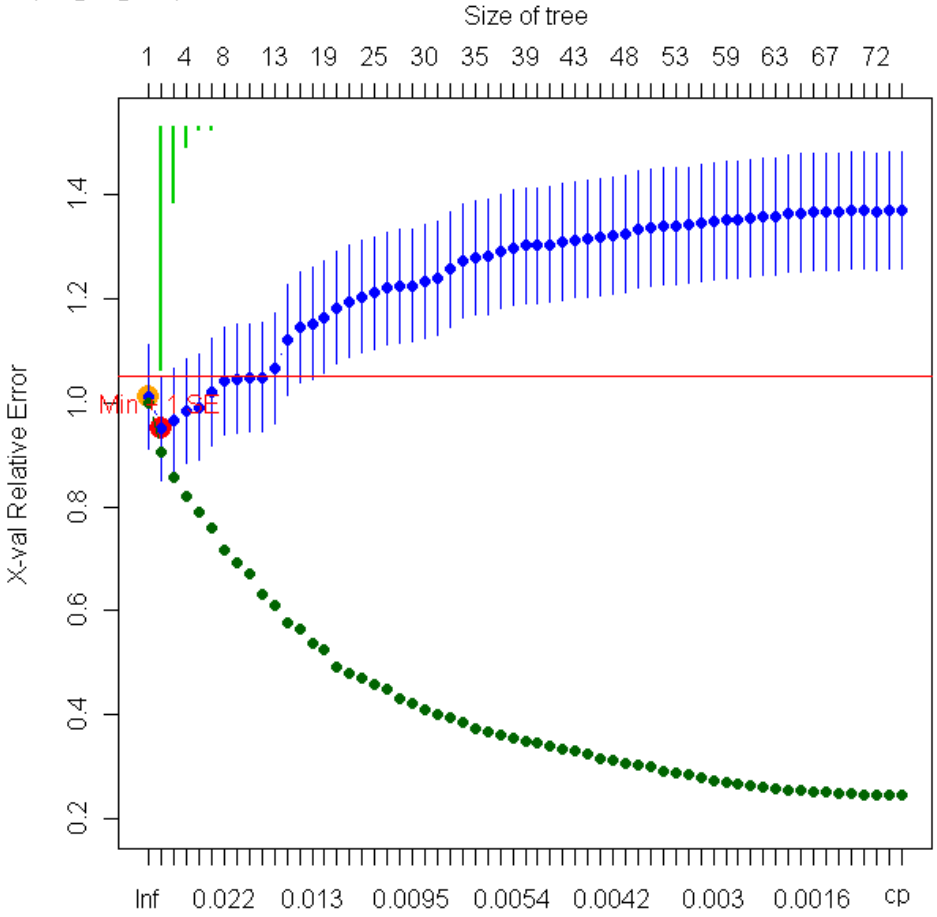


Рис. 4.20. Зависимость относительной ошибки перекрестной проверки от числа узлов дерева

Из представленного графика видно, что при увеличении размеров дерева ошибка на обучающей выборке (зеленые точки) постоянно уменьшается, а ошибка перекрестной проверки – монотонно возрастает (синие точки). Рекомендуемое число листьев в точке их пересечения – 2. Однако если в предыдущих сообщениях нам было важно получить максимальную точность моделей при прогнозировании, то в случае моделей MRT более важна их объясняющая составляющая. Поэтому, чтобы выполнить содержательный анализ зависимости свойств найденных групп водорослей от внешних факторов, разделим все множество наблюдений на 4 кластера и изобразим полученное дерево графически:

```
plot(spe.mvpart) ; text(spe.mvpart)
```

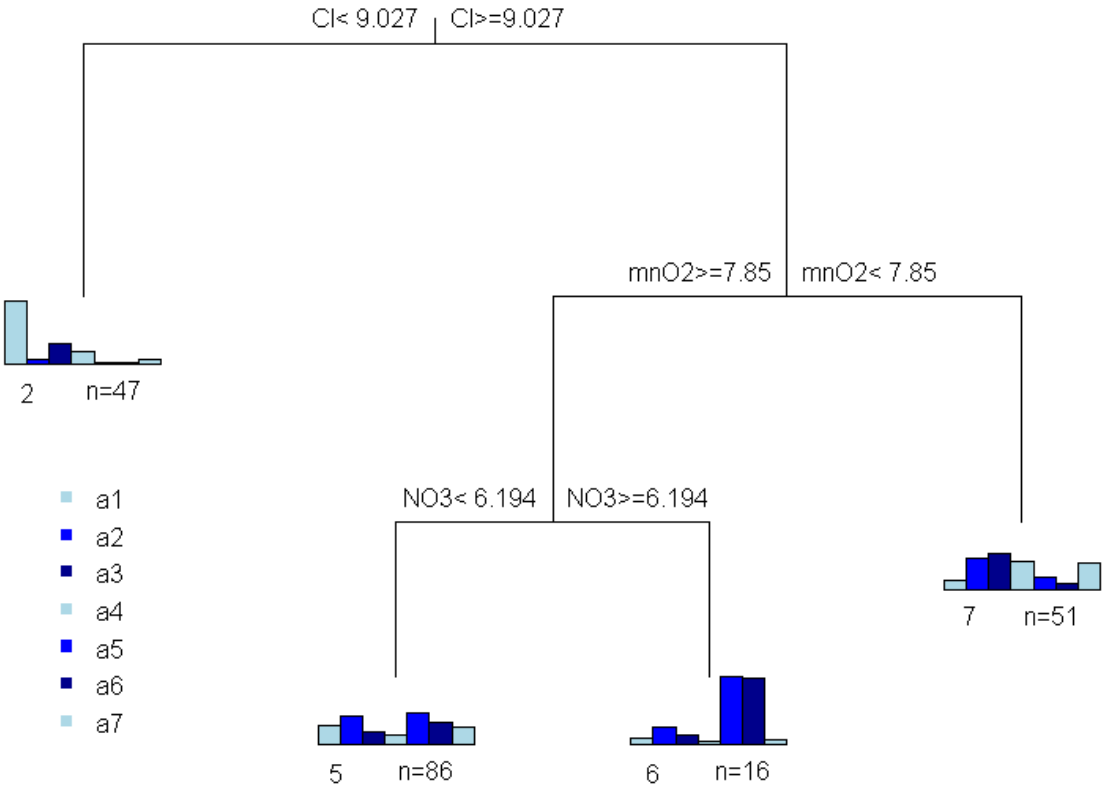


Рис. 4.21. Построенное дерево с многомерным откликом

Рассмотрим более внимательно некоторые нюансы итеративной процедуры построения модели MRT:

```
summary(spe.mvpart)  # Протокол приведен с сокращениями
           CP nsplit rel error    xerror       xstd
1 0.09564230      0 1.0000000 1.0102813 0.09977143
2 0.04933011      1 0.9043577 0.9492214 0.10004843
3 0.03391020      2 0.8550276 0.9647968 0.09961182
4 0.03286729      3 0.8211174 0.9886724 0.10069152

Node number 1: 200 observations,    complexity param=0.0956423
  left son=2 (47 obs) right son=3 (153 obs)
  Primary splits:
      Cl   < 9.0275   to the left,  improve=0.09564230, (0 missing)
      PO4  < 31.97933 to the left,  improve=0.09327022, (0 missing)
      oPO4 < 27.375   to the left,  improve=0.08805637, (0 missing)
```

```
Node number 2: 47 observations
  Means=1.89,0.1381,0.6381,0.3959,0.07981,0.06642,0.1394

Node number 3: 153 observations,    complexity param=0.04933011
  left son=6 (102 obs) right son=7 (51 obs)
  Primary splits:
      mnO2 < 7.85     to the right, improve=0.06307793, (0 missing)
      NH4  < 7588.8   to the left,  improve=0.05783644, (0 missing)
      oPO4 < 448.125  to the left,  improve=0.05719325, (0 missing)

Node number 6: 102 observations,    complexity param=0.0339102
  left son=12 (86 obs) right son=13 (16 obs)
  Primary splits:
      NO3  < 6.194    to the left,  improve=0.07251793, (0 missing)
      mxPH < 7.55     to the right, improve=0.05984036, (0 missing)
      Cl   < 44.0875  to the left,  improve=0.05770594, (0 missing)

Node number 7: 51 observations
  Means=0.307,0.943,1.1,0.8536,0.3646,0.2085,0.7994, Summed MSE=7.241111

Node number 12: 86 observations
  Means=0.5884,0.8402,0.3861,0.3035,0.9321,0.6551,0.5444

Node number 13: 16 observations
  Means=0.2169,0.5259,0.2977,0.1231,2.044,2.013,0.1636
```

На первом шаге `Node number 1` рассматриваются все варианты разбиения исходной выборки на две части при разных опорных значениях независимых факторов и выбирается такой из них (в нашем случае хлориды `Cl < 9.0275`), который в наибольшей мере обеспечивает статистическую гомогенность формируемых кластеров (`complexity param=0.0956423`). Кластер 2 из 47 наблюдений на последующих шагах не разбивается, а 153 наблюдения слева вновь делятся на два подмножества по условию `mnO2 < 7.85` в соотношении 51 + 102. Последнее подмножество по условию `NO3 < 6.194` в свою очередь делится на два кластера, на чем итерации завершаются.

Анализ многомерного отклика сообщества водорослей с помощью деревьев MRT предоставляет исследователю много дополнительных возможностей интерпретации результатов. В первую очередь, это связано с оценкой того, какие компоненты отклика и их ассоциации инициируют разбиение исходной совокупности на узлах дерева и предопределяют состав сформированных подмножеств объектов:

```
# Относительная доля групп водорослей в кластерах
groups.mrt <- levels(as.factor(spe.mvpart$where))
leaf.sum <- matrix(0, length(groups.mrt), ncol(Species))
 colnames(leaf.sum) <- colnames(Species)
 rownames(leaf.sum) <- groups.mrt
for(i in 1:length(groups.mrt)){
   leaf.sum[i,] <- apply(Species[which
   (spe.mvpart$where==groups.mrt[i]),], 2, sum)
 }
leaf.sum
```

```
        a1       a2       a3       a4       a5       a6       a7
2 88.81706  6.492446 29.991924 18.608213  3.750967  3.121594  6.552211
5 50.60338 72.260194 33.201232 26.101301 80.158567 56.334476 46.815353
6  3.47099  8.414790  4.763592  1.969482 32.704161 32.202155  2.617007
7 15.65459 48.094876 56.083746 43.532352 18.594652 10.634001 40.767158
```

Разумеется, эти суммы не соответствуют значениям абсолютных численностей водорослей, поскольку набор исходных данных подвергался преобразованию Бокса-Кокса и шкалированию.

```
opar <- par()
#  Вывод диаграммы типа "разрезанный пирог"
par(mfrow = c(2,2)) ; for(i in 1:length(groups.mrt)){
pie(leaf.sum[i, which(leaf.sum[i,]>0)], radius = 1,
       main = paste("Кл. №", groups.mrt[i])) }
par(opar)
```

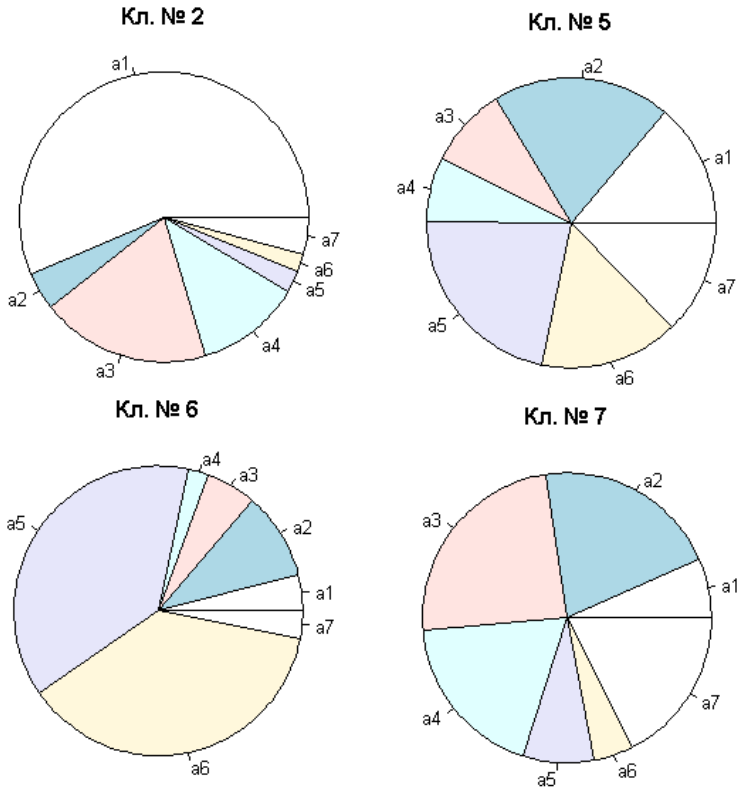


Рис. 4.22. Диаграммы долей средних численностей групп водорослей

Мы получили диаграммы долей усредненных численностей групп водорослей для четырех кластеров, составивших "листья" MRT. Их визуальный анализ показывает, что подмножество 2 составлено с явным доминированием группы a1, подмножество 6 – с доминированием групп a5 и a6, а два остальных кластера не имеют столь характерных признаков. Видимо, перекрестная проверка все же имела все основания рекомендовать нам разбиение только на два кластера.

Можно также оценить, насколько велика неоднородность между кластерами, выполнив редукцию многомерного отклика Species и проецирование данных из многомерного пространства на плоскость с осями из двух первых главных компонент (PC). На сформированной диаграмме

конкретные наблюдения, отнесенные MRT к разным кластерам, выделены отдельные цветами и обозначены контуром, проведенным через крайние точки. В центрах тяжести областей каждого из четырех блоков данных помещен крупный кружок, обозначающий их центроид.

```
#  Вывод диаграммы PCA
rpart.pca(spe.mvpart)
```

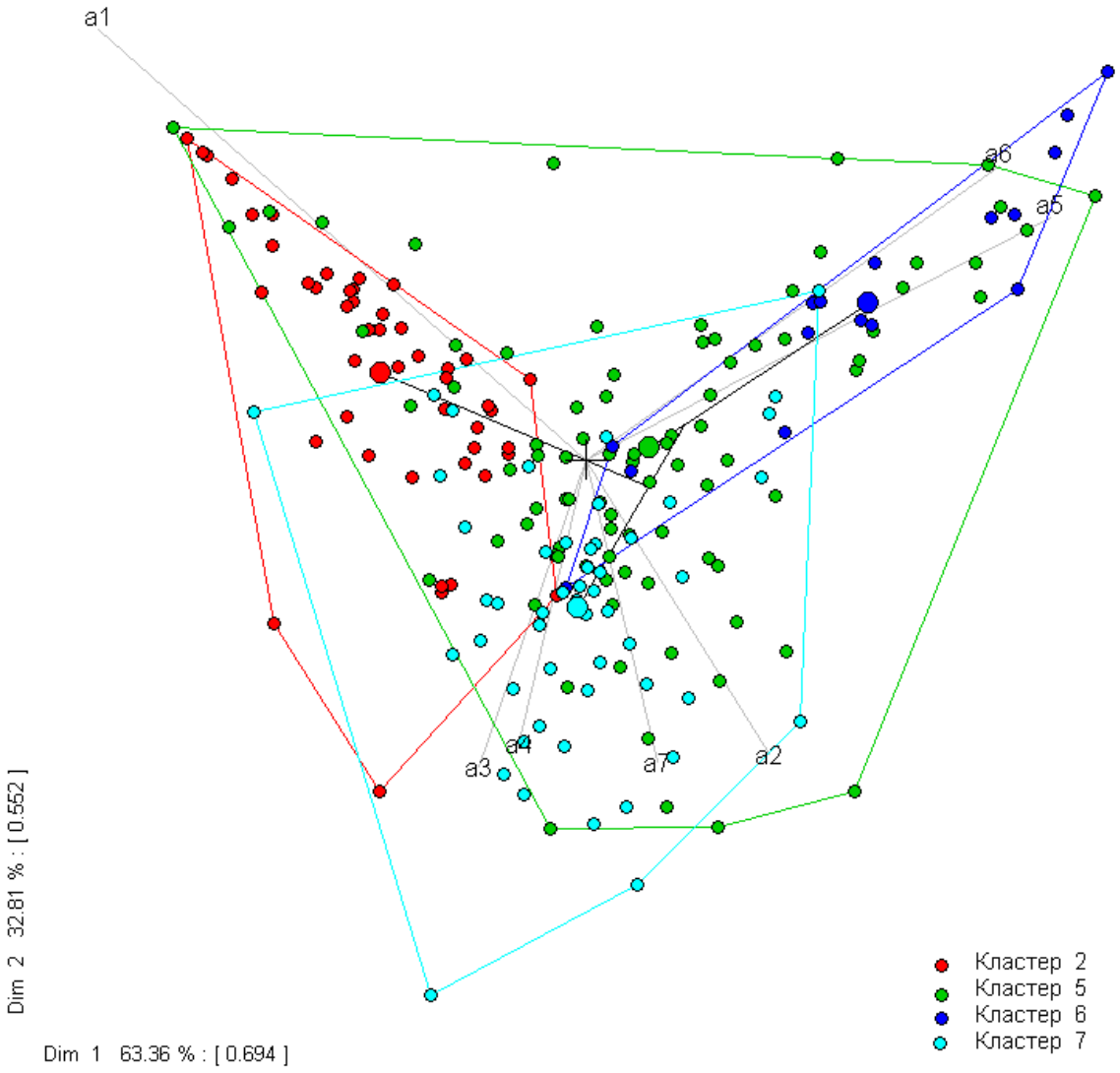


Рис. 4.23. Диаграмма четырех групп водорослей в пространстве двух главных компонент

Из центра координат диаграммы проводятся дополнительные оси ординации, косинусы углов между которыми соответствуют коэффициентам корреляции между каждой парой из 7 групп водорослей. Проекции точек на каждую ось ординации определяют характер распределения показателя по кластерам с разными внешними факторами. Например, на ось `a1` проецируются в основном красные точки кластера 2, объединяющего наблюдения с низким содержанием хлоридов `Cl < 9.0275`.

Настоящий раздел можно рассматривать в качестве "реквиема" по этому весьма оригинальному и практически полезному пакету: `mvpart` был удален из репозитория CRAN и для версий R выше 3.1 отсутствует. Это уже не первый случай, когда авторы перестают поддерживать важные и популярные функциональные компоненты: например, та же участь постигла удачный пакет `lmRepm`, в котором функции построения линейных моделей `lm()` и `anova()` при оценке статистических критериев используют перестановочные алгоритмы и не столь жестко связаны с предположениями о характере распределения данных.

Выхода из подобной ситуации может быть два. Во-первых, ничто не мешает установить на компьютер несколько версий R (так, один авторов этой книги при любой возможности отдает предпочтение старенькой, но симпатичной версии 2.12). Другая возможность – осуществить компиляцию последней версии пакета `mvpart_1.6-2` из архива CRAN в вашей текущей версии R (см. рекомендации в сообщении на http://stackoverflow.com).

5. БИНАРНЫЕ МАТРИЦЫ И АССОЦИАТИВНЫЕ ПРАВИЛА

5.1. Классификация в бинарных пространствах с использованием классических моделей

Большое количество задач связано с нахождением закономерностей в пространствах бинарных переменных: расшифровка геномного кода, анализ пиков на спектрофотометрах, обработка социологических анкет с ответами "Да/Нет" и т.д. Методы анализа таких выборок рассмотрим на не слишком серьезном примере, фигурирующем в книге Дюка и Самойленко (2001).

Предположим, что некая избирательная комиссия озаботилась выделением характерных черт групп электората, голосующих за различные общественные платформы. На представленном рис. 5.1 схематично изображены лица людей, относящихся к двум классам (для определенности пусть это будут "Патриоты" и "Демократы"). Ставится задача найти комбинации признаков, характеризующие это разделение.

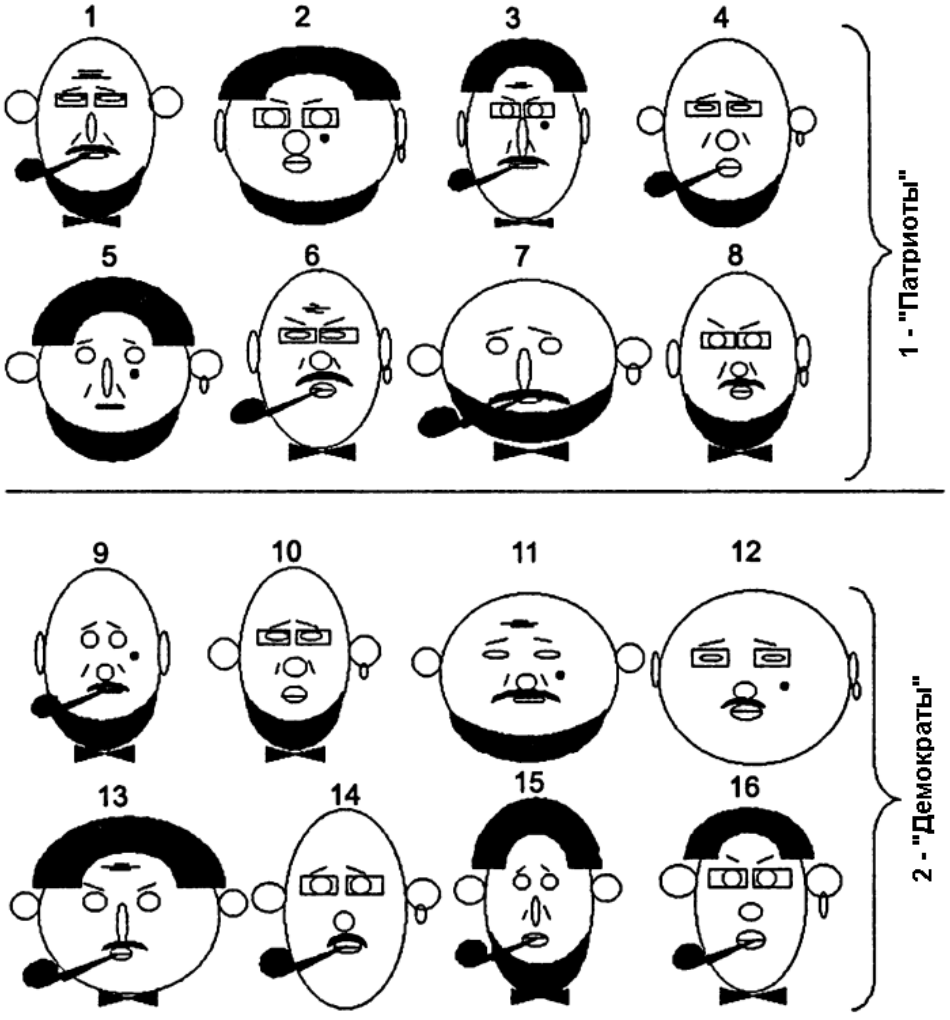


Рис. 5.1. Изображения лиц людей, относящихся к двум различным классам (Дюк, Самойленко, 2001)

Скорее всего, визуальный анализ, проведенный читателем, с большими трудностями позволит определить, чем лица разных классов отличаются друг от друга и что объединяет лица одного класса. Даже такую, на первый взгляд простую, задачу обнаружения скрытых закономерностей лучше поручить компьютерным методам анализа данных.

Прежде всего, выделим признаки, характеризующие изображенные лица. Это следующие характеристики:

x_1 (голова) – круглая (1) или овальная (0);
x_2 (уши) – оттопыренные (1) или прижатые (0);
x_3 (нос) – круглый (1) или длинный (0);
x_4 (глаза) – круглые (1) или узкие (0);
x_5 (лоб) – с морщинами (1) или без морщин (0);
x_6 (носогубная складка) – есть (1) или нет (0);
x_7 (губы) – толстые (1) или тонкие (0);
x_8 (волосы) – есть (1) или нет (0);
x_9 (усы) – есть (1) или нет (0);
x_{10} (борода) – есть (1) или нет (0);
x_{11} (очки) – есть (1) или нет (0);
x_{12} (родинка на щеке) – есть (1) или нет (0);
x_{13} (бабочка) – есть (1) или нет (0);
x_{14} (брови) – подняты кверху (1) или опущены книзу (0);
x_{15} (серьга) – есть (1) или нет (0);
x_{16} (курительная трубка) – есть (1) или нет (0).

Загрузим исходную матрицу данных, соответствующую изображенным лицам, строки которой соответствуют объектам (n = 16), а столбцы – выделенным бинарным признакам (m = 16). Объекты с номерами 1-8 относятся к классу 1, а с номерами 9-16 – к классу 2.

```
DFace <- read.delim(file="Faces.txt",
                    header=TRUE, row.names=1)
head(DFace,8)
```

```
  x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 Class
1  0  1  0  0  1  1  0  0  1   1   1   0   1   1   0   1     1
2  1  0  1  1  0  0  1  1  0   1   1   1   0   0   1   0     1
3  0  0  0  1  1  1  0  1  1   0   1   1   1   0   0   1     1
4  0  1  1  0  0  1  1  0  0   1   1   0   0   1   1   1     1
5  1  1  0  1  0  1  0  1  0   1   0   1   0   1   1   0     1
6  0  0  1  0  1  1  1  0  1   0   1   0   1   0   1   1     1
7  1  1  0  1  0  0  0  0  1   1   0   0   1   1   1   1     1
8  0  0  1  1  0  1  1  0  1   1   1   0   1   0   1   0     1
```

Попытаемся решить поставленную задачу с помощью классических статистических методов. *Множественная логистическая регрессия* – один из распространенных подходов к моделированию значений отклика, заданного альтернативной шкалой 0/1 (Мастицкий, Шитиков, 2015).

Если $\boldsymbol{x}$ – вектор значений m независимых переменных, то вероятность P того, что отклик y примет значение 1, может быть описана моделью

$$P(y = 1|\boldsymbol{x}) \equiv p(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + ... + \beta_m x_m,$$

которая после оценки коэффициентов регрессии β_i будет возвращать искомую вероятность на интервале [0,1]. Для того, чтобы обеспечить линейность функции относительно предикторов x_i и робастность процедуры оценивания коэффициентов, модель переписывают как функцию *логита*:

$$g(y) = \log\left(\frac{p(x)}{1-p(\bar{x})}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_m x_m .$$

Неизвестные параметры модели $\beta_0,\beta_1,\ldots,\beta_m$ обычно находят с использованием функции максимального правдоподобия, а предсказанный результат g(y) трансформируют обратно в вероятности $p(\boldsymbol{x})$ в диапазоне между 0 и 1.

Построенной модели регрессии при использовании представленных данных и полного комплекта переменных m = 16 соответствует значение критерия Акаике AIC = 28. Компактную модель логита, состоящую только из 7 переменных, можно получить в ходе селекции набора информативных переменных на основе стандартной пошаговой процедуры `step()`:

```
logit <- glm((Class-1) ~ ., data=DFace, family=binomial)
logit.step <- step(logit, trace=0)
summary(logit.step)
```

```
Call:
glm(formula = (Class-1) ~ x2 + x3 + x4 + x5 + x7 + x13 + x14,
    family = binomial, data = DFace)
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   -329.06  763966.35       0        1
x2              50.59  190798.82       0        1
x3             101.32  274757.62       0        1
x4             102.33  279732.65       0        1
x5             101.83  278492.43       0        1
x7              50.59  190798.82       0        1
x13             49.90  202466.80       0        1
x14            101.32  274757.62       0        1
    Null deviance: 2.2181e+01  on 15  degrees of freedom
Residual deviance: 2.2719e-10  on  8  degrees of freedom
  AIC: 16
```

Значение информационного критерия значительно снизилось AIC = 16, остаточный девианс практически равен 0, но при этом все оцененные коэффициенты оказались статистически незначимыми. Вероятно, параметрический алгоритм оценки значимости коэффициентов не вполне работоспособен при небольшом числе измерений и/или особой конфигурации данных. Однако насколько эффективен построенный классификатор при предсказании? Рассчитаем число выполненных ошибок и таблицу неточностей:

```
mp <- predict(logit.step, type="response")
pred = ifelse(mp > 0.5, 2, 1)
CTab <- table(Факт=DFace$Class, Прогноз=pred)
Acc = mean(pred == DFace$Class)
paste("Точность=", round(100*Acc, 2), "%", sep="")
barplot(mp-0.5, col = "steelblue", xlab = "Члены выборки",
        ylab = "Вероятности P - 0.5")
```

```
      Прогноз
Факт 1 2
   1 8 0
   2 0 8
[1] "Точность=100.00%"
```

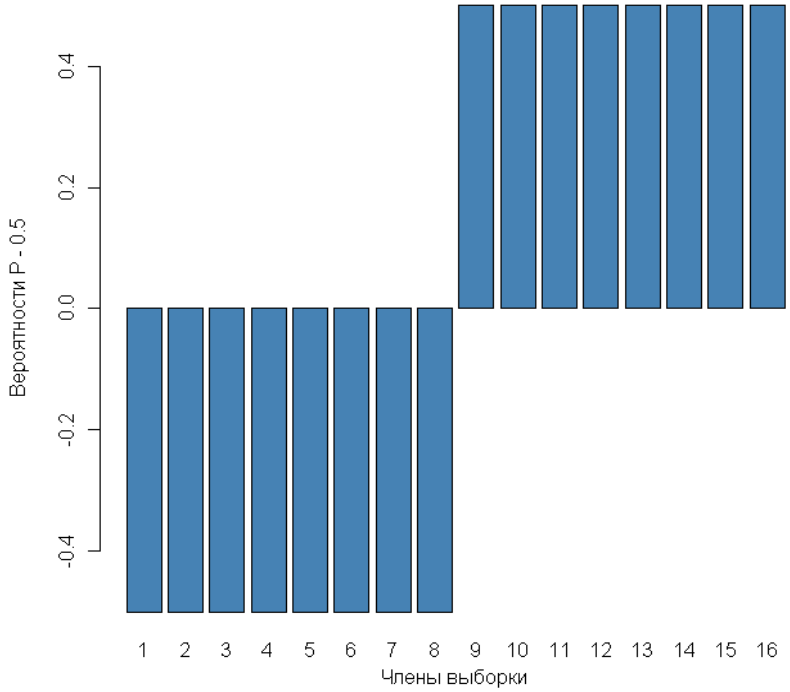


Рис. 5.2. Диаграмма вероятностей, предсказанных логит-регрессией

Все члены обучающей выборки с использованием полученной регрессионной модели были верно классифицированы и с максимальной вероятностью (см. рис. 5.2). Перекрестную проверку построенной модели можно выполнить, например, с использованием функции `cv.glm()` из пакета `boot`. Если параметр `K`, определяющий число блоков разбиения, опущен, то выполняется скользящий контроль:

```
library(boot)
cv.glm(DFace, logit.step)$delta
```

```
[1] 2.564444e-17   2.404152e-17
```

В результате получены среднеквадратичное отклонение по координате логита от проверочных наблюдений до гиперплоскости, найденной на обучающих выборках, а также его скорректированное значение (adjusted cross-validation estimate of prediction error). Поскольку ошибки перекрестной проверки ничтожно малы и все объекты в ее ходе были правильно опознаны, то в целом полученную модель можно считать высоко эффективной.

Мы убедились в том, что логистическая регрессия и функция `glm()` отлично работают с индикаторными переменными. Однако строгие аналитики (Дюк, Самойленко, 2001) считают, что «вряд ли такое правило способно удовлетворить разработчика интеллектуальной системы. Оно формально, не дает нового знания, а попытка дать интерпретацию коэффициентам регрессии приводит к сомнительным выводам. Непонятно, например, почему вес формы носа (x_3) в два раза превышает вес

оттопыренности ушей (x_2) и т. д.». Несмотря на высказанные сомнения, мы достигли желаемой цели – достоверное и эффективное правило классификации построено, а из анализа коэффициентов модели легко сделать вывод, что признаки `x3`, `x4`, `x5` и `x14` склонны иметь демократы, а показатели `x2`, `x7` и `x13` характеризуют патриотические устремления.

Рассмотрим теперь результаты стандартного линейного дискриминантного анализа, не вдаваясь в его исходные статистические предпосылки, смысл и технику вычислений (это будет подробно обсуждаться в следующих главах):

```
library("MASS")
Fac.lda <- lda(DFace[,1:16], grouping = DFace$Class)
pred <- predict(Fac.lda, DFace[,1:16])
table(Факт=DFace$Class, Прогноз=pred$class)
Acc = mean(DFace$Class == pred$class)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
Warning message:
In lda.default(x, grouping, ...) : variables are collinear
     Прогноз
Факт 1 2
   1 6 2
   2 2 6
[1] "Точность=75.00%"
```

Обратим внимание на неприятное сообщение о коллинеарности переменных и мало впечатляющие результаты прогноза.

Стохастические модели многомерных двоичных случайных величин основываются на распределении Бернулли $\mathbf{X} = \{x_1, \ldots, x_m\} \sim \mathrm{Be}_m(\mu, \zeta)$, где μ – вектор математических ожиданий, а ζ включает $(2^m - m - 1)$ параметров, оценивающих взаимодействие между переменными x_j. Как и в одномерном случае, дисперсии полностью определяются через средние $\mathrm{Var}(x_j) = \mu_j (1 - \mu_j)$.

Во многих случаях, особенно для предсказания на основе выборок малого объема, но с большим числом предикторов, представляется разумным следовать "наивному" байесовскому подходу, т.е. игнорировать зависимости между индивидуальными переменными x_j. Тогда количество параметров многомерной модели Бернулли существенно сокращается, и функция объединенной плотности вероятности определяется произведением μ_j и диагональной ковариационной матрицей $\mathrm{Var}(\mathbf{X}) = \mathrm{diag}\{\mu_j (1 - \mu_j)\}$.

С учетом этих положений разработан *метод бинарного дискриминантного анализа* (`binDA` – binary discriminant analysis; Gibb, Strimmer, 2015). Дискриминантная функция, полученная на основе теоремы Байеса, для каждого класса с меткой y имеет следующий вид:

$$d_y(x) = \log \pi_y + \log(\Pr(\boldsymbol{x}|y)) + C,$$

где π_y - априорная вероятность класса y, C – константа, а совокупная условная вероятность предикторов в зависимости от значения отклика вычисляется по формуле

$$\Pr(\boldsymbol{x}|y) = \prod_{j=1}^{m} \begin{cases} 1-\mu_{yj} & \text{если } x_j = 0 \\ \mu_{yj} & \text{если } x_j = 1 \end{cases}.$$

Тестируемый объект относится к тому классу y, дискриминантная функция которого $d_y(x)$ максимальна. Параметром алгоритма является λ – степень ослабления дисбаланса в частотах классов (lambda.freqs – shrinkage intensity for the frequencies). При `lambda.freqs=0` используются эмпирические частоты классов $\pi_y = n_y/n$, а при `lambda.freqs=1` все вероятности принимаются равными.

Важным разделом алгоритма является устранение подмножества предикторов, влияние которых на значение метки y статистически незначимо. Разность между значениями логарифма функции правдоподобия полной модели и нуль-модели без параметров равна относительной энтропии, или дивергенции D Кульбака-Лейблера (Kullback-Leibler). Относительные вклады S_j каждого индивидуального предиктора в величину D определяют меру значимости j-й переменной в результате классификации. Отсюда легко найти матрицу t-значений, соответствующих разностям между общим средним и средними для каждой переменной относительно каждого класса.

Алгоритм `binDA` реализован в пакете `binda` для R. Выполним построение дискриминантной модели и оценим точность прогнозирования на обучающей выборке:

```
library(binda)
Xtrain <- as.matrix(DFace[,1:16])
is.binaryMatrix(Xtrain) # Проверяем бинарность матрицы
Class = as.factor(DFace$Class)
binda.fit = binda(Xtrain,Class,lambda.freq=1)
pred <- predict(binda.fit, Xtrain)
table(Факт=DFace$Class, Прогноз=pred$class)
Acc = mean(DFace$Class!= pred$class)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
     Прогноз
Факт 1 2
   1 7 1
   2 0 8
[1] "Точность=93.75%"
```

При равных апостериорных вероятностях классов 0.5/0.5 объект №4 был ошибочно отнесен ко второму классу.

Выполним теперь оценку информативности предикторов по t-критерию (рис. 5.3):

```
# ранжируем предикторы
binda.x <-  binda.ranking(Xtrain, Class)
plot(binda.x, top=40, arrow.col="blue", zeroaxis.col="red",
ylab="Предикторы", main="")
```

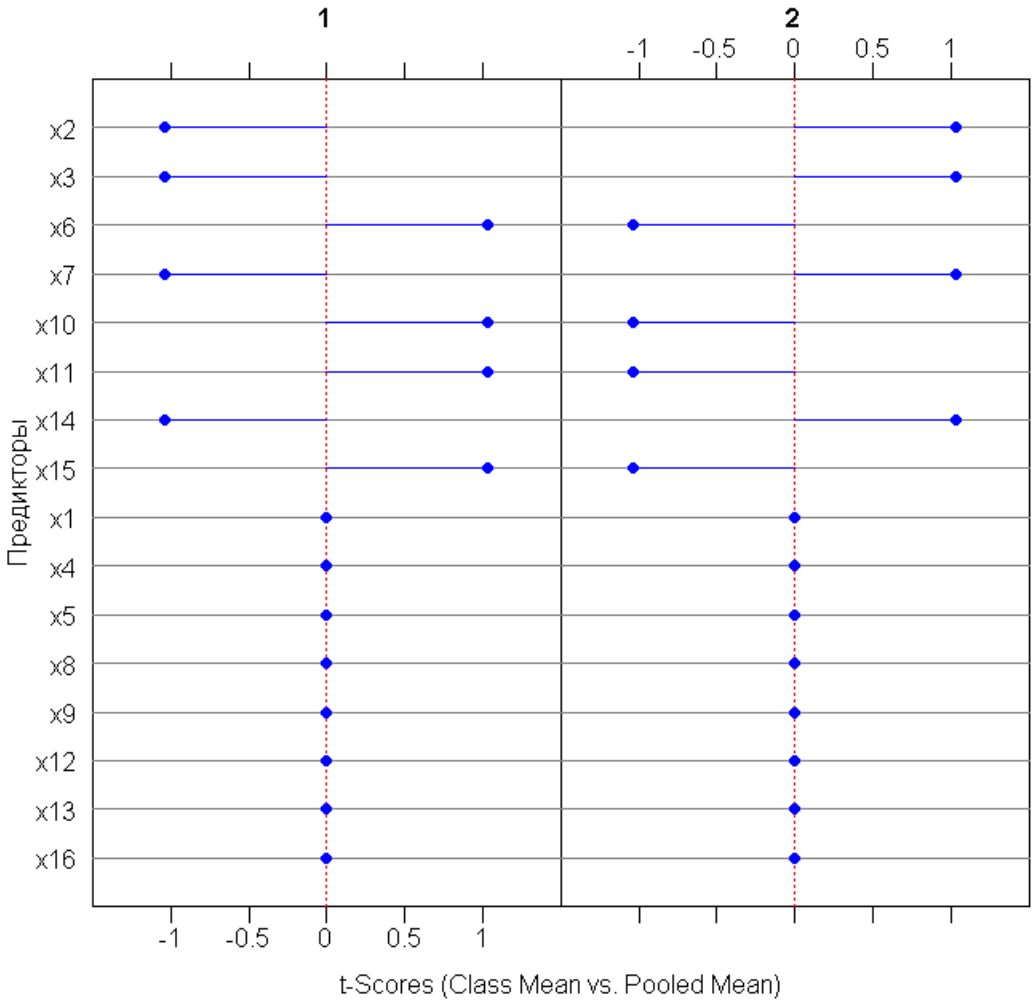


Рис. 5.3. Оценка значимости предикторов модели `binda`

Все переменные разделились на две группы: значимые для классификации (`x2, x3, x6, x7, x10, x11, x14, x15`) со значением $t = \pm 1.032$ и незначимые (`x1, x4, x5, x8, x9, x12, x13, x16`) с $t = 0$. Интересно сравнить этот список со списком информативных предикторов, полученных пошаговой процедурой логистической регрессии.

К сожалению, использование функции `train()` из пакета `caret` как с параметрами `method = "glm", family=binomial`, так и `method = "binda"` привело к неудаче. Видимо при $n = 16$ еще не достигнут порог репрезентативности исходных данных, при котором эта функция начинает стабильно выдавать адекватные результаты.

```
library(caret)
(binda.train <- train(Xtrain, Class, method = "binda"))
```

```
Resampling results across tuning parameters:
  lambda.freqs  Accuracy   Kappa        Accuracy SD  Kappa SD
  0.0           0.3435714  -0.11575021  0.1753883    0.2124598
  0.5           0.3435714  -0.10959636  0.1753883    0.2135358
  1.0           0.3649048  -0.08130732  0.1929750    0.2404408
Accuracy was used to select optimal model using largest value.
The final value used for the model was lambda.freqs = 1.
```

Отрицательная величина Kappa практически означает, что тестируемый классификатор хуже случайного угадывания. Однако итоговая модель подобных неожиданностей не несет:

```
binda.train$finalModel
pred.train <- predict(binda.train, Xtrain)
CTab <- table(Факт=DFace$Class, Прогноз=pred.train)
Acc = mean(DFace$Class == pred.train)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
    Прогноз
Факт 1 2
   1 8 0
   2 2 6
[1] "Точность=87.5%"
```

5.2. Бинарные деревья решений

Как было отмечено в разделе 4.3, использование иерархических древовидных структур (decision trees) является одним из наиболее универсальных и эффективных методов классификации и регрессии (Hunt et al., 1966; Breiman at al., 1984; Loh and Shih, 1997). Классификационные модели деревьев рекурсивно делят набор данных на подмножества, которые являются все более и более гомогенными относительно определенных признаков. Создается решающее правило классификации иерархического типа и формируется ассоциативный логический ключ, дающий возможность выполнять распознавание объектов из новых выборок.

Самым простым, частным случаем деревьев решений являются деревья бинарной классификации, в узлах которых ветвление может вестись только в двух направлениях, т.е. осуществляется выбор из двух альтернатив ("да" и "нет"). Создание такой дихотомической классификационной модели осуществляется с использованием алгоритма ID3 (Interactive Dichotomizer).

В основе алгоритма (Quinlan, 1986) лежит циклическое разбиение обучающей выборки на классы в соответствии с переменной, имеющей наибольшую "классифицирующую силу". Каждое подмножество наблюдений (объектов), выделяемое такой переменной, вновь разбивается на классы с использованием следующей переменной с наибольшей классифицирующей способностью и т.д. Разбиение заканчивается, когда в итоговом подмножестве оказываются объекты лишь одного класса. Пути движения по полученному дереву решений с верхнего уровня на самые нижние определяются логическими правилами в виде цепочек конъюнкций.

При выборе критерия, оценивающего классифицирующую силу, обычно используют теоретико-информационный подход. Мера информационного выигрыша *IG* (Information Gain) для разбиения *A* определяется как разность энтропии Шеннона $H(S)$ для исходного набора данных и суммы энтропий всех фрагментов данных после разбиения $H(S_v)$ с соответствующим весом в зависимости от размера каждой части:

$$IG\,(S, A) = H(S) - \sum_{v \in V} \frac{|\,S_v\,|}{|\,S\,|} H(S_v),$$

где v – выбранное опорное значение переменной V, использованное для разбиения A. Если разделение на классы выполнено оптимально, то каждое подмножество S_v будет иметь небольшую энтропию, и, следовательно, значение IG будет максимальным.

На языке R этот критерий может быть рассчитан следующим образом:

```
Entropy <- function( vls ) {
  res <- vls/sum(vls)*log2(vls/sum(vls)) ; res[vls == 0] <- 0
  -sum(res)
}
InformationGain <- function( tble ) {
  entropyBefore <- Entropy(colSums(tble))
  s <- rowSums(tble)
  entropyAfter <- sum(s/sum(s)*
         apply(tble, MARGIN = 1, FUN = Entropy ))
  informationGain <- entropyBefore - entropyAfter
  return (informationGain)
}
```

Для реализации алгоритма ID3 используем компоненты пакета `data.tree`: объект класса `tree` (дерево) и метод `AddChild` , добавляющий к выстраиваемому дереву очередной дочерний узел (http://ipub.com/wp-content):

```
IsPure <- function(data) {
              length(unique(data[,ncol(data)])) == 1
}
TrainID3 <- function(node, data) {
      node$obsCount <- nrow(data)
  #  если текущий набор данных принадлежит к одному классу, то
  if (IsPure(data)) {
    #создается лист дерева с экземплярами этого класса
    child <- node$AddChild(unique(data[,ncol(data)]))
    node$feature <- tail(names(data), 1)
    child$obsCount <- nrow(data)
    child$feature <- ''
  } else {
    # рассчитывается вектор информационных выигрышей IG
    ig <- sapply(colnames(data)[-ncol(data)],
           function(x) InformationGain(
             table(data[,x], data[,ncol(data)])))
    #  выбирается значение признака с наибольшей величиной IG
    feature <- names(which.max(ig))
    node$feature <- feature
    # создается подмножества данных на основе этого значения
    childObs <- split(data[, names(data) != feature,
          drop = FALSE], data[ ,feature], drop = TRUE)
    #  создаются дочерние узлы дерева с именем признака
    for(i in 1:length(childObs)) {
      child <- node$AddChild(names(childObs)[i])
    # осуществляется рекурсия алгоритма для дочернего узла
      TrainID3(child, childObs[[i]])
    }
  }
}
```

Обратим внимание, что наша функция `TrainID3` является рекурсивной, т.е. для выполнения итераций вызов функции осуществляется из тела ее самой. Функция завершает работу, если все ветви дерева завершаются листьями с объектами одного класса.

Для иллюстрации работы алгоритма ID3 обратимся к нашему примеру по классификации лиц избирателей (см. рис. 5.1 в разделе 5.1). Для получения осмысленной визуализации дерева перейдем от численных обозначений к предметным названиям признаков и их градаций. В результате получаем следующее дерево решений:

```
DFace <- read.delim(file="Faces.txt",header=TRUE, row.names=1)
DFaceN <-DFace[,-17]; Class <- DFace$Class
DFaceN[DFaceN==1]<- "Да" ; DFaceN[DFaceN==0] <- "Нет"
Class[Class==1] <- "Патриот" ; Class[Class==2] <- "Демократ"
DFaceN <- cbind(DFaceN,Class)
colnames(DFaceN) <- c("голова_круглая","уши_оттопырен",
"нос_круглый","глаза_круглые","лоб_морщины",
"носогубн_складка","губы_толстые","волосы","усы","борода",
"очки","родинка_щеке","бабочка","брови_подняты","серьга",
"курит_трубка","Группа")
library(data.tree)
tree <- Node$new("DFaceN")  # Создаем «пустое» дерево
TrainID3(tree, DFaceN)
print(tree, "feature", "obsCount")
```

```
                      levelName       feature obsCount
1  DFaceN                          уши_оттопырен       16
2   ¦--Да                                борода       10
3   ¦   ¦--Да                      нос_круглый        7
4   ¦   ¦   ¦--Да                 курит_трубка        3
5   ¦   ¦   ¦   ¦--Да                   Группа        1
6   ¦   ¦   ¦   ¦   °--Патриот                       1
7   ¦   ¦   ¦   °--Нет                  Группа        2
8   ¦   ¦   ¦       °--Демократ                     2
9   ¦   ¦   °--Нет                  губы_толстые        4
10  ¦   ¦       ¦--Да                    Группа        1
11  ¦   ¦       ¦   °--Демократ                     1
12  ¦   ¦       °--Нет                   Группа        3
13  ¦   ¦           °--Патриот                     3
14  ¦   °--Нет                           Группа        3
15  ¦       °--Демократ                            3
16  °--Нет                          брови_подняты        6
17      ¦--Да                            Группа        2
18      ¦   °--Демократ                            2
19      °--Нет                           Группа        4
20          °--Патриот                             4
```

Выращивание дерева начинается с переменной, имеющей наибольшее значение критерия IG, каковой оказались оттопыренные уши: ее значению "Да" соответствует группа из 10 объектов, а значению "Нет" – из 6. Если в первой ветви для классификации последовательно привлекаются еще 4 признака (наличие бороды, круглого носа, курительной трубки и толстых губ), то во втором случае безошибочное разбиение сразу происходит при использовании поднятых бровей. Проверим теперь, насколько эффективен полученный классификатор на обучающей выборке:

```
Predict <- function(tree, features) {
  if (tree$children[[1]]$isLeaf)
             return (tree$children[[1]]$name)
  child <- tree$children[[features[[tree$feature]]]]
  return ( Predict(child, features))
}
pred <- apply(DFaceN[,-17],1, function(x) Predict(tree,x))
table(Факт=DFaceN$Группа, Прогноз=pred)
```

```
          Прогноз
Факт       Демократ Патриот
  Демократ        8       0
  Патриот         0       8
```

Предсказание электоральной группы по всей обучающей выборке, как и в случае логистической регрессии, также оказалось безошибочным. Однако нет уверенности, что построенные деревья будут столь же успешны при практической реализации на "свежих" данных. Для оценки эффективности модели выполним перекрестную проверку:

```
Nerr <- 0
for (i in 1:nrow(DFaceN)) {
    tree <- Node$new("DFaceN")
    TrainID3(tree, DFaceN[-i,])
    Nerr = Nerr +
           sum(DFaceN[i,17]!=Predict(tree,DFaceN[i,-17]))  }
 (Nerr/nrow(DFaceN))
```

```
[1] 0.25
```

При выполнении скользящего контроля четыре объекта из 16 оказались неверно предсказанными с помощью бинарного дерева, построенного после исключения тестируемого примера. Считается (Дюк, Самойленко, 2001), что алгоритм ID3 в случае взаимозависимых признаков нередко направляет ход логического вывода по ложному пути и создает лишь иллюзию правильного рассуждения. Но для более обоснованных выводов следует рассматривать результаты тестирования на более серьезных примерах, чем наш.

5.3. Поиск логических закономерностей в данных

Основное требование к математическому аппарату Data Minning заключается не только в эффективности классификаторов, но и в интерпретируемости результатов. Методы поиска логических закономерностей в бинарных матрицах наблюдений (как и в некоторых классических методах многомерного анализа) апеллируют к информации, заключенной не в отдельных признаках, а в различных сочетаниях их значений. Для признаков, измеренных в альтернативных шкалах, их значения $x_i = 0$ или $x_i = 1$ рассматриваются как элементарные события, что позволяет сформулировать решающие правила на простом и понятном человеку языке логических высказываний, таких как

ЕСЛИ {(событие 1) и (событие 2) и ... и (событие *N*)} *ТО* ...

Так, классификация лиц в рассмотренном примере рис. 5.1 может быть произведена, например, с помощью логических высказываний 1 и 2:

1. ЕСЛИ {(голова овальная) и (есть носогубная складка) и (есть очки) и (есть трубка)} ИЛИ {(глаза круглые) и (лоб без морщин) и (есть борода) и (есть серьга)} ТО ("Патриот")
2. ЕСЛИ {(нос круглый) и (лысый) и (есть усы) и (брови подняты кверху)} ИЛИ {(оттопыренные уши) и (толстые губы) и (нет родинки на щеке) и (есть бабочка)} ТО ("Демократ").

Математическая запись этих правил выглядит следующим образом:

$[(x_1=0) \wedge (x_6=1) \wedge (x_{11}=1) \wedge (x_{16}=1)] \vee [(x_4=1) \wedge (x_5=0) \wedge (x_{10}=1) \wedge (x_{15}=1)] \Rightarrow \varpi_1$

$[(x_3=1) \wedge (x_8=0) \wedge (x_9=1) \wedge (x_{14}=1)] \vee [(x_2=1) \wedge (x_7=1) \wedge (x_{12}=0) \wedge (x_{13}=1)] \Rightarrow \varpi_2.$

Здесь значки $\vee$ – конъюнкция ("и"), $\wedge$ – дизъюнкция ("или"), $\Rightarrow$ – импликация ("если, то").

Будем использовать логический метод распознавания, известный под названием алгоритм "Кора", который в свое время широко использовался в геологоразведочных работах и скрининге лекарственных препаратов (Бонгард, 1967; Вайнцвайг, 1973). В детерминистической версии алгоритма обучающая выборка $\boldsymbol{x}$, предварительно разделенная на два класса (1 и 2), многократно просматривается и из всего множества высказываний выделяются так называемые непротиворечивые логические высказывания $\Psi(\boldsymbol{x}, \tau)$, покрывающие все множество примеров. Непротиворечивым высказыванием для каждого класса считается конъюнкция, которая встречается с вероятностью не менее τ только в одном классе и ни разу не встречается в другом.

Представленный ниже скрипт выполняет перебор всех возможных комбинаций столбцов x_i, $i = 1, \ldots, m=16$, по 2, 3 и `maxKSize` = 4 с использованием функции `combn()`. Для каждой комбинации столбцов перебираются все возможные варианты событий ($x_i = 0$ или $x_i = 1$). Для сокращения объема вычислений подмножества исходной матрицы трансформировались в векторы символьных бинарных переменных, а комбинации значений x_i выражались наборами "битовых масок" (например, `"000", "001", "010", "011", "100", "101", "110", "111"`).

В ходе перебора конъюнкции, отобранные по совокупности условий, будем сохранять в трех глобальных объектах класса `list`, для чего используется оператор глобального присваивания "<<-". Определим предварительно несколько функций:

```
# Преобразование числа в набор битов (5 -> "0101")
number2binchar <- function(number, nBits) {
      paste(tail(rev(as.numeric(intToBits(number))), nBits),
        collapse="")
}
#  Поиск конъюнкций по заданной битовой маске
MaskCompare <- function(Nclass, KSize, BitMask,
                    vec_pos, vec_neg, ColCom) {
```

```
    nK <- sapply(BitMask, function(x) {
       if (sum(x==vec_neg)>0) return (0)
       if (minNum>(countK= sum(x==vec_pos))) return (0)
       #  Сохранение конъюнкции в трех объектах list
       Value.list[[length(Value.list)+1]] <<-
            list(Nclass=Nclass, KSize=KSize, countK=countK,
                 Bits=x)
       ColCom.list[[length(ColCom.list)+1]] <<- list(ColCom)
       RowList.list[[length(RowList.list)+1]] <<-
            list(which(vec_pos %in% x))
               return (countK) } )
  }
```

Зададим минимальную частоту встречаемости конъюнкций `minNum = 4` (т.е. $\tau = 50\%$) и выполним формирование всех логических правил для рассматриваемого примера:

```
  DFace <- read.delim(file="Faces.txt",header=TRUE, row.names=1)
  maxKSize <- 4
  minNum <- 4
  #  Списки для хранения результатов
  Value.list <- list()   # Nclass, KSize, BitMask, countK
  ColCom.list <- list()  # Наименования переменных ColCom
  RowList.list <- list()  # Номера индексов строк RowList
  #  Перебор конъюнкций разной длины
  for (KSize in 2:maxKSize) {
     BitMask <- sapply(0:(2^KSize-1),
              function(x) number2binchar(x,KSize))
     cols <-  combn(colnames(DFace[,-17]),KSize)
       for (i in 1:ncol(cols))  {
          SubArr <- DFace[,(names(DFace) %in% cols[,i])]
          vec1 <- apply(SubArr[DFace$Class==1,],1,
             function (x) paste(x,collapse=""))
          vec2 <- apply(SubArr[DFace$Class==2,],1,
             function (x) paste(x,collapse=""))
          MaskCompare(1, KSize, BitMask, vec1, vec2, cols[,i])
          MaskCompare(2, KSize, BitMask, vec2, vec1, cols[,i])
       }
  }
  #  Создание результирующей таблицы
  DFval = do.call(rbind.data.frame, Value.list)
  nrow = length(Value.list)
  DFvar <- as.data.frame(matrix(NA, ncol=maxKSize+1, nrow=nrow,
                dimnames = list(1:nrow, c(
                paste("L", 1:maxKSize, sep=""),"Объекты:"))))
  for (i in 1:nrow) {
      Varl <- unlist(ColCom.list[[i]])
      DFvar[i, 1:length( Varl)] <- Varl
      Objl <- unlist(RowList.list[[i]])
      DFvar[i, maxKSize+1] <- paste(Objl,collapse=" ")
                    }
  DFvar[is.na(DFvar)] <- " "
  DFout <- cbind(DFval, DFvar)
  #  Вывод результатов
  print ("Конъюнкции класса 1") ; DFout[DFout$Nclass==1,]
  print ("Конъюнкции класса 2") ; DFout[DFout$Nclass==2,]
```

Результат, содержащий логические высказывания для каждого класса (`Nclass`) будет выглядеть следующим образом, где `KSize` – длина коньюнкции, `Bits` – ее битовая маска, `L1-L4` – наименования исходных переменных, `countK` – встречаемость конъюнкции на объектах своего класса.

```
[1] "Конъюнкции класса 1"
   Nclass KSize countK Bits L1  L2  L3  L4 Объекты:
1       1     2      4   00 x2 x14          2 3 6 8
2       1     2      4   00 x3  x7          1 3 5 7
7       1     3      4  010 x2 x11 x14      2 3 6 8
10      1     3      4  111 x4 x10 x15      2 5 7 8
12      1     3      4  111 x6  x9 x11      1 3 6 8
13      1     3      4  111 x6 x11 x16      1 3 4 6
14      1     3      4  111 x9 x11 x13      1 3 6 8
16      1     4      4 0111 x1  x6  x9 x11  1 3 6 8
17      1     4      4 0111 x1  x6 x11 x16  1 3 4 6
18      1     4      4 0111 x1  x9 x11 x13  1 3 6 8
23      1     4      4 1011 x4  x5 x10 x15  2 5 7 8
25      1     4      4 1111 x6  x9 x11 x13  1 3 6 8
26      1     4      4 0101 x8  x9 x12 x13  1 6 7 8

[1] "Конъюнкции класса 2"
   Nclass KSize countK Bits  L1  L2  L3  L4 Объекты:
3       2     2      4   00  x6 x10          4 5 6 8
4       2     2      4   00 x11 x15          1 3 5 7
5       2     3      4  111  x2  x4  x7      5 6 7 8
6       2     3      4  111  x2  x7 x13      2 5 7 8
8       2     3      4  111  x3  x9 x14      1 3 4 6
9       2     3      4  111  x4  x7 x16      5 6 7 8
11      2     3      4  010  x6  x7 x10      4 5 6 8
15      2     4      4 0101  x1  x4  x5 x16  1 6 7 8
19      2     4      4 1110  x2  x4  x7 x12  5 6 7 8
20      2     4      4 1111  x2  x4  x7 x16  5 6 7 8
21      2     4      4 1101  x2  x7 x12 x13  2 5 7 8
22      2     4      4 1011  x3  x8  x9 x14  1 3 4 6
24      2     4      4 1101  x4  x7 x12 x16  5 6 7 8
```

Дальнейшая работа с выделенными логическими высказываниями сводится к следующему (реализация алгоритма в R находится в стадии доработки). Из генерируемого списка необходимо исключить подчиненные (или дочерние) конъюнкции, включающие более короткие претенденты: например, для класса 1 высказывание №7 является дочерним по отношению к конъюнкции №1 и должно было быть удалено. После такой операции поглощения ликвидируется избыточность решающего правила, в котором остаются конъюнкции минимального ранга, содержащие выделенные закономерности в концентрированной форме.

Хорошим правилом было бы также исключить высказывания, которые по определенным критериям считаются "предрассудками". К ним относятся конъюнкции, не связанные с объективным правилом классификации, но, в силу ограниченности выборки, получившие хорошие оценки на обучении. Для выделения признаков, склонных к предрассудкам, выполняют бутстреп-процедуру, в которой обучающая выборка многократно случайным образом разбивается на классы. Разбитая таким образом выборка является тестом,

позволяющим присвоить каждому исходному признаку штрафные баллы за склонность к предрассудкам. Конъюнкции, набравшие сверхнормативное количество штрафных очков, из решающего правила исключаются.

Описанием каждого класса является логическая сумма (дизъюнкция) некоторого количества непротиворечивых и продуктивных конъюнкций, прошедших описанные выше этапы отбора. Комбинация этих логических высказываний представляет собой своеобразную мозаично-фрагментарную разделяющую поверхность специального типа (в отличии, например, от линейной плоскости логита). Более общая версия алгоритма "Кора" предполагает выделение логических закономерностей для K классов, $K > 2$. Среди конъюнкций выделяются те, которые верны на обучающей выборке чаще, чем некоторый порог τ_1, для одного из классов и не характерны для любого другого (верны реже, чем в доле случаев τ_2).

Существует возможность использовать сгенерированные конъюнкции для экзамена тестируемых примеров по принципу голосования. Чтобы классифицировать новое наблюдение $\boldsymbol{x}$, подсчитывается число отобранных конъюнкций L_k, характерных для каждого k-го класса, которые верны для тестируемого бинарного вектора. Если L_k является максимальным из всех, то принимается решение о принадлежности объекта k-му классу.

Алгоритм "Кора", как и другие логические методы распознавания образов, является достаточно трудоемким, поскольку при отборе конъюнкций необходим полный или частично направленный перебор. Здесь может быть полезным использование процедур эволюционного поиска: генетический алгоритм (genetic algorithm), случайный поиск с адаптацией (СПА – см. книгу Г.С. Лбова, 1981 на сайте math.nsc.ru) и др.

5.4. Алгоритмы выделения ассоциативных правил

Ассоциативные правила представляют собой механизм нахождения логических закономерностей между связанными элементами (событиями или объектами). Пусть имеется $\mathbf{A} = \{a_1, a_2, a_3, \ldots, a_n\}$ – конечное множество уникальных элементов (list of items). Из этих компонентов может быть составлено множество наборов **T** (sets of items), т.е. $\mathbf{T} \subseteq \mathbf{A}$.

Ассоциативные правила $\mathscr{A} \rightarrow \mathscr{C}$ имеют следующий вид:

если <условие> *то* <результат>,

где, в отличие от деревьев классификации, <условие> – не логическое выражение, а набор объектов из множества **A**, с которыми связаны (ассоциированы) объекты того же множества, включенные в <результат> данного правила. Например, ассоциативное правило

если (смородина, тля) *то* (муравьи)

означает, что если на кусте смородины встретилась тля, то ищи поблизости и муравьев.

Понятие «вид элемента a_k» легко может быть обобщено на ту или иную его категорию или вещественное значение, т.е. концепция ассоциативного

анализа может быть применена для комбинаций любых переменных. Например, при прогнозировании погоды одно из ассоциативных правил может выглядеть так:

$$\text{″направление ветра″} = \text{NNW} \rightarrow \text{″завтра будет дождь″} = \text{TRUE}$$

Выделяют три вида правил:

- *полезные правила*, содержащие действительную информацию, которая ранее была неизвестна, но имеет логическое объяснение;
- *тривиальные правила*, содержащие действительную и легко объяснимую информацию, отражающую известные законы в исследуемой области, и поэтому не приносящие какой-либо пользы;
- *непонятные правила*, содержащие информацию, которая не может быть объяснена (такие правила или получают на основе аномальных исходных данных, или они содержат глубоко скрытые закономерности, и поэтому для интерпретации непонятных правил нужен дополнительный анализ).

Поиск ассоциативных правил обычно выполняют в два этапа:

- в пуле имеющихся признаков **A** находят наиболее часто встречающиеся комбинации элементов **T**;
- из этих найденных наиболее часто встречающихся наборов формируют ассоциативные правила.

Для оценки полезности и продуктивности перебираемых правил используются различные частотные критерии, анализирующие встречаемость кандидата в массиве экспериментальных данных. Важнейшими из них являются *поддержка* (support) и *достоверность* (confidence). Правило $\mathcal{A} \rightarrow \mathcal{C}$ имеет поддержку s, если оно справедливо для s% взятых в анализ случаев:

$$\text{support}(\mathcal{A} \rightarrow \mathcal{C}) = P(\mathcal{A} \cup \mathcal{C}).$$

Достоверность правила показывает, какова вероятность того, что из наличия в рассматриваемом случае условной части правила следует наличие заключительной его части (т.е. из $\mathcal{A}$ следует $\mathcal{C}$):

$$\text{confidence}(\mathcal{A} \rightarrow \mathcal{C}) = P(\mathcal{A} \cup \mathcal{C})/P(\mathcal{A}) = \text{support}(\mathcal{A} \rightarrow \mathcal{C})/\text{support}(\mathcal{A}).$$

Алгоритмы поиска ассоциативных правил отбирают тех кандидатов, у которых поддержка и достоверность выше некоторых наперед заданных порогов: *minsupport* и *minconfidence*. Если поддержка имеет большое значение, то алгоритмы будут находить правила, хорошо известные аналитику или настолько очевидные, что нет никакого смысла проводить такой анализ. Большинство интересных правил находят именно при низком значении порога поддержки. С другой стороны, низкое значение *minsupport* ведет к генерации огромного количества вариантов, что требует существенных вычислительных ресурсов или ведет к генерации статистически необоснованных правил.

В пакете `arules` для R используются и другие показатели – *подъемная сила*, или *лифт* (lift), которая показывает, насколько повышается вероятность нахождения $\mathcal{C}$ в анализируемом случае, если в нем уже имеется $\mathcal{A}$:

$$\text{lift}(\mathcal{A} \rightarrow \mathcal{C}) = \text{confidence}(\mathcal{A} \rightarrow \mathcal{C})/\text{support}(\mathcal{C}),$$

и *усиление* (leverage), которое отражает, насколько интересной может быть более высокая частота $\mathcal{A}$ и $\mathcal{C}$ в сочетании с более низким подъемом

$$\text{leverage}(\mathcal{A} \rightarrow \mathcal{C}) = \text{support}(\mathcal{A} \rightarrow \mathcal{C}) - \text{support}(\mathcal{A}) * \text{support}(\mathcal{C}).$$

Первый алгоритм поиска ассоциативных правил был разработан в 1993 г. сотрудниками исследовательского центра IBM, что сразу возбудило интерес к этому направлению. Каждый год появлялось несколько новых алгоритмов (DHP, Partition, DIC и др.), из которых наиболее известным остался алгоритм "Apriori" (Agrawal, Srikant, 1994).

Пакет `arules` позволяет находить часто встречающиеся сочетания элементов в данных (*frequent itemsets*) и отбирать ассоциативные правила, обеспечивая интерфейс к модулям на языке C, которые реализуют алгоритмы "Apriori" и "Eclat". Так как обычно обрабатываются большие множества наборов и правил, то для уменьшения объёмов требуемой памяти пакет содержит развитый инструментарий преобразования разреженных входных матриц в компактные наборы транзакций (Hahsler et al., 2016; Огнева, 2012).

Для работы с алгоритмами выделения ассоциаций в `arules` реализованы специальные типы данных, относящиеся к объектам трех классов: входной массив транзакций (`transactions`) и на выходе – часто встречающиеся фрагменты данных (`itemsets`) и правила (`rules`).

Объекты класса `transactions` представляют собой специально организованные бинарные матрицы со строками-наборами и столбцами-признаками, содержащие значения элемента 1, если соответствующий признак есть в транзакции, и 0, если он отсутствует. В зависимости от типа данных и способа их загрузки, эти объекты могут иметь разные способы организации и состав дополнительных слотов. В частности, подкласс `itemMatrix` является одновременно средством представления разреженных матриц с использованием функционала пакета `Matrix`. Другим способом формирования экземпляров класса `transactions` является загрузка данных из файла функцией `read.transactions()`.

Для выделения ассоциативных правил вновь обратимся к нашему примеру по классификации лиц избирателей (см. рис. 5.1). Метод `"basket"` функции `read.transactions()` предполагает, что каждая строка в файле представляет собой одну транзакцию, в которой признаки (т.е. их метки) разделены символами `sep` (по умолчанию – запятая). Переформируем исходную таблицу данных в файл необходимого формата:

```
DFace <- read.delim(file="Faces.txt",
                            header=TRUE, row.names=1)
Class <- DFace$Class; DFaceN <-DFace[,-17]
colnames(DFaceN) <- c("голова_круглая","уши_оттопырен",
   "нос_круглый","глаза_круглые","лоб_морщины",
   "носогубн_складка","губы_толстые","волосы","усы","борода",
   "очки","родинка_щеке","бабочка", "брови_подняты","серьга",
   "курит_трубка")
Class[Class==1] <- "Патриот"
Class[Class==2] <- "Демократ"
items_list <- sapply(1:nrow(DFaceN),function(i)
```

```
  paste(c(Class[i], colnames(DFaceN[i,DFaceN[i,]==1])),
              collapse=",", sep="\n"))
  head(items_list)
  write(items_list, file = "face_basket.txt")
```

```
Патриот,уши_оттопырен,лоб_морщины,носогубн_складка,усы,борода,очки,бабочка,брови_подняты,курит_трубка
Патриот,голова_круглая,нос_круглый,глаза_круглые,губы_толстые,волосы,борода,очки,родинка_щеке,серьга
Патриот,глаза_круглые,лоб_морщины,носогубн_складка,волосы,усы,очки,родинка_щеке,бабочка,курит_трубка
Патриот,уши_оттопырен,нос_круглый,носогубн_складка,губы_толстые,борода,очки,брови_подняты,серьга,курит_трубка
Патриот,голова_круглая,уши_оттопырен,глаза_круглые,носогубн_складка,волосы,борода,родинка_щеке,брови_подняты,серьга
Патриот,нос_круглый,лоб_морщины,носогубн_складка,губы_толстые,усы,очки,бабочка,серьга,курит_трубка
```

Загрузим данные из файла и создадим объект `itemMatrix`. Информацию о сформированном массиве транзакций можно получить, выполнив команды `inspect()` и `summary()`.

```
  library("arules")
  trans = read.transactions("face_basket.txt",
                      format = "basket", sep=",")
  inspect(trans)    #  Выводимые данные не показаны
  summary(trans)    #  Выводимые данные показаны частично
```

```
transactions as itemMatrix in sparse format with
 16 rows (elements/itemsets/transactions) and
 18 columns (items) and a density of 0.5555556
```

Для объектов класса `itemMatrix` можно осуществить быструю визуализацию данных или построить график распределения частот встречаемости признаков (рис. 5.4а, б).

```
  image(trans)
  itemFrequencyPlot(trans, support = 0.1, cex.names=0.8)
```

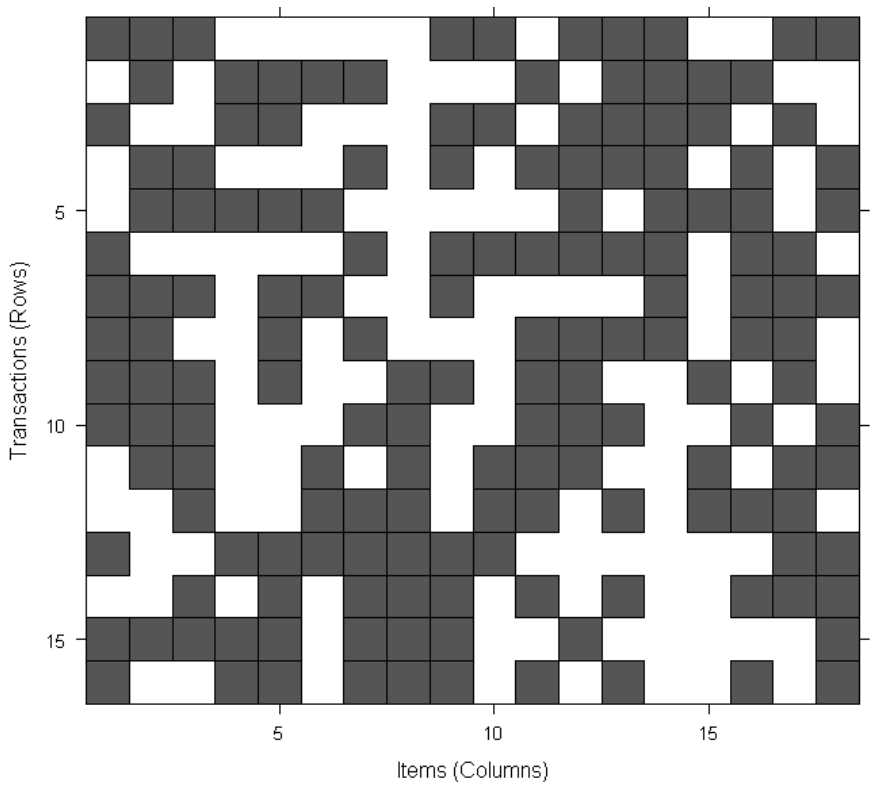


Рис. 5.4а. Матрица признаки/транзакции

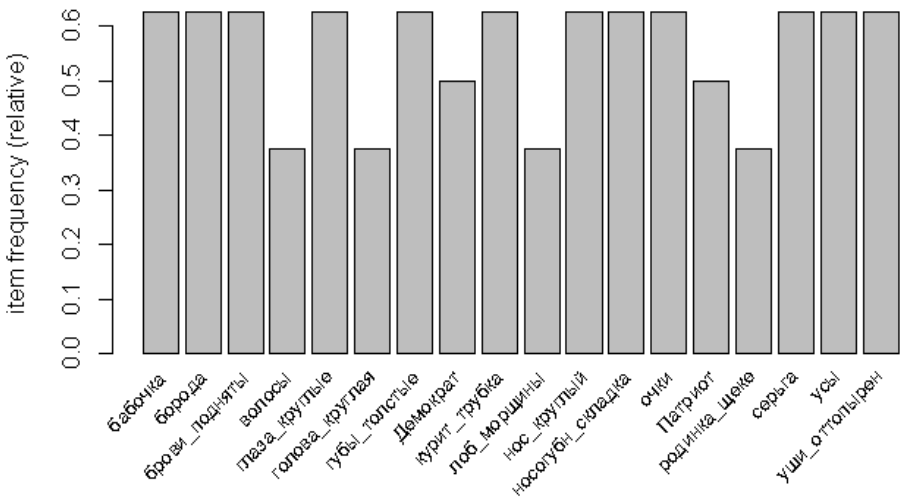


Рис. 5.4б. Частотное распределение встречаемости признаков в транзакциях

Поиск ассоциативных правил является не вполне тривиальной задачей, т.к. с ростом числа элементов в **A** экспоненциально растет число их потенциальных комбинаций. Алгоритм "Apriori" является итерационным: при этом сначала выполняются действия для одноэлементных наборов, затем для 2-х, 3-х элементных и т.д. (т.е. он во многом напоминает алгоритм "Кора").

На первом шаге первой итерации алгоритма подсчитываются одноэлементные часто встречающиеся наборы. Для этого необходимо пройтись по всему массиву данных и подсчитать для них поддержку, т.е. сколько раз набор встречается в имеющемся наборе данных. При последующем поиске *k*-элементных наборов генерация претендентов состоит из двух фаз – формирование кандидатов нового уровня на основе (*k* - 1)-элементных наборов, которые были определены на предыдущей итерации алгоритма, и удаление избыточных кандидатов. После того как найдены все часто встречающиеся наборы элементов, выполняют процедуру непосредственного извлечения правил из построенного хеш-дерева (Зайцев, 2009).

Результатом анализа транзакций с помощью пакета `arules` являются объекты класса `associations`, включающие описания множества отношений между признаками (в виде часто встречающихся фрагментов, или правил), которые отбираются в соответствии с различными перечисленными выше мерами качества. Подкласс `rules` состоит из двух объектов `itemMatrix`, представляющих левую `lhs` (left-hand-side) и правую `rhs` (right-hand-side) сторону правила $\mathscr{A} \rightarrow \mathfrak{C}$, т.е. $\mathscr{A}$ – `lhs`, $\mathfrak{C}$ – `rhs`.

Формирование правил осуществляется функцией `apriori()` с указанием пороговых значений поддержки и достоверности. Функция `summary()` обеспечивает частотный анализ правил по их длине и достигнутым мерам качества:

```
rules <- apriori(trans,
    parameter = list(support = 0.01, confidence = 0.6))
summary(rules)  #  Результаты показаны частично
```

```
set of 48306 rules
rule length distribution (lhs + rhs):sizes
    1     2     3     4     5     6     7     8     9    10
   12   138   916  3892  9860 14560 11814  5526  1428   160
summary of quality measures:
    support          confidence          lift
 Min.   :0.06250   Min.   :0.6000   Min.   :0.960
 1st Qu.:0.06250   1st Qu.:1.0000   1st Qu.:1.600
 Median :0.06250   Median :1.0000   Median :1.600
 Mean   :0.07788   Mean   :0.9781   Mean   :1.750
 3rd Qu.:0.06250   3rd Qu.:1.0000   3rd Qu.:1.600
 Max.   :0.62500   Max.   :1.0000   Max.   :2.667
```

Всего было отобрано 48306 правил, длина которых большей частью составляла от 5 до 7 элементов. Функция `plot()` из пакета `arulesViz` позволяет получать различные формы визуализации синтезированных правил, в том числе, анализ изменчивости их мер качества (рис. 5.5):

```
library("arulesViz")
plot(rules, measure=c("support","lift"), shading="confidence")
```

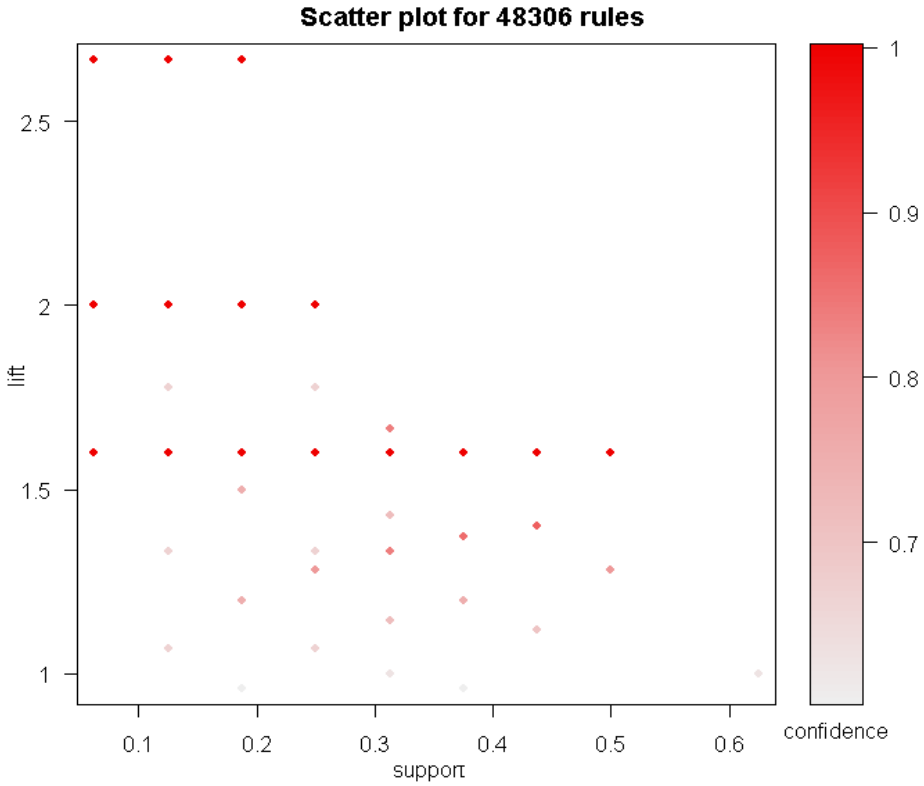


Рис. 5.5. Поддержка, лифт и достоверность сгенерированных правил

Для решения задачи выявления характерных особенностей групп электората нас интересуют, в первую очередь, высококачественные правила, имеющие соответствующий признак группы в правой части. Тогда патриотов можно будет легко узнать по их облику (рис. 5.6):

```
rulesPat <- subset(rules, subset = rhs %in% "Патриот" &
                                   lift > 1.8)
inspect(head(rulesPat, n = 10, by = "support"))
plot(head(sort(rulesPat, by="support"), 10),
         method="paracoord")
```

```
set of 2206 rules
     lhs                                                  rhs       support confidence lift
[1]  {борода,глаза_круглые,серьга}                     => {Патриот} 0.2500  1          2
[2]  {носогубн_складка,очки,усы}                       => {Патриот} 0.2500  1          2
[3]  {бабочка,очки,усы}                                => {Патриот} 0.2500  1          2
[4]  {курит_трубка,носогубн_складка,очки}              => {Патриот} 0.2500  1          2
[5]  {бабочка,носогубн_складка,очки,усы}               => {Патриот} 0.2500  1          2
[6]  {волосы,родинка_щеке}                             => {Патриот} 0.1875  1          2
[7]  {волосы,глаза_круглые,родинка_щеке}               => {Патриот} 0.1875  1          2
[8]  {лоб_морщины,носогубн_складка,очки}               => {Патриот} 0.1875  1          2
[9]  {курит_трубка,лоб_морщины,очки}                   => {Патриот} 0.1875  1          2
[10] {бабочка,лоб_морщины,очки}                        => {Патриот} 0.1875  1          2
```

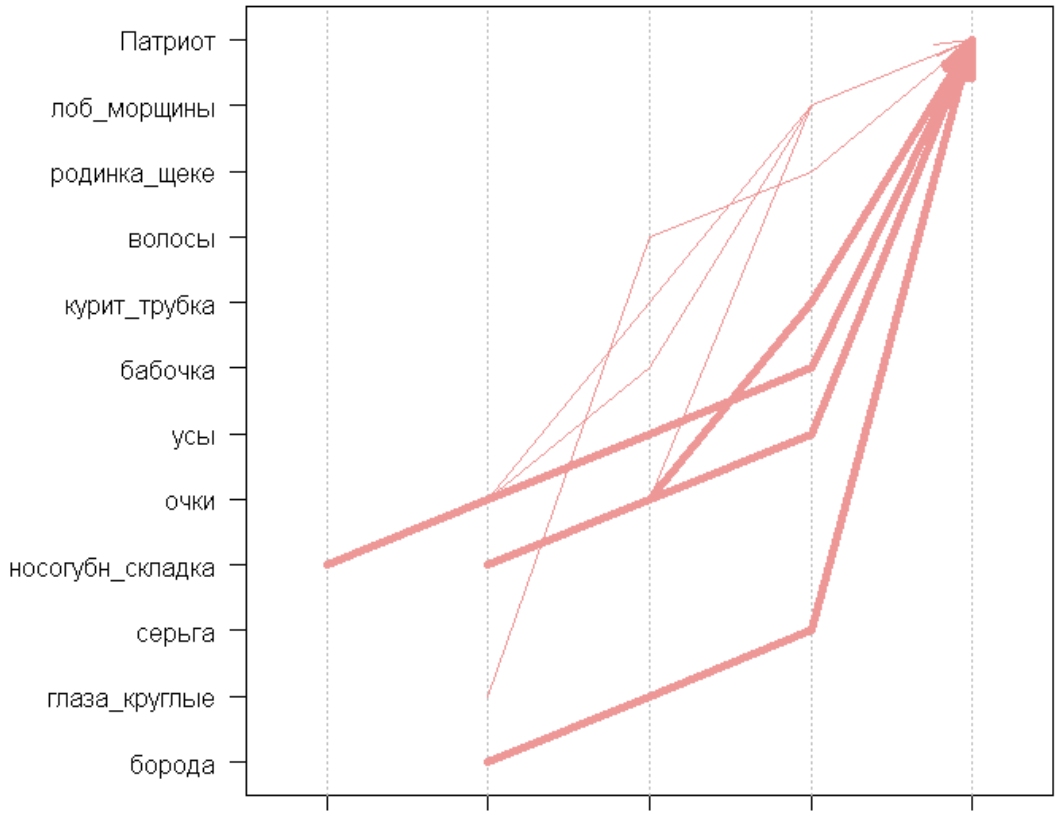


Рис. 5.6. График 10 лучших правил для патриотов в параллельных координатах.

График в параллельных координатах (`method="paracoord"`) на рис. 5.6 показывает, как формируются комбинации признаков правой части при увеличении ее размера, а толщина линий соответствует уровню поддержки.

Аналогичные правила могут быть отобраны для группы демократично настроенных избирателей:

```
rulesDem <- subset(rules, subset = rhs %in% "Демократ" &
                           lift > 1.8)
inspect(head(rulesDem, n = 10, by = "support"))
plot (head(sort(rulesDem, by="support"), 10), method="graph",
        control = list(nodeCol = grey.colors(10),
        edgeCol = grey(.7), alpha = 1))
```

```
     lhs                                                    rhs     support conf lift
[1]  {брови_подняты,нос_круглый,усы}                     => {Демократ} 0.2500  1   2
[2]  {глаза_круглые,губы_толстые,уши_оттопырен}          => {Демократ} 0.2500  1   2
[3]  {глаза_круглые,губы_толстые,курит_трубка}           => {Демократ} 0.2500  1   2
[4]  {бабочка,губы_толстые,уши_оттопырен}                => {Демократ} 0.2500  1   2
[5]  {глаза_круглые,губы_толстые,курит_трубка,уши_оттопырен} => {Демократ} 0.2500  1   2
```

```
[6]  {голова_круглая,лоб_морщины}                          => {Демократ} 0.1875  1   2
[7]  {нос_круглый,родинка_щеке,усы}                          => {Демократ} 0.1875  1   2
[8]  {брови_подняты,нос_круглый,родинка_щеке}                => {Демократ} 0.1875  1   2
[9]  {брови_подняты,родинка_щеке,усы}                        => {Демократ} 0.1875  1   2
[10] {голова_круглая,лоб_морщины,усы}                        => {Демократ} 0.1875  1   2
```

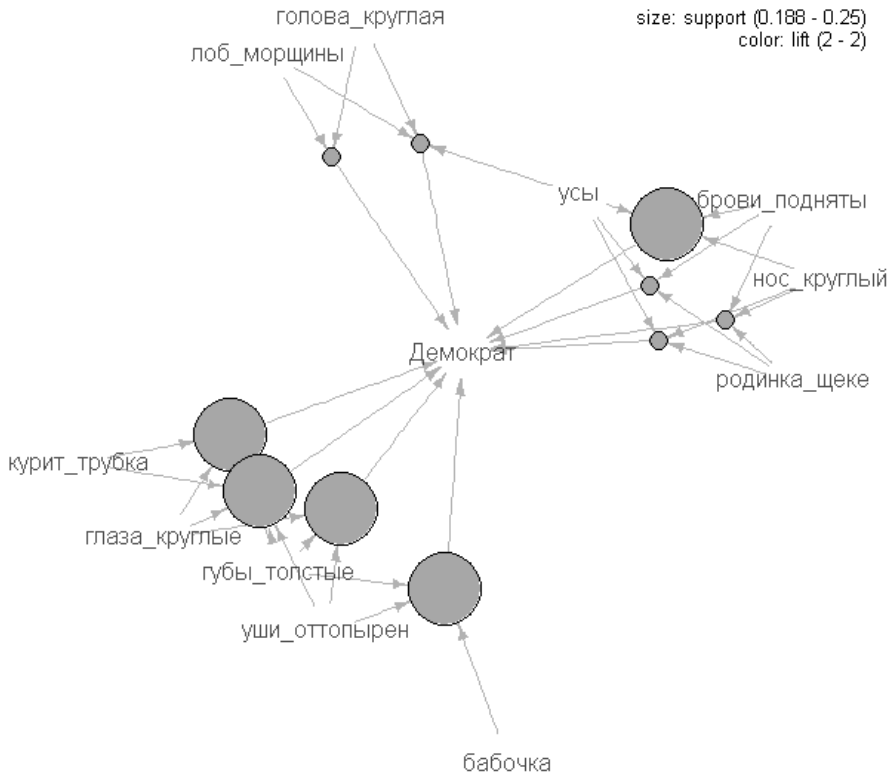


Рис. 5.7. Визуализация в форме графа 10 лучших правил для демократов

Метод `"graph"` функции `plot()` показывает правила и составляющие их признаки в виде графа, размер узлов которого пропорционален уровню поддержки каждого представленного правила (рис. 5.7).

5.5. Анализ последовательностей знаков или событий

Проблема статистического анализа последовательностей возникла, в первую очередь, с появлением быстрых методов секвенирования ДНК, в результате чего объем баз первичных нуклеотидных последовательностей растет экспоненциально. Параллельно интерес к этим методам возник при проведении маркетинговых исследований. Появилось желание выяснить, какова спонтанная траектория передвижения покупателя в торговом центре или в каком порядке пользователь нажимает кнопки и ссылки в Интернет-магазине, когда покупает товар (Горбач, Цейтлин, 2011).

Под последовательностью будем понимать специфическую переменную нечисловой природы, которая представляет собой упорядоченный набор знаков, обозначающих некоторые случайные состояния или события (например, наличие элемента структуры молекулы):

Оказалось, что с последовательностями можно работать почти так же, как и с обыкновенными числами: оценивать их близость, выполнять кластерный анализ, включать в регрессионные модели и проч. Убедимся в этом, рассматривая функции пакета `TraMineR` (**Tra**jectory **Mine**r in **R**), разработанного в Женевском университете (Gabadinho et al., 2011).

Пакет `TraMineR` предоставляет инструментарий для описания, вычисления и графического изображения основных параметров последовательностей (Бирюкова, 2014), в том числе:

- оценки характеристик отдельных последовательностей, таких как длина, время пребывания в каждом из состояний, энтропия и проч.;
- расчета агрегированных характеристик совокупностей: скорости переходов, средней длительности пребывания в каждом состоянии, частоты повторения последовательностей;
- вычисления матрицы попарных различий между последовательностями с использованием разных методов формализации «расстояния» и выделение кластеров;
- статистического анализа различий между группами последовательностей (ANOVA) и построение регрессионных моделей с использованием связанных ковариат;
- графической визуализации последовательностей, их частотных распределений, связей между ними и проч.

Рассмотрим в качестве примера набор данных `biofam`, который прослеживает семейный статус 2000 швейцарских граждан на протяжении 16 лет их жизни между 15 и 30 годами. В столбцах `a15-a30` таблицы данных представлены коды семейного состояния от 0 до 7, имеющие следующий смысл:

Код	Живет отдельно	Семей-ный	Есть дети	Разведен	Описание	Метка
0	нет	нет	нет	нет	Живет с родителями	Родит
1	да	нет	нет	нет	Живет отдельно	Отд
2	нет	да	-	нет	Семья у родителей	Сем+Род
3	да	да	нет	нет	Семья без детей	Сем.
4	нет	нет	да	нет	Одиночка с родителями	Один+Род
5	да	нет	да	нет	Одиночка отдельно	Один+Отд
6	да	да	да	нет	Семья с детьми	Сем+Дет
7	-	-	-	да	Разведен	Развод

Функция `seqdef()` обеспечит нам создание объекта `stslist`, включающего последовательности с необходимыми метками. Первое, что может представить интерес – это визуализация 10 наиболее часто

встречающихся последовательностей и частотного распределения семейного статуса по мере взросления молодых людей:

```
library(TraMineR) ; data(biofam)
summary(biofam) #  Результаты не приведены
biofam.lab <- c("Родит", "Отд", "Сем+Род",
        "Сем.", "Один+Род", "Один+Отд", "Сем+Дет", "Развод")
biofam.seq <- seqdef(biofam[, 10:25], labels = biofam.lab)
par(mfrow = c(2, 2)); seqiplot(biofam.seq, withlegend = FALSE)
seqdplot(biofam.seq, withlegend = FALSE)
seqfplot(biofam.seq, withlegend = FALSE)
seqlegend(biofam.seq)
```

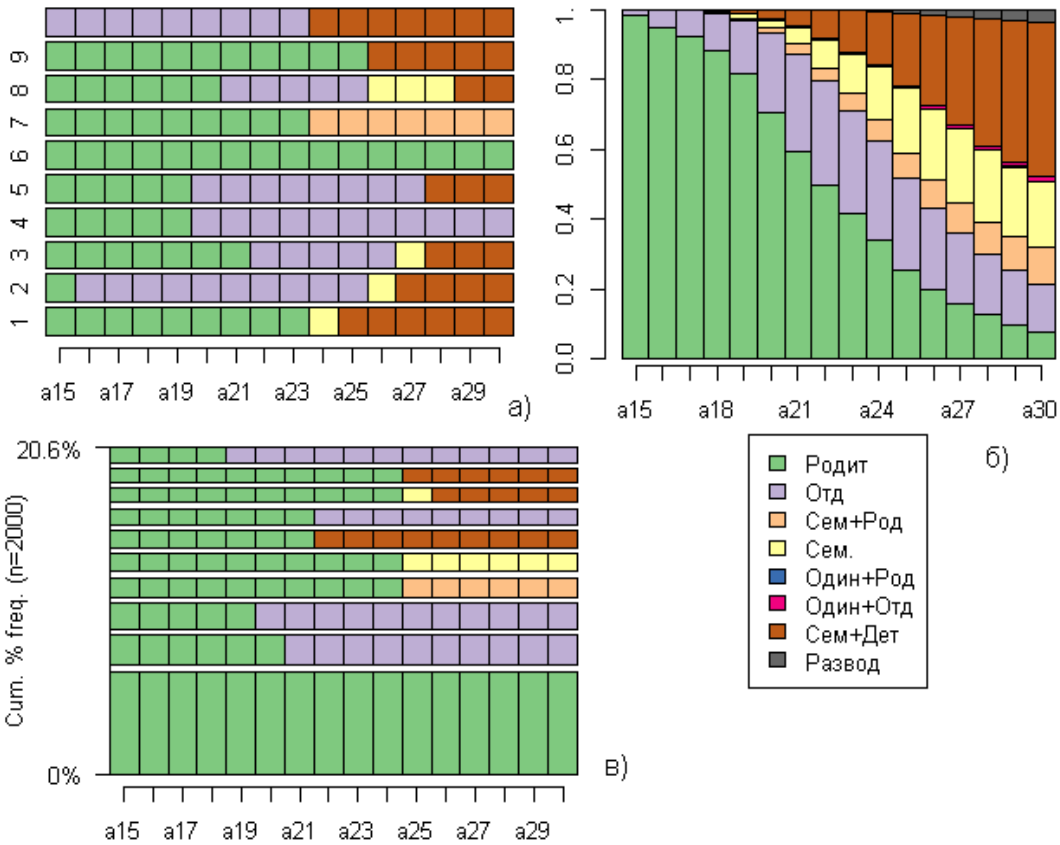


Рис. 5.8. Визуализация 10 первых (а), наиболее часто встречающихся последовательностей (б) и график частотного распределения градаций (в)

В этом наборе данных содержится также ряд дополнительных показателей об анкетируемых респондентах: пол (`sex`), год рождения от 1909 до 1957 г. (`birthyr`), национальность (`nat_1_02`), родной язык (`plingu02`), религиозность (`p02r01, p02r04`), профессиональный статус отца и матери (`cspfaj, cspmoj`). С использованием семейства функций `seq*plot()` можно визуализировать данные с разбивкой по группам. Например, можно сделать группировку по полу и установить, что только 20% 25-летних швейцарских девушек продолжают жить с родителями, тогда как таких юношей – более 33% (рис. 5.9):

```
seqdplot(biofam.seq,group=biofam$sex, withlegend = FALSE)
```

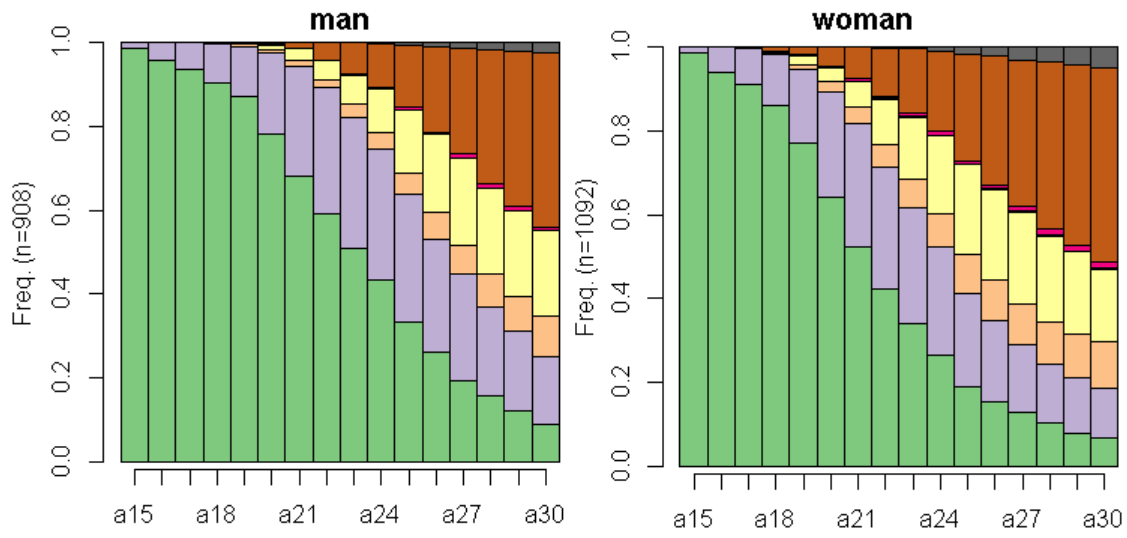


Рис. 5.9. График частотного распределения градаций с группировкой по полу

Для набора исходных данных можно оценить два вида энтропии Шеннона. Вертикальная, или "кросс-секторальная", энтропия рассчитывается на основе распределения состояний в каждый момент времени для всех анализируемых последовательностей (рис. 5.10):

```
seqHtplot(biofam.seq)
```

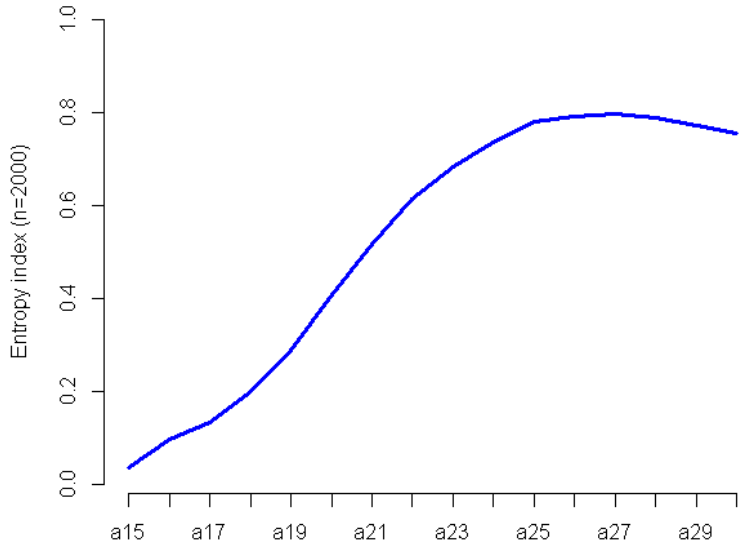


Рис. 5.10. Изменение относительной энтропии по градациям возрастов

Очевидно, что в 15 лет все респонденты находятся в одном и том же состоянии – проживают в доме родителей и энтропия равна 0. Возраст от 24 до 27 лет связан с активной сменой семейного статуса.

Для каждой из последовательностей можно рассчитать набор разнообразных метрик, таких как длина, число подпоследовательностей (number of subsequences) или различных состояний (distinct states), длина пребывания в стабильном состоянии (durations) и число внутренних переходов (transitions). Важной обобщенной мерой нестабильности последовательности является ее горизонтальная энтропия (longitudinal, или

within-sequence entropy). В нормированном выражении она принимает значение 0, если последовательность состоит из одинаковых состояний, и 1, если каждое состояние из общего словаря встречается с равной вероятностью (т.е. напимер, каждый из 8 кодов семейного состояния встречается точно два раза).

Рассмотрим, как изменяется средняя энтропия последовательностей за исторический период середины XX века (рис. 5.11):

```
Entropy <- seqient(biofam.seq)
ageg = cut(biofam$birthy, c(1909,1918,1928,1938,1948,1958),
label = c("1909-18","1919-28","1929-38","1939-48","1949-58"),
          include.lowest = TRUE)
boxplot(Entropy ~ ageg, data = biofam,
    xlab = "Диапазон годов рождения",
    ylab = "Энтропия последовательностей", col = "cyan")
```

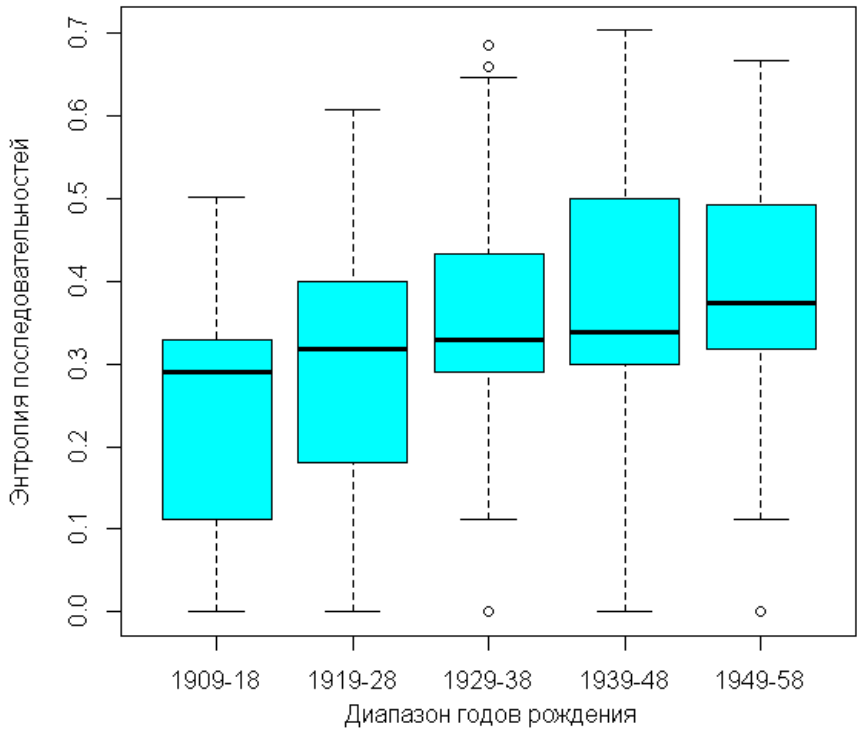


Рис. 5.11. Изменение относительной энтропии семейного состояния в течение XX века

Дотошный аналитик наверняка сможет усмотреть на рис. 5.11, что молодые швейцарцы 70-80-х годов существенно активнее меняли свой семейный статус, чем в 30-40 годы.

Индекс сложности (complexity) последовательности рассчитывается как геометрическое среднее энтропии и числа внутренних переходов, отнесенных к их максимальным значениям. По еще более сложным формулам, учитывающим длины фрагментов с постоянным состоянием и разнообразие переходов, рассчитывается мера турбулентности (turbulence). Достоинства и недостатки отдельных индексов постоянно дискутируются (Gabadinho et al., 2011a), поэтому интересно проследить зависимость между ними и убедиться

в том, что каждый индекс характеризует последовательность несколько по-разному (рис. 5.12):

```
Turbulence <- seqST(biofam.seq)
plot(Turbulence, Entropy, xlab = "Турбулентность",
                          ylab = "Энтропия")
```

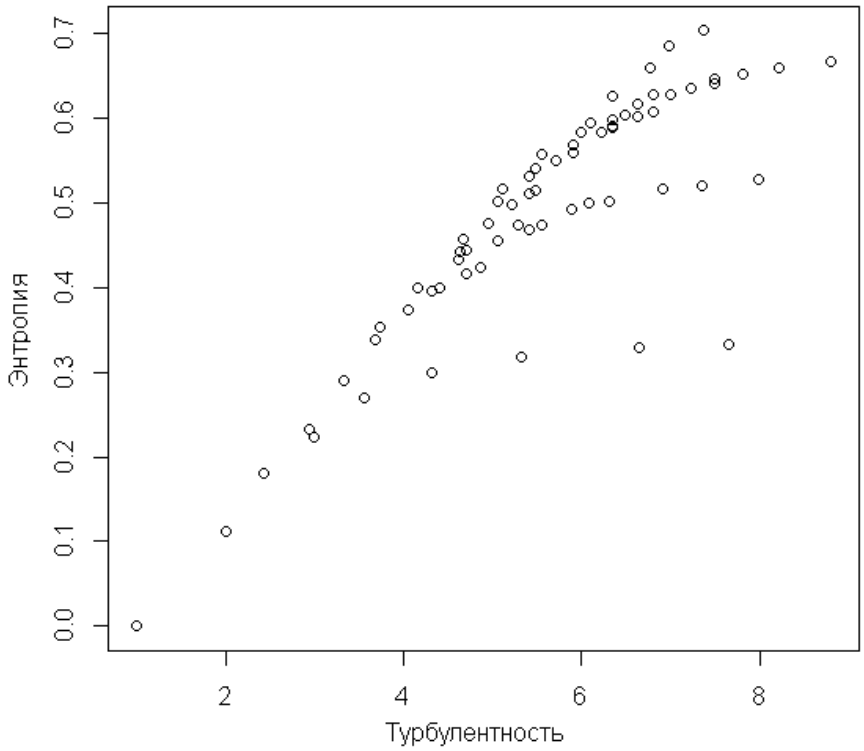


Рис. 5.12. Зависимость энтропии от турбулентности

Поскольку любой из этих индексов является численной характеристикой последовательности, они могут использоваться как компоненты обычных линейных моделей:

```
m.turb <- lm(Turbulence ~ sex + birthyr, data = biofam)
summary(m.turb)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -50.534580   7.355288  -6.871 8.52e-12 ***
sexwoman      0.373527   0.079528   4.697 2.82e-06 ***
birthyr       0.028381   0.003786   7.495 9.88e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Нетрудно сделать вывод, что турбулентность семейной жизни статистически значимо выше у женщин и увеличивается по мере приближения к XXI веку.

Основная проблема анализа последовательностей – как формализовать различия между двумя произвольными последовательностями и выразить их в виде численной метрики – дистанции *D*? Описанные выше обобщенные показатели сложности тут непригодны, и обычно используют несколько подходов:

1. Вычисление расстояния по Хеммингу, или числа несовпадающих позиций (nomatching positions).
2. Выделение длины совпадающей части с начала последовательности S_p (LCP, Longest Common Prefix, $D_p = 1 - S_p$) или максимальной длины общей подпоследовательности (LCS, Longest Common Subsequence).
3. Оценка метрики Левенштайна (Горбач, Цейтлин, 2011), т.е. суммарной "стоимости" преобразования одной последовательности в другую.

В общем случае операциям редактирования, используемым в преобразовании по п. 3, можно назначить разную "стоимость", а именно, вставке события ***а***, его удалению или замене ***а*** → ***б*** (что также интерпретируется как "удаление ***б*** – вставка ***а***"). В пакете `TraMineR` функция `seqsubm()` с использованием `method = "TRATE"` выполняет оценку матрицы стоимостей переходов от ***а*** к ***б***, а другая функция - `seqdist()` - с использованием этих коэффициентов рассчитывает матрицу дистанций "оптимального" редактирования (OM, optimal matching distances) между каждой парой взятых в анализ последовательностей:

```
couts <- seqsubm(biofam.seq, method = "TRATE")
biofam.om <- seqdist(biofam.seq, method = "OM",
                     indel = 3, sm = couts)
round(biofam.om[1:10, 1:10], 1)
```

```
 [>] 2000 sequences with 8 distinct events/states
 [>] 537 distinct sequences
 [>] min/max sequence length: 16/16
 [>] computing distances using OM metric
 [>] total time: 0.99 secs
# --------------------
 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]  0.0 21.3 11.6 21.6 15.6 13.9 13.9 15.1  4.0  19.3
 [2,] 21.3  0.0 15.4 17.6 11.7 29.4 29.5 13.3 21.3   7.7
 [3,] 11.6 15.4  0.0 11.7  5.8 17.7 17.8  5.7 11.6  21.4
 [4,] 21.6 17.6 11.7  0.0  5.9 21.4 21.8 11.6 21.6  23.6
 [5,] 15.6 11.7  5.8  5.9  0.0 21.5 21.7  7.6 15.6  17.6
 [6,] 13.9 29.4 17.7 21.4 21.5  0.0 13.9 19.6  9.9  31.4
 [7,] 13.9 29.5 17.8 21.8 21.7 13.9  0.0 19.8 13.9  31.4
 [8,] 15.1 13.3  5.7 11.6  7.6 19.6 19.8  0.0 15.1  21.0
 [9,]  4.0 21.3 11.6 21.6 15.6  9.9 13.9 15.1  0.0  21.5
[10,] 19.3  7.7 21.4 23.6 17.6 31.4 31.4 21.0 21.5   0.0
```

Мы привели таблицу парных расстояний между первыми 10 последовательностями, и читатель может сопоставить ее с результатом их визуализации на рис. 5.8а. Отметим, что для расчета матрицы размерностью 2000 x 2000 и объемом 30 Мб на стандартном компьютере понадобилось меньше 1 сек.

Рассчитанная матрица дистанций может быть использована для построения кластеров последовательностей с применением любой пригодной функции R из других профильных пакетов. Не останавливаясь на ассортименте и специфике применения методов кластеризации, о чем речь пойдет в главе 10, осуществим построение иерархической дендрограммы по

алгоритму Уорда. Используем функцию agnes(), на вход которой подается рассчитанная выше матрица расстояний:

```
library(cluster)
clusterward <- agnes(biofam.om, diss = TRUE, method = "ward")
plot(clusterward, which.plots = 2)
```

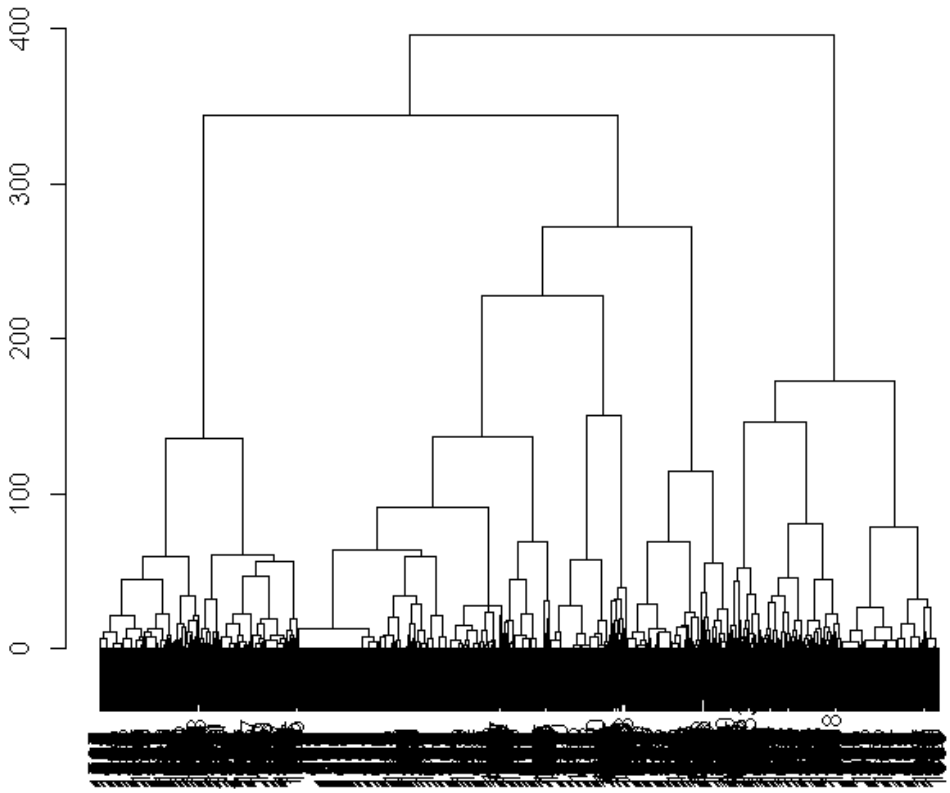


Рис. 5.13. Иерархическая кластеризация последовательностей по методу Уорда

Анализ результатов кластеризации мы можем продолжить с использованием функций seqdplot() или seqmtplot(), чтобы оценить, например, простое распределение частот семейных состояний по группам, выбранным по тому или иному уровню рассечения дендрограммы:

```
# "Распилим" дерево на три части
cluster3 <- cutree(clusterward, k = 3)
cluster3 <- factor(cluster3,
        labels = c("Кластер 1", "Кластер 2", "Кластер 3"))
table(cluster3)
par(mfrow = c(2,2))
seqmtplot(biofam.seq, group = cluster3)
```

```
cluster3
Кластер 1 Кластер 2 Кластер 3
      472       502      1026
```

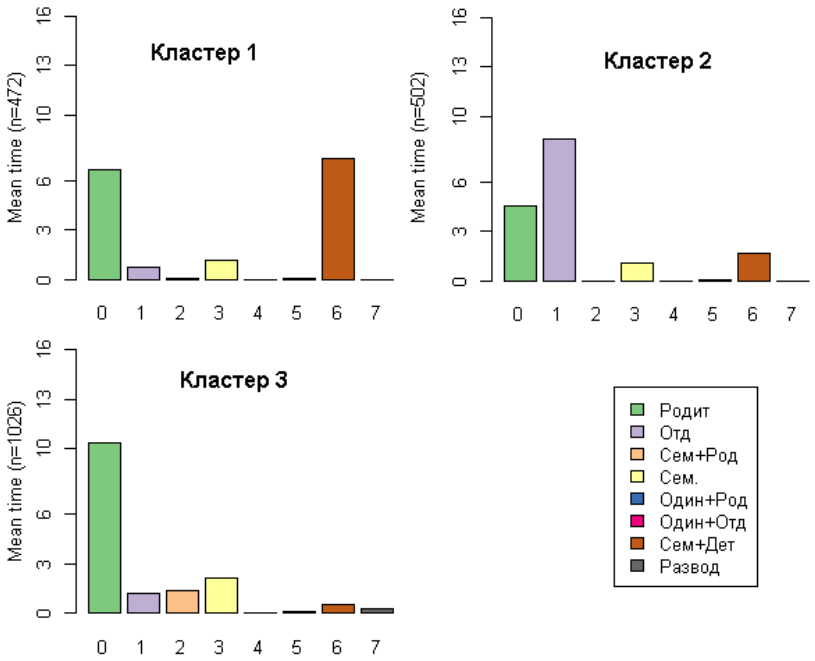


Рис. 5.14. Распределение частот семейных состояний по выделенным кластерам

Матрица расстояний между последовательностями может быть использована для непараметрического дисперсионного анализа по методу Андерсона (Anderson, 2001), подробному описанию сущности которого посвящен раздел 9.3. В общих чертах, если группировка задается в соответствии с уровнями изучаемых факторов, то алгоритм осуществляет разложение многомерной дисперсии, заключенной в матрице расстояний, на общую SS_T и внутригрупповую SS_W суммы квадратов расстояний d_{ij}. Пакет `TraMineR` включает несколько эффективно работающих функций непараметрического ANOVA, которые с успехом могут использоваться в любом анализе произвольных матриц дистанций.

Функция `dissassoc()` оценивает зависимость между степенью изменчивости элементов матрицы парных расстояний и дискретной ковариатой (номинальной переменной). Рассчитываются обычные компоненты таблицы дисперсионного анализа (суммы квадратов и их средние по числу степеней свободы), дисперсионное отношение F и коэффициент детерминации R^2. Функция выполняет также тест на однородность дисперсий в выделенных группах с использованием статистик Левене (Levene) и Бартлетта (Bartlett). Статистическая значимость тестов оценивается путем рандомизации (т.е. элементы матрицы дистанций случайным образом перемешиваются `R` раз относительно уровней фактора).

Оценим, как влияет пол респондента (`sex`) на взаимные расстояния между последовательностями:

```
dissvar(biofam.om) # Общая дисперсия парных расстояний
da <- dissassoc(biofam.om, group=biofam$sex, R = 1000)
print(da)
```

```
[1]  8.316977
```

```
Pseudo ANOVA table:
               SS   df        MSE
Exp     169.1489    1 169.148898
Res   16456.4883 1998   8.236481
Total 16625.6372 1999   8.316977

Test values  (p-values based on 1000 permutation):
                     t0 p.value
Pseudo F   20.53655013   0.001
Pseudo Fbf 20.77949020   0.001
Pseudo R2   0.01017398   0.001
Bartlett    3.37903015   0.001
Levene     62.60262292   0.001
```

Из результатов следует, что межгрупповая вариация `MSE Exp` статистически значимо превышает остаточную вариацию `MSE Res`, а величина псевдо- R^2 достоверно больше нуля.

Функция `dissmfac()` является, в некотором смысле, обобщением `dissassoc()` для оценивания линейных вкладов нескольких объясняющих переменных, задаваемых формулой:

```
dlm <- dissmfac(biofam.om ~ sex + ageg, data =biofam, R = 100)
```

```
  Variable  PseudoF   PseudoR2 p_value
1      sex 32.20272 0.01538802    0.01
2     ageg 16.87843 0.03226131    0.01
3    Total 19.74283 0.04717040    0.01
```

Очевидно, что оба фактора – пол и год рождения, – значимо влияют на вариацию расстояний между последовательностями, т.е. среднее внутригрупповое расстояние существенно меньше общей средней дистанции.

Наконец, функция `disstree()` осуществляет рекурсивное построение дерева решений. На каждом последовательном шаге выбирается уровень заданного фактора и матрица дистанций разбивается на блоки, обеспечивая при этом уменьшение внутригрупповых расстояний Чтобы получить изображение дерева на диаграмме, необходимо установить пакет `DiagrammeR`:

```
disstree (biofam.om ~ sex + birthyr, data =biofam, R = 100)
```

```
Global R2: 0.036755
|-- Root  (n: 2000 disc: 8.3128)
   |-> birthyr 0.014749
             |-- <= 1940  (n: 773 disc: 7.7673)
               |-> birthyr 0.011076
                   |-- <= 1928    (n: 229 disc: 7.2023)[376] *
                   |-- > 1928  (n: 544 disc: 7.8828)
                     |-> sex 0.0082047
                         |-- [ man ]    (n: 263 disc: 7.381)[1638] *
                         |-- [ woman ]    (n: 281 disc: 8.2274)[636] *
             |-- > 1940  (n: 1227 disc: 8.4567)
               |-> sex 0.013006
                   |-- [ man ]  (n: 549 disc: 7.8792)
                     |-> birthyr 0.012191
                         |-- <= 1951    (n: 330 disc: 7.9669)[644] *
                         |-- > 1951    (n: 219 disc: 7.5061)[886] *
                   |-- [ woman ]  (n: 678 disc: 8.7252)
                     |-> birthyr 0.0084755
                         |-- <= 1946    (n: 204 disc: 8.5457)[1021] *
                         |-- > 1946  (n: 474 disc: 8.6967)
                           |-> birthyr 0.0063931
                               |-- <= 1953    (n: 289 disc: 8.7975)[306] *
                               |-- > 1953    (n: 185 disc: 8.3967)[361] *
```

6. БИНАРНЫЕ КЛАССИФИКАТОРЫ С РАЗЛИЧНЫМИ РАДЕЛЯЮЩИМИ ПОВЕРХНОСТЯМИ

6.1. Дискриминантный анализ

Линейный дискриминантный анализ (Linear Discriminant Analysis, LDA) является разделом многомерного анализа, который позволяет оценивать различия между двумя и более группами объектов по нескольким переменным одновременно (Афифи, Эйзенс, 1982; Айвазян и др., 1989). Он реализует две тесно связанные между собой статистические процедуры:

- *интерпретацию межгрупповых различий*, когда нужно ответить на вопрос: насколько хорошо используемый набор переменных в состоянии сформировать разделяющую поверхность для объектов обучающей выборки и какие из этих переменных наиболее информативны?
- *классификацию*, т.е предсказание значения группировочного фактора для экзаменуемой группы наблюдений.

В основе дискриминантного анализа лежит предположение о том, что описания объектов каждого k-го класса представляют собой реализации многомерной случайной величины, распределенной по нормальному закону $N_m(\boldsymbol{\mu}_k; \boldsymbol{\Sigma}_k)$ со средними $\boldsymbol{\mu}_k$ и ковариационной матрицей

$$\mathbf{C}_k = \frac{1}{n_k - 1}\sum_{i=1}^{n_k}(\mathbf{x}_{ik} - \boldsymbol{\mu}_k)^{\mathrm{T}}(\mathbf{x}_{ik} - \boldsymbol{\mu}_k)$$

(индекс m указывает на размерность признакового пространства).

Рассмотрим несколько упрощенную геометрическую интерпретацию алгоритма LDA для случая двух классов. Пусть дискриминантные переменные $\boldsymbol{x}$ – оси m-мерного евклидова пространства. Каждый объект (наблюдение) является точкой этого пространства с координатами, представляющими собой фиксируемые значения каждой переменной. Если оба класса отличаются друг от друга по наблюдаемым пртзнакам, их можно представить как скопления точек в разных областях рассматриваемого пространства, которые могут частично перекрываться. Для определения положения каждого класса можно вычислить его "центроид", который является воображаемой точкой, координатами которой являются средние значения переменных в данном классе.

Задача дискриминантного анализа заключается в проведении дополнительной оси z, которая проходит через облако точек таким образом, что проекции на нее обеспечивают наилучшую разделяемость на два класса. Ее положение задается линейной дискриминантной функцией (linear discriminant, LD) с весовыми коэффициентами β_j, определяющими вклад каждой исходной переменной x_j: $z(\boldsymbol{x}) = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m$.

Если сделать предположение, что ковариационные матрицы объектов классов 1 и 2 равны, т.е. $\mathbf{C} = \mathbf{C}_1 = \mathbf{C}_2$, то вектор коэффициентов $\{\beta_1, \ldots, \beta_m\}$ линейного дискриминанта $z(\boldsymbol{x})$ может быть вычислен по формуле

$\boldsymbol{\beta} = \mathbf{C}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, где $\mathbf{C}^{-1}$ – матрица, обратная к ковариационной, $\boldsymbol{\mu}_k$ – вектор средних k-го класса. Полученная ось совпадает с уравнением прямой, проходящей через центроиды двух групп объектов классов, а обобщенное расстояние Махаланобиса, равное дистанции между ними в многомерном пространстве признаков, оценивается как

$$D^2 = \boldsymbol{\beta}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2).$$

Таким образом, в LDA кроме предположения о нормальности распределения данных в каждом классе, которое на практике выполняется довольно редко, выдвигается еще и более серьезное предположение о статистическом равенстве внутригрупповых матриц дисперсий и корреляций. Если между ними нет серьезных отличий, их объединяют в расчетную ковариационную матрицу $\mathbf{C} = [\mathbf{C}_1(n_1 - 1) + \mathbf{C}_2(n_2 - 1)]/ (n_1 + n_2 - 2)$.

По поводу искусственного объявления ковариационных матриц статистически неразличимыми существуют два различных мнения: одни исследователи считают, что могут оказаться отброшенными наиболее важные индивидуальные черты, характерные для каждого из классов и имеющие большое значение для хорошего разделения, тогда как другие – что это условие не является критическим для эффективного применения дискриминантного анализа. Тем не менее, проверка исходных предположений всегда остается правилом хорошего тона в статистике.

Для проверки гипотезы о многомерном нормальном распределении данных используется многомерная версия критерия согласия Шапиро-Уилка, которая реализована в функции `mshapiro.test()` из пакета `mvnormtest`. На вход этой функции подается матрица, строки которой соответствуют переменным, а столбцы – наблюдениям.

В разделах 2.4-2.5 мы подробно рассматривали пример анализа зависимости между двумя различными способами производства листового стекла (флеш-стекло и по принципу вертикального вытягивания) и составом его химических ингредиентов. С использованием статистики Пиллая и критерия Хотеллинга была показана статистическая значимость такой связи. Применим многомерный критерий Шапиро-Уилка к оценке характера распределения этой выборки:

```
DGlass <- read.table(file="Glass.txt", sep=",",
                     header=TRUE, row.names=1)
DGlass$F <- as.factor(ifelse(DGlass$Class==2, 2, 1))
library(mvnormtest)
mshapiro.test(t(DGlass[DGlass$F == 1, 1:9]))
mshapiro.test(t(DGlass[DGlass$F == 2, 1:9]))
```

```
        Shapiro-Wilk normality test
data:  F == 1
W = 0.1965, p-value < 2.2e-16
data:  F == 2
W = 0.1368, p-value < 2.2e-16
```

Для обеих групп выявлены значимые отклонения от многомерного нормального распределения.

Для проверки гипотезы о гомогенности матриц ковариаций используется так называемый M-критерий Бокса, который реализован в функции `boxM()` из пакета `biotools`:

```
library(biotools)
boxM(as.matrix(DGlass[, 1:9]), DGlass$F )
```

```
        Box's M-test for Homogeneity of Covariance Matrices
data:  as.matrix(DGlass[, 1:9])
Chi-Sq (approx.) = 443.1846, df = 45, p-value < 2.2e-16
```

Гетерогенность матриц ковариаций также статистически значима.

Дискриминантный анализ реализован в нескольких пакетах для R, но мы рассмотрим применение функции `lda()` из базового пакета `MASS`. Поскольку важной характеристикой прогнозирующей эффективности модели является ее ошибка при перекрестной проверке, то в функции `lda()` пакета `MASS` заложена реализация скользящего контроля (leave-one-out CV). Напомним, что при этом из исходной выборки поочередно отбрасывается по одному объекту, строится n моделей дискриминации по $(n - 1)$ выборочным значениям, а исключенное наблюдение каждый раз используется для учета ошибки классификации.

Составим предварительно функцию, которая по построенной модели выводит нам важные показатели для оценки ее качества: матрицы неточностей на обучающей выборке и при перекрестной проверке, ошибку распознавания и расстояние Махалонобиса между центроидами двух классов. С использованием этой функции оценим эффективность дискриминационной модели прогнозирования способа производства стекла по его химическому составу:

```
# Функция вывода результатов классификации
Out_CTab <- function (model, group, type="lda") {
# Таблица неточностей "Факт/Прогноз" по обучающей выборке
   classified<-predict(model)$class
   t1 <-  table(group,classified)
# Точность классификации и расстояние Махалонобиса
   Err_S <- mean(group != classified) ; mahDist <- NA
   if (type=="lda")
       { mahDist <- dist(model$means %*% model$scaling) }
# Таблица "Факт/Прогноз" и ошибка при скользящем контроле
   t2 <-  table(group, update(model, CV=T)$class->LDA.cv)
   Err_CV <- mean(group != LDA.cv)
   Err_S.MahD  <-c(Err_S, mahDist)
   Err_CV.N <-c(Err_CV, length(group))
   cbind(t1, Err_S.MahD, t2, Err_CV.N)
}
# --- Выполнение расчетов
library(MASS)
lda.all <- lda(F ~ ., data=DGlass[,-10])
Out_CTab(lda.all, DGlass$F)
```

```
   1  2 Err_S.MahD  1  2    Err_CV.N
1 69 18  0.2515337 66 21   0.3128834
2 23 53  1.2414682 30 46         163
```

Отметим существенный рост ошибки распознавания до 31% при выполнении скользящего контроля.

Естественно задаться вопросом, какие из имеющихся 9 признаков являются информативными при разделении, а какие – сопутствующим балластом. Шаговая процедура выбора переменных при классификации, реализованная функцией `stepclass()` из пакета `klaR`, основана на вычислении сразу четырех параметров качества моделей-претендентов: *а*) индекса ошибок (correctness rate), *б*) точности (accuracy), основанной на евклидовых расстояниях между векторами "факта" и "прогноза", *в*) способности к разделимости (ability to seperate), также основанной на расстояниях, и *г*) доверительных интервалах центроидов классов. Все эти параметры оцениваются в режиме многократной перекрестной проверки.

```
library ( klaR)
stepclass(F ~ ., data=DGlass[,-10],   method="lda")
`stepwise classification', using 10-fold cross-validated
correctness rate: 0.69228;  in: "Al";  variables (1): Al
correctness rate: 0.76618;  in: "Ca";  variables (2): Al, Ca
final model : F ~ Al + Ca
```

```
lda.step <- lda(F ~ Mg + Al, data=DGlass[,-10])
Prior probabilities of groups:
        1         2
0.5337423 0.4662577
Group means:
         Mg       Al
1 3.550690 1.171149
2 3.002105 1.408158
Coefficients of linear discriminants:
           LD1
Mg -0.8294482
Al  2.6894108
```

В результате получили компактную дискриминантную функцию

$$z\,(\boldsymbol{x}) = 2.69\mathrm{Al} - 0.83\mathrm{Mg},$$

зависящую только от двух переменных. Найдем ошибку предсказания как на обучающей выборке, так и при скользящем контроле:

```
partimat(F ~ Mg + Al, data=DGlass[,-10], main='',
              method="lda")
Out_CTab(lda.step, DGlass$F)
```

```
   1  2 Err_S.MahD  1  2    Err_CV.N
1 72 15  0.2331288 72 15   0.2392638
2 23 53  1.0924355 24 52         163
```

По обоим критериям ошибка предсказания сокращенной модели существенно ниже аналогичных показателей при использовании полного набора переменных. Это отражает общие методические закономерности, важные при выборе алгоритмов классификации и анализе результатов их работы:

- ошибка перекрестной проверки E_{CV} всегда превышает внутреннюю ошибку модели E_S на самой обучающей выборке и является объективной характеристикой качества распознавания на внешнем дополнении;

- сокращение размерности модели за счет выбора комплекса информативных переменных может увеличить ошибку E_S, но, как правило, снижает ошибку скользящего контроля E_{CV}.

Сокращение числа переменных до двух позволяет построить частный двумерный график, интерпретирующий процесс классификации (рис. 6.1).

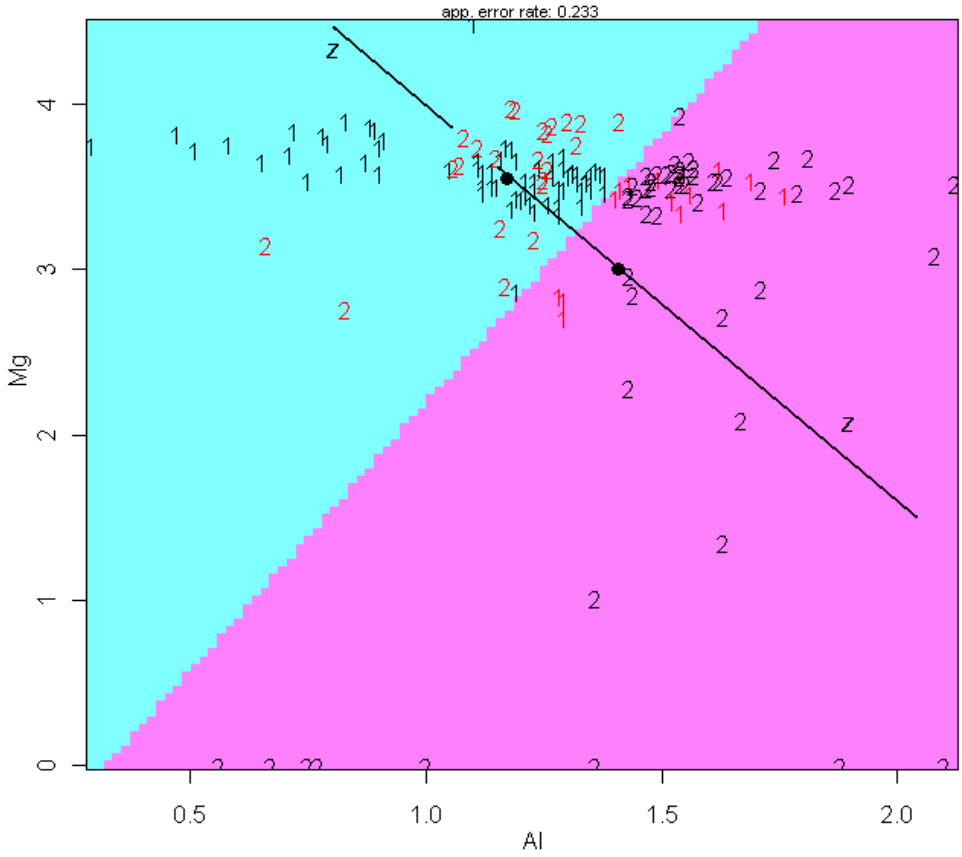


Рис. 6.1. Диаграмма дискриминации двух классов 1 и 2 (ошибочное распознавание выделено шрифтом красного цвета); показаны линейная дискриминантная функция *z* и жирными точками – положение центроидов.

Вместо функции `stepclass()` из пакета `klaR` для выбора оптимального набора предикторов можно воспользоваться функцией `rfe()` из пакета `caret` (процедура рекурсивного исключения – см. раздел 4.1):

```
ldaProfile <- rfe(DGlass[,1:9], DGlass$F1,
sizes = 2:9,
rfeControl = rfeControl(functions = ldaFuncs,
                        method = "repeatedcv",repeats = 6))
```

```
Recursive feature selection
Outer resampling: Cross-Validated (10 fold, repeated 6 times)
   Variables Accuracy  Kappa AccuracySD KappaSD Selected
           2   0.7212 0.4372     0.1091  0.2188
           3   0.7249 0.4453     0.1116  0.2231        *
           4   0.7187 0.4325     0.1174  0.2351
           5   0.7208 0.4370     0.1128  0.2247
           6   0.7011 0.3984     0.1189  0.2365
           7   0.7043 0.4010     0.1237  0.2480
           8   0.7243 0.4404     0.1040  0.2091
           9   0.7118 0.4193     0.1090  0.2179
```

```
The top 3 variables (out of 3):    Al, K, Fe
```

Для того чтобы уточнить, какую из трех построенных моделей следует предпочесть, выполним их тестирование на основе 10-кратной перекрестной проверки с 5 повторностями:

```
# Модель на основе всех 9 предикторов
lda.full.pro <- train(DGlass[, 1:9], DGlass$F,
        data= DGlass,method="lda",
  trControl = trainControl(method="repeatedcv",repeats=5,
  classProbs = TRUE), metric = "Accuracy")
# Модель на основе 2 предикторов stepclass
lda.step.pro <- train(F1 ~ Mg + Al, data= DGlass,method="lda",
  trControl = trainControl(method="repeatedcv",repeats=5,
  classProbs = TRUE), metric = "Accuracy")
# Модель на основе 3 предикторов rfe
lda.rfe.pro <- train(F1 ~ Al + K + Fe,
       data= DGlass,method="lda",
    trControl = trainControl(method="repeatedcv",repeats=5,
    classProbs = TRUE), metric = "Accuracy")
plot(varImp(lda.full.pro))
```

```
Linear Discriminant Analysis
  9 predictor  -  полная модель
  Accuracy   Kappa
  0.6974559  0.3875972

  2 predictor  -  модель stepclass()
  Accuracy   Kappa
  0.7613382  0.5175581

  3 predictor   -  модель rfe()
  Accuracy   Kappa
  0.7253725  0.4456987
```

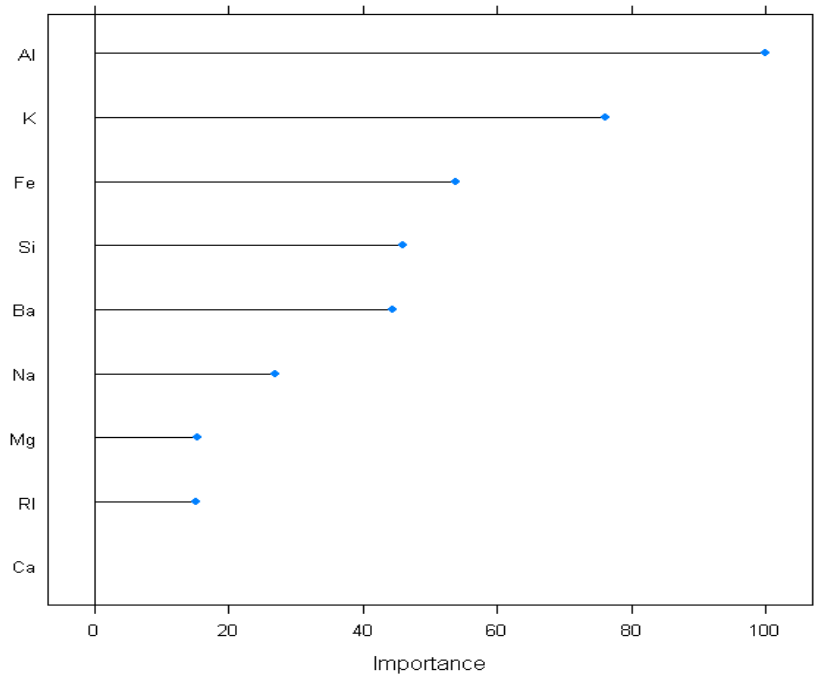


Рис. 6.2. Значения важности отдельных переменных на основе абсолютных значений *t*–статистики при построении модели дискриминантного анализа

Функция `rfe()` провела отбор переменных в полном соответствии с рейтингом их важности на рис. 6.2, но это решение оказалось неоптимальным. Модель `lda.step`, полученная с использованием функции `stepclass()`, оказалась существенно эффективней.

6.2. Метод опорных векторов

Метод *опорных векторов* (support vector), называемый ранее алгоритмом "обобщенного портрета", был разработан советскими математиками В. Н. Вапником и А. Я. Червоненкисом (1974) и с тех пор приобрел широкую популярность.

Основная идея классификатора на опорных векторах заключается в том, чтобы строить разделяющую поверхность с использованием только небольшого подмножества точек, лежащих в зоне, критической для разделения, тогда как остальные верно классифицируемые наблюдения обучающей выборки вне этой зоны игнорируются (точнее, являются "резервуаром" для оптимизационного алгоритма).

Если имеется два класса наблюдений и предполагается линейная форма границы между классами, то возможны два случая. Первый из них связан с возможностью идеального разделения данных при помощи некоторой гиперплоскости $z_k(x) = \sum_{i=1}^{p} \beta_i x_i + \beta_0$ (двумерный вариант представлен слева на рис. 6.3). Поскольку таких гиперплоскостей может быть множество, то оптимальной является разделяющая поверхность, которая максимально удалена от обучающих точек, т.е. имеющая максимальный зазор *M* (margin).

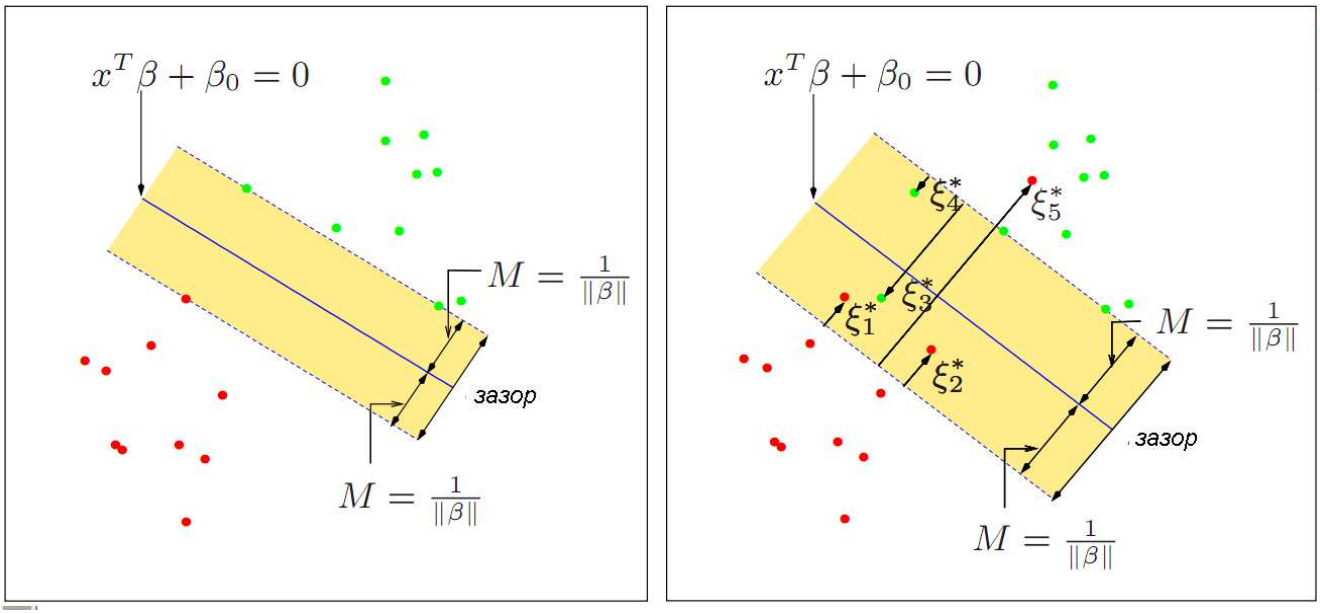


Рис. 6.3. Классификаторы с минимальным зазором (слева) и на опорных векторах (справа)

На рис. 6.3 справа показан другой случай, когда облако точек перекрывается и оба класса линейно неразделимы. Собственно *опорными*

векторами называются наблюдения, лежащие непосредственно на границе разделяющей полосы, либо на неправильной для своего класса стороне относительно границ зазора (такие точки отмечены ξ_j^* на рис. 6.3). Для граничных и всех остальных точек принимается $\xi_j^* = 0$.

Оптимальную разделяющую гиперплоскость такого классификатора $z_k(x) = \sum_{i=1}^{p} \beta_i x_i + \beta_0$ также находят из условия максимизации ширины зазора *M*, но при этом разрешается неверно классифицировать некоторую небольшую группу наблюдений, относящихся к опорным векторам. Для этого задаются дополнительным условием оптимизации $\sum_j \xi_j^* \le C$, где C – допустимое число нарушений границы зазора и их выраженность, которое обычно выбирается с использованием перекрестной проверки. Математически поиск решения сводится к удобной задаче квадратичной оптимизации с линейными ограничениями, которая гарантировано сходится к одному глобальному минимуму (Vapnik, 1995; Джеймс и др., 2016).

Поскольку на расположение гиперплоскости оказывают влияние только те наблюдения, которые лежат на границах зазора или нарушают его, то решающее правило такого классификатора довольно устойчиво к выбросам большинства точек, расположенных вне "критической зоны" разделения. Это свойство отличает его от свойств других классификаторов. Например, разделяющая плоскость LDA на рис. 6.3 проводится перпендикулярно дискриминационной функции *z* и зависит от средних и ковариационной матрицы, вычисляемых по всему множеству имеющихся наблюдений.

Для реализации метода опорных векторов в среде R обычно используется функция `svm()` из пакета `e1071`. Рассмотрим построение с помощью этой функции линейного классификатора для прогнозирования способа производства стекла по его химическому составу (см. разделы 2.4-2.5, 6.1). Для подбора параметров модели зададим перекрестную проверку с делением исходной выборки на 10 равных частей (`cross=10`):

```
DGlass <- read.table(file="Glass.txt", sep=",",
                     header=TRUE, row.names=1)
DGlass$F <- as.factor(ifelse(DGlass$Class==2,2,1))
svm.all <-  svm(formula = F ~ ., data=DGlass[,-10],
       cross=10, kernel = "linear")
table(Факт=DGlass$F, Прогноз=predict(svm.all))
Acc = mean(predict(svm.all) == DGlass$F )
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
    Прогноз
Факт  1  2
   1 72 15
   2 27 49
[1] "Точность=74.23%"
```

Для подтверждения того, что `svm()` действительно проводила построение модели с применением кросс-проверки, выполним дополнительно скользящий контроль с использованием самостоятельно составленной функции:

```
#   Функция для вычисления ошибки перекрестной проверки
library(e1071)
CVsvm <- function(x, y) {
   n <- nrow(x) ; Err_S  <- 0
   for(i in 1:n) {
      svm.temp <- svm(x=x[-i,], y = y[-i],  kernel = "linear")
      if (predict(svm.temp, newdata=x[i,]) != y[i])
            Err_S <- Err_S +1 }
   Err_S/n }
Acc <- 1-CVsvm(DGlass[,1:9],DGlass$F)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
[1] "Точность=74.17%"
```

Ошибка предсказания изменилась лишь незначительно.

Определенным резервом повышения эффективности модели является выбор оптимального набора предикторов. Селекция переменных может быть выполнена несколькими способами:

- с помощью известной нам функции `stepclass()` из пакета `klaR`, но, в отличии от LDA, здесь необходимо использовать метод построения модели `svmlight`;
- с использованием алгоритма рекурсивного извлечения переменных SVM-RFE (Recursive Feature Extraction Algorithm; Guyon et al., 2002);
- на основе функций `penalizedSVM` или хорошо знакомого нам пакета `caret`.

Для реализации функций `stepclass()` и `svmlight()` из пакета `klaR` необходимо скачать (http://svmlight.joachims.org) и поместить в рабочую директорию бинарные исполняемые файлы метода `SVMlight` (Joachims, 1999), написанные на языке C (эти файлы включены в приложения к этой книге):

```
stepclass(F ~ ., data = DGlass[, -10], method = "svmlight",
          pathsvm = "D:/R/SVMlight")
```

Однако выполнение этой команды оказалось в наших условиях столь продолжительным, что мы не смогли продемонстрировать читателям эту возможность.

Рекурсивное извлечение переменных SVM-RFE происходит с использованием специального критерия, оценивающего релевантность каждого предиктора по отношению к качеству полученного классификатора. Авторы метода (Guyon et al., 2002) написали и предоставили в открытый доступ (http://citeseer.ist.psu.edu) скрипт небольшой функции `svmrfeFeatureRanking()`, возвращающей ранжированный список переменных. Мы также включили файл с командами этой функции в наш набор скриптов, сопутствующих книге:

```
source("SVM-RFE.r")
featureRankedList = svmrfeFeatureRanking(DGlass[,1:9],
              DGlass$F)
ErrSvm <- sapply(1:9, function(nf)  {
     svmModel = svm(DGlass[,featureRankedList[1:nf]],
     DGlass$F,   kernel="linear")
     mean( predict(svmModel) != DGlass$F )  }  )
```

```
data.frame(Index=featureRankedList, NameFeat =
       names(DGlass[,featureRankedList]), ErrSvm = ErrSvm)
```

```
  Index NameFeat    ErrSvm
1     4       Al 0.2699387
2     6        K 0.2883436
3     9       Fe 0.2515337
4     8       Ba 0.2576687
5     3       Mg 0.2515337
6     5       Si 0.2576687
7     7       Ca 0.2331288
8     2       Na 0.2331288
9     1       RI 0.2576687
```

Отметим, что метод опорных векторов слабо зависит от эффектов коллинеарности предикторов и, по сравнению с LDA, значительно меньше переменных было исключено из модели как малоинформативные. Построим гиперплоскость на основе 7 отобранных признаков:

```
DGlass.sel <- DGlass[, c(featureRankedList[1:7], 11)]
(svm.sel <- svm(formula = F ~ ., data=DGlass.sel, cross=10,
       kernel = "linear", prob=TRUE))
table(Факт=DGlass.sel$F, Прогноз=predict(svm.sel))
Acc = mean(predict(svm.all) == DGlass$F )
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
SVM-Kernel:  linear
      cost:  1
     gamma:  0.1428571
Number of Support Vectors:  109
    Прогноз
Факт  1  2
   1 74 13
   2 25 51
[1] "Точность=76.69%"
```

Действительно, число правильно классифицированных образцов оказалось на 4 единицы больше, чем при полном наборе признаков.

Рассмотрим процедуры селекции признаков и тестирования моделей с использованием пакета `caret`. Обратим внимание, что для построения модели опорных векторов здесь используется функция `svm()` из другого пакета – `kernlab`:

```
DGlass$F1 <- as.factor(ifelse(DGlass$Class==2,"No","Flash"))
svmProfile <- rfe(DGlass[,1:9], DGlass$F1, sizes = 2:9,
    rfeControl = rfeControl(functions = caretFuncs,
    method = "cv"), method = "svmLinear")
```

```
Recursive feature selection
Variables Accuracy   Kappa AccuracySD KappaSD Selected
         2   0.5702 0.08002    0.03966 0.09278
         3   0.6180 0.19280    0.07887 0.16994
         4   0.7108 0.41206    0.11048 0.21729
         5   0.7285 0.44707    0.12608 0.25065        *
         6   0.6991 0.38878    0.09543 0.18624
         7   0.6991 0.39014    0.10365 0.20313
         8   0.7175 0.42935    0.11787 0.23245
         9   0.6670 0.32381    0.11461 0.22449
The top 5 variables (out of 5):
   Al, K, Fe, Si, Ba
```

Информативный набор предикторов, полученный методом рекурсивного исключения, состоит из 5 признаков.

Выполним тестирование моделей на основе разного числа предикторов (9, 7 и 5) с использованием показателя AUC (площади под ROC-кривой – см. раздел 2.4), для чего в `trainControl()` зададим параметр `summaryFunction = twoClassSummary`:

```
ctrl <- trainControl(method="repeatedcv",repeats=5,
     summaryFunction=twoClassSummary, classProbs=TRUE)
print(" Модель на основе всех 9 предикторов ")
train(DGlass[,1:9], DGlass$F1, method = "svmLinear",
                metric="ROC", trControl=ctrl)
print(" Модель на основе 7 предикторов ")
train(DGlass[,c(4,6,9,8,3,5,7)], DGlass$F1,
   method = "svmLinear", metric="ROC", trControl=ctrl)
print(" Модель на основе 5 предикторов ")
train(DGlass[,c(4,6,9,8,5)], DGlass$F1, method = "svmLinear",
                metric="ROC", trControl=ctrl)
```

```
Модель на основе всех 9 предикторов
163 samples
  9 predictor
  2 classes: 'Flash', 'No'
  ROC        Sens       Spec
  0.7564236  0.7730556  0.5703571

Модель на основе 7 предикторов
  ROC       Sens       Spec
  0.755119  0.8083333  0.6232143

Модель на основе 5 предикторов
  ROC        Sens       Spec
  0.6986111  0.8169444  0.5396429
```

Последняя модель на основе RFS-метода оказалась наименее качественной по AUC, но имеет наибольшую чувствительностью `Sens`.

6.3. Ядерные функции машины опорных векторов

При наличии нелинейной связи между признаками и откликом качество линейных классификаторов, как показано на рис. 6.4, часто может оказаться неудовлетворительным. Для учета нелинейности обычно расширяют пространство переменных, включая различные функциональные преобразования исходных предикторов (полиномы, экспоненты и проч.). Машину опорных векторов SVM (Support Vector Machine) можно рассматривать как нелинейное обобщение линейного классификатора, представленного в разделе 6.2, основанное на расширении размерности исходного пространства предикторов с помощью специальных *ядерных функций*. Это позволяет строить модели с использованием разделяющих поверхностей самой различной формы.

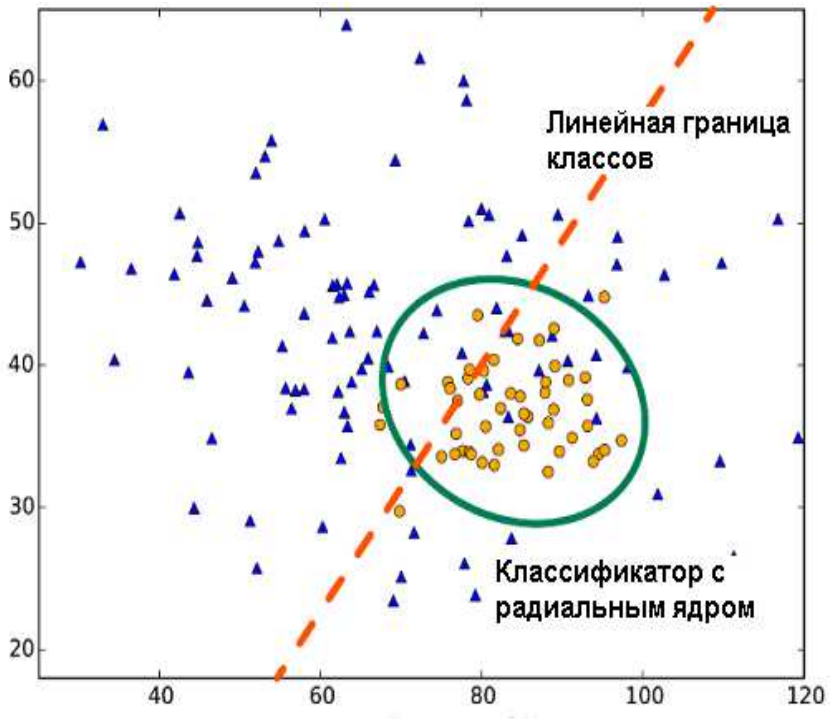


Рис. 6.4. Линейная и радиальная границы классов

Собственно ядром является любая симметричная, положительно полуопределенная матрица **K**, которая составлена из скалярных произведений пар векторов $\mathbf{x}_i$ и $\mathbf{x}_j$: $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, характеризующих меру их близости. Здесь ϕ – произвольная преобразующая функция, формирующая ядро. В качестве таких функций чаще всего используют следующие:

- линейное ядро: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$, что соответствует классификатору на опорных векторах в исходном пространстве (см. рис. 6.3);
- полиномиальное ядро со степенью p: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j)^p$;
- гауссово ядро с радиальной базовой функцией (RBF):
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\{\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\};$$
- сигмоидное ядро: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + \beta_0)$.

Каждое ядро характеризуется параметрами (p, γ, β_0 и т.д.), которые подлежат оптимизации.

Основная идея использования ядер заключается в том, что при отображении данных в пространство более высокой размерности исходное множество точек может стать линейно разделимым (Cristianini, Shawe-Taylor, 2000). Тогда естественно предположить, что оптимальная разделяющая гиперплоскость машины опорных векторов может быть найдена путем подбора коэффициентов α_i выражения $z_k(x) = \sum_{i=1}^{p} \alpha_i K(x, x_i) + \beta_0$. Таким образом, как и в линейном случае, решается математически удобная задача квадратичной оптимизации с использованием множителей Лагранжа. Вычисление экстремума в расширенных пространствах столь огромных размерностей оказалось также возможным потому, что ядро формируется только для ограниченного набора опорных векторов.

Построение классификатора на опорных векторах с использованием перечисленных выше ядер можно осуществить с помощью знакомой нам функции `svm()` из пакета `e1071`. Отметим, что по умолчанию в этой функции установлено значение параметра `kernel = "radial"`, т.е. принято RBF-ядро.

Для подгонки модели распознавания типа стекла (разделы 6.1-6.2) нам необходимо предварительно оценить значения двух параметров: C (`cost`) – допустимый штраф за нарушение границы зазора и γ (`gamma`) – параметр радиальной функции. При увеличении C происходит уменьшение зазора и количества используемых опорных векторов: граница между классами становится более извилистой, а классификатор более точным на обучающей выборке и менее точным – на контрольной выборке. В состав пакета входит функция `tune.svm()`, которая по информации о строящейся модели и с использованием перекрестной проверки осуществляет предварительный отбор необходимых параметров:

```
DGlass <- read.table(file="Glass.txt", sep=",",
                  header=TRUE,row.names=1)
DGlass$F <- as.factor(ifelse(DGlass$Class==2,2,1))
library(e1071)
tune.svm(F ~ ., data=DGlass[,-10], gamma = 2^(-1:1),
                        cost = 2^(2:4))
```

```
Parameter tuning of ‘svm’:
- sampling method: 10-fold cross validation
- best parameters:
 gamma cost
   0.5    4
- best performance: 0.1886029
```

```
svm.rbf <- svm(formula = F ~ ., data=DGlass[,-10],
             kernel = "radial",cost = 4, gamma = 0.5)
table(Факт=DGlass$F, Прогноз=predict(svm.rbf))
Acc <- mean(predict(svm.rbf) == DGlass$F )
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
     Прогноз
Факт  1  2
   1 85  2
   2  7 69
[1] "Точность=94.48%"
```

Мы получили классификатор, значительно превышающий по точности рассмотренные выше модели линейного дискриминантного анализа и опорных векторов с линейным ядром. Полученная разделяющая поверхность имеет многомерный характер, но мы можем оценить ее внешний вид, создавая двумерные проекции любых двух исходных переменных из 9 (например, концентрации алюминия и магния):

```
svm2.rbf <- svm(formula = F ~ Al+Mg, data=DGlass[,-10],
   kernel = "radial",cost = 4, gamma = 2.5)
Acc <- mean(predict(svm2.rbf) == DGlass$F )
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
[1] "Точность=85.89%"
```

```
plot(svm2.rbf, DGlass[,c(3,4,11)],
svSymbol = 16, dataSymbol = 2, color.palette = terrain.colors)
legend("bottomleft", c("Опорные кл.1", "Опорные кл.2",
       "Данные кл.1", "Данные кл.2"), col=c(1,2,1,2),
        pch=c(16,16,2,2))
```

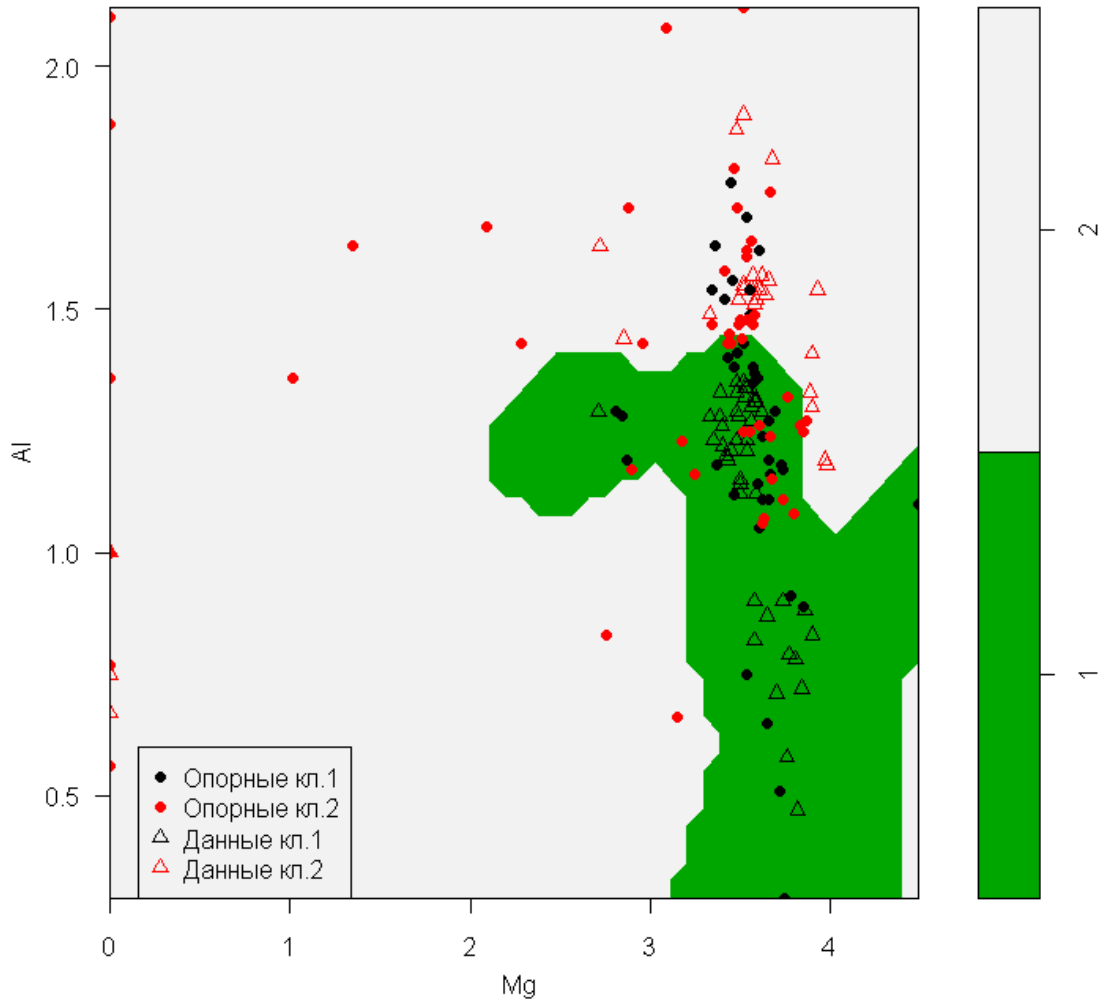


Рис. 6.5. Пример классификатора на опорных векторах с использованием в качестве ядра радиальной базисной функции

Выполним аналогичные действия с использованием функции `train()` из пакета `caret`, где для построения модели опорных векторов используется функция `svm()` из пакета `kernlab`. Здесь также осуществляется подбор двух параметров – `C` и `sigma` (вероятно, то же самое, что и `gamma`). Определим диапазоны их варьирования в таблице `grid`: `C` – от 2 до 5, а `sigma` – от 0.3 до 0.6:

```
ctrl <- trainControl(method="repeatedcv",repeats=5,
   summaryFunction=twoClassSummary, classProbs=TRUE)
grid <- expand.grid(sigma = (3:6)/10, C = 2:5)
(svmRad.tune <- train(DGlass[,1:9], DGlass$F1,
   method = "svmRadial", metric="ROC",
   tuneGrid = grid, trControl=ctrl))
```

```
  sigma  C  ROC        Sens       Spec
  0.3    2  0.8808209  0.8211111  0.7957143
  0.3    3  0.8832316  0.8300000  0.8110714
  0.3    4  0.8780878  0.8372222  0.8053571
  0.3    5  0.8761682  0.8300000  0.8078571
  0.4    2  0.8864906  0.8300000  0.8242857
  0.4    3  0.8766642  0.8297222  0.8189286
  0.4    4  0.8692634  0.8322222  0.8164286
  0.4    5  0.8648983  0.8344444  0.8192857
  0.5    2  0.8797693  0.8300000  0.8221429
  0.5    3  0.8730729  0.8247222  0.8189286
  0.5    4  0.8693080  0.8180556  0.8192857
  0.5    5  0.8623041  0.8158333  0.8085714
  0.6    2  0.8826364  0.8230556  0.8192857
  0.6    3  0.8721453  0.8063889  0.8192857
  0.6    4  0.8663715  0.8066667  0.8167857
  0.6    5  0.8638814  0.7950000  0.8142857

ROC was used to select optimal model using  the largest value.
The final values used for model were sigma = 0.4 and C = 2.
```

```
pred <- predict(svmRad.tune,DGlass[,1:9])
table(Факт=DGlass$F1, Прогноз=pred)
Acc <- mean(pred == DGlass$F1)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
       Прогноз
Факт    Flash No
  Flash    83  4
  No        9 67
[1] "Точность=92.02%"
```

При рассмотрении приведенного примера (рис. 6.5) может создаться впечатление, что полученные нелинейные плоскости хорошо зарекомендовали себя при предсказании, но трудно интерпретируемы с точки зрения объяснения. Так и есть. Однако основная задача подобных моделей – научиться распознавать образы (pattern recognition), что нашло широкое применение в вычислительной биологии, генетике, спектроскопии, анализе данных, полученных с использованием микрочипов (microarray) и т.д.

Большие возможности использования разнообразных ядерных функций представлены в пакете `kernlab`. Рассмотрим пример, представленный в документации к пакету и связанный с задачей распознавания двух вложенных друг в друга спиралей:

```
library('kernlab') ; library('ggplot2')
set.seed(2335246L) ; data('spirals')
#  Спектральная функция выделяет две различные спирали
sc <- specc(spirals, centers = 2)
s <- data.frame(x=spirals[,1],y=spirals[,2],
                class=as.factor(sc))
#  Данные делятся на обучающую и проверочную выборки
s$group <- sample.int(100,size=dim(s)[[1]],replace=T)
sTrain <- subset(s,group>10)
sTest <- subset(s,group<=10)
#  Снова используем Гауссово или радиальное ядро
mSVMG <- ksvm(class~x+y,data=sTrain,kernel='rbfdot')
sTest$predSVMG <- predict(mSVMG,newdata=sTest,type='response')
```

```
table(Факт=sTest$class, Прогноз=sTest$predSVMG)
```

```
     Прогноз
Факт  1  2
   1 13  1
   2  2 13
```

```
ggplot() +
   geom_text(data=sTest, aes(x=x, y=y,
      label=predSVMG), size=12) +
   geom_text(data=s, aes(x=x, y=y,
      label=class, color=class), alpha=0.7) +
   coord_fixed() +
   theme_bw() + theme(legend.position='none')
```

В результате использования радиальной ядерной функции была построена разделяющая поверхность между двумя вложенными спиралями таким образом, что ошибочное предсказание правильного класса наблюдалось лишь для 3 объектов проверочной выборки из 29.

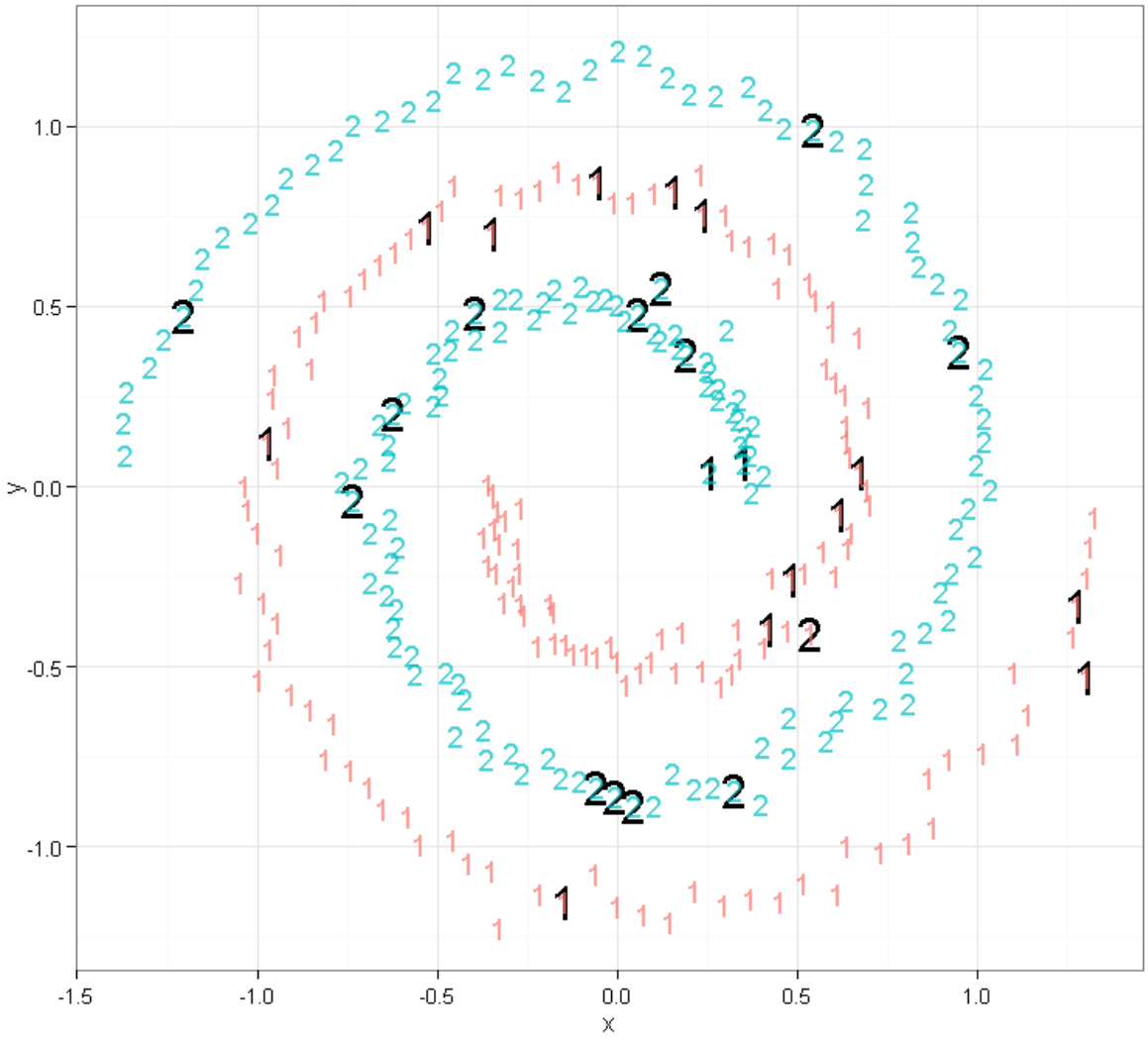


Рис. 6.6. Использование радиального ядра для разделения двух вложенных спиралей

6.4. Деревья классификации, случайный лес и логистическая регрессия

Выполним построение еще трех бинарных классификаторов для предсказания марки стекла на основе методов, использованных нами в главе 4 при выполнении регрессионного анализа.

Как упоминалось ранее, деревья решений CART могут эффективно применяться и для прогнозирования бинарного отклика. При построении дерева важно установить оптимальный уровень ветвления и выбрать значение относительного параметра стоимости сложности cp. Визуализацию дерева

```
DGlass <- read.table(file=»Glass.txt», sep=",",
                  header=TRUE,row.names=1)
DGlass$F1 <- as.factor(ifelse(DGlass$Class==2,"No","Flash"))
control<-trainControl(method="repeatedcv",number=10,repeats=3)
set.seed(7)
fit.cart <- train(DGlass[,1:9], DGlass$F1,
                  method="rpart", trControl=control)
library(rpart.plot)
rpart.plot(fit.cart$finalModel)
```

выполним с помощью пакета rpart.plot (рис. 6.7):

```
Resampling: Cross-Validated (10 fold, repeated 3 times)
  cp          Accuracy   Kappa
  0.05263158  0.7822549  0.5586296
  0.13157895  0.7577124  0.5022827
  0.46052632  0.6075490  0.1805564
Accuracy was used to select optimal model using largest value.
The final value used for the model was cp = 0.05263158.
```

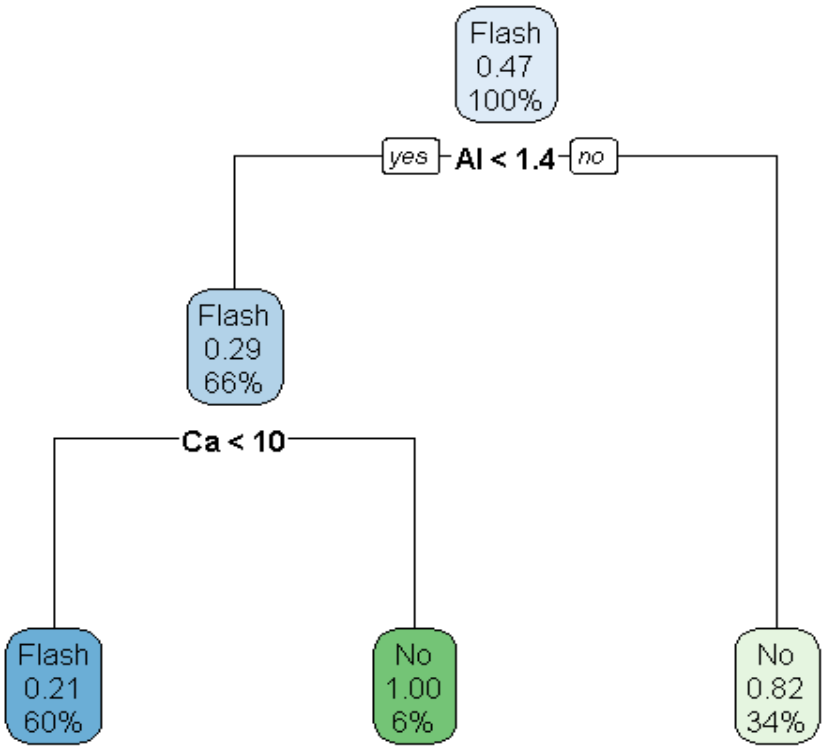


Рис. 6.7. Дерево rpart для прогнозирования типа стекла

В узлах и листьях дерева на рис. 6.7 указаны доля примеров в этой группе от объема всей обучающей выборки (%) и вероятность прогноза "не флеш-стекло". Оценим адекватность модели по точности прогноза на обучающей выборке:

```
pred <- predict( fit.cart,DGlass[,1:9])
table(Факт=DGlass$F1, Прогноз=pred)
Acc <- mean(pred == DGlass$F1)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
        Прогноз
Факт    Flash No
  Flash    77 10
  No       21 55
[1] "Точность=80.98%"
```

Метод *случайного леса* (random forest) осуществляет бутстреп-агрегирование некоторого множества деревьев решений. Для создания оптимального ансамбля необходимо выбрать значение `mtry`, определяющее число исходных переменных, которые участвуют в выращивании деревьев:

```
set.seed(7)
fit.rf <- train(DGlass[,1:9], DGlass$F1,
                method="rf", trControl=control)
fit.rf$finalModel
pred <- predict(fit.rf, DGlass[,1:9])
table(Факт=DGlass$F1, Прогноз=pred)
Acc <- mean(pred == DGlass$F1)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
Random Forest
  mtry  Accuracy   Kappa
  2     0.8801879  0.7575337
  5     0.8556618  0.7075469
  9     0.8295588  0.6551760
Accuracy was used to select optimal model using largest value.
The final value used for the model was mtry = 2.

Call:
 randomForest(x = x, y = y, mtry = param$mtry)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2
        OOB estimate of  error rate: 12.27%
Confusion matrix:
      Flash No class.error
Flash    81  6  0.06896552
No       14 62  0.18421053

       Прогноз
Факт    Flash No
  Flash    87  0
  No        0 76
 [1] "Точность=100%"
```

Из протокола расчетов видно, что: *а*) на каждом шаге ветвления выбор идет только из двух случайных переменных, *б*) в итоге был составлен ансамбль из 500 деревьев, *в*) средняя ошибка по наблюдениям, не попавшим в

"сумку" (OOB error), равна 12.27%, *г*) модель правильно предсказывает класс всех объектов обучающей выборки.

Приступим теперь к построению логистической регрессии LR, как это было сделано в разделе 5.1:

```
DGlass$F1 <- ifelse(DGlass$Class==2,1,0)
logit.all <- glm(F1 ~ .,data=DGlass[,-10], family=binomial)
mp.all <- predict(logit.all, type="response")
Acc <- mean(DGlass$F1 == ifelse(mp.all>0.5,1,0))
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
[1] "Точность=73.62%"
```

Получим теперь ошибку перекрестной проверки по методу скользящего контроля с использованием функции cv.glm() из пакета boot. Обратим внимание на необходимость функции cost, которая пересчитывает значение delta из ошибки оценки вероятностей в ошибку предсказания класса:

```
library(boot)
cost <- function(r, pi = 0) mean(abs(r-pi) > 0.5)
Acc <- 1 - cv.glm(DGlass[,-10], logit.all, cost)$delta[1]
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
 [1] "Точность=68.72%"
```

Выполним пошаговую процедуру селекции независимых переменных и оценим ошибки предсказания:

```
# Регрессия на информативные переменные
logit.step <- step(logit.all)
summary(logit.step)
```

```
Start:  AIC=185.56
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -328.3768   352.0813  -0.933 0.350989
RI           472.1567   235.8428   2.002 0.045285 *
Na            -3.6626     1.1071  -3.308 0.000939 ***
Mg            -5.9264     1.1531  -5.140 2.75e-07 ***
Si            -3.7897     0.9963  -3.804 0.000143 ***
K             -3.1497     2.1582  -1.459 0.144455
Ca            -4.9433     1.1783  -4.195 2.73e-05 ***
Ba            -5.6440     3.0682  -1.840 0.065838 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
    Null deviance: 225.22  on 162  degrees of freedom
Residual deviance: 166.92  on 155  degrees of freedom
AIC: 182.92
```

```
# Ошибка на обучающей выборке
mp.step <- predict(logit.step, type="response")
Acc <- mean(DGlass$F1== ifelse(mp.step>0.5,1,0))
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
[1] "Точность=71.78%"
```

```
# Ошибка при скользящем контроле
Acc <- 1 - cv.glm(DGlass[,-10], logit.step, cost)$delta[1]
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
[1] "Точность=69.33%"
```

Исключение нескольких переменных не привело к улучшению характеристик модели логистической регрессии. Если использовать метод RFE, то можно придти к тому же подмножеству предикторов, а тестирование моделей с применением функции `train()` приведет примерно к такому же соотношению эффективности обоих моделей (читатели могут проверить это самостоятельно).

6.5. Процедуры сравнения эффективности моделей классификации

Итак, мы использовали шесть методов для построения моделей бинарной классификации. Осуществим их сравнение для данного примера с полным сохранением всех канонов тестирования моделей прогнозирования (Kuhn, Johnson, 2013) и в соответствии с методикой, представленной в сообщении http://machinelearningmastery.com:

1. Проводим разделение имеющейся выборки на обучающую и проверочную (использование фактора в качестве аргумента функции `createDataPartition` обеспечивает необходимый баланс его уровней при разделении). Для обучения выделим 70% (115 элементов) исходной выборки:

```
set.seed(7)
train <- unlist(createDataPartition(DGlass$F1, p=0.7))
# определение схемы тестирования
control <- trainControl(method="repeatedcv",
    number=10, repeats=3, classProbs = T)
```

2. Выполняем обучение шести моделей:

```
# LDA - линейный дискриминантный анализ
set.seed(7)
fit.lda <- train(DGlass[train,3:4], DGlass$F1[train],
       method="lda", trControl=control)
# SVML - метод опорных векторов с линейным ядром
set.seed(7)
fit.svL <- train(DGlass[train,1:9], DGlass$F1[train],
      method="svmLinear", trControl=control)
# SVMR  - метод опорных векторов с радиальным ядром
set.seed(7)
fit.svR <- train(DGlass[train,1:9], DGlass$F1[train],
    method="svmRadial", trControl=control,
    tuneGrid = expand.grid(sigma = 0.4, C = 2))
# CART - дерево классификации
set.seed(7)
fit.cart <- train(DGlass[train,1:9], DGlass$F1[train],
    method="rpart", trControl=control)
# RF - случайный лес
set.seed(7)
fit.rf <- train(DGlass[train,1:9], DGlass$F1[train],
    method="rf", trControl=control)
# GLM - Логистическая регрессия
set.seed(7)
fit.glm <- train(DGlass[train,-c(1,4,10,11)],DGlass$F1[train],
 method="glm", family=binomial, trControl=control)
```

3. Осуществляем ресэмплинг полученных моделей, выполняем статистический анализ двух сформированных метрик качества моделей (Accuracy и индекс Kappa Дж. Коэна) на обучающей выборке, ранжируем модели и оцениваем доверительные интервалы использованных критериев:

```
caret.models <-  list(LDA=fit.lda, SVML=fit.svL,
    SVMR=fit.svR, CART=fit.cart, RF=fit.rf, GLM=fit.glm)
# ресэмплинг коллекции моделей
results <- resamples(caret.models)
# обобщение различий между моделями
summary(results, metric = "Accuracy")
#  Оценка доверительных интервалов и построение графика
scales <- list(x=list(relation="free"),
                        y=list(relation="free"))
dotplot(results, scales=scales)
```

```
Number of resamples: 30
Accuracy
       Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
LDA  0.4545  0.6818 0.7273 0.7314  0.8295 0.9167    0
SVML 0.2500  0.6364 0.6795 0.6989  0.8182 0.9167    0
SVMR 0.3636  0.7500 0.8182 0.7809  0.8333 0.9231    0
CART 0.4545  0.7273 0.8182 0.7610  0.8295 0.9231    0
RF   0.6364  0.8365 0.9091 0.8958  0.9808 1.0000    0
GLM  0.3333  0.6206 0.6667 0.6959  0.8182 1.0000    0
```

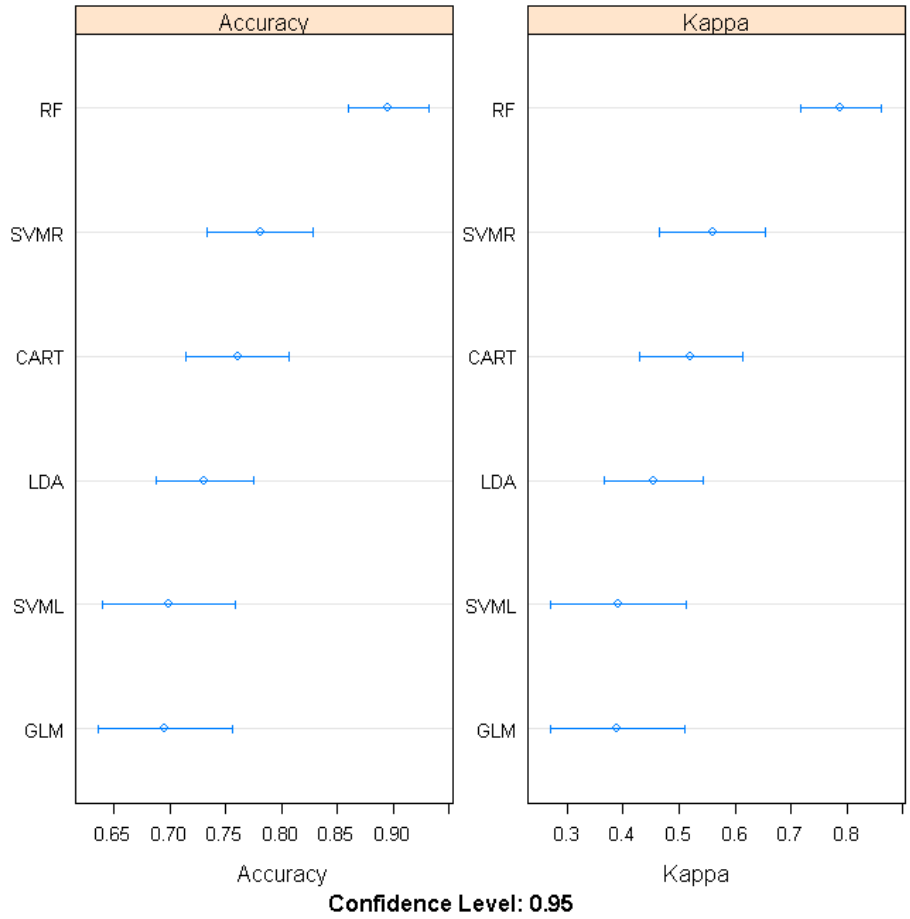


Рис. 6.8. Сравнительный статистический анализ двух метрик эффективности шести моделей прогнозирования типа стекла

4. Наконец, можно вычислить статистическую значимость различий между моделями на основе распределения оценок критериев качества. Для этого воспользуемся функцией `diff()`:

```
diffs <- diff(results)
# summarize p-values for pair-wise comparisons
summary(diffs)
```

```
p-value adjustment: bonferroni
Accuracy
     LDA       SVML      SVMR      CART      RF        GLM
LDA             0.032440 -0.049573 -0.029584 -0.164472  0.035509
SVML 1.00000             -0.082012 -0.062024 -0.196911  0.003069
SVMR 0.08830   0.04908              0.019988 -0.114899  0.085082
CART 0.22316   0.42928   1.00000             -0.134887  0.065093
RF   5.143e-09 2.525e-06 7.359e-05 1.335e-07            0.199981
GLM  1.00000   1.00000   0.03557   0.20047   1.763e-07
```

Результаты выше главной диагонали таблицы статистической значимости содержат разности между центрами распределения. Эти значения показывают, как модели соотносятся друг с другом по абсолютному значению точности. Данные ниже диагонали таблицы представляют собой *p*-значения для нулевой гипотезы, которая утверждает, что параметры распределения одинаковы. Например, очевидно, что модель Random Forrest статистически значимо точнее всех остальных.

5. Составим таблицу прогноза всеми шестью моделями для 48 проверочных наблюдений:

```
pred.fm <- data.frame(
    LDA=predict(fit.lda,DGlass[-train,3:4]),
    SVML=predict(fit.svL, DGlass[-train,1:9]),
    SVMR=predict(fit.svR, DGlass[-train,1:9]),
    CART=predict(fit.cart, DGlass[-train,1:9]),
    RF=predict(fit.rf, DGlass[-train,1:9]),
    GLM=predict(fit.glm, DGlass[-train,-c(1,4,10,11)])
)
```

Резерв повышения надежности моделей прогнозирования многими исследователями видится в объединении отдельных моделей в "ансамбль" (ensemble) и построении системы коллективного распознавания.

Простейшим способом комбинирования результатов бинарной классификации является выбор класса, за который "проголосовало" большинство предсказывающих моделей. Добавим в таблицу прогнозов столбец с результатами голосования (методу Random Forrest, как наиболее эффективному предсказателю, отдано два голоса):

```
CombyPred <- apply(pred.fm, 1, function (voice) {
     voice2 <- c(voice, voice[5]) # У RF двойной голос
     ifelse(sum(voice2=="Flash") > 3,"Flash", "No") }
        )
pred.fm <- cbind(pred.fm, COMB=CombyPred)
head(pred.fm)
```

```
     LDA  SVML  SVMR  CART    RF   GLM  COMB
1  Flash    No    No Flash    No    No    No
2     No    No    No    No    No    No    No
3  Flash    No Flash Flash Flash    No Flash
4  Flash Flash Flash Flash Flash Flash Flash
5  Flash Flash Flash Flash Flash Flash Flash
6  Flash Flash Flash Flash Flash Flash Flash
```

6. Определимся со списком критериев, которые следует использовать для тестирования. Кроме традиционных оценок точности AC, чувствительности SE и специфичности SP, воспользуемся дополнительными метриками, рекомендованными в обзоре (http://datareview.info) и позволяющими получить несмещенную оценку, особенно в случае несбалансированных классов:

$$F\text{-мера (F1 score)} = 2*AC * SE / (AC + SE)$$

и *коэффициент корреляции Мэтьюса* (MCC, Matthews correlation coefficient)

$$MCC = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

(условные обозначения см. в разделе 2.3).

```
# Функция формирования строки критериев
ModCrit <- function (fact, pred) {
cM <- table(fact, pred)
c( Accur <- (cM[1,1]+cM[2,2])/sum(cM),
   Sens <- cM[1,1]/(cM[1,1]+cM[2,1]),
   Spec <- cM[2,2]/(cM[2,2]+cM[1,2]),
   F.score <-  2 * Accur * Sens / (Accur + Sens),
   MCC <- ((cM[1,1]*cM[2,2])-(cM[1,2]*cM[2,1])) /
        sqrt((cM[1,1]+cM[1,2])*(cM[1,1]+cM[2,1])*
        (cM[2,2]+cM[1,2])*(cM[2,2]+cM[2,1])) )
}
Result <- t(apply(pred.fm,2, function (x)
                  ModCrit(DGlass$F1[-train], x)))
colnames(Result) <- c("Точность","Чувствит.","Специфичн.",
            "F-мера","КК Мэтьюса")
round(Result,3)
```

```
     Точность Чувствит. Специфичн.  F-мера КК Мэтьюса
LDA     0.688     0.704      0.667   0.696      0.369
SVML    0.562     0.600      0.522   0.581      0.122
SVMR    0.792     0.786      0.800   0.789      0.580
CART    0.708     0.714      0.700   0.711      0.410
RF      0.771     0.800      0.739   0.785      0.541
GLM     0.542     0.571      0.500   0.556      0.071
COMB    0.688     0.690      0.684   0.689      0.367
```

По существу проведенных расчетов можно сделать два вывода:

- при тестировании на "свежих" данных модель Random Forrest уступает по эффективности модели опорных векторов с радиальным ядром;
- прогноз "голосующего коллектива" COMB имеет умеренные показатели точности предсказания на фоне индивидуальных моделей.

В разделе 2.3 было показано, что универсальным способом сравнения точности классификаторов является ROC-анализ. Воспользовавшись тем, что при обучении мы сохранили вероятности принадлежности к классам

(classProbs = T), построим ROC-кривые для трех используемых методов распознавания (LDA, SVMR и RF) по полной таблице исходных данных, включая обучающую и тестовую последовательности. Будем анализировать вероятности отнесения к классу Flash, т.е. 1-й столбец таблицы вероятностей, возвращаемой функцией predict() при значении параметра type = "prob":

```
# Извлекаем вероятности классов по всей выборке
pred.lda.roc <- predict(fit.lda,DGlass[,3:4],type = "prob")
pred.svmR.roc <- predict(fit.svR,DGlass[,1:9],type = "prob")
pred.rf.roc <- predict(fit.rf,DGlass[,1:9],type = "prob")
library(pROC)          # Строим три ROC-кривые
m1.roc <- roc(DGlass$F1, pred.lda.roc[,1])
m2.roc <- roc(DGlass$F1, pred.svmR.roc[,1])
m3.roc <- roc(DGlass$F1, pred.rf.roc[,1])
plot(m1.roc, grid.col=c("green", "red"), grid=c(0.1, 0.2),
      print.auc=TRUE,print.thres=TRUE)
plot(m2.roc , add = T, col="green", print.auc=T,
      print.auc.y=0.45,print.thres=TRUE)
plot(m3.roc , add = T, col="blue", print.auc=T,
      print.auc.y=0.40,print.thres=TRUE)
legend("bottomright", c("LDA","SVM","RF","COMBY"),lwd=2,
      col=c("black","green","blue","red"))
```

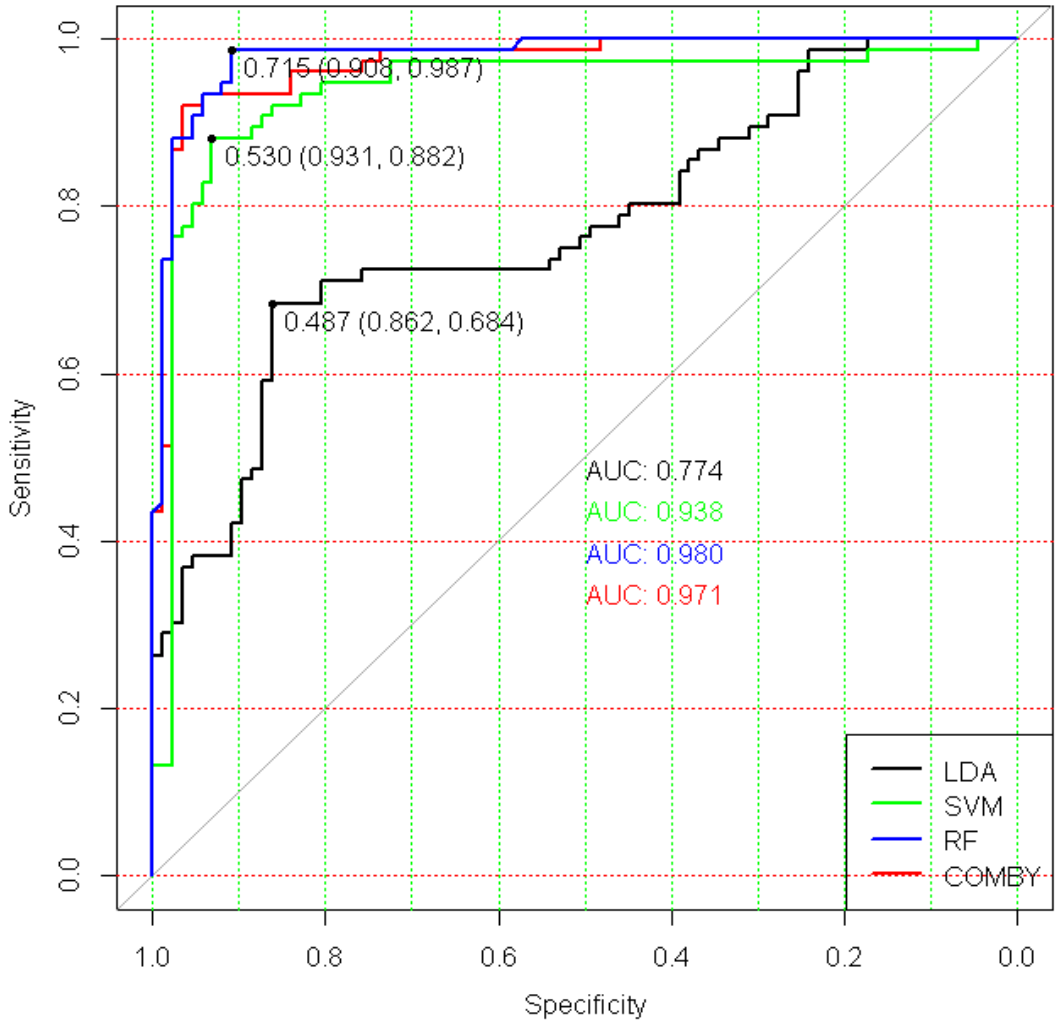


Рис. 6.9. ROC-кривые для четырех моделей классификации

Нетрудно заметить, что каждая модель, в силу характерных для нее стратегий распознавания, имеет различную эффективность классификации на каждом из диапазонов значений апостериорных вероятностей p класса `Flash`. Например, классификатор SVM на опорных векторах имеет определенное преимущество при предсказании объектов в критической зоне разделения (p = 0.4 ÷ 0.6) и при пороговом значении p = 0.53 он достигает максимальной точности (чувствительность `SE` = 93.1% и специфичность `SP` = 88.2%). В других диапазонах p эффективность модели SVM несколько уступает остальным классификаторам, что имеет свои объяснения, если вспомнить механизм распознавания на основе опорных векторов.

Максимум значения AUC на рис. 6.8 принадлежит модели случайного леса, но использование функций `ci.auc()` и `roc.test()` даст нам доверительные интервалы для AUC и позволит сделать вывод о значимости различий между классификаторами по этому показателю.

```
# Доверительные интервалы для параметров ROC-анализа:
ci.auc(m2.roc); ci.auc(m3.roc)
roc.test(m2.roc, m3.roc)
```

```
95% CI: 0.8949-0.9814 (DeLong)
95% CI: 0.9619-0.9988 (DeLong)
        DeLong's test for two correlated ROC curves
data:  m2.roc and m3.roc
Z = 2.7057, p-value = 0.006815
```

Результаты свидетельствуют, что по величине AUC метод случайного леса (`m3.roc`) на полной выборке значимо превосходит метод опорных векторов (`m2.roc`). Аналогичные функции `ci.se()` и `ci.sp()` дадут возможность сделать такой же вывод относительно чувствительности и специфичности.

Вернемся к нашим попыткам осуществить коллективный прогноз. Кроме голосования моделей, вторым простым способом комбинирования результатов бинарной классификации является расчет взвешенной средней апостериорной вероятности отнесения объекта к тому или иному классу $p_c = \sum_j w_j p_j$, где p_j – частные вероятности, полученные g различными классификаторами, w_j – веса, нормирующие "компетентность" отдельных моделей. В качестве весов могут использоваться оценки экспертов или любые другие статистики. Будем использовать, например, значения AUC по трем выбранным методам – см. рис. 6.3, предварительно возведенные в квадрат для увеличения "контраста" и нормированные на их сумму:

```
weight <- c(as.numeric(m1.roc$auc)^2,as.numeric(m2.roc$auc)^2,
          as.numeric(m3.roc$auc)^2)
weight <- weight/sum(weight)
Pred=data.frame(lda=pred.lda.roc[,1], svm=pred.svmR.roc[,1],
              rf=pred.rf.roc[,1])
CombyPred <- apply(Pred,1, function (x) sum(x*weight))
```

Построим ROC-кривую для комбинированного прогноза по этому методу и добавим ее на рис. 6.9 – COMBY. С помощью функции coord() можно вывести полную информацию об оптимальной пороговой точке (threshold) полученного классификатора.

```
m.Comby <-roc(DGlass$F1,  CombyPred)
plot(m.Comby, add = T, col="red", print.auc=T,
      print.auc.y=0.35)
coords(m.Comby, "best", ret = c("threshold", "specificity",
                                "sensitivity", "accuracy"))
```

```
threshold specificity sensitivity    accuracy
0.4965734   0.9655172   0.9210526   0.9447853
```

Коллективные прогнозы, полученные как усреднением вероятностей, так и голосованием, имели более низкие показатели точности предсказания по сравнению с лучшими индивидуальными моделями. Видимо, такова объективная закономерность процедуры, когда решения в коллективе принимаются механическим усреднением. Далее мы снова вернемся к обсуждению этой проблемы в главе 8 при рассмотрении функций пакета ForecastCombinations.

7. МОДЕЛИ КЛАССИФИКАЦИИ ДЛЯ НЕСКОЛЬКИХ КЛАССОВ

7.1. Ирисы Фишера и метод *k*-ближайших соседей

Ирисы Фишера – самый популярный в статистической литературе набор данных, часто используемый для иллюстрации работы различных алгоритмов классификации. При всем желании мы не смогли без него обойтись, поскольку в современных реальных приложениях редко встречаются такие компактные наборы данных, позволяющие построить хороший классификатор при минимуме исходных признаков.

Выборка состоит из 150 экземпляров ирисов трех видов (рис. 7.1), для которых измерялись четыре характеристики: длина и ширина чашелистика (`Sepal.Length` и `Sepal.Width`), длина и ширина лепестка (`Petal.Length` и `Petal.Width`). Поставим своей задачей на основе этого набора данных, включенного в базовый дистрибутив R, построить различные модели многоклассовой классификации, прогнозирующие каждый из трех видов растения по данным проведенных измерений.

Ирис щетинистый (*Iris setosa*) | Ирис разноцветный (*Iris versicolor*) | Ирис виргинский (*Iris virginica*)

Рис. 7.1. Виды ирисов в выборке Фишера-Андерсона

Исследуем характер статистической вариации признаков этой выборки с использованием категоризованных диаграмм, где данные сгруппированы по отдельным видам. Из графиков, представленных на рис. 7.2, видно, что размеры лепестка `Petal.Width` являются гораздо более выраженным классифицирующим признаком, чем ширина чашелистика.

```
data(iris) ; library(ggplot2)
qplot(Sepal.Length, Sepal.Width, data = iris) +
      facet_grid(facets = ~ Species)+
      geom_smooth(color="red", se = FALSE)
qplot(Petal.Length, Petal.Width,data = iris) +
      facet_grid(facets = ~ Species)+
        geom_smooth(color="red", se = FALSE)
```

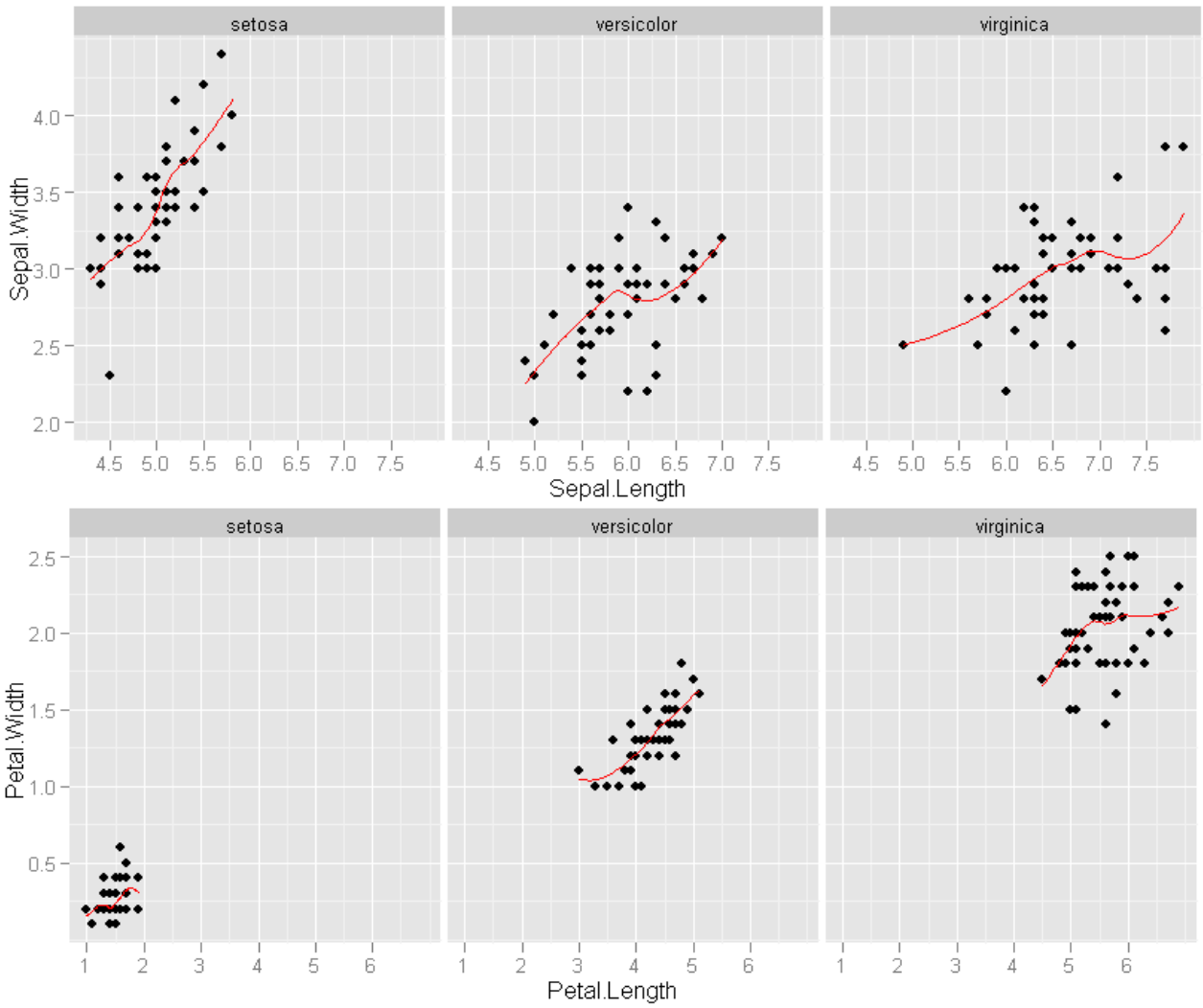


Рис. 7.2. Категоризованная диаграмма выборки ирисов

Выполним теперь визуализацию анализируемых данных в пространстве первых двух главных компонент:

```
library(vegan)
mod.pca <- rda(iris[,-5], scale = TRUE)
scores <- as.data.frame(scores(mod.pca, display ="sites",
                               scaling = 3))
scores$Species <- iris$Species
# Включаем в названия осей доли объясненных дисперсий
axX <- paste("PC1 (",
   as.integer(100*mod.pca$CA$eig[1]/sum(mod.pca$CA$eig)),"%)")
axY <- paste("PC2 (",
   as.integer(100*mod.pca$CA$eig[2]/sum(mod.pca$CA$eig)),"%)")
# Составляем таблицу для "каркаса" точек
l <- lapply(unique(scores$Species), function(c)
        { f <- subset(scores,Species==c); f[chull(f),]})
hull <- do.call(rbind, l)
# Выводим ординационную диаграмму
ggplot() +
  geom_polygon(data=hull, aes(x=PC1,y=PC2, fill=Species),
                                  alpha=0.4, linetype=0) +
  geom_point(data=scores, aes(x=PC1,y=PC2,shape=Species,
                                  colour=Species), size=3) +
```

```
    scale_colour_manual(values = c('purple', 'green', 'blue'))+
      xlab(axX) + ylab(axY) + coord_equal() + theme_bw()
```

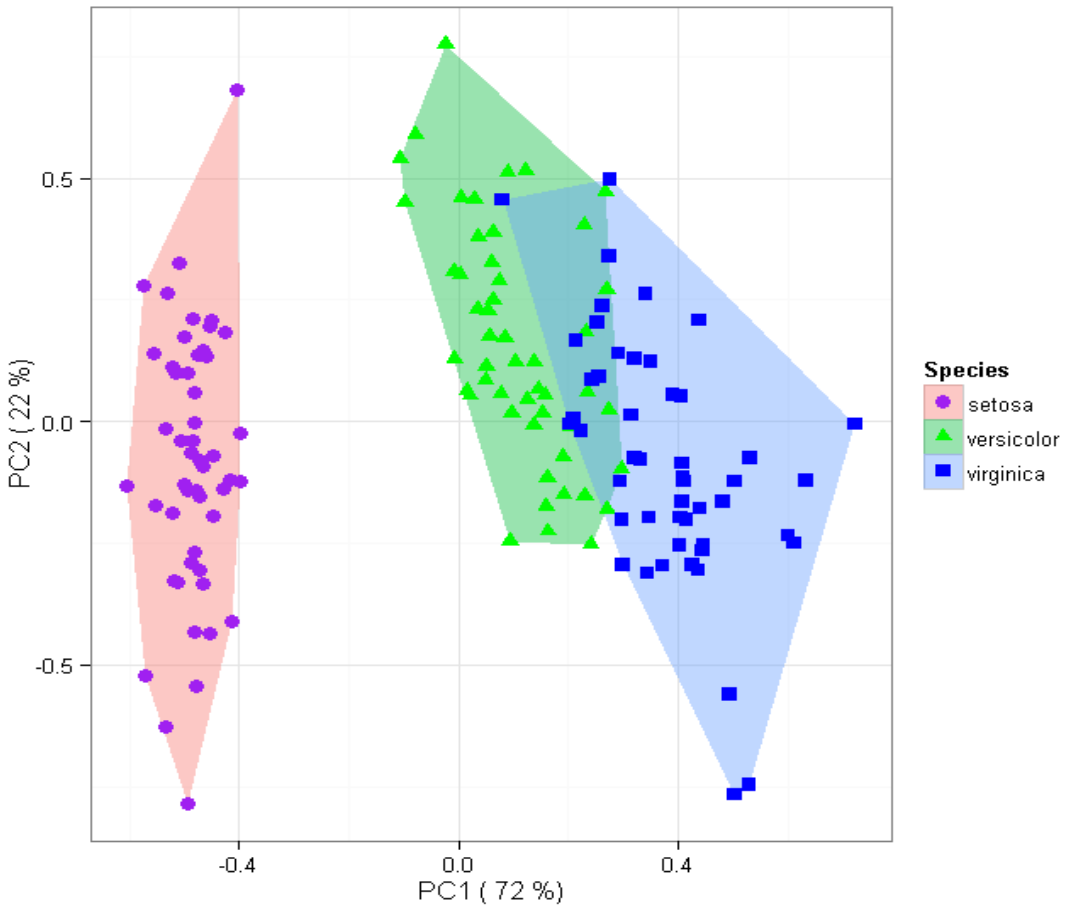


Рис. 7.3. Ординационная диаграмма данных по ирисам в осях двух главных компонент

Как видно на рис. 7.3, один из классов (*Iris setosa*) линейно отделим от двух остальных.

В разделах 3.4 и 4.2 мы использовали простейший, но достаточно эффективный алгоритм *k ближайших соседей* (*k* nearest neighbors), или kNN-регрессию. В основе kNN-классификатора лежит *гипотеза компактности* (Загоруйко, 1999) которая предполагает, что тестируемый объект *d* будет иметь такую же метку класса, как и обучающие объекты в локальной области его ближайшего окружения. В варианте 1NN анализируемый объект относят к определенному классу в зависимости от информации о его ближайшем соседе. В варианте kNN каждый объект относится к преобладающему классу ближайших соседей, где *k* – параметр алгоритма.

Решающие правила в методе kNN определяются границами смежных сегментов диаграммы Вороного (Voronoi resselation), разделяющей плоскость на |*n*| выпуклых многоугольников, каждый из которых содержит один и только один объект обучающей выборки (рис. 7.4). В *p*-мерных пространствах границы решений состоят уже из сегментов (*p* - 1)-мерных полуплоскостей, образованных выпуклыми многогранниками Вороного.

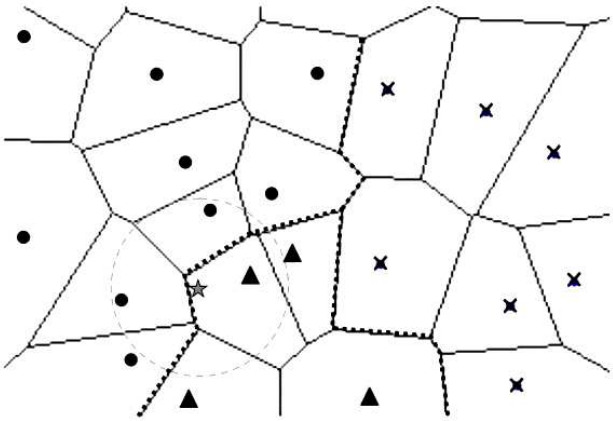


Рис. 7.4. Иллюстрация работы алгоритма *k* ближайших соседей

Алгоритм предсказания строится по принципу "большинства голосов", т.е. в качестве результата объявляется метка класса-победителя. На рис. 7.4 тестируемый объект "звёздочка" попадает в ячейку объекта класса "треугольников" и при $k = 1$ будет отнесён к этому классу. Однако при $k = 3$ по голосам двух ближайших соседей из трех экзаменуемая точка будет отнесена к классу "кружочков". Вероятностный вариант метода kNN использует для ранжирования предполагаемых классов сумму "голосов" соседей с учетом их весов, в частности, евклидовой меры расстояния между тестируемым объектом и каждым из соседей.

Вариант 1NN всегда обеспечивает 100% правильное распознавание примеров обучающей выборки (самый ближайший сосед – он сам), однако часто ошибается на неизвестных ему данных. При увеличении $k > 1$ до некоторых пределов качество распознавания на контрольной выборке будет возрастать. Оптимальное в смысле точности предсказаний значение *k* может быть найдено с использованием перекрестной проверки. Для этого по фиксированному значению *k* строится модель *k*-ближайших соседей и оценивается *CV*-ошибка классификации. Эти действия повторяются для различных *k* и значение, соответствующее наименьшей ошибке распознавания, принимается как оптимальное.

В среде R классификация методом ближайших соседей может быть выполнена функциями `knn()` из базового пакета `class` и `kknn()` из одноименного пакета с расширенными возможностями настройки меры близости и формы используемой ядерной функции. Существует несколько возможностей оценить оптимальное значение *k* с помощью кросс-проверки:

1. Воспользоваться функцией `train.kknn()` из пакета `kknn` (по умолчанию задается скользящий контроль):

```
library(kknn)   #   -------- Метод к-ближайших соседей  kNN
train.kknn(Species ~ ., iris, kmax = 50, kernel="rectangular")
```

```
Type of response variable: nominal
Minimal misclassification: 0.02666667
Best kernel: rectangular
Best k: 16
```

2. Составить собственный скрипт для выполнения перекрестной проверки, например (рис. 7.5):

```
max_K=20 ; gen.err.kknn <- numeric(max_K)
mycv.err.kknn <- numeric(max_K) ; n <- nrow(iris)
#  Рассматриваем число возможных соседей от 1 до 20
for (k.val in 1:max_K)  {
  pred.train <- kknn(Species ~ .,
     train=iris,test=iris,k=k.val,kernel="rectangular")
  gen.err.kknn[k.val] <- mean(pred.train$fit != iris$Species)
  for (i in 1:n)   {
   pred.mycv <- kknn(Species~., train=iris[-i,],test=iris[i,],
         k=k.val,kernel="rectangular")
   mycv.err.kknn[k.val] <- mycv.err.kknn[k.val] +
        (pred.mycv$fit != iris$Species[i])   }
}  ;  mycv.err.kknn <- mycv.err.kknn/n
plot(1:20,gen.err.kknn,type="l",xlab='k', ylim=c(0,0.07),
    ylab='Ошибка классификации', col="limegreen", lwd=2)
points(1:max_K,mycv.err.kknn,type="l",col="red", lwd=2)
legend("bottomright", c("При обучении",
   "Скользящий контроль"), lwd=2, col=c("limegreen", "red"))
```

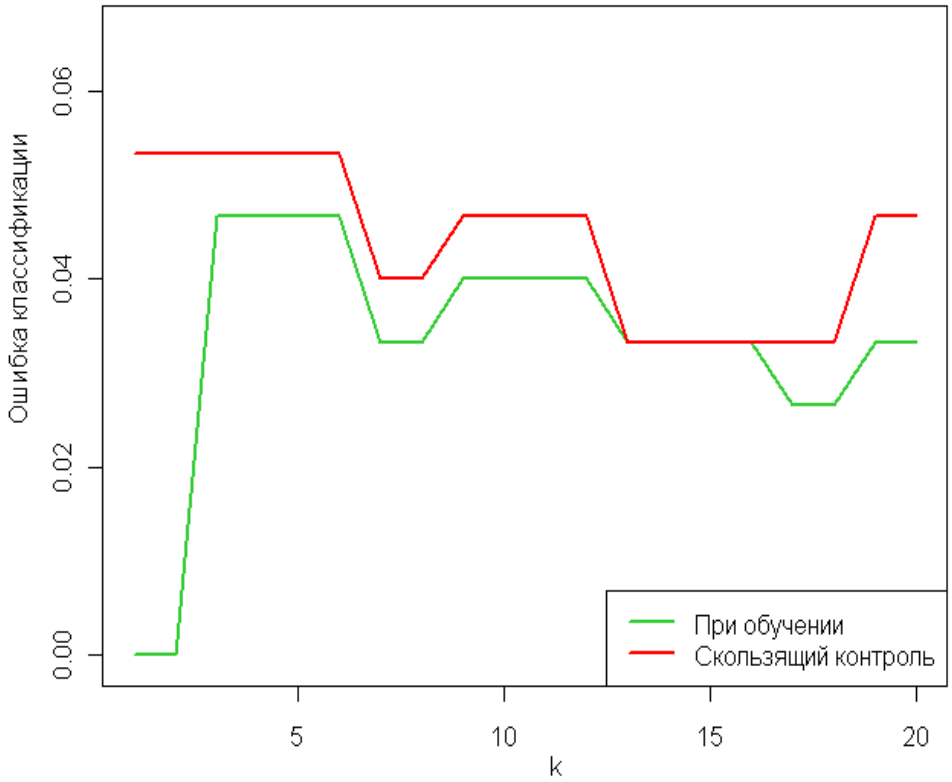


Рис. 7.5. Нахождение оптимального числа *k* ближайших соседей при классификации видов ириса

3. Использовать функцию `train()` из пакета `caret`, с которой мы уже неоднократно сталкивались ранее (см. главу 3):

```
library(caret) ; library(class) ; set.seed(123)
contrl <- trainControl(method="repeatedcv", repeats = 3)
train(Species ~ ., data = iris, method = "kknn",
       trControl = contrl, preProcess = c("center", "scale"),
       tuneLength = 20)
```

```
Resampling: Cross-Validation (10 fold, repeated 3 times)
  k   Accuracy  Kappa  Accuracy SD  Kappa SD
  7   0.96      0.94   0.0414       0.0621
  9   0.96      0.94   0.0414       0.0621
  11  0.962     0.943  0.0453       0.0679
  13  0.967     0.95   0.042        0.063
  15  0.964     0.947  0.0454       0.0681
  17  0.958     0.937  0.0479       0.0718
  19  0.958     0.937  0.0446       0.0669
  21  0.951     0.927  0.0461       0.0691
Accuracy was used to select the optimal model using  the
largest value.The final value used for the model was k = 13.
```

Здесь была осуществлена трехкратная перекрестная проверка с разбивкой на 10 блоков, т.е. случайным образом исходная выборка делилась на 10 частей, после чего поочередно 9 частей использовались для оценки k, а по одной части оценивалась точность классификации, и такая процедура повторялась три раза.

Выполним теперь стандартную процедуру тестирования модели при $k = 13$. Исходную выборку разбиваем на две части: обучающую (`train`) и проверочную (`test`) в соотношении 3:1:

```
set.seed(123) ;  (samp.size <- floor(nrow(iris) * .75))
train.ind <- sample(seq_len(nrow(iris)), size = samp.size)
train <- iris[train.ind,]  ;  test <- iris[-train.ind, ]
```

```
[1] 112
```

Функция `knn()` из пакета `class` сразу на выходе дает предсказываемые значения для тестовой последовательности, которые можно использовать для построения матрицы неточностей:

```
knn.iris <- knn(train = train[,-5], test = test[,-5],
            cl = train[,"Species"], k = 13, prob = T)
table(Факт=test$Species,Прогноз=knn.iris)
Acc = mean(knn.iris == test$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
             Прогноз
Факт          setosa versicolor virginica
  setosa          11          0         0
  versicolor       0         13         0
  virginica        0          1        13
[1] "Точность=97.37%"
```

Метод дал одну ошибку при предсказании 38 "свежих" примеров, что является вполне приличным результатом.

7.2. Наивный байесовский классификатор

В основе байесовской классификации лежит *гипотеза максимальной вероятности*, т.е. объект d считается принадлежащим классу c_j ($c_j \in \mathbf{C}$), если при достигается наибольшая апостериорная вероятность: $\max_{\mathbf{C}} P(c_j \mid d)$. По формуле Байеса,

$$P(c_j \mid d) = \frac{P(c_j)P(d \mid c_j)}{P(d)} \approx P(c_j)P(d \mid c_j),$$

где $P(d|c_j)$ – вероятность встретить объект d среди объектов класса c_j; $P(c_j)$ и $P(d)$ – априорные вероятности класса c_j и объекта d (последняя, не влияет на выбор класса и может быть опущена).

Если сделать "наивное" предположение, что все признаки, описывающие классифицируемые объекты, совершенно равноправны между собой и не связаны друг с другом, то $P(d|c_j)$ можно вычислить как произведение вероятностей встретить признак x_i ($x_i \in \mathbf{X}$) среди объектов класса c_j:

$$P(d \mid c_j) = \prod_{i=1}^{|\mathbf{X}|} P(x_i \mid c_j),$$

где $P(x_i|c_j)$ – вероятностная оценка вклада признака x_i в то, что $d \in c_j$.

На практике при умножении очень малых условных вероятностей может наблюдаться потеря значащих разрядов, в связи с чем вместо самих оценок вероятностей $P(x_i|c_j)$ применяют логарифмы этих вероятностей. Поскольку логарифм – монотонно возрастающая функция, то класс c_j с наибольшим значением логарифма вероятности останется наиболее вероятным. Тогда решающее правило наивного байесовского классификатора (Naive Bayes Classifier) принимает следующий окончательный вид:

$$c^* = \arg_{c_j \in \mathbf{C}} \max \left[\log P(c_j) + \sum_{i=1}^{\mathbf{X}} P(x_i \mid c_j) \right].$$

Остается только позаботиться, чтобы значения логарифмируемых вероятностей были не слишком близки к 0 (чтобы этого не случилось, можно применить лапласовское сглаживание – http://stats.stackexchange.com).

В среде R расчеты выполняются обычно с использованием функций `NaiveBayes()` из пакета `klaR` или `naiveBayes()` из пакета `e1071`. Рассмотрим их использование на примере классификации цветков ириса:

```
library(klaR)
naive_iris <- NaiveBayes(iris$Species ~ ., data = iris)
naive_iris$tables$Petal.Width
```

```
            [,1]      [,2]
setosa     0.246 0.1053856
versicolor 1.326 0.1977527
virginica  2.026 0.2746501
```

Для каждой метрической независимой переменной были выведены средние значения `Petal.Width` (первый столбец) и их стандартные отклонения (второй столбец) для каждого выделенного класса. Можно установить, что ширина лепестка у виргинского ириса существенно выше, чем у остальных двух видов.

Для визуальной сравнительной оценки связи измеренных переменных с метками классов удобно рассмотреть ядерные функции плотности условной вероятности (рис. 7.6):

```
plot(naive_iris, lwd=2)
```

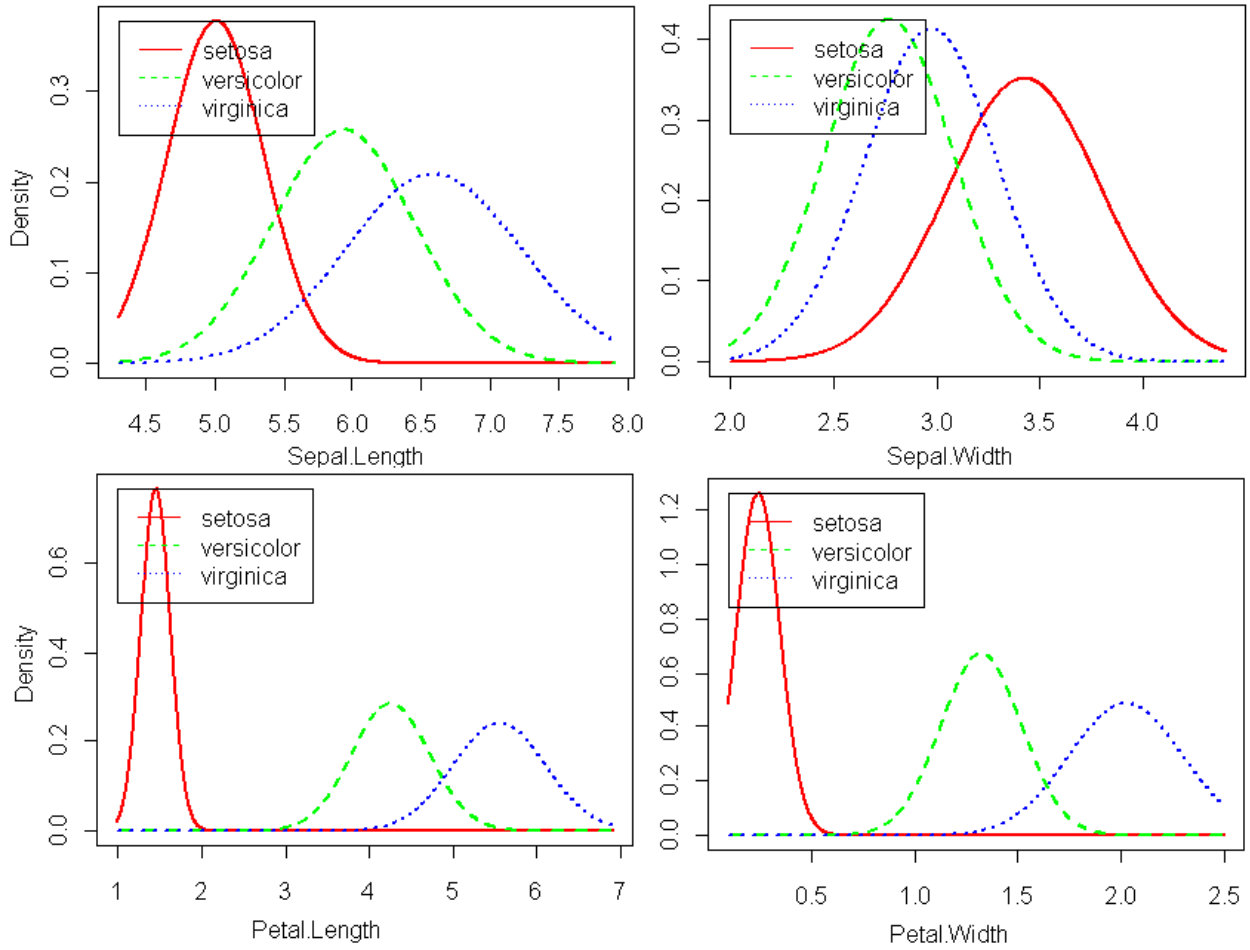


Рис. 7.6. Кривые ядерной плотности условной вероятности для предикторов из набора данных по ирисам

Выполним прогноз вида ирисов для объектов из обучающей выборки:

```
pred <- predict(naive_iris, iris[,-5])$class
table(Факт=iris$Species, Прогноз=pred)
Acc = mean(pred == iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
            Прогноз
Факт         setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         47         3
  virginica       0          3        47
[1] "Точность=96.00%"
```

Осуществим 10-кратную перекрестную проверку построенной модели, чтобы установить качество ее предсказаний на контрольных примерах. Одновременно уточним значения некоторых гиперпараметров процедуры вычисления вероятностей по частотам: надо ли использовать ядерное сглаживание (`usekernel=TRUE`) или можно предположить нормальное распределение, а также следует ли проводить коррекцию вероятностей по

Лапласу fL. Для этого снова воспользуемся функцией train() из пакета caret:

```
library(caret)
#  Определим условия перекрестной проверки:
train_control <- trainControl(method='cv',number=10)
Test <- train(Species~., data = iris, trControl=train_control,
method="nb")
print(Test)
Acc = mean(predict(Test$finalModel,iris[,-5])$class
                          == iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
Resampling: Cross-Validation (10 fold)
  usekernel  Accuracy  Kappa  Accuracy SD  Kappa SD
  FALSE      0.96      0.94   0.0562       0.0843
  TRUE       0.96      0.94   0.0466       0.0699
Tuning parameter 'fL' was held constant at a value of 0
Accuracy was used to select optimal model using largest value.
The final values used for the model were fL = 0 and
                                        usekernel = FALSE.
[1] "Точность=96.00%"
```

Более качественной модели в результате перекрестной проверки построить не удалось, но точность классификатора на обучающей и контрольной выборках осталась стабильно высокой.

7.3. Классификация в линейном дискриминантном пространстве

Исходные предпосылки линейного дискриминантного анализа, изложенные нами в разделе 6.1, естественным образом распространяются на случай произвольного числа классов *k*. Если вновь обратиться к аналогам евклидовой геометрии, то центроиды двух классов определяют положение прямой линии, трех – плоскости, четырех – четырехмерной гиперплоскости и т.д. Принцип сводится к тому, что тем самым определяется "натянутое на центроиды" дискриминантное пространство, размерность которого на единицу меньше, чем число классов.

В полученном пространстве необходимо задать точку начала системы координат: наиболее удобен для этого "главный центроид", занимающий положение, определяемое *m*-мерными средними значениями для всей совокупности объектов. Теперь если мы в рассматриваемом пространстве проведем дополнительную ось под некоторым углом, относительно которой средние значения классов разделяются в большей степени, чем для любого другого направления, то получим главную разделяющую ось. Предполагая, что есть более чем два класса, можно ориентировать вторую и последующие дискриминтные оси таким образом, чтобы также было обеспечено максимальное разделение классов, но при дополнительном ограничении – все построенные оси ортогональны между собой.

Такая процедура напоминает анализ главных компонент (см. раздел 2.4) за тем основным отличием, что дискриминантные оси проводятся в

направлении максимумов не общей, а межгрупповой изменчивости данных. Ее основным результатом также является совокупность линейных комбинаций исходных переменных, которые называются *линейными дискриминантами* (linear discriminant, LD) или *каноническими переменными* (canonical variate), и вычисляются по формуле:

$$LD_k = \sum_{j=1}^{m} \beta_{kj} \cdot (x_j - \bar{x}_j),$$

где k – номер линейного дискриминанта, x_j – значение j-й переменной, $\bar{x}_j$ – среднее значение j-й переменной (по всем группам), β_{kj} – весовой коэффициент j-й переменной в k-ом линейном дискриминанте; суммирование ведется по m переменным. Первый дискриминант подбирается таким образом, чтобы максимизировать основные различия между группами, второй и последующие также максимизируют оставшуюся межгрупповую изменчивость, но подбираются ортогонально первому и всем предыдущим. Отсюда следует, что любая каноническая дискриминантная функция LD_k имеет нулевую корреляцию с $LD_1, \ldots, LD_{g-1}$. Если число групп равно g, то число канонических дискриминантных функций будет на единицу меньше числа групп.

Один из методов поиска оптимальных дискриминантных функций заключается в максимизации отношения межгрупповой вариации к внутригрупповой, т.е. $\mathbf{B} / \mathbf{W} \Rightarrow \max$, где $\mathbf{B}$ – межгрупповая, а $\mathbf{W}$ – внутригрупповая матрица рассеяния наблюдаемых переменных относительно средних. Тогда нахождение канонических коэффициентов дискриминантных функций сводится к нахождению собственных значений λ_k и векторов $\mathbf{u}_k$: если спроецировать g групп m–мерных выборок на $(g - 1)$-мерное пространство, порожденное собственными векторами $\mathbf{u}_k$, то отношение $\lambda = \mathbf{B} / \mathbf{W}$ будет максимальным, т.е. разброс между группами будет наиболее контрастным при заданной внутригрупповой вариации.

Обратимся вновь к распознаванию видов цветков ириса, выборка которых носит имя сэра Р. Фишера, поскольку именно на ней он и продемонстрировал возможности дискриминантного анализа:

```
require(MASS)
LDA.iris <- lda(formula = Species ~ .,data = iris)
LDA.iris$scaling # Коэффициенты линейных дискриминантов
```

```
                    LD1         LD2
Sepal.Length  0.8293776  0.02410215
Sepal.Width   1.5344731  2.16452123
Petal.Length -2.2012117 -0.93192121
Petal.Width  -2.8104603  2.83918785
```

```
LDA.iris$svd
(prop = LDA.iris$svd^2/sum(LDA.iris$svd^2))
```

```
 [1] 48.642644   4.579983
 [1] 0.991212605 0.008787395
```

Последними действиями мы вывели значения λ, которые являются отношением межгрупповой дисперсии к внутригрупповой дисперсии, и вычислили долю межгрупповой дисперсии, объясненную каждым линейным дискриминантом. На рис. 7.7 представлена ординационная диаграмма наблюдений в осях LD1-LD2, подобная графику главных компонент на рис. 7.3. Можно обратить внимание на то, что трансгрессия точек между видами практически исчезла, а главная ось дискриминации объясняет 99% межгруппового разброса:

```
prop <- percent(prop)
pred <-predict(LDA.iris, newdata = iris)
scores <- data.frame(Species = iris$Species,pred$x)
# Выводим ординационную диаграмму
require(ggplot2)
ggplot() +   geom_point(data=scores,
  aes(x=LD1,y=LD2,shape=Species,colour=Species),size=3) +
  scale_colour_manual( values = c('purple', 'green', 'blue'))+
  labs(x = paste("LD1 (", prop[1], ")", sep=""),
       y = paste("LD2 (", prop[2], ")", sep="")) + theme_bw()
```

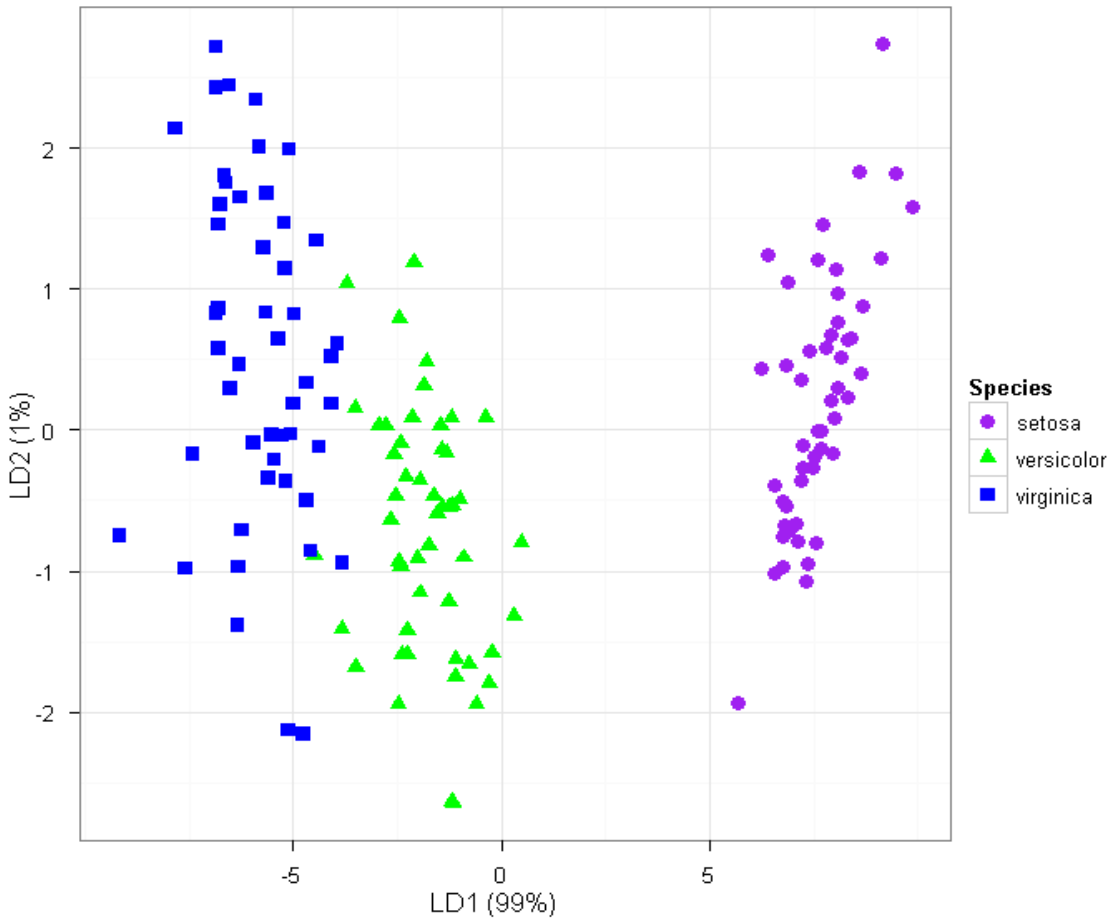


Рис. 7.7. Ординационная диаграмма выборки ирисов в двух осях линейных дискриминантов

Чтобы оценить относительный вклад отдельных переменных в линейные дискриминанты анализ необходимо проводить с использованием стандартизованных данных. Тогда полученные результаты в плане

апостериорной классификации и даже значений линейных дискриминантов для каждого наблюдения не изменятся. Однако, поскольку переменные масштабированы, весовые коэффициенты переменных теперь можно сравнивать между собой: очевидно, что межвидовые различия определяются в первую очередь размерами лепестка:

```
lda(scale(iris[,1:4]), gr = iris$Species)$scaling
```

```
                    LD1         LD2
Sepal.Length  0.6867795  0.01995817
Sepal.Width   0.6688251  0.94344183
Petal.Length -3.8857950 -1.64511887
Petal.Width  -2.1422387  2.16413593
```

В результате применения линейного дискриминантного анализа к примерам из обучающей выборки можно сформировать решающее правило, позволяющее отнести новый объект к классу, имеющему наибольшую плотность вероятности ("уровень правдоподобия") в точке **x** m-мерного пространства признаков. Сама процедура классификации основана на вычислении апостериорных вероятностей $\Pr(G_k \mid \mathbf{x})$ принадлежности произвольного вектора наблюдений **x** к группе G_k, $k = 1, 2,\ldots, g$, с учетом анализа функций плотности распределения $f_k(\mathbf{x})$.

Величина $\Pr(G_k \mid \mathbf{x})$ связана с дистанцией между точкой **x** и k-м центроидом, и в случае выполнения предпосылок дискриминантного анализа может быть оценена с использованием значений *классифицирующих функций*:

$$d_k(\mathbf{x}) = \boldsymbol{\mu}_k^{\mathrm{T}} \mathbf{C}^{-1}\mathbf{x} \; - 0.5\, \boldsymbol{\mu}_k^{\mathrm{T}} \mathbf{C}^{-1}\boldsymbol{\mu}_k + \ln\pi_k,$$

где $\mathbf{C}^{-1}$ – матрица, обратная к ковариационной, $\boldsymbol{\mu}_k$ и $\ln\pi_k$ – вектор средних и априорная вероятность наблюдения объектов k-го класса соответственно. Объект **x** относится к тому классу k из g, для которого значение d_k, а следовательно и апостериорная вероятность, максимальны.

Если выделить из обучающей выборки 10 объектов для проверки качества предсказаний, то получим следующие (безошибочные) результаты:

```
train <- sample(1:150, 140)
LDA.iris3 <- lda(Species ~ ., iris, subset = train)
plda = predict(LDA.iris3, newdata = iris[-train, ])
data.frame(Species=iris[-train,5], plda$class, plda$posterior)
```

```
       Species plda.class       setosa   versicolor    virginica
32      setosa     setosa 1.000000e+00 2.075304e-20 1.232723e-39
38      setosa     setosa 1.000000e+00 1.274233e-24 1.270493e-45
45      setosa     setosa 1.000000e+00 1.185866e-18 1.103632e-36
56  versicolor versicolor 7.594352e-24 9.982990e-01 1.700985e-03
81  versicolor versicolor 1.751434e-18 9.999970e-01 2.952626e-06
94  versicolor versicolor 1.020832e-14 9.999999e-01 9.062344e-08
97  versicolor versicolor 4.091110e-20 9.998948e-01 1.052484e-04
116  virginica  virginica 1.386213e-41 3.491571e-05 9.999651e-01
119  virginica  virginica 5.658468e-63 5.997144e-10 1.000000e+00
146  virginica  virginica 8.347791e-41 9.898162e-05 9.999010e-01
```

Подчеркнуты значения максимальных апостериорных вероятностей, на основании которых предсказывался вид цветка `plda.class`.

При анализе всей обучающей выборки цветков ириса мы имеем три ошибочных результата классификации из 150.

```
Acc = mean(pred$class == iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
[1] "Точность=98.00%"
```

Тот же итог мы получим и при проведении скользящего контроля:

```
LDA.irisCV <- lda(Species ~ ., data = iris,  CV = TRUE)
table(Факт=iris$Species,Прогноз=LDA.irisCV$class)
Acc = mean(LDA.irisCV$class==iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
            Прогноз
Факт         setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         48         2
  virginica       0          1        49
[1] "Точность=98.00%"
```

7.4. Нелинейные классификаторы в R

Квадратичный дискриминантный анализ (QDA) является нелинейным обобщением метода LDA, который не использует предположения об однородности ковариационной матрицы. В качестве решающего правила применяется квадратичная функция

$$d_k(\mathbf{x}) = -0.5\,(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{C}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) - 0.5\ln|\mathbf{C}_k| + \ln\pi_k,$$

где $|\mathbf{C}_k|$ – детерминант ковариационной матрицы k-го класса, $\mathbf{C}_k^{-1}$ – ее обратная матрица; π_k – априорная вероятность наблюдения объектов k-го класса. Экзаменуемый объект, как и в линейном случае, относится к классу с максимальным значением d_k.

Квадратичный дискриминантный анализ весьма эффективен, когда разделяющая поверхность между классами имеет ярко выраженный нелинейный характер (например, параболоид или эллипсоид в 3D-случае). Однако он сохраняет большинство недостатков LDA: использует предположение о нормальности распределения и не работает, когда матрицы ковариаций вырождены (например, при большом числе переменных). Другим недостатком QDA является то, что уравнение разделяющей гиперповерхности выражено в неявном виде и не может быть использовано для "объяснения" результатов.

Для проведения квадратичного дискриминантного анализа можно воспользоваться функцией `qda()` из пакета `MASS`:

```
require(MASS)
QDA.iris = qda(Species~ Petal.Length + Petal.Width,
                        data = iris)
pred = predict(QDA.iris)$class
table(Факт=iris$Species,Прогноз=pred)
Acc = mean(pred ==iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
          Прогноз
Факт       setosa versicolor virginica
  setosa       50          0         0
  versicolor    0         49         1
  virginica     0          2        48
[1] "Точность=98.00%"
```

Чтобы показать на рис. 7.8 форму разделяющих поверхностей для классов цветков ириса используем только две наиболее значимые переменные, определяющие размеры лепестка:

```
library(klaR)
partimat(Species ~ Petal.Length + Petal.Width,
         data = iris, method="qda")
```

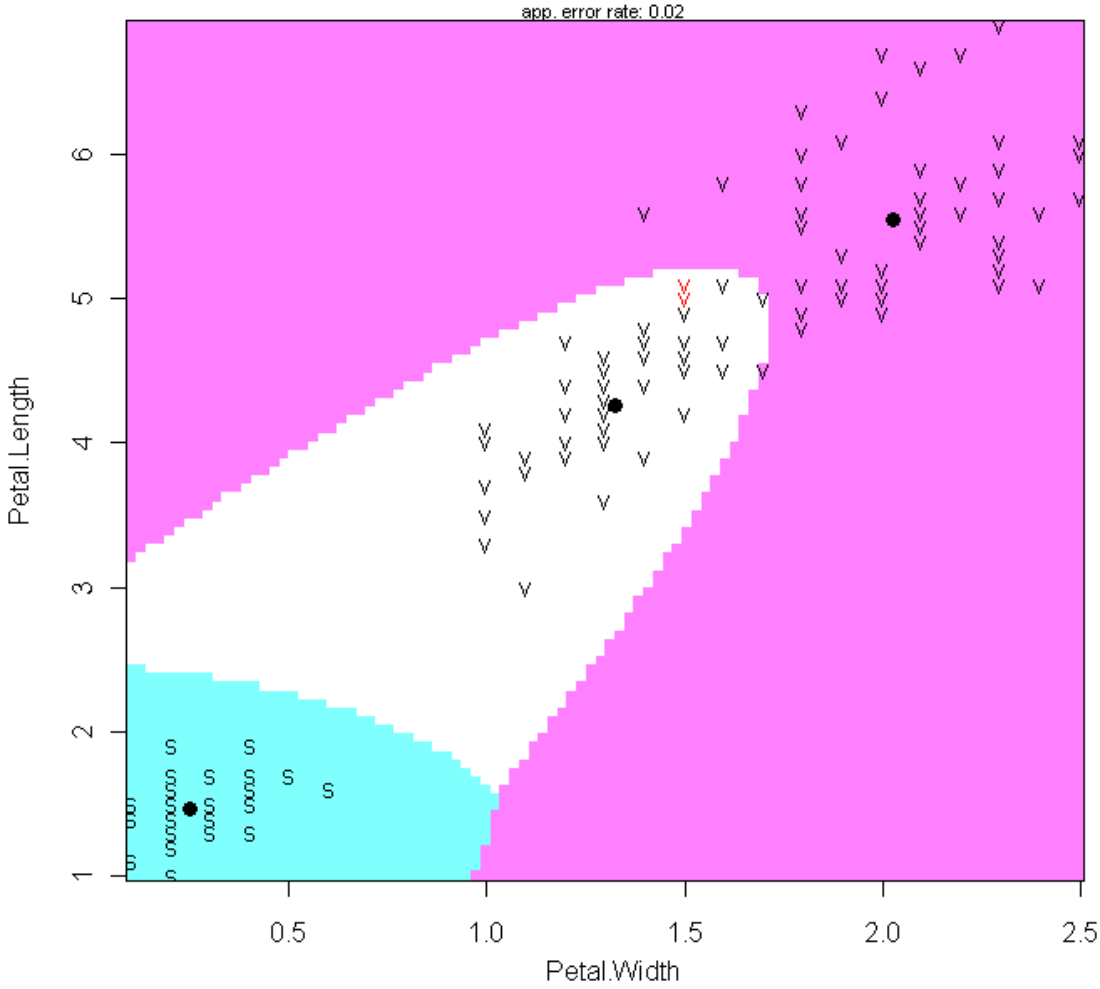


Рис. 7.8. Разделяющие границы, полученные в результате применения квадратичного дискриминантого анализа для классификации цветков ириса (ошибочное распознавание выделено шрифтом красного цвета); жирными точками показаны центроиды классов

С использованием функции `train()` из пакета `caret` сравним точность классификации по полной модели и модели с двумя предикторами на основании 10-кратной перекрестной проверки:

```
library(caret)
set.seed(123)
# Используем только размеры лепестка
train(Species ~ Petal.Length + Petal.Width, data = iris,
method = "qda", trControl = trainControl(method = "cv"))
```

```
Accuracy  Kappa  Accuracy SD  Kappa SD
0.973     0.96   0.0344       0.0516
```

```
# Используем весь набор признаков
train(Species ~ ., data = iris, method = "qda",
trControl = trainControl(method = "cv"))
```

```
Accuracy  Kappa  Accuracy SD  Kappa SD
0.973     0.96   0.0344       0.0516
```

Очевидно, что ботаникам вполне можно было обойтись без утомительной работы по фиксации размеров чашелистника.

Регуляризованный дискриминантный анализ (RDA – Regularized Discriminant Analysis), предложенный Дж. Фридманом, является своеобразным промежуточным звеном между LDA и QDA. Пусть λ – положительный параметр, подобранный таким образом, чтобы управлять степенью кривизны разделяющей поверхности. При его использовании регуляризованная ковариационная матрица $\mathbf{C}_k(\lambda)$ оказывается как бы более "притянутой" к общей ковариационной матрице $\mathbf{C}_0$, чем индивидуальная $\mathbf{C}_k$ для k-го класса:

$$\mathbf{C}_k(\lambda) = (1 - \lambda)\,\mathbf{C}_k + \lambda\,\mathbf{C}_0.$$

Новая ковариационная матрица всегда обратима и может быть использована в вычислениях. При этом гарантируется, что полученная разделяющая поверхность имеет большую выпуклость, чем при LDA, но меньшую, чем при QDA, к которой она сходится при $\lambda = 0$.

Более точная регуляризация получается при искусственном увеличении числа доступных объектов обучения за счет добавления к данным шума, интенсивность которого связана со вторым параметром – γ:

$$\mathbf{C}_k(\lambda, \gamma) = (1 - \gamma)\,\mathbf{C}_k(\lambda) + \gamma\,\mathrm{trace}[\mathbf{C}_k(\lambda)]\,\mathbf{I} / m,$$

где $\mathrm{trace}[\mathbf{C}_k(\lambda)]$ – диагональные элементы матрицы $\mathbf{C}_k(\lambda)$, или вектор дисперсий исходных m переменных, описывающих объекты k-го класса; $\mathbf{I}$ – единичная матрица.

При экстремальных значениях λ и γ ковариационная структура сводится к четырем особым случаям:

- (γ=0, λ=0): QDA – индивидуальная ковариация в каждой группе;
- (γ=0, λ=1): LDA – общая матрица ковариации;
- (γ=1, λ=0): исходные переменные условно независимы (подобно тому, что предполагает наивный байесовский классификатор), но их внутригрупповые дисперсии однородны;
- (γ=1, λ=1): дисперсии равны для всех групп и классификация объектов осуществляется по евклидовым расстояниям до центроидов.

Подбор оптимальных значений (λ, γ) можно осуществить с помощью функции `train()` из пакета `caret`, для чего можно задать, например, 10-кратную перекрестную проверку:

```
library(klaR); set.seed(123)
#  1 - этап грубой оптимизации
train(Species ~ ., data = iris, method = "rda",
trControl = trainControl(method = "cv"))
```

```
  gamma  lambda  Accuracy  Kappa  Accuracy SD  Kappa SD
  0      0       0.973     0.96   0.0344       0.0516
  0      0.5     0.98      0.97   0.0322       0.0483
  0      1       0.98      0.97   0.0322       0.0483
  0.5    0       0.967     0.95   0.0471       0.0707
  0.5    0.5     0.967     0.95   0.0471       0.0707
  0.5    1       0.96      0.94   0.0466       0.0699
```

```
#  2 - этап с диапазоном lambda = 0.1:0.5 и gamma = 0.02:0.1
RDAGrid <- expand.grid(.lambda = (1:5)/10, .gamma = (1:5)/50)
RDA.iris <- train(Species ~ ., data = iris, method = "rda",
tuneGrid =RDAGrid, trControl = trainControl(method = "cv"))
```

```
  gamma  lambda  Accuracy  Kappa  Accuracy SD  Kappa SD
  0.02   0.3     0.98      0.97   0.0322       0.0483
  0.02   0.4     0.98      0.97   0.0322       0.0483
  0.02   0.5     0.98      0.97   0.0322       0.0483
  0.04   0.1     0.98      0.97   0.0322       0.0483
  0.04   0.2     0.98      0.97   0.0322       0.0483
Accuracy was used to select optimal model using the largest value.
The final values used for model were gamma = 0.02 and lambda = 0.5.
```

```
#  Оценка точности итоговой модели:
pred = predict(RDA.iris)
Acc = mean(pred ==iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
 [1] "Точность=98.00%"
```

Обратим внимание, что при выполнении скрипта нужно отключить загруженный пакет `vegan`, где также есть функция `rda()`.

Машина опорных векторов SVM, описанная в разделах 6.2-6.3, является бинарным классификатором и естественным образом на случай с несколькими классами не переносится. Однако имеется возможность использовать этот метод, применяя стратегии "каждый против каждого" (one against one) или "один против всех" (one against all). Применим метод опорных векторов к данным по ирисам для нахождения разделяющих поверхностей, которые по умолчанию в функции `svm()` формируются с использованием радиального ядра RBF:

```
library(e1071)
SVM.iris <- svm(Species~., data=iris)
pred.DV <- predict(SVM.iris, iris[,1:4],
                           decision.values = TRUE)
ind <- sort(sample(150,9))
data.frame(attr(pred.DV, "decision.values")[ind,],
                 iris$Species[ind])
```

```
    setosa/versicolor setosa/virginica versicolor/virginica
Species
19  1.0094978         1.0167159        0.6879294             setosa
24  1.0000423         1.0003943        1.1089452             setosa
33  1.0554817         1.0503686        0.2549067             setosa
39  1.0641799         1.0304548        0.5549086             setosa
65  -1.0604694        -0.4843803       1.7638663         versicolor
68  -1.1579383        -0.5659738       1.9179151         versicolor
85  -1.1107460        -0.7633837       0.8171571         versicolor
145 -0.6676827        -1.0986948       -1.6444716         virginica
150 -1.1809673        -1.0602949       -0.4345316         virginica
```

Были получены пороговые значения (decision values) при распознавании "класс1"/"класс2": если пороговое значение положительно, то побеждает "класс1", иначе – "класс2". Окончательный результат определяется

```
pred <- predict(SVM.iris, iris[,1:4], type="response")
table(Факт=iris$Species, Прогноз=pred)
Acc = mean(pred ==iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

голосованием, т.е. в данном случае достаточно победить обоих оппонентов.

```
            Прогноз
Факт         setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         48         2
  virginica       0          2        48
[1] "Точность=98.00%"
```

Вид ядерных функций многоклассовой SVM не может быть непосредственно визуализирован, поэтому для обозначения решающих границ применим косвенный метод метрического многомерного шкалирования MDS (multidimensional scaling).

```
plot(cmdscale(dist(iris[,-5])), col = as.integer(iris[,5])+1,
     pch = c("o","+")[1:150 %in% SVM.iris$index + 1], font=2,
       xlab="Шкала 1",ylab="Шкала 2" )
legend (0,1.2, c("setosa","versicolor","virginica"),pch = "o",
       col =2:4)
```

Здесь функция `dist()` строит матрицу евклидовых расстояний размерностью 150×150 для всех пар наблюдений по всей совокупности измеренных показателей. Функция `cmdscale()` проецирует эту матрицу на плоскость таким образом, чтобы минимизировать искажения взаимных дистанций между объектами при сокращении исходного многомерного пространства до двумерной ординации (рис. 7.9). Опорные векторы, представленные на диаграмме крестиками, задают границы разделяющих RBF-поверхностей, аккуратно окаймляющих области каждого класса.

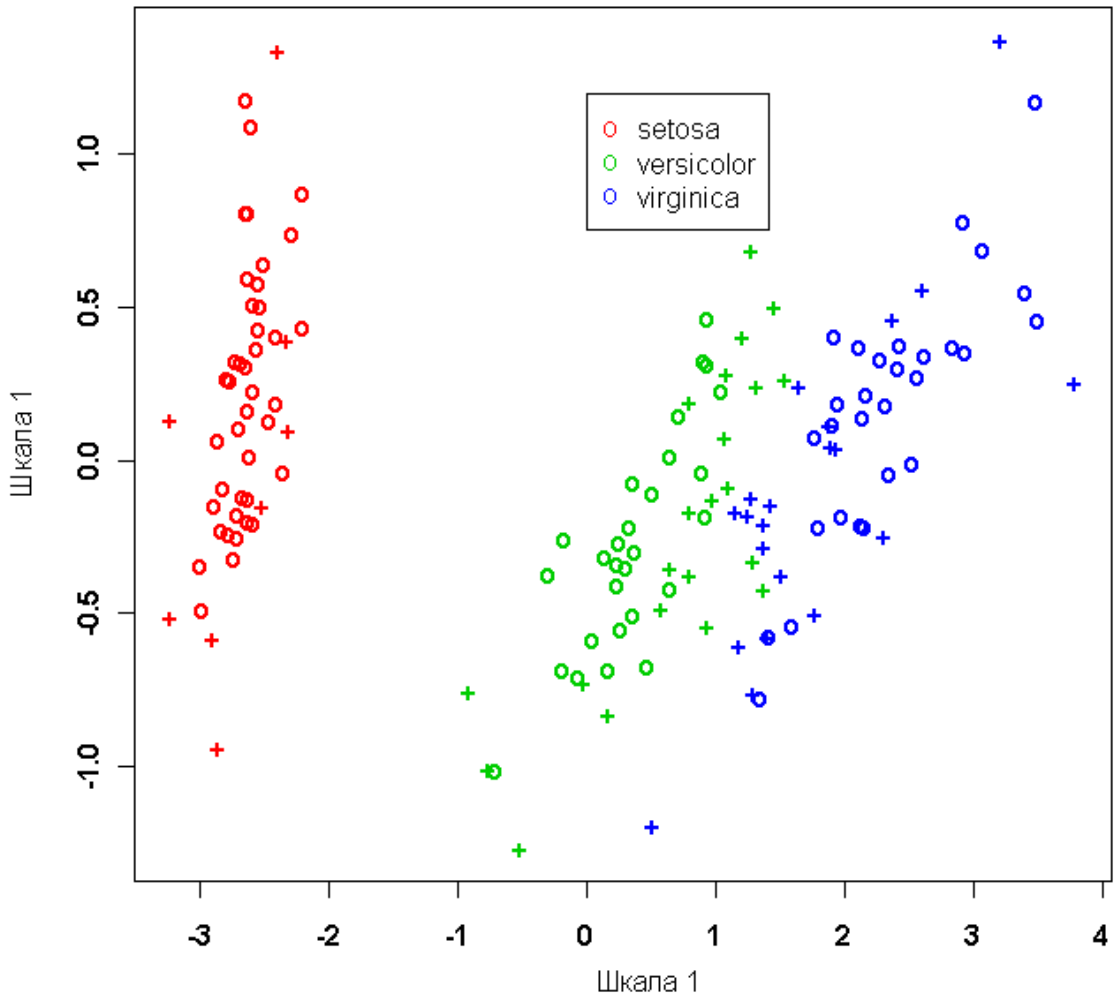


Рис. 7.9. Ординационная диаграмма наблюдений из набора данных по ирисам, построенная методом многомерного шкалирования (крестиками соответствующего цвета показаны опорные векторы)

7.5. Модель мультиномиального логита

Логистическая регрессия обычно используется как бинарный классификатор для выборок с альтернативным откликом. Однако этот метод можно обобщить и на случай с несколькими классами. В качестве моделируемого отклика Y могут использоваться номинальные или порядковые переменные, и в обоих случаях мы предполагаем многомерное биномиальное распределение.

Рассмотрим сначала модель логита (multinomial logit) для отклика с несколькими классами. Пусть J обозначает число классов для случайной независимой величины Y (требование независимости всех J альтернатив тут играет важное значение). Многомерное распределение вероятности наблюдения каждого из ее классов может быть представлено вектором $\{\pi_1, ..., \pi_J\}$, $\sum_j \pi_j = 1$. Тогда для каждого из n независимых наблюдений оценка π_j будет соответствовать вероятности того, насколько предпочтительно присвоение произвольному тестируемому объекту метки класса j.

Отметим, что для оценки всех вероятностей достаточно построить J - 1 моделей, поскольку недостающее значение легко получить из условий нормировки (сумма π_j всегда равна 1). Поэтому один из классов можно принять за базовый (примем для определенности, что это класс с номером 1), и тогда совокупность логит-моделей (baseline-category logit) будет иметь вид:

$$\log(\pi_j / \pi_1) = \alpha_j + \beta_j \mathbf{x}, \quad j = 2,..., J.$$

Коэффициенты системы логит-уравнений оцениваются совместно путем максимизации общего критерия правдоподобия, т.е. другая форма анализа, основанная, например, на трех последовательно построенных моделях регрессии локально для каждого класса, может дать совсем иные результаты (Agresti, 2007).

Для анализа категориальных данных в R существуют мощные специализированные пакеты `VGAM` и `mlogit`. Мы же воспользуемся более простой в освоении функцией `multinom()` из пакета `nnet`, которая осуществляет оценку коэффициентов системы логит-моделей с использованием алгоритмов построения искусственных нейронных сетей (см. раздел 7.6). Для данных по ирисам Фишера получаем:

```
library(nnet)
MN.iris <- multinom(Species~., data=iris)
summary(MN.iris)
```

```
Coefficients:
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
versicolor    18.69037    -5.458424   -8.707401     14.24477   -3.097684
virginica    -23.83628    -7.923634  -15.370769     23.65978   15.135301
Std. Errors:
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
versicolor    34.97116     89.89215    157.0415     60.19170    45.48852
virginica     35.76649     89.91153    157.1196     60.46753    45.93406
Residual Deviance: 11.89973
AIC: 31.8997
```

Анализируя коэффициенты, можно заметить, например, что ширина чашелистика `Sepal.Width` уменьшается при переходе от `setosa` к `virginica`, причем логарифм отношения шансов $\log(\pi_{vir} / \pi_{set})$ уменьшается на величину $\beta_{23} = -15.3$ при увеличении ширины на 1 см.

Метод `fitted()`, применяемый к объекту `multinom`, возвращает матрицу апостериорных вероятностей принадлежности каждого наблюдения к соответствующим классам. Предсказание класса осуществляется с учетом максимума такой вероятности:

```
Probs <- fitted(MN.iris)
```

```
          setosa   versicolor    virginica
1   1.000000e+00 1.526406e-09 2.716417e-36
2   9.999996e-01 3.536476e-07 2.883729e-32
3   1.000000e+00 4.443506e-08 6.103424e-34
```

```
pred=apply(Probs,1,function(x)
        colnames(Probs)[which(x==max(x))])
table(Факт=iris$Species,Прогноз=pred)
Acc = mean(pred ==iris$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
            Прогноз
Факт         setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         1
  virginica       0          1        49
[1] "Точность=98.67%"
```

Поскольку функции, выполняющие подгонку моделей мультиноминального логита, обычно не проводят анализа статистической значимости коэффициентов, рассчитаем *p*-значения самостоятельно с использованием теста Вальда (https://ru.wikipedia.org) и *t*-статистики:

```
z <- summary(MN.iris)$coefficients/
       summary(MN.iris)$standard.errors
# p-значения на основе теста Вальда
(1 - pnorm(abs(z), 0, 1))*2
```

```
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
versicolor   0.5930295    0.9515807   0.9557828    0.8129231   0.9457075
virginica    0.5051288    0.9297757   0.9220685    0.6955898   0.7417773
```

```
# p-значения на основе  t-статистики
pt(z, df = nrow(iris) - 5, lower=FALSE)
```

```
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
versicolor   0.2969240    0.5241679   0.5220705    0.4066286   0.5270993
virginica    0.7469061    0.5350513   0.5388982    0.3480821   0.3711264
```

И мы с недоумением видим, что, несмотря на приличную точность предсказаний модели, все отношения $\log(\pi_k / \pi_1)$ согласно вычисленным *p*-значениям статистически не отличаются от 0, т.е. все шансы равны. Ответ видится в резковатой реплике проф. Б. Рипли (B. Ripley) при дискуссии в Интернете: «Попробуйте почитать наши книги. Там объясняется, почему тесты Вальда выполнять нельзя, и что асимптотическая теория может здесь дико вводить в заблуждение». Читатели, желающие убедиться в правоте сказанного могут пересчитать *p*-значения коэффициентов с использованием теста множителей Лагранжа или бутстрепа, в частности, функцией `cluster.im.mlogit()` из пакета `clusterSEs`.

7.6. Классификаторы на основе искусственных нейронных сетей

Нейросетевые модели, родившиеся в процессе развития концепции искусственного интеллекта, имеют две вполне прозрачные аналогии – биологическая нейронная система мозга и компьютерная сеть. Их основная парадигма состоит в том, что решение в сети формируется множеством простых нейроно-подобных элементов, образующих граф с взвешенными синаптическими (информационными) связями, которые совместно и целенаправленно работают на получение общего результата. Таким образом, искусственная нейронная сеть (ИНС) является (Дук, Самойленко, 2001):

- *параллельной* системой, поскольку в любой момент времени в активном состоянии могут находиться несколько процессов;
- *распределенной* системой, поскольку каждый из процессов может независимо обрабатывать локальные данные;

- *адаптивной* системой, наделенной свойствами самообучения и подстройки своих параметров при изменении профиля данных.

Главным строительным блоком ИНС является формальный нейрон, структура которого имеет вид, представленный на рис. 7.10:

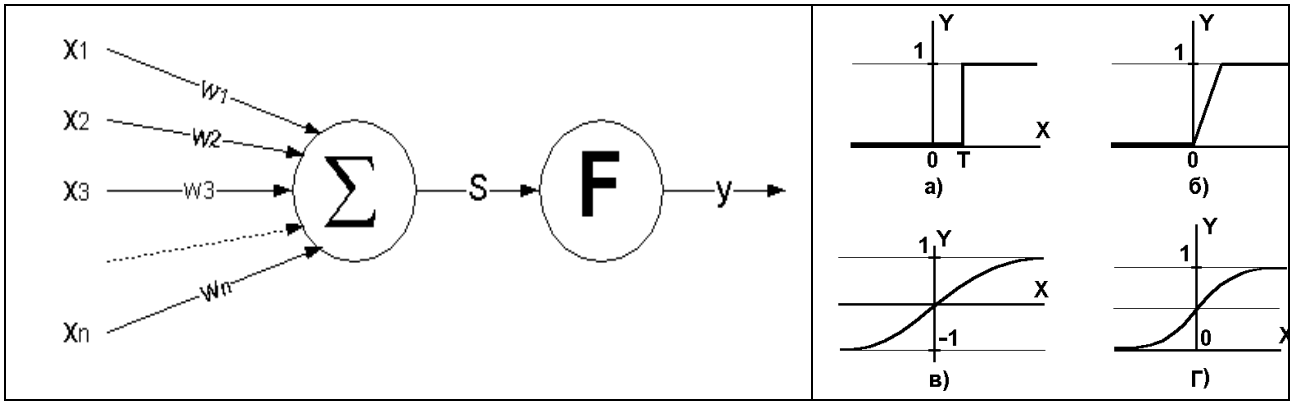


Рис. 7.10. Структура искусственного нейрона (слева) и вид некоторых функций активации **F** (справа); обозначения по тексту

Основная функция искусственного нейрона – сформировать выходной сигнал y в зависимости от сигналов $x_1 ... x_n$, поступающих на его входы. Эти значения могут усиливаться или "тормозиться" в зависимости от знака *весов* синапсов $w_1 ... w_n$. Входные сигналы обрабатываются *адаптивным сумматором* $S = \sum_{i=1}^{n} w_i x_i - T$, где T – *порог* нейрона, а затем выходной сигнал сумматора поступает в *нелинейный преобразователь* **F** с некоторой *функцией активации*, после чего результат подается на выход (в точку ветвления).

Вид функции активации может иметь различное математическое выражение, выбор которого определяется характером решаемых задач. Например, преобразование может осуществляться функцией, определенной как единичный скачок (*а* на рис. 7.10 справа), линейный порог (*гистерезис*, *б*), гиперболический тангенс (*в*) или *сигмоид* (*г*). Наиболее распространена нелинейная функция с насыщением – так называемая логистическая функция, или сигмоид, $y = \frac{1}{1 + e^{-cS}}$, которая имеет много ценных свойств (монотонность и дифференцируемость, устойчивость к выбросам, простое выражение для ее производной и т.д.). Выходное значение сигмоидального нейрона лежит в диапазоне [0, 1]. При уменьшении коэффициента c сигмоид становится более пологим, вырождаясь в пределе при $c = 0$ в горизонтальную линию на уровне 0.5. При увеличении c сигмоид приближается по внешнему виду к функции единичного скачка с порогом T в точке $x = 0$.

По способу соединения нейронов выделяют сети с разной архитектурой: персептроны, сети адаптивного резонанса, рециркуляционные, рекуррентные, встречного распространения, ИНС с обратными связями Хэмминга и Хопфилда, с двунаправленной ассоциативной памятью, с радиально-базисной функцией активации, самоорганизующиеся ИНС

Кохонена и т.п. Мы будем рассматривать возможности обучения ИНС на примере многослойного персептрона с прямым распространением сигнала (и обратным распространением ошибки), структура которого представлена на рис 7.11:

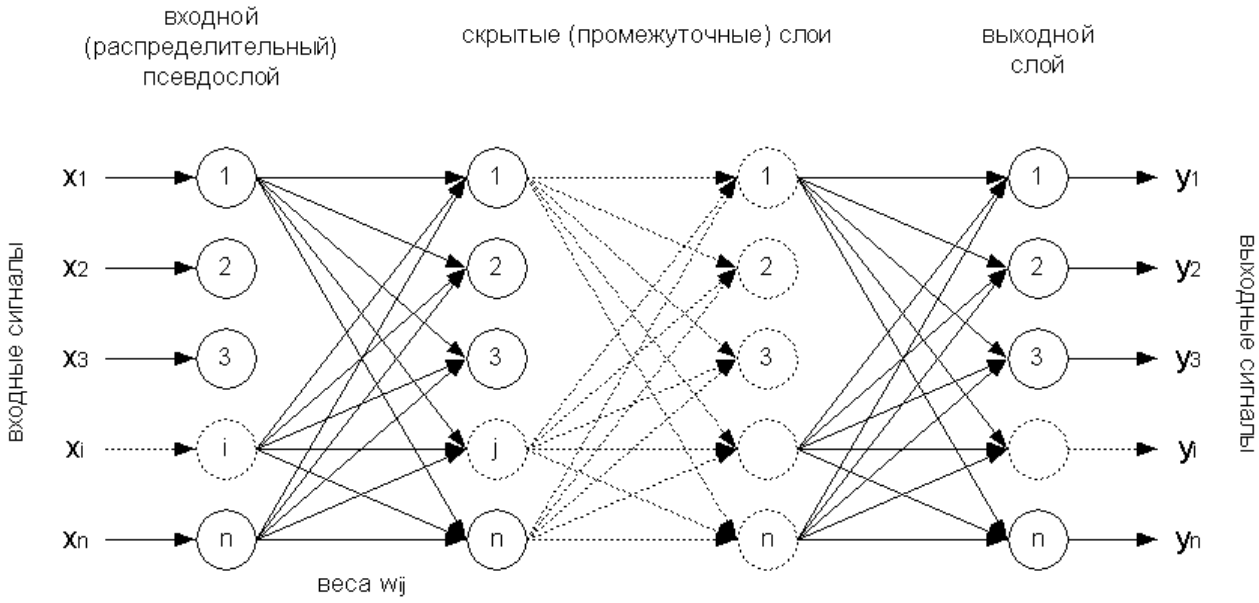


Рис. 7.11. Структура многослойного персептрона

Нейроны персептрона регулярным образом организованы в слои, причем информация направленно распространяется от предыдущих слоев к последующим. Входной слой состоит из n нейронов, на которые подаются значения исходных переменных X_i, никакой обработки информации не совершает и выполняет лишь распределительные функции. Если сеть настроена на классификацию, то взвешенные комбинации y выходного слоя представляют собой прогноз, указывающий на принадлежность распознаваемого объекта к определенной группе. Если распознаются только два класса, то в выходном слое персептрона находится только один элемент, который обладает двумя реакциями – положительной и отрицательной. Если классов больше двух, то для каждой группы устанавливается свой нейрон, и тогда выход y_k каждого такого элемента представляет нелинейную комбинацию его входов, аккумулировавших результаты работы всех предыдущих слоев.

Двухслойный персептрон Розенблата, в принципе, аналогичен нелинейной регрессии или логиту, но добавление произвольного числа промежуточных (скрытых) слоев позволяет существенно усложнить структуру модели и реализовать различные множественные связи нейронов друг с другом с помощью последовательного взятия их линейных комбинаций и применения нелинейных функций активации f (Ripley, 1995):

$$y_k = f_0\left(\sum_{i\to k} w_{ik}x_i + \sum_{j\to k} w_{jk} f_h\left(\sum_{i\to j} w_{ij}x_i\right)\right).$$

В многослойной сети такие функции отклика дают возможность практически точно аппроксимировать любые выпуклые многомерные функциональные зависимости.

Настройка сети представляет достаточно сложную задачу: необходимо подобрать количество промежуточных слоев и число нейронов в каждом из них, установить типы активационной функции и оценить коэффициенты w_i, c_i каждого нейрона, которые минимизировали бы ошибку прогноза, выдаваемого сетью. Нахождение глобального оптимума аналитическими методами тут невозможно, но разработаны вполне эффективные методы исследования поверхностей ошибок и движения по градиенту (Горбань и др., 1998). Выбор нейросети "правильной" сложности, как и в рассмотренных ранее случаях, сводится к двум рецептам: использование контрольных выборок и экспериментирование.

Для обучения ИНС в среде R используются два пакета – `neuralnet` и `nnet`, – обеспечивающие гибкие функциональные возможности построения моделей классификации и регрессии на основе многослойного персептрона. Вернемся снова к данным по ирисам Фишера и разделим ее на обучающую и тестовую выборки в соотношении 7:3, а также добавим в исходную таблицу три столбца, содержащих значения `TRUE/FALSE` для каждого вида растений:

```
data(iris)
ind = sample(2, nrow(iris), replace = TRUE, prob=c(0.7, 0.3))
trainset = iris[ind == 1,]
testset = iris[ind == 2,]
trainset$setosa = trainset$Species == "setosa"
trainset$virginica = trainset$Species == "virginica"
trainset$versicolor = trainset$Species == "versicolor"
```

Построим сеть с тремя слоями, содержащими три нейрона в скрытом слое:

```
library(neuralnet)
net.iris = neuralnet(versicolor + virginica + setosa ~
    Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
      trainset, hidden=3)
net.iris$result.matrix
```

```
error                          2.753538364739
reached.threshold              0.009945475689
steps                      59246.000000000000
```

Из сводки итоговых результатов видно, что процесс обучения нуждался в 59246 шагах, пока максимумы всех частных производных функции ошибок `reached.threshold` по абсолютной величине не оказались ниже чем 0.01 (порог по умолчанию). Ошибка `error` оценивается как уровень правдоподобия по критерию AIC. Остальные величины, представленные в `net.iris$result.matrix` и `net.iris$generalized.weights`, содержат весовые коэффициенты w и значения сигмоид c для каждого нейрона, которые можно должным образом проанализировать и интерпретировать. Это легко сделать, получив изображение ИНС (рис. 7.12):

```
plot(net.iris)
```

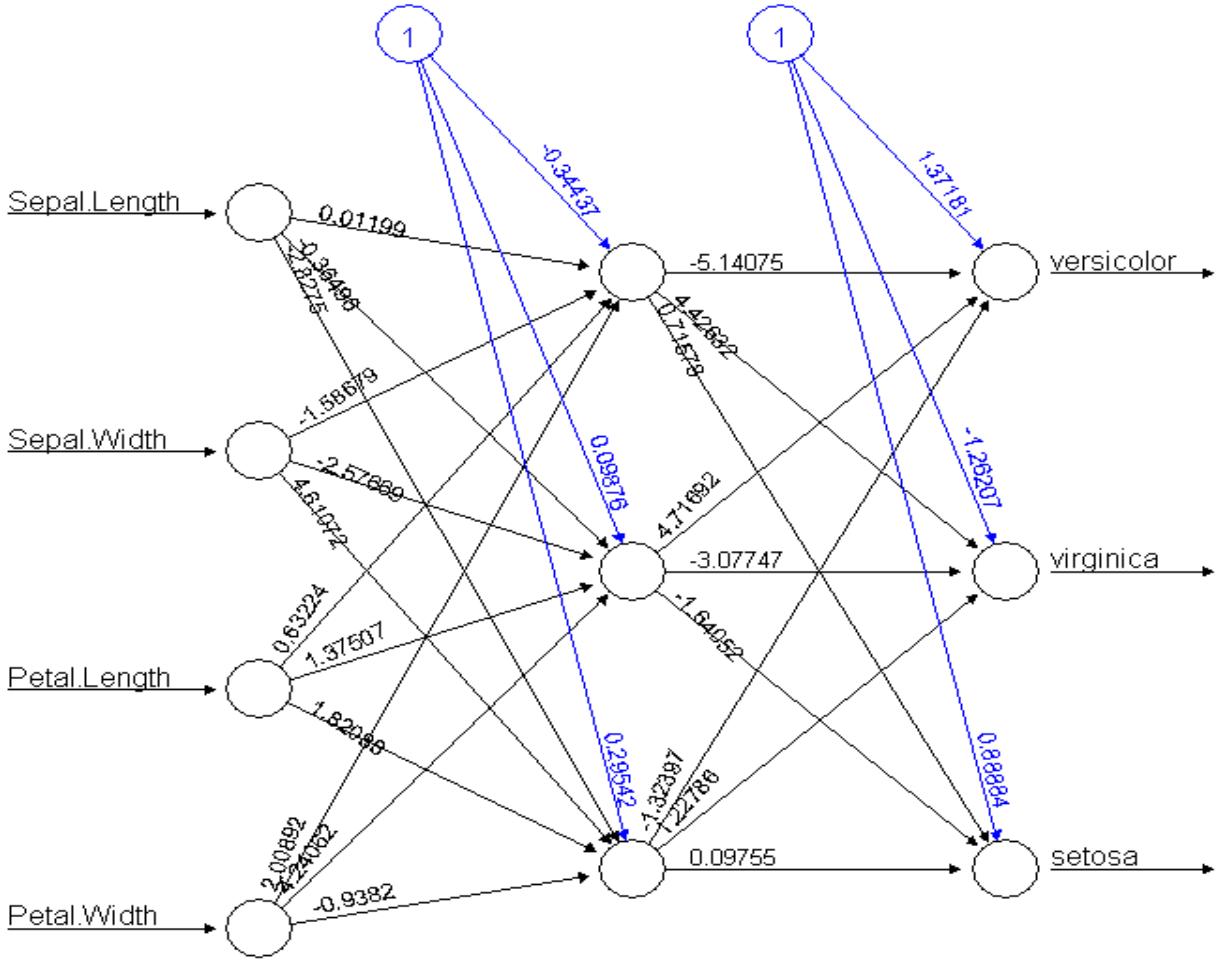


Рис. 7.12. График обученной нейронной сети

На рис. 7.12 видно, что на выходной слой поступает 12 сигналов, которые представляют собой скомбинированные и трансформированные значения четырех исходных переменных. С их использованием и на основе обобщенных весов и функций активации последнего слоя оцениваются вероятности отнесения растения к каждому из трех видов. Для проверочной выборки процесс распознавания выглядит так:

```
net.prob = compute(net.iris, testset[-5])$net.result
```

```
                [,1]            [,2]            [,3]
3   -0.003883942574  0.009369070501  0.994531521671 52
7   -0.017620571917  0.022468391113  0.995187439839 79
11   0.019655437423 -0.010584995810  0.990950139538 26
```

```
pred = c("versicolor",   "virginica",   "setosa")
       [apply(net.prob,   1,   which.max)]
table(Факт=testset$Species, Прогноз= pred)
Acc = mean(pred == testset$Species)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
            Прогноз
Факт         setosa versicolor virginica
  setosa         13          0         0
  versicolor      0         14         0
  virginica       0          0        20
[1] "Точность=100.00%"
```

Дальнейшее рассмотрение особенностей нейросетевых моделей мы продолжим в следующей главе на более представительном примере.

8. МОДЕЛИРОВАНИЕ ПОРЯДКОВЫХ И СЧЕТНЫХ ПЕРЕМЕННЫХ

8.1. Модель логита для порядковой переменной

Если категории отклика являются упорядоченными, то можно воспользоваться этой дополнительной информацией и построить потенциально более качественную модель логистической регрессии с более простой интерпретацией результатов, чем модель, основанную на использовании категориальных классов отклика. Пусть для произвольной порядковой случайной величины Y, изменяющейся на интервале от 1 до J, справедливо неравенство

$$P(Y \le 1) \le P(Y \le 2) \le \ldots \le P(Y \le J) = 1 ,$$

определяющее процесс накопления вероятности

$$P(Y \le 1) = \pi_1 + \ldots + \pi_j , \quad j = 1, \ldots, J.$$

Тогда модель для оценки логарифма отношения накопленных шансов, или кумулятивного логита (cumulative logit), будет иметь вид:

$$\text{logit}[P(Y \le j)] = \log\left[\frac{\pi_1 + \ldots + \pi_j}{\pi_{j+1} + \ldots + \pi_J}\right] = \alpha_j + \beta\mathbf{x} , \quad j = 1, \ldots, J - 1,$$

где параметр α_j определяет величину, на которую увеличивается логарифм отношения шансов при включении вероятности π_j, а коэффициенты линейной модели β не зависят от номера группы j и отражают эффекты воздействия независимых переменных. На рис. 8.1 показан пример такой модели с одинаковым эффектом независимой переменной x на каждую из трех функций накопленных вероятностей отклика с четырьмя категориями: видно, что происходит пропорциональный сдвиг кривых вправо, определяемый величиной коэффициента α_j.

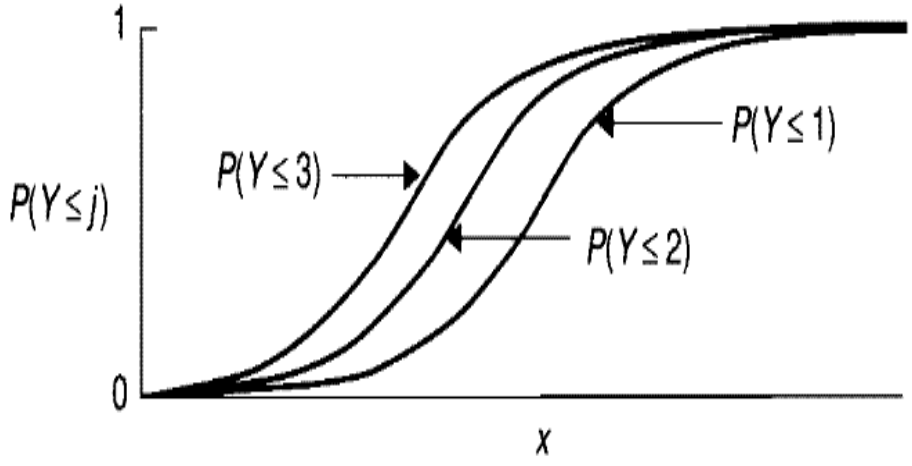


Рис. 8.1. Кривые накопленных вероятностей для модели пропорциональных шансов

Мы распрощаемся с ирисами Фишера, классы которого независимы, и рассмотрим пример модели кумулятивного логита на основе данных о популяции вкусных брюхоногих моллюсков – морских ушек (*Haliotis species*).

Выборка из 8 морфометрических показателей 4177 особей этого вида, отловленных у берегов Тасмании, также может быть скачана из архива UCI Machine Learning Repository. Возраст моллюсков определяют, прорисовывая и окрашивая раковину по конусу, после чего подсчитывают число колец под микроскопом. Это – скучная и отнимающая много времени работа, поэтому ставится задача предсказания возраста морского ушка по физическим измерениям.

Познакомимся с анализируемой выборкой:

```
# abalone <- read.csv("http://archive.ics.uci.edu/ml/
#       machine-learning-databases/abalone/abalone.data",
#       header=FALSE)
abalone <- read.csv("abalone.data", header = FALSE)
names(abalone)  <-  c("пол",  "длина",  "диаметр",  "высота",
"вес.общ", "вес.тела", "вес.внут", "вес.рак", "rings")
  summary(abalone[,c(1,2,9)])
```

```
пол          длина           rings
 F:1307   Min.   :0.075   Min.   : 1.000
 I:1342   1st Qu.:0.450   1st Qu.: 8.000
 M:1528   Median :0.545   Median : 9.000
          Mean   :0.524   Mean   : 9.934
          3rd Qu.:0.615   3rd Qu.:11.000
          Max.   :0.815   Max.   :29.000
```

Показатель `rings`, который мы не стали переводить на русский язык – упомянутое число возрастных колец на раковине моллюска. Отловленные животные принадлежат к трем группам по показателю `пол`: мужской (`M`), женский (`F`) и ювенильные особи (`I`). Построим несколько графиков с использованием пакетов `ggplot2` и `ggcorrplot`, которые позволяют сделать некоторые важные предварительные выводы. В частности, а) морфометрические показатели сильно коррелируют как между собой и, в несколько меньшей мере, с числом колец `rings` (рис. 8.2), б) между ними существует визуально отчетливая линейная зависимость, имеющая разный коэффициент угла наклона для разных половых групп (рис. 8.3):

```
require(ggplot2) ;  library(ggcorrplot)
# Изображение корреляционной матрицы (рис. 8.2)
M <- cor(abalone[,2:8])
ggcorrplot(M, hc.order = TRUE, type = "lower",
  colors = c("white","yellow","purple" ),     lab = TRUE)
# Изображение линейной зависимости (рис. 8.3)
ggplot(abalone) + aes(длина, rings, color = пол) +
    geom_point() + labs(x = "Длина раковины",
                y = "Число колец", color = "Пол") +
    stat_smooth(method = "lm", se = FALSE, size = 2)
```

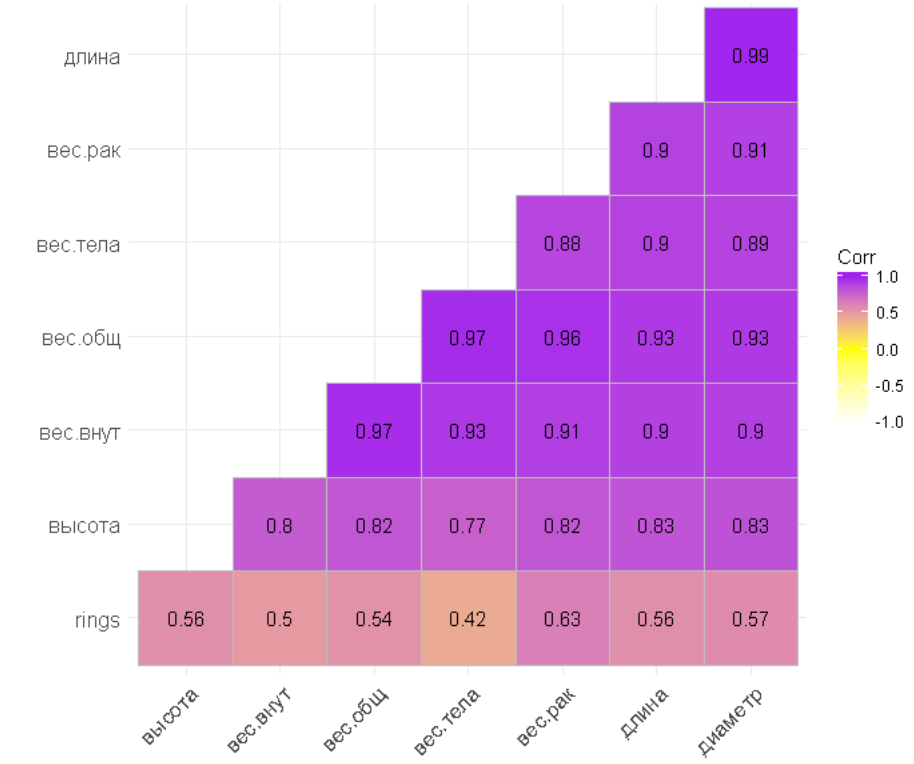


Рис. 8.2. Корреляционная матрица морфометрических показателей

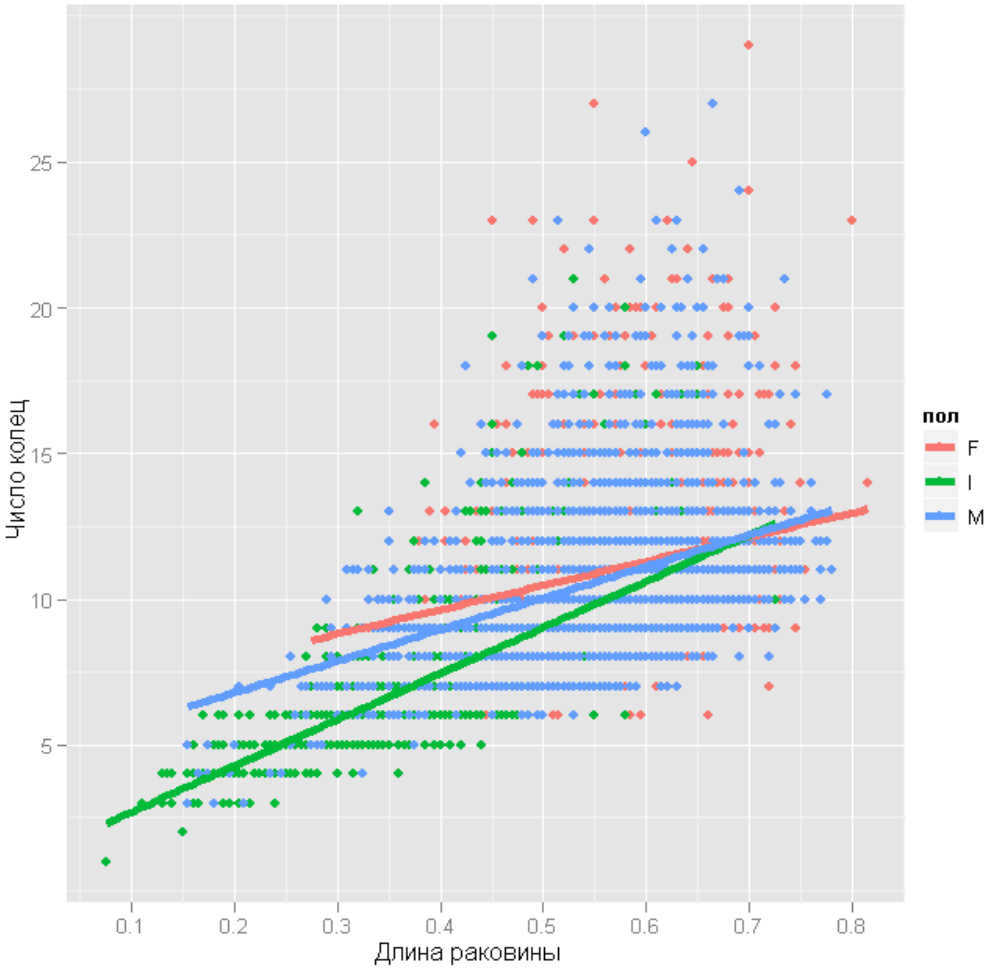


Рис. 8.3. Зависимость числа колец от длины раковины

Разделим интервал варьирования `rings` на диапазоны и выделим четыре возрастные группы. С точки зрения теории информации назначение границ предпочтительнее всего осуществить из условия равной численности групп. Рассмотрим предварительно функцию плотности распределения и границы квартилей для возможного разделения (рис. 8.4):

```
ggplot(abalone) + aes(rings, fill = пол) +
     geom_density(position = "stack")+
     geom_vline(xintercept = quantile(abalone$rings,
     p = c(0.25, 0.5, 0.75)),colour = "blue",
     linetype = 5, size = 1.5)
table (cut(abalone$rings,breaks=quantile(abalone$rings,
     c(0,.25,.50,.75,1), include.lowest=TRUE)))
```

```
  [1,8]   (8,9]  (9,11] (11,29]
   1407     689    1121     960
```

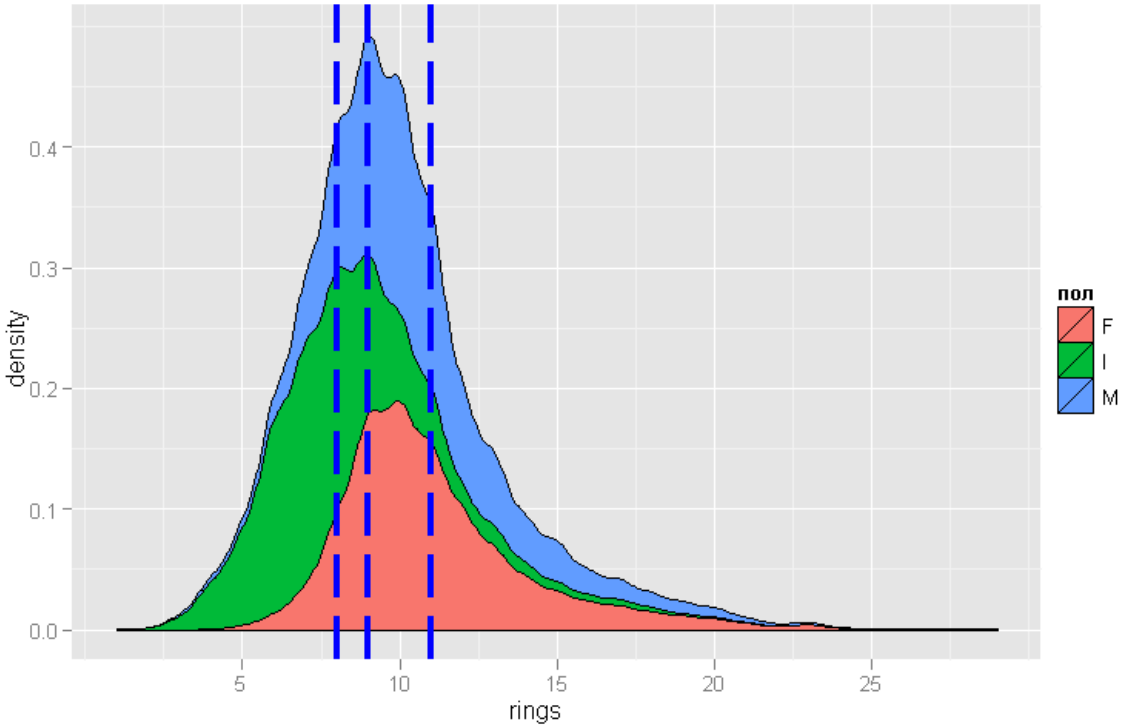


Рис. 8.4. Плотность распределения числа колец раковины и границы квартилей

Однако из практических соображений мы несколько скорректируем диапазоны, обозначенные квартилями, переместив объекты с 8 возрастными кольцами в группу более старых особей. Добавим новый столбец `Возраст` с четырьмя возрастными категориями в исходную таблицу `abalone` и сохраним ее в преобразованном виде в файле `abalone.RData`:

```
abalone$Возраст <- cut(abalone$rings, breaks=c(0,7,9,11,29),
  labels=c("Q1","Q2","Q3","Q4"), include.lowest=TRUE)
save(abalone, file="abalone.RData")
table (abalone$Возраст )
```

```
  Q1   Q2   Q3   Q4
 839 1257 1121  960
```

Сама по себе идея перейти от счетной переменной к категориальной переменной-фактору может показаться дискуссионной, но она дает нам возможность приступить к построению моделей с порядковым откликом. Модель кумулятивного логита получим с использованием функции `polr()` из пакета `MASS`.

```
library(MASS)
CL.aq <- polr(Возраст ~ .,  data = abalone[,-9])
summary(CL.aq, digits = 3)
```

```
Coefficients:
             Value Std. Error t value
полI      -0.9424     0.0934 -10.090
полМ       0.0542     0.0748   0.724
длина      1.5643     1.6722   0.935
диаметр    9.2270     2.0472   4.507
высота    13.4736     2.1206   6.354
вес.общ    6.3932     0.7897   8.095
вес.тела -15.3736     0.9126 -16.846
вес.внут  -7.1420     1.2760  -5.597
вес.рак    7.3099     1.2157   6.013
Intercepts:
      Value   Std. Error t value
Q1|Q2   4.056   0.326     12.447
Q2|Q3   6.540   0.341     19.206
Q3|Q4   8.487   0.344     24.644
Residual Deviance: 8224.074
AIC: 8248.074
```

```
# Оценка доверительных интервалов коэффициентов
confint.default(CL.aq)
```

```
                2.5 %      97.5 %
полI      -1.12547618  -0.7593602
полМ      -0.09236741   0.2006869
длина     -1.71315347   4.8417381
диаметр    5.21461784  13.2394216
высота     9.31738606  17.6298645
вес.общ    4.84535656   7.9410052
вес.тела -17.16223162 -13.5850167
вес.внут  -9.64297569  -4.6411202
вес.рак    4.92703181   9.6926746
```

Заметим, что блок коэффициентов состоит из двух разделов: коэффициенты β для независимых переменных и оценки параметра α, корректирующие отношение шансов при каждом шаге объединения альтернатив. Расчет доверительных интервалов показал, что два коэффициента β статистически незначимы, поскольку их доверительный интервал включает число 0 (отмечены курсивом).

Выше отмечалась высокая мультиколлинеарность данных, т.е. высокая степень взаимной зависимости морфометрических показателей. Считается, что наиболее эффективный путь устранения мультиколлинеарности – исключение из регрессионной модели малозначимых коэффициентов, или, выражаясь точнее, отбор информативного комплекса из q переменных ($q < m$). Пошаговый регрессионный анализ, выполняемый функцией `stepAIC()`, представляет собой последовательную процедуру включения и исключения

отдельных предикторов в модель, пока не будет достигнута оптимальная регрессия по критерию Акаике.

В результате проведенной пошаговой процедуры был исключен показатель длина, а AIC-критерий уменьшился с 8248 до 8247. Однако последующие итерации не привели к его дальнейшей оптимизации (мы опускаем итоги расчетов, поскольку они незначительно отличаются от приведенных выше). Тест на адекватность полученной модели в целом проведем, сравнивая отношения правдоподобия нуль-модели без параметров (1) и модели CLs.aq (2):

```
CLs.aq <- stepAIC (CL.aq)
summary(CLs.aq, digits = 3)
confint.default(CLs.aq)
CL0.aq <- polr(Возраст ~ 1,  data = abalone[,-9])
anova(CL0.aq, CLs.aq)
```

```
Likelihood ratio tests of ordinal regression models
Model Resid. df  Resid. Dev   Test    Df LR stat. Pr(Chi)
1     4174        11484.659
2     4166         8224.946 1 vs 2     8 3259.712       0
```

Модель имеет высокую статистическую значимость, что практически всегда бывает, если число наблюдений превышает несколько тысяч.

Рассмотрим теперь эффективность использования построенной модели для прогнозирования возраста моллюсков. С помощью функции fitted() найдем значения вероятностей для каждой из 4 рассматриваемых градаций, которые вычисляются на основе оценок коэффициентов α_j и β для модели CLs.aq. Отнесение каждого объекта выборки к конкретной возрастной категории будем осуществлять по максимальной вероятности. Как всегда, функция table() поможет нам выделить несовпадение действительных и предсказанных значений возрастных категорий.

```
Probs <- fitted(CLs.aq)
head(Probs, 3)
```

```
          Q1        Q2         Q3          Q4
1 0.16982105 0.5401767 0.23502603 0.054976171
2 0.42821924 0.4714134 0.08472626 0.015641138
3 0.03303907 0.2572052 0.45143960 0.258316097
```

```
#  Построение матрицы неточностей "Факт-Прогноз"
pred=apply(Probs,1,function(x)
colnames(Probs)[which(x==max(x))])
table(Факт=abalone$Возраст,Прогноз=pred)
Acc = mean(pred == abalone$Возраст)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
    Прогноз
Факт  Q1  Q2  Q3  Q4
  Q1 589 233  14   3
  Q2 113 773 306  65
  Q3  22 379 446 274
  Q4   6 144 292 518
 [1] "Точность=55.69%"
```

Можно отметить, что модель недостаточно хорошо справилась с предсказанием смежных возрастных категорий.

Многомерность параметров модели не позволяет показать графически изменение вероятности классов в зависимости от значений всех предикторов одновременно. Распространено мнение, что это можно сделать для отдельных предикторов, если зафиксировать значения остальных независимых переменных на уровне их средних значений. Выполним такой частный анализ, например, для диаметра раковины (см. рис. 8.5):

```
#   Подготовка данных для графика
VM <- apply(abalone[,2:8],2, mean)
d.plot <- as.data.frame(matrix(VM, ncol=7, nrow=50,
        byrow=TRUE, dimnames = list(1:50,names(VM))))
d.plot <- cbind(пол=rep("F",50), d.plot)
d.plot$диаметр <- seq(min(abalone$диаметр),
         max(abalone$диаметр), len = 50)
d.pplot <- cbind(d.plot, predict(CL.aq, newdata = d.plot,
         type = "probs", se = TRUE))
#   Прорисовка компонент графика
plot (1,1, xlim=c(0,max(abalone$диаметр)),ylim=c(0,0.8),
  type='n', xlab="Диаметр раковины", ylab="Вероятность P")
lines(d.pplot[,c(3,9)],lwd=2, col="green")
lines(d.pplot[,c(3,10)],lwd=2, col="blue")
lines(d.pplot[,c(3,11)],lwd=2, col="black")
lines(d.pplot[,c(3,12)],lwd=2, col="red")
legend("topright", c("Q1","Q2","Q3","Q4"), lwd=2,
       col=c(3,4,1,2))
```

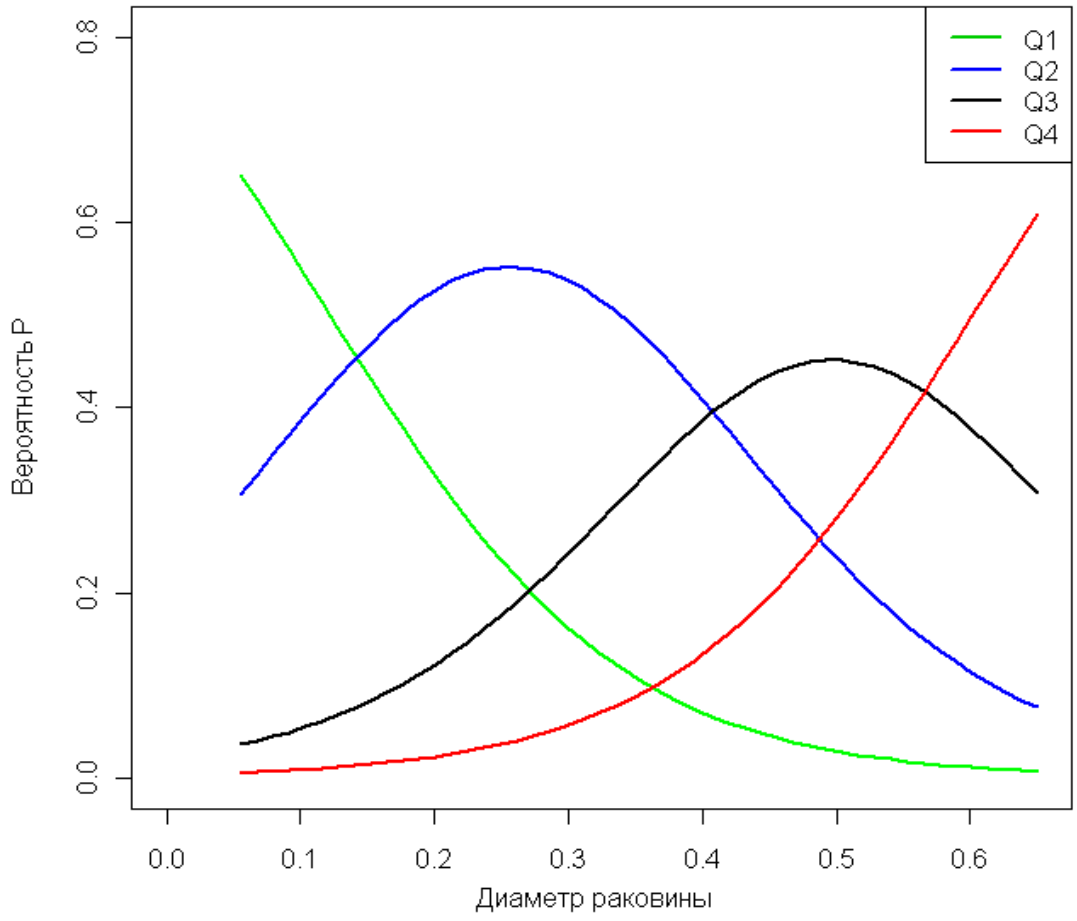


Рис. 8.5. Вероятности отнесения моллюсков к возрастным категориям в зависимости от диаметра раковины

На рис. 8.5 видно, что экстремальные значения диаметра раковины соответствуют максимальным вероятностям отнесения к младшим и старшим возрастным категориям соответственно слева и справа. Промежуточные значения диаметра, вероятнее всего, приведут к средним возрастам Q4 и Q3.

8.2. Настройка параметров нейронных сетей средствами пакета caret

В разделе 7.5 мы рассмотрели основные принципы построения искусственных нейронных сетей на примере многослойного персептрона. Основная проблема обучения ИНС заключается в необходимости предварительно исследовать поверхность ошибок и задать архитектуру сети и параметры оптимизации, по возможности, приводящие в окрестность глобального минимума. С использованием функции `train()` из пакета `caret` путем перекрестной проверки можно оценить оптимальные значения числа скрытых нейронов `size` и параметр "ослабления весов" `decay`, который осуществляет регуляризацию точности подстройки коэффициентов (при `decay` = 0 стремление к точности может перерасти в эффект переусложнения модели). Покажем, как это можно сделать, на примере оценки возрастной категории морских ушек:

```
library(nnet); library(caret)
set.seed(123); load (file="abalone.RData")
train.aba <- train(Возраст ~ ., data = abalone[,c(3:8,10)],
   method = "nnet", trace = F, linout = 1,
   tuneGrid = expand.grid(.decay = c(0,0.05,0.2), .size = 4:9),
trControl = trainControl(method = "cv"))
```

```
4177 samples     6 predictors
   4 classes: 'Q1', 'Q2', 'Q3', 'Q4'
Resampling: Cross-Validation (10 fold)
Resampling results across tuning parameters:
  size  decay  Accuracy  Kappa  Accuracy SD  Kappa SD
  5     0.2    0.589     0.446  0.0211       0.0291
  6     0      0.593     0.452  0.0255       0.0348
  6     0.05   0.592     0.45   0.0251       0.0346
  6     0.2    0.59      0.447  0.0231       0.0318
  7     0      0.595     0.455  0.0287       0.0391
  7     0.05   0.591     0.449  0.0245       0.034
  7     0.2    0.59      0.447  0.0226       0.0312
  8     0      0.59      0.448  0.0257       0.035
  8     0.05   0.588     0.445  0.0256       0.0353
Accuracy was used to select optimal model using largest value.
The final values used for model were size = 7 and decay = 0.
```

Мы выполнили 10-кратную перекрестную проверку 18 нейросетевых моделей с числом нейронов в скрытом слое от 4 до 9 и разных значениях "ослабления". При найденных значениях `size = 7` и `decay = 0`, приводящих к максимальной точности `Accuracy`, построим далее модель с помощью функции `nnet()`. Для визуализации сети (рис. 8.6) применим функцию из скрипта `nnet_plot_update.r`, которая имеет ряд полезных опций (можно скачать с https://www.r-bloggers.com).

```
source("nnet_plot_update.r")
# plot.nnet(train.aba)  - можно все извлечь из train-объекта
# nn.aba <- train.aba$finalModel
nn.aba <- nnet(Возраст ~ .,  data = abalone[, c(3:8, 10)],
               decay = 0, size = 7, niter=200)
plot.nnet(nn.aba)
summary(nn.aba)
```

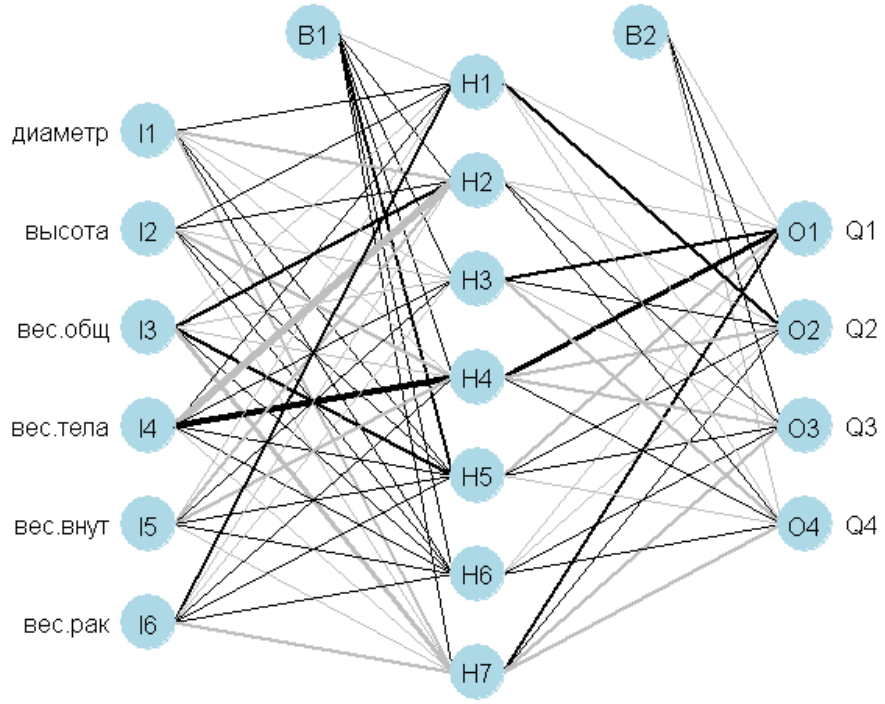


Рис. 8.6. Персептрон для оценки возрастной категории морских ушек

Мы получили сеть из 17 нейронов (6 ⇒7 ⇒4), для которых рассчитано 81 нижеприведенных весовых коэффициентов и порогов b.

```
a 6-7-4 network with 81 weights
b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1
 -8.67  10.81   0.22  -4.89   7.14  -1.40  16.56
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2
  3.03  -3.80  -4.03   4.59  -4.55   7.01 -14.24
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3
 -5.70  34.35   6.02  -2.66  -1.47   1.00  -5.93
 b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4
  7.87 -18.03  -0.85  18.30 -24.87  -6.31 -21.53
 b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5
 -2.20   9.24 -16.03 -23.37  41.34  10.69  16.89
 b->h6 i1->h6 i2->h6 i3->h6 i4->h6 i5->h6 i6->h6
 -7.12  -8.78  -5.86   1.73  -5.04  18.64   5.11
 b->h7 i1->h7 i2->h7 i3->h7 i4->h7 i5->h7 i6->h7
 -2.62  10.64   1.02  -7.28  11.45   1.54 -13.55
 b->o1 h1->o1 h2->o1 h3->o1 h4->o1 h5->o1 h6->o1 h7->o1
  7.05   0.24  -0.42 -15.02   5.22   4.68  -1.05  11.34
 b->o2 h1->o2 h2->o2 h3->o2 h4->o2 h5->o2 h6->o2 h7->o2
  4.49  -2.53   0.54  -3.46  -1.97   1.39  -6.17   2.48
 b->o3 h1->o3 h2->o3 h3->o3 h4->o3 h5->o3 h6->o3 h7->o3
 -3.14   0.81   1.71   5.18  -2.48  -1.39  -1.04  -3.68
 b->o4 h1->o4 h2->o4 h3->o4 h4->o4 h5->o4 h6->o4 h7->o4
 -7.55   1.40  -1.86  12.61  -1.04  -3.73   8.48 -10.25
```

Рассмотрим теперь, насколько хороша полученная нейросетевая модель при выполнении предсказаний:

```
pred = predict(nn.aba,   abalone[,3:8],   type="class")
nn.table = table(abalone[,10],  pred)
confusionMatrix(nn.table)
```

```
Confusion Matrix and Statistics
    pred
      Q1  Q2  Q3  Q4
  Q1 660 166   7   6
  Q2 177 764 231  85
  Q3  46 367 492 216
  Q4  10 129 239 582
Overall Statistics
               Accuracy : 0.598
                 95% CI : (0.583, 0.613)
    No Information Rate : 0.3414
    P-Value [Acc > NIR] : < 2.2e-16
                  Kappa : 0.4591
 Mcnemar's Test P-Value : 2.33e-13
Statistics by Class:
                     Class: Q1 Class: Q2 Class: Q3 Class: Q4
Sensitivity             0.7391    0.5358    0.5077    0.6547
Specificity             0.9455    0.8208    0.8039    0.8850
Pos Pred Value          0.7867    0.6078    0.4389    0.6063
Neg Pred Value          0.9302    0.7733    0.8439    0.9046
Prevalence              0.2138    0.3414    0.2320    0.2128
Detection Rate          0.1580    0.1829    0.1178    0.1393
Detection Prevalence    0.2009    0.3009    0.2684    0.2298
```

Точность предсказания возрастных категорий несколько возросла по сравнению с кумулятивным логитом (0.598 против 0.557), даже несмотря на то, что ковариату пол мы не использовали.

Рассмотрим кратко еще две функции из пакета `caret`, предназначенные для работы с искусственными нейронными сетями. Серьезной проблемой при обучении ИНС может стать высокая мультиколлинеарность предикторов. Одним из вариантов ее решения является проецирование исходной выборки в пространство главных компонент (см. метод PCR в разделе 4.2). Если задаться некоторым порогом (аргумент `thresh`), то значения p главных компонент, для которых накопленная доля объясненной дисперсии превышает этот порог, можно использовать как входы в нейронную сеть и настроить ее обычным способом как для регрессии, так и для классификации.

Функция `pcaNNet()` является своеобразной оберткой для совместного выполнения предобработки данных, анализа главных компонент, и запуска функции `nnet()` для обучения сети на полученных главных компонентах.

Выполним обучение сети для предсказания возрастной категории морских ушек с использованием параметра `thresh = 0.975` и числа нейронов в скрытом слое `size = 7`:

```
pcaNNet.Fit <- pcaNNet(abalone[,3:8], abalone[,10],
     size = 7, thresh = 0.975, linout = TRUE, trace = FALSE)
```

```
Neural Network Model with PCA Pre-Processing
PCA needed 4 components to capture 97.5 % of the variance
```

```
a 4-7-4 network with 67 weights
```

При значении thresh = 0.975 на вход сети подается 4 главных компоненты, отвечающих этому условию. Для тестируемых данных такое же преобразование (основанное на факторных нагрузках для обучающего множества) применяется к новым значениям предикторов.

```
pred <- predict(pcaNNet.Fit ,abalone[,3:8],   type="class")
table(Факт=abalone$Возраст,Прогноз=pred)
Acc = mean(pred == abalone$Возраст)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
    Прогноз
Факт  Q1  Q2  Q3  Q4
  Q1 654 170   8   7
  Q2 187 694 266 110
  Q3  54 302 511 254
  Q4  15 137 224 584
 [1] "Точность=58.49%"
```

В разделе 4.4 была описана процедура бэггинга, как общего метода агрегирования моделей произвольной структуры, построенных на бутстреп-выборках из исходного набора данных. Функция avNNet() осуществляет обучение заданного множества моделей нейронной сети на одном и том же наборе данных. Модели могут различаться как из-за случайного дрейфа стартовых параметров калибровки сети (Ripley, 1996), так и вследствие использования бутстреп-выборок, извлеченных из исходного обучающего множества. Для моделей классификации функция avNNet() оценивает среднее значение вероятностей классов на основе частных прогнозов каждой из моделей созданного ансамбля и далее производит заключительное предсказание класса.

Для рассматриваемого примера сформируем 10 экземпляров моделей ИНС с использованием бэггинга:

```
avNNet.Fit <- avNNet(Возраст ~ ., data = abalone[,c(3:8,10)],
    size = 7, repeats = 10, linout = TRUE,
    trace = FALSE, bag = TRUE)
pred <- predict(avNNet.Fit ,abalone[,3:8],   type="class")
Acc = mean(pred == abalone$Возраст)
paste("Точность=", round(100*Acc, 2), "%", sep="")
```

```
Model Averaged Neural Network with 10 Repeats
a 6-7-4 network with 81 weights
[1] "Точность=59.48%"
```

8.3. Методы комплексации модельных прогнозов

Существует два возможных подхода к получению некоторого обобщенного предсказания на основе формального или неформального объединения частных прогнозов. В разделе 4.4 нами подробно рассматривалось агрегирование результатов моделирования, когда каждая модель имеет фактически одну и ту же структуру, но многократно обучается

на слегка модифицированных исходных данных или при небольшой вариации управляющих параметров (bagging). Другой путь – построить на одних и тех же исходных данных некоторый "коллектив" (ensemble) из принципиально разнотипных моделей и выполнить комбинирование прогнозов (forecast combinations).

По аналогии с методами коллективного решения, столь эффективно используемым в человеческом обществе, считается, что суммарная эффективность системы коллективного прогнозирования теоретически будет в среднем значительно выше любого из его членов. Этот оптимизм мало чем подтверждается на практике, но нельзя не признать, что комбинированный прогноз значительно более устойчив к не вполне объяснимым "провалам", которые свойственны отдельным индивидуальным методам, и поэтому может быть эффективным инструментом страхования на случай возможных стохастических рисков (в мире трейдинга это называется хеджированием, англ. hedging).

Пусть имеется m прогнозов $\boldsymbol{y_1}$, $\boldsymbol{y_2}$, ..., $\boldsymbol{y_m}$ для переменной $\boldsymbol{Y}$, полученных с помощью m различных моделей. Под *коллективным прогнозом* $\boldsymbol{g}$ будем понимать прогноз той же переменной $\boldsymbol{Y}$ как некоторую функции от индивидуальных прогнозов $\boldsymbol{y_k}$: $\boldsymbol{g} = F(\boldsymbol{y_1}, \boldsymbol{y_2},..., \boldsymbol{y_m}; \mathbf{X})$.

Число различных алгоритмов синтеза коллективов из нескольких моделей постоянно растет (даже неполная библиография по этой проблеме насчитывает свыше 500 наименований), а их классификация может быть весьма условной. Наиболее простыми являются *методы без адаптации*.

Коллективный прогноз $\boldsymbol{g}$, построенный без адаптации, представляют в виде линейной комбинации из базового или суженного (наиболее информативного) множества частных прогнозов $g = \sum_{k=1}^{m} y_k \omega_k$,

где $\boldsymbol{\omega}$ – вектор неизвестных весовых коэффициентов, компоненты которого неизменны на всем диапазоне варьирования исходных данных $\mathbf{X}$. Тогда задача комплексации эквивалентна определению совокупности значений вектора ω_k, удовлетворяющих заданным ограничениям и минимизирующих некоторый критерий качества. Очевидно, что качество (надежность) прогноза $\boldsymbol{g}$ тем выше, чем "ближе" расчетные значения $\boldsymbol{Y}$ к фактическим.

Обратимся вновь к примеру из предыдущих разделов, но будем теперь строить модели прогнозирования не для возрастных категорий, а непосредственно для числа колец в раковинах `rings` тасманских морских ушек (см. рис. 8.4). Заменим фактор `пол` на две метрические переменные `полM` и `полI`, после чего разделим исходную выборку на обучающую и проверочную в соотношении 2:1.

```
load(file="abalone.RData")
# Формирование матрицы предикторов
X <- model.matrix(rings ~ .,data=abalone[,1:9])[,-1]
Y <- abalone$rings
# Разделение на обучающую и проверочную выборки
train <- runif(nrow(X)) <= .66
```

На объектах обучающей выборки (2748 наблюдений) построим шесть моделей оценки rings с использованием функции train() из пакета caret:

```
library(caret); library(party)
myControl <- trainControl(method='cv', number=10,
  savePredictions=TRUE, returnData=FALSE, verboseIter=TRUE)
PP <- c('center', 'scale')
# Обучение избранных моделей
# Регрессия на k-ближайших соседей
m.knn <- train(X[train,], Y[train], method='knn',
               trControl=myControl, preProcess=PP)
# Линейная модель
m.lm <- train(X[train,], Y[train], method='lm',
               trControl=myControl, preProcess=PP)
# Гребневая регрессия с регуляризацией
m.rlm <- train(X[train,], Y[train], method='glmnet',
               trControl=myControl, preProcess=PP)
# Модель опорных векторов
m.svm <- train(X[train,], Y[train], method='svmRadial',
               trControl=myControl, preProcess=PP)
#  Метод случайного леса
m.rf <- train(X[train,], Y[train], method='rf',
               trControl=myControl)
# Бэггинг деревьев условного вывода
m.ctr <- train(X[train,], Y[train], "bag",
    B = 10, bagControl = bagControl(fit = ctreeBag$fit,
    predict = ctreeBag$pred, aggregate = ctreeBag$aggregate))
```

Обратим внимание на модель m.ctr, построенную методом "bag". При построении этой модели используется специальная функция bag() из пакета caret, позволяющая выполнить бэггинг широкого класса различных моделей (daBag, plsBag, nbBag, ctreeBag, svmBag, nnetBag). Мы осуществили бэггинг деревьев условного вывода (см. раздел 4.3), агрегируя каждый раз по 10 деревьев.

Упакуем результаты моделирования в объект класса list и извлечем значения среднеквадратичной ошибки RMSE каждой индивидуальной модели на обучающей выборке:

```
# Создание списка всех моделей
all.models <- list(m.knn, m.lm, m.rlm, m.svm, m.rf, m.ctr)
names(all.models) <- sapply(all.models, function(x) x$method)
sort(sapply(all.models, function(x) min(x$results$RMSE)))
```

```
       rf svmRadial    glmnet        lm       bag       knn
 2.136772  2.153634  2.172855  2.178365  2.222380  2.233542
```

В целом все модели показали близкие результаты, хотя можно отметить традиционное лидерство методов случайного леса (rf) и опорных векторов (svmRadial).

Сформируем таблицу частных прогнозов для объектов тестовой выборки (1429 наблюдений) и оценим ошибки индивидуальных моделей RMSE:

```
preds.all <- data.frame(sapply(all.models,
                  function(x){predict(x, X)}))
head(preds.all)
```

```
        knn       lm    glmnet svmRadial       rf      bag
1  7.111111  7.732803  7.781386  7.389098  8.417767  8.031008
2  9.000000  9.614801  9.622614  9.537425  9.006100  9.451514
3 11.888889 13.043430 12.944758 12.409405 14.279467 12.646068
4 11.888889 10.927220 10.908910 11.563814 12.673133 10.957280
5  9.555556  8.813379  8.783278  9.038425  8.866833  8.803584
6  8.111111  8.360550  8.393360  8.198748  8.211833  8.467949
```

```
rmse <- function(x,y){sqrt(mean((x - y)^2))}
print("Среднеквадратичная ошибка на тестовой выборке")
print(sort(apply(preds.all[!train,], 2, rmse, y = Y[!train]))
                , digits= 3)
```

```
[1] "Среднеквадратичная ошибка на тестовой выборке"
      rf svmRadial      bag       lm    glmnet      knn
    2.17      2.20     2.21     2.28     2.28     2.36
```

В целом незначительное (2-4%) увеличение ошибки RMSE на свежих наблюдениях свидетельствует о хорошем качестве моделей.

Рассмотрим теперь возможные алгоритмы комплексации частных прогнозов, реализованные в пакете ForecastCombinations. Два из них основаны только на самих предсказаниях и не требуют предварительного обучения:

- простое среднее (simple) при $\omega_k = \text{const} = 1/m$;
- по лучшему частному прогнозу (best), т.е. $\omega_k = 1$ для $\max(y_k)$ и $\omega_k = 0$ для остальных y_k.

Комплексация, основанная на дисперсиях (variance based), требует оценки на некоторой обучающей выборке значений среднеквадратичных отклонений MSE_k для каждой индивидуальной модели и тогда

$$\omega_k = (1/MSE_k)/\sum_{k=1}^{m}(1/MSE_k).$$

Другим способом комплексации прогнозов разных моделей является использование линейной модели, в которой коэффициенты играют роль весов, определяющих вклад каждой из объединяемых моделей. С легкой руки букмекеров эту процедуру стали часто называть *стэкингом* (stacking).

Разумеется, обычный метод наименьших квадратов (ols) является одним из претендентов на создание такой "модели моделей". Однако, поскольку прогнозы частных моделей сильно коррелируют между собой, то для получения комплексного результата предлагают использовать иные методы регрессии, которые, по мнению авторов, могут лучше справиться с этой напастью. В пакет ForecastCombinations включены метод наименьших абсолютных отклонений (robust) и наименьших квадратов с ограничениями (cls - Constrained Least Squares). Во втором случае на значения коэффициентов регрессии накладываются ограничения: все они неотрицательны и в сумме равны 1, т.е. по сути отражают вклад каждой частной модели в общий прогноз. Коэффициенты CLS-модели рассчитываются с использованием метода квадратичного программирования, представленного функцией solve.QP() из пакета quadprog.

На вход функции Forecast_comb() из пакета ForecastCombinations подаются объекты для обучения (матрица fhat со значениями частных

прогнозов y_1, y_2,..., y_m и вектор obs с соответствующими наблюдаемыми величинами y) и прогноза (матрица fhat_new со значениями частных прогнозов, которые следует объединить с использованием метода Averaging_scheme). На выходе формируются векторы прогнозов pred и весов weights. С учетом выполненного нами разбиения исходной таблицы abalone рассчитаем сравнительную оценку RMSE, полученную каждым методом комплексации на проверочной выборке:

```
library(ForecastCombinations)
scheme= c("simple",  "variance based", "ols", "robust",
          "cls", "best")
combine_f <- list()
for (i in scheme) {
   combine_f[[i]] <- Forecast_comb(obs = Y[train],
        fhat = as.matrix(preds.all[train, ]),
        fhat_new = as.matrix(preds.all[!train, ]),
        Averaging_scheme = i)
    tmp <- round(sqrt(mean((combine_f[[i]]$pred -
           Y[!train])^2)), 3)
    cat(i, ":", tmp, "\n")
 }
```

```
simple : 2.16
variance based : 2.143
ols : 2.266
robust : 2.266
cls : 2.166
best : 2.166
```

Рассмотрим веса, которые получили каждые модели при реализации каждой схемы усреднения (приведем их предварительно к единой шкале):

```
w.list <- sapply(combine_f, function(sp.list) sp.list$weights)
w.list$ols <- w.list$ols[-1]
w.list$robust <- w.list$robust[-1]
weights <- as.data.frame.list(w.list)
rownames(weights) <- scheme
wstan <- function (x) abs(x)/sum(abs(x))
weights[3,] <- wstan(weights[3,])
weights[4,] <- wstan(weights[4,])
print(round(weights,3))
```

```
                knn    lm glmnet svmRadial    rf   bag
simple        0.167 0.167  0.167     0.167 0.167 0.167
variance based 0.111 0.094  0.094     0.105 0.489 0.108
ols           0.056 0.077  0.063     0.088 0.660 0.056
robust        0.043 0.137  0.128     0.086 0.562 0.043
cls           0.000 0.000  0.000     0.000 1.000 0.000
best          0.000 0.000  0.000     0.000 1.000 0.000
```

Комплексные прогнозы, созданные по схемам cls и best, эквивалентны мнению "непогрешимого" авторитета (а таковым оказался метод rf). Соответственно они дали ошибку, в точности равную погрешности наиболее качественной индивидуальной модели. Схемы ols и robust потерпели относительную неудачу, тогда как наиболее точным оказался прогноз, основанный на дисперсии (variance based).

П. Полищук (http://www.qsar4u.com) предлагает применять метод неотрицательных наименьших квадратов (non-negative least squares), используя пакет `nnls`:

```
require(nnls)
m.nnls <- nnls(as.matrix(preds.all[train, ]), Y[train])
coef(m.nnls)
```

```
[1] 0.000000 0.000000 0.000000 0.000000 1.008773 0.000000
```

Этот метод дал те же значения весов, что и CLS-регрессия (впрочем, эти методы идентичны, по-видимому, и по своей сути).

Кроме линейного стэкинга для комплексации можно применять любые изощренные алгоритмы статистического моделирования. Например, набор частных прогнозов можно выразить через небольшое число главных компонент и на последующих шагах применить любой другой алгоритм взвешивания (Горелик, Френкель, 1983)[3]. П. Брусиловский и Г. Розенберг (1983) использовали для комплексации многорядный алгоритм МГУА, строили самообучающиеся иерархические модели и назвали этот метод "модельным штурмом". Дело устойчиво идет к проблематике создания "коллектива коллективов".

Пакет `ForecastCombinations` также позволяет реализовать эксперименты подобного рода. Из 6 векторов частных прогнозов, полученных нами по индивидуальным моделям, можно построить 53 модели регрессии с различными наборами переменных от одного до 5. Функция `Forecast_comb_all()` выполняет настройку всех этих моделей и сводит результаты в новую таблицу (т.е. вместо 6 прогнозов предлагается иметь дело с 53-мя прогнозами на основе прогнозов). Далее их можно просто усреднить:

```
combine_f_all <- Forecast_comb_all(Y[train],
       fhat = as.matrix(preds.all[train, ]),
   fhat_new = as.matrix(preds.all[!train, ]))
# Усредняем комбинированные прогнозы по всем регрессиям
Combined_f_all_simple <- apply(combine_f_all$pred, 1, mean)
print(sqrt(mean((Combined_f_all_simple - Y[!train])^2)),
      digits= 3 )
```

```
[1] 2.16
```

Для моделей регрессии по каждому подмножеству отдельных моделей функция `Forecast_comb_all()` рассчитывает также набор информационных критериев (AIC, BIC и Mallow's Cp), которые могут быть использованы для расчета взвешенного среднего:

```
# Комбинирование прогнозов с использованием  Mallow's Cp:
Combined_f_all_mal <-
         t( combine_f_all$mal %*% t(combine_f_all$pred) )
print(sqrt(mean((Combined_f_all_mal - Y[!train])^2)),
      digits= 3)
```

[3] Информация о всех литературных источниках этого раздела приведена в (Розенберг и др., 1994)

```
[1] 2.24
```

Итогом всех этих достаточно трудоемких вычислительных процедур оказался весьма скромный результат.

Альтернативой механическому усреднению прогнозов является учет структурных связей в ансамбле таким образом, чтобы положительные свойства той или иной модели (метода) доминировали при формировании коллективного отклика, а отрицательные – отфильтровывались (Растригин, Эренштейн, 1981). В частности, существенным влиянием должны обладать самые "непохожие" между собой модели-индивидуумы, являющиеся носителями нетривиальных тенденций моделируемого процесса. Основным направлением развития в этой области является разработка методов адаптации, в основе которых лежат алгоритмы Бейтса-Гренджера (Bates, Grander, 1965), Ньюболда-Гренджера (Newbald, Grander, 1974), Ершова (1975) и др. К сожалению, нам не удалось обнаружить пакеты R, в которых были бы реализованы вменяемые методы комплексации прогнозов с адаптацией.

8.4. Обобщенные линейные модели для счетных данных

В предыдущих главах нами рассматривались примеры использования методов построения моделей для различных типов распределения отклика. При этом значительная часть реальных наблюдений сводится к подсчету числа экземпляров: это могут быть обследуемые респонденты, проданные гаджеты, тыс. клеток водорослей, нажатия кнопок мыши или птицы, пролетающие мимо наблюдателя. Эти данные по своей природе дискретны и может возникнуть вопрос, а какое распределение выбрать для их описания?

Этот выбор, в первом приближении, делается априори на основании имеющихся знаний относительно отклика. Например, если моделируется присутствие (1) и отсутствие (0) животных на некоторой изучаемой площади как функция нескольких факторов, то выбор прост: необходимо использовать биномиальное распределение. Однако это, вероятно, единственный сценарий, где ситуация столь очевидна.

Обычно, если это позволяет характер вариации переменной Y, прибегают к аппроксимации счетных данных тем или иным непрерывным распределением – нормальным, гамма или Вейбулла, – что мы в конце концов и сделали с числом колец в раковинах морских ушек (рис. 8.4). Хорошему результату подгонки параметров модели и гарантированной неотрицательности модельных значений может способствовать преобразование исходных данных с помощью различных функций, таких как, логарифмирование или трансформация Бокса-Кокса (см. раздел 3.3).

Однако зачастую дискретность отклика оказывается столь очевидной ($y_i = 0, 1, 2, \ldots$), что аппроксимация непрерывным распределением оказывается неудовлетворительной и это сильно влияет на адекватность регрессии с использованием МНК. Если счетные данные имеют тенденцию к гетерогенности (т.е. в большинстве случаев можно не встретить ни одного

экземпляра, реже попадутся единичные особи, и лишь иногда их большие скопления) и фиксированный верхний предел у них отсутствует, то хорошим вариантом является аппроксимация наблюдений распределением Пуассона (Zuur et al., 2009).

Распределение Пуассона P(λ) имеет случайная величина *Y*, отражающая количество событий, произошедших за некоторый промежуток времени, когда эти события независимы и происходят с постоянной интенсивностью λ, $\lambda \in (0, \infty)$. Предполагается, что среднее $\mu = E(Y) = \lambda$ и дисперсия $\mathrm{Var}(Y) = \lambda$, а функция плотности вероятности $p(k) = e^{-\lambda} \lambda^k/k!$, $k = 1, 2, \ldots, \infty$, где $k!$ – факториал от числа событий.

Пуассоновскую регрессию применяют, когда отклик является *Y* счетной переменной, имеющей такое распределение. Соответствующая модель имеет вид

$$\ln(\lambda) = \beta_0 + \sum_p \beta_j x_j$$

где $x_1, \ldots, x_p$ – набор из *p* независимых переменных, β_0 – математическое ожидание *Y* при равенстве нулю всех предикторов x_j, β_j – коэффициенты независимых переменных.

Нередкими проблемами подгонки распределением Пуассона являются:

- значительное по сравнению с теорией количество нулевых значений в выборке, сопровождающееся сильно разреженными таблицами данных;
- эксцесс – избыток некоторых наблюдаемых значений по сравнению с ожидаемыми частотами;
- наличие большой гетерогенности данных, вследствие чего оценка дисперсии начинает значимо превышать среднее, $\mathrm{Var}(Y) >> \mu$ (overdispersion).

Если наблюдается отчетливая избыточная дисперсия ("перерассеяние"), то альтернативой распределению Пуассона для счетных данных является отрицательное биномиальное распределение.

Отрица́тельное биномиа́льное распределе́ние NB(*r*, *p*), также называемое распределением Паскаля, – это распределение дискретной случайной величины, которая отражает количество произошедших неудач в последовательности испытаний Бернулли с вероятностью успеха *p*, проводимой до *r*-го успеха (https://ru.wikipedia.org). Его обычно интерпретируют как смесь распределений гамма Γ(.) и Пуассона, т.е. функция плотности вероятности имеет вид: $f(y;\mu,\theta) = \frac{\Gamma(y+\theta)}{\Gamma(y)\cdot\theta!}\frac{\mu^y\theta^y}{\mu+\theta^{(y+\theta)}}$,

где $\mu = E(Y)$ – среднее, а θ – параметр формы распределения.

Включение в функцию распределения дополнительного параметра позволяет учесть превышение дисперсии над средним:

$$\mathrm{Var}(Y) = \mu + \mu^2 / \theta.$$

При $\theta \to \infty$ отрицательное биномиальное распределение сводится к распределению Пуассона (Agresti, 2007; Zeileis et al., 2008).

Обобщенная линейная модель для отрицательного биномиального распределения оценивает зависимость μ от совокупности объясняющих переменных с использованием функции связи (link function), как и в случае с лог-линейной моделью Пуассона. Принято считать, что параметр формы θ постоянен на всей области определения модели, что аналогично предположению о гомоскедастичности для моделей с нормальным распределением остатков.

Если аналитик колеблется в выборе между двумя конкурирующими статистическими моделями, то основанием для выбора может служить оценка согласия между наблюдаемыми данными и GLM с соответствующим теоретическим распределением. Рассмотрим эту задачу на примере двухлетнего подсчета числа амфибий, раздавленных автомобилями на 52 участках дорожной сети в южной части Португалии. Файл с данными `RoadKills.txt` можно скачать с полным комплектом данных к книге Zuur et al. (2009) на сайте http://www.highstat.com/book2.htm.

Само по себе распределение отклика `TOT.N` (числа погибших животных на каждом участке, экз.) трудно принять как пуассоновское (рис. 8.7):

```
RK <- read.table(file = "RoadKills.txt",
                 header = TRUE, dec = ".")
y <- RK$TOT.N
library(vcd)
gf<-goodfit(table(y),type= "poisson",method= "ML")
#  Визуализация подогнанных данных гистограммой
plot(gf,ylab="Частоты", xlab="Число классов")
```

```
        Goodness-of-fit test for poisson distribution
                      X^2 df      P(> X^2)
Likelihood Ratio 970.7709 29 1.290368e-185
```

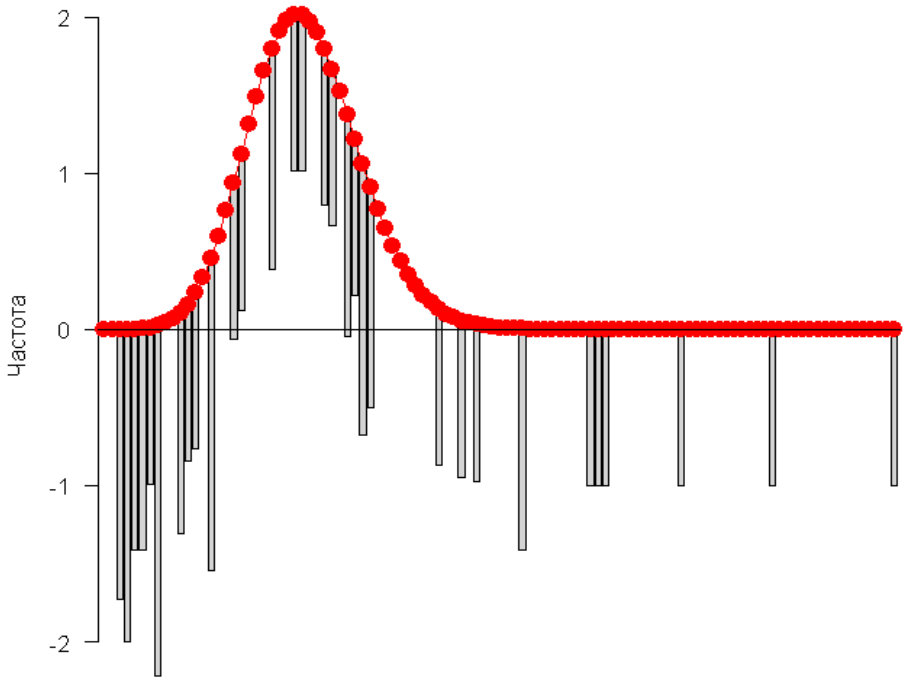


Рис. 8.7. Гистограмма отклонений эмпирических частот от теоретических частот распределения Пуассона (красные кружки)

Модель регрессии Пуассона дает нам возможность объяснить эти отклонения за счет влияния внешних факторов. На первом этапе оценим общий вид моделируемых зависимостей, для чего применим одну независимую переменную – D.PARK (расстояние до заповедника), которая нам представляется наиболее экологически значимой:

```
M1 <-  glm(TOT.N ~ D.PARK, family =  poisson, data =  RK)
summary(M1)
```

```
Coefficients:
             Estimate   Std. Error  z value    Pr(>|z|)
(Intercept) 4.316e+00   4.322e-02    99.87     <2e-16
D.PARK     -1.059e-04   4.387e-06   -24.13     <2e-16
Null deviance: 1071.4  on 51  degrees of freedom
Residual deviance:  390.9  on 50  degrees of freedom
AIC: 634.29
```

Была построена модель $\log(\mu_i) = 4.13 - 0.0000106 \times \text{D.PARK}_i$, или в другой форме, $\mu_i = \exp(4.13 - 0.0000106 \times \text{D.PARK}_i)$, где μ_i - прогнозируемая величина пуассоновской частоты с учетом значения расстояния D.PARK_i. Построим график полученной зависимости:

```
require(ggplot2)
MyData=data.frame(D.PARK=seq(from=0,to=25000,by=1000))
G <- predict(M1, newdata=MyData, type="link", se=T)
MyData$FIT <- exp(G$fit)
MyData$FSEUP <- exp(G$fit+1.96*G$se.fit)
MyData$FSELOW <- exp(G$fit-1.96*G$se.fit)
ggplot(MyData, aes(D.PARK, FIT)) +
  geom_ribbon(aes(ymin = FSEUP, ymax = FSELOW),
  fill = 3, alpha = .25) +   geom_line(colour = 3) +
  labs(x = "Расстояние до парка", y = "Убийства на дорогах")+
  geom_point(data=RK, aes(D.PARK, TOT.N) )
```

Обращает на себя внимание очень узкая полоса 95%-го доверительного интервала относительно линии регрессии на рис. 8.8. Это обусловлено, возможно, как неверной спецификацией построенной модели, так и заниженной оценкой остаточной дисперсии.

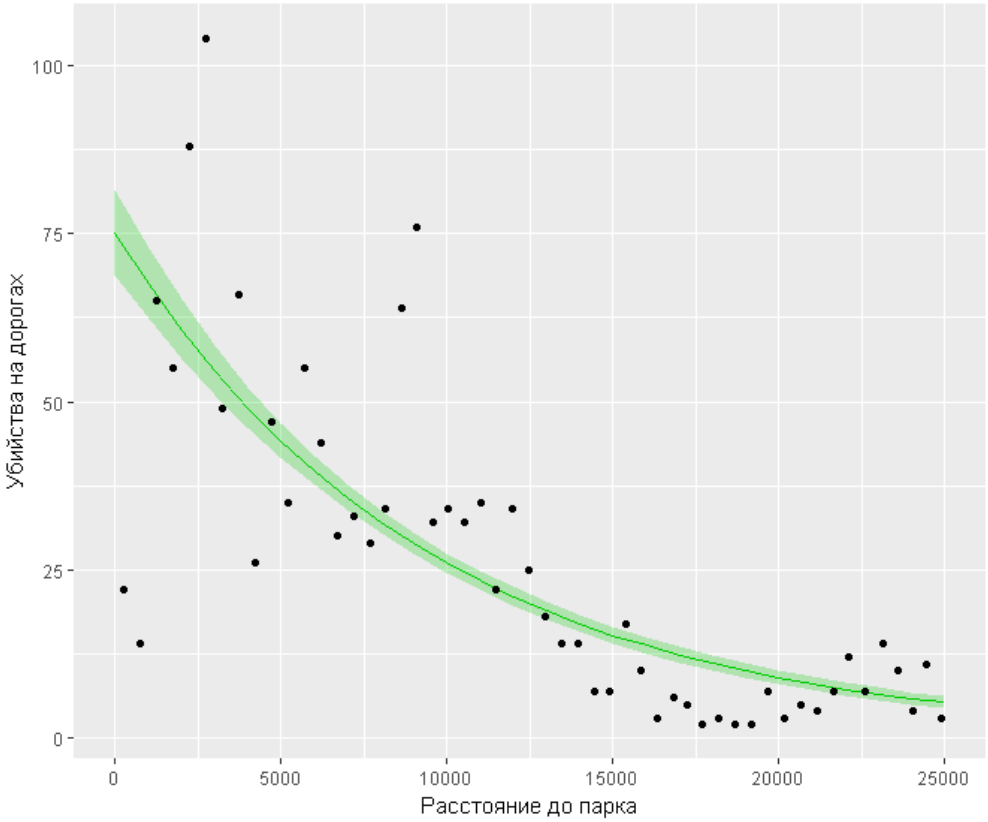


Рис. 8.8. Зависимость числа раздавленных лягушек от расстояния до натурального лесного массива

Включим в модель дополнительные предикторы с целью повышения ее адекватности. В исходной таблице данных для этого имеется 17 полезных переменных-кандидатов. Выполним, однако, предварительно две операции – преобразование данных и удаление высоко коррелированных признаков.

Поскольку 8 переменных имеют размерность площади (га), а другие заданы длиной (расстоянием в м), то разумно привести их к единой функциональной форме и извлечь квадратный корень из значений первой группы признаков. Далее оценим степень мультиколлинеарности 17-мерного комплекса переменных и вычислим для каждого предиктора фактор инфляции дисперсии (Variance Inflation Factor, VIF).

```
RK[, 7:14] <- sqrt(RK[, 7:14])
library(car)
res <- sort(vif(glm(TOT.N ~ ., data=RK[,c(5,7:23)],
      family =  poisson))) ; print(res,4)
RK.11 <- cbind(RK$D.WAT.RES, RK$D.WAT.COUR, RK$WAT.RES,
    RK$D.PARK, RK$L.D.ROAD, RK$L.P.ROAD, RK$MONT.S, RK$SHRUB,
    RK$L.WAT.C, RK$POLIC, RK$URBAN)
```

```
D.WAT.RES D.WAT.COUR     D.PARK    WAT.RES   L.P.ROAD   L.D.ROAD
    1.876     2.315      2.630      2.667      4.323      4.538
  MONT.S   L.WAT.C      SHRUB      POLIC      URBAN     OPEN.L
    4.801     5.196      5.469      5.786      6.822     18.955
   OLIVE     L.SDI    N.PATCH    P.EDGE       MONT
   20.382    24.535     25.856     26.869     36.951
```

Превышение VIF некоторого порогового значения (обычно это критическое значение принимается равным 5) свидетельствует о наличии проблемы мультиколлинеарности для X_j. Удалим из таблицы данных 6 переменных с максимальными VIF.

Построим модель регрессии Пуассона с использованием предикторов, отобранных выше, и выполним процедуру пошаговой селекции исключений с включениями, пока не минимизируем значение AIC-критерия:

```
Y <- RK$TOT.N
M2 <-  glm(Y ~ ., family =  poisson, data =  RK.11)
summary(M2)
M3 <- step(M2)
summary(M3)
```

```
Model 2: Y ~ D.WAT.RES + D.WAT.COUR + WAT.RES + D.PARK +
L.D.ROAD + L.P.ROAD + MONT.S + SHRUB + L.WAT.C + POLIC + URBAN
    Null deviance: 1071.44  on 51  degrees of freedom
Residual deviance:  237.16  on 40  degrees of freedom
AIC: 500.55
Model 3: Y ~ D.WAT.RES + WAT.RES + D.PARK + L.P.ROAD + MONT.S
            + SHRUB + L.WAT.C + POLIC + URBAN
    Null deviance: 1071.44  on 51  degrees of freedom
Residual deviance:  238.98  on 42  degrees of freedom
AIC: 498.37
```

```
anova(M2, M3, test = "Chi")
```

```
Analysis of Deviance Table
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1        40     237.16
2        42     238.98 -2  -1.8141   0.4037
```

После исключения двух переменных в ходе пошаговой процедуры значение AIC-критерия несколько снизилось при статистически незначимом увеличении (с 237.2 до 239) остаточного девианса

$$D = 2(L(\mathrm{y};\,\mathrm{y}) - L(\mathrm{y};\,\mu)) = 2\sum_i (y_i \log\frac{y_i}{\mu_i} - (y_i - \mu_i)).$$

Напомним, что распределение разности девиансов асимптотически приближается к χ^2-распределению с p_1-p_2 степенями свободы

$$D_1 - D_2 \sim \chi^2_{p1\text{-}p2},$$

чем мы воспользовались, чтобы оценить статистическую значимость однородности ошибок двух регрессионных моделей с помощью команды `anova(M2, M3, test = "Chi")`. Та же функция для одной модели показывает, как уменьшается нуль-девианс при включении в модель последовательно каждого очередного предиктора:

```
anova(M3, test = "Chi")
```

```
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                        51    1071.44
D.WAT.RES  1   109.55        50     961.89 < 2.2e-16 ***
WAT.RES    1     0.28        49     961.61 0.5947268
D.PARK     1   595.60        48     366.01 < 2.2e-16 ***
L.P.ROAD   1     7.17        47     358.84 0.0074114 **
```

```
MONT.S      1    16.35         46     342.49 5.272e-05 ***
SHRUB       1    20.79         45     321.71 5.138e-06 ***
L.WAT.C     1    51.96         44     269.75 5.666e-13 ***
POLIC       1    12.80         43     256.95 0.0003475 ***
URBAN       1    17.97         42     238.98 2.240e-05 ***
```

Очевидно, что наибольший вклад в снижение ошибки модели вносят расстояния до лесного массива, реки или другого водного резервуара (`D.PARK, D.WAT.RES, L.WAT.C`), в меньшей мере это относится к наличию кустарников (`SHRUB`), поликультурных насаждений (`POLIC`) или урбанизированных территорий (`URBAN`), а доля девианса, связанного с площадью водного резервуара (`WAT.RES`), не отличается от нуля.

Однако следует задаться вопросом, насколько точно была оценена нами дисперсия аппроксимирующего распределения. Поскольку остаточный девианс имеет χ^2-распределение с $(n - p)$ степенями свободы, то одним из признаков избыточной дисперсии является значение $\phi = D/(n - p)$. Если это отношение приблизительно равно 1, то можно благополучно предположить, что избыточной дисперсии нет. В рассматриваемом примере $\phi = 239/42 = 5.69$ и опасность недооценки дисперсии существует.

Мы можем компенсировать возможную избыточную дисперсию, приняв квази-распределение Пуассона и задав в функции `glm()` параметр `family = quasipoisson`. Для этого типа распределения среднее и дисперсия для Y будет равно $E(Y) = \mu$ и $var(Y) = \varphi \times \mu$:

```
M4 <- glm(Y ~ D.WAT.RES + WAT.RES + D.PARK + L.P.ROAD +
            MONT.S + SHRUB + L.WAT.C + POLIC + URBAN,
            family = quasipoisson, data =  RK.11)
summary(M4)
```

```
(Dispersion parameter for quasipoisson family taken 5.181428)
    Null deviance: 1071.44  on 51  degrees of freedom
Residual deviance:  238.98  on 42  degrees of freedom
AIC: NA
```

Другой формой команды `anova()` является функция `drop1()`, которая поочередно удаляет из модели по одной объясняющей переменной и каждый раз пересчитывает значение остаточного девианса. Значимость различий отдельных моделей оценивается по χ^2-критерию.

```
drop1(M4, test = "Chi")
```

```
Single term deletions
          Df Deviance scaled dev.  Pr(>Chi)
<none>         238.98
D.WAT.RES  1   252.26       2.564 0.1092996
WAT.RES    1   258.82       3.829 0.0503756 .
D.PARK     1   888.32     125.321 < 2.2e-16 ***
L.P.ROAD   1   284.12       8.712 0.0031606 **
MONT.S     1   268.17       5.633 0.0176229 *
SHRUB      1   285.09       8.899 0.0028526 **
L.WAT.C    1   301.24      12.017 0.0005271 ***
POLIC      1   247.95       1.731 0.1882444
URBAN      1   256.95       3.469 0.0625355 .
```

Отметим, что в результате применения квази-пуассоновского распределения при наличии избыточной дисперсии мы получаем следующую коррекцию модели Пуассона:

- значения коэффициентов модели и оценки ее девианса не меняются;
- *p*-величины коэффициентов увеличиваются и некоторые параметры модели оцениваются уже как статистически незначимые;
- величина AIC-критерия рассчитана быть не может.

Заметим также, что ширина доверительного интервала модели на рис. 8.8 должна быть увеличена в $\phi = 7.63$ раз.

Мы получили регрессионную модель, достаточно убедительную с экологической точки зрения и позволяющую ранжировать и интерпретировать степень влияния отдельных предикторов. Однако насколько она хороша для прогнозирования на новых данных? Загрузим пакет `caret` и проведем отбор информативного комплекса переменных методом RFE (Recursive Feature Elimination – см. раздел 4.1):

```
library(caret)
glmFuncs <- lmFuncs
glmFuncs$fit <- function (x, y, first, last, ...) {
  tmp <- as.data.frame(x)
  tmp$y <- y
  glm(y ~ ., data = tmp, family=quasipoisson(link='log'))
}
set.seed(13)
ctrl <- rfeControl(functions = glmFuncs, method = "cv",
           verbose = FALSE, returnResamp = "final")
lmProfileF <- rfe(x = RK.11, y = RK$TOT.N, sizes = 1:10,
       rfeControl = ctrl)
```

```
Recursive feature selection
Outer resampling method: Cross-Validated (10 fold)
Variables  RMSE Rsquared RMSESD RsquaredSD Selected
         1 30.82   0.6908  10.85     0.2590        *
         2 30.88   0.6144  10.83     0.2984
         3 30.92   0.5726  10.81     0.3393
         4 30.98   0.5394  10.80     0.3345
         5 30.98   0.4872  10.80     0.3166
         6 30.96   0.5077  10.80     0.3137
         7 30.94   0.5203  10.81     0.2751
         8 30.91   0.5650  10.79     0.3059
…          …                …
        11 30.91   0.5619  10.78     0.2890
The top 1 variables (out of 1):   D.PARK
```

Проверим справедливость сделанного выбора с помощью функции `train()`:

```
train(TOT.N ~ D.PARK, data=RK, method='glm',
   family = quasipoisson, trControl =
   trainControl(method = "cv"))
```

```
Resampling: Cross-Validated (10 fold)
1 predictor
  RMSE     Rsquared
  15.3978  0.7070051
  15.3979
```

```
train(TOT.N ~ D.WAT.RES + WAT.RES + D.PARK + L.P.ROAD +
    MONT.S + SHRUB + L.WAT.C + POLIC + URBAN, data=RK,
    method='glm', family = quasipoisson,
   trControl = trainControl(method = "cv"))
```

```
Resampling: Cross-Validated (10 fold)
9 predictor
  RMSE      Rsquared
  17.29405  0.6148829
```

Тестирование показало, что модель с 1 предиктором при перекрестной проверке дает меньшую ошибку, чем множественная регрессия на основе 9 параметров.

Выполним теперь построение модели на основе отрицательного биномиального распределения (NB), для чего воспользуемся функцией `glm.nb()` из пакета `MASS`:

```
library(MASS)
M5 <-  glm.nb(Y ~ .,  data =  RK.11)
summary(M5)
```

```
(Dispersion parameter for Negative Binomial(5.5397) family
taken to be 1)
    Null deviance: 214.316  on 51  degrees of freedom
Residual deviance:  54.764  on 40  degrees of freedom
AIC: 396.92
```

```
M6 <- step(M5, trace=0)
summary(M6)
```

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.943e+00  2.435e-01  16.194   <2e-16 ***
D.WAT.RES    4.478e-04  2.217e-04   2.020   0.0434 *
WAT.RES      2.546e-01  1.719e-01   1.482   0.1385
D.PARK      -1.447e-04  1.385e-05 -10.449   <2e-16 ***
L.P.ROAD     2.820e-01  1.419e-01   1.987   0.0469 *
MONT.S       2.024e-01  8.882e-02   2.279   0.0227 *
SHRUB       -6.690e-01  3.074e-01  -2.176   0.0296 *
L.WAT.C      1.644e-01  8.351e-02   1.968   0.0491 *
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
---
Dispersion parameter for Negative Binomial(5.2047)
    Null deviance: 204.391  on 51  degrees of freedom
Residual deviance:  54.054  on 44  degrees of freedom
AIC: 390.57
              Theta:  5.20
          Std. Err.:  1.37
 2 x log-likelihood:  -372.571
```

В дополнение к обычному набору показателей в протокол включены оценка дисперсии и параметр распределения `Theta`, обозначенный выше как θ. В отношении построенных моделей NB можно также выполнить анализ девиансов, как это мы делали выше на примере пуассоновской регрессии. При сравнении моделей, построенных для разных распределений, необходимо использовать величины логарифмов функции правдоподобия или значения AIC-критерия: значение последнего для NB-модели `AIC = 390.57`

существенно меньше, чем для оптимальной модели регрессии Пуассона AIC = 498.37.

Выполним в заключение тестирование в режиме перекрестной проверки моделей отрицательного биномиального распределения с одним и семью предикторами соответственно:

```
train(TOT.N ~ D.PARK, data=RK, method='glm.nb',
   trControl = trainControl(method = "cv"))
```

```
Negative Binomial Generalized Linear Model
1 predictor
Resampling: Cross-Validated (10 fold)
  link      RMSE      Rsquared
  identity       NaN         NaN
  log       31.23333  0.7155445
  sqrt      29.53605  0.7155445
RMSE was used to select optimal model using smallest value.
The final value used for the model was link = sqrt.
```

```
train(TOT.N ~ D.WAT.RES + WAT.RES + D.PARK + L.P.ROAD +
    MONT.S + SHRUB + L.WAT.C, data = RK, method='glm.nb',
   trControl = trainControl(method = "cv"))
```

```
7 predictor
  link      RMSE      Rsquared
  identity       NaN         NaN
  log       30.99842  0.7018376
  sqrt      29.16430  0.7072774
The final value used for the model was link = sqrt.
```

В этом случае обе модели показали весьма близкие результаты и окончательный вывод зависит от решения аналитика. Заметим, что функция `train()` попутно сравнила два вида функции связи (link) и более точным оказалось использование квадратного корня.

8.5. ZIP- и барьерные модели счетных данных

Как обсуждалось выше, реальные данные часто характеризуются существенными отклонениями от теоретического однопараметрического распределения Пуассона: избытком нулевых значений и/или нарушениями предположения о параметре дисперсии Var(Y) ≠ E(Y). Учесть повышенную разреженность данных можно с использованием *модели Пуассона со смешанными параметрами*, учитывающей избыток нулей (Zero Inflated Poisson, ZIP), или специальной двухкомпонентной *модели с измененными нулями* (Zero-altered Poisson, ZAP), которую часто называют "барьерной" моделью (hurdle model). Если наряду с избытком нулей имеет место отчетливая избыточная дисперсия, то можно воспользоваться соответствующими моделями на основе отрицательного биномиального распределения: ZINB (Zero-inflated Negative Binomial) и ZANB (Zero-altered Negative Binomial). Эти четыре модели будут предметом нашего дальнейшего обсуждения (подробности см. в Zeileis et al., 2008; Zuur et al., 2009; Cameron, Trivedi, 2013; Кшнясев, 2010).

Рассмотрим в качестве примера таблицу NMES1988 из пакета AER, содержащий данные о визитах 4406 респондентов старше 66 лет в учреждения бесплатного медицинского обслуживания США в 1988 г. Число посещений visits примем в качестве отклика, а три метрических переменных и три фактора используем в качестве предикторов:

```
library(AER)
data("NMES1988")        # Загружаем данные и отбираем столбцы
nmes <- NMES1988[, c(1, 6:8, 13, 15, 18)]
plot(table(nmes$visits))
sum(nmes$visits < 1)   # наблюдаемое число нулей
```

```
[1] 683
```

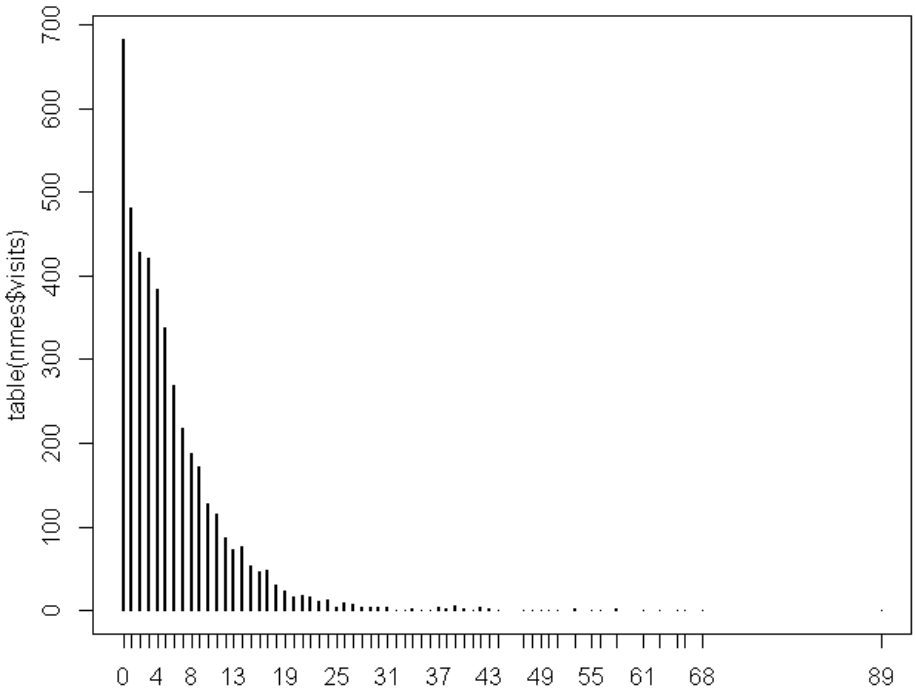


Рис. 8.9. Наблюдаемые частоты посещений врача

Вычислим теоретическую частоту нулевых значений:

```
mod1 <- glm(visits ~ ., data = nmes, family = "poisson")
mu <- predict(mod1, type = "response") # среднее Пуассона
exp <- sum(dpois(x = 0, lambda = mu))  # теоретическая частота
round(exp)                             # нулевых значений
```

```
[1] 47
```

Модели ZIP и ZINB называют моделями со смешанными параметрами (mixture models), поскольку присутствие нулей считается итогом двух различных процессов: биномиального процесса и процесса формирования счетной случайной величины. Здесь биномиальный процесс моделирует вероятность измерения ложно-положительного результата против всех других типов данных (включая истинные нули и конкретные счетные значения). Иными словами, мы делим данные на три воображаемые группы: первая группа содержит только ложные нули – наблюдения с вероятностью появления π_i, вторая группа – истинные нули с вероятностью $(1 - \pi_i)$, и третья

группа – ненулевые наблюдения, которые (вместе со второй группой!) моделируются регрессией Пуассона с учетом независимых переменных. Тогда распределение вероятностей по ZIP-модели определяется следующими выражениями:

$$\Pr(y_i = 0) = \pi_i + (1 - \pi_i) \times e^{-\mu_i}, \qquad \mu_i = e^{\alpha+\beta_1 X_{1i}+\beta_2 X_{2i}+\ldots},$$

$$\Pr(Y_i = y_i \,|\, y_i > 0) = (1 - \pi_i) \times \frac{\mu^{y_i} \times e^{-\mu_i}}{y_i!}, \qquad \pi_i = \frac{e^{\upsilon}}{1+e^{\upsilon}}.$$

Модель ZINB формулируется так же, но значение μ_i подгоняется с использованием отрицательного биномиального распределения.

Построение моделей с избыточными нулями осуществляется с использованием функции `zeroinfl()` из пакета `pscl`, для чего необходимо выполнить следующий код:

```
library(pscl)
M.ZIP <- zeroinfl(visits ~ ., data = nmes,
                  dist = "poisson", link="logit")
summary(M.ZIP)
```

```
Count model coefficients (poisson with log link):
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     1.405812   0.024175  58.152  < 2e-16 ***
hospital        0.159011   0.006060  26.239  < 2e-16 ***
healthpoor      0.253454   0.017705  14.315  < 2e-16 ***
healthexcellent -0.304134   0.031151  -9.763  < 2e-16 ***
chronic         0.101836   0.004721  21.571  < 2e-16 ***
gendermale      -0.062332   0.013054  -4.775 1.80e-06 ***
school          0.019144   0.001873  10.221  < 2e-16 ***
insuranceyes    0.080557   0.017145   4.699 2.62e-06 ***

Zero-inflation model coefficients (binomial with logit link):
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     -0.08102    0.14233  -0.569 0.569219
hospital        -0.30330    0.09158  -3.312 0.000927 ***
healthpoor       0.02166    0.16170   0.134 0.893431
healthexcellent  0.23786    0.14990   1.587 0.112550
chronic         -0.53117    0.04601 -11.545  < 2e-16 ***
gendermale       0.41527    0.08919   4.656 3.22e-06 ***
school          -0.05677    0.01223  -4.640 3.49e-06 ***
insuranceyes    -0.75294    0.10257  -7.341 2.12e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Log-likelihood: -1.613e+04 on 16 Df
```

Построенная модель ZIP включает следующие статистически значимые переменные: время пребывания в больнице (`hospital`), оценка хроничности заболевания (`chronic`), длительности обучения (`school`), пола (`gender`), наличия страховки (`insurance`). Самооценка же здоровья (`health`) оказалась незначимой.

Аналогичным образом можно построить модель с использованием отрицательного биномиального распределения, которая имеет аналогичную структуру и поэтому полный протокол результатов не приводится:

```
M.ZINB <- zeroinfl(visits ~ ., data = nmes,
                   dist = "negbin", link="logit")
summary(M.ZINB)
```

```
Theta = 1.484
Log-likelihood: -1.209e+04 on 17 Df
```

Визуально сравнить степень согласия эмпирических и оцененных моделью частот можно с использованием гистограмм (поскольку для лучшего восприятия графика из частот извлекают квадратный корень, их называют еще "рутограммами" – от англ. "root", корень):

```
library(countreg)
rootogram(M.ZIP)
rootogram(M.ZINB)
```

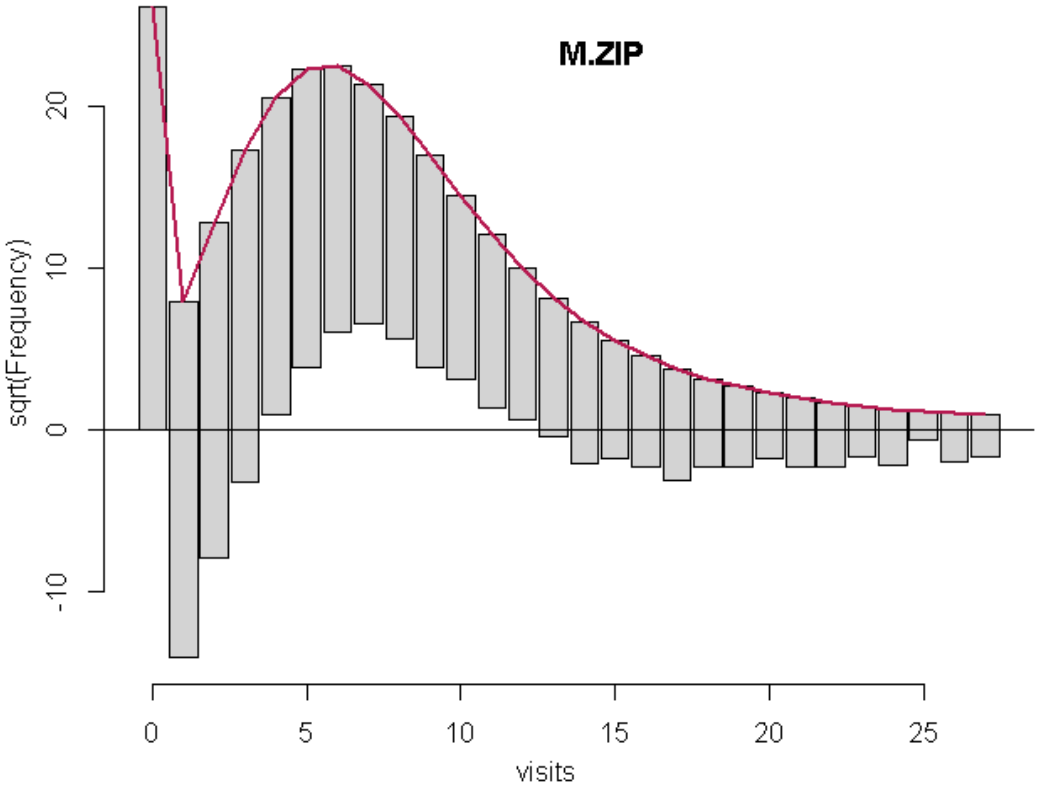


Рис. 8.10. Рутограмма частот для модели ZIP

В результате введения в модель на основе отрицательного биномиального распределения ZINB дополнительного параметра `Theta`, компенсирующего избыточную дисперсию, оценки коэффициентов и их уровней значимостей были уточнены, максимум логарифма функции правдоподобия увеличился, а гистограмма на рис. 8.11 приобретает более плавный характер.

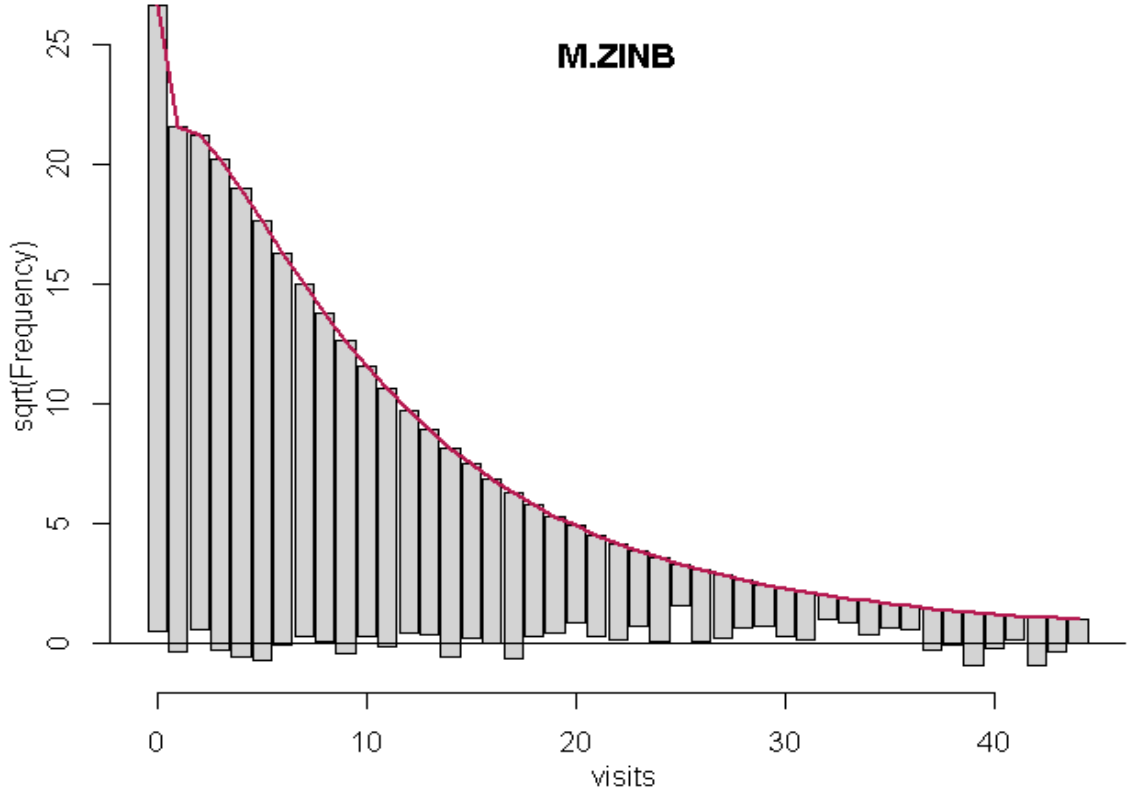


Рис. 8.11. Рутограмма частот для модели ZINB

Заметим, что мы построили простейший вариант моделей, исходя из предположения о том, что параметр π_i равен постоянному значению (т.е. $\upsilon = \text{const}$). Изменяя структуру правой части объекта `formula`, мы можем задать любую зависимость для υ от имеющихся ковариат (Zuur et al., 2009).

Барьерная модель по своему смыслу является уже не смешанной, а двухкомпонентной моделью, т.е. задается как комбинация двух самостоятельных уравнений регрессии: логит-регрессии для моделирования вероятности $\Pr(y_i = 0)$ на основе биномиального распределения, и множественной регрессии для ненулевой средней численности в предположении одно- или двухпараметрического распределения для стохастической компоненты, например:

$$f_{ZAP}(y;\beta,\gamma) = \begin{cases} f_{bin}(y=0;\gamma) & \text{для} y = 0 \\ (1 - f_{bin}(y=0;\gamma)) \times \dfrac{f_{Pois}(y;\beta)}{1 - f_{Pois}(y;\beta)} & \text{для} y > 0 \cdot \end{cases}$$

Выполним построение моделей ZAP и ZANB, для чего воспользуемся функцией `hurdle()` из пакета `pscl`:

```
M.ZAP <-  hurdle(visits ~ ., data = nmes,
    dist = "poisson", zero.dist = "binomial", link = "logit")
summary(M.ZAP)
rootogram(M.ZAP)
```

```
Count model coefficients (truncated poisson with log link):
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)       1.406459   0.024180  58.167  < 2e-16 ***
hospital          0.158967   0.006061  26.228  < 2e-16 ***
healthpoor        0.253521   0.017708  14.317  < 2e-16 ***
healthexcellent -0.303677   0.031150  -9.749  < 2e-16 ***
chronic           0.101720   0.004719  21.557  < 2e-16 ***
gendermale       -0.062247   0.013055  -4.768 1.86e-06 ***
school            0.019078   0.001872  10.194  < 2e-16 ***
insuranceyes      0.080879   0.017139   4.719 2.37e-06 ***
Zero hurdle model coefficients (binomial with logit link):
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)       0.043147   0.139852   0.309 0.757688
hospital          0.312449   0.091437   3.417 0.000633 ***
healthpoor       -0.008716   0.161024  -0.054 0.956833
healthexcellent -0.289570   0.142682  -2.029 0.042409 *
chronic           0.535213   0.045378  11.794  < 2e-16 ***
gendermale       -0.415658   0.087608  -4.745 2.09e-06 ***
school            0.058541   0.011989   4.883 1.05e-06 ***
insuranceyes      0.747120   0.100880   7.406 1.30e-13 ***
---
Log-likelihood: -1.613e+04 on 16 Df
```

```
M.ZANB <-  hurdle (visits ~ ., data = nmes,
                   dist = "negbin", link="logit")
summary(M.ZANB)
rootogram(M.ZANB)
```

```
Theta = 1.484
Log-likelihood: -1.209e+04 on 17 Df
```

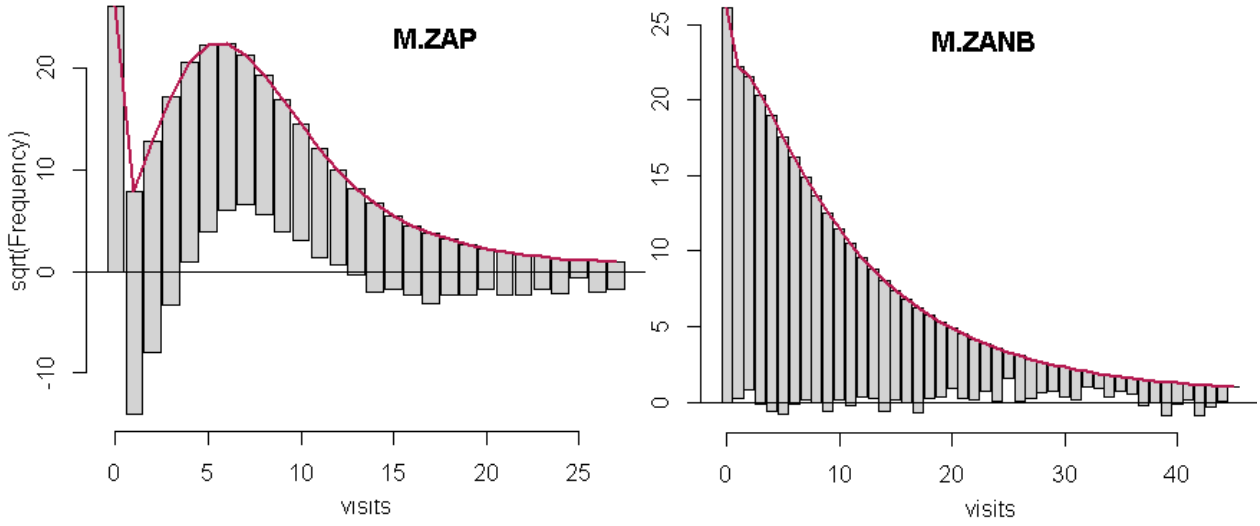


Рис. 8.12. Рутограмма частот моделей ZAP и ZANB

На вопрос о том, какая модель эффективнее, можно дать очевидный ответ (`M.ZANB`), основанный на оптимальных значениях логарифма максимального правдоподобия и информационного критерия:

```
fm <- list("ZIP" = M.ZIP, "ZINB" = M.ZINB,
      "Hurdle-Pois" = M.ZAP,  "Hurdle-NB" = M.ZANB)
rbind(logLik = sapply(fm, function(x) round(logLik(x),
                          digits = 0)),
     AIC = sapply(fm, function(x) round(AIC(x), digits = 0)))
```

```
          ZIP   ZINB Hurdle-Pois Hurdle-NB
logLik -16134 -12091      -16134    -12088
AIC     32300  24215       32301     24210
```

9. МЕТОДЫ МНОГОМЕРНОЙ ОРДИНАЦИИ

9.1. Преобразование данных и вычисление матрицы расстояний

В большинстве случаев статистика оперирует с результатами предметно-ориентированной системы наблюдений, включающей два множества переменных: $\mathbf{X} \rightarrow \mathbf{Y}$,

где **X** – фиксированные или случайно варьируемые факторы (в экологии, например, условия среды или наличие ресурсов), которые потенциально определяют свойства изучаемых объектов **Y** (обилие видов, показатели здоровья и т.д.). С формальной точки зрения результаты мониторинга еще не представляют собой строго определенного информативного пространства, для создания которого необходимо в рамках поставленной задачи задать структуру данных и количественную меру отношений между объектами.

Как упоминалось в главе 1 при обсуждении природы многомерного отклика, современные сложные системы (прежде всего, экономические и экологические) часто включают связанные ансамбли данных, состоящие из некоторого количества однородных компонент S, что определяет ряд особенностей построения моделей на их основе. Если отсортировать список $S=\{x_1, x_2, ..., x_s\}$ по частотам встречаемости элементов каждого класса $N(x_i)$, то получим функцию рангового распределения $\Phi(r) = N(\boldsymbol{x})$, оценивающую вероятности $N(r)/N$ в зависимости от ранга $r(\boldsymbol{x})$. Все устойчивые плотности таких распределений стартуют с больших величин $N(r)$ и круто убывают при увеличении r приблизительно как *гиперболы*, имеющие длинные правосторонние "хвосты". Математики по этому поводу обнаружили шутливую закономерность: «20% жителей выпивают 80% пива».

Разумеется, в таких условиях классическая статистика, основанная на предположениях о нормальности, практически оказывается не вполне полезной. В частности, целью трансформации данных является уже не стабилизация дисперсии и среднего (сам смысл которых становится неясен), а повышение адекватности данных относительно поставленной задачи.

Поясним смысл сказанного на примере. Пусть мы планируем оценить сходство двух водоемов по обилию донных беспозвоночных организмов, взяв соответствующие гидробиологические пробы. Животный мир бентоса состоит из самых различных видов: от крупных моллюсков до мелких нематод, индивидуальный вес которых отличается в тысячи раз. Сравним водоемы, рассчитав евклидово расстояние двух векторов биомасс $\{x_1, x_2, \ldots, x_m\}$ для обнаруженных m видов. Мы увидим, что это расстояние целиком определяется разностью масс 1-2 видов крупнейших моллюсков, а остальные $(m-1)$ видов даже не имело никакого смысла отлавливать. Если мы выполним, например, стандартизацию биомассы каждого вида на диапазоне [0, 1] и снова рассчитаем расстояние Евклида, то оно также не будет способствовать выяснению сути дела, поскольку доминирующие и функционально важные виды будут иметь ту же относительную значимость,

что и случайные, маргинальные или редкие виды, часто играющие ничтожную роль в экосистеме. Читатель, далекий от проблем беспозвоночных, может мысленно трансформировать этот пример на сравнение уровней материальной обеспеченности жителей двух регионов, включая олигархов и дворников, или любой иной вариант.

П. Лежандр с соавторами (Legendre, Gallagher 2001; Legendre, Legendre, 2012) разработали общие правила многомерного анализа данных и построения хорошо интерпретируемых ординационных диаграмм. Основные способы трансформации и стандартизации данных, рекомендуемые ими (применительно к подсчету численности видов в экологических исследованиях), представлены функцией `decostand()` в пакете `vegan`:

```
decostand(x, method, MARGIN),
```

где `MARGIN = 1`, если операция применяется к строкам таблицы `x`, и 2 – если к столбцам (обычно именно это значение принимают по умолчанию).

Параметр `method` может принимать следующие значения:

`normalize`	Сумма квадратов значений по строкам делается равной 1
`total`	Деление на суммы по строкам
`hellinger`	Корень квадратный из значений по методу `total`
`max`	Деление на максимумы по столбцам
`freq`	Деление на максимумы по столбцам и умножение на число ненулевых компонент
`chi.square`	См. формулу (9.1) ниже
`log`	Логарифмическая трансформация (не требует добавления 1)
`range`	Данные по столбцам стандартизуются на диапазоне [0, 1]
`standardize`	Обычная стандартизация `x` к нулевому среднему и единичной дисперсии; см. `scale(x, center = TRUE, scale = TRUE)`
`pa`	Приведение `x` к бинарной шкале (0/1)

В представленной выше ситуации с бентосом, рекомендуется либо простейшая логарифмическая трансформация, либо преобразование, приводящее к χ^2-дистанции, которая является, видимо, наиболее разумным компромиссом при учете как роли ведущих компонент, так и вклада длинного правого "хвоста". Такое преобразование имеет вид

$$x_{ij}^{'} = \sqrt{x_{++}} \frac{x_{ij}}{x_{i+}\sqrt{x_{+j}}}, \qquad (9.1)$$

где x_{i+} – сумма по строкам, x_{+j} – сумма по столбцам, x_{++} – общая сумма элементов таблицы **x**.

Вторым важным этапом является спецификация метрики отношений между объектами. В частном случае при применении статистических методов информативное пространство может интерпретироваться как вероятностное: тогда пара векторов действительных чисел $\{x_1, x_2, \ldots, x_m\}$ и $\{y_1, y_2, \ldots, y_m\}$, описывающих произвольные объекты x и y, будут трактоваться как выборочные реализации m-мерной случайной величины. В этом случае в качестве мер сходства между объектами могут выступать оценки ковариации

$\mathrm{cov}(x, y) = \sum_{i=1}^{m}(x_i - m_x)(y_i - m_y)$, коэффициент корреляции $r_{xy} = \mathrm{cov}(x, y)/\sigma_x\sigma_y$ или произвольное ковариационное отношение $K = [\mathrm{cov}(x, y) - \mathrm{cov}_{\min}]/(\mathrm{cov}_{\max} - \mathrm{cov}_{\min})$, где m – математическое ожидание, σ – стандартное отклонение, $\mathrm{cov}_{\min}$ и $\mathrm{cov}_{\max}$ – экстремальные значения ковариации для теоретической ("эталонной") выборки (Воробейчик, 1993).

В общем случае использование вероятностных представлений совершенно не обязательно. Часто пространство измеряемых переменных рассматривают как *метрическое* пространство, расстояния в котором определяются некоторой функцией ρ, обладающей нехитрыми свойствами: а) тождества $\rho(\mathbf{x}, \mathbf{y}) = 0$ при $\mathbf{x} = \mathbf{y}$, б) симметрии $\rho(\mathbf{x}, \mathbf{y}) = \rho(\mathbf{y}, \mathbf{x})$ и в) правила треугольника $\rho(\mathbf{x}, \mathbf{y}) + \rho(\mathbf{y}, \mathbf{z}) \geq \rho(\mathbf{x}, \mathbf{z})$. Конкретной дефиницией функции ρ может быть, например, обобщенная мера Минковского (см. раздел 3.4), наиболее популярными реализациями которой являются манхэттенская ("`manhattan`") или евклидова ("`euclidean`") дистанции, а также расстояние Хемминга ("`binary`"), равное числу совпавших единиц для двух бинарных кодов. Эти опции являются основными для базовой функции R `dist(x, method = "euclidean")`, вычисляющей матрицу дистанций между всеми парами объектов, которые представлены строками таблицы `x`.

Более широкими возможностями обладает функция из пакета `vegan`

`vegdist(x, method = "bray", binary = FALSE)`,

где параметр `method` охватывает значительную часть "изобретений" экологов в области мер сходства/расстояния: `"manhattan"`, `"euclidean"`, `"canberra"`,`"bray"`, `"kulczynski"`, `"jaccard"`, `"gower"`, `"altGower"`, `"morisita"`, `"horn"`, `"mountford"`, `"raup"`, `"binomial"`, `"chao"` и `"cao"`. По умолчанию используется мера Брея–Кёртиса[5]:

$$M_{xy} = 1 - 2\sum_{i=1}^{m_{ab}} \min[x_i, y_i]/(\sum_{i=1}^{m} x_i + \sum_{i=1}^{m} y_i),$$

которая в точности соответствует нормированному манхэттенскому расстоянию. Если использовать `binary=TRUE`, то эта метрика идентична популярному расстоянию Съеренсена для альтернативных признаков (или нормированному расстоянию Хемминга).

Чтобы обеспечить достижение "истинности" различий, предлагаются (Boyce, Ellison, 2001) функции расстояния с различными более сложными нормировками, например, индексы Мориситы-Хорна (`"morisita", "horn"`):

$$M_{xy} = \frac{\sum_{i=1}^{m}[(x_i + y_i)\log(x_i + y_i)] - \sum_{i=1}^{m} x_i \log x_i - \sum_{i=1}^{m} y_i \log y_i}{[(N_x + N_y)\log(N_x + N_y)] - N_x \log N_x - N_y \log N_y}.$$

В ряде обзоров делаются попытки оценить, какие меры "завышают" или "занижают" сходство между объектами и каким коэффициентам следует отдать предпочтение в работе. По этому поводу можно заметить, что

---

[5] Другие его названия: индекс Ренконена, процентное подобие, коэффициент общности, индекс Штейнгауза, количественная мера сходства Чекановского и т.д.

сходство/расстояние между объектами является типичным искусственно сконструированным (латентным) понятием, поэтому с теоретических позиций наилучшую формулу его количественного выражения нельзя найти без связи с каким-то внешним критерием оптимизации. Например, одним из критериев адекватности метрик является устойчивость последовательностей агрегирования объектов в более крупные таксоны, иерархические деревья и проч., которая часто оказывается специфичной для каждого набора данных.

Отметим также, что некоторые из вышеперечисленных мер (например, коэффициенты Жаккара и Съеренсена) являются *коэквивалентными*, т.е. они порождают одну и ту же предупорядоченность анализируемых объектов. Более сильное влияние на характер результатов оказывают различия в функциональной форме разностей *x* и *y*, такие как абсолютная, квадратичная или логарифмическая.

9.2. Непараметрический дисперсионный анализ матриц дистанций

Результаты наблюдений, представленные в виде многомерной таблицы объемом $m \times p$, обычно можно разделить по строкам на r групп, (например, в соответствии со списком регионов, где оценивается покупательский спрос, или шириной водотока, где выполняются гидробиологические пробы). Тогда, в соответствии с моделью дисперсионного анализа, общую изменчивость каждого k-го фиксированного показателя x_k (k = 1, 2, …, p) можно разложить на компоненты:

$$\mathrm{Var}\, x_k = \mathrm{Var}\, \tau + \mathrm{Var}\, \varepsilon,$$

где Var τ – вариация, обусловленная влиянием группирующего фактора, Var ε – изменчивость, связанная с воздействием неконтролируемых (в том числе, случайных) факторов.

Однако, если количество p одновременно варьируемых показателей велико (например, номенклатура из 300-400 продаваемых товаров), то дисперсионный анализ не дает возможности оценить совокупную изменчивость всего изучаемого многомерного комплекса объектов под воздействием группирующего фактора. Один из приемов избежать "проклятия размерности" – выполнить дисперсионный анализ симметричной матрицы **D** размерностью $m \times m$, элементами d_{ij} которой являются коэффициенты расстояния между каждой парой изучаемых объектов i и j.

Метод, известный как *непараметрический многофакторный дисперсионный анализ* (npMANOVA), осуществляет разложение многомерной изменчивости, заключенной в матрице расстояний, в соответствии с уровнями влияния изучаемых факторов. В первом приближении, как это показано на рис. 9.1, можно рассчитать общую SS_T и внутригрупповую SS_W суммы квадратов d_{ij}, после чего оценить их дисперсионное отношение:

$$SS_T = \frac{1}{m}\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} d_{ij}\ ;\ \ SS_W = \frac{1}{r}\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} d_{ij}\omega_{ij}\ ;\ F = \frac{SS_W\,(m-r)}{SS_T\,(r-1)}, \qquad (9.2)$$

где ω_{ij} равны 1, если объекты i и j принадлежат одной группе, и 0 в противном случае (Anderson, 2001).

Рис. 9.1. Схема группировки матрицы расстояний

Более корректный способ вычисления тестовой статистики F использует ряд конструкций линейной модели ANOVA (McArdle, Anderson, 2001). При применении этого подхода на основе исходной матрицы расстояний **D** с использованием преобразований Гувера рассчитывается матрица **G**, описывающая "центроиды", т. е. центры распределения значений d_{ij} в пределах каждой группы. Другая проекционная матрица **H** (или "шляпа", hat) вычисляется с использованием вектора ортогональных контрастов, связанных с уровнями группировочных факторов. Тогда тестовая статистика для проверки нулевой гипотезы об отсутствии различий между r группами, или псевдо-F-критерий, находят как

$$F = \frac{tr(\mathbf{HGH})(m-r)}{tr[(\mathbf{I}-\mathbf{H})\mathbf{G}(\mathbf{I}-\mathbf{H})](r-1)}, \qquad (9.3)$$

где tr – сумма диагональных элементов матрицы, **I** – единичная матрица. При выполнении требований к конфигурации данных и использовании евклидова расстояния обе формулы (9.2) и (9.3) становятся эквивалентными и воспроизводят традиционное дисперсионное отношение Фишера.

Распределение многомерной псевдо-F-статистики (9.2-9.3) при справедливости нулевой гипотезы теоретически неизвестно, и поэтому для его аппроксимации используется рандомизационная процедура. Ее практическая стратегия состоит в том, чтобы получить большое подмножество значений F^*, извлеченных случайно, независимо и с равной вероятностью из нуль-распределения $\boldsymbol{F^*}$, полученного в результате многократного хаотичного перемешивания наблюдений относительно выделенных групп. P-значение, соответствующее верной нулевой гипотезе, вычисляется как доля значений F^*, которые больше или равны значению F, найденному на реальных данных.

Непараметрический дисперсионный анализ Андерсона реализован в пакете `TraMineR` (см. раздел 5.5) или пакете `vegan` следующей функцией:

```
adonis(formula, data, permutations = 999, method = "bray") ,
```

где объект `formula` чаще всего имеет форму `Y ~ A + B*C`, в которой `Y` – таблица исходных данных, которая пересчитывается в матрицу дистанций по

выражению method, а A, B и C – независимые ("объясняющие") факторы или непрерывные переменные. Если факторные пространства являются непересекающимися по какому-то показателю, то, указав его в качестве параметра strata, можно выполнить гнездовой дисперсионный анализ (nested ANOVA).

Рассмотрим проведение анализа на примере таблицы dune, которая содержит данные по 20 пробным участкам, заложенным в дюнах одного датского острова, на которых было обнаружено 30 видов луговой травянистой растительности. Обилие каждого вида оценивалось в баллах проективного покрытия от 0 до 7. Параллельно для каждого участка измерялась толщина гумусового слоя почвы A1 и оценивалась одна из 4-х категорий Management агротехнических мероприятий по улучшению травостоя (таблица dune.env):

```
library(vegan)
data(dune)
data(dune.env)
results <- adonis(dune ~ Management*A1, data=dune.env,
                  permutations=499)
```

```
Terms added sequentially (first to last)
             Df SumsOfSqs MeanSqs F.Model      R2 Pr(>F)
Management    3    1.4686 0.48953  3.2629 0.34161  0.002 **
A1            1    0.4409 0.44089  2.9387 0.10256  0.018 *
Management:A1 3    0.5892 0.19639  1.3090 0.13705  0.212
Residuals    12    1.8004 0.15003         0.41878
Total        19    4.2990                 1.00000
---
Signif. codes:  0 C***T 0.001 C**T 0.01 C*T 0.05 C.T 0.1 C T 1
```

Традиционная таблица дисперсионного анализа полученных результатов показывает статистическую значимость обоих основных факторов и незначимость эффекта их парного взаимодействия. Например, *p*-значение, равное 0.018, означает, что только в 9 случаях из 500 имитируемое значение *F**, полученное при справедливости нулевой гипотезы, превысило значение *F*, вычисленное по (9.3) для эмпирических данных.

Для однофакторного дисперсионного анализа и оценки однородности многомерной вариации в группах наблюдений удобно использовать также другую функцию vegan – betadisper(), на вход которой подается произвольная матрица дистанций d. Выполним расчеты с применением этой функции на основе группировочного фактора group = Management:

```
#  Предварительно рассчитываем матрицу расстояний Брея-Кёртиса
D <- vegdist(dune)
mod <- betadisper(D, dune.env$Management,  type = "centroid")
plot(mod, hull= FALSE, main="") ; legend("topright",
            levels(dune.env$Management), pch=1:4, col=1:4)
# Выполнение рандомизационного теста и парных сравнений
permutest(mod, pairwise = TRUE)
```

```
Permutation test for homogeneity of multivariate dispersions
Response: Distances
          Df  Sum Sq  Mean Sq      F N.Perm Pr(>F)
Groups     3 0.12457 0.041525 3.8687    999  0.025 *
```

```
Residuals 16 0.17174 0.010733
Pairwise comparisons: (Observed p-value below diagonal,
                       permuted p-value above diagonal)
         BF       HF       NM       SF
BF              0.3660   0.0060   0.2220
HF    0.3698             0.0060   0.3020
NM    0.0047   0.0040             0.2620
SF    0.2094   0.2982   0.2392
```

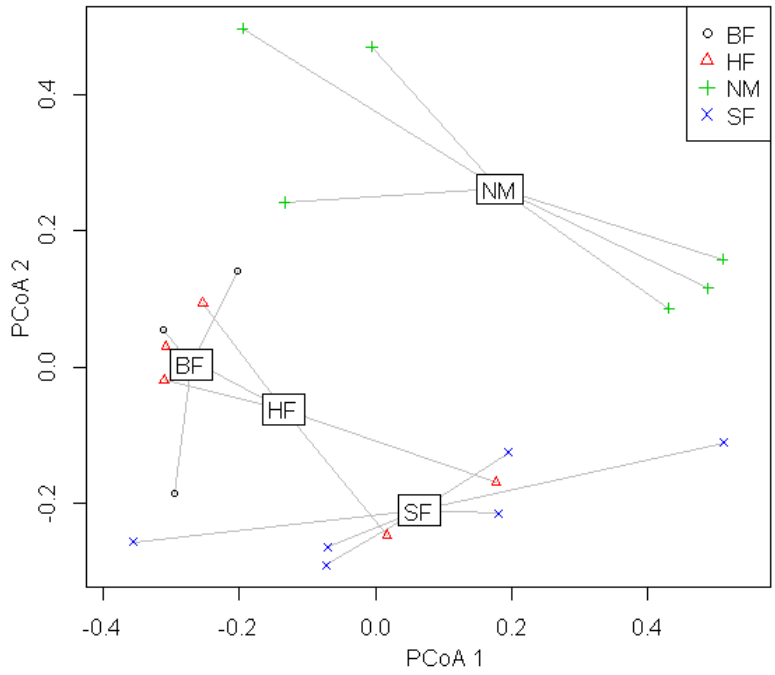


Рис. 9.2. Центроиды групп в пространстве двух главных координат

В целом перестановочный тест показал значимое (p = 0.025) влияние агротехнических мероприятий на вариацию видов растительности. Однако выполненные парные сравнения выявили значимые отличия только для метода NM (оставить луга в естественном состоянии) от биологического BF или рекреационного HF вмешательства. Относительные расстояния между центроидами групп можно приблизительно оценить по ординационной диаграмме (рис. 9.2).

Для выполнения множественных сравнений наиболее распространённым и рекомендуемым в литературе является тест Тьюки, использующий критерий "подлинной" значимости (Honestly Significant Difference, HSD). HSD оценивает наименьшую величину разности математических ожиданий в группах, которую можно считать значимой. Тест Тьюки-HSD реализуется в R для модели `betadisper` с использованием базовой функции `TukeyHSD()`. На рис. 9.3 представлены рассчитанные этой функцией одновременные 95%-ные доверительные интервалы разности между каждой парой групп. Поскольку этот интервал не включает 0 только для одной сравниваемой пары (NM–BF), то значимые различия в видовом составе растительности имеются лишь между этими группами.

```
#   Тест Тьюки HSD и график доверительных интервалов
(mod.HSD <- TukeyHSD(mod)) ;  plot(mod.HSD)
```

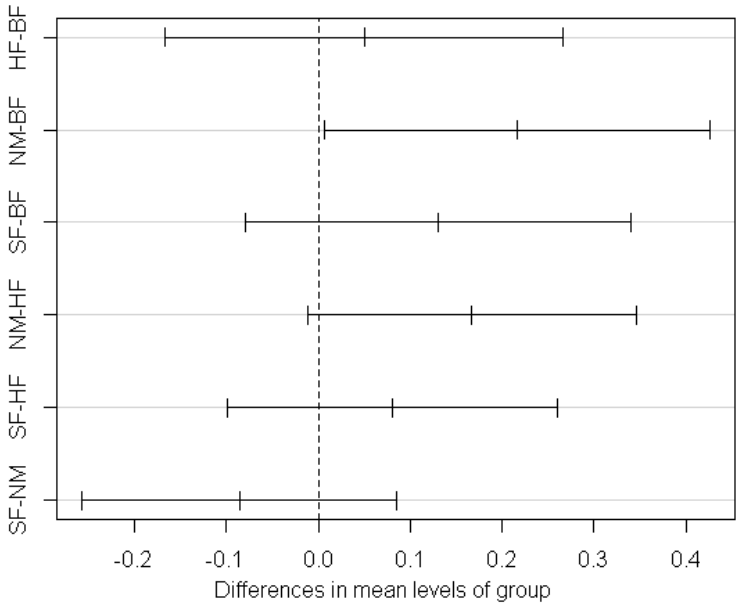


Рис. 9.3. Множественные сравнения методов улучшения травостоя с использованием критерия HSD Тьюки

9.3. Методы ординации объектов и переменных: построение и сравнение диаграмм

Ординация в узком смысле представляет собой нахождение таких координатных осей на плоскости, относительно которых можно выполнить оптимальное проецирование многомерных анализируемых объектов. Ординация связана с поиском наиболее "естественных" системных закономерностей и особенно эффективна при наличии комплекса взаимно скоррелированных влияющих факторов, «которые являются кошмаром, например, для множественной регрессии». К настоящему времени в среде R разработано и реализовано значительное количество алгоритмов сжатия информационного пространства, в частности:

- метод главных компонент PCA (см. раздел 2.4), оперирующий с ковариационной (корреляционной) матрицей;
- метод главных координат PCoA и неметрическое многомерное шкалирование NMDS (nonmetric multidimensional scaling), выполняющие последовательную процедуру преобразования любой матрицы дистанций;
- анализ соответствий или корреспондентный анализ CA (correspondence analysis), основанный на итерационной процедуре встречного усреднения взвешивающих коэффициентов для объектов и переменных (Джогман и др., 1999);
- функции многомерного факторного анализа, включая версии для иерархических и смешанных данных, представленные в пакете `FactoMineR`.

Метод главных координат PCoA, или многомерное шкалирование (MDS, multidimensional scaling), во многом похож на PCA, но вместо корреляционной матрицы выполняет вычисление собственных значений и собственных векторов произвольной квадратной симметричной матрицы

расстояний **D**. Так можно компенсировать некоторые отклонения от предпосылок в отношении статистического распределения данных, принятых для корреляционного анализа, но одновременно возникает проблема выбора подходящей метрики дистанции. На странице gastonsanchez.com описана работа с 7 функциями различных пакетов R, в которых реализован метод PCoA, но обычно используют базовую функцию `cmdscale()`.

В качестве примера рассмотрим достаточно представительный набор геоботанических данных `bryceveg` из пакета `labdsv`, в котором 165 видов травянистой растительности было обнаружено на 160 пробных площадках, заложенных на территории Брайс-Каньо́н (Bryce Canyon, штат Юта, США):

```
library(labdsv)
data(bryceveg) ;  data(brycesite)
#  Рассчитываем три матрицы расстояний по разным формулам
dis.mht <- dist(bryceveg,"manhattan")
dis.euc <- dist(bryceveg,'euclidean')
dis.bin <- dist(bryceveg,"binary")
#  Создаем объекты cmdscale
mht.pco <- cmdscale(dis.mht, k=10,eig=TRUE)
euc.pco <- cmdscale(dis.euc, k=10,eig=TRUE)
bin.pco <- cmdscale(dis.bin, k=10,eig=TRUE)
# График изменения относительных собственных чисел
plot(1:10, (mht.pco$eig/sum(mht.pco$eig))[1:10],
     type="b",xlab="Число координат",
     ylab="Относительные собственные значения")
lines((euc.pco$eig/sum(euc.pco$eig))[1:10],type="b",col=2)
lines((bin.pco$eig/sum(bin.pco$eig))[1:10],type="b",col=3)
legend ("topright",c("Манхэттен","Евклид", "Хемминг"),
      lwd=1, col=1:3)
```

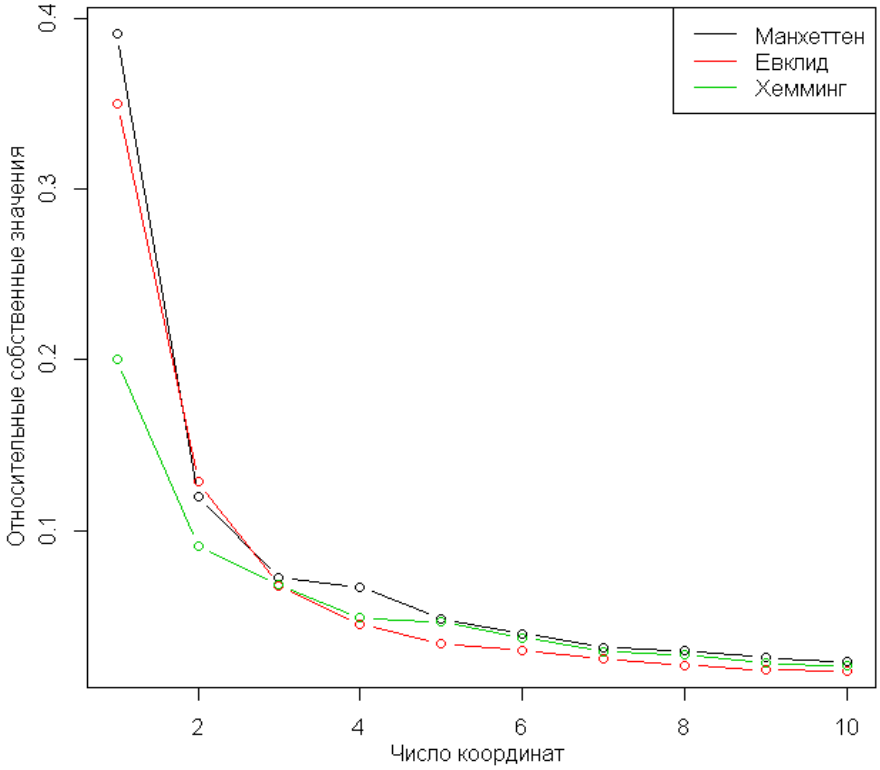


Рис. 9.4. Уменьшение собственных значений матриц с увеличением числа координат

Возникает естественный вопрос, какая из трех наиболее характерных метрик расстояния приводит к наилучшей ординации? Если ориентироваться на величину первых двух собственных значений матрицы **D** (рис. 9.4), которые соответствуют доле объясненной дисперсии видовой структуры участков, то наименее адекватной оказывается ординация, полученная при использовании расстояния Хемминга, т.е. когда учитывалось только наличие или отсутствие видов растений. Второй способ – оценить коэффициент корреляции двух матриц расстояния: исходной и рассчитанной на основе первых двух главных координат, что соответствует доле информации, сохраненной после сжатия 165-мерного пространства до двумерного. Наконец, всегда полезно просто посмотреть на характер распределения точек на выполненной проекции и прислушаться к голосу собственной интуиции:

```
plot(euc.pco$points[,1:2], col = 4, pch=17, xlab = "PCO1",
        ylab = "PCO2", main="Расстояние Евклида")
plot(0-bin.pco$points[,1:2], col = 3,pch=17, xlab = "PCO1",
        ylab = "PCO2", main="Расстояние Хемминга")
CCor <-c(cor(dis.euc, dist(euc.pco$points[,1:2],'euclidean')),
cor(dis.mht, dist(mht.pco$points[,1:2],'manhattan')))
names(CCor) <- c("Евклидово","Манхэттенское")
```

```
Евклидово Манхэттенское
0.8171305     0.8256334
```

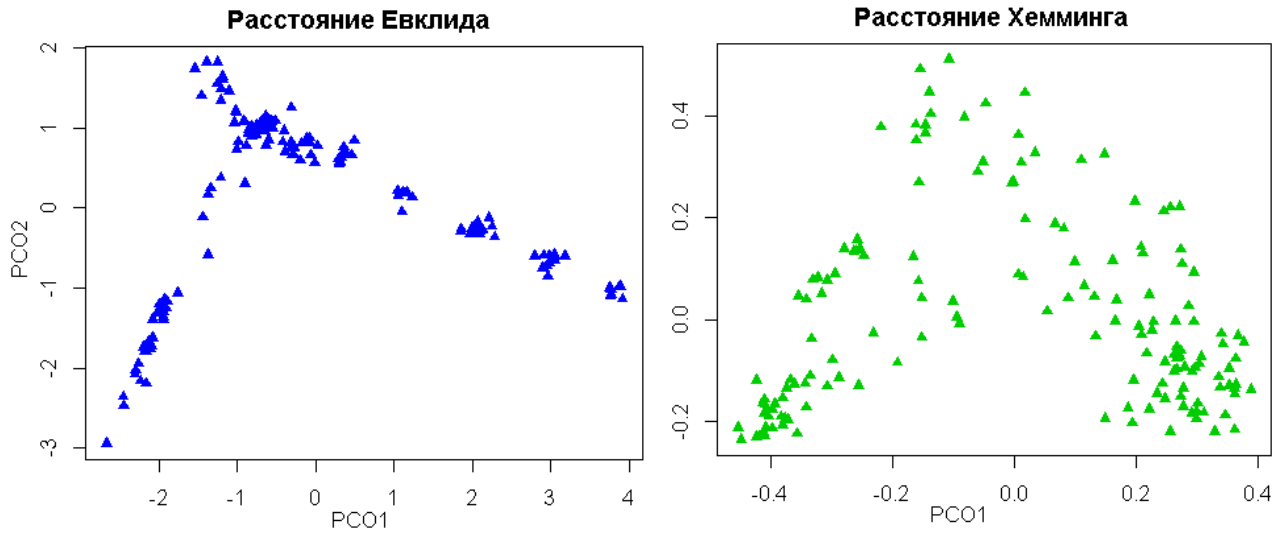


Рис. 9.5. Распределение точек ординации PCoA для разных метрик дистанции

Важной косвенной оценкой, насколько полученная ординация адекватна задаче исследования, является ее тесная взаимосвязь с ведущими внешними факторами. Функция `ordtest()` из пакета `labdsv` выполняет оценку *p* значимости такой связи с использованием рандомизационного теста, рассчитывая сумму квадратов разностей двух матриц расстояния (исходной и построенной на основе фактора). Другой подход основан на аппроксимации двумерной сглаживающей поверхностью значений предиктора в точках, определенных главными координатами. Подробности изложены в курсе лекций проф. Д. Робертса студентам университета штата Монтана (http://ecology.msu.montana.edu). Ниже используется функция `surf.pco()`,

которая выполняет сглаживание сплайнами и визуализацию на диаграмме линий изоном фактора. Ее скрипт приведен в лекции № 8 и скопирован нами в файл surf_pco.r:

```
data(brycesite) ; H <- brycesite$elev
mht.pco <- pco(dis.mht, k = 2) # аналог функции cmdscale()
source("surf_pco.r")
plot(mht.pco, pch=16, col="red")
surf.pco(mht.pco, H, col="blue")
ordtest(mht.pco,H)$p
```

```
var ~ s(x, y)
Estimated degrees of freedom:
16.077  total = 17.07685
GCV score: 189195.3
D^2 =  0.5999
[1] 0.001
```

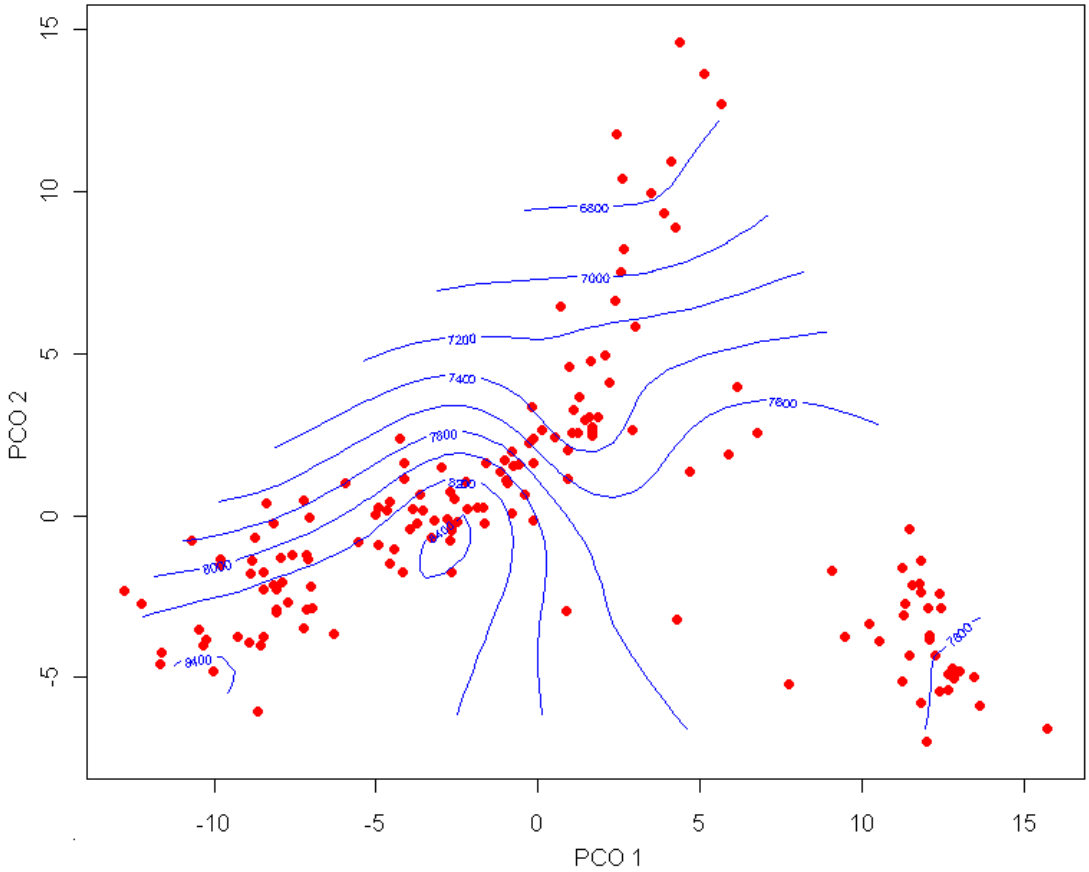


Рис. 9.6. Ординация геоботанических данных с использованием манхэттенского расстояния и ее связь с высотой пробных площадок

Для построения сглаживающей поверхности высоты участков над уровнем моря по осям двух главных координат использовалась функция gam() из пакета mgcv, выполняющая аппроксимацию сплайном s(x, y). Качество подгонки может быть оценено с помощью псевдо-коэффициента детерминации, который для произвольной GAM-модели m рассчитывается как

D2 = (m$null.deviance - m$deviance)/m$null.deviance.

Если построить различные варианты ординации с использованием разных методов или метрик, то наилучшей будет считаться та, которая доставляет

максимум D2. При этом, разумеется, необходимо выбрать для тестирования ведущий внешний фактор. Например, если вместо высоты над уровнем моря использовать угол наклона площадки к горизонту brycesite$av, то его статистическая связь с изменчивостью видового состава растительности будет незначимой p=0.544, а величина D2 будет мала D^2 = 0.06873.

Для детального сравнения двух диаграмм их необходимо привести к единой *форме*, т.е. к стандартной конфигурации, которая будет независима от положения, масштаба и поворота координатных осей. Один из способов сравнения графиков заключается в "прокрустовом" преобразовании геометрии точек. Для этого проводится ряд последовательных итераций подгонки масштаба осей:

- сравниваемые точки помещаются внутри единичного круга |δ| = 1;
- вся структура вращается относительно центра координат;
- находится минимум суммы квадратов расстояний между двумя ординациями $m^2 = [\mathbf{x}_1 - T(\mathbf{x}_2)]^2 \rightarrow \min$ или максимум прокрустовой корреляции $r = \sqrt{1 - m^2}$.

Используя функции из пакета vegan, выполним сравнение расположения площадок на двух диаграммах, построенных методами главных координат (mht.pco на рис. 9.6) и анализом соответствий CA:

```
library(vegan)
ca <- cca(bryceveg)
pro.comp <- procrustes(mht.pco, ca)
plot(pro.comp)
```

```
Procrustes sum of squares: 1.164e+04
```

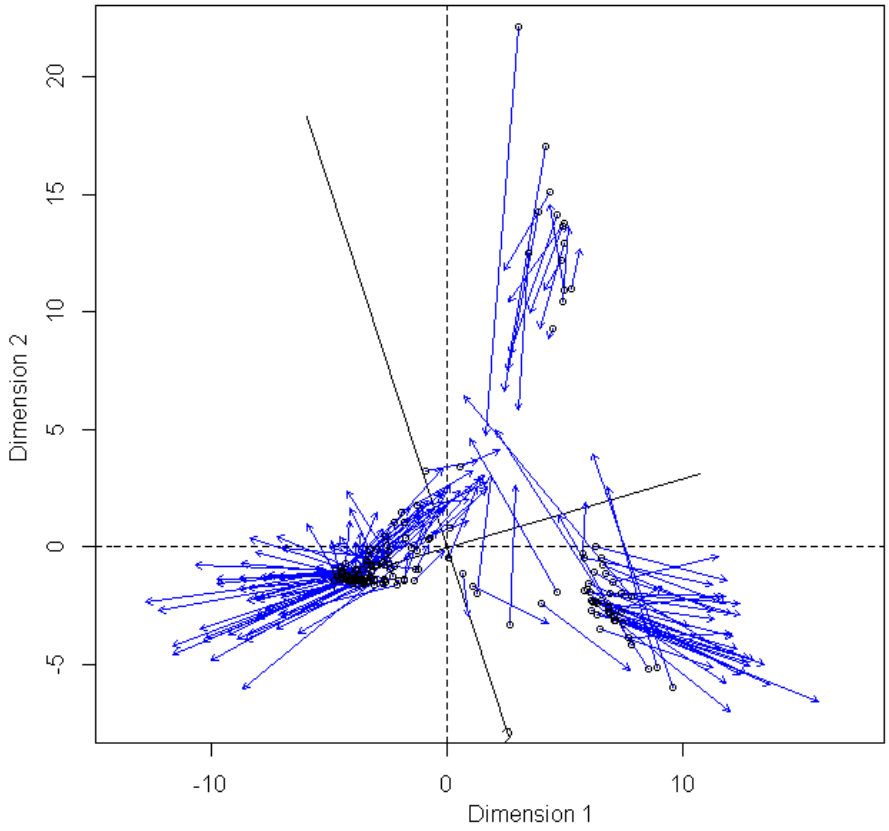


Рис. 9.7. Сравнение двух диаграмм после прокрустового преобразования

Использование линейных преобразований в ходе проецирования на плоскость дает возможность не только построить упорядоченную диаграмму *объектов*, но и выполнить ординацию *переменных*, использованных в анализе. В случае главных компонент взаимный пересчет координат легко сделать по формулам: $F_s(i) = 1/\sqrt{\lambda_s}\sum_k x_{ik} G_s(k)$; $G_s(k) = 1/\sqrt{\lambda_s}\sum_i x_{ik} F_s(i)$,

где $F_s(i)$ и $G_s(k)$ – координаты i-го объекта и k-й переменной на s-й оси компоненты, а λ_s – собственное значение, связанное с s-й осью.

В случае анализа соответствий такая диаграмма для 165 видов на территории Брайс-Каньо́н выглядит как на рис. 9.8:

```
plot(ca,  type="n")   # Отрисовка «пустой» диаграммы
ordilabel(ca, dis = "sp", cex=0.7, font=3 ,add = TRUE)
```

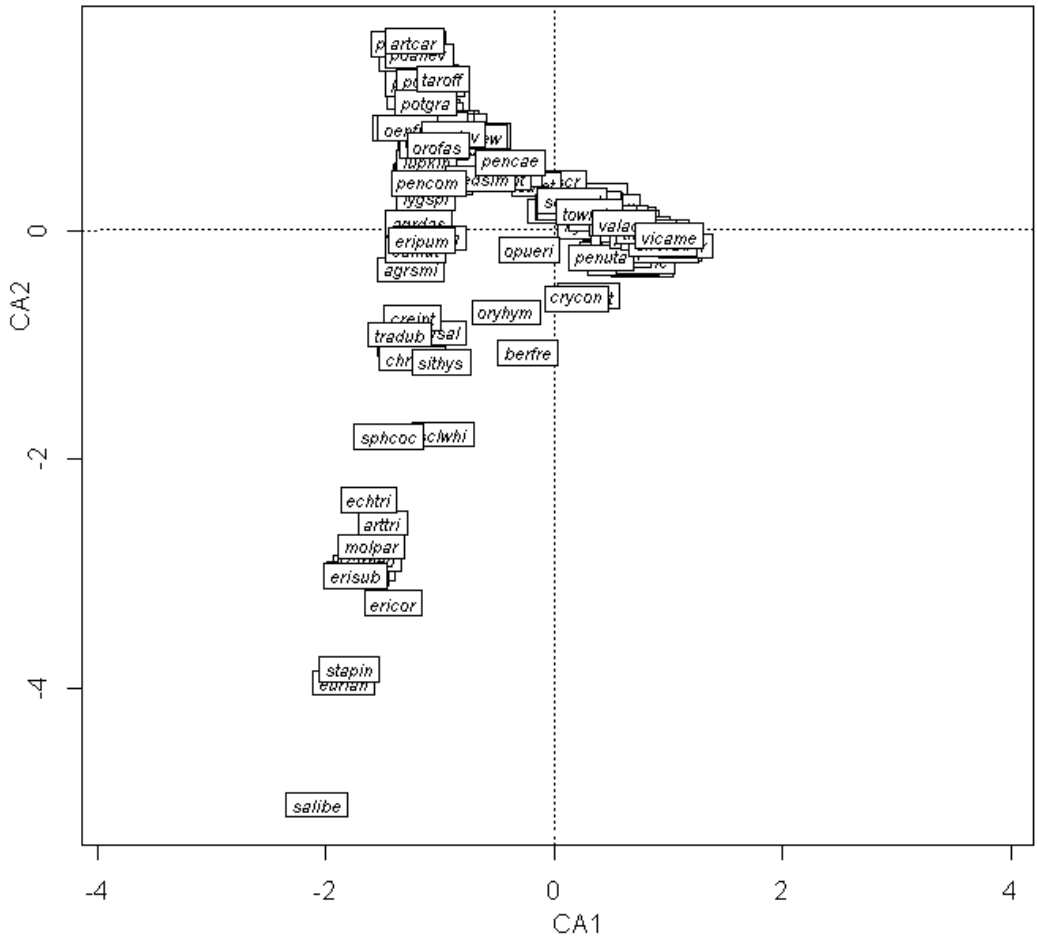


Рис. 9.8. Ординационная диаграмма видов растительности, полученная методом анализа соответствий

Если вид встречается только в одном месте, то его положение на диаграмме совпадает с точкой, обозначающей соответствующую пробную площадку. В противном случае местоположение вида в системе осей CA_1-CA_2 определяется средневзвешенными координатами нескольких его возможных местообитаний. По-видимому, эта точка определяет экологический оптимум вида, где его появление наиболее вероятно, либо обилие максимально.

Функции `ordipointlabel()` и `ordilabel()` из пакета `vegan` дают возможность расположить на графике метки данных наилучшим образом и задать последовательность их визуализации: в верхнем слое диаграммы на рис. 9.8 показаны экологически важные виды с наибольшей суммой обилия по столбцам.

Важное место в современном статистическом анализе занимает построение внешне элегантных и информационно насыщенных графиков. Ведущая роль при этом отводится системе визуализации `ggplot2` (Мастицкий, 2016). В предыдущих главах мы приводили ординационные диаграммы, полученные с использованием функций `ggplot` (см. рис. 2.10, 2.11, 2.13, 7.3, 7.7), когда обсуждали метод главных компонент и дискриминантный анализ. Но читатель, вероятно, смог заметить, что построение графиков `ggplot2` – непростой, хотя и увлекательный процесс.

Возможность построения элегантных графиков "одной строкой" предоставляется специализированными пакетами, использующими "в темную" инструментарий графической системы `ggplot2`. К таким относится пакет `factoextra`, предназначенный для визуализации результатов многомерного и кластерного анализа (см. также главу 10). Подробный рассказ об этом пакете см. на сайте http://www.sthda.com (STHDA, Statistical tools for high-throughput data analysis).

Функция `viz_pca()` из пакета `factoextra` осуществляет построение ординационных диаграмм для объектов и/или переменных (отдельно или совмещенно) на основе PCA-объектов, полученных функциями `prcomp()`, `princomp()`, `PCA()` из пакета `FactoMineR` и некоторых других.

Набор данных `decathlon` из пакета `FactoMineR` (Le et al., 2008) содержит результаты 41 ведущего десятиборца, показанные в 2004 г. на олимпийских играх и турнире "ДекаСтар". Выполним его PCA-ординацию:

```
library(FactoMineR)
library("factoextra")
data("decathlon")
colnames(decathlon) <- c("100м","Длина.прыжок","Ядро",
   "Высота.прыжок","400м","110м.барьер","Диск","Шест.прыжок",
   "Копье","1500м", "Место","Очки", "Соревнование" )
res.pca <- PCA(decathlon[, 1:10], scale.unit = TRUE, ncp = 5,
               graph = FALSE)
head(res.pca$eig)
```

```
       eigenvalue   percent of var   cumulative percent of var
comp 1  3.2719055        32.719055                    32.71906
comp 2  1.7371310        17.371310                    50.09037
comp 3  1.4049167        14.049167                    64.13953
comp 4  1.0568504        10.568504                    74.70804
comp 5  0.6847735         6.847735                    81.55577
comp 6  0.5992687         5.992687                    87.54846
```

Первые две компоненты объясняют только 50% статистического разброса результатов. Вклад (contribution) каждой переменной при формировании каждой главной компоненты может быть оценен как отношение квадрата факторной нагрузки (score) к соответствующему

собственному значению (Abdi, Williams, 2010). Построим ординационную диаграмму признаков (т.е. видов спорта в десятиборье), но их координаты обозначим не точкой, а стрелкой из начала координат, причем цвет стрелки определим в зависимости от важности каждой переменной:

```
res.pca$var$contrib
fviz_pca_var(res.pca, col.var="contrib",
   gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE) # Раскрасим также текст
```

```
                  Dim.1     Dim.2       Dim.3
100м          18.34376957  2.016090  2.42049891
Длина.прыжок  16.82246707  6.868559  2.36319121
Ядро          11.84353954 20.606785  0.03890276
Высота.прыжок  9.99788710  7.063694  4.79362526
400м          14.11622887 18.666374  1.23027094
110м.барьер   17.02011495  3.013382  0.61083225
Диск           9.32848615 21.162245  0.13131711
Шест.прыжок    0.07745541  1.872547 34.06090024
Копье          2.34696326  5.784369 10.80714169
1500м          0.10308808 12.945954 43.54331962
```

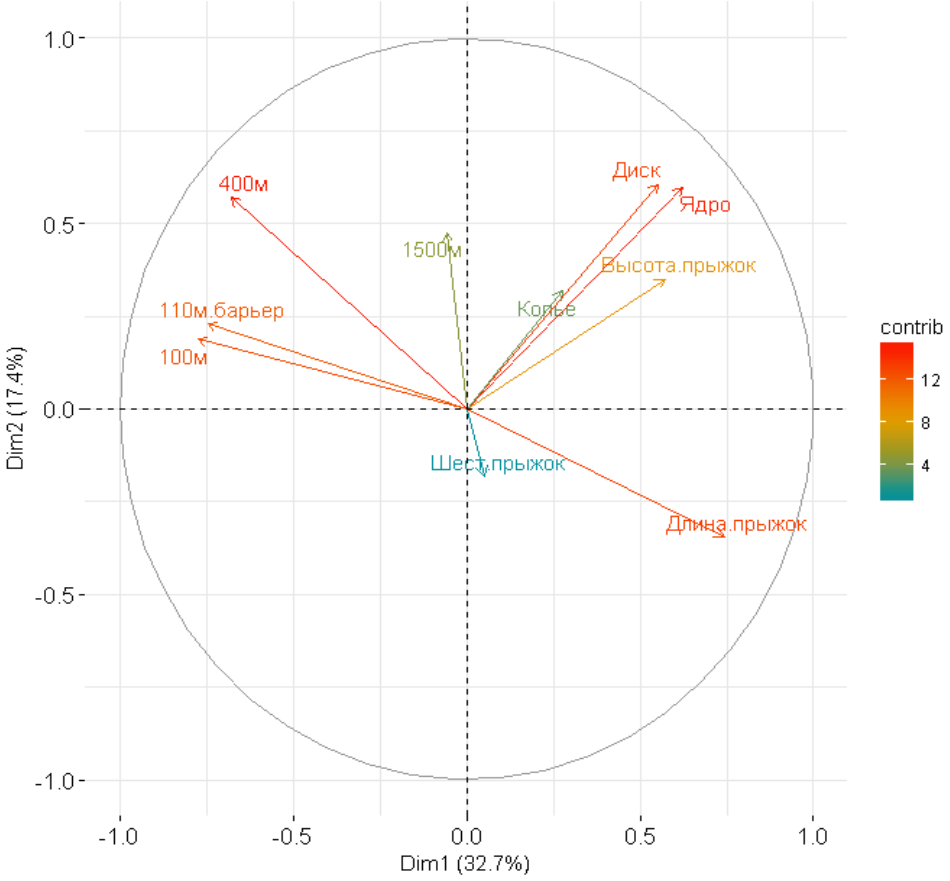


Рис. 9.9. Ординационная диаграмма видов соревнований в десятиборье

По сути примера видно, что первая компонента связана с темповыми видами спорта (бег на короткие дистанции), а вторая – с силовыми (метание диска и ядра).

Оценим теперь, имеются ли различия между результатами, показанными на Олимпиаде и на турнире "ДекаСтар". Для этого в структуру

объекта PCA включим в качестве вспомогательных переменных признаки, не участвующие в построении ординации:

```
res.pca <- PCA(decathlon, quanti.sup = 11:12, quali.sup = 13,
               graph=FALSE)
fviz_pca_ind(res.pca, geom = "point", habillage = 13,
  addEllipses =TRUE, ellipse.level = 0.68) +
  scale_color_brewer(palette="Dark2") +  theme_minimal()
```

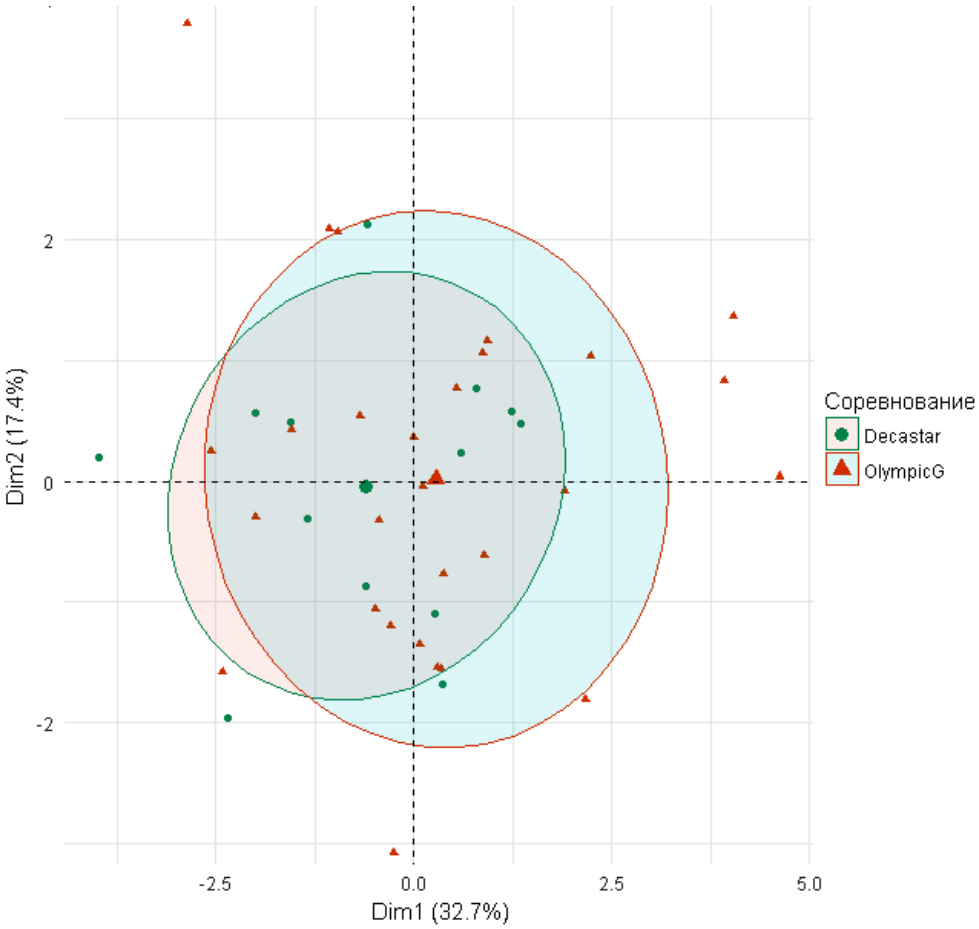


Рис. 9.10. Ординационная диаграмма участников соревнований в десятиборье

Нетрудно убедиться в том, что по оси первой главной компоненты обозначился "зазор" между групповыми центроидами и результаты, показанные на Олимпиаде, можно считать существенно более высокими. В то же время, разброс показателей на турнире "ДекаСтар" оказался меньше, т.е. состав участников был более однородным.

9.4. Оценка связи ординации с внешними факторами

Ранее мы рассматривали ординационный анализ только одной матрицы **Y,** чтобы получить отображение в ортогональной системе координат некоторых частных структурных особенностей изучаемой системы в форме графических проекций объектов и их признаков. Однако это – пассивный путь, поскольку он ограничивается фиксацией некоторого распределения

точек на плоскости и не дает никаких объяснений этому феномену. Такую ординацию называют также непрямой, или неограниченной (unconstrained), в отличие от прямой, или ограниченной (constrained) ординации, которая ставит задачу связать внутреннюю изменчивость матрицы **Y** с теми или иными внешними воздействиями **X**. Нам кажется, что непосредственный перевод приведенных терминов не вполне отражает суть дела и было бы точнее назвать эти методы ординации "необъясняющим" и "объясняющим" соответственно.

Концептуально *канонический анализ* позиционируется как расширение регрессионного анализа при моделировании многомерного отклика данных:

$$\mathbf{Y} = f(\mathbf{X}),$$

где **Y** – матрица размером $n \times m$ содержит значения отклика y_{ij}, измеренные по m признакам (столбцы) для n объектов (строки); **X** – матрица размером $n \times q$, в которой i-я строка содержит значения объясняющих переменных x_{ik}.

Наиболее часто используются две математические процедуры канонической ординации, известные под аббревиатурами RDA и CCA. В обоих случаях формой канонизации является вычисление собственных чисел и собственных векторов.

Анализ избыточности (RDA, redundancy analysis – Rao, 1964; Legendre, Legendre, 2012), общий формализм и алгебра которого были описаны в разделе 2.5, является расширенной канонической формой, объединяющей линейную модель множественной регрессии и метод главных компонент PCA. *Канонический анализ соответствий* (CCA, canonical correspondence analysis – ter Braak, 1986) является модификацией RDA, выполняющей "взвешивание" ординационных компонент пропорционально их вкладу в статистику χ^2, которая оценивает расстояния между объектами в многомерном пространстве. В связи с этим CCA в экологических приложениях придает существенно бóльшее значение редко встречающимся объектам, чем это делает RDA, основанный на евклидовых дистанциях.

В целом, оба метода – RDA и CCA – осуществляют аппроксимацию данных многомерной гауссовой регрессией и выполняют подгонку таких коэффициентов $\mathbf{a} = (a_i)$ для переменных матрицы **Y**, $i = 1, ..., m$, и $\mathbf{c} = (c_k)$ для факторов среды **X**, $k = 1, ..., q$, которые делают максимальной корреляцию между $\mathbf{z}^* = \mathbf{Y'a}$ и $\mathbf{z} = \mathbf{X'c}$. Расчеты в среде R обычно проводят с помощью функций `rda()` или `cca()` из пакета `vegan`. С этими функциями связано еще не менее двух десятков сервисных функций, выполняющих вычисление различных статистик, проверку гипотез о значимости как внутренних взаимодействий, так и о влиянии внешних факторов, и т.д.

Рассмотрим еще один геоботанический пример – набор `varespec`, описывающий относительное обилие 44 видов травянистых растений на 24 пробных площадках. Параллельный набор `varechem` содержит данные о 14 показателях химического состава почвы на тех же площадках.

Оценим предварительно характер распределения частот встречаемости объектов по рангам r списка видов S. Осуществим аппроксимацию данных

моделью Ципфа $N_r = N \cdot p_1 \ r^{\ \gamma}$ с двумя оцениваемыми параметрами: p_1 – долей самого многочисленного класса в общей численности N и γ – коэффициентом затухания гиперболической зависимости. Правда, мы имеем дело не с числом экземпляров растений, а с проективным покрытием в %, но нет никакой сложности превратить формально непрерывные значения обилия в ряд положительных целочисленных величин:

```
library(vegan)
data(varechem) ; data(varespec)
B <- rev(sort(colSums(varespec)))
BF <- as.matrix(t(round(B/min(B))))
rad <- rad.zipf(BF)
barplot(BF)
lines(1:ncol(BF), rad$fitted, col="green", lwd=2)
```

```
RAD model: Zipf              Family: poisson
No. of species:  44          Total abundance: 24175
        p1       gamma    Deviance         AIC         BIC
   0.34724    -1.29554 9036.36623 9308.19192 9311.76030
```

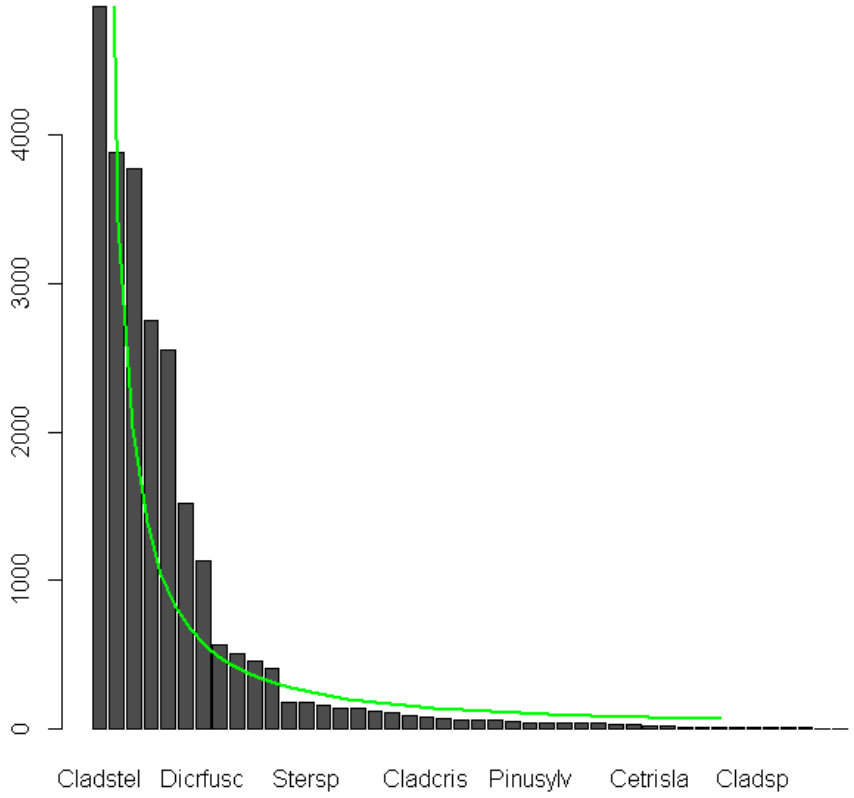


Рис. 9.11. Аппроксимация рангового распределения видов моделью Ципфа

Поскольку полученное распределение имеет очевидную склонность к доминированию видов, подвергнем эти исходные данные χ^2-преобразованию. Оценим влияние отдельных химических элементов в почве на встречаемость видов с использованием канонического анализа соответствий CCA (рис. 9.12):

```
varespec.chi <- decostand(varespec, method="chi.square")
mod1.cca <- cca(varespec.chi ~ ., varechem)
spec <- as.data.frame(mod1.cca$CCA$v[,1:2])  # Виды
sites <- as.data.frame(mod1.cca$CCA$u[,1:2])  # Площадки
```

```
fact <-  as.data.frame(mod1.cca$CCA$biplot[,1:2])  # Факторы
library(ggplot2)
ggplot() +
  geom_point(data=sites,aes(x=CCA1,y=CCA2),size=3,shape=17,
   color="red") + # добавляем точки площадок
  geom_text(data=spec,aes(x=CCA1,y=CCA2,
   label = abbreviate(rownames(spec),4)),colour="darkgreen",
   fontface = 3) +  # добавляем метки видов
  geom_segment(data=fact,aes(x=0,xend=CCA1,y=0,yend=CCA2),
      arrow = arrow(length = unit(0.5, "cm")),
      colour="darkblue") +  # добавляем стрелки факторов
  geom_text(data = fact, aes(x = CCA1, y = CCA2,
      label = rownames(fact), size = 6), hjust = 1.2,
       fontface = 2) + theme_bw()
```

```
              Inertia Proportion Rank
Total          3.2761     1.0000
Constrained    2.2603     0.6899   14
Unconstrained  1.0158     0.3101    9
Eigenvalues for constrained axes:
  CCA1   CCA2   CCA3   CCA4   CCA5   CCA6   CCA7   …  CCA14
0.4624 0.3017 0.2815 0.2651 0.2333 0.2028 0.1476  … 0.0190
Eigenvalues for unconstrained axes:
    CA1     CA2     CA3     CA4     CA5     CA6 …      CA9
0.29087 0.21535 0.17088 0.09812 0.07670 0.06552 … 0.02342
```

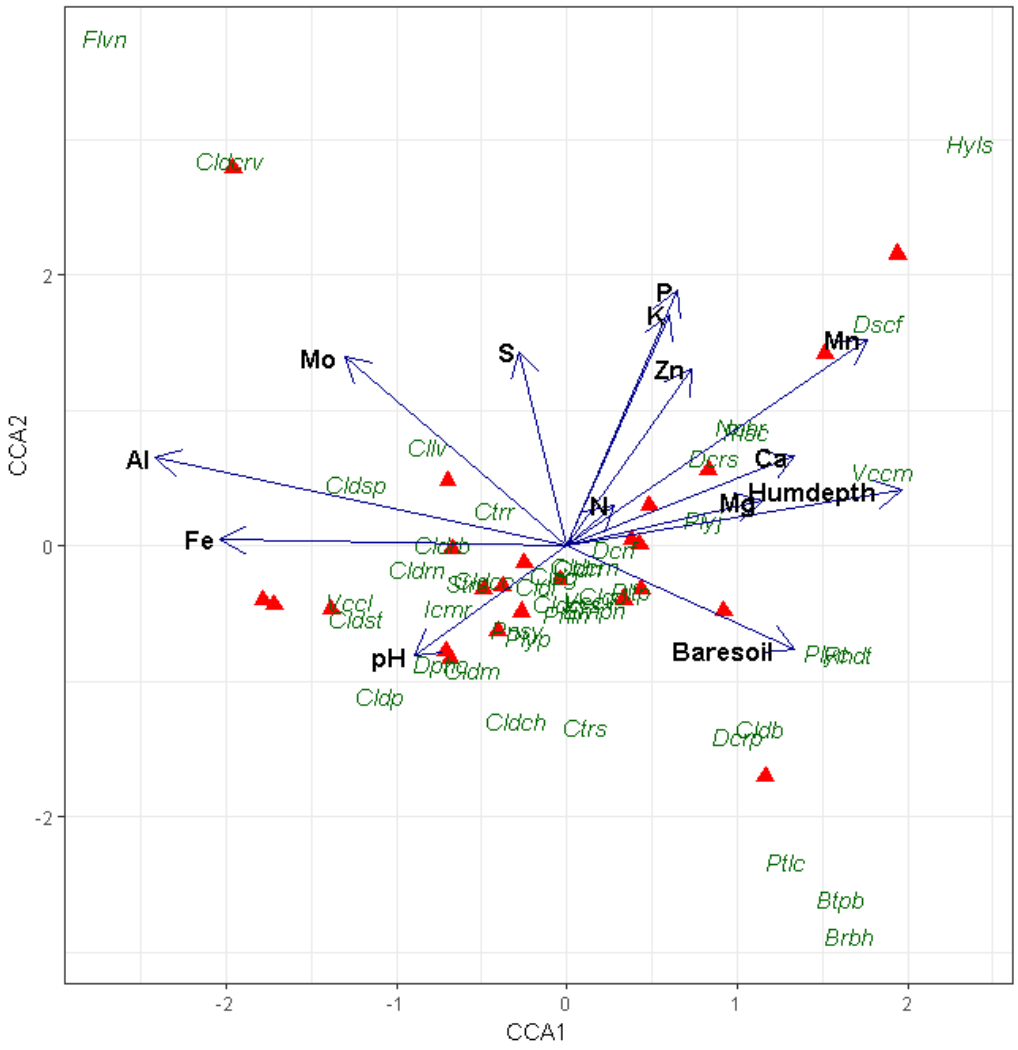


Рис. 9.12. Ординационный триплот, полученный методом CCA

Из результатов следует, что всего было сформировано 23 (т.е. $n - 1$) линейно независимых оси ординации, в том числе 14 осей объясняющих переменных (`Constrained`). 69% совокупной многомерной дисперсии (`Inertia`) матрицы связаны с химическим составом почвы, а 31% – со случайными или неучтенными факторами.

Результирующая диаграмма ("триплот" – см. рис. 9.12) отражает не только изменчивость видового состава относительно двух осей проецирования CCA_1-CCA_2, но и статистические связи между видами и каждой независимой переменной x_{ji}. Для этого из центра координат стрелками синего цвета проведены дополнительные оси (`constrained axes`), ориентация которых зависит от значений канонических собственных векторов. Относительная длина стрелки пропорциональна силе связи внешнего фактора с осями ординации CCA_1-CCA_2, а косинус угла между каждой парой стрелок приблизительно равен коэффициенту корреляции между факторами (положительной или отрицательной).

Как и в случае биплота, координаты каждого вида находятся вблизи точек тех площадок, где его обилие наиболее высоко. Если виды представить стрелками, исходящими из начала координат, то косинус угла между стрелкой соответствующего вида и стрелкой фактора среды приблизительно равен коэффициенту корреляции между ними. Например, обилие вида *Callvulg* (*c11v*) в значительной степени коррелирует с содержанием в почве молибдена, в отличие от *Cladphyl* (*c1dp*). Проекция точки вида на каждую стрелку показывает его экологический оптимум (точнее, центр тяжести распределения обилия) относительно этого химического компонента.

Нетрудно убедиться из графика на рис. 9.12, что показатели состава почвы очень высоко коррелирут друг с другом, о чем свидетельствует фактор инфляции дисперсии VIF.

```
library(car)
sort(vif(mod1.cca))
paste("AIC=", extractAIC(mod1.cca)[2])
```

```
       N  Baresoil        Mn  Humdepth        Mo         P        pH
1.637648  1.966942  5.163652  5.258233  5.760964  6.398181  6.593768
      Zn        Fe        Ca        Mg         K         S        Al
8.446984  9.100191  9.295130 10.531890 11.705305 20.424393 21.546563
[1] "AIC= 65.58352"
```

Дисперсионный анализ `anova()` в отношении объектов CCA или RDA имеет важный параметр `by`: при `by = "axis"` оценивается значимость объясняющих осей, значение `by = "term"` дает возможность последовательно проверить по F- и χ^2-критериям значимость отдельных коэффициентов регрессионной модели, а в случае `by = "margin"` проверяется наличие маргинального эффекта. Если параметр `by` не задан, то оценивается значимость ординационной модели в целом. Естественно, что все проверки осуществляются с использованием перестановочного теста, т.е. проводится многократное случайное перемешивание строк исходных таблиц

и оценивается независимость результатов проверки нулевой гипотезы от процесса хаотического обмена (exchangeability) анализируемых единиц.

Высокая мультиколлинеарность таблицы **X** обусловливает низкую статистическую значимость включения в линейную модель почти каждой из объясняющих переменных:

```
anova(mod1.cca)
anova(mod1.cca, by = "term", step=200)
```

```
Permutation test for cca under reduced model
         Df ChiSquare      F Pr(>F)
Model    14    2.2603 1.4304  0.077 .
Residual  9    1.0158
---
         Df ChiSquare      F Pr(>F)
N         1   0.16487 1.4607  0.223
P         1   0.23009 2.0385  0.020 *
K         1   0.22026 1.9514  0.128
Ca        1   0.17213 1.5250  0.215
Mg        1   0.11727 1.0390  0.531
S         1   0.23605 2.0913  0.017 *
Al        1   0.17953 1.5906  0.121
Fe        1   0.11987 1.0620  0.512
Mn        1   0.13858 1.2278  0.380
Zn        1   0.14076 1.2470  0.324
Mo        1   0.13971 1.2377  0.343
Baresoil  1   0.14812 1.3123  0.269
Humdepth  1   0.12972 1.1493  0.395
pH        1   0.12330 1.0924  0.415
Residual  9   1.01585
```

Для построения оптимальной модели выполним алгоритм пошагового включения переменных, стартуя от нуль-модели `mod0.cca`. На каждом шаге будем проводить перестановочный тест значимости включаемого члена:

```
mod0.cca <- cca(varespec.chi ~ 1, varechem)
mod.cca <- step(mod0.cca, scope = formula(mod1.cca),
                test = "perm", trace=0)
mod.cca$anova
anova(mod.cca) ; anova(mod.cca, by="axis")
```

```
  Step Df Deviance Resid. Df Resid. Dev      AIC
1      NA       NA        23   340.9154 65.68593
2 + Al -1 37.46399        22   303.4514 64.89202
3 +  P -1 25.64418        21   277.8072 64.77296
4 +  K -1 24.38595        20   253.4212 64.56798
---
Permutation test for cca under reduced model
Model: cca(formula = varespec.chi ~ Al+P+K, data = varechem)
         Df ChiSquare      F Pr(>F)
Model     3    0.8408 2.3017  0.001 ***
Residual 20    2.4353
---
         Df ChiSquare      F Pr(>F)
CCA1      1   0.38691 3.1775  0.001 ***
CCA2      1   0.24503 2.0123  0.013 *
CCA3      1   0.20886 1.7153  0.081 .
Residual 20   2.43532
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

При выполнении пошаговой процедуры уже после 4-го шага началось увеличение AIC-критерия до значения 65.58 при полном составе переменных. В итоге была получена модель с высокой степенью статистической значимости в целом, однако использование уже третьей факторной оси возможно оказалось излишним. Судя по всему, алгоритм не смог четко распорядиться высокой корреляционной зависимостью между содержанием фосфора и калия (см. рис. 9.11).

9.5. Неметрическое многомерное шкалирование и построение распределения чувствительности видов

Перспективным методом ординации, находящим все большее применение в различных предметных областях, является алгоритм неметрического многомерного шкалирования (NMDS, nonmetric multidimensional scaling – Краскел, 1986; McCune, Grace, 2002). Его главным преимуществом является то, что он не требует от исходных данных никаких априорных предположений о характере статистического распределения. Считается, что этот метод дает наиболее адекватные результаты, особенно для больших матриц с сильными "шумами".

Неметрическое шкалирование, как и метод главных координат, использует произвольную матрицу дистанций **D** и ставит задачу создать такую двумерную проекцию изучаемых объектов, на которой взаимные расстояния между этими объектами оказались бы наименее искажены по сравнению с исходным состоянием **D**. Главные оси геометрической метафоры данных в пространстве меньшей размерности обычно находят путем выполнения последовательности итераций для минимизации критерия Δ, оценивающего меру расхождений между исходной и моделируемой матрицами расстояний. Для этого обычно используют величину "стресса": $\Delta = \sum_{i,j=1}^{n} d_{ij}^{\alpha} \left| \hat{d}_{ij} - d_{ij} \right|^{\beta}$, где d_{ij} и $\hat{d}_{ij}$ – расстояния между объектами i и j в исходном и редуцированном пространствах, α и β – задаваемые коэффициенты.

Пакет `vegan` содержит ряд функций для проведения расчетов методом NMDS и построения объясняющих ординаций. Рассмотрим пример, включающий наборы данных `varespec` и `varechem`, которые использовались нами в предыдущем разделе для ординации каноническим методом CCA, основанным на линейных преобразованиях. На первом этапе определим наилучшую метрику расстояния, рассчитав для каждого из ее вариантов коэффициент корреляции Спирмена между двумя матрицами дистанций (в пространствах видов растительности и химического состава почвы). Эту операцию выполняет функция `rankindex()`:

```
library(vegan)
data(varechem) ; data(varespec)
varespec.chi <- decostand(varespec, method="chi.square")
# Коэффициенты корреляции Спирмена для различных метрик
rankindex(varechem, varespec.chi,
                  c("euc","man","bray","jac","kul"))
```

```
      euc       man      bray       jac       kul
0.2228233 0.1894385 0.3022988 0.3022988 0.3071600
```

Используем для расчета матрицы дистанций формулу Кульчицкого. Далее функция `metaMDS()` осуществляет неметрическое шкалирование и минимизацию стресса Δ, а функция `envfit()` – расчет коэффициентов корреляции каждого из показателей химического состава почвы с осями ординации $NMDS_1$-$NMDS_2$ и оценку статистической значимости p этих коэффициентов на основе перестановочного теста. Функция `MDSrotate()` вращает систему координат таким образом, чтобы совместить ось $NMDS_1$, например, с осью концентрации алюминия:

```
vare.mds <- metaMDS(varespec,distance = "kul", trace = FALSE)
ord.mds <- MDSrotate(vare.mds, varechem$Al)
ef <- envfit(vare.mds, varechem, permu = 999)
```

```
global Multidimensional Scaling using monoMDS
Data:     wisconsin(sqrt(varespec))
Distance: kulczynski
Stress:     0.1825658
----
            NMDS1    NMDS2     r2 Pr(>r)
N        -0.43946  0.89826 0.2536  0.043 *
P        -0.15541 -0.98785 0.1939  0.106
K        -0.35324 -0.93553 0.1809  0.115
Ca       -0.24020 -0.97072 0.4119  0.004 **
Mg       -0.17161 -0.98517 0.4270  0.004 **
S         0.31435 -0.94931 0.1752  0.123
Al        1.00000  0.00000 0.5269  0.001 ***
Fe        0.98839  0.15193 0.4450  0.002 **
Mn       -0.99114  0.13279 0.5231  0.002 **
Zn       -0.15262 -0.98828 0.1879  0.117
Mo        0.99764  0.06863 0.0610  0.521
Baresoil -0.99256 -0.12179 0.2508  0.052 .
Humdepth -0.98969 -0.14321 0.5201  0.002 **
pH        0.93818 -0.34615 0.2308  0.053 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Number of permutations: 999
```

Построим ординационные диаграммы с использованием стандартных графических средств R (как это можно сделать на основе `ggplot2` см. github.com/gavinsimpson/ggvegan, chrischizinski.github.io или stackoverflow.com). Первая диаграмма (рис. 9.13) включает ординацию площадок и оси девяти внешних факторов, для которых $p < 0.1$.

```
plot(vare.mds, type = "t", disp="sites", font=2)
points(vare.mds, disp="sites",cex=3, col="red")
abline(h = 0, lty = 3) ;   abline(v = 0, lty = 3)
plot(ef, p.max = 0.1, col="blue", font=2, lwd=2, cex=1)
```

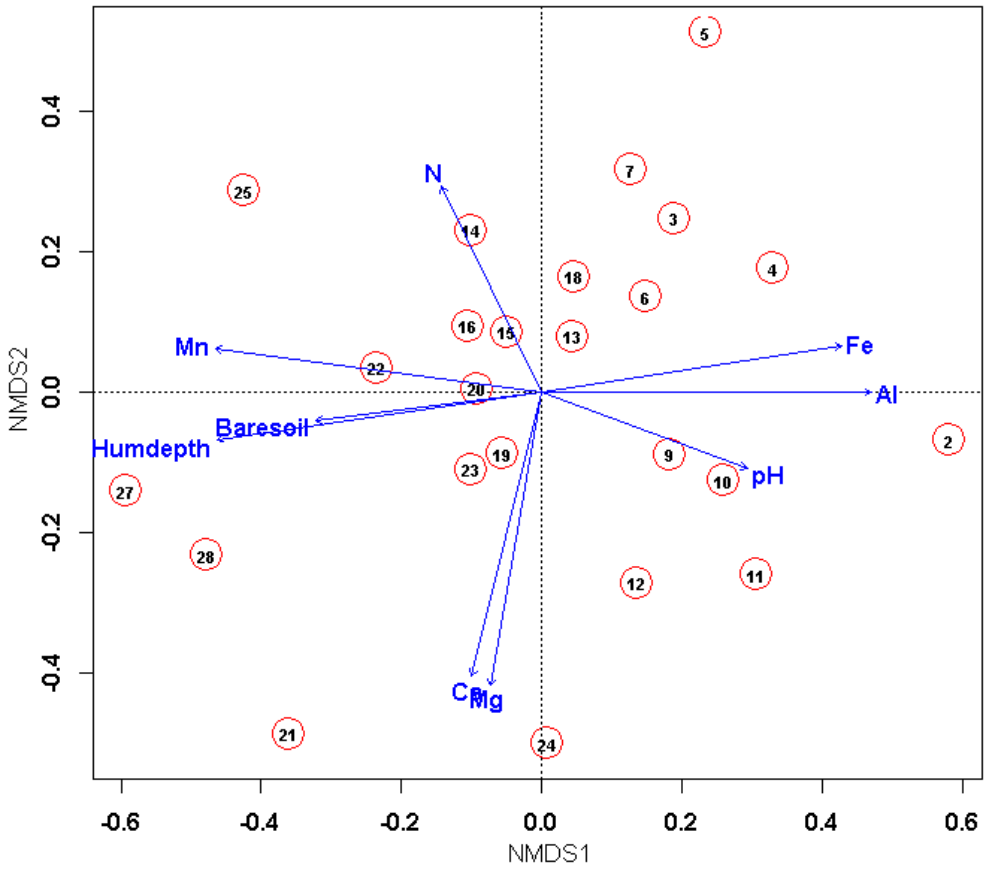


Рис. 9.13. Ординация площадок, полученная методом NMDS

Прежде чем построить вторую диаграмму, для концентрации в почве того или иного химического компонента $y_{хэ}$ по его эмпирическим значениям для каждой площадки подгонялись обобщенные аддитивные модели (Wood, 2006):

$$y_{хэ} = s(v_1, v_2) + \varepsilon, \qquad (9.4)$$

где s – функция двумерного сплайна с k степенями свободы от NMDS-координат v_1 и v_2. Построение этой модели осуществляется функцией `ordisurf()`, которая параллельно отрисовывает изолинии трехмерной сглаживающей поверхности на ординационной диаграмме.

Обратим особое внимание на отличие описываемого метода от обычных пространственных моделей кригинга в естественных географических координатах, которые сильно подвержены случайным флуктуациям свойств рельефа. Построение моделей сглаживания по проекциям матрицы отклика на плоскость с абстрактными осями, непосредственно связанными с изучаемой экологической структурой, обеспечивает получение более гладких и устойчивых поверхностей.

На рис. 9.14 показано, как на ординацию видов растительности могут быть наложены изолинии по моделям сглаживания содержания `Al` и `Ca`.

```
plot(vare.mds,  type="n")
 ordipointlabel(vare.mds,
         display = "sp", font=4, col="darkgreen", add=TRUE)
abline(h = 0, lty = 3) ;   abline(v = 0, lty = 3)
with(varechem, ordisurf(vare.mds, Al, add = TRUE, col =2))
with(varechem, ordisurf(vare.mds, Ca, add = TRUE, col =4))
```

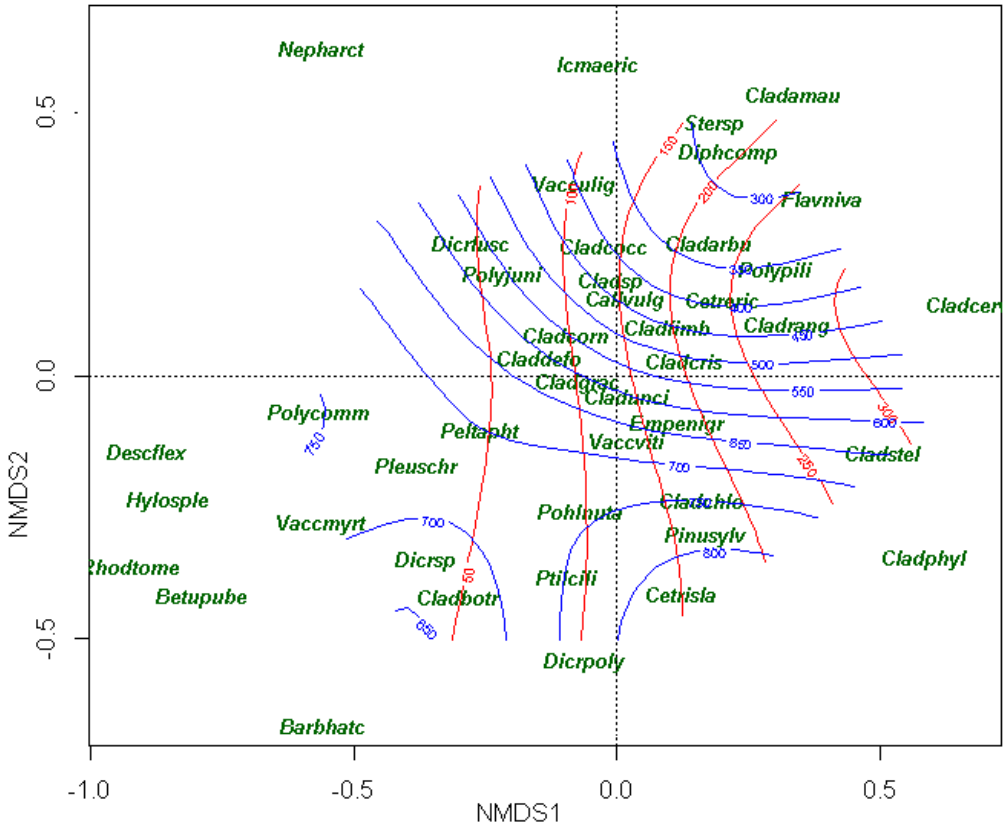


Рис. 9.14. Ординация видов методом NMDS
и изолинии сглаживающих поверхностей для Al и Ca

Вероятно, можно предположить (по крайней мере, в рамках имеющейся выборки), что каждая точка ординации, соответствующая тому или иному виду, определяет координаты наилучших природных условий существования этого вида, а прогнозное значение модели (9.4) оценивает его экологический оптимум по сглаживаемому фактору. Функция `calibrate(mod, newdata)`, аналогичная функции `predict()`, осуществляет с использованием модели `mod` вычисление для произвольной таблицы `newdata` прогнозных значений и их стандартных ошибок `se.fit`.

Оценим экологические оптимумы `val` содержания алюминия `Al` в почве для каждого обнаруженного вида растительности. Вычислим также верхние 95%-ные доверительные границы предсказания `valC` по формуле `valC = val + tc*se.fit`, где `tc` – квантиль распределения Стьюдента при $\alpha = 95\%$. Чтобы блокировать появление отрицательных значений, значение внешнего фактора предварительно прологарифмируем.

```
Al.log <-log(varechem$Al)
fit.Al <- ordisurf(vare.mds, Al.log)
a <- as.data.frame(vare.mds$species)
Sp.Al <- cbind(a,
         calibrate(fit.Al, newdata = a, se.fit = TRUE))
Sp.Al$val <- exp(Sp.Al$fit)
tc <- qt(0.975, nrow(Sp.Al)-1)
Sp.Al$valC <- exp(Sp.Al$fit)+ exp(Sp.Al$se.fit)*tc
head(Sp.Al)
```

```
            MDS1     MDS2    fit  se.fit    val    valC
Callvulg  0.0981   0.1191 4.9728 0.1875 144.438 146.870
Empenigr  0.0208  -0.0648 4.6641 0.1924 106.073 108.518
Rhodtome -0.8127  -0.3608 2.1142 0.5222   8.281  11.681
Vaccmyrt -0.6651  -0.2820 2.4965 0.4284  12.140  15.235
Vaccviti  0.0202  -0.0892 4.6554 0.1975 105.154 107.611
Pinusylv  0.1673  -0.2725 5.0908 0.2559 162.520 165.125
```

Одним из методов нормирования техногенных загрязнений на исследуемой территории является построение статистических моделей распределения чувствительности видов SSD (Species Sensitivity Distribution – Posthuma et al., 2001). Кривая SSD рассчитывается как интегральная функция некоторого теоретического распределения плотности вероятности встречаемости видов, параметры которого оцениваются по выборке того или иного показателя жизнедеятельности таксономических групп (чаще всего для этого используются показатели токсикометрии NOEC, DC_{50}, LC_{50} и т.д.).

Поскольку для травянистых растений параметры токсичности отсутствуют, то в качестве предположительно опасных значений содержания алюминия в почве для *j*-го вида примем настолько большие его величины, которые маловероятны в рамках сглаживающей модели GAM, т.е. столбец `valC` сформированной таблицы `Sp.Al`. Если отсортировать этот показатель по его величине, то функция `ppoints()` сгенерирует последовательность эмпирических вероятностей `frac`, т.е. долю значений `valC`, которые не превышают величину `valC` для каждого *j*-го вида.

```
# Сортировка по возрастанию val
df <- Sp.Al[,5:6]
df <- df[order(df$val), ]
df$frac <- ppoints(df$val, 0.5)
head(df)
```

```
               val     valC       frac
Hylosple  5.272568  8.87682 0.01136364
Nepharct  6.750738 10.92165 0.03409091
Descflex  6.925953 10.27115 0.05681818
Rhodtome  8.281667 11.68154 0.07954545
Polycomm 10.531262 13.60922 0.10227273
Betupube 12.113196 15.33292 0.12500000
```

Для аппроксимации данных теоретическим распределением воспользуемся функцией `fitdist()` из пакета `fitdistplus`. Наилучшая аппроксимация, процедуру подбора которой мы опускаем, соответствует логнормальному распределению. Найдем параметры этого распределения, т.е. оценки среднего `meanlog` и дисперсии `sdlog`, а также некоторые квантильные значения `Al`.

```
library(fitdistrplus)
fit <- fitdistr(df$valC, "lognormal")
ep1s <- fit$estimate[1]; ep2s <- fit$estimate[2]
hcs <- qlnorm(c(0.05, 0.1, 0.2), meanlog=ep1s,sdlog=ep2s)
```

```
  meanlog     sdlog
 4.290294  1.007003 13.928272 20.080861 31.273734
```

Для построения кривой распределения чувствительности видов (SSD) будем использовать скрипт, представленный в блоге «Data in Environmental Science and Ecotoxicology» (http://edild.github.io/ssd/, автор E. Szöcs). Оценка доверительных интервалов теоретической кривой может быть осуществлена с использованием *параметрического* бутстрепа, который использует предположение о том, что исходные выборочные данные представляют собой случайные реализации вероятностного процесса, определяемого параметрами заданного распределения (Davison, Hinkley, 2006). Подробные комментарии к тексту скрипта см. в нашей книге (Шитиков, 2016).

Определим предварительно две функции, необходимые нам в основных расчетах и выполняющие генерацию псевдо-выборок:

```
# 1. функция для нахождения p-квантили случайной выборки из
#  логнормального распределения с параметрами  fit
myboot <- function(fit, p){
  xr <- rlnorm(fit$n, meanlog = fit$estimate[1],
                      sdlog = fit$estimate[2])
  fitr <- fitdist(xr, 'lnorm')
    hc5r <- qlnorm(p, meanlog = fitr$estimate[1],
                    sdlog = fitr$estimate[2])
  return(hc5r)
}
# 2. функция, возвращающая значения вероятностей для случайной
#  выборки из логнормального распределения с параметрами  fit
myboot2 <- function(fit, newxs){
  xr <- rlnorm(fit$n, meanlog = fit$estimate[1],
                      sdlog = fit$estimate[2])
  fitr <- fitdist(xr, 'lnorm')
  pyr <- plnorm(newxs, meanlog = fitr$estimate[1],
                       sdlog = fitr$estimate[2])
  return(pyr)
}
```

Выполним построение 1000 бутстреп-кривых из заданного распределения и определим координаты графиков основной функции распределения и ее 95%-ных доверительных огибающих:

```
set.seed(1234) # Установка генератора случайных чисел
require(reshape2)
# новые данные для построения плавной кривой
newxs <- 10^(seq(log10(0.01), log10(max(df$valC)),
                              length.out = 1000))
# получение матрицы для построения 1000 кривых
boots <- replicate(1000, myboot2(fit, newxs))
bootdat <- data.frame(boots) ; bootdat$newxs <- newxs
bootdat <- melt(bootdat, id = 'newxs')
# извлечение доверительных интервалов
cis <- apply(boots, 1, quantile, c(0.025, 0.975))
rownames(cis) <- c('lwr', 'upr')
# формирование итоговой таблицы подогнанных значений и  cis
pdat <- data.frame(newxs, py = plnorm(newxs,
       meanlog = fit$estimate[1], sdlog = fit$estimate[2]))
pdat <- cbind(pdat, t(cis))
# координаты x для названия видов
df$fit <- 10^(log10(qlnorm(df$frac,
   meanlog = fit$estimate[1], sdlog = fit$estimate[2])) -0.4)
```

Осуществим вывод полноценного графика SSD с использованием пакета `ggplot2` (рис. 9.15):

```
library(ggplot2)
ggplot() +
geom_line(data = bootdat, aes(x = newxs, y = value,
     group = variable), col = 'steelblue', alpha = 0.05) +
geom_point(data = df, aes(x = valC, y = frac)) +
geom_line(data = pdat, aes(x = newxs, y = py), col = 'red') +
geom_line(data = pdat, aes(x = newxs, y = lwr),
             linetype = 'dashed') +
geom_line(data = pdat, aes(x = newxs, y = upr),
             linetype = 'dashed') +
geom_text(data = df, aes(x = fit, y = frac,
    label = rownames(df)), hjust = 1, size = 4) +
scale_x_log10(breaks = c(5,  10, 50, 100, 300),
        limits = c(1, max(df$valC))) +
labs(x = 'Содержание алюминия в почве',
           y = 'Доля видов с неблагоприятными условиями') +
theme_bw()
```

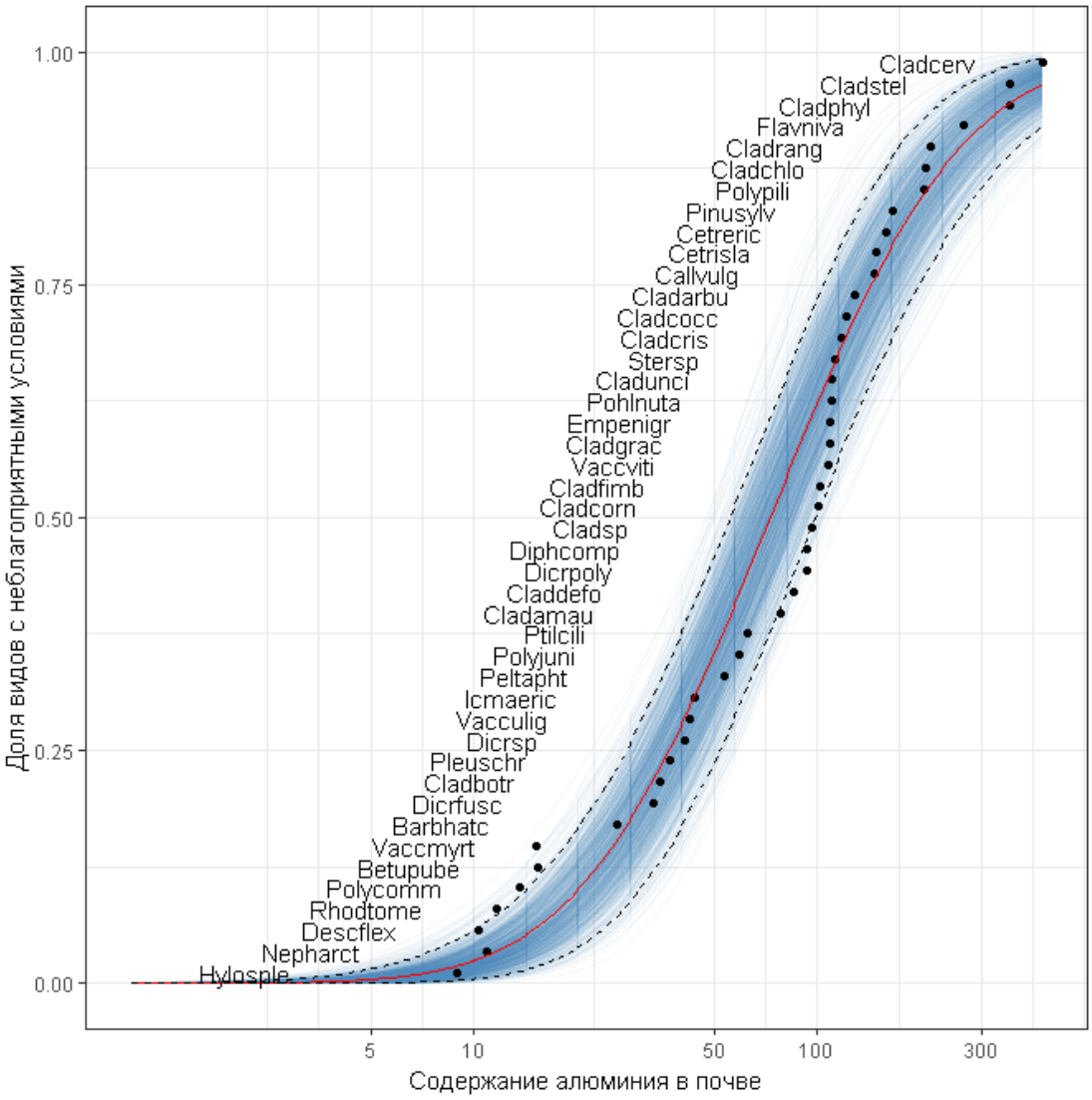


Рис. 9.15. Распределение чувствительности видов в зависимости от содержания алюминия в почве

Если задаться произвольной критической вероятностью на оси ординат SSD (см. рис. 9.15), то с использованием кумулятивной кривой распределения можно оценить величину потенциально опасного уровня воздействия внешнего фактора (Шитиков, 2016). Из представленных результатов следует, что уже при содержании алюминия в почве Al = {13.9, 20.1 и 31.3} из травянистого сообщества могут исчезнуть соответственно доли p = {5, 10, и 20%} видов от их общего числа.

Конечно, мы здесь не собирались делать каких-либо конкретных выводов в отношении вредности для растений тех или иных ингредиентов в почве, и даже не исключаем, что алюминий полезен для травостоя. Мы только использовали этот пример, чтобы описать реализацию в R алгоритмов ординации многомерных данных, подбора параметров статистических распределений и оценки степени критичности воздействия внешнего фактора.

Поскольку современные тенденции исследований связаны с интенсивным заимствованием методов и приемов из разнородных отраслей, описанные алгоритмы ординации, вероятно, пригодны для использования не только в экологических приложениях. Например, в социологических исследованиях виды растений можно вполне заменить наблюдаемыми предпочтениями избирателей в отношении потенциальных кандидатур на ту или иную должность. А интенсивность ремонта дорог может служить важным внешним фактором, обусловливающим вероятность победы на выборах требуемого кандидата.

10. КЛАСТЕРНЫЙ АНАЛИЗ

10.1. Алгоритмы кластеризации, основанные на разделении

Алгоритмы неиерахического разделения (partitioning algorithms) осуществляют декомпозицию набора данных, состоящего из n наблюдений, на k групп (кластеров) с заранее неизвестными параметрами. При этом выполняется поиск *центроидов* – максимально удаленных друг от друга центров сгущений точек C_k с минимальным разбросом внутри каждого кластера. К разделяющим алгоритмам относятся:

- метод k средних Мак-Кина (k-means clustering**,** MacQueen, 1967), в котором каждый из k кластеров представлен центроидом;
- разделение вокруг k медоидов или PAM (Partitioning Around Medoids – Kaufman, Rousseeuw, 1990), где *медоид* – это центроид, координаты которого смещены к ближайшему из исходных объектов данных;
- алгоритм CLARA (Clustering Large Applications) – метод, весьма похожий на PAM и используемый для анализа больших наборов данных.

Метод k средних выполняет кластеризацию следующим образом:

1. Назначается число групп (k), на которые должны быть разбиты данные. Случайно выбирается k объектов исходного набора как первоначальные центры кластеров.

2. Каждому наблюдению присваивается номер группы по самому близкому центроиду, т.е. на основании наименьшего евклидова расстояния между объектом и точкой C_k.

3. Пересчитываются координаты центроидов μ_k всех k кластеров и вычисляются внутригрупповые разбросы (within-cluster variation) $W(C_k) = \sum_{x_i \in C_k} (x_i - \mu_k)^2$. Если набор данных включает p переменных, то μ_k представляет собой вектор средних с p элементами.

4. Минимизируется общий внутригрупповой разброс $W_{total} = \sum_k W(C_k) \rightarrow$ min, для чего шаги 2 и 3 повторяются многократно, пока назначения групп не прекращают изменяться или не достигнуто заданное число итераций `iter.max`. Предельное число итераций для минимизации W_{total}, установленное функцией `kmeans()` по умолчанию, составляет `iter.max = 10`.

В разделе 2.6 мы уже подробно разбирали пример кластеризации 50 штатов США по криминогенной обстановке на 5 групп с использованием функции `kmeans()` из пакета `cluster`. Пример сопровождается скриптами, графической интерпретацией разбиения и приводятся конкретные вычисленные значения вышеприведенных статистик. В связи с этим в дальнейшем мы будем обсуждать только нюансы практического использования метода.

Объединение в кластеры методом k средних – очень простой и эффективный алгоритм, имеющий, однако, две существенные проблемы. Во-первых, итоговые результаты чувствительны к начальному случайному выбору центров групп. Возможное решение этой проблемы состоит в многократном выполнении алгоритма с различным случайным назначением начальных центроидов. Итерация с минимальным значением W_{total} отбирается как конечный вариант кластеризации. Число таких итераций можно задать параметром `nstart` функции `kmeans()`:

```
library(cluster)
data("USArrests")
df.stand <- as.data.frame(scale(USArrests))
c(kmeans(df.stand,centers=5, nstart=1)$tot.withinss,
  kmeans(df.stand,centers=5, nstart=25)$tot.withinss)
```

```
[1] 50.72761 48.94420
```

Нам удалось за 25 повторов алгоритма несколько уменьшить значение критерия оптимальности.

Вторая проблема – необходимость априори задавать фиксированное число кластеров для разбиения, которое, безусловно, далеко не всегда выбирается оптимальным. Поэтому одной из задач кластерного анализа является подбор оптимального значения k, для которой существует несколько версий решения.

Метод "локтя" (elbow method) рассматривает характер изменения разброса W_{total} с увеличением числа групп k. Объединив все n наблюдений в одну группу, мы имеем наибольшую внутрикластерную дисперсию, которая будет снижаться до 0 при $k \to n$. На каком-то этапе можно усмотреть, что снижение этой дисперсии замедляется – на графике это происходит в точке, называемой "локтем" (родственник "каменистой осыпи" для анализа главных компонент). Построить такой график можно в результате прямого перебора, либо с использованием функции `fviz_nbclust()` из прекрасного пакета `factoextra`, предназначенного для визуализации результатов кластерного анализа на основе графической системы `ggplot2`:

```
k.max <- 15 # максимальное число кластеров
wss <- sapply(1:k.max, function(k)
     {kmeans(df.stand, k, nstart=10 )$tot.withinss})
plot(1:k.max, wss, type="b", pch = 19, frame = FALSE,
       xlab="Число кластеров K",
       ylab="Общая внутригрупповая сумма квадратов")
# формируем график с помощью fviz_nbclust()
library(factoextra)
fviz_nbclust(df.stand, kmeans, method = "wss") +
    geom_vline(xintercept = 4, linetype = 2)
```

Ограничимся представлением только одного графика, полученного функцией `fviz_nbclust()` (рис. 10.2). Два вдумчивых аналитика, безусловно, могут поспорить о месте расположения "максимально согнутого локтевого сустава": для числа групп $k = 2$ или $k = 4$.

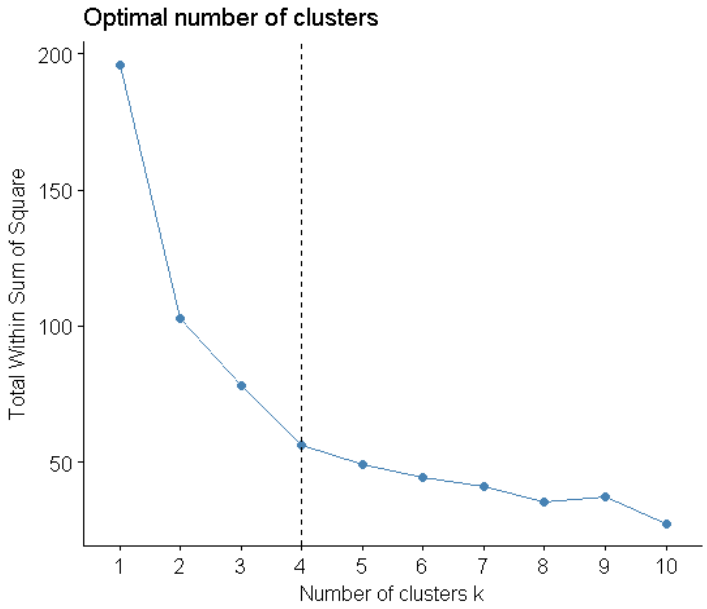


Рис. 10.1. Выбор оптимального числа кластеров по методу "локтя"

Альтернативой субъективно понимаемым графикам "локтя" является использование *статистики разрыва* (gap statistic – Tibshirani et al., 2001), которые генерируются на основе ресэмплинга и имитационных процедур Монте-Карло. Пусть $E_n^*\{\log(W_k^*)\}$ обозначает оценку средней дисперсии W_k^*, полученной бутстреп-методом, когда k кластеров образованы случайными наборами объектов из исходной выборки размером n. Тогда статистика

$$Gap_n(k) = E_n^*\{\log(W_k^*)\} - \log(W_k)$$

определяет отклонение наблюдаемой дисперсии W_k от ее ожидаемой величины при справедливости нулевой гипотезы о том, что исходные данные образуют только один кластер.

При сравнительном анализе последовательности значений $Gap_n(k)$, $k = 2, \ldots, K_{max}$ наибольшее значение статистики соответствует наиболее полезной группировке, дисперсия которой максимально меньше внутригрупповой дисперсии кластеров, собранных из случайных объектов исходной выборки:

```
set.seed(123)
gap_stat <- clusGap(df.stand, FUN = kmeans, nstart = 10,
                    K.max = 10, B = 50)
# Печать и визуализация результатов
print(gap_stat, method = "firstmax")
fviz_gap_stat(gap_stat)
```

```
          logW   E.logW       gap     SE.sim
 [1,] 3.458369 3.633279 0.1749100 0.04956376
 [2,] 3.135112 3.370352 0.2352405 0.04192860
 [3,] 2.977727 3.234819 0.2570927 0.04085179
 [4,] 2.826221 3.120886 0.2946656 0.04153549
 [5,] 2.738868 3.022296 0.2834282 0.04377128
 [6,] 2.669860 2.935234 0.2653739 0.04201917
 [7,] 2.612957 2.855663 0.2427060 0.04029606
 [8,] 2.545027 2.782767 0.2377398 0.04031744
 [9,] 2.471474 2.715964 0.2444898 0.03998257
[10,] 2.397200 2.654907 0.2577068 0.04074831
```

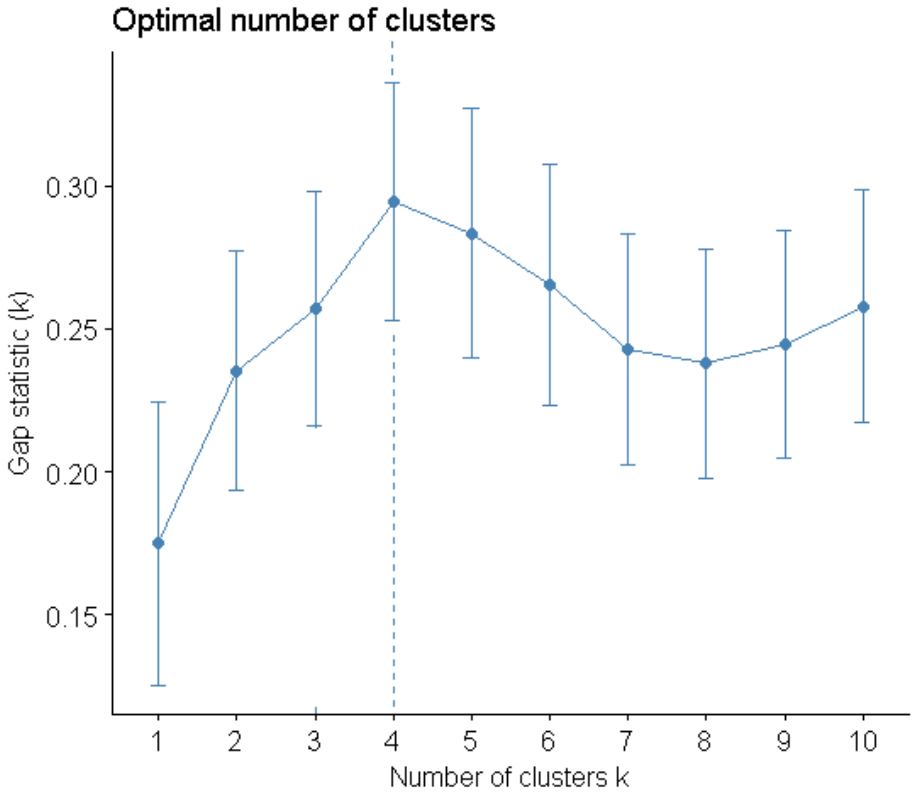


Рис. 10.2. График GAP-статистики для выбора оптимального числа кластеров

Параметр `FUNcluster` использованной выше функции `clusGap()` указывает на необходимый метод кластеризации и при `FUN = pam` осуществляется разделение вокруг *k* медоидов:

```
set.seed(123)
gap_stat <- clusGap(df.stand, FUN =  pam, K.max = 7, B = 100)
print(gap_stat, method = "firstmax")
(k.pam <- pam(df.stand, k=4))
```

```
          logW   E.logW       gap     SE.sim
[1,] 3.458369 3.632850 0.1744804 0.03737864
[2,] 3.135112 3.380017 0.2449056 0.04476136
[3,] 2.981802 3.250681 0.2688793 0.04601583
[4,] 2.834744 3.141994 0.3072506 0.04709690
[5,] 2.745099 3.049445 0.3043454 0.04729300
[6,] 2.685908 2.962081 0.2761727 0.04687566
[7,] 2.634287 2.886058 0.2517713 0.04560393
Medoids:
               ID     Murder    Assault   UrbanPop         Rape
Alabama         1  1.2425641  0.7828393 -0.5209066 -0.003416473
Michigan       22  0.9900104  1.0108275  0.5844655  1.480613993
Oklahoma       36 -0.2727580 -0.2371077  0.1699510 -0.131534211
New Hampshire  29 -1.3059321 -1.3650491 -0.6590781 -1.252564419
```

Метод PAM во многом идентичен алгоритму *k* средних за исключением того, что вместо вычисления центроидов осуществляется поиск *k* наиболее представительных объектов (или медоидов) среди анализируемых наблюдений. Внутрикластерный разброс оценивается при этом по манхэттенскому, а не евклидовому расстоянию, как в `kmeans()`. Мы прибегли

к простейшему варианту вычислений – использованию функции `pam()` из пакета `cluster`, которая выделила в качестве центральных представителей классов Алабаму, Мичиган, Оклахому и Нью-Гэмпшир.

Поскольку параметром функции `pam()` также является число кластеров k, проверим оптимальность его значения третьим из наиболее популярных методов – по *средней ширине силуэта* (average silhouette width). Для каждого найденного кластера может быть вычислена "ширина силуэта" $s_i = \frac{b(i) - a(i)}{\max[b(i), a(i)]}$, где $a(i)$ – среднее расстояние между объектами i-го кластера, $b(i)$ – среднее расстояние от объектов i-го кластера до другого кластера, самого близкого к i-му. Диаграмму силуэтов можно построить с использованием функции `fviz_silhouette()` из пакета `factoextra`:

```
fviz_silhouette(silhouette(k.pam))
```

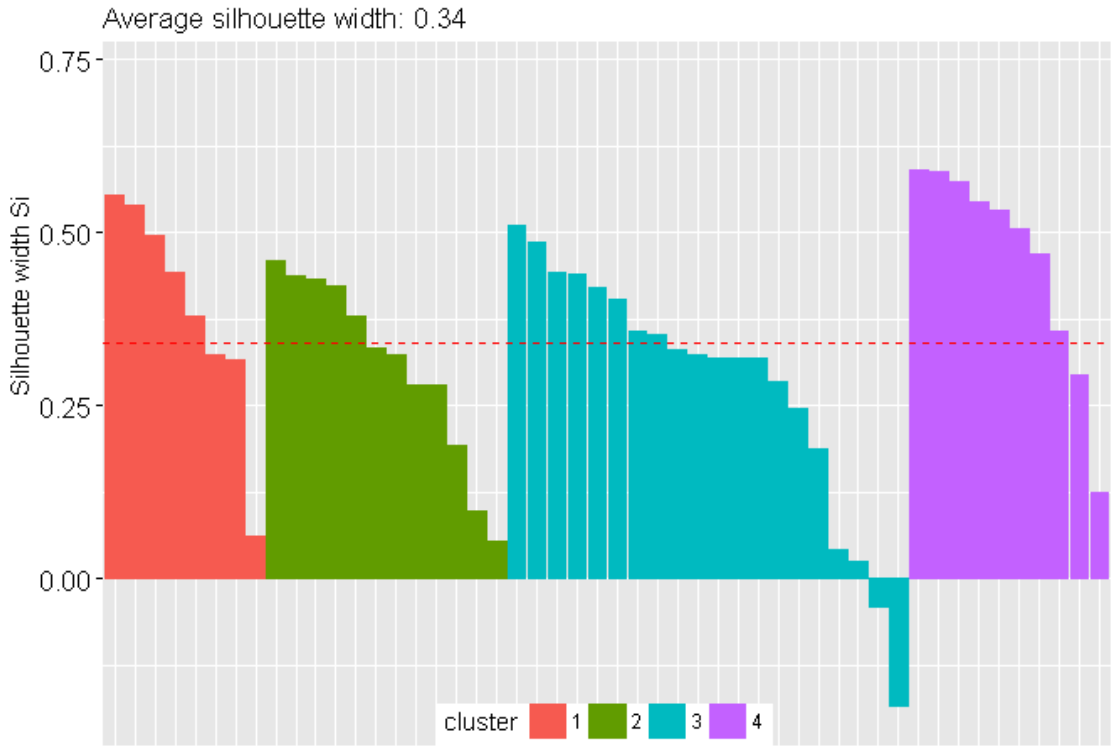


Рис. 10.3. Диаграмма силуэтов при разбиении на 4 кластера

На рис. 10.3 для каждого из 50 штатов показана разность s средних расстояний до объектов своего кластера и чужого кластера, ближайшего к анализируемому объекту. Общее среднее из этих значений S_{mean} определяет качество выполненной кластеризации. Функция `fviz_nbclust()` выполняет сканирование разбиений с различными значениями k, и строит график зависимости S_{mean} от k, подобный представленным на рис. 10.1-10.2.

```
fviz_nbclust(df.stand, pam, method = "silhouette")
```

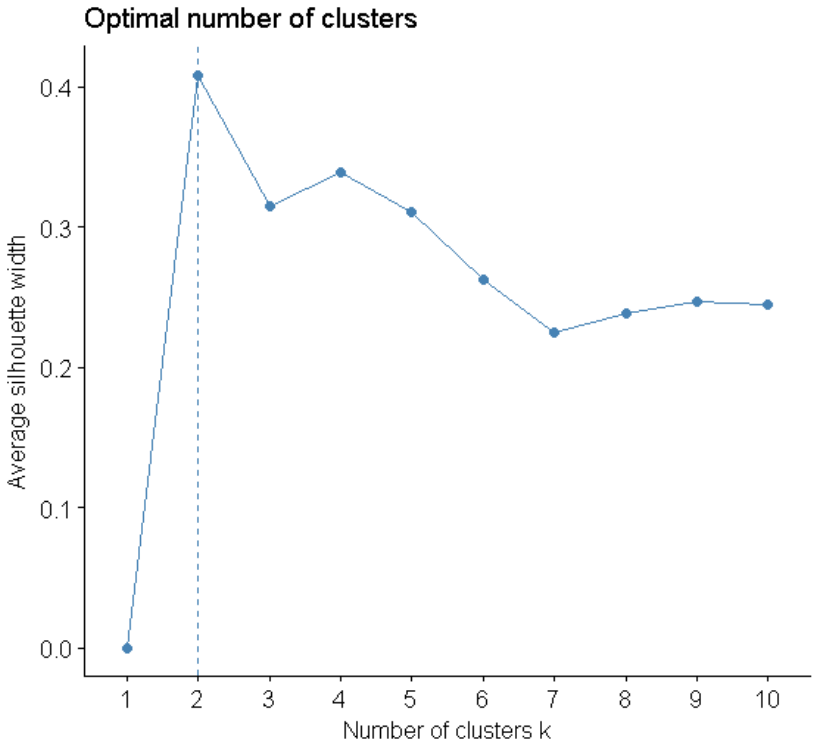


Рис. 10.4. График ширины силуэтов для выбора оптимального числа кластеров

Метод силуэтов оценил как наилучшее разбиение на два кластера $k = 2$. Отметим, что в условиях неопределенности различные алгоритмы могут порождать конкурирующие решения. Колеблющийся читатель может в таких случаях пользоваться функцией `pamk()` из пакета `fpc`, которая не требует задавать число классов, а оценивает его самостоятельно.

Алгоритм CLARA мало чем отличается от метода PAM, но приспособлен для обработки больших массивов данных (миллионы наблюдений и более). Соответствующая функция `clara()` из пакета `cluster` специально оптимизирована для сокращения времени расчетов и рационального использования оперативной памяти.

Значительный интерес представляет построение двумерных диаграмм (ординационных "биплотов") распределения наблюдений по кластерам, которые формируются с предварительным приведением исходного пространства признаков к двум главным компонентам (см. рис. 2.13 из раздела 2.6). Обычно для этого используется функция `clustplot()`, но мы предлагаем вниманию читателей функцию `fviz_cluster()` из пакета `factoextra`, которая использует для создания диаграмм графическую систему `ggplot2`.

Эта функция может быть использована для визуализации результатов по методам *k* средних, PAM, CLARA и Fanny. Ее простейший формат имеет вид:

```
fviz_cluster(object, data = NULL, stand = TRUE,
             geom = c("point", "text"),
             frame = TRUE, frame.type = "convex")
```

где:

- `object`: объект класса `"partition"`, созданный функциями `pam()`, `clara()` или `fanny()` из пакета `cluster`, или объект, возвращаемый функцией `kmeans()`;
- `data`: таблица с исходными данными для кластеризации (этот аргумент необходим только, если используется объект `kmeans`);
- `stand = TRUE`: данные стандартизуются перед выполнением анализа главных компонент;
- `geom`: три возможных комбинации стиля отображения наблюдений на графике – текстовые метки `"text"`, точки `"point"` или оба типа одновременно `c("point", "text")`;
- `frame = TRUE`: проводится контур вокруг точек для каждого кластера;
- `frame.type`: возможные значения – `"convex"` или типы эллипсов `ggplot2::stat_ellipse`, включая один из трех `c("t", "norm", "euclid")`.

Построим ординационные диаграммы для результатов кластеризации методами PAM и CLARA. Чтобы получить аккуратные графики, заменим длинные названия штатов США на их сокращенную аббревиатуру из файла St_USA.txt. Во втором случае будем использовать для построения эллипсов многомерное *t*-распределение с доверительным интервалом $\eta = 0.7$:

```
ShState <- read.delim("St_USA.txt")
rownames(df.stand) <- ShState$Short
fviz_cluster(pam(df.stand, 4), stand=FALSE)
fviz_cluster(clara(df.stand, 4), stand=FALSE,
             ellipse.type = "t", ellipse.level = 0.7)
```

Рис. 10.5. Диаграмма распределения штатов США по кластерам, полученным методом PAM

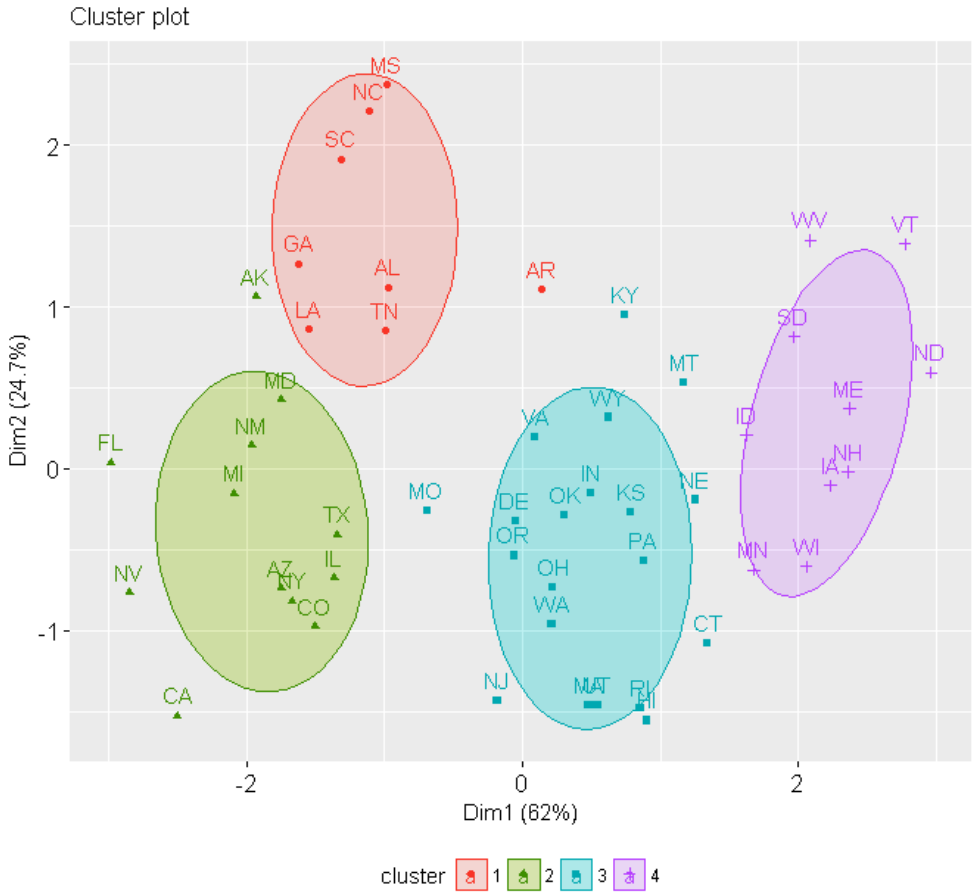


Рис. 10.6. Диаграмма распределения штатов США по кластерам, полученным методом CLARA; эллипсы построены на основе *t*-распределения ($\eta = 0.7$)

10.2. Иерархическая кластеризация

Методы иерархической кластеризации основываются на двух идеях: *агломерации* (AGNES, Agglomerative Nesting), т.е. последовательного объединения индивидуальных объектов или их групп во все более крупные подмножества, или обратном по смыслу процессе *разбиения* (DIANA, Divise Analysis), который начинается с корня и на каждом шаге делит образующие группы по степени их гетерогенности (Ким и др., 1989; Kassambara, 2017). В обоих случаях результат работы алгоритма представляет собой древовидную структуру, или *дендрограмму*.

Отдельные версии агломеративной иерархической процедуры отличаются правилами вычисления расстояния между кластерами. Например, алгоритм *средней связи* (average linkage clustering) на каждом следующем шаге объединяет два ближайших кластера, рассчитывая среднюю арифметическую дистанцию между всеми парами объектов. К алгоритму средней связи естественно сразу добавить еще два со взаимно противоположными тенденциями:

- алгоритм *одиночной связи*, или "ближайшего соседа" (single linkage clustering), когда расстояние между кластерами оценивается как минимальное

из дистанций между парами объектов, один из которых входит в первый кластер, а другой – во второй;

- и алгоритм *полной связи* или "*дальнего соседа*" (complete linkage clustering), когда вычисляется расстояние между наиболее удаленными объектами.

Более глубокий статистический смысл в понятие расстояния между кластерами вкладывается при использовании *центроидного* метода (centroid linkage clustering) и метода минимума дисперсии *Уорда* (Ward).

Утверждать, какой из методов предпочтительней, довольно просто – это тот, который чаще всего приводит к *естественной* кластеризации. Однако имеющиеся огромные расхождения в понимании столь тонкого предмета, как "естественность", вынуждают признать этот термин нечетким. Поэтому мы ограничимся в этом разделе лишь анализом сходства результатов кластеризации, полученных различными алгоритмами.

Предварительно рассмотрим функции, которые используют в R для проведения иерархической кластеризации:

```
hclust(d, method = "complete")
agnes(x, metric = "euclidean", stand = FALSE,
                                    method = "average")
diana(x, metric = "euclidean", stand = FALSE)
```

где

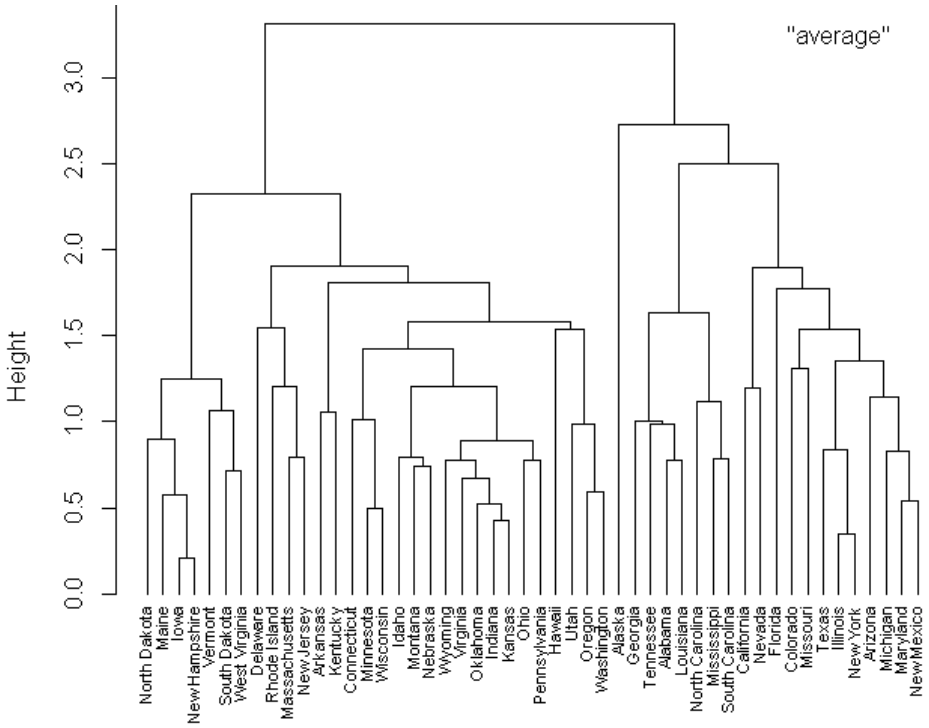
- `d`: матрица дистанций, полученная функцией `dist()` или как-то иначе;
- `method`: метод агломерации, определяемый одним из значений `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"` или `"centroid"`;
- `x`: таблица данных для вычисления расстояний (наблюдения – по строкам, признаки – по столбцам);
- `metric`: метрика расстояний между наблюдениями, т.е. `"euclidean"` или `"manhattan"`;
- `stand = TRUE`: осуществляется стандартизация каждой переменной таблицы (по столбцам).

Очевидно, что агломерационные процедуры `hclust()` и `agnes()` приведут к одному и тому же результату, а `diana()` выполнит алгоритм последовательного разбиения.

Приведем различные варианты формирования дендрограмм и способы их визуализации на примере кластеризации штатов США по криминогенной обстановке. Первые три диаграммы на рис. 10.7 построены с использованием базовой функции `plot.hclust()`, имеющей ограниченный набор настраиваемых параметров. Для варианта `"complete"` показан пример разрезания дерева на 4 группы, которые на дендрограмме можно показать в виде прямоугольников.

Четвертый рисунок получен с помощью функции `plot.dendrogram()` , которая имеет такие дополнительные параметры как `nodePar` – спецификацию графического изображения узлов дерева, и `edgePar` – список графических параметров для представления ветвей разными цветами.

```
library(cluster)
data("USArrests")
d <- dist(scale(USArrests), method = "euclidean")
# Параметр hang=-1  выравнивает метки
plot(hclust(d, method = "average" ), cex = 0.7, hang=-1)
```




```
plot(hclust(d, method = "single" ), cex = 0.7)
```

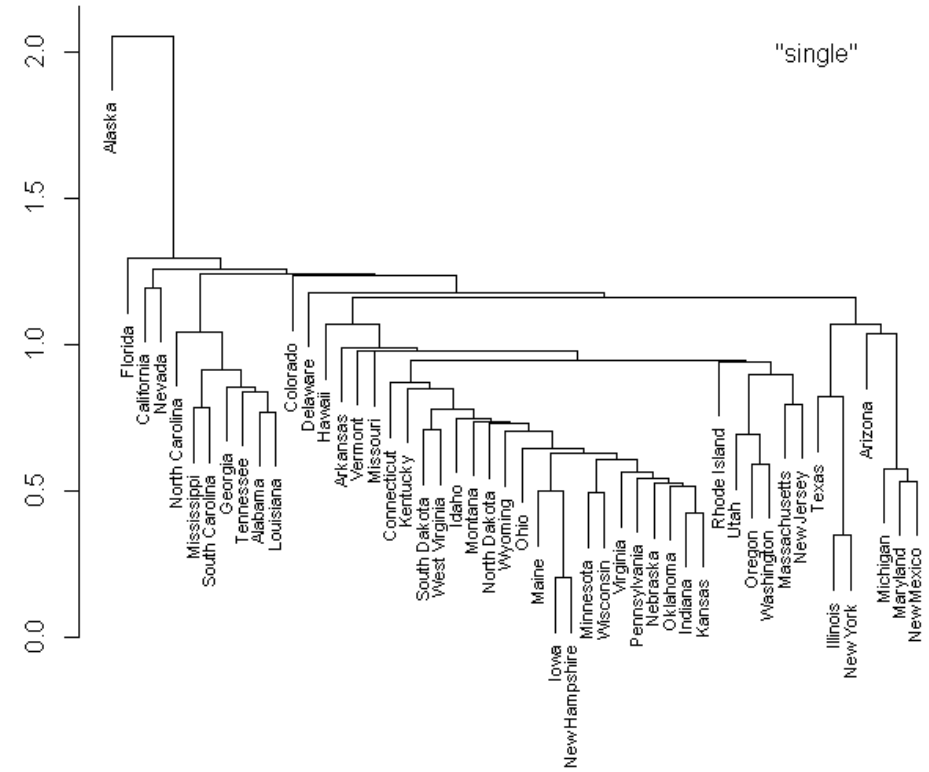
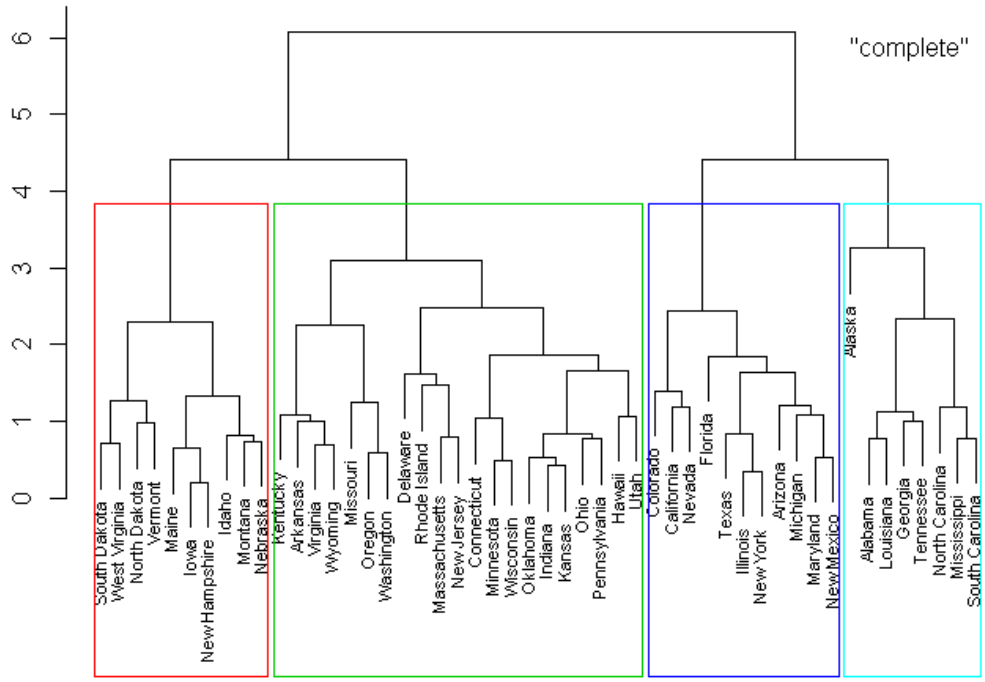
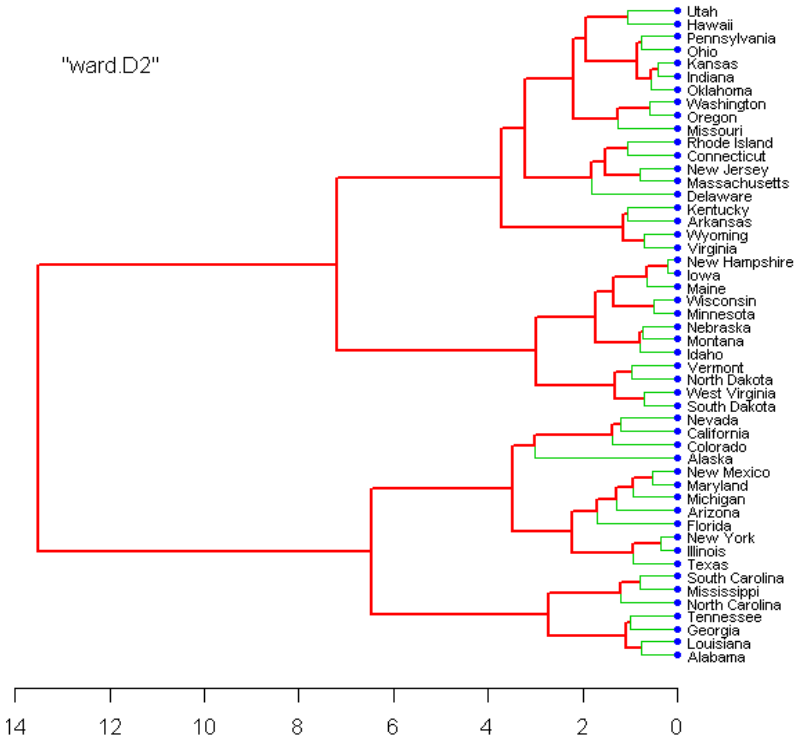


Рис. 10.7. Дендрограмма распределения штатов США по кластерам; `"average"` – метод средней связи, `"single"` – метод одиночной связи

```
res.hc <- hclust(d, method = "complete" )
grp <- cutree(res.hc, k = 4)  # Разрезание дерева на 4 группы
plot(res.hc, cex = 0.7)
rect.hclust(res.hc, k = 4, border = 2:5)
```




```
hcd <- as.dendrogram(hclust(d, method = "ward.D2" ))
nodePar <- list(lab.cex = 0.7, pch = c(NA, 19),
                cex = 0.7, col = "blue")
plot(hcd,  xlab = "Height", nodePar = nodePar, horiz = TRUE,
                edgePar = list(col = 2:3, lwd = 2:1))
```



Продолжение рис. 10.7. Дендрограмма распределения штатов США по кластерам; `"complete"` – метод полной связи, `"ward.D2"` – метод Уорда

Дополнительные возможности для графического представления дендрограмм можно найти в функциях пакетов ape (анализ филогенетики и эволюции) и ggdendro, использующего графическую систему ggplot2.

Визуальные впечатления от сравнения различных дендрограмм на рис. 10.7 и опыт специалистов свидетельствует о том, что метод Уорда и полной связи дают весьма сходные результаты (к ним приближается также метод средней связи). Метод одиночной связи и центроидный метод могут дать иногда весьма специфические дендрограммы. Однако для анализа сходства дендрограмм предпочтительнее использовать пакет dendextend, который содержит несколько функций для сравнения результатов кластеризации, включая: dend_diff(), tanglegram(), entanglement(), all.equal.dendrogram(), cor.dendlist(). Выполним построение диаграммы типа "клубок", показывающей различия двух дендрограмм:

```
library(dendextend)
# Вычисляем 2 иерархические кластеризации
hc1 <- hclust(d, method = "average")
hc2 <- hclust(d, method = "ward.D2")
# Создаем две дендрограммы
dend1 <- as.dendrogram (hc1); dend2 <- as.dendrogram (hc2)
# На диаграмме типа "клубок" показываем цветом общие ветви
tanglegram(dend1, dend2,
  common_subtrees_color_branches = TRUE)
```

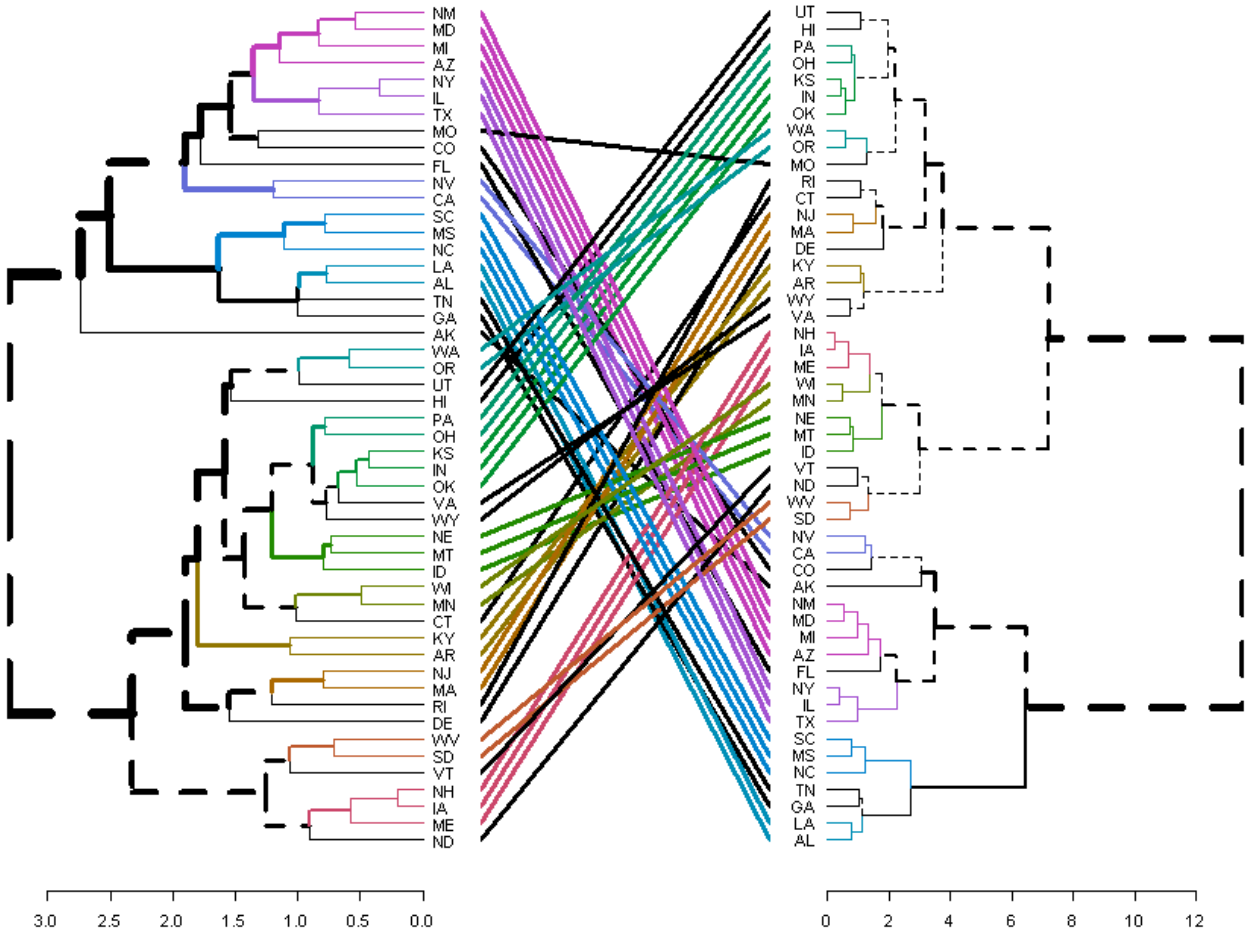


Рис. 10.8. Диаграмма сравнения деревьев кластеризации "average"-"ward"

Для количественной оценки различий между вариантами кластеризации могут быть использованы кофенетическая корреляция (cophenetic correlation) или различные ранговые индексы. Кофенетическая дистанция определяет расстояние между двумя объектами на дендрограмме и отражает уровень внутригрупповых различий, при котором эти объекты были впервые объединены в один кластер (Ким и др., 1989). Все возможные пары таких дистанций образуют апостериорную матрицу расстояний между объектами по результатам кластеризации. Кофенетическая корреляция вычисляется как линейный коэффициент корреляции Пирсона (или ранговый коэффициент Спирмена) между всеми элементами двух таких матриц:

```
c(cor_cophenetic(dend1, dend2), # кофенетическая корреляция
cor_bakers_gamma(dend1, dend2))  # корреляция Бейкера
```

```
[1] 0.8431430 0.8400675
```

Аналогично можно получить матрицу коэффициентов кофенетической корреляции для всех обсуждаемых методов иерархической кластеризации:

```
# Создаем множество дендрограмм для сравнения
dend1 <- d %>% hclust("com") %>% as.dendrogram
dend2 <- d %>% hclust("single") %>% as.dendrogram
dend3 <- d %>% hclust("ave") %>% as.dendrogram
dend4 <- d %>% hclust("centroid") %>% as.dendrogram
dend5 <- d %>% hclust("ward.D2") %>% as.dendrogram
# Создаем список дендрограмм и корреляционную матрицу
dend_list <- dendlist("Complete" = dend1, "Single" = dend2,
     "Average" = dend3, "Centroid" = dend4, "Ward.D2" = dend5)
cors <- cor.dendlist(dend_list) ; round(cors, 2)
library(corrplot) # изображение корреляционной матрицы
corrplot(cors, "pie", "lower")
```

```
          Complete Single Average Centroid Ward.D2
Complete      1.00   0.50    0.86     0.49    0.94
Single        0.50   1.00    0.58     0.71    0.49
Average       0.86   0.58    1.00     0.61    0.84
Centroid      0.49   0.71    0.61     1.00    0.49
Ward.D2       0.94   0.49    0.84     0.49    1.00
```

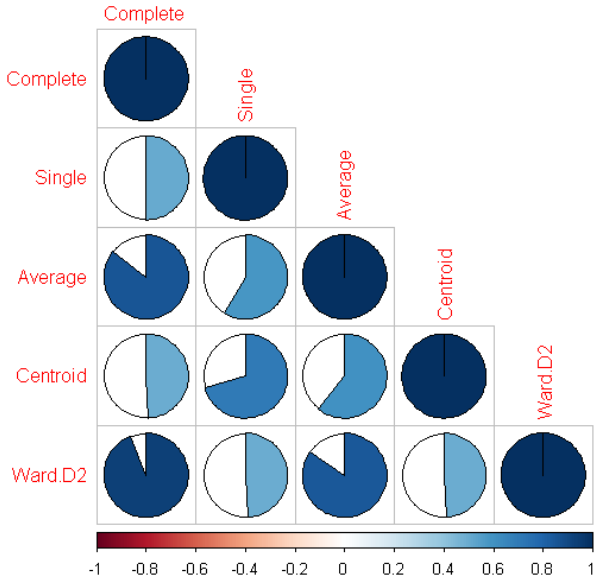


Рис. 10.9. Корреляция между кластеризациями на основе разных методов

10.3. Оценка качества кластеризации

После создания кластерного решения обычно возникает вопрос, насколько оно устойчиво и статистически значимо. Здесь существует эмпирическое правило – устойчивая группировка должна сохраняться при изменении методов кластеризации: например, если результаты иерархического кластерного анализа имеют долю совпадений более 70% с группировкой по методу *k* средних, то предположение об устойчивости принимается.

В теоретическом плане проблема проверки адекватности кластеризации не решена, по крайней мере, без использования другого вида анализа или априорного знания принадлежности объектов к соответствующим классам. Авторы сборника Ким и др. (1989, с. 192 и далее) подробно рассматривают и в итоге отвергают пять методов проверки адекватности кластеризации (ru.wikipedia.org/wiki):

1. Кофенетическая корреляция – не рекомендуется и ограниченна в использовании;
2. Тесты на значимость разбиения данных на кластеры (многомерный дисперсионный анализ) – всегда дают значимый результат;
3. Методика повторных (случайных) выборок – не доказывает обоснованность решения;
4. Тесты значимости для признаков, не использованных при кластеризации, – пригодны только при наличии повторных измерений;
5. Методы Монте-Карло очень сложны и доступны только опытным математикам.

С тех пор в литературе предложено множество методов и критериев оценки качества результатов кластеризации (clustering validation). Можно выделить несколько подходов к валидации[5] кластеров (Kassambara, 2017):

- *внешняя валидация*, которая заключается в сравнении итогов кластерного анализа с заранее известным результатом (т.е. метки кластеров известны априори);
- *относительная валидация*, которая оценивает структуру кластеров, изменяя различные параметры одного и того же алгоритма (например, число групп *k*);
- *внутренняя валидация*, которая использует внутреннюю информацию процесса объединения в кластеры (если внешняя информация отсутствует);
- *оценка стабильности* объединения в кластеры (или специальная версия внутренней валидации), использующая методы ресэмплинга.

Одна из проблем машинного обучения без учителя состоит в том, что методы кластеризации будут формировать группы, даже если анализируемый набор данных представляет собой полностью случайную структуру. Поэтому первой задачей валидации, которую рекомендуется выполнить перед началом

---

[5] Мы используем термин "валидность", чтобы избегнуть некоторой двусмысленности таких понятий как "адекватность", "эффективность", "значимость"

кластерного анализа, является оценка общей предрасположенности имеющихся данных к объединению в кластеры (clustering tendency).

Статистика Хопкинса (Hopkins) является одним из индикаторов тенденции к группированию. Для ее расчета создается B псевдо-наборов данных, сгенерированных случайным образом на основе распределения с тем же стандартным отклонением, что и оригинальный набор данных. Для каждого наблюдения i из n рассчитывается среднее расстояние до k ближайших соседей: w_i между реальными объектами и q_i между искусственными объектами и их самыми близкими реальными соседями. Тогда статистика Хопкинса

$$H_{ind} = \sum_n w_i / (\sum_n q_i + \sum_n w_i),$$

превышающая 0.5, будет соответствовать нулевой гипотезе о том, что q_i и w_i подобны, а группируемые объекты распределены случайно и однородно. Величина $H_{ind} < 0.25$ на 90%-ном уровне уверенности указывает на имеющуюся тенденцию к группированию данных.

Весьма полезна также визуальная оценка тенденции (VAT, Visual Assessment of cluster Tendency): потенциальные группы представлены темными квадратами вдоль главной диагонали "VAT-диаграммы". Функция `get_clust_tendency()` из пакета `factoextra` попутно с графиком рассчитывает также и статистику Хопкинса:

```
library(cluster)  ; data("USArrests")
df.stand <- as.data.frame(scale(USArrests))
library("factoextra")
get_clust_tendency(df.stand, n = 30,
     gradient = list(low = "steelblue", high = "white"))
```

```
$hopkins_stat
[1] 0.386467
```

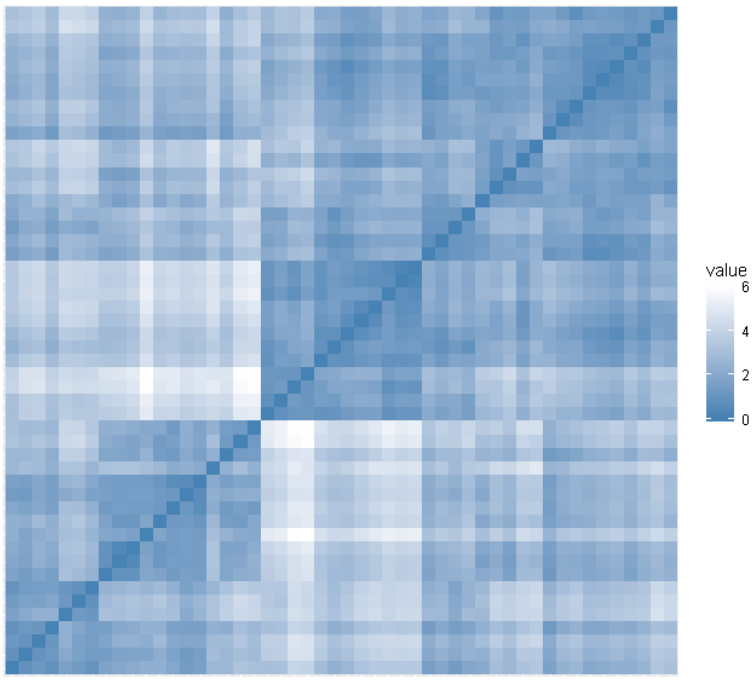


Рис. 10.10. Визуальная оценка тенденции данных к группированию

В случае криминогенности штатов США мы имеем весьма умеренную склонность к образованию групп.

В разделе 10.1 мы рассмотрели три метода выбора оптимального числа кластеров, которые не всегда давали однозначные оценки. «Но это еще не все…». С помощью пакета `NbClust` можно найти оптимальную схему объединения в кластеры, используя ни много ни мало целых 30 индексов качества! При этом происходит перебор различных комбинаций числа групп, метрик дистанции и методов кластеризации:

```
library(NbClust)
nb <- NbClust(df.stand, distance = "euclidean", min.nc = 2,
        max.nc = 8, method = "average", index ="all")
nb$Best.nc
fviz_nbclust(nb) + theme_minimal()
```

```
                    KL      CH Hartigan    CCC   Scott  Marriot   TrCovW
Number_clusters  8.0000  2.000   5.0000  2.000  5.0000      5.0   4.0000
Value_Index     24.0818 43.462  12.3077 -0.141 43.9763 180497.5 383.4582
                 TraceW Friedman   Rubin Cindex     DB Silhouette   Duda
Number_clusters  5.0000   6.0000  5.0000 4.0000 4.0000     2.0000 2.0000
Value_Index     15.3518   2.8394 -0.5185 0.3713 0.7784     0.4085 1.0397
                PseudoT2   Beale Ratkowsky    Ball PtBiserial Frey
Number_clusters   2.0000  2.0000    2.0000  3.0000     3.0000    1
Value_Index      -0.6878 -0.0874    0.4466 18.8709     0.6361   NA
                McClain   Dunn Hubert SDindex Dindex   SDbw
Number_clusters   2.000 8.0000      0  4.0000      0 8.0000
Value_Index       0.547 0.2509      0  1.2638      0 0.2786
```

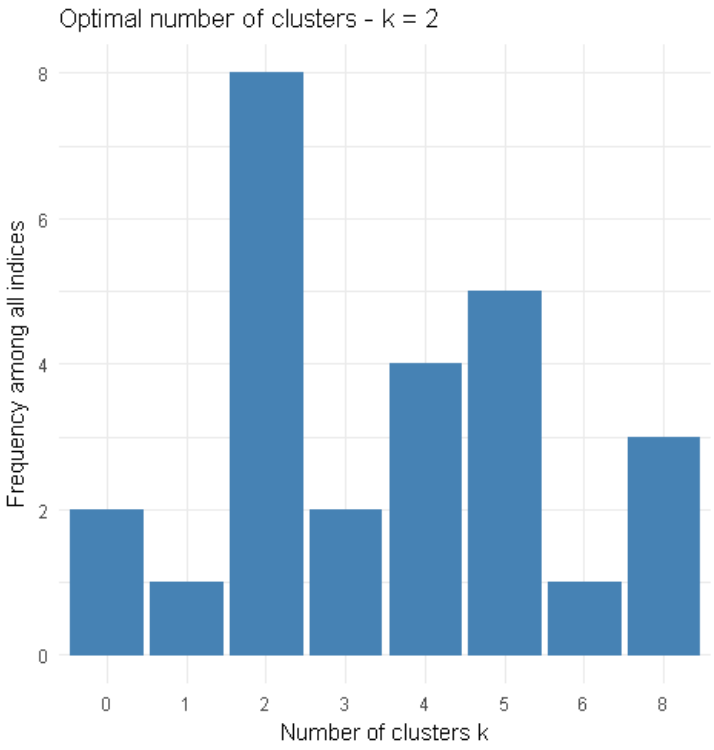


Рис. 10.11. Оптимальное число кластеров по оценкам различных индексов

Разброс оценок числа классов наилучшего разбиения весьма велик: от 2 по индексу МакКлайна до 8 по индексу Данна, поэтому приходится прибегать к тривиальному голосованию.

Кофенетическую корреляцию можно также рассчитать между исходной матрицей дистанции и матрицей кофенетических расстояний, и тогда она может служить мерой адекватности кластерного решения исходным данным (Borcard et al., 2011). Оценим по этому показателю пять иерархических кластеризаций, сравниваемых между собой в предыдущем

```
d <- dist(df.stand, method = "euclidean")
library(vegan)
hc_list <- list(hc1 <- hclust(d,"com"),
hc2 <-  hclust(d,"single"), hc3 <-  hclust(d,"ave"),
hc4 <-  hclust(d, "centroid"),hc5 <-   hclust(d, "ward.D2"))
Coph <- rbind(
MantelStat <- unlist(lapply(hc_list,
     function (hc) mantel(d, cophenetic(hc))$statistic)),
MantelP <- unlist(lapply(hc_list,
     function (hc) mantel(d, cophenetic(hc))$signif)))
colnames(Coph) <- c("Complete", "Single","Average",
     "Centroid","Ward.D2")
rownames(Coph) <- c("W Мантеля","P-значение")
round(Coph, 3)
```

разделе:

```
             Complete Single Average Centroid Ward.D2
W Мантеля       0.698  0.541   0.718    0.607   0.698
P-значение      0.001  0.001   0.001    0.001   0.001
```

Таким образом, максимальное значение коэффициента *W* матричной корреляции Мантеля (а, следовательно, и наибольшая адекватность матрице расстояний, построенной по исходным данным) принадлежит кластеризации по методу средней связи. Нелишне заметить, что все рассмотренные кластеризации статистически значимы, т.е. не могут быть объяснены случайными причинами (впрочем, такое будет почти всегда). Наконец, Ким и др. (1989) не рекомендуют использовать кофенетическую корреляцию в основном по причине несоответствия кофенетических расстояний нормальному распределению. Однако с развитием методов ресэмплинга появилась мощная аргументация, показывающая откровенную слабость подобных утверждений.

После того, как мы попытались оценить, насколько хорошо топология дендрограммы отображает предрасположенность объектов к группированию, остаются некоторые вопросы, вызывающие несомненный интерес. Можно ли вычислить *p*-значения в целом для полученной иерархической кластеризации? Какие фрагменты древовидной структуры являются "слабым звеном" в полученной конструкции?

Однако правомерен и контрвопрос: «Нужно ли вычислять эти *p*-значения?» Как убедительно сказано в книге Джеймс и др. (2016), не существует правильных или неправильных результатов кластеризации, поскольку, по определению, это метод обучения без учителя. Все определяется соответствием полученного решения поставленной задаче, которая на практике в большинстве случаев сводится просто к тому, чтобы *приблизительно* оценить, на сколько групп целесообразно разделить данные. При этом степень этого соответствия всегда будет субъективной.

Ответ на второй вопрос об устойчивости фрагментов кластерной структуры состоит в том, чтобы взять из исходной таблицы множество повторных выборок, построить для каждой из них свою дендрограмму и вычислить частоту встречаемости каждого фрагмента в сформированной последовательности разбиений. Разумеется, здесь невозможно обойтись без бутстрепа, позволяющего подсчитать вероятность BP (Bootstrap Probability) встречаемости произвольного узла в бутстреп-копиях. Обычно фрагменты древовидной структуры считаются статистически значимыми, если с ветвями дерева связывается бутстреп-вероятность, превышающая 70-80%.

Х. Шимодейра (Shimodaira, 2002), сравнивая центры распределения исходной и бутстреп-выборок, показал, что величина BP является приближенной оценкой вероятности появления узла в дереве. Несмещенную оценку вероятности AU (Approximately Unbiased) можно получить, выполнив повторную серию бутстрепа в различных масштабах (multiscale bootstrap resampling). Для этого отдельно вычисляют BP-значения, формируя бутстреп-выборки разного объема: например, 0.5n, 0.6n, …, 1.4n, 1.5n, где n – объем исходной выборки. Несмещенная бутстреп-вероятность AU находится аппроксимацией ряда полученных значений BP. Оптимальные оценки AU для каждого кластера дендрограммы, найденные путем подбора параметрических моделей с использованием метода максимального правдоподобия, могут быть получены с использованием пакетов `pvclust` и `scaleboot` для R.

Некоторая проблема заключается в том, что функция `pvclust()`, ориентированная на генетические исследования, выполняет кластеризацию признаков (т.е. столбцов таблицы данных), а для 4 показателей нашей демонстрационной таблицы `USArrests` это особого интереса не представляет. Если попробовать выполнить анализ с транспонированной матрицей 4×60, выполнив команду `pvclust(t(USArrests))`, то решение найти нельзя из-за проблем с сингулярными преобразованиями:

```
Error in solve.default(crossprod(X, X/vv)),
system is exactly singular: U[2,2] = 0
```

Воспользуемся тогда в качестве примера набором `Boston` из пакета `MASS`, включающим 14 признаков привлекательности 506 участков города для проживания:

```
data(Boston, package = "MASS")
library(pvclust)
set.seed(123)
#  Бутстреп деревьев и расчет BP- и AU- вероятностей для узлов
boston.pv <- pvclust(Boston, nboot=100, method.dist="cor",
                     method.hclust="average")
plot(boston.pv)  # дендрограмма с p-значениями
pvrect(boston.pv) # выделение боксами достоверных фрагментов
```

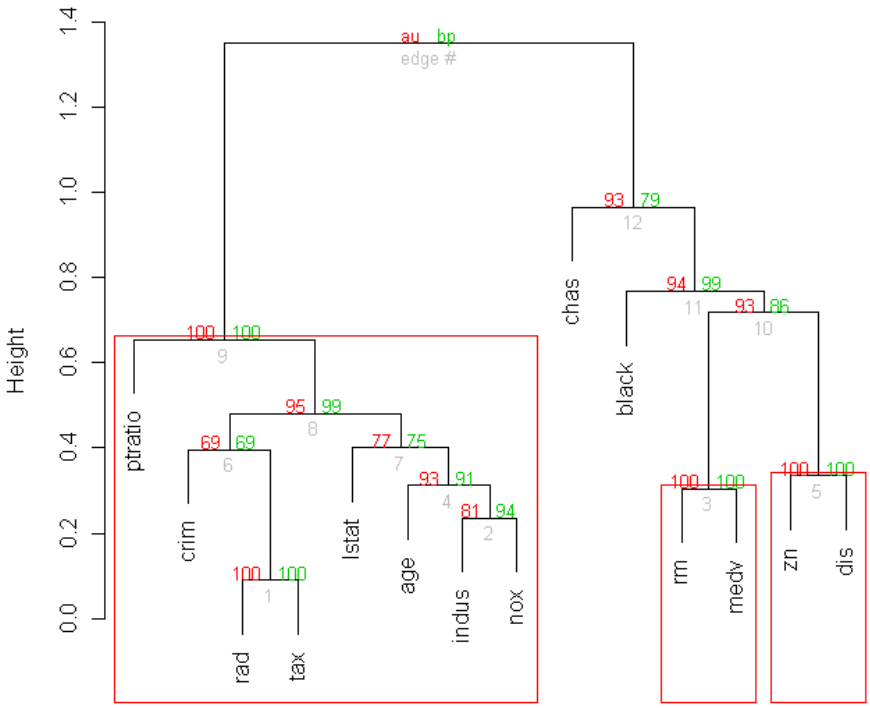


Рис. 10.12. Дендрограмма с нанесенными значениями AU/BP для каждого узла

Этот пример показывает, что с практической точки зрения кластеризация признаков (переменных) может быть столь же важна, как и группировка объектов. Использование коэффициентов корреляции для расчета матрицы дистанций `method.dist="cor"` избавляет нас от необходимости стандартизовать данные.

При анализе дендрограммы на рис. 10.12 можно увидеть, что признаки иногда образуют кластеры с ясно интерпретируемой зависимостью ("среднее число комнат в жилье `rm`" и "медианная стоимость дома `medv`"), но часто их тесная связь нуждается в дополнительном осмыслении ("индекс доступности к кольцевым дорогам `rad`" и "сумма налога на недвижимость `tax`", или "доля участков, продаваемых в розницу `indus`" и "концентрация окислов азота `nox`"). В целом была получена весьма стабильная кластеризация: довольно низкую бутстреп-вероятность имеют такие важнейшие признаки, как "криминальный индекс `crim`" и "процент жителей с низким социальным статусом `lstat`", которые вполне могут объединиться в группу с любым другим из имеющихся показателей (например, "долей афроамериканцев `black`").

10.4. Другие алгоритмы кластеризации

Иерархическая кластеризация на главные компоненты

Использование главных компонент для иерархической кластеризации – один из возможных путей гибридизации алгоритмов, которая все чаще привлекает внимание специалистов. Предварительное сжатие пространства признаков – хороший метод сглаживания случайных флуктуаций в данных,

что является предпосылкой построения более стабильных кластерных структур. Это особенно важно для матриц с большим количеством признаков, например, в генной инженерии.

В среде R кластеризация на главные компоненты реализована в пакете `FactoMineR` и состоит из двух шагов. На первом этапе функцией `PCA()` выполняется обычный анализ главных компонент и выбирается их число. Далее функция `HCPC()` использует эти результаты и строит иерархическую кластеризацию. При этом используется метод Уорда, который, как и анализ главных компонент, основан на анализе многомерной дисперсии (inertia).

Опять воспользуемся в качестве примера набором `Boston` из пакета `MASS`. Выполним снижение исходной размерности данных, определяемой 14 признаками привлекательности земельных участков Бостона, до 5 главных компонент:

```
library(FactoMineR) ; library(factoextra)
data(Boston, package = "MASS")
df.scale <- scale(Boston)
res.pca <- PCA(df.scale, ncp = 5, graph=TRUE)
get_eig(res.pca)
```

```
       eigenvalue variance.percent cumulative.variance.percent
Dim.1  6.54598958       46.7570684                     46.75707
Dim.2  1.64953191       11.7823708                     58.53944
Dim.3  1.34890592        9.6350423                     68.17448
Dim.4  0.88653987        6.3324276                     74.50691
Dim.5  0.85089944        6.0778531                     80.58476
```

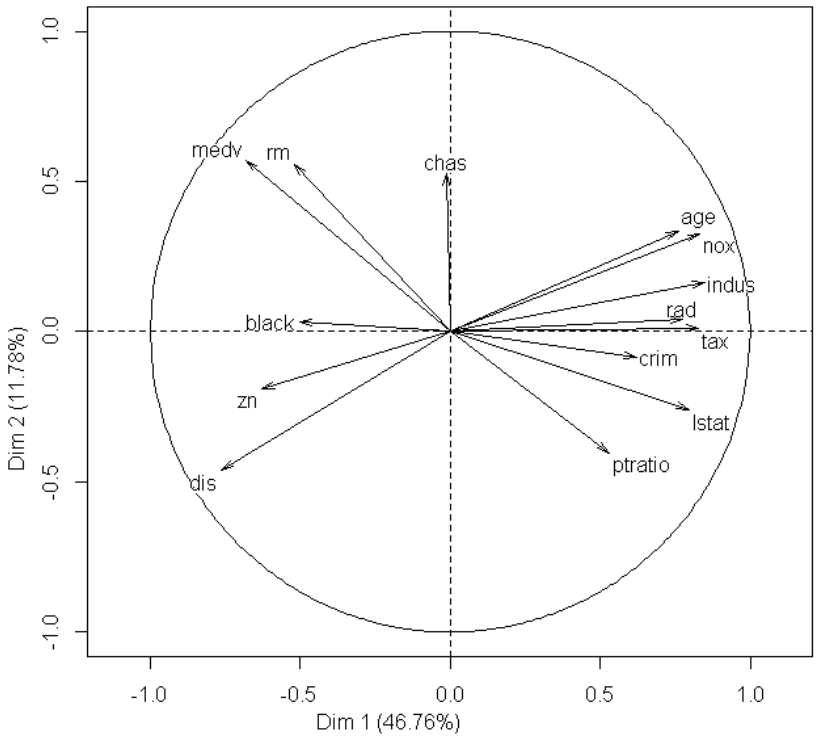


Рис. 10.13. Разложение исходных признаков по осям двух главных компонент

Напомним, что угол между двумя любыми осями (факторов и/или компонент) на рис. 10.13 соответствует уровню корреляции между ними.

Заинтересованный читатель может сравнить этот график с иерархической дендрограммой признаков на рис. 10.12.

Выполним теперь иерархическую кластеризацию на главные компоненты. Построим две диаграммы – обычную дендрограмму, совмещенную с графиком "каменистой осыпи" дисперсии (`choice ="tree"`), и трехмерную дендрограмму, совмещенную с ординационной диаграммой (`choice = "3D.map"`):

```
res.hcpc <- HCPC(res.pca, graph = TRUE)
plot(res.hcpc, choice ="tree")
plot(res.hcpc, choice = "3D.map", ind.names=FALSE)
```

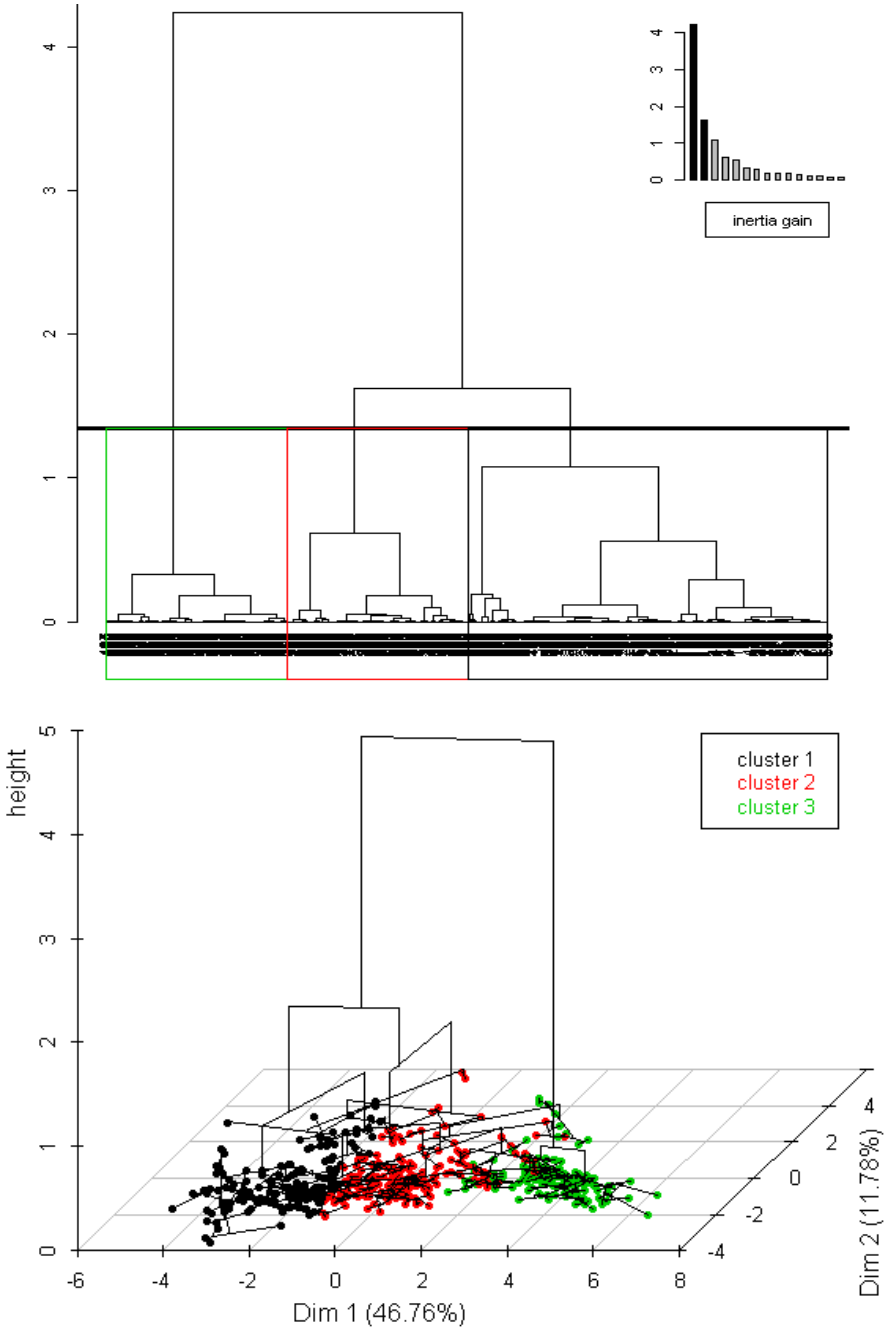


Рис. 10.14. Дендрограмма с разделением на кластеры (а) и гибрид дендрограммы с ординационной диаграммы (б)

Метод нечетких k-средних (*fuzzy analysis clustering*)

Традиционные принципы кластерного анализа предполагают, что выделяемые группы представляют собой детерминированные совокупности, т. е. каждый объект может принадлежать только к одному таксону. Ограниченность такого подхода часто приводит к аналитической неопределенности, и поэтому разумной альтернативой понятию абсолютной дискретности является интерпретация компонентов систем как нечетких объектов в составе гибко настраиваемых ординационных структур.

В конце 80-х годов, после того, как основные концепции нечетких множеств (fuzzy sets) были разработаны Лотфи Заде (Zadeh, 1965), началось бурное развитие технических устройств на базе нечетких контроллеров, а нечеткая логика (fuzzy logic) стала неотъемлемой составной частью современных систем искусственного интеллекта.

Множество **C** является нечетким, если существует функция принадлежности (membership function) $\mu_C(x)$, принимающая на этом множестве значения в интервале [0, 1], где $\mu_C(x) = 0$ означает полную несовместимость, т. е. $x \notin \mathbf{C}$, а $\mu_C(x) = 1$ означает полную принадлежность, или $x \in \mathbf{C}$. Применительно к кластеризации методом нечетких *k* средних функция $\mu_{ir}(\mathbf{x})$ задает в масштабе от 0 до 1 степень принадлежности каждого объекта $\mathbf{x}_i$ к каждому выделяемому кластеру *r* (Bezdek, 1981).

Пусть $D_{ir} = \sum_j (x_{ij} - v_{rj})^2$ – расстояние между каждым *i*-м объектом (i = 1, 2, …, n), описанным набором признаков x_{ij}, и центрами тяжести v_{rj} каждого из k кластера (r = 1, 2, …, k). Тогда в общем случае кластеризацию объектов **X** можно сформулировать как задачу оптимизации, связанную с нахождением такой матрицы $\boldsymbol{\mu}$, которая минимизировала бы критерий

$$F_{km}(\boldsymbol{\mu}) = \sum_{i=1}^{n} \sum_{r=1}^{k} \mu_{ir}^{m} D_{ir}$$

Экспоненциальный вес (m) в алгоритме нечетких k средних задает уровень нечеткости получаемых кластеров: чем больше m, тем нечеткое разбиение более "размазано". "Штатный" диапазон варьирования m – от 1.2 до 2. Другим важным параметром является количество классов k, которое принимается из описанных выше соображений.

Выполним построение "нечетких" кластеров для традиционного примера по криминогенной обстановке американских штатов с использованием функции `fanny()` из пакета `cluster`:

```
library(cluster)
data("USArrests")
set.seed(123)
res.fanny <- fanny(USArrests, k = 4, memb.exp = 1.7,
      metric = "euclidean", stand = TRUE, maxit = 500)
print(head(res.fanny$membership),3)
res.fanny$coeff
```

```
Membership coefficients
             [,1]  [,2]  [,3]   [,4]
Alabama    0.639 0.183 0.113 0.0645
Alaska     0.337 0.357 0.179 0.1280
Arizona    0.213 0.595 0.131 0.0611
Arkansas   0.371 0.170 0.264 0.1959
California 0.224 0.529 0.160 0.0869
Colorado   0.224 0.519 0.174 0.0830

Fuzzyness coefficients:
dunn_coeff normalized
 0.3927126  0.1902834
```

При k = 4 в результате выводится матрица коэффициентов принадлежности, максимальный из которых определяет назначаемый кластер. Так из приведенного фрагмента Алабама и Арканзас включены в кластер 1, а остальные четыре штата – в кластер 2.

Для оценки меры нечеткости полученной классификации используется коэффициент разделения Данна (Dunn):

$$F_k = \sum_{i=1}^{n} \sum_{r=1}^{k} \mu_{ir}^2 / k ,$$

который принимает минимальное значение при полной нечеткости разбиения, когда расстояния от каждого объекта до центра тяжести любого кластера равновелики: $\mu = 1/k$. Напротив, в случае четкой кластеризации ($\mu = 1$ или 0) коэффициент Данна F_k принимает значение 1. Для представленного примера F_k = 0.39, а его нормированная версия, изменяющаяся от 0 до 1 и характеризующая степень нечеткости $F_k' = (kF_k - 1)/(k-1)$ = 0.19.

Построим две диаграммы (рис. 10.15). На первой из них (а) покажем матрицу $\sum_{r=1}^{k} \mu_{ir}^2 / k$, отсортированную по убыванию (т.е. по степени нечеткости). На второй же (б) представим ординационную диаграмму в пространстве двух главных компонент с результатами кластеризации (аналогична рис. 10.6).

```
# Визуализация с использованием corrplot
library(corrplot)
Dunn <- res.fanny$membership^2
corrplot(Dunn[rev(order(rowSums(Dunn))),], is.corr = FALSE)
# Ординационная диаграмма
library(factoextra)
fviz_cluster(res.fanny, frame.type = "norm",
             frame.level = 0.7)
```

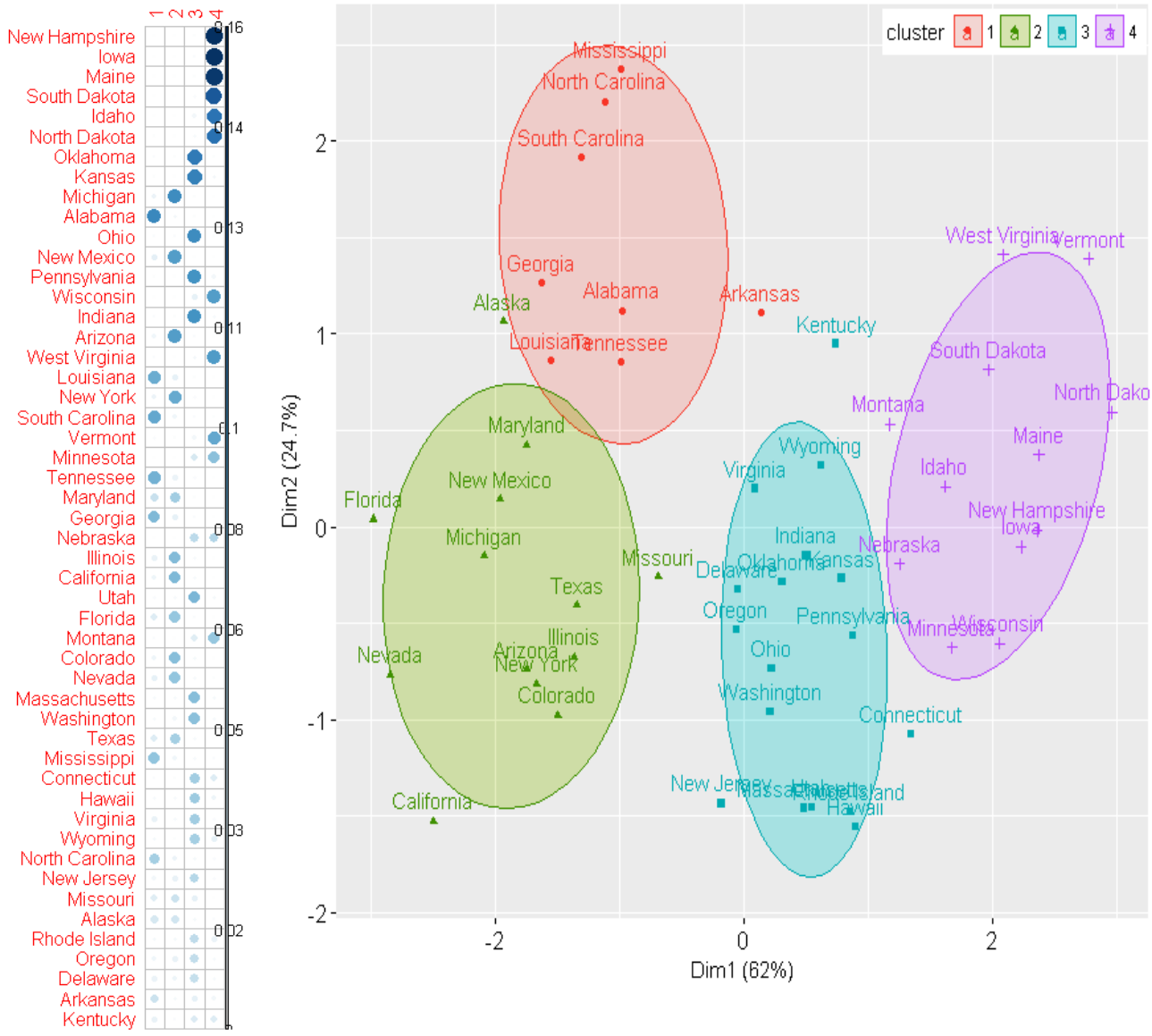


Рис. 10.15. Матрица степени нечеткости (а) и ординационная диаграмма (б), полученные с применением метода нечеткой кластеризации

Статистическая модель кластеризации

Важнейшими свойствами кластеров являются плотность, дисперсия, размер, форма и отделимость (Ким и др., 1989). Характер изменчивости облака экспериментальных точек относительно центроидов искомых кластеров может быть описан статистической моделью со смешанными параметрами (Banfield, Raftery, 1993).

Предположим, что входное множество векторов наблюдений $x_1, x_2, ..., x_n$ представляет собой случайные реализации из K неизвестных распределений $E_1, E_2, ..., E_K$. Допустим, что плотность распределения x_i, связанная с E_k, задана функцией $f_k(x_i, \theta)$, включающей некоторое неизвестное множество параметров θ. Если принять допущение, что классифицируемый объект принадлежит только одному распределению, то можно ввести переменную $\gamma_I = k$, если x_i принадлежит E_k, $k = 1, 2,\ldots, K$. Тогда цель оптимизации модели

заключается в нахождении таких параметров $\boldsymbol{\theta}$ и $\boldsymbol{\gamma}$, которые максимизируют совокупную вероятность: $L(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \prod_{i=1}^{n} f(\gamma_i, x_i \mid \theta)$.

Поскольку вероятность $L(\boldsymbol{\theta}, \boldsymbol{\gamma})$ основана на смеси K распределений, то такая модель называется смешанной (mixture model). Если использовать в качестве распределений многомерные нормальные функции, то неизвестные параметры $\boldsymbol{\theta}$ определяются как вектор средних μ_k и матрица ковариации $\sum_k$ каждого k-го распределения.

Если рассматривать изложенный формализм смешанных моделей применительно к кластеризации (model-based clustering), то каждой группе k соответствует центроид μ_k с повышенной плотностью точек в его окрестностях. Геометрические особенности (форма, объем, ориентация) каждого кластера определены матрицей ковариации $\sum_k$. Главным преимуществом этого подхода, по сравнению с другими методами кластеризации, является возможность автоматической оценки оптимального числа кластеров в процессе подгонки параметров модели.

Выборочные оценки параметров θ могут быть получены с использованием алгоритма *максимизации математических ожиданий* (EM, Expectation-Maximization), который оценивает максимумы вероятности $L(\boldsymbol{\theta}, \boldsymbol{\gamma})$ серии моделей: для заданного диапазона значений k и с различной параметризацией матрицы ковариации. Оптимальная модель обычно отбирается на основе максимума байесовского информационного критерия BIC.

Рассмотрим двумерный набор данных, который содержит время ожидания между извержениями (`waiting`) и продолжительностью извержения (`eruptions`) гейзера "Старый служака" в Йеллоустонском национальном парке. Для иллюстрации характера распределения наблюдаемых данных изобразим диаграмму диаграмму рассеяния (рис. 10.16):

```
data("faithful")
head(faithful, 3)
library("ggplot2")
ggplot(faithful, aes(x=eruptions, y=waiting)) +
  geom_point() + geom_density2d()
```

```
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
```

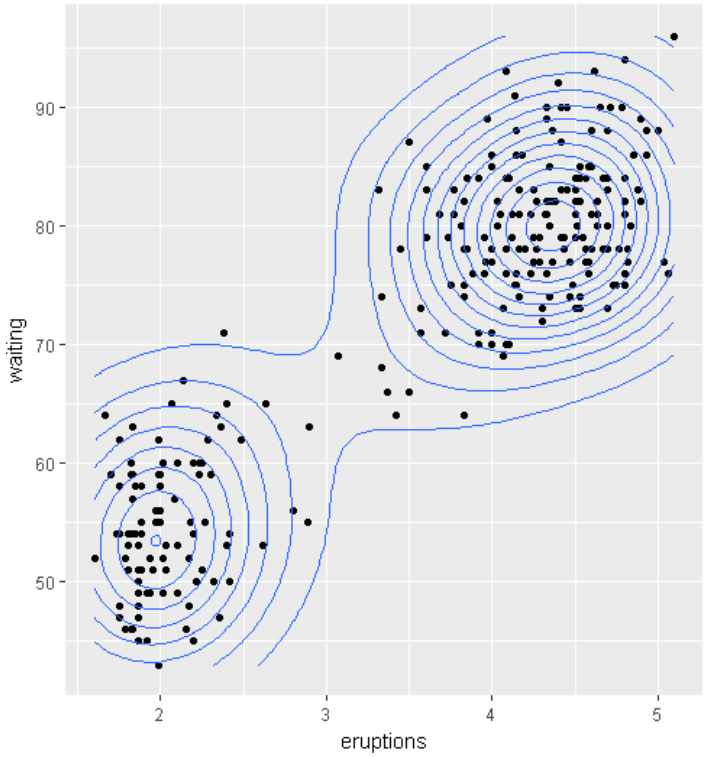


Рис. 10.16. Диаграмма двумерной плотности распределения данных `faithful`

Подогнать смешанную модель кластеризации по этим данным можно с использованием функции `Mclust()` из пакета `mclust`.

```
mc <- Mclust(faithful)
summary(mc) ; head(mc$z)
```

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
Mclust EEE (ellipsoidal, equal volume, shape and orientation)
model with 3 components:
 log.likelihood   n df       BIC       ICL
      -1126.361 272 11 -2314.386 -2360.865
Clustering table:
  1   2   3
130  97  45
----------------------------------------------------
Probality for an observation to be in a given cluster
          [,1]         [,2]         [,3]
1 2.181744e-02 1.130837e-08 9.781825e-01
2 2.475031e-21 1.000000e+00 3.320864e-13
3 2.521625e-03 2.051823e-05 9.974579e-01
4 6.553336e-14 9.999998e-01 1.664978e-07
5 9.838967e-01 7.642900e-20 1.610327e-02
6 2.104355e-07 9.975388e-01 2.461029e-03
```

Мы получили разбиение исходных наблюдений на три кластера с оптимальным `BIC = -2314`. Каждому наблюдению назначается кластер с максимальной оцененной вероятностью `z`. Для визуализации полученных кластеров можно использовать различные варианты функции `plot()` или `fviz_cluster()`:

```
plot(mc, "classification")
plot(mc, "uncertainty")
```

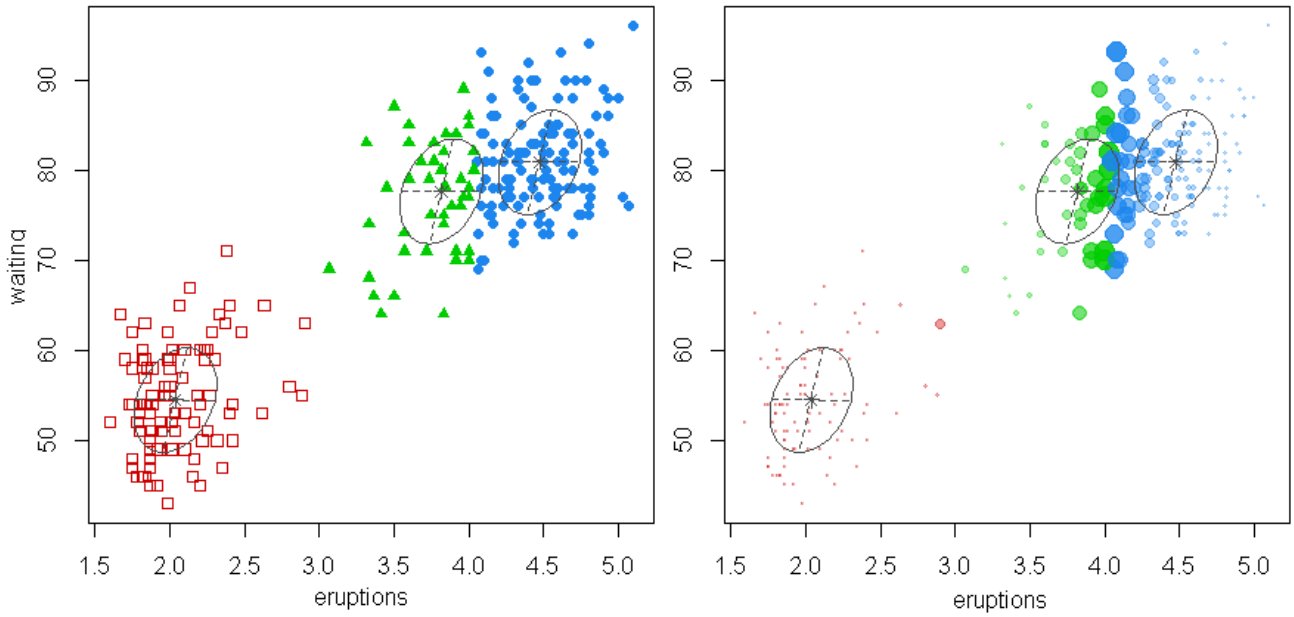


Рис. 10.17. Распределение наблюдений по кластерам (на графике справа диаметр точек соответствует мере неопределенности – `uncertainty`)

Как следует из результатов `summary()`, ковариационные матрицы оптимизированы под вариант структурной организации кластеров `EEE` (эллипсоидальная с равным объемом, формой и ориентацией). Зависимость критерия BIC от числа кластеров для различных вариантов параметризации можно увидеть на следующем графике:

```
plot(mc, "BIC")
```

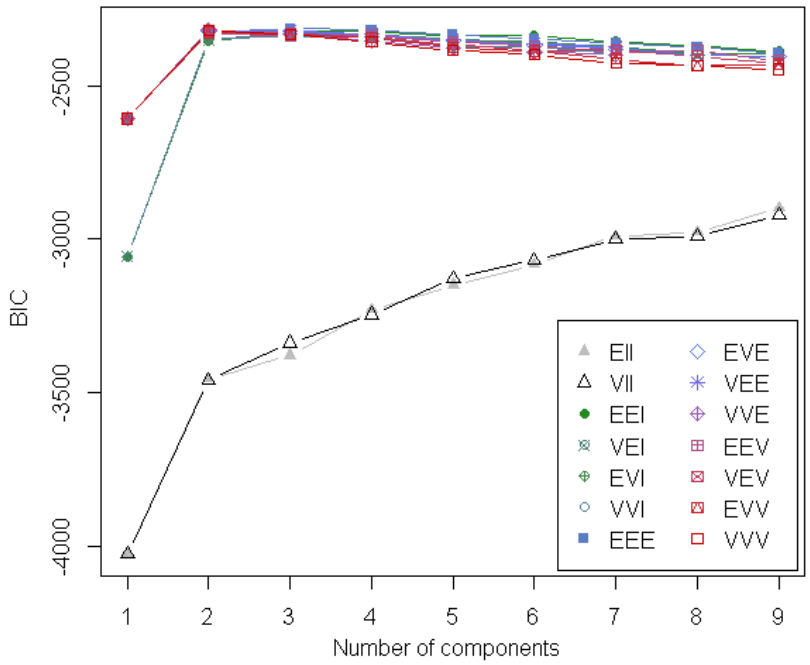


Рис. 10.18. Зависимость критерия BIC от числа кластеров для различных вариантов параметризации ковариационной матрицы

Каждая опция модели, представленная на рис. 10.18, описана идентификатором, первый символ которого относится к объему, второй – к форме, а третий – к ориентации. Символы могут принимать значения `"E"` – равный, `"V"` – переменный и `"I"` – расположение относительно осей координат. Так, `"VEI"` означает, что кластеры имеют разный объем, одинаковую форму и одинаково ориентированы по координатным осям. Подробно о составе и смысле опций можно узнать, выполнив команду `? mclustModelNames`.

10.5. Самоорганизующиеся карты Кохонена

Самоорганизующиеся карты (SOM, Self Organizing Maps), разработанные Т. Кохоненом (Kohonen, 1982), представляют собой мощный инструмент, объединяющий две важные парадигмы анализа данных – кластеризацию и проецирование, т.е. визуализацию многомерных данных на плоскости. В отличие от рассмотренной в разделах 7.7 и 8.2 нейронной сети обратного распространения, процедура настройки SOM относится к алгоритмами обучения без учителя.

Сеть Кохонена имеет всего два слоя: входной и выходной, составленный из радиальных нейронов упорядоченной структуры (выходной слой называют также слоем топологической карты, или "экраном"). Нейроны выходного слоя располагаются в узлах двумерной сетки с прямоугольными или шестиугольными ячейками. Количество нейронов в сетке p определяет степень детализации результата работы алгоритма, и, в конечном счете, от этого зависит точность обобщающей способности карты.

Самоорганизующиеся карты в ходе своего обучения анализируют характер расположения точек входного слоя в m-мерном пространстве и стремятся воспроизвести на выходе нейронной сети топологический порядок и определенную степень регулярности исходных данных (т.е. метрическую близость векторов). Подгонка SOM заключается в итеративной настройке вектора весовых коэффициентов $\boldsymbol{w}_j$ каждого нейрона, $j = 1, 2, \ldots, p$, для чего используется модифицированный алгоритм соревновательного обучения Хебба, который учитывает не только вклад нейрона-победителя, но и ближайших его соседей, расположенных в R-окрестности:

1. На стадии инициализации всем весовым коэффициентам присваиваются небольшие случайные значения w^0_{ij}, $i = 1, 2, \ldots m$.

2. На выходы сети подаются последовательно в случайном порядке образы $\boldsymbol{y}$ объектов входного слоя и для каждого из них выбирается "нейрон-победитель" (BMU, Best Matching Unit) с минимальным расстоянием $\sum_{i=1}^{m}(y_i - w_{ij}^t)^2$ – см. рис 10.19.

3. Определяется подмножество "ближайшего окружения" BMU, радиус которого R уменьшается с каждой итерацией t.

4. Пересчитываются веса $\boldsymbol{w}^t_j$ выделенных узлов с учетом их расстояний до нейрона-победителя и близости к вектору $\mathbf{y}$.

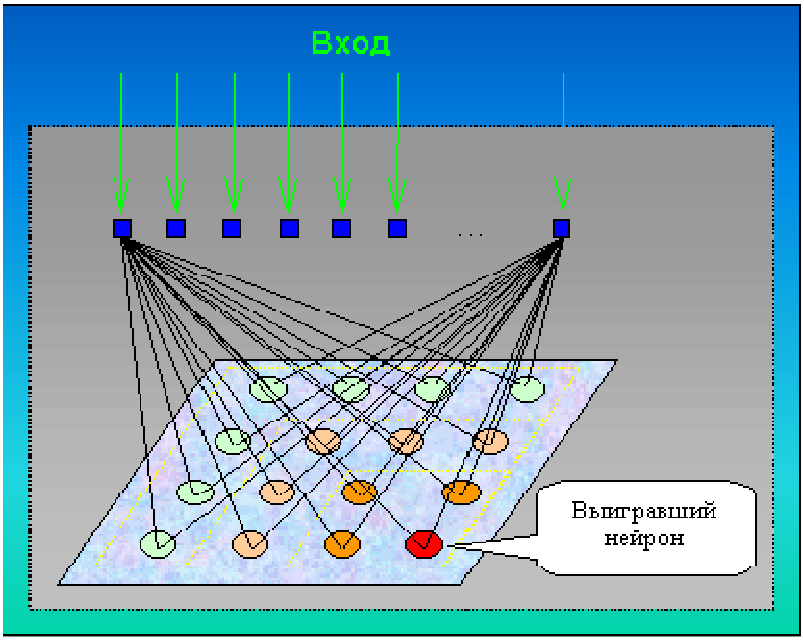


Рис. 10.19. Схема активации нейронов в сети Кохонена

Шаги 2-4 алгоритма повторяются, пока выходные значения сети не будут стабилизированы с заданной точностью. При этом качество проецирования многомерных данных на плоскость достигается в SOM на нескольких уровнях: *сохранение топологии* (т.е. на множествах точек исходных данных и нейронов обученной сети структура соседства одинакова), *сохранение порядка* (т.е. расстояния между эквивалентными парами точек пропорциональны) и *сохранение метрических свойств при сжатии пространства*.

"Проекционный экран" в результате обучения приобретает свойства упорядоченной структуры, в которой величины синапсов нейронов плавно меняются вдоль двух измерений. Цвет и расположение фрагментов двумерной решетки используется для анализа закономерностей, связываемых с компонентами набора данных. В частности, с каждым узлом (нейроном) могут ассоциироваться локальные сгущения исходных объектов, которые могут служить потенциальными центрами кластеров.

Для обучения сети обычно используются функции `somgrid()` и `som()` из пакета `kohonen`. По завершении итерационного процесса функции `plot()` становится доступным для визуализации следующий комплект карт:

- "`codes`" – показывается распределение по решетке соотношение долей участия отдельных исходных переменных;
- "`counts`" – число исходных объектов в каждом узле сети;
- "`mapping`" – координаты исходных объектов на сформированной карте;
- "`property`", "`quality`", "`dist.neighbours`" – различными цветами изображается целый набор свойств каждого узла: доли участия отдельных исходных переменных, меры парных или средних расстояний между нейронами и т.д.

Опять воспользуемся в качестве примера набором `Boston` из пакета `MASS`. Чтобы график "codes" был более лаконичен, из исходного набора признаков выделим 7 переменных, предположительно наиболее значимых для кластеризации (см. рис. 10.13). Смысл сокращенных наименований переменных приведен в разделе 10.3. Выполним предварительно стандартизацию данных.

Укажем функции `somgrid()` создать для проекционного экрана гексагональную решетку 6×9, т.е. 506 земельных участков Бостона будут "самоорганизовываться" на 54 нейронах выходного слоя:

```
data(Boston, package = "MASS")
VarName = c("indus", "dis", "nox",  "medv", "lstat", "age",
"rad")
# отбор переменных для обучения SOM
data_train <- Boston[, VarName]
data_train_matrix <- as.matrix(scale(data_train))
set.seed(123)
som_grid <- somgrid(xdim = 9, ydim=6, topo="hexagonal")
som_model <- som(data_train_matrix,
     grid=som_grid, rlen=100, alpha=c(0.05,0.01),
     keep.data = TRUE)
plot(som_model, type = "changes")
```

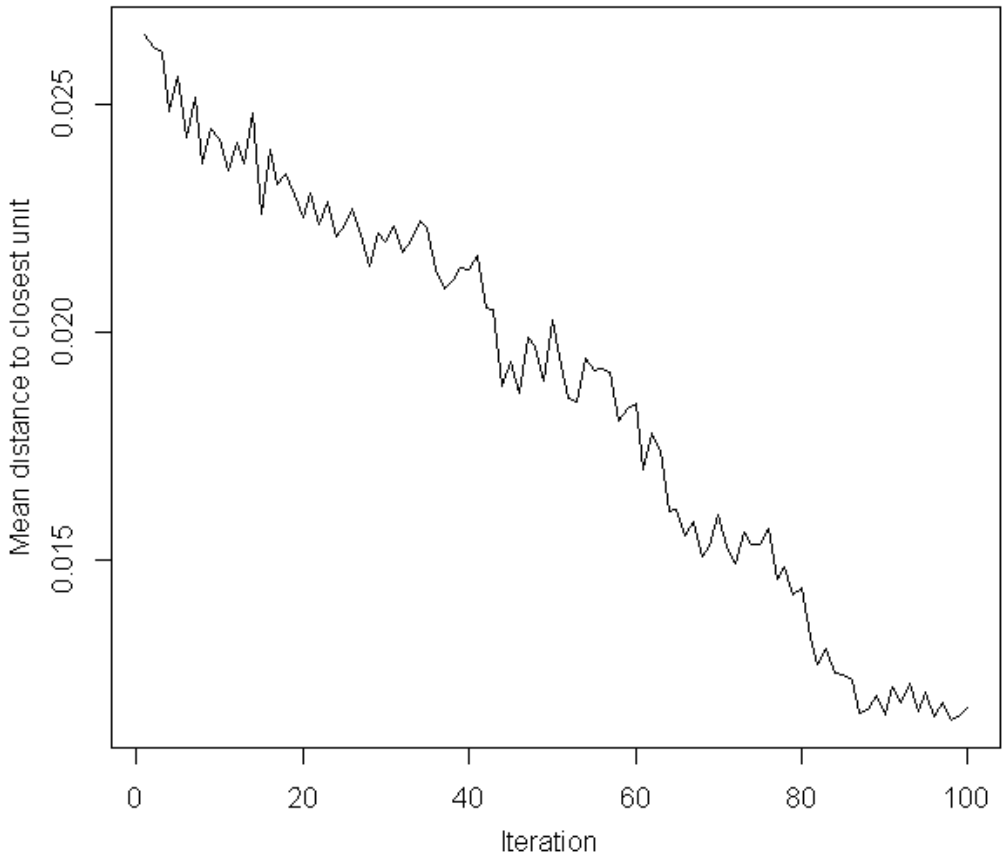


Рис. 10.20. Снижение среднего расстояния до ближайших нейронов в ходе 100 итераций (`rlen`) обучения сети SOM при заданных значениях гипер-параметра `alpha`

Выполним теперь визуализацию комплекта карт Кохонена с разными управляющими параметрами функции `plot()`.

```
# Зададим палитру цветов
coolBlueHotRed <- function(n, alpha = 1) {
        rainbow(n, end=4/6, alpha=alpha)[n:1] }
# Сколько объектов связано с каждым узлом?
plot(som_model, type = "counts", palette.name=coolBlueHotRed)
# Каково среднее расстояние объектов узла до его прототипов?
plot(som_model, type = "quality", palette.name=coolBlueHotRed)
```

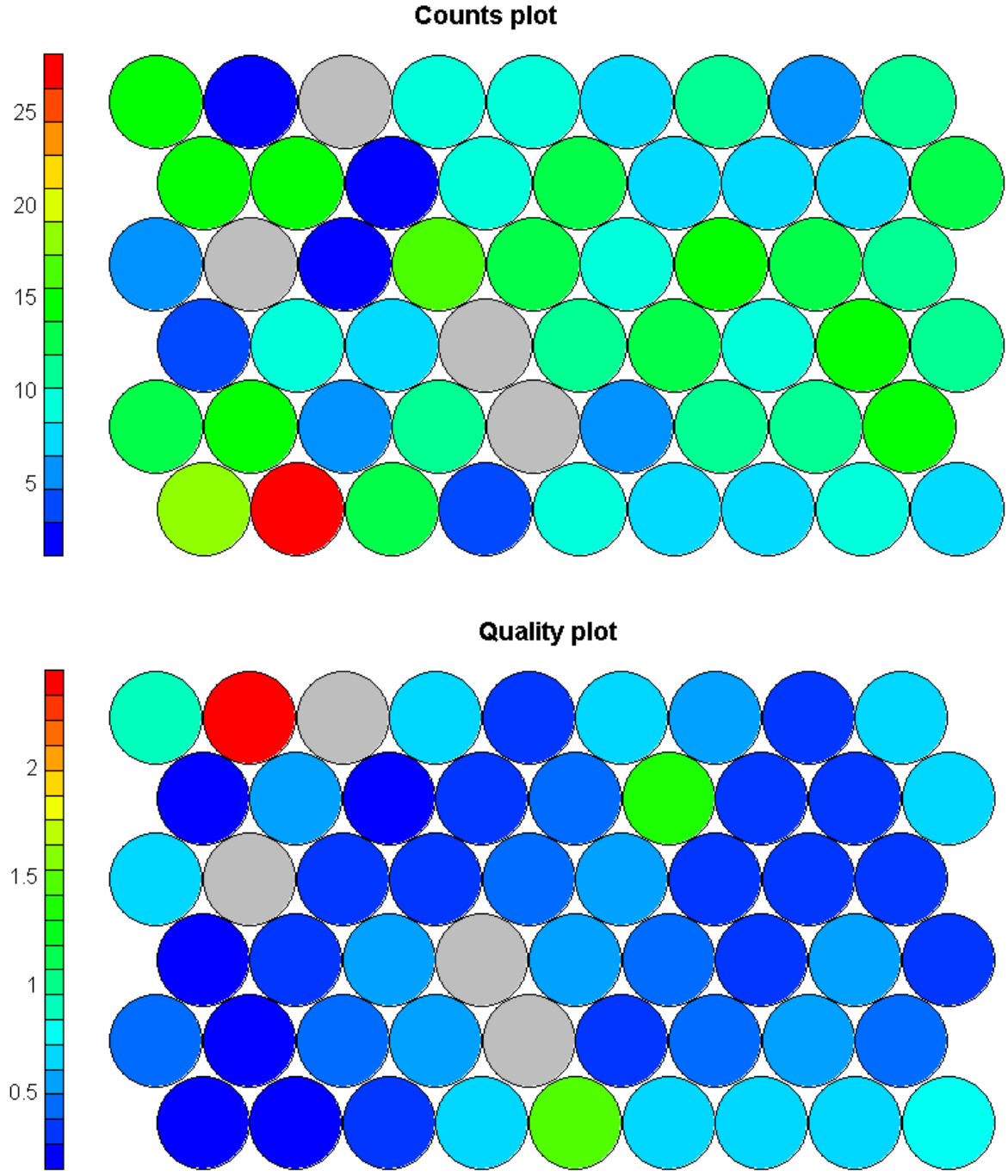


Рис. 10.21. Карты SOM типа "counts" и "quality"

Тип карты "mapping" позволяет получить распределение объектов по узлам, удовлетворяющее любому заданному условию. Например, мы желаем выделить участки с низкой долей афроамериканцев (black – признак, который в настройке сети не участвовал). А также показать, как при этом распределяются доли участия отдельных исходных переменных:

```
colB <- ifelse(Boston$black <= 100, "red", "gray70")
plot(som_model, type = "mapping", col =colB, pch = 16)
plot(som_model, type = "codes")
```

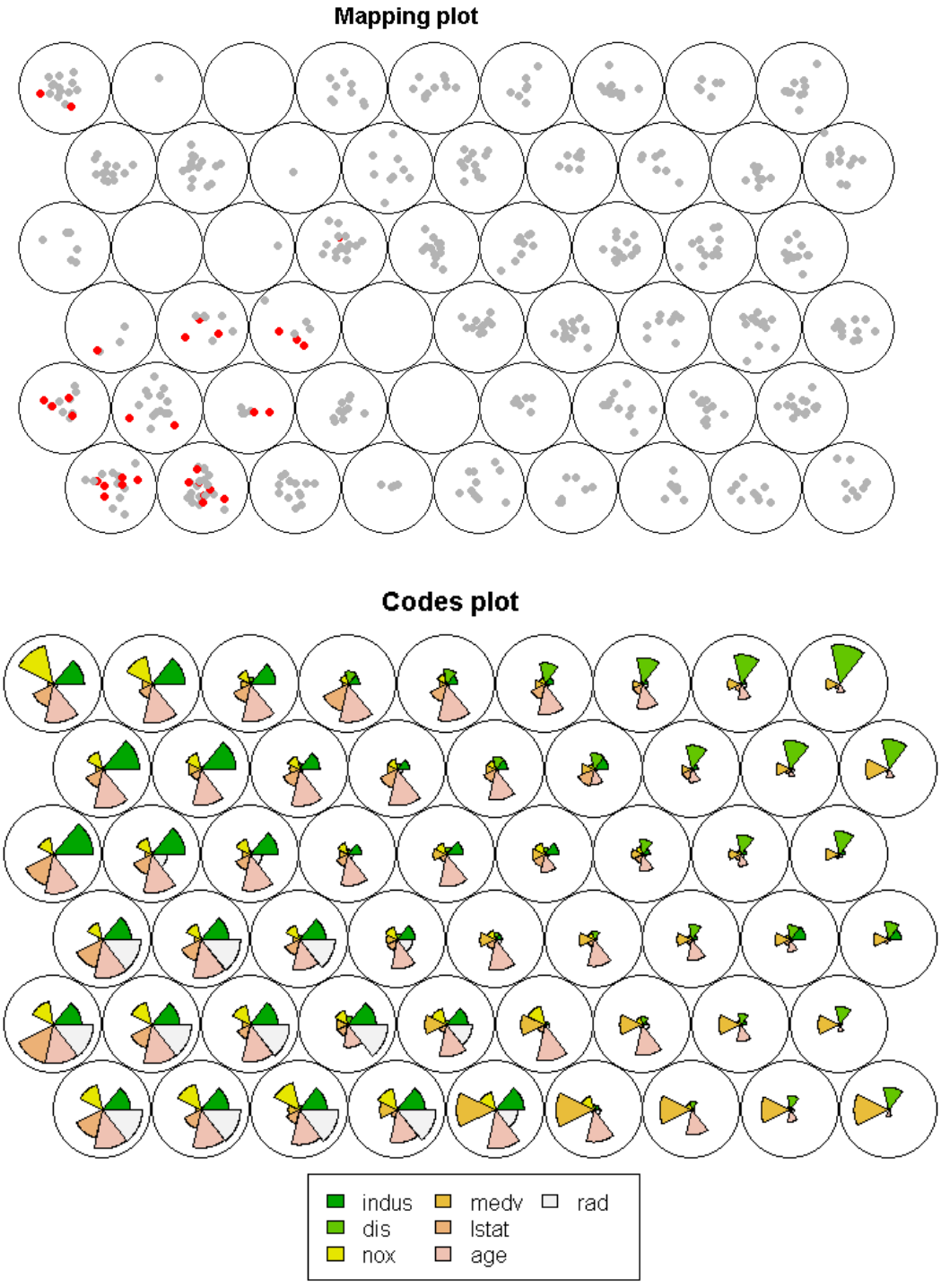


Рис. 10.22. Карты SOM типа "mapping" и "codes"

Поскольку объединенная карта "codes" не всегда бывает хорошо интерпретируемой, можно получить карту распределения любого показателя в его стандартизованной или исходной шкале:

```
plot(som_model, type = "property",
        property = som_model$codes[[1]][,1],
        main = "indus - доля домов, продаваемых в розницу",
        palette.name=coolBlueHotRed)
var_unscaled <- aggregate(as.numeric(data_train[,3]),
 by=list(som_model$unit.classif), FUN=mean, simplify=TRUE)[,2]
plot(som_model, type = "property", property=var_unscaled,
        main="nox - содержание окислов азота",
        palette.name=coolBlueHotRed)
```

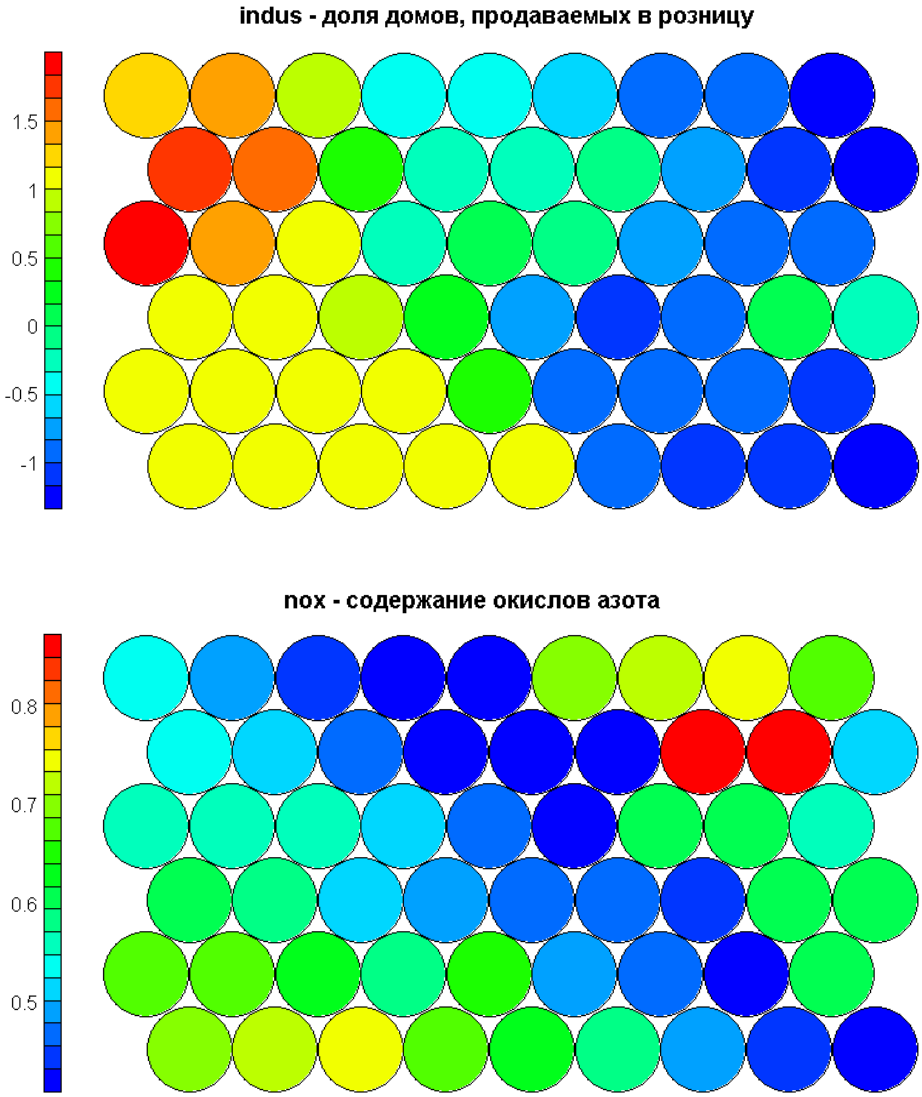


Рис. 10.23. Карты SOM для стандартизованного и исходного показателя

Естественно, что значения активации каждого нейрона по каждому предиктору можно использовать для группировки узлов. Зададимся числом кластеров $k = 5$ и выполним иерархическую кластеризацию (по умолчанию используются method = "complete" и distance = "euclidean"). Можно

построить карты типа "mapping" (с метками объектов) или "codes" (с распределением доли вклада переменных). Остановимся на втором типе:

```
## Формируем матрицу "узлы × переменные"
mydata <- as.matrix(som_model$codes[[1]])
# Используем иерархическую кластеризацию с порогом при k=5
som_cluster <- cutree(hclust(dist(mydata)), 5)
# Определяем палитру цветов
pretty_palette <- c("#1f77b4", '#ff7f0e', '#2ca02c',
          '#d62728', '#9467bd', '#8c564b', '#e377c2')
# Показываем разными цветами кластеры узлов и переменные
plot(som_model, type="codes",
                bgcol = pretty_palette[som_cluster])
add.cluster.boundaries(som_model, som_cluster)
```

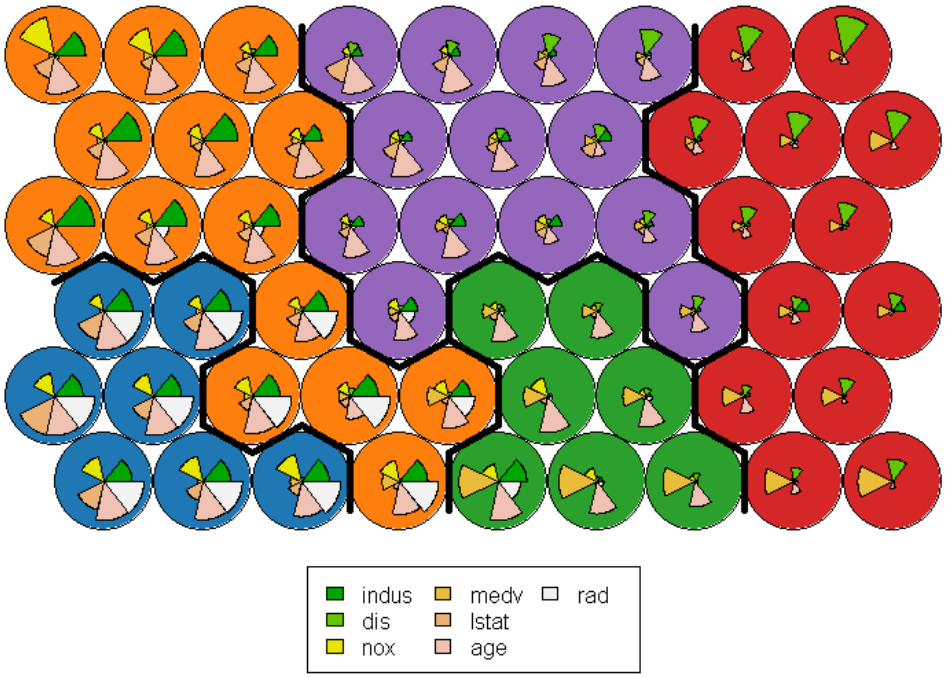


Рис. 10.24. Кластеризация узлов карты SOM

Традиционно метод Кохонена рассматривается как эмпирический алгоритм, а выводы (в первую очередь, качественные) о структуре данных делаются на основе визуального анализа представленных карт. Основная трудность применения SOM, как и в случае анализа главных компонент, заключается в смысловой интерпретации топологии сети и связывании ее отдельных участков с некоторыми конкретными обобщениями из предметной области.

Однако алгоритм SOM нашел широкое применение в ГИС-технологиях, поскольку легко реализовать информационную цепочку от исходной таблицы к узлам решетки, а от них – к конкретным координатам на географической карте. Например, участники коллектива «Dublin R Users Group» использовали результаты переписи населения 2011 г., разбили территорию Дублина на 18500 маленьких площадок и описали проживающее на них население с использованием 767 переменных из 15 разделов. На ресурсах (https://www.r-

bloggers.com, http://www.slideshare.net) доступны полный комплект исходных данных, подробное описание и скрипты R. Основные результаты представлены на рис. 10.25:

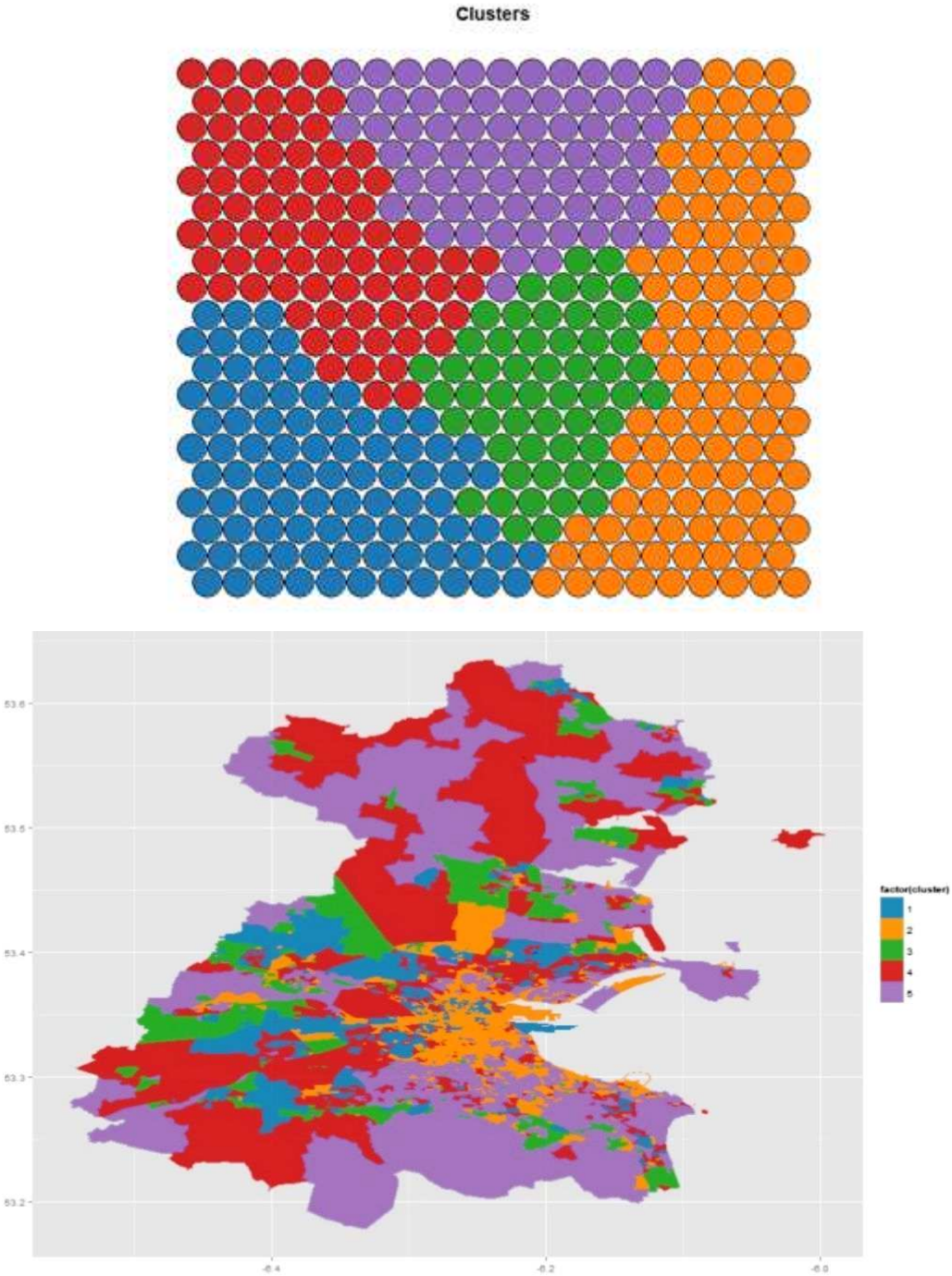


Рис. 10.25. Кластерная форма карты SOM и ее отображение на карте Дублина (категории от 1 до 6 связываются с градациями жилой застройки и обеспеченности населения – от трущоб до благополучных элитных районов)

Похожий подход использован в программе ScanEx IMAGE Processor v3.6 (http://www.scanex.ru), руководство к которой содержит также достаточно подробное русскоязычное описание техники расчетов и формул, применяемых при обучении SOM.

11. RATTLE: ГРАФИЧЕСКИЙ ИНТЕРФЕЙС R ДЛЯ РЕАЛИЗАЦИИ АЛГОРИТМОВ DATA MINING

11.1. Начало работы с пакетом `rattle`

Читатель, вероятно, помнит, что удобным средством освоения вычислений в R для начинающего пользователя является R Commander. Аналогичный платформо-независимый графический интерфейс с кнопками и меню, позволяющий выполнять целый набор алгоритмов Data Mining, реализован в пакете `rattle` (Williams, 2009, 2011). С помощью этого пакета можно реализовать многие поцедуры обработки данных, не прибегая к предварительному заучиванию функций на командном языке.

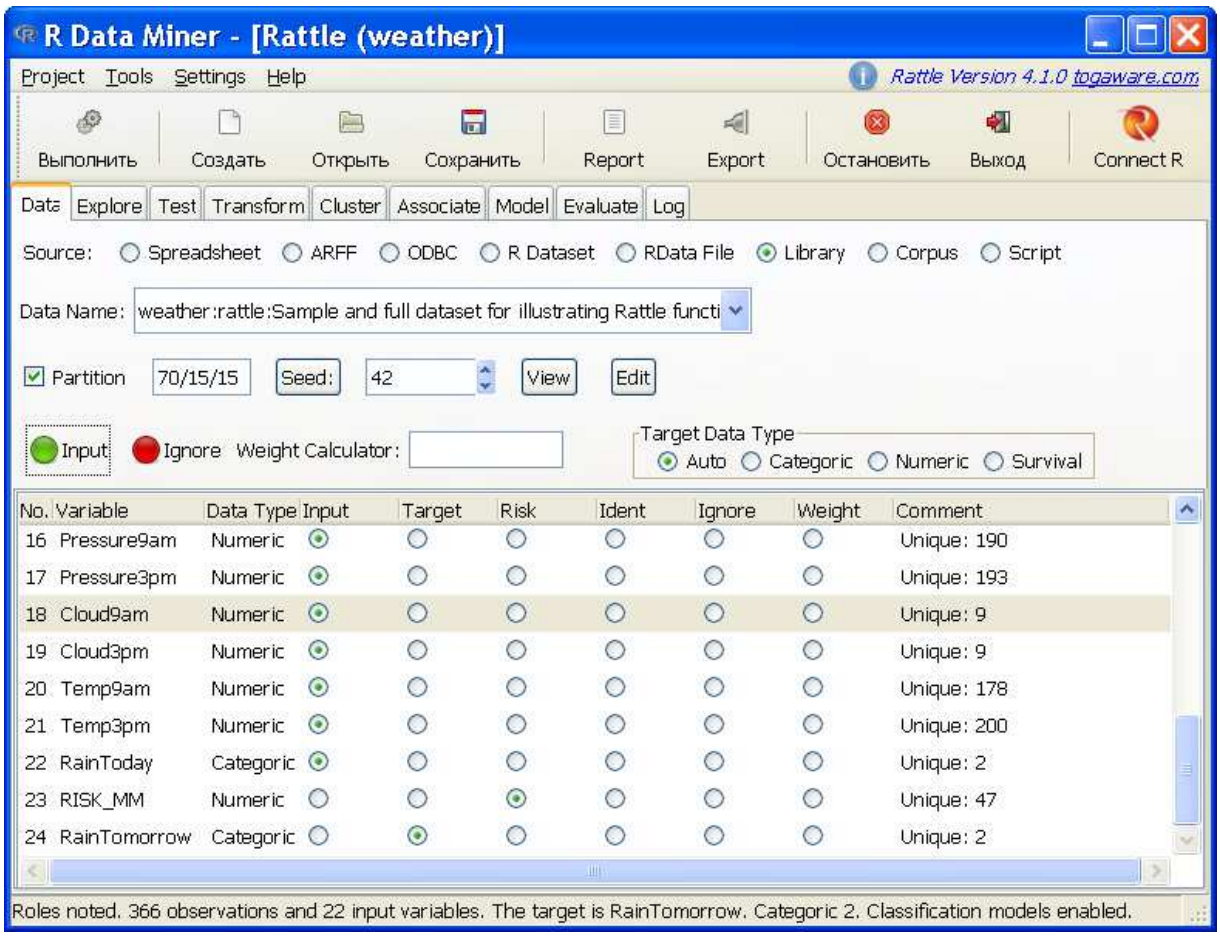
Установить пакет `rattle`, как и любые другие расширения в R, можно командой

```
install.packages("rattle").
```

В процессе инсталляции загружается графическая динамическая библиотека `RGtk2.dll`, а также необходимые для работы `rattle` пакеты `cairoDevice` и `XML`.

Интерактивная среда запускается командой `'rattle()'`.

Вы можете приступить к работе, загрузив необходимый набор данных, или продолжить сеанс анализа, открыв файл проекта с сохраненной историей Ваших действий:



На приведенном рисунке загружена таблица `weather` из пакета `rattle`, содержащая данные наблюдений за погодой на метеорологической станции в г. Канберра, Австралия (24 переменных разного типа в 366 строках).

Таблицы данных (вкладка **Data**) могут загружаться в среду `rattle` из следующих источников:

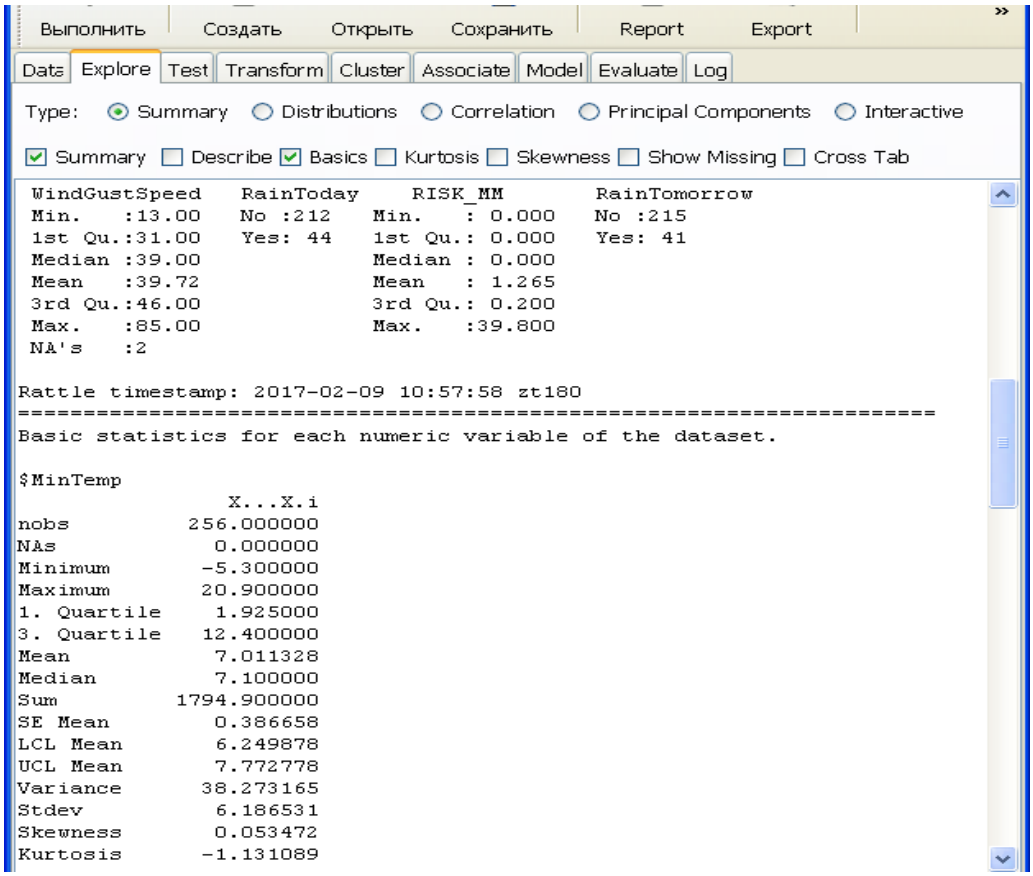
- из произвольного текстового файла с разделителями типа .csv (Spreadsheet), или ARFF (Attribute-Relation File Format);
- через программный интерфейс доступа к базам данным ODBC (Open Database Connectivity), а также через буфер обмена, из Web, SPSS-файлов и проч.;
- из бинарных или обыкновенных файлов `.RData`, а также к наборам данных любых установленных пакетов (R dataset, RData File, Library).

Для тестирования моделей исходные данные могут быть разделены на обучающую, проверочную и экзаменационную последовательности (на рисунке это соотношение составляет 75/15/15 процентов и может быть изменено в любом соотношении).

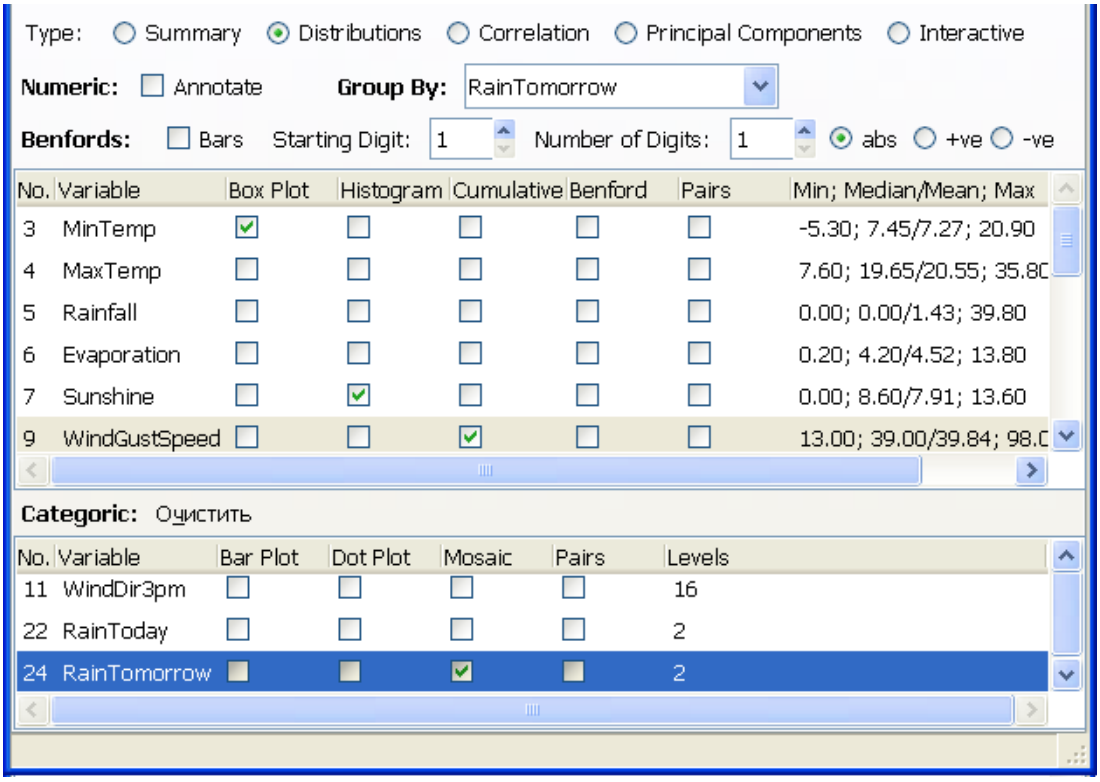
Список переменных выводится в нижней части окна и можно определить их роль в дальнейших действиях: они могут быть предикторами, откликом (Target), а также соответствовать весам (Weight), так называемым "переменным риска" (Risk), или игнорироваться (Ignore).

11.2. Описательная статистика и визуализация данных

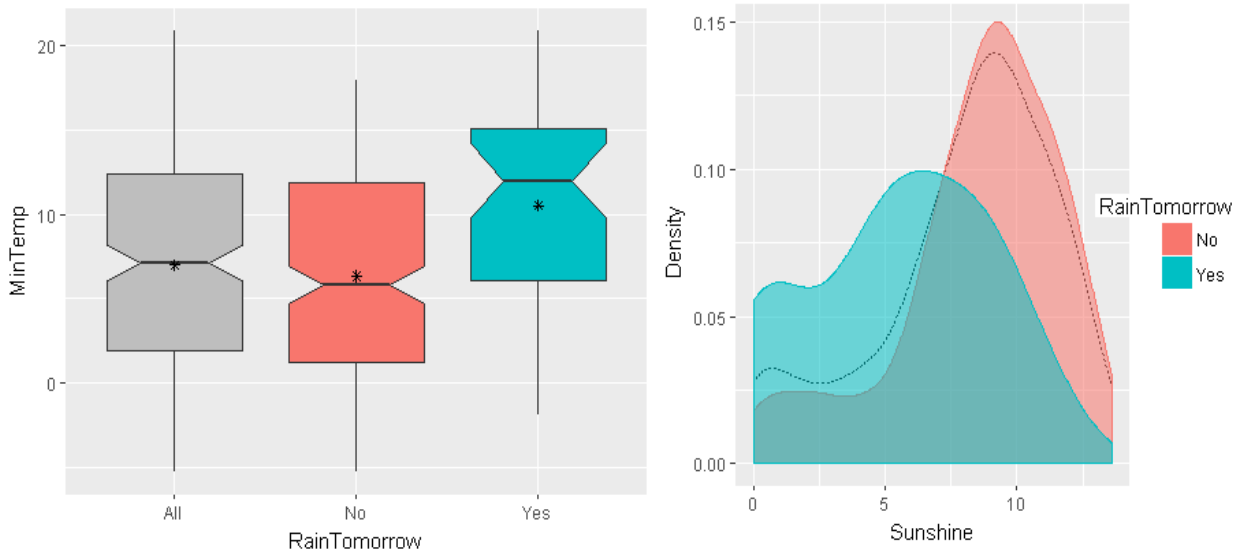
Вкладка **Explore** с опцией Summary обеспечивает различные варианты вывода выборочных описательных статистик:



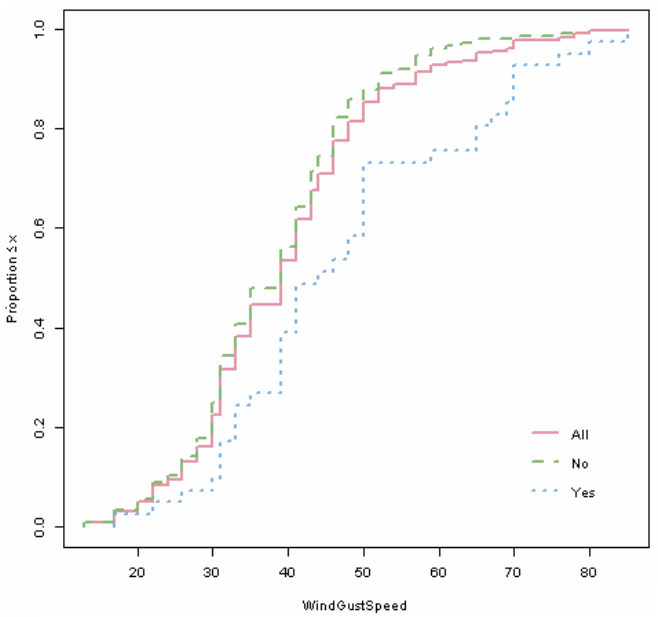
Опция Distribution управляет выводом различных комбинаций стандартных графиков R: диаграмм размахов, ядерных и кумулятивных функций распределения, мозаичных диаграмм и проч. Отображаемые переменные и их пары можно легко выбрать из списка:

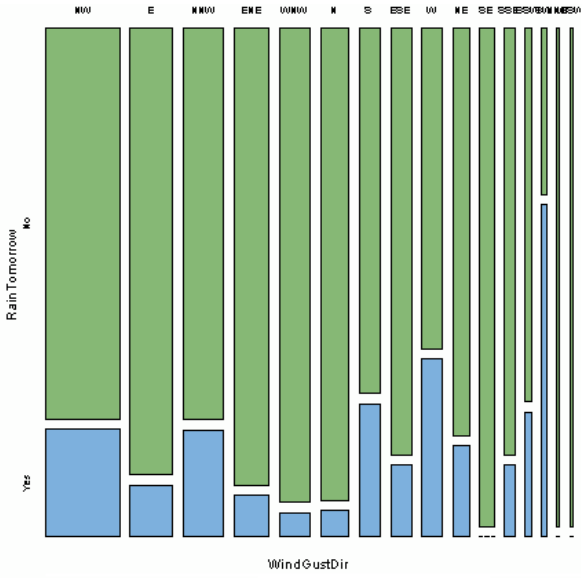


Поскольку ранее показатель "Завтра дождь" (`RainTomorrow`) был установлен как отклик, данные на диаграммах разбиваются по его категориям. В частности, из диаграммы слева видно, что вероятность выпадения дождя на следующий день сопряжена с повышенной минимальной дневной температурой (`MinTemp`). Также вполне понятно, что перед дождем наблюдается пониженная солнечная активность (`Sunshine`):

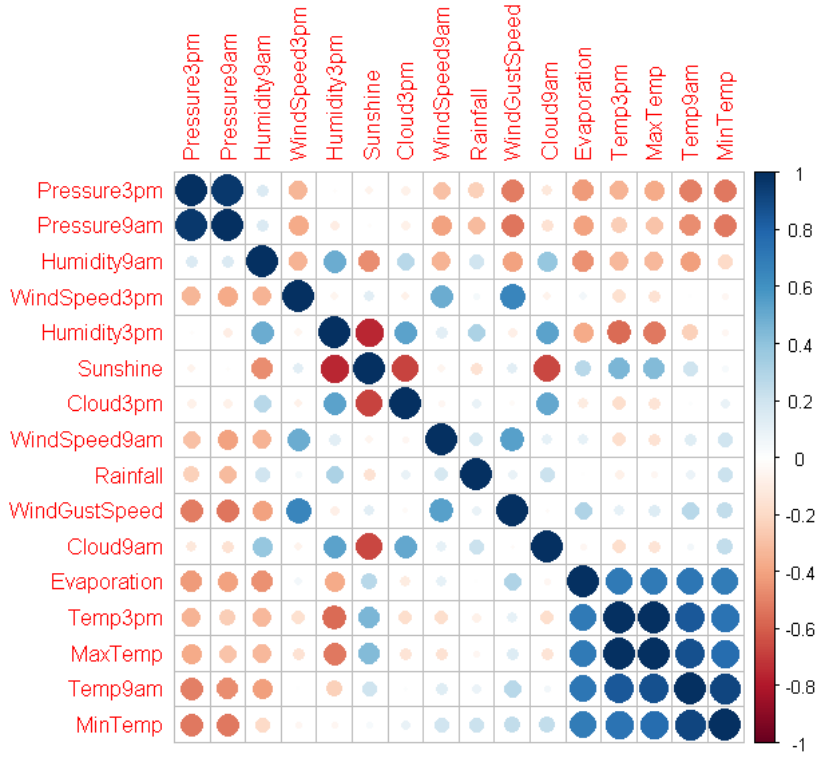


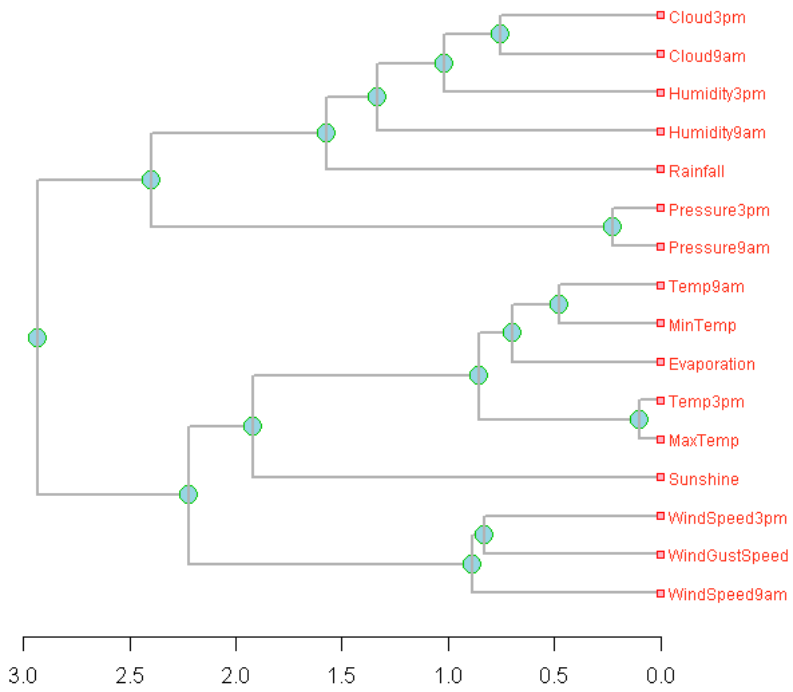
Вероятно, специалистам будут полезны кумулятивная функция распределения «Скорости порывов ветра» (`WindGustSpeed`) и мозаичная диаграмма «Изменения направления ветра» (`WindGustDir`) в зависимости от вероятности предстоящего дождя `RainTomorrow`:





Аналогично можно выполнить анализ главных компонент (Principial Component) или оценить тесноту корреляционных связей по Пирсону, Кендаллу или Спирмену (Correlation), представив их в виде матрицы коэффициентов или корреляционной дендрограммы:





Применить прекрасный графический пакет GGobi для построения графиков взаимодействия переменных (Interactive) нам не удалось, т.к. в ходе инсталляции куда-то запропастился модуль `rggobi.dll`.

Вкладка **Test** позволяет выполнить разнообразные тесты для проверки статистических гипотез относительно двух выборок (или двух переменных из таблицы данных):

- отличаются ли выборочные распределения по критерию Колмогорова-Смирнова или критерию рангов признаков Уилкоксона?;
- различаются ли положения средних занчений по *t*-критерию или сумме рангов Уилкоксона –Манна–Уитни?;
- различны ли их дисперсии по *F*-критерию?
- коррелируют ли две выборки по значению коэффициента Пирсона?

Вкладка **Transform** включает вполне полный джентльменский набор функций обслуживания и преобразования данных: нормализации, шкалирования, логарифмирования, деления на интервалы (binning), модификации категорий факторов, заполнения пропущенных значений, удаления лишних переменных и др.

Наконец, вкладка **Log** содержит полный перечень команд R, выполненных на интерактивном уровне, что позволяет скопировать фрагмент скрипта в командную консоль R, внести, например, исправления в детали графика и повторить его вывод в измененном формате.

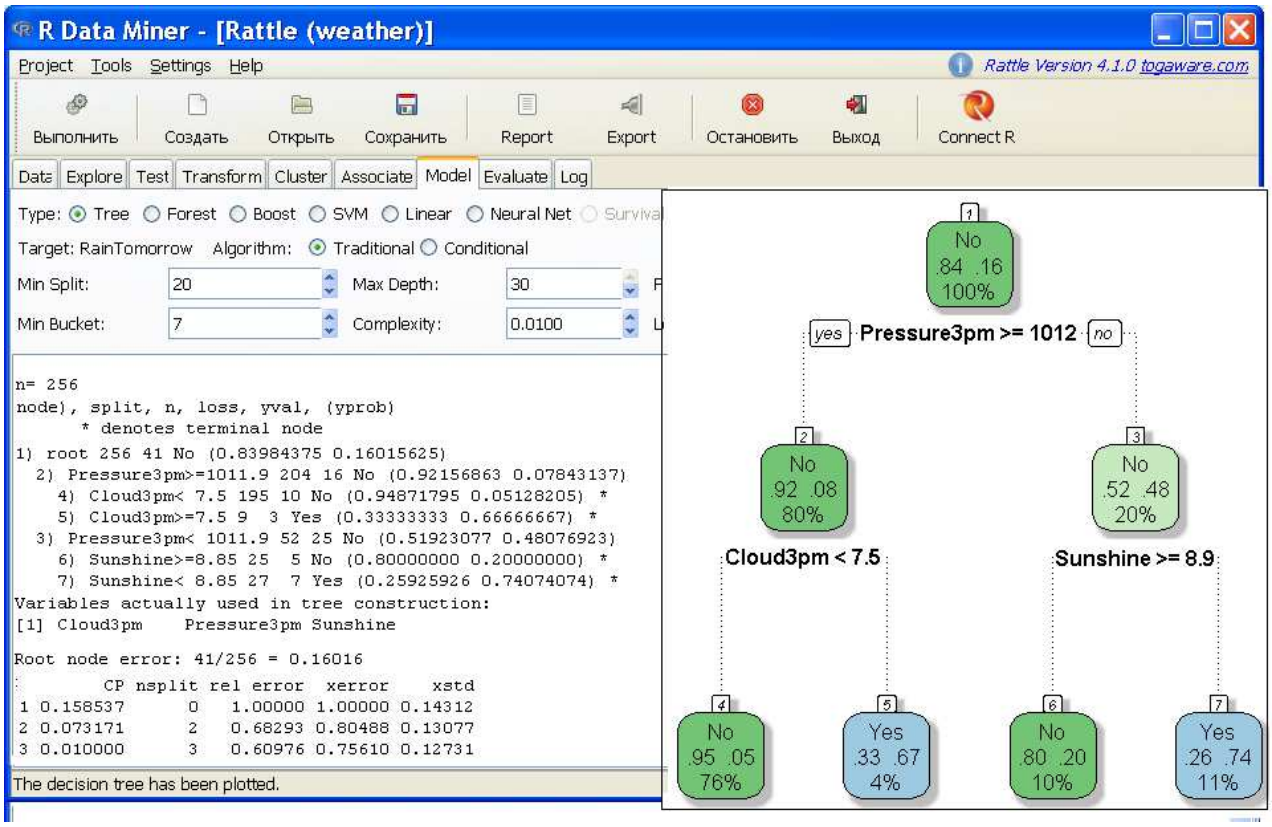
11.3. Построение и тестирование моделей классификации

С использованием интерактивной оболочки Rattle можно построить шесть моделей классификации (вкладка **Model**), подробно описанных нами в главах 3-8: деревья решений (Tree), случайные леса и бустинг (Forrest и Boost), машину опорных векторов (SVM), линейную модель (Linear) и

нейронную сеть (Neural Net).

Для построения моделей разделим всю выборку на две части – обучающую и тестовую – в соотношении 70/30, т.е. отдельно проверочную выборку выделять не будем. Также заблокируем сопутствующие переменные `Date` и `Location`.

Начнем с построения традиционных деревьев рекурсивного разделения `rpart` и построим модель прогнозирования дождя на следующий день `RainTomorrow` с использованием параметров, принятых по умолчанию:



Построенное дерево основано на трех переменных из 20: облачности (`Cloud3pm`), атмосферном давлении в 15:00 ч (`Pressure3pm`), а также количества часов яркого света за день (`Sunshine`).

Для оценки эффективности модели переходим во вкладку **Evaluate**:

Построенную модель мы можем оценить по 9 группам показателей. Кроме того, тестирование можно выполнить как по всей выборке (Full), так и отдельно по обучающей (Training) и экзаменационной (Testing)

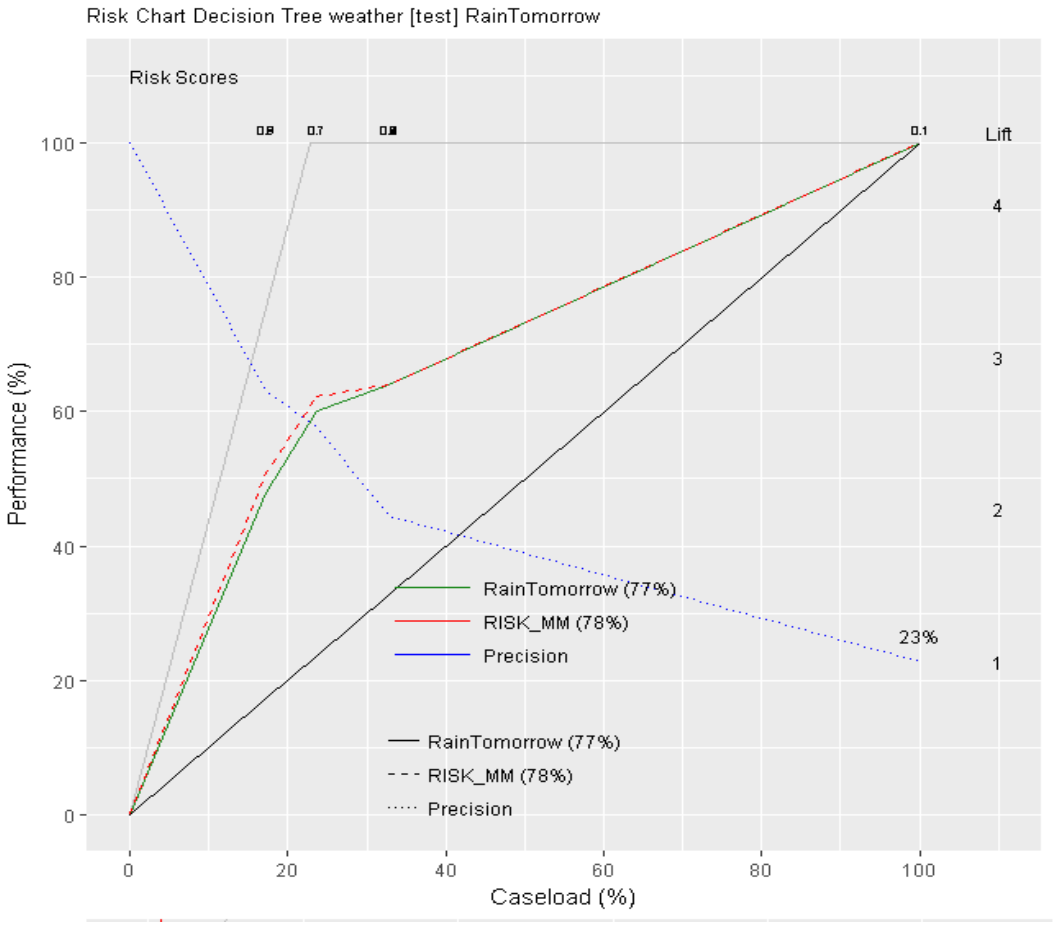
последовательностям.

Матрица неточностей (Error Matrix) является наиболее общим результатом выполненного анализа:

	На всей выборке	Экзамен	Обучение
По количеству объектов	Predicted Actual No Yes No 279 21 Yes 25 41	Predicted Actual No Yes No 74 11 Yes 10 15	Predicted Actual No Yes No 205 10 Yes 15 26
В долях	Predicted Actual No Yes Error No 0.76 0.06 0.07 Yes 0.07 0.11 0.38	Predicted Actual No Yes Error No 0.67 0.10 0.13 Yes 0.09 0.14 0.40	Predicted Actual No Yes Error No 0.80 0.04 0.05 Yes 0.06 0.10 0.37
Общая ошибка	13%	19%	10%
Ошибка класса	22%	26%	21%

Другой подход к оценке эффективности бинарного классификатора заключается в построении кумулятивной версии ROC-кривой – диаграммы риска (risk chart), также известной как диаграмма полезности (gain).

```
Summary Decision Tree model (rpart) on weather [test]
                by probability cutoffs.
              Recall      Risk  Caseload Precision   Measure
0.0512820512821   1.00 1.0000000 1.0000000 0.2272727 0.0000000
0.2               0.64 0.6391960 0.3272727 0.4444444 0.6246505
0.6666666666667   0.60 0.6221106 0.2363636 0.5769231 0.7493833
0.7407407407407   0.48 0.5075377 0.1727273 0.6315789 0.6420831
1.0               0.00 0.0000000 0.0000000 1.0000000 0.0000000
Area under the Recall (green) curve: 77% (0.771)
Area under the Risk   (red)   curve: 78% (0.777)
```

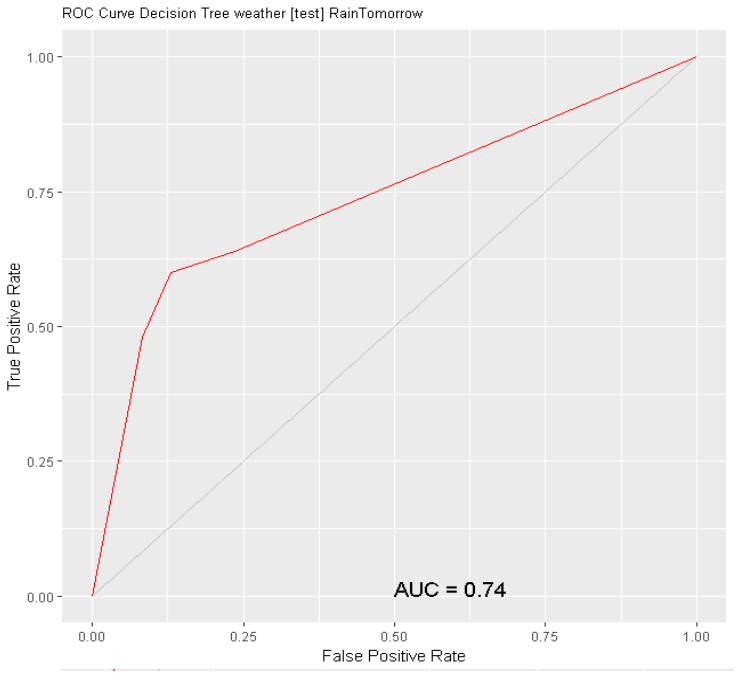


Для построения диаграмм риска, как правило, используются дополнительно контрольные наборы данных (audit dataset). Так, для таблицы `weather` в обучении любой модели классификации могут участвовать две переменные: бинарный отклик `RainTomorrow` и переменная риска `RISK_MM`, связанная с мерой вероятности наступления прогнозируемого события, оцененная аудитом для каждого наблюдения.

Наблюдения упорядочивают по убыванию предсказанных для них оценок риска `RISK_MM` и на оси абсцисс диаграммы последовательно откладывают доля наблюдений (Case load, %), а по оси ординат – доля удачных прогнозов (Perfomance) на основании построенной модели. Линия главной диагонали не связана с какой-либо моделью прогнозирования, т.е. для 50% наблюдений случайное угадывание окажется верным лишь в 50% случаев. Если использовать предсказанные деревом решений вероятности предстоящего дождя, то мы получим уже 73% случаев верного прогноза. Сканируя долю наблюдений от 0 до 100%, можно получить также сглаженную диаграмму изменения риска `RISK_MM` (показана красной пунктирной линией). Пунктирная синяя линия указывает на графике изменение специфичности (precision). Подробности анализа диаграмм риска и других показателей эффективности моделей представлены в монографии по `rattle` (Williams, 2009, 2011).

Для оценки традиционной AUC можно построить ROC-кривую (ROC) и рассчитать такие критерии, как энтропия `H` и индекс Джини `Gini`:

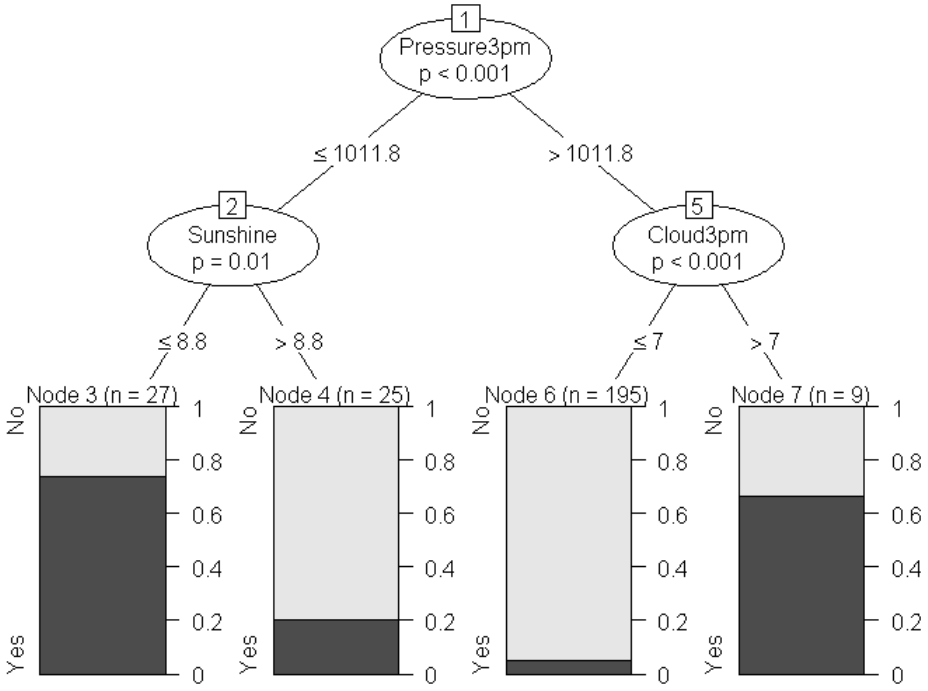
```
H=0.302928, Gini=0.475765, AUC=0.737882,AUCH=0.741647, KS=0.470588
```



Точность остальных шести моделей прогнозирования оценивается по тому же принципу и столь подробно далее рассматриваться не будет. Мы приведем только основные характеристики построенных моделей, AUC и точность классификации на тестовой выборке, а в некоторых случаях также информацию о наиболее важных предикторах.

Деревья условного вывода (Conditional Trees):

```
Conditional inference tree with 4 terminal nodes
Area under ROC curve for ctree model on weather [test] is 0.7379
Overall error: 19%, Averaged class error: 26%
```



Модель случайного леса (Random Forrest):

```
                Number of trees: 500
No. of variables tried at each split: 4
        OOB estimate of  error rate: 14.45%
Area under ROC curve for rf model on weather [test] is 0.9009
Overall error: 11%, Averaged class error: 22%

Variable Importance
                 No   Yes MeanDecreaseAccuracy MeanDecreaseGini
Pressure3pm   12.53 10.96                15.42             5.45
Cloud3pm      13.40  9.38                15.10             3.26
Sunshine      13.10  7.59                14.40             4.16
WindGustSpeed 10.17  6.71                11.90             2.94
MaxTemp        9.10 -0.97                 9.26             2.12
```

Модель бустинга (Ada Boost):

```
Loss: exponential Method: discrete   Iteration: 50
Train Error: 0.047
Out-Of-Bag Error:  0.07  iteration= 37
Area under the ROC curve for ada model on weather [test] is 0.8593
Overall error: 15%, Averaged class error: 20%
Frequency of variables actually used:
     Sunshine   Pressure3pm   Cloud3pm    Temp3pm WindGustSpeed
           32            28         17          13            13
```

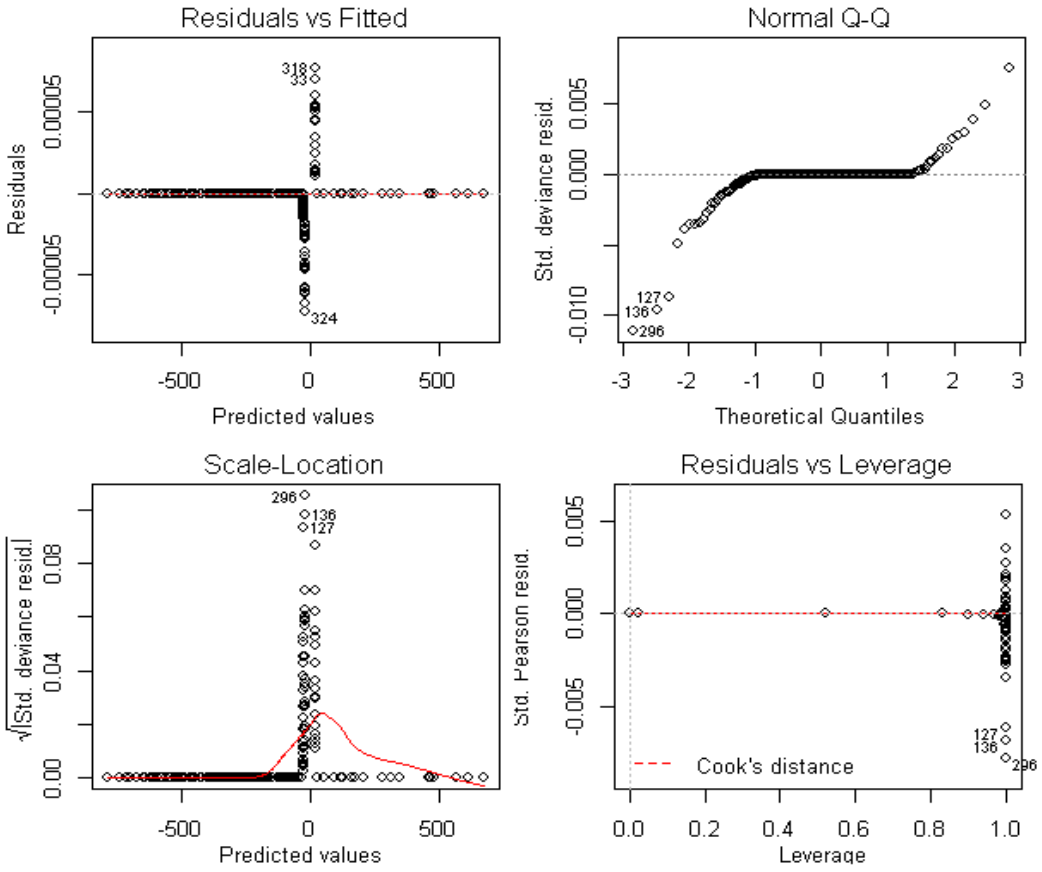
Модель опорных векторов (Support Vector Machine):

```
SV type: C-svc  (classification)  parameter : cost C = 1
Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0379412611254591
Number of Support Vectors : 107
```

```
Objective Function Value : -59.5602
Training error : 0.100877
Area under ROC curve for ksvm model on weather[test] is 0.9248
Overall error: 18%, Averaged class error: 38%
```

Модель логистической регрессии:

```
Model: binomial, link: logit    Response: RainTomorrow
Area under ROC curve for the glm model on weather [test] is 0.6597
Overall error: 29%, Averaged class error: 34%
```




```
Analysis of Deviance Table
             Df Deviance Resid. Df Resid. Dev     Pr(>Chi)
NULL                          227    205.456
MinTemp       1   14.151       226    191.305    0.0001688 ***
MaxTemp       1    2.131       225    189.175    0.1443620
Rainfall      1    0.022       224    189.152    0.8811792
Evaporation   1    0.012       223    189.141    0.9143115
Sunshine      1   25.223       222    163.918 0.0000005108 ***
WindGustDir  15   41.721       207    122.197    0.0002481 ***
WindGustSpeed 1    7.516       206    114.681    0.0061156 **
WindDir9am   15   19.016       191     95.665    0.2130107
 …                                              …
Pressure9am   1    6.431       171     71.042    0.0112151 *
Pressure3pm   1   71.042       170      0.000    < 2.2e-16 ***
…                                              …
RainToday     1    0.000       165      0.000    0.9998435
```

Нейронная сеть прямого распространения сигнала:

```
Summary of the Neural Net model (built using nnet):
A 62-10-1 network with 703 weights.
Sum of Squares Residuals: 0.0017.
Area under ROC curve for nnet model on weather [test] is 0.6766
Overall error: 25%, Averaged class error: 32%
```

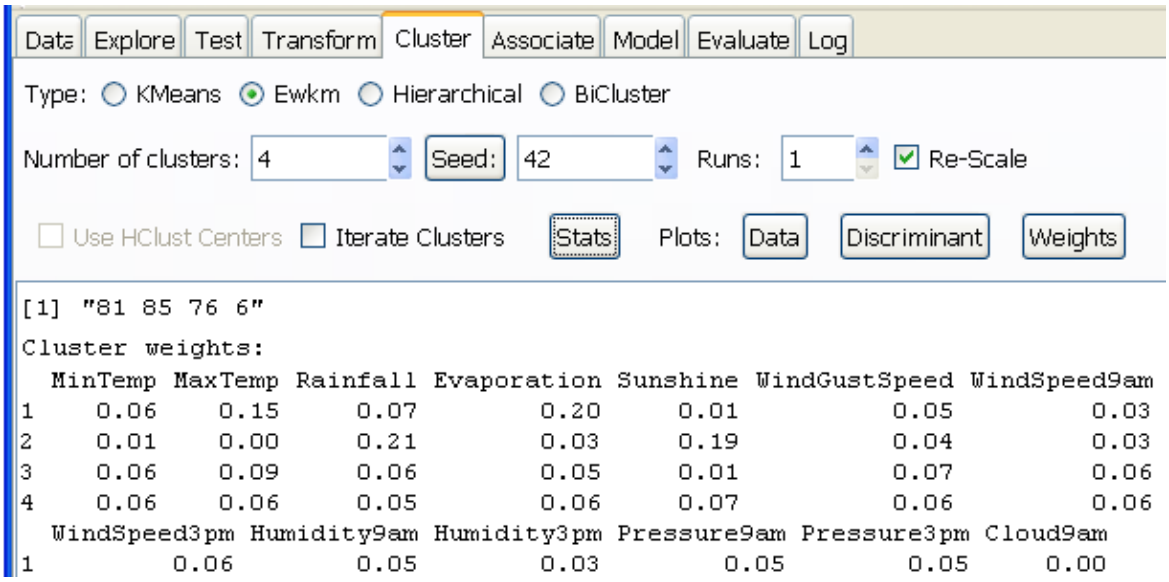
При иных формах представления отклика (в метрической или счетной шкале) возможно также построение обычной или обобщенной модели на основе распределений Гаусса или Пуассона, а также логит- или пробит-моделей для откликов с несколькими классами.

11.4. Дескриптивные модели (обучение без учителя)

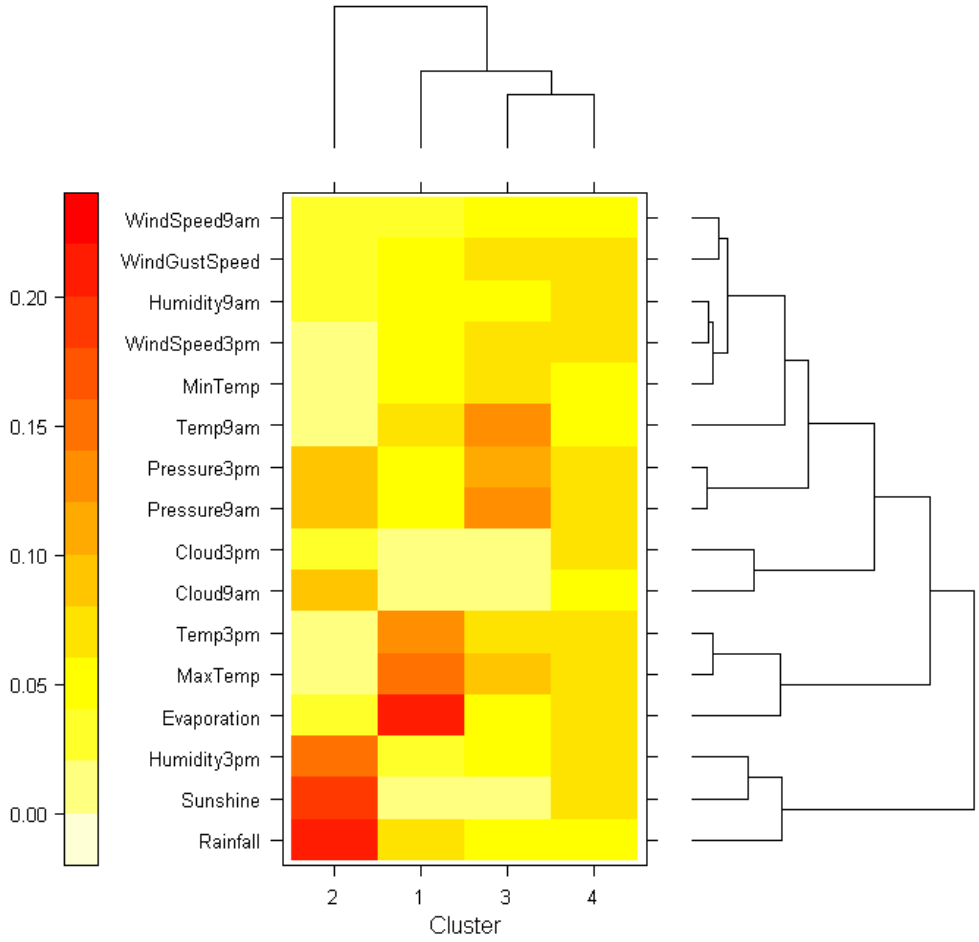
Кластерный анализ

Методы кластерного анализа подробно рассматривались в главе 10, и мы уже не рассчитывали обнаружить в среде `rattle` какие-то новые сюжеты. Однако, приступив к кластеризации признаков привычного набора наблюдений за погодой в г. Канберра, мы увидели еще не представленный читателям алгоритм EWKM (Entropy Weighted K-Means), который является усовершенствованной версией метода *k* средних для обработки больших массивов данных.

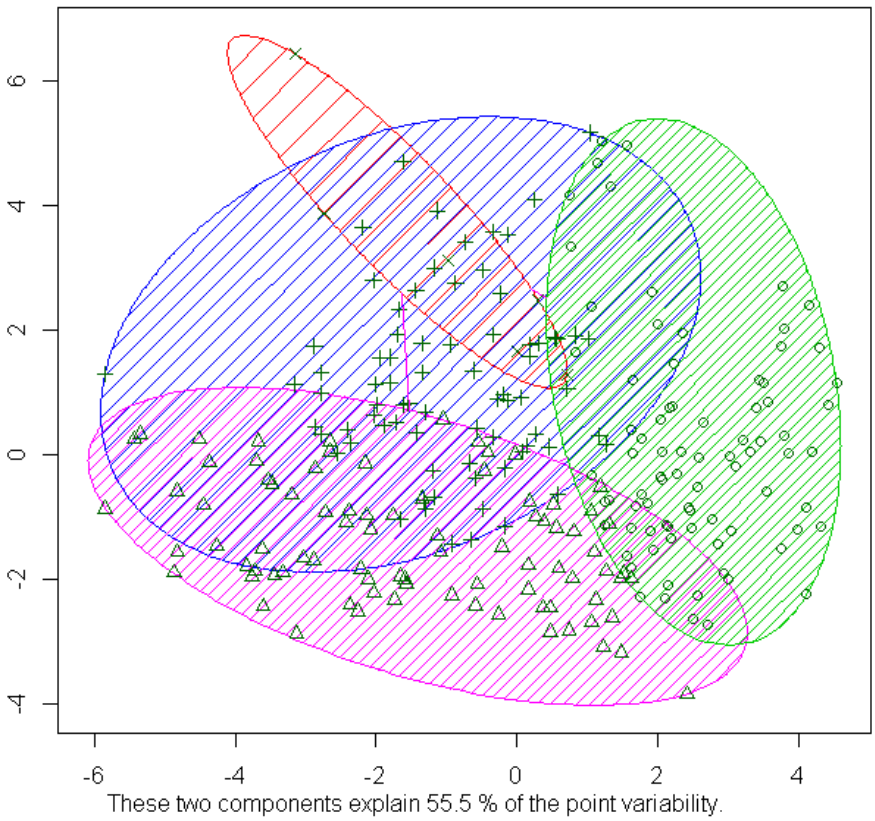
Отличие алгоритма, реализованного функцией `ewkm()` из одноименного пакета, заключается в вычислении весов, оценивающих меру относительной важности участия каждой переменной в формировании каждого кластера. Эти веса включаются в функцию расстояния, тем самым уменьшая его для более значимых переменных, и пересчитываются на каждой итерации объединения в кластеры.



Нажатие кнопки Weights приводит к выводу весьма полезной диаграммы, которая для каждой переменной и каждого кластера показывает итоговые относительные веса в стиле "тепловой карты" (реализовано функцией `levelplot()` из пакета `ewkm`). Одновременно приведены дендрограммы иерархического объединения в группы как использованных переменных, так и исходных наблюдений, что облегчает проблему выбора числа кластеров.



Нажатие кнопки Discriminant выводит ординационную диаграмму с нанесенной структурой кластеров – продукт использования функции `clusplot()`:



Ассоциативные правила

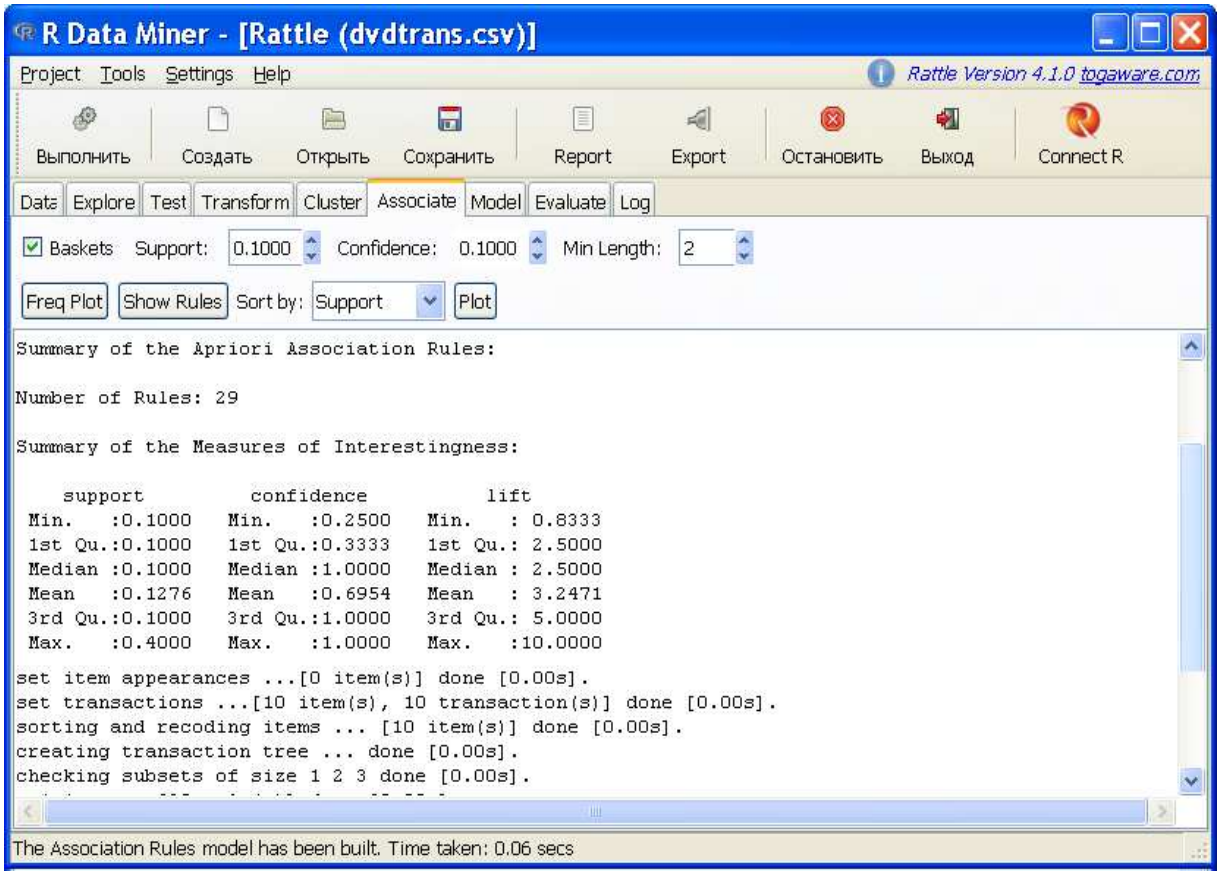
Подробно механизм формирования ассоциативных правил был рассмотрен нами в разделе 5.4. Здесь мы остановимся на небольшом примере распределения корзины товаров из 10 фильмов на DVD по заявкам 10 покупателей. Цель анализа – обнаружить закономерности формирования заявок.

Исходные данные загрузим из файла `dvdtrans.csv`, представленном на сайте разработчиков пакета `rattle`. Этот файл имеет следующую структуру:

```
ID,Item
1,Sixth Sense
1,LOTR1
1,Harry Potter1
1,Green Mile
1,LOTR2
2,Gladiator
2,Patriot
2,Braveheart
```

К великому сожалению, русификация названий фильмов приводит к ошибке при выводе ассоциативных правил, поэтому нам придется знакомый всем "Властелин колец" скрывать под непонятной меткой "LOTR".

После загрузки файла и нажатия кнопки **Выполнить** получаем итоги выполнения алгоритма "Apriory":



Опция Freq Plot выполняет визуализацию графика частотного распределения встречаемости комбинаций (см. рис. 5.4б). Опция Show Rules приводит к формированию списка найденных ассоциативных правил, отсортированных по уровню поддержки (Support):

```
All Rules
     lhs                          rhs             support confidence lift
[1]  {Gladiator}               => {Patriot}       0.4     1.0000000   2.5000000
[2]  {Patriot}                 => {Gladiator}     0.4     1.0000000   2.5000000
[3]  {LOTR1}                   => {LOTR2}         0.2     1.0000000   5.0000000
[4]  {LOTR2}                   => {LOTR1}         0.2     1.0000000   5.0000000
[5]  {Harry Potter1}           => {Harry Potter2} 0.1     1.0000000  10.0000000
[6]  {Harry Potter2}           => {Harry Potter1} 0.1     1.0000000  10.0000000
…                 …                 …
[22] {Braveheart,Patriot}      => {Gladiator}     0.1     1.0000000   2.5000000
[23] {Gladiator,Patriot}       => {Braveheart}    0.1     0.2500000   2.5000000
[24] {LOTR1,LOTR2}             => {Green Mile}    0.1     0.5000000   2.5000000
[25] {Green Mile,LOTR1}        => {LOTR2}         0.1     1.0000000   5.0000000
[26] {Green Mile,LOTR2}        => {LOTR1}         0.1     1.0000000   5.0000000
[27] {Gladiator,Sixth Sense}   => {Patriot}       0.1     1.0000000   2.5000000
[28] {Patriot,Sixth Sense}     => {Gladiator}     0.1     1.0000000   2.5000000
[29] {Gladiator,Patriot}       => {Sixth Sense}   0.1     0.2500000   0.8333333
```

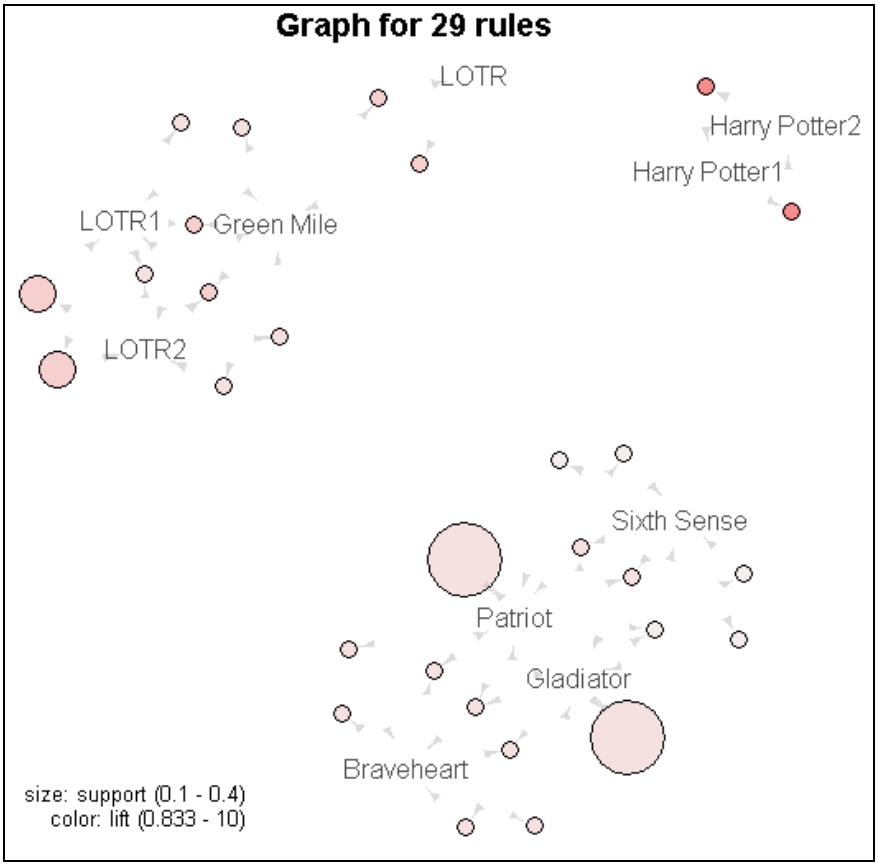
Вполне понятно, что посмотревшие первую серию "Властелина колец" или "Гарри Поттера" будут, вероятно, смотреть и вторую, а любители "Гладиатора" и "Патриота" захотят ознакомиться и с "Храбрым сердцем".

Для аналитиков, склонных изучать большое количество разных индексов, пакет позволяет вывести дополнительную таблицу мер "интересности" правил:

```
Interesting Measures
    chiSquared hyperLift hyperConfidence leverage oddsRatio         phi
1  10.00000000 1.3333333       0.9952381     0.24        NA  1.00000000
2  10.00000000 1.3333333       0.9952381     0.24        NA  1.00000000
3  10.00000000 1.0000000       0.9777778     0.16        NA  1.00000000
4  10.00000000 1.0000000       0.9777778     0.16        NA  1.00000000
5  10.00000000 1.0000000       0.9000000     0.09        NA  1.00000000
6  10.00000000 1.0000000       0.9000000     0.09        NA  1.00000000

22  1.66666667 1.0000000       0.6000000     0.06        NA  0.40824829
23  1.66666667 1.0000000       0.6000000     0.06        NA  0.40824829
24  1.40625000 0.5000000       0.6222222     0.06 7.0000000  0.37500000
25  4.44444444 1.0000000       0.8000000     0.08        NA  0.66666667
26  4.44444444 1.0000000       0.8000000     0.08        NA  0.66666667
27  1.66666667 1.0000000       0.6000000     0.06        NA  0.40824829
28  1.66666667 1.0000000       0.6000000     0.06        NA  0.40824829
29  0.07936508 0.3333333       0.1666667    -0.02 0.6666667 -0.08908708
```

Опция Plot выводит знакомый нам по рис. 5.7 граф сформированных правил, где достаточно четко прослеживаются покупательские предпочтения:


Graph for 29 rules
LOTR
Harry Potter2
Harry Potter1
LOTR1
Green Mile
LOTR2
Sixth Sense
Patriot
Gladiator
Braveheart
size: support (0.1 - 0.4)
color: lift (0.833 - 10)

Список рекомендуемой литературы

Айвазян С.А., Бухштабер В.М., Енюков И.С., Мешалкин Л.Д. Прикладная статистика: Классификация и снижение размерности М.: Финансы и статистика, 1989. 607 с.

Айвазян С.А., Мхитарян В.С. Прикладная статистика и основы эконометрии. М.: ЮНИТИ, 1998. 1022 с.

Арапов М.В., Ефимова Е.Н., Шрейдер Ю.А. О смысле ранговых распределений // Науч.-техн. информ. 1975. Сер. 2. № 1. С. 9–20.

Афифи А., Эйзен С. Статистический анализ: Подход с использованием ЭВМ. М.: Мир, 1982. 488 с.

Барсегян А.А., Куприянов М.С., Холод И.И. и др. Анализ данных и процессов. СПб.: БХВ-Петербург, 2009. 512 с.

Бирюкова С. Анализ последовательностей в R: TraMineR. НИУ ВШЭ. 2014. URL: https://www.hse.ru

Бокс Дж., Дженкинс Г. Анализ временных рядов. Прогноз и управление. М.: Мир, 1974. Вып. 1. 406 с.

Бонгард М. М. Проблема узнавания. М.: Наука, 1967. 320 с.

Вайнцвайг М. Н. Алгоритм обучения распознаванию образов «Кора». // Алгоритмы обучения распознаванию образов. Ред. В. Н. Вапник. М.: Сов. радио, 1973. С. 110-116.

Вапник В.Н. (ред.). Алгоритмы и программы восстановления зависимостей. М.: Наука, 1984. 816 с.

Вапник В.Н., Червоненкис А.Я. Теория распознавания образов. М.: Наука, 1974. 487 с.

Венэбльз У.Н., Смит Д.М. Введение в R. Заметки по R: среда программирования для анализа данных и графики. Вер. 3.1.0. Москва, 2014. 109 с.

Воробейчик Е.Л. О некоторых индексах ширины и перекрывания экологических ниш // Журн. общ. биологии. 1993. Т. 54, № 6. С. 706-712.

Горбань А.Н., Дунин-Барковский В.Л., Миркес Е.М. и др. Нейроинформатика. Новосибирск: Наука. Сиб. предприятие РАН, 1998. 296 с.

Горбач А.Н., Цейтлин Н.А. Покупательское поведение: анализ спонтанных последовательностей и регрессионных моделей в маркетинговых исследованиях. Киев: Освіта УкраЇны, 2011. 220 с.

Джеймс Г., Уиттон Д., Хасти Т., Тибширани Р. Введение в статистическое обучение с примерами на языке R. Пер. С.Э. Мастицкого. М.: ДМК Пресс, 2016. 450 с.

Джонгман Р.Г.Г., тер Браак С.Дж.Ф., ван Тонгерен О.Ф.Р. Анализ данных в экологии сообществ и ландшафтов. М.: РАСХН, 1999. 306 с.

Дрейпер Н., Смит Г. Прикладной регрессионный анализ. М.: Финансы и статистика. Кн. 1, 1986. 366 с. Кн. 2, 1987. 352 с.

Дэйвисон М. Многомерное шкалирование. Методы наглядного представления данных. М.: Финансы и статистика, 1988. 348с.

Дюк В.А., Самойленко А.П. Data Mining: учебный курс. СПб.: Питер, 2001. 368 с.

Загоруйко Н.Г. Прикладные методы анализа данных и знаний. Новосибирск: ИМ СО РАН, 1999. 270 с.

Зайцев К.С. Применение методов Data Mining для поддержки процессов управления IT-услугами. М.: МИФИ, 2009. 96 с.

Зарядов И.С. Введение в статистический пакет R: типы переменных, структуры данных, чтение и запись информации, графика. Москва: Изд-во РУДНБ, 2010а. 207 с.

Зарядов И.С. Статистический пакет R: теория вероятностей и математическая статистика. Москва: Изд-во РУДНБ, 2010б. 141 с.

Зиновьев А.Ю. Визуализация многомерных данных. Красноярск: КГТУ, 2000. 168 с.

Кабаков Р.И. R в действии: Анализ и визуализация данных в программе. М.: ДМК Пресс, 2014. 580 с.

Кендалл М., Стьюарт А. Многомерный статистический анализ и временные ряды. М.: Наука, 1976. 736 с.

Кендалл М., Стьюарт А. Статистические выводы и связи. М.: Наука, 1973. 899 с.

Ким Дж.-О., Мюллер Ч.У., Клекка У.Р. и др. Факторный, дискриминантный и кластерный анализ. М.: Финансы и статистика , 1989. 215 с.

Классификация и кластер. / Под ред. Дж. Вэн-Райзина. М.: Мир, 1980. 390 с.

Кобзарь А.И. Прикладная математическая статистика. Для инженеров и научных работников. М.: Физматлит, 2006. 816 с.

Краскэл Дж.Б. Многомерное шкалирование и другие методы поиска структуры // Статистические методы для ЭВМ / Под ред. К. Энслейна, Э. Рэлстона, Г. С. Уилфа. М.: Наука, 1986. С. 301-347.

Кудрин Б.И. Математика ценозов: видовое, ранговидовое, ранговое по параметру гиперболические H-распределения и законы Лотки, Ципфа, Парето, Мандельброта // Математический аппарат структурного описания ценозов и гиперболические H-ограничения. Ценологические исследования. М.: Центр системн. исслед., 2002. Вып. 19. С. 357-412.

Кшнясев И.А. Анализ обилия организмов: мультимодельный вывод как альтернатива проверки нуль-гипотезы. // Биологические системы: устойчивость, принципы и механизмы функционирования. Сб. материалов III Всеросс. Науч.-практ. Конф. Нижний Тагил, 2010. С. 348-353. URL: https://www.ipae.uran.ru/user/69.

Лбов Г.С. Методы обработки разнотипных экспериментальных данных. Новосибирск: Наука, 1981. 160 с.

Ллойд Э., Ледерман У. (ред). Справочник по прикладной статистике. В 2-х т. М.: Финансы и статистика. Т. 1. 1989. 510 с., Т. 2. 1990. 526 с.

Люк Д. Анализ сетей (графов) в среде R. Руководство пользователя. М.: ДМК Пресс, 2016.

Мастицкий С.Э. Визуализация данных с помощью ggplot2. М.: ДМК Пресс, 2016.

Мастицкий С.Э., Шитиков В.К. Статистический анализ и визуализация данных с помощью R. М.: ДМК Пресс, 2015. 496 с. URL: ievbras.ru/ecostat.

Налимов В.В. Теория эксперимента. М.: Наука, 1971. 207 с.

Огнева Д. Пакет “arules” системы R. МГУ им. Ломоносова. 2012. URL: www.machinelearning.ru

Розенберг Г.С. Введение в теоретическую экологию. / В 2-х т.; Тольятти: Кассандра, 2013. Т. 1. 565 с. URL: ievbras.ru/ecostat

Розенберг Г.С., Шитиков В.К., Брусиловский П.М. Экологическое прогнозирование (Функциональные предикторы временных рядов). Тольятти: ИЭВБ РАН, 1994. 185 с. URL: ievbras.ru/ecostat

Самарский А.А. Что такое вычислительный эксперимент? // Наука и жизнь. 1979. № 3. С. 27-33.

Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. М.: ДМК Пресс, 2015. 400 с.

Форрестер Дж. Антиинтуитивное поведение сложных систем // Современные проблемы кибернетики. М.: Знание, 1977. С. 9-25.

Хокинс Д., Блейксли С. Об интеллекте. М.: ООО «И.Д.Вильямс», 2007. 204 с.

Храмов Д.А. Сбор данных в Интернете на языке R. М.: ДМК Пресс, 2016.

Хромов-Борисов Н.Н. Синдром статистической снисходительности или значение и назначение *p*-значения // Телеконференция по медицине, биологии и экологии, 2011. №4. URL: http://tele-conf.ru .

Чубукова И.А. Data Mining. Курс лекций интернет-университета INTUIT, 2006. 328 с.

Шипунов А.Б. и др. Наглядная статистика. Используем R! М.: ДМК Пресс, 2014. 298 с.

Шитиков В.К. Экотоксикология и статистическое моделирование эффекта с использованием R. Тольятти: ИЭВБ РАН , 2016. 149 с. URL: ievbras.ru/ecostat.

Шитиков В.К., Зинченко Т.Д., Розенберг Г.С. Макроэкология речных сообществ: концепции, методы, модели. Тольятти: СамНЦ РАН, Кассандра, 2012. 257 с. URL: ievbras.ru/ecostat .

Шитиков В.К., Розенберг Г.С. Рандомизация и бутстреп: статистический анализ в биологии и экологии с использованием R. Тольятти : Кассандра, 2014. 314 с. URL: URL: ievbras.ru/ecostat.

Эфрон Б. Нетрадиционные методы многомерного статистического анализа. М.: Финансы и статистика, 1988. 263 с.

Abdi H., Williams L. Principal component analysis // WIREs Computational Statistics. 2010. V. 2 (4). P. 433-459. URL: http://onlinelibrary.wiley.com

Agrawal R., Srikant R. Fast Discovery of Association Rules. // Proc. of the 20th Intern. Conf. on VLDB. Santiago, Chile, 1994.

Agresti A. An introduction to categorical data analysis. Wiley. 2007. 372 p.

Anderson D. R. Model-based inference in life sciences: A primer on evidence. Springer, 2008. 184 p.

Banfield J., *Raftery* A. Model-Based Gaussian and Non-Gaussian Clustering. // Biometrics. V. 49. P. 803-821.

Bates D.M., *Watts D.G.* Nonlinear Regression Analysis and Its Applications. New-York: John Wiley, 2007. 392 p.

Bezdek J.C. Pattern Recognition with Fuzzy Objective Function Algorithms. New York: Plenum Press, 1981.

Borcard D., Gillet F., Legendre P. Numerical Ecology with R. N.Y.: Springer, 2011. 306 p.

Box G., Cox D.R. An analysis of Transformation // Journal of Royal Statistical Society B. 1964. V. 26. P. 211-243.

Boyce R.L, Ellison P.C Choosing the best similarity index when performing fuzzy set ordination on binary data // Journal of Vegetation Science. 2001. V. 12. P. 711-720.

Breiman L. Random forests // Machine Learning. 2001. V. 45 (1). P. 5-32.

Breiman L., Friedman J.H., Olshen R.A. et al. Classification and Regression Trees. Belmont (CA): Wadsworth Int. Group, 1984. 368 p.

Burnham K.P., Anderson D.R. Model selection and multimodel inference: a practical information-theoretic approach. N.Y.: Springer-Verlag, 2002. 496 p.

Cameron A.C, Trivedi P.K. Regression Analysis of Count Data. Cambridge University Press, Cambridge. 2013.

Chiu YW. Machine learning with R. Cookbook. Packt Publishing, 2015.

Cristianini N., Shawe-Taylor J. An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, 2000.

Davison A.C., Hinkley D.V. Bootstrap methods and their application. Cambridge: Cambridge University Press, 2006. 592 p.

De'Ath G. Multivariate regression trees: a new technique for modeling species environment relationships // Ecology. 2002. V.83. P. 1105-1117.

Delgado M.F., Cernadas E., Barro S., Amorim D. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? // Journal of Machine Learning Research. 2014. V. 15. P. 3133-3181.

Edgington E.S. Randomization tests. N.Y.:Marcel Dekker, 1995. 341 p.

Everitt B.S., Howell D.C. Encyclopedia of Statistics in Behavioral Science. Chichester: Wiley & Sons Ltd., 2005. 2132 p.

Faraway J.J. Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models. Chapman, Hall/CRC, 2006. 345 p.

Fox J. Applied Regression Analysis and Generalized Linear Models, 2nd ed. Sage Publications, 2008.

Gabadinho A., Ritschard G., Muller N.S., Studer M. Analyzing and Visualizing State Sequences in R with TraMineR. // Journal of Statistical Software. 2011. V.40 (4). p. 1-37. URL: http://www.jstatsoft.org/v40/i04/.

Gabadinho A., Ritschard G., Muller N.S., Studer M. Mining sequence data in R with the `TraMineR` package: A user's guide. University of Geneva, Switzerland. 2011a. URL: http://mephisto.unige.ch/traminer/

Gibb S., Strimmer K. Differential protein expression and peak selection in mass spectrometry data by binary discriminant analysis. // Bioinformatics. 2015. 31(19), 3156–3162. URL: http://strimmerlab.org/software/binda

Glur C. `data.tree` sample applications. 2016. URL: http://ipub.com/

Goddard M.J., Hinberg I. Receiver operator characteristic (ROC) curves and non-normal data: An empirical study. // Statistics in Medicine. 1989. Vol. 9 (3). P. 325–337.

Hahsler M., Grun B, Hornik K., Buchta C. Introduction to arules – A computational environment for mining association rules and frequent item sets. URL: https://cran.r-project.org

Hand D. Classifier Technology and the Illusion of Progress // Statistical Science. 2006. V. 21. P. 1-14.

Hastie T., Tibshirani R., Friedman J. The Elements of Statistical Learning: Data Mining, Inference and Prediction. N.Y.: Springer-Verlag, 2009. 763 p

Hunt E.B., Marin J., Stone P.J. Experiments in Induction. New York: Academic Press, 1966.

Kassambara A. Practical Guide to Cluster Analysis in R: Unsupervised Machine Learning. Statistical tools for high-throughput data analysis STHDA, 2017. URL: http://www.sthda.com/

Kaufman L., Rousseeuw P.J. Finding Groups in Data. An Introduction to Cluster Analysis. Hoboken (NJ): Wiley & Sons Inc., 2005. P. 342.

Kohonen T. Self-organized formation of topologically correct feature maps // Biological Cybernetics. 1982. № 43. P. 59-69.

Kruskal J.B. Multidimensional Scaling by Optimizing Goodness-of-Fit to a Nonmetric Hypothesis. // Psychometrika. 1964. V. 29. P. 1–28.

Kuhn M. Building Predictive Models in R Using the caret Package // Journal of Statistical Software. 2008. V. 5. P. 113-142. DOI 10.18637/jss.v028.i05

Kuhn M. Predictive Modeling with R and the caret Package. 2013. URL: https://www.r-project.org/

Kuhn M. Variable Selection Using The caret Package. 2012. URL: http://citeseerx.ist.psu.edu

Kuhn M., Johnson K. Applied Predictive Modeling. Springer-Verlag New-York, 2013. 574 p.

Kursa M., Rudnicki W. Feature Selection with the Boruta Package // Journal of Statistical Software. 2010. V.36 (11). P. 2-12.

Le S., Josse J, Husson F. FactoMineR: An R Package for Multivariate Analysis // Journal of Statistical Software. 2008. V. 25 (1).

Legendre P, Legendre L. Numerical Ecology. Amsterdam: Elsevier Sci. BV, 2012.

Legendre P., Gallagher E. Ecologically meaningful transformations for ordination of species data // Oecologia. 2001. V. 129. P. 271-280.

Leskovec J., Rajaraman A., Ullman J. The Mining of Massive Datasets. Cambridge University Press. 2014. URL: http://www.mmds.org/#book

Loh W.-Y, Shih Y.-S. Split selection methods for classification trees. // Statistica Sinica1997. V. 7. P. 815-840.

MacQueen J. Some methods for classification and analysis of multivariate observations. // The Fifth Berkeley Symposium on Mathematical Statistics and Probability. eds L.M. Le Cam, J. Neyman. Berkeley, CA: University of California Press. 1967. P. 281–297.

Maindonald J., Braun W.J. Data Analysis and Graphics Using R. Cambridge: University Press, 2010. 525 p.

McArdle B.H., Anderson M.J. Fitting multivariate models to community data: a comment on distance-based redundancy analysis // Ecology. 2001. V. 82, № 1. P. 290-297.

McCune B., Grace J.B., Urban D.L. Analysis of Ecological Communities. Gleneden Beach (OR): MjM Software, 2002. 285 p.

Mitchell M. An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press. 1996.

Mount J., Zumel N. Exploring Data Science. Manning Publications Co., 2016. 184 p.

Mount J., Zumel N. Practical Data Science with R. Manning Publications Co., 2014. 416 p.

Myers W.L., Patil G.P. Multivariate Methods of Representing Relations in R for Prioritization Purposes: Selective Scaling, Comparative Clustering, Collective Criteria, and Sequenced Sets. N.-Y.: Springer, 2012. 297 p.

Oksanen J., Blanchet F.G., Kindt R. et al. `vegan`: Community Ecology Package. R package version 2.0-2. 2011. URL: https://cran.r-project.org/.

Posthuma L., Suter G.W.H., Traas T.P. Species Sensitivity Distributions in Ecotoxicology. CRC Press, 2001. 616 p.

Quinlan J.R. Induction of Decision Trees. // Mach. Learn. 1986. V. 1 (1). P. 81-106

Ripley B.D. Pattern Recognition and Neural Networks. Cambridge University Press, 1996.

Sakurai S. Theory and Applications for Advanced Text Mining. InTech Chapters published, 2012. URL: http://www.intechopen.com/

Serneels S., Van Espen P., De Nolf E. Spatial sign preprocessing: a simple way to impart moderate robustness to multivariate estimators. // J. Chem. Inf. Model. 2006. V. 46 (3). P. 1402-1409

Shimodaira H. An approximately unbiased test of phylogenetic tree selection // Syst. Biol. 2002. V. 51. P. 492-508.

Stanton JM. Introduction To Data Science. Syracuse University, 2013.

Torgo L. Data mining with R : learning with case studies. Chapman & Hall/CRC, 2011. 272 p.

Vapnik V.N. The Nature of Statistical Learning Theory. Springer, 1995

Wang W., Yang J. Mining Sequential Patterns from Large Data Sets. N.-Y.: Springer, 2005. 162 p.

Williams G. Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery. N.-Y.: Springer, 2011. 365 p.

Williams G. Rattle: A Data Mining GUI for R // Journal of Statistical Software. 2009. V. 2(1). P. 45-55. URL: http://journal.r-project.org

Wood S.N. Generalized Additive Models: An Introduction with R. Chapman, Hall/CRC, 2006. 410 p.

Chiu Y-W. (Chiu D.). Machine Learning with R Cookbook. Packt Publising, 2015. 405 p.

Zacharski R. A Programmer's Guide to Data Mining. 2015. URL: http://guidetodatamining.com/

Zafarani R., Abbasi M., Liu H. Social Media Mining. An Introduction. Cambridge University Press, 2014. URL: http://dmml.asu.edu/smm/

Zaki M., Meira W. Data Mining and Analysis. Fundamental Concepts and Algorithms. Cambridge University Press, 2014. URL: http://www.dataminingbook.info/

Zeileis A., Kleiber C., Jackman S. Regression Models for Count Data in R. // Journal of Statistical Software. 2008. V. 27(8). URL: https://www.jstatsoft.org

Zhao Y. R and Data Mining: Examples and case studies. Academic Press, 2012. 256 p.

Zhao Y., Zhang C., Cao L. Post-Mining of Association Rules: Techniques for Effective Knowledge Extraction. Information Science Reference. 2009.

Zhao Y., Cen Y. Data Mining Applications with R. Academic Press, 2014. 470 p.

Zuur A. F., Ieno E.N., Walker N. et al. Mixed Effects, Models and Extensions in Ecology with R. Berlin: Springer Sci., 2009. 574 p. URL: www.highstat.com

Приложение

R: Справочная карта по Data Mining

Приведена по Yanchang Zhao, yanchang@rdatamining.com (с незначительными изменениями и дополнениями). Последнюю версию см. на http://www.RDataMining.com. Имена пакетов представлены в круглых скобках с подчеркиванием. Рекомендуемые пакеты и функции показаны жирным шрифтом.

СТАТИСТИКА (STATISTICS)

Описательная статистика (Summarization)

summary()	обобщение результатов
describe()	краткие описательные статистики данных (Hmisc)
boxplot.stats()	диаграммы размахов (box plot) и сопряженные с ними статистики

Дисперсионный анализ (Analysis of Variance)

aov()	оценивание и вывод таблицы дисперсионного анализа
anova()	расчет таблицы анализа дисперсий (или девианса) для одной или более моделей

Статистические тесты (Statistical Tests)

chisq.test()	тест хи-квадрат для таблиц сопряженности и оценки качества подгонки моделей
ks.test()	тест Колмогорова-Смирнова
t.test()	тест по *t* -критерию Стьюдента
prop.test()	тест на значимость заданной пропорции или их эквивалентность
binom.test()	точный биномиальный тест

Регрессионный анализ (Regression Functions)

lm()	линейные модели
glm()	обобщенные линейные модели
gbm()	обобщенные регрессионные бустинг-модели (gbm)
predict()	метод для получения предсказанных значений
residuals()	остатки модели, разности наблюдаемых и предсказанных значений
nls()	нелинейная регрессия
gls(), gnls()	построение линейных и нелинейных моделей методом обобщенных наименьших квадратов (nlme)

Модели со смешанными эффектами (nlme) (Mixed Effects Models)

lme(), nlme()	линейные и нелинейные модели со смешанными эффектами

Principal Components and Factor Analysis

princomp(), prcomp()	анализ главных компонент и факторный анализ

Обнаружение выбросов (Outlier Detection)

boxplot.stats()$out	перечень наблюдений, выходящих за пределы интервала,

	который ограничен "усами" диаграммы размахов
lofactor()	расчет фактора локальных выбросов по LOF-алгоритму (DMwR или dprep)
lof()	параллельная реализация LOF-алгоритма (Rlof)

Прочие функции

var(), cov(), cor()	дисперсия, ковариация, корреляция
density()	вычисление оценки ядерной плотности
cmdscale()	многомерное шкалирование (MDS)

Пакеты

gbm	обобщенные регрессионные бустинг-модели
nlme	линейные и нелинейные модели со смешанными эффектами
Rlof	параллельная реализация LOF-алгоритма
extremevalues	поиск экстремальных значений в одномерных данных
outliers	использование нескольких общих методов обнаружения выбросов
mvoutlier	поиск многомерных выбросов с использованием робастных методов

АНАЛИЗ ВРЕМЕННЫХ РЯДОВ (TIME SERIES ANALYSIS)

Преобразование и отображение (Construction & Plot)

ts()	создание объектов класса "временной ряд"
plot.ts()	метод для визуализации объектов класса "временной ряд"
smoothts()	сглаживание временных рядов (ast)
sfilter()	удаление сезонных флуктуаций с использованием скользящего среднего (ast)

Декомпозиция (Decomposition)

decomp()	декомпозиция временного ряда по фильтру квадратного корня (timsac)
decompose()	классическая сезонная декомпозиция по скользящему среднему
stl()	сезонная декомпозиция временного ряда по локальной регрессии
tsr()	декомпозиция временного ряда (ast)
ardec()	декомпозиция временного ряда по авторегрессии (ArDec)

Прогнозирование (Forecasting)

arima()	построение моделей ARIMA для одномерного временного ряда
predict.Arima()	прогнозируемые значения по моделям ARIMA
auto.arima()	подгонка оптимальной модели ARIMA для одномерного ряда (forecast)
forecast.stl(), forecast.ets(), forecast.Arima()	прогнозирование ряда с использованием STL, ETS и ARIMA моделей (forecast)

Корреляция и ковариация (Correlation and Covariance)

acf()	автокорреляция и автоковариация временного ряда
ccf()	кросс-ковариация и кросс-корреляция двух одномерных временных рядов

Пакеты

forecast	анализ и визуализация прогнозов одномерного временного ряда
hts	анализ и прогнозирование иерархических и сгруппированных рядов
TSclust	функции для кластеризации временных рядов
dtw	динамическая трансформация шкалы времени (Dynamic Time Warping, DTW)
timsac	функции для анализа и преобразования временных рядов
ast	анализ временных рядов
ArDec	декомпозиция временного ряда, основанная на авторегрессии
dse	средства для создания многомерных, линейных и инвариантных моделей временных рядов

ФУНКЦИИ ДЛЯ ВЫДЕЛЕНИЯ АССОЦИАТИВНЫХ ПРАВИЛ И ПАТТЕРНОВ ПОСЛЕДОВАТЕЛЬНОСТЕЙ (ASSOCIATION RULES AND SEQUENTIAL PATTERNS FUNCTIONS)

apriori()	поиск ассоциаций по алгоритму APRIORI, который последовательно подсчитывает частоты одновременно происходящих событий, априори отсекая маловероятные (arules)
eclat()	поиск частых наборов событий по алгоритму Eclat, который перебирает классы эквивалентности и их пересечения (arules)
cspade()	поиск частых фрагментов последовательностей по алгоритму cSPADE (arulesSequences)
seqefsub()	поиск частых подпоследовательностей (TraMineR)

Пакеты

arules	поиск частых, максимальных или закрытых наборов событий и ассоциативных правил с использованием двух алгоритмов: Apriori и Eclat.
arulesViz	визуализация ассоциативных правил
arulesSequences	дополнение к arules для обработки и выделения частых последовательностей
TraMineR	поиск, описание и визуализация последовательностей объектов или событий

КЛАССИФИКАЦИЯ И ПРЕДСКАЗАНИЕ (CLASSIFICATION & PREDICTION)

Деревья решений (Decision Trees)

ctree()	деревья условного вывода, рекурсивное разбиение для непрерывных, цензурированных, упорядоченных, категориальных и многомерных откликов в рамках условного вывода (party)
rpart()	деревья рекурсивного разбиения и регрессии (rpart)
mob()	рекурсивное разбиение, приводящее к созданию деревьев, конечные узлы которого содержат статистические модели для соответствующих поднаборов данных (party)
varimp()	важность предикторов (party)

Случайный лес (Random Forest)

cforest()	ансамбли моделей, создаваемые с использованием алгоритмов "случайный лес" и "бэггинг" (party)

randomForest()	случайный лес (randomForest)
importance()	важность предикторов (randomForest)

Нейронные сети (Neural Networks)

nnet()	построение нейронной сети с одним скрытым слоем (nnet)
neuralnet()	обучение нейронных сетей (neuralnet)
mlp(), dlvq(), rbf(), rbfDDA(), elman(), jordan(), som(), art1(), art2(), artmap(), assoz()	различные типы нейронных сетей (RSNNS)

Машины опорных векторов (Support Vector Machine - SVM)

svm()	обучение машины опорных векторов для регрессии, классификации или оценки плотности вероятности (e1071)
ksvm()	машины опорных векторов (kernlab)

Байесовские классификаторы (Bayes Classifiers)

naiveBayes()	наивный баейсовский классификатор (e1071)

Оценка эффективности моделей (Performance Evaluation)

performance()	рассчитывает различные меры качества предсказательных моделей (ROCR)
PRcurve()	кривые чувствительности и специфичности классификатора (DMwR)
CRchart()	графики кумулятивной чувствительности классификатора (DMwR)
roc()	построение ROC-кривых (pROC)
auc()	вычисление площади под ROC-кривой (pROC)
ROC()	визуализация ROC-кривой (DiagnosisMed)

Пакеты

party	рекурсивное разбиение
rpart	деревья рекурсивного разбиения и регрессии
rpartOrdinal	деревья классификации для откликов с упорядоченными категориями
rpart.plot	метод для визуализации rpart-моделей
randomForest	классификация и регрессия на основе случайного леса
caret	модели классификации и регрессии
nnet	нейронные сети встречного распространения и лог-линейные модели для откликов с несколькими классами
RSNNS	реализация Штутгартского Симулятора Нейронных Сетей в R (SNNS)
neuralnet	обучение нейронных сетей обратного распространения и устойчивого обратного распространения с учетом или без учета весовых коэффициентов
e1071	функции для анализа латентных классов, Фурье-преобразования коротких временных рядов, нечеткой кластеризации, обучения машин опорных векторов, вычисления кратчайшего пути, бэггинг-кластеризации, построения наивного байесовского классификатора и др.
ROCR	визуализация результатов оценки качества классификаторов
pROC	визуализация и анализ ROC-кривых

КЛАСТЕРИЗАЦИЯ (CLUSTERING)

Кластеризация, основанная на разбиение (Partitioning based Clustering)

Разбиение данных на *k* групп с последующей попыткой улучшить качество кластеризации путем перемещения объектов из одной группы в другую

kmeans()	выполняет кластеризацию по методу *k* средних для некоторой матрицы с данными
kmeansruns()	вызывает функцию kmeans() для выполнения кластеризации по методу *k* средних и выполняет нахождение оптимального числа кластеров с использованием нескольких начальных расположений их центров (fpc)
pam()	реализация метода "разделение вокруг медоидов" (PAM) (cluster)
pamk()	реализация метода "разделение вокруг медоидов" (PAM) с одновременным нахождением оптимального числа кластеров (fpc)
kmeansCBI()	интерфейс для взаимодействия с функциями из пакета kmeans (fpc)
cluster.optimal()	поиск оптимального числа кластеров в некотором наборе данных (bayesclust)
clara()	кластеризация для больших наборов данных (cluster)
fanny(x,k,…)	выполнение нечеткой кластеризации с *k* кластерами (cluster)
kcca()	кластеризация по *k* центроидам (flexclust)
ccfkms()	кластеризация с использованием конъюгатных выпуклых функций (cba)
apcluster()	кластеризация по методу "передачи сообщений" (affinity propagation) на основе входной матрицы сходств (apcluster)
apclusterK()	кластеризация по методу "передачи сообщений" для нахождения *k* кластеров (apcluster)
cclust()	выпуклая кластеризация, включая метод *k* средних и два других метода (cclust)
KMeansSparseCluster()	кластеризация по методу *k* средних с одновременным нахождением информативных переменных (sparcl)
tclust(x, k, alpha,…)	кластеризация по методу *k* усеченных средних (часть наблюдений удаляется из рассмотрения) (tclust)

Иерархическая кластеризация (Hierarchical Clustering)

Иерархическая декомпозиция данных либо снизу вверх (агломерация), либо сверху вниз (разделение)

hclust()	иерархический кластерный анализ по матрице расстояний
birch()	алгоритм BIRCH для кластеризации очень больших объемов данных с использованием CF-деревьев (birch)
pvclust()	иерархическая кластеризация с одновременным вычислением *p*-значений путем извлечения бутстреп-выборок разного размера (pvclust)
agnes()	агломеративный иерархический кластерный анализ (cluster)
diana()	иерархический кластерный анализ на основе разделения (cluster)

`mona()`	иерархический кластерный анализ на основе разделения для данных, представленных только бинарными переменными (`cluster`)
`rockCluster()`	кластеризация с использованием алгоритма Rock (`cba`)
`proximus()`	кластеризация на основе алгоритма Proximus для данных, представленных только бинарными переменными (`cba`)
`isopam()`	алгоритм кластеризации Isopam (`isopam`)
`flashClust()`	оптимальная иерархическая кластеризация (`flashClust`)
`fastcluster()`	быстрая иерархическая кластеризация (`fastcluster`
`cutreeDynamic(), cutreeHybrid()`	выделение кластеров на дендрограммах (`dynamicTreeCut`)
`Hierarchical SparseCluster()`	иерархическая кластеризация с одновременным нахождением информативных переменных (`sparcl`)

Модели, основанные на кластерах (Model based Clustering)

`Mclust()`	кластеризация на основе статистических моделей (`mclust`)
`HDDC()`	кластеризация на основе статистических моделей для данных большой размерности (`HDclassif `)
`fixmahal()`	кластеризация с использованием фиксированных точек и расстояния Махаланобиса (`fpc`)
`fixreg()`	кластеризация на основе регрессии по фиксированным точкам (`fpc`)
`mergenormals()`	кластеризация, основанная на слиянии компонент смешанного гауссова распределения (`fpc`)

Кластеризация, основанная на плотности (Density based Clustering)

Формирование кластеров путем объединения участков с плотным расположением точек

`dbscan(data, eps, MinPts,…)`	нахождение кластеров произвольной формы с использованием параметров `eps` (радиус, в пределах которого лежат точки-соседи) и `MinPts` (пороговое значение плотности расположения точек) (`fpc`)
`pdfCluster()`	кластеризация на основе ядерной плотности вероятности (`pdfCluster`)

Другие техники кластеризации (Other Clustering Techniques)

`mixer()`	кластеризация на основе случайных графов (`mixer`)
`nncluster()`	быстрая кластериазация на основе алгоритма "restarted minimum spanning tree" (`nnclust`)
`orclus()`	кластеризация на основе алгоритма ORCLUS (`orclus`)

Отображение результатов кластеризации (Plotting Clustering Solutions)

`plotcluster()`	визуализация групп данных (`fpc`)
`bannerplot()`	горизонтально ориентированный "биплот", изображающий результат иерархической кластеризации (`cluster`)

Оценка качества кластеризации (Cluster Validation)

`silhouette()`	вычисление и извлечение информации по "силуэтам" кластеров (`cluster`)
`cluster.stats()`	вычисление нескольких статистик качества кластеризации на основе матрицы расстояний (`fpc`)

`clValid()`	вычисление статистик качества для нескольких алгоритмов кластеризации и числа кластеров (`clValid`)
`clustIndex()`	вычисление нескольких индексов кластеризации, которые можно использовать для нахождения оптимального числа кластеров (`cclust`)
`NbClust()`	позволяет вычислить 30 индексов для оценки качества кластеризации и нахождения оптимального числа кластеров (`NbClust`)

Пакеты

`cluster`	кластерный анализ
`fpc`	различные методы кластеризации и валидации полученных решений
`mclust`	кластеризация, основанная на статистических моделях
`birch`	кластеризация очень больших наборов данных с использованием алгоритма BIRCH
`pvclust`	иерархическая кластеризация с расчетом *p*-значений
`apcluster`	кластеризация на основе метода "передачи сообщений"
`cclust`	методы выпуклой кластеризации, включая алгоритм *k* средних, алгоритм обновления на новых данных, алгоритм "нейронного газа", а также вычисление индексов для нахождения оптимального числа кластеров
`cba`	кластеризация для решения бизнес-задач, включая такие алгоритмы, как Proximus и Rock
`bclust`	байесовская кластеризация на основе иерархической модели, подходящая для нахождения групп в данных большой размерности
`biclust`	алгоритмы для нахождения двумерных кластеров
`clue`	ансамбли кластерных решений
`clues`	метод кластеризации, основанный на локальной регуляризации
`clValid`	валидация кластерных решений
`clv`	методы валидации кластерных решений, включая популярные методы внутренней и внешней валидации
`cluster.bayesclust`	статистические тесты для валидации кластеров, полученных на основе геномных данных
`clustsig`	анализ статистической значимости кластеров, а также разницы между кластерами
`clusterSim`	поиск оптимальной процедуры кластеризации для имеющихся данных
`clusterGeneration`	имитация кластеров
`gcExplorer`	инструмент для графического анализа результатов кластеризации
`hybridHclust`	гибридный иерархический кластерный анализ на основе "совместных кластеров"
`Modalclust`	иерархический кластерный анализ на основе мод
`iCluster`	кластеризация на основе разнородных генетических данных
`EMCC`	эволюционные методы Монте-Карло (EMC) для кластеризации
`rEMM`	расширяемая марковская модель (EMM) для кластеризации потоковых данных

ОБРАБОТКА ТЕКСТОВ (TEXT MINING)

Импорт, очистка и подготовка текстов (Importing Text, Cleaning and Preparation)

`readPDF()`	извлечение текста и метаданных из документов формата PDF (`tm`)
`Corpus()`	построение корпуса, т.е. коллекции из нескольких документов (`tm`)
`tm_map()`	преобразование текстовых документов, т.е. стемминг, удаление стоп-слов и т.д. (`tm`)
`tm_filter()`	отфильтровывание документов из корпуса (tm)
`TermDocumentMatrix()`, `DocumentTermMatrix()`	создание терм-документных и документ-термных матриц (`tm`)
`Dictionary()`	создание словаря их текстового вектора или терм-документной матрицы (`tm`)
`stemDocument()`	стемминг слов в документе (`tm`)
`stemCompletion()`	восстановление полной формы слов после стемминга (`tm`)
`SnowballStemmer()`	стемминг по алгоритму Snowball (`Snowball`)
`stopwords(language)`	возвращает стоп-слова из разных языков (`tm`)
`removeNumbers()`, `removePunctuation()`, `removeWords()`	удаление чисел, знаков пунктуации или некоторого набора слов из документа (`tm`)
`removeSparseTerms()`	удаление редких термов из терм-документной матрицы (`tm`)

Обнаружение часто встречающихся термов и ассоциаций (Frequent Terms and Association)

`findAssocs()`	находит связи между термами в терм-документных матрицах (`tm`)
`findFreqTerms()`	находит часто встречающиеся термы в терм-документных матрицах (`tm`)
`termFreq()`	формирование вектора с частотами термов для заданного документа (`tm`)

Тематическое моделирование (Topic Modelling)

`LDA()`	подгонка LDA-модели (Latent Dirichlet Allocation) (`topicmodels`)
`CTM()`	подгонка CTM-модели (Correlated Topics Model) (`topicmodels`)
`terms()`	извлечение наиболее вероятных термов для заданной темы (`topicmodels`)
`topics()`	извлечение наиболее вероятных тем для заданного документа (`topicmodels`)
`polarity()`	индекс полярности (в анализе тональности текстов) (`qdap`)
`textcat()`	классификация текстов на основе *n*-грам (`textcat`)

Визуализация текста (Text Visualization)

`wordcloud()`	создание "облака слов" (`wordcloud`)
`comparison.cloud()`	создание "облака слов" для сравнения частоты встречаемости этих слов в разных документах (`wordcloud`)

commonality.cloud()	"облако слов", общих для нескольких документов (wordcloud)

Пакеты

tm	набор утилит для анализа текстов
topicmodels	подгонка LDA- и CTM-моделей
wordcloud	создание "облака слов"
lda	подгонка LDA-моделей
wordnet R	интерфейс к лексической базе данных WordNet
RTextTools	автоматическая классификация документов путем обучения с учителем
qdap	набор утилит для анализа естественного языка и документов
sentiment140	анализ тональности текстов с использованием бесплатного сервиса sentiment140
tm.plugin.dc	дополнительный модуль для пакета tm, позволяющий организовать распределенные вычисления при анализе текстов
tm.plugin.mail	дополнительный модуль для пакета tm, облегчающий работу с текстами электронной почты
textir	набор утилит для выполнения анализа тональности текста и оценивания статистических различий между документами
tau	набор утилит для анализа текстов

АНАЛИЗ СОЦИАЛЬНЫХ СЕТЕЙ И ФУНКЦИИ ДЛЯ РАБОТЫ С ГРАФАМИ (SOCIAL NETWORK ANALYSIS AND GRAPH MINING FUNCTIONS)

graph(), graph.edgelist(), graph.adjacency(), graph.incidence()	создание графов на основе структур данных таких типов, как "ребра" (edges), "список ребер" (edge list), "матрица расстояний" (adjacency matrix) и "матрица встречаемости" (incidence matrix) соответственно (igraph)
plot(), tkplot(), rglplot()	статичные, интерактивные и трехмерные изображения графов (igraph)
gplot(), gplot3d()	визуализация графов (sna)
vcount(), ecount()	подсчет числа ребер и узлов (igraph)
V(), E()	доступ к узлам и ребрам графа (igraph)
is.directed()	является ли граф направленным? (igraph)
are.connected()	связаны ли два узла графа? (igraph)
degree(), betweenness(), closeness(), transitivity(), evcent()	различные меры центральности графа (igraph, sna)
edge.density()	плотность графа (igraph)
add.edges(), add.vertices(), delete.edges(), delete.vertices()	добавление узлов и ребер (igraph)
neighborhood()	нахождение соседей заданного узла графа (igraph, sna)
get.adjlist()	получение списков соседних узлов или ребер (igraph)
nei(), adj(), from(), to()	индексирование узлов и ребер графа (igraph)

`cliques()`, `largest.cliques()`, `maximal.cliques()`, `clique.number()`	обнаружение клик, т.е. полных подграфов (igraph)
`clusters()`, `no.clusters()`	нахождение максимально связанных элементов графа и их количества (igraph)
`fastgreedy.community()`, `spinglass.community()`	алгоритмы обнаружения сообществ в графах (igraph)
`cohesive.blocks()`	нахождение "сцепленных блоков" (кластеров) в графах (igraph)
`induced.subgraph()`	извлечение части графа (igraph)
`mst()`	алгоритм минимального остовного дерева (igraph)
`components()`	нахождение максимально связанных компонентов графа (igraph)
`shortest_paths()`	нахождение кратчайшего пути между узлами (igraph)
`%->%, %<-%, %--%`	операторы индексирования ребер графа (igraph)
`get.edgelist()`	возвращает список ребер в виде матрицы с двумя столбцами (igraph)
`read.graph()`, `write.graph()`	импорт и экспорт графов в виде файлов разных форматов (igraph)

Пакеты

igraph	анализ и визуализация графов
sna	анализ социальных сетей
d3Network, networkD3	R-интерфейс к JavaScript-библиотеке D3 для построения графов, деревьев, дендрограмм и диаграмм Санки
RNeo4j	взаимодействие с базами данных Neo4j из среды R
statnet	набор инструментов для описания, визуализации, анализа и имитации графов
egonet	меры центральности для анализа социальных сетей
network	инструменты для создания и модификации графов
bipartite	визуализация двураздельных графов и вычисление некоторых описательных статистик
blockmodeling	обобщенное и классическое моделирование блоков в размеченных графах
diagram	визуализация простых графов (сетей), построение потоковых диаграмм
NetCluster	кластеризация элементов графа
NetData	наборы данных к лабораторным работам по анализу социальных сетей с помощью R от McFarland et al.
NetIndices	расчет различных индексов, включая индексы для описания структуры пищевых сетей
NetworkAnalysis	оценка статистических различий между взвешенными или невзвешенными графами
tnet	анализ взвешенных, бимодальных и динамических графов

ФУНКЦИИ ДЛЯ РАБОТЫ С ПРОСТРАНСТВЕННЫМИ ДАННЫМИ (SPATIAL DATA ANALYSIS FUNCTIONS)

`geocode()`	геокодирование с использованием сервиса Google Maps (ggmap)

plotGoogleMaps()	визуализация пространственных данных на картах Google Maps (plot-GoogleMaps)
qmap()	быстрое построение карт (ggmap)
get_map()	запросы к сервисам Google Maps, OpenStreetMap, или Stamen Maps для построения карт (ggmap)
gvisGeoChart(), gvisGeoMap(), gvisIntensityMap(), gvisMap()	гео-диаграммы и карты от Google (googleVis)
GetMap()	загрузка статичной карты с сервера Google (RgoogleMaps)
ColorMap()	цветовое кодирование и изображение уровней некоторой переменной на карте (RgoogleMaps)
PlotOnStaticMap()	совмещение графиков с географическими картами (RgoogleMaps)
TextOnStaticMap()	нанесение текстовых меток на карты (RgoogleMaps)

Пакеты

plotGoogleMaps	визуализация пространственных данных на картах Google Maps и сохранение результатов в виде HTML-виджетов
RgoogleMaps	работа с картами Google Maps в R
ggmap	визуализация пространственных данных с использованием сервисов Google Maps и OpenStreetMap
plotKML	визуализация пространственных и пространственно-временных объектов с использованием сервиса Google Earth
SGCS	кластеризация геоинформационных данных с использованием пространственных графов
spdep	инструменты для поиска пространственных зависимостей

ГРАФИЧЕСКИЕ ФУНКЦИИ (GRAPHICS FUNCTIONS)

plot()	функция общего назначения для визуализации данных
barplot(), pie(), hist()	столбиковые диаграммы, круговые диаграммы и гистограммы
boxplot()	диаграммы размахов
stripchart()	одномерные диаграммы рассеяния
dotchart()	точечная диаграмма Кливленда
qqnorm(), qqplot(), qqline()	графики квантиль-квантиль
coplot()	категоризованные графики
splom()	категоризованные матрицы диаграмм рассеяния (lattice)
pairs()	матрицы диаграмм рассеяния
cpairs()	улучшенные матрицы диаграмм рассеяния (gclus)
parcoord()	диаграммы параллельных координат (MASS)
cparcoord()	улучшенные диаграммы параллельных координат (gclus)
parallelplot()	диаграммы параллельных координат (lattice)
densityplot()	график ядерной функции плотности (lattice)
contour(), filled.contour()	контурные диаграммы
levelplot(), contourplot()	контурные диаграммы (lattice)
mosaicplot()	мозаичная диаграмма
assocplot()	диаграмма ассоциаций

smoothScatter()	диаграммы рассеяния с цветным представлением плотности вероятности; позволяют визуализировать большие массивы данных
sunflowerplot()	диаграмма рассеяния типа "подсолнух"
matplot()	изображение столбцов одной матрицы в зависимости от столбцов другой матрицы
fourfoldplot()	визуализация таблиц сопряженности размером 2×2×*k*
persp()	трехмерные диаграммы поверхностей
cloud(), wireframe()	трехмерные диаграммы рассеяния и диаграммы поверхностей (lattice)
interaction.plot()	график взаимодействий между двумя переменными
iplot(), ihist(), ibar(), ipcp()	интерактивные диаграммы рассеяния, гистограммы, столбиковые диаграммы и диаграммы параллельных координат (iplots)
pdf(), postscript(), win.metafile(), jpeg(), bmp(), png(), tiff()	сохранение графиков в файлах разных форматов
gvisAnnotatedTimeLine(), gvisAreaChart(), gvisBarChart(), gvisBubbleChart(), gvisCandlestickChart(), gvisColumnChart(), gvisComboChart(), gvisGauge(), gvisGeoChart(), gvisGeoMap(), gvisIntensityMap(), gvisLineChart(), gvisMap(), gvisMerge(), gvisMotionChart(), gvisOrgChart(), gvisPieChart(), gvisScatterChart(), gvisSteppedAreaChart(), gvisTable(), gvisTreeMap()	различные интерактивные диаграммы, созданные с использованием Google Visualisation API (googleVis)

Пакеты

ggplot2	реализация принципов "грамматики графических элементов"
ggvis	интерактивный вариант реализации принципов "грамматики графических элементов"
googleVis	интерфейс между R и Google Visualisation API для создания интерактивных диаграмм
d3Network, networkD3	R-интерфейс к JavaScript-библиотеке D3 для построения графов, деревьев, дендрограмм и диаграмм Санки
rCharts	создание интерактивных диаграмм с использованием различных JavaScript-библиотек
lattice	продвинутая высокоуровневая система для визуализации данных с упором на многомерные данные
vcd	визуализация категориальных данных
iplots	интерактивные диаграммы

ПРЕОБРАЗОВАНИЕ ДАННЫХ (DATA MANIPULATION)

transform()	преобразование таблицы данных
scale()	центрирование и нормирование столбцов матриц и таблиц
t()	транспонирование матриц
aperm()	транспонирование массивов
table(), tabulate(), xtabs()	формирование сводных таблиц
stack(), unstack()	объединение и декомпозиция векторов в соответствии с уровнями некоторого фактора

split(), unsplit()	разбиение данных на группы в соответствии с уровнями некоторого фактора(-ов) и обратная этому операция
reshape()	конвертирование таблицы данных в "широкий" или "длинный" формат
merge()	слияние двух таблиц данных (подобно join-операциям в базах данных)
aggregate()	вычисление сводных статистик для отдельных групп данных
by()	применение произвольной функции к таблице данных, разбитой на группы в соответствии с уровнями некоторого фактора(-ов)
melt(), cast()	разбиение таблицы данных на составляющие элементы с последующим формированием новой таблицы с измененной формой и/или содержимым (reshape)
sample()	формирование случайных выборок
complete.cases()	обнаружение записей в таблице данных, которые не содержат пропущенных значений
na.fail, na.omit, na.exclude, na.pass	обработка пропущенных значений

Пакеты

dplyr	высокоэффективный набор утилит со стандартизованным синтаксисом для работы с таблицами данных
reshape	гибкий инструмент для изменения формы таблиц данных и их агрегирования
reshape2	усовершенствованная версия пакета reshape
tidyr	результат дальнейшего усовершенствования пакета reshape2; позволяет легко изменять форму таблиц данных при помощи функций spread() и gather()
data.table	набор утилит для высокоэффективной работы с таблицами данных (индексирование, join-операции с сохранением порядка, присваивание значений, группирование и т.д.)
gdata	различные утилиты для манипуляций с данными
lubridate	набор функций для работы с датами и временем
stringr	набор функций для работы с символьными данными

ФУНКЦИИ ДОСТУПА К ДАННЫМ (DATA ACCESS FUNCTIONS)

save(), load()	сохранение и загрузка объектов типа RData
read.csv(), write.csv()	импорт и экспорт файлов формата .csv
read.table(), write.table(), scan(), write()	импорт и экспорт данных
read.xlsx(), write.xlsx()	импорт и экспорт Excel-файлов (xlsx)
read.fwf()	ипорт данных, которые хранятся в виде файлов с фиксированной шириной по лей
write.matrix()	экспорт матрицы или таблицы данных (MASS)
readLines(), writeLines()	запись и чтение текстовых строк из файла
sqlQuery()	выполенение SQL-запросов к базе данных ODBC (RODBC)

`sqlFetch()`	чтение таблицы из базы данных ODBC (RODBC)
`sqlSave()`, `sqlUpdate()`	сохранение и обновление таблиц в базе данных ODBC (RODBC)
`sqlColumns()`	выяснение структуры таблиц в базе данных (RODBC)
`sqlTables()`	получение списка таблиц, имеющихся в базе данных (RODBC)
`odbcConnect()`, `odbcClose()`, `odbcCloseAll()`	открытие и закрытие соединения с базой данных ODBC (RODBC)
`dbSendQuery`	выполенение SQL-запроса к базе данных (DBI)
`dbConnect()`, `dbDisconnect()`	открытие и закрытие соединения с системой управления базами данных (DBI)

Пакеты

RODBC	доступ к базам данных ODBC
foreign	чтение и запись данных в сторонних форматах, таких как Minitab, S, SAS, SPSS, Stata, Systat и др.
sqldf	выполнение SQL-подобных SELECT-запросов к таблицам R
DBI	DBI-интерфейс между R и реляционными DBMS
RMySQL	драйвер для соединения с базами данных MySQL
RJDBC	доступ к базам данным через интерфейс JDBC
RSQLite	драйвер для соединения с базами данных RSQLite
ROracle	DBI-драйвер для соединения с базами данных Oracle
RpgSQL	DBI/RJDBC-интерфейс для работы с базами данных PostgreSQL
RODM	интерфейс для работы с Oracle Data Mining
xlsx	чтение и запись файлов в форматах Excel 97/2000/XP/2003/2007
xlsReadWrite	чтение и запись Excel-файлов
WriteXLS	создание файлов формата Excel 2003 (xls) из таблиц данных R
SPARQL	SPARQL-драйвер для выполнения запросов SELECT и UPDATE

ФУНКЦИИ ДОСТУПА К ДАННЫМ ЧЕРЕЗ ВЕБ-ИНТЕРФЕЙС (WEB DATA ACCESS FUNCTIONS)

`download.file()`	загрузка файлов из Интернета
`xmlParse()`, `htmlParse()`	разбор файлов XML и HTML (XML)
`userTimeline()`, `homeTimeline()`, `mentions()`, `retweetsOfMe()`	извлечение разнотипных данных из сети Twitter (twitteR)
`searchTwitter()`	поиск в сети Twitter по поисковой фразе (twitteR)
`getUser()`, `lookupUsers()`	получение информации о пользователе сети Twitter (twitteR)
`getFollowers()`, `getFollowerIDs()`, `getFriends()`, `getFriendIDs()`	получение списка фоловеров и друзей (или их идентификаторов) того или иного пользователя сети Twitter (twitteR)
`twListToDF()`	конвертирование списков twitteR в стандартные таблицы данных (twitteR)

Пакеты

twitteR	набор утилит для работы с Twitter API
RCurl	R-клиент для выполнения запросов по стандартным сетевымпротоколам (HTTP/FTP/...)

XML	чтение и создание документов в форматах XML и HTML
httr	набор утилит для работы с URL и HTTP (построен на основе RCur, но проще в использовании)

ФУНКЦИИ ДЛЯ ВЗАИМОДЕЙСТВИЯ С ИНСТРУМЕНТАМ ОБРАБОТКИ "БОЛЬШИХ ДАННЫХ" MAPREDUCE, HADOOP И SPARK

`mapreduce()`	спецификация и выполнение задач MapReduce (rmr2)
`keyval()`	создание объектов типа "ключ – значение" (rmr2)
`from.dfs(), to.dfs()`	чтение и запись объектов R при работе со сторонними файловыми системами (rmr2)
`hb.get(), hb.scan(), hb.get.data.frame()`	чтение таблиц HBase (rhbase)
`hb.insert(), hb.insert.data.frame()`	запись таблиц HBase (rhbase)
`hb.delete()`	удаление записей из таблиц HBase (rhbase)

Пакеты

rmr2	анализ данных в среде R в стиле MapReduce на Hadoop-кластере
rhdfs	соединение с Hadoop Distributed File System (HDFS)
rhbase	соединение с NoSQL базой данных HBase
Rhipe	инструменты для работы с Hadoop из среды R
SparkR	тонкий R-клиент для работы с Apache Spark
RHive	распределенные вычисления на основе запросов к HIVE
Segue	выполнение параллельных вычислений с использованием облачного сервиса Amazon Elastic Map Reduce (EMR)
HadoopStreaming	утилиты для использования R-скриптов при обработке потоковых данных на Hadoop-кластере
hive	распределенные вычисление, основанные на парадигме MapReduce
rHadoopClient	R-клиент для работы Hadoop

БОЛЬШИЕ МАССИВЫ ДАННЫХ (LARGE DATA)

`as.ffdf()`	преобразование таблицы данных в формат ffdf (ff)
`read.table.ffdf(), read.csv.ffdf()`	чтение данных из текстового файла и сохранение в виде ffdf-объекта (ff)
`write.table.ffdf(), write.csv.ffdf()`	сохранение ffdf-объектов в виде текстовых файлов (ff)
`ffdfappend()`	добавление обычной таблицы данных или таблицы ffdf к существующей таблице ffdf (ff)
`big.matrix()`	создание стандартной "большой матрицы" (объект типа `big.matrix`), размер которой ограничен доступным объемом RAM (bigmemory)
`read.big.matrix()`	создание "большой матрицы" путем чтения из ASCII-файла (bigmemory)
`write.big.matrix()`	запись "большой матрицы" в файл (bigmemory)
`filebacked.big.matrix()`	создание "большой матрицы" в виде файла хранящегося на диске (размер такой матрицы может превышать доступный объем памяти) (bigmemory)
`mwhich()`	усовершенствованные "which"-подобные команды для работы с большими матрицами (bigmemory)

Пакеты

ff	хранение больших массивов данных на диске с эффективным использованием памяти, а также набор функций для быстрого доступа к таким данным
ffbase	стандартные статистические функции для пакета ff
filehash	простая база данных типа "ключ – значение" для работы с большими массивами данных
g.data	создание и поддержка пакетов для работы с данными типа "delayed data"
BufferedMatrix	объекты для хранения матриц с данными во временных файлах
biglm	регрессионный анализ для данных, которые не помещаются в памяти компьютера
bigmemory	набор утилит для работы с матрицами данных очень большого объема
biganalytics	расширение пакета bigmemory, содержащее дополнительный набор аналитических функций
bigtabulate	table-, tapply-, и split-подобные функции для работы с объектами классов matrix и big.matrix

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Функции для организации параллельных вычислений

sfInit(), sfStop()	запуск и завершение работы вычислительного кластера (snowfall)
sfLapply(), sfSapply(), sfApply()	параллельные версии функций lapply(), sapply(), apply() (snowfall)
foreach(...) %dopar%	параллельное выполнение циклов (foreach)
registerDoSEQ(), registerDoSNOW(), registerDoMC()	регистрация последовательного, SNOW и многопоточного бэк-энда для выполнения параллельных вычислений с помощью пакета foreach (foreach, doSNOW, doMC)

Пакеты

snowfall	"обертка" на основе функционала пакета snow, предназначенная для более эффективной разработки программ для параллельных вычислений в среде R
snow	организация параллельных вычислений в R
multicore	параллельное исполнение R-кода на машинах с несколькими процессорами
snowFT	расширение пакета snow для разработки робастных и воспроизводимых приложений, и для удобной организации параллельных вычислений
Rmpi	интерфейс для работы с MPI (Message-Passing Interface)
rpvm	R-интерфейс для работы с PVM (Parallel Virtual Machine)
nws	набор утилит для координации параллельных вычислений
foreach	конструктов foreach-циклов для R
doMC	адаптор к пакету multicore для выполнения параллельных foreach-вычислений
doSNOW	адаптор к пакету snow для выполнения параллельных foreach-вычислений
doMPI	адаптор к пакету Rmpi для выполнения параллельных foreach-

	вычислений
doParallel	адаптор к пакету multicore для выполнения параллельных foreach-вычислений
doRNG	генератор случайных чисел, позволяющий выполнять воспроизводимые параллельные вычисления на основе foreach-циклов
GridR	исполнение R-кода на удаленных машинах и кластерах
fork	набор функций для одновременной работы с несколькими процессами R

ИНТЕРФЕЙС К WEKA И ДРУГИМ ЯЗЫКАМ ПРОГРАММИРОВАНИЯ (INTERFACE TO WEKA AND OTHER PROGRAMMING LANGUAGES FUNCTIONS)

Пакет RWeka – это R-интерфейс к Weka, который позволяет работать с функциями Weka из среды R:

- Ассоциативные правила: `Apriori()`, `Tertius()`
- Регрессия и классификация: `LinearRegression()`, `Logistic()`, `SMO()`
- "Ленивые" классификаторы: `IBk()`, `LBR()`
- Мета-классификаторы: `AdaBoostM1()`, `Bagging()`, `LogitBoost()`, `MultiBoostAB()`, `Stacking()`, `CostSensitiveClassifier()`
- Классификаторы на основе правил: `JRip()`, `M5Rules()`, `OneR()`, `PART()`
- Деревья классификации и регрессии: `J48()`, `LMT()`, `M5P()`, `DecisionStump()`
- Кластеризация: `Cobweb()`, `FarthestFirst()`, `SimpleKMeans()`, `XMeans()`, `DBScan()`
- Фильтры: `Normalize()`, `Discretize()`
- Стемминг слов: `IteratedLovinsStemmer()`, `LovinsStemmer()`
- Токенайзеры: `AlphabeticTokenizer()`, `NGramTokenizer()`, `WordTokenizer()`

Другие языки

`.jcall()`	вызов метода Java (rJava)
`.jnew()`	создание нового объекта Java (rJava)
`.jinit()`	инициализация Java Virtual Machine (JVM) (rJava)
`.jaddClassPath()`	добавляет JAR-файлы к пути класса (rJava)

Пакеты

rJava	низкоуровневый интерфейс между R и Java
rPython	вызов функций Python из R

ФУНКЦИИ ДЛЯ ГЕНЕРАЦИИ ДОКУМЕНТОВ И ОТЧЕТОВ (GENERATING DOCUMENTS AND REPORTS FUNCTIONS)

`Sweave()`	сочетание текста с R или S-кодом для автоматического формирования отчетов
`xtable()`	экспорт таблиц в форматах LaTeX или HTML (xtable)

Пакеты

knitr	пакет общего назначения для формирования динамических отчетов в среде R
xtable	экспорт таблиц в форматах LaTeX или HTML

R2HTML	создание HTML-отчетов
R2PPT	формирование презентаций Microsoft PowerPoint
RPMG	графический интерфейс пользователя (GUI) для интерактивных R-сессий
Red-R	графический интерфейс пользователя с открытым кодом для визуального программирования на языке R
rattle	графический интерфейс пользователя для Data Mining на языке R
latticist	графический интерфейс пользователя для выполнения визуального разведочного анализа данных

Создание графических интерфейсов пользователя и веб-приложений

shiny	фреймворк для разработки веб-приложений в R
svDialogs	создание диалоговых окон
gWidgets	платформо-независимый набор инструментов для разработки графических интерфейсов пользователя
R AnalyticFlow	программа для выполнения анализа данных путем рисования блок-схем, определяющих последовательность аналитических операций

Редакторы для разработки кода на R

RStudio	бесплатная интегрированная среда разработки (IDE) для R
Tinn-R	бесплатный графический интерфейс пользователя для R
Rpad	веб-интерфейс для R в виде рабочих книг

ССЫЛКИ НА ОБУЧАЮЩИЕ РЕСУРСЫ В ИНТЕРНЕТЕ

Веб-сайт RDataMining:	http://www.rdatamining.com http://www2.rdatamining.com
RDataMining группа в LinkedIn (20,000+ подписчиков):	http://group.rdatamining.com или https://www.linkedin.com/groups/4066593
RDataMining в сети Twitter (2,500+ фоловеров):	@RDataMining
Проект RDataMining на сайте R-Forge:	http://www.rdatamining.com/package http://package.rdatamining.com