Анализ данных с R (II).

© А. Б. Шипунов*, А. И. Коробейников‡, Е. М. Балдин**

*dactylorhiza@gmail.com
‡asl@math.spbu.ru
**E.M.Baldin@inp.nsk.su

Эмблема **R** взята с официального сайта проекта `http://developer.r-project.org/Logo/`

Оглавление

Глава 5

Работа с двумя переменными

Повествование об **R** возобновляется на новом уровне. С января (LXF100/101) по апрель (LXF104) в журнале была опубликована серия из четырёх статей представляющих этот замечательный язык обработки данных. То был предварительный обзор возможностей, а сейчас начинается серьёзная работа.

5.1. Проверка гипотез однородности

Две разные выборки называются однородными, если они одинаково распределены. Проверка «гипотез однородности» в математической статистике занимает особое место. Этот факт связан с тем, что для практических приложений, как правило, характерны задачи сравнения двух (и более) групп наблюдений. Сравнивать выборки можно совершенно разными способами: исследователя может интересовать различие в средних, медианах, дисперсиях при разнообразных предположениях относительно самих наблюдений.

5.1.1. Параметрические критерии проверки однородности выборок

Величины, имеющие нормальное распределение, в реальных экспериментах возникают совершенно естественным образом, При измерении любой характеристики всегда имеется ошибка измерения. Если предполагать, что ошибка прибора имеет нормальное распределение, то среднее отвечает за систематическую ошибку, а дисперсия — за величину случайной ошибки. Поэтому, критерии представленные в данном разделе предполагают, что выборка имеет нормальное распределение. Если это заранее не известно, то этот факт нужно проверять отдельно

(см. раздел 5.2). В противном случае все выводы, полученные на основе этих критериев, будут ошибочными.

Двухвыборочный критерий Стьюдента равенства средних

Двухвыборочный t-критерий используется для проверки гипотезы о равенстве средних в двух независимых выборках, имеющих нормальное распределение. В своей классической постановке критерий проводится в предположении равенства дисперсий в двух выборках. В **R** реализована модификация критерия, позволяющая избавиться от этого предположения.

Воспользуемся классическим набором данных, который использовался в оригинальной работе Стьюдента (псевдоним Уильяма Сили Госсета). В упомянутой работе производилось сравнение влияния двух различных снотворных на увеличение продолжительности сна (рис. 5.1). В **R** этот массив данных доступен под названием `sleep` в пакете **stats**. В столбце `extra` содержится среднее приращение продолжительности сна после начала приёма лекарства (по отношению к контрольной группе), а в столбце `group` — код лекарства (первое или второе).

```
> plot(extra ~ group, data = sleep)
```

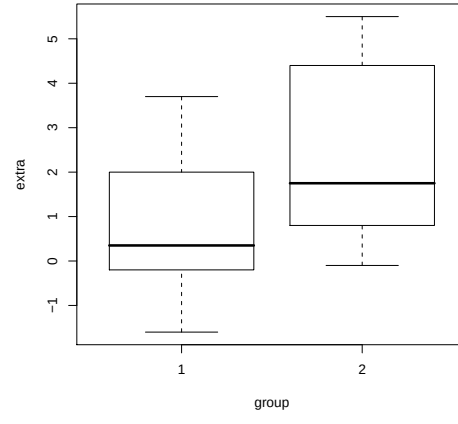

Рис. 5.1. Среднее приращение продолжительности сна после начала приёма разных лекарств в двух группах по отношению к контрольной.

Влияние лекарства на каждого человека индивидуально, но среднее увеличение продолжительности сна можно считать вполне логичным показателем «силы» лекарства. Основываясь на этом предположении, попробуем проверить при

помощи t-критерия, значимо ли различие в средних для этих двух выборок (соответствующих двум разным лекарствам):

```
> with(sleep, t.test(extra[group == 1],
+             extra[group == 2], var.equal = FALSE))

        Welch Two Sample t-test

data:  extra[group == 1] and extra[group == 2]
t = -1.8608, df = 17.776, p-value = 0.0794
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.3654832  0.2054832
sample estimates:
mean of x mean of y
     0.75      2.33
```

Параметр `var.equal` позволяет выбрать желаемый вариант критерия: оригинальный t-критерий Стьюдента в предположении одинаковых дисперсий (`TRUE`) или же t-критерий в модификации Уэлча (Welch), свободный от этого предположения (`FALSE`).

Хотя формально гипотеза о равенстве средних не отвергается на стандартных уровнях значимости, мы видим, что возвращаемое p-значение (0.0794) достаточно маленькое, поэтому к данном результату стоит относится критично. Это означает, что возможно, стоит попробовать другие методы для проверки гипотезы, увеличить количество наблюдений, ещё раз убедиться в нормальности распределений и т. д.

Можно ли использовать t-критерий, если необходимо сравнить среднее в *зависимых* выборках (например, при сравнении какого-либо жизненного показателя у пациента до и после лечения)? Ответ на этот вопрос: «Да, можно, но не обычный, а модифицированный специальным образом под такую процедуру». В литературе такая модификация называется *парным t-критерием*. Ничего специального для использования парного t-критерия в **R** делать не надо. Достаточно выставить опцию `paired` в `TRUE`:

```
> with(sleep, t.test(extra[group == 1],
+             extra[group == 2], paired = TRUE))

        Paired t-test

data:  extra[group == 1] and extra[group == 2]
t = -4.0621, df = 9, p-value = 0.002833
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
```

```
 -2.4598858 -0.7001142
sample estimates:
mean of the differences
                  -1.58
```

Здесь видно, что парный t-критерий отвергает гипотезу о равенстве средних с достаточно большой надёжностью. Следует отметить, что использованные в этом примере выборки не были «парными», поэтому, строго говоря, применять парный t-критерий нельзя, и все выводы носят сугубо иллюстративный характер.

Двухвыборочный критерий Фишера равенства дисперсий

Естественной характеристикой «размаха» распределения при нормальной модели является дисперсия. Предположим, что потребовалось проверить гипотезу об отсутствии различий в дисперсиях двух выборок. При этом не хочется делать абсолютно никаких допущений относительно значений средних в этих выборках. Такая задача может возникнуть, например, при сравнении точности двух приборов. Напомним, что при наличии ошибок измерений мерой систематической ошибки прибора является среднее, а случайной — дисперсия. Систематическую ошибку иногда можно уменьшить за счёт точной калибровки прибора. Случайную же ошибку убрать почти никогда не представляется возможным. В связи с этим задача проверки равенства дисперсий (например, при сравнении эталонного прибора и проверяемого) становится достаточно актуальной.

Решением такой задачи служит F-критерий Фишера (Fisher). В **R** он реализован в функции var.test():

```
> x <- rnorm(50, mean = 0, sd = 2);
> y <- rnorm(30, mean = 1, sd = 1);
> var.test(x, y)

        F test to compare two variances

data:  x and y
F = 3.8414, num df = 49, denom df = 29, p-value = 0.0002308
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 1.930003 7.227256
sample estimates:
ratio of variances
          3.841391
```

В этом примере участвуют две выборки с разным количеством наблюдений (50 и 30 для x и y, соответственно), разными средними (0 и 1) и дисперсиями ($2^2 = 4$ и $1^2 = 1$). Гипотеза о равенстве дисперсий безусловно отвергается (p-значение

мало). Кроме самого значения тестовой статистики, p-значения и величин степеней свободы функция выводит оценку отношения дисперсий и доверительный интервал от него. Значение оценки 3.55 не очень сильно отличается от истинного значения отношения дисперсий 4.

На самом деле, критерий проверяет несколько более общую гипотезу. Идёт проверка того, что отношение дисперсий двух выборок имеет какое-то наперёд заданное значение. Проверка гипотезы о равенстве дисперсий является частным случаем такой гипотезы.

Предполагаемое значение отношения дисперсий можно задать с помощью опции `ratio`:

```
> x <- rnorm(50, mean = 0, sd = 2);
> y <- rnorm(30, mean = 1, sd = 1);
> var.test(x, y, ratio = 4)

        F test to compare two variances

data:  x and y
F = 1.1097, num df = 49, denom df = 29, p-value = 0.7778
alternative hypothesis: true ratio of variances is not equal to 4
95 percent confidence interval:
 2.230136 8.351157
sample estimates:
ratio of variances
           4.43876
```

Здесь видно, что при задании истинного значения отношения дисперсий гипотеза не отвергается.

5.1.2. Непараметрические критерии проверки однородности выборок

Критерии, приведённые в предыдущем разделе работают только в предположении нормальности распределения данных. Что делать, если заранее известно, что выборки имеют другое распределение, или по каким-либо причинам проверить нормальность не получается? В таких случаях используются так называемые *непараметрические* критерии, т. е. критерии свободные от предположения какой-либо параметрической модели данных. Естественно, ввиду того, что эти критерии оперируют гораздо меньшим «объёмом информации», то они не смогут заметить те тонкие различия, которые были бы обнаружены при использовании параметрических критериев.

Критерий Вилкоксона

Критерий Вилкоксона (Wilcoxon), двухвыборочный вариант которого ещё известен под именем критерия Манна-Уитни, является непараметрическим аналогом t-критерия.

Стандартный набор данных `airquality` содержит информацию о величине озона в воздухе города Нью-Йорка с мая по сентябрь 1973 года. Проверим, например, гипотезу о том, что распределения уровня озона в мае и в августе было одинаковым:

```
> wilcox.test(Ozone ~ Month, data = airquality,
+                       subset = Month %in% c(5, 8))

        Wilcoxon rank sum test with continuity correction

data:  Ozone by Month
W = 127.5, p-value = 0.0001208
alternative hypothesis: true location shift is not equal to 0
```

Критерий отвергает гипотезу о согласии распределений уровня озона в мае и в августе с достаточно большой надёжностью. В принципе это достаточно легко вытекает из «общих соображений», так как уровень озона в воздухе сильно зависит от солнечной активности, температуры и ветра. И если с солнечной активностью всё в порядке:

```
> wilcox.test(Solar.R ~ Month, data = airquality,
+                       subset = Month %in% c(5, 8))

        Wilcoxon rank sum test with continuity correction

data:  Solar.R by Month
W = 422.5, p-value = 0.4588
alternative hypothesis: true location shift is not equal to 0
```

то распределения ветра и температуры, напротив, сильно различаются:

```
> wilcox.test(Temp ~ Month, data = airquality,
+                       subset = Month %in% c(5, 8))

        Wilcoxon rank sum test with continuity correction

data:  Temp by Month
W = 27, p-value = 1.747e-10
alternative hypothesis: true location shift is not equal to 0
```

```
> wilcox.test(Wind ~ Month, data = airquality,
+                         subset = Month %in% c(5, 8))

        Wilcoxon rank sum test with continuity correction

data:  Wind by Month
W = 687.5, p-value = 0.003574
alternative hypothesis: true location shift is not equal to 0
```

Эти различия легко видны, скажем, на изображении ящиков с усами (рис. 5.2):

```
> boxplot(Temp ~ Month, data = airquality,
+                       subset = Month %in% c(5, 8))
```

но в сложных случаях только использование критериев позволяет получать объективные результаты.

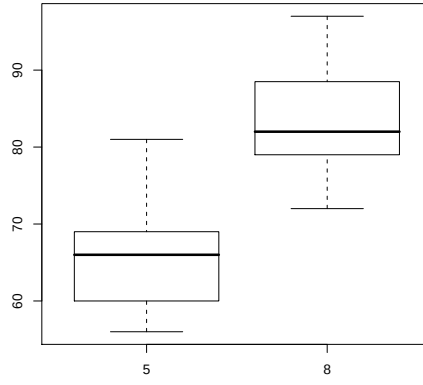

Рис. 5.2. Распределение температуры в Нью-Йорке в мае и в августе 1973 года.

По умолчанию критерий проверяет гипотезу о том, что распределения двух выборок отличаются лишь постоянным и известным сдвигом (который, в свою очередь, по умолчанию равен нулю). Задать его можно при помощи параметра `mu`. Например:

```
> x <- rnorm(50, mean = 0);
> y <- rnorm(50, mean = 2);
> wilcox.test(x, y);
```

```
        Wilcoxon rank sum test with continuity correction

data:  x and y
W = 230, p-value = 2.091e-12
alternative hypothesis: true location shift is not equal to 0

> wilcox.test(x, y, mu = -2);

        Wilcoxon rank sum test with continuity correction

data:  x and y
W = 1335, p-value = 0.5602
alternative hypothesis: true location shift is not equal to -2
```

Критерий Манна-Уитни так же, как и t-критерий, тоже бывает парным. Для использования парной модификации необходимо выставить опцию `paired` в значение `TRUE`. Задание опции `conf.int` позволяет получить оценку различия в сдвиге между двумя выборками[1] и доверительный интервал для него:

```
> x <- rnorm(50, mean = 0);
> y <- rnorm(50, mean = 2);
> wilcox.test(x, y, conf.int = TRUE);

        Wilcoxon rank sum test with continuity correction

data:  x and y
W = 227, p-value = 1.803e-12
alternative hypothesis: true location shift is not equal to 0
95 percent confidence interval:
 -2.418617 -1.500210
sample estimates:
difference in location
             -1.941988

> wilcox.test(x, y, mu = -2);

        Wilcoxon rank sum test with continuity correction

data:  x and y
W = 1292, p-value = 0.7748
alternative hypothesis: true location shift is not equal to -2
```

[1] Естественно, при условии, что кроме сдвига распределения двух выборок ничем не отличаются

Гипотеза о полном равенстве распределений была отвергнута, а гипотеза о том, что одно распределение отличает от другого просто сдвигом не отвергнута, что и требовалось показать.

Непараметрические критерии сравнения масштаба

Дисперсия является естественным параметром масштаба для выборки из нормальной совокупности. Этот факт позволяет заменить гипотезу о совпадении масштабов на гипотезу о совпадении дисперсий в случае нормального распределения. При отказе от нормальной модели дисперсия уже не является характеристикой масштаба и поэтому надо честно проверять гипотезу именно о совпадении масштабов распределений двух выборок.

Формально предполагается, что одна из выборок имеет распределение с плотностью $f(x-a)$, другая — $sf\left(s\left(x-a\right)\right)$. Здесь функция плотности f и параметр сдвига a считаются неизвестными. Мы заинтересованы в проверке совпадения масштабов у двух выборок, то есть проверке того, что $s=1$.

В стандартном пакете **stats** реализованы два классических непараметрических критерия, позволяющих проверить равенство масштабов: критерий Ансари-Брэдли (Ansari-Bradley) и критерий Муда (Mood). Начнём с первого.

```
> ansari.test(runif(50), rucunif(50, max = 2))

        Ansari-Bradley test

data:  runif(50) and runif(50, max = 2)
AB = 1404, p-value = 0.07526
alternative hypothesis: true ratio of scales is not equal to 1
```

Распределение выборок в данном случае отличается только масштабом. Критерий отвергает гипотезу о совпадении дисперсий на стандартных уровнях значимости, как это и должно быть. Всё аналогично для критерия Муда:

```
> mood.test(runif(50), runif(50, max = 2))

        Mood two-sample test of scale

data:  runif(50) and runif(50, max = 2)
Z = -2.5685, p-value = 0.01021
alternative hypothesis: two.sided
```

5.2. Проверка гипотез нормальности распределения

Большая часть инструментов статистического вывода работает в предположении о том, что выборка получена из нормальной совокупности. За примерами далеко ходить не надо: t-критерий, критерий Фишера, построение доверительных интервалов для линейной регрессии и проверка гипотезы о линейной независимости двух выборок.

Как уже отмечалось ранее, нормальное распределение естественным образом возникает практически везде, где речь идёт об измерении с ошибками. Более того, в силу центральной предельной теоремы, распределение многих выборочных величин (например, выборочного среднего) при достаточно больших объёмах выборки хорошо аппроксимируется нормальным распределением вне зависимости от того, какое распределение было у выборки исходно.

В связи с этим становится понятным, почему проверке распределения на нормальность стоит уделить особое внимание. В дальнейшем речь пойдёт о так называемых *критериях согласия* (goodness-of-fit tests). Проверяться будет не просто факт согласия с нормальным распределением с определёнными фиксированными значениями параметров, а несколько более общий факт принадлежности распределения к семейству нормальных распределений со всевозможными значениями параметров.

Основные классические критерии проверки на нормальность собраны в пакете **nortest**. Пакет можно установить с CRAN при помощи вызова функции install.packages():

```
> install.packages(pkgs=c("nortest"))
```

При этом Tcl/Tk-форма предлагает выбрать зеркало CRAN откуда скачать пакет. Подключить установленный пакет можно при помощи функции library():

```
> library(nortest)
```

Может возникнуть вопрос: «А зачем столько много разных критериев для проверки одного факта? Нельзя ли выбрать наилучший и всегда его использовать?». Ответ на этот вопрос не утешителен: «В общем случае, к сожалению, нельзя». Попробуем это объяснить.

Напомним, что ошибкой первого рода статистического критерия называется факт принятия альтернативной гипотезы, в ситуации когда верна гипотеза по умолчанию. Например, пусть статистический критерий используется для разграничения доступа к какому-нибудь ресурсу. Тогда отказ в доступе для авторизованного пользователя и будет ошибкой первого рода для такого критерия. Ясно, что возможна «симметричная» ошибочная ситуация, заключающаяся в предоставление доступа к ресурсу не авторизованному пользователю. Такая ошибка называется *ошибкой второго рода*: принятие критерием гипотезы по умолчанию в ситуации, когда она не верна (то есть имеет место альтернативная гипотеза).

Как правило, чувствительность выбранного критерия к ошибкам первого рода мы настраиваем самостоятельно (как раз выбором тех самых «стандартных уровней значимости», с которым и сравниваем p-значение, выданное критерием). С ошибкой второго рода всё гораздо хуже: её вероятность сильно зависит от выбранной альтернативной гипотезы и является неотъемлемой характеристикой самого критерия. В редких случаях удаётся построить критерий, который является наилучшим (это так называемые *равномерно наиболее мощные критерии*), и, к сожалению, это невозможно для нашей задачи. В нашем случае нулевая гипотеза формулируется просто: *выборка имеет нормальное распределение с некоторыми неизвестными параметрами*, а альтернативная гипотеза — это полное её отрицание. Альтернативная гипотеза гораздо «богаче» нулевой, туда входят все распределения, отличные от нормального. Для того и понадобилась целая батарея критериев: какие-то работают лучше против одного семейства альтернатив, другие — против другого. Использование всего набора позволяет быть хоть как-то уверенным в том, что распределение, не являющееся нормальным будет «распознано» хотя бы одним из критериев.

Критерий Лиллифорса

Критерий Лиллифорса (Lilliefors) является вариантом известного классического критерия Колмогорова-Смирнова, специально модифицированного для проверки нормальности. Эта модификация существенна. Для проверки гипотезы нормальности *нельзя* использовать классический непараметрический критерий Колмогорова-Смирнова, реализованный в функции `ks.test()`. Критерий Лиллифорса реализован в функции `lillie.test()`:

```
> lillie.test(rnorm(100, mean = 6, sd = 4));

        Lilliefors (Kolmogorov-Smirnov) normality test

data:  rnorm(100, mean = 6, sd = 4)
D = 0.0463, p-value = 0.8621

> lillie.test(runif(100, min = 2, max = 4));

        Lilliefors (Kolmogorov-Smirnov) normality test

data:  runif(100, min = 2, max = 4)
D = 0.0732, p-value = 0.2089
```

Критерии Крамера-фон Мизеса и Андерсона-Дарлинга

Первый критерий известен в русскоязычной литературе под именем критерия ω^2 или критерия Смирнова. Эти критерии менее известны, но обычно работают

гораздо лучше, нежели критерий Лиллифорса. Они реализованы в функциях cvm.test() и ad.test() соответственно:

```
> cvm.test(rnorm(50, mean = 6, sd = 4));

        Cramer-von Mises normality test

data:  rnorm(50, mean = 6, sd = 4)
W = 0.0321, p-value = 0.8123

> ad.test(runif(50, min = 2, max = 4));

        Anderson-Darling normality test

data:  runif(50, min = 2, max = 4)
A = 1.5753, p-value = 0.0004118
```

Критерий Шапиро-Франсиа

Этот критерий работает достаточно хорошо в большинстве не очень «сложных» случаев. Получить p-значение можно посредством функции sf.test():

```
> sf.test(rexp(50, rate = 2));

        Shapiro-Francia normality test

data:  rexp(50, rate = 2)
W = 0.7803, p-value = 2.033e-06
```

Критерий хи-квадрат Пирсона

В отличие от задач проверки пропорций, критерий хи-квадрат обычно очень плохо работает в задачах проверки распределения на нормальность. Вероятность ошибки второго рода очень велика для достаточно широкого класса альтернативных распределений. В связи с этим, использовать его не рекомендуется.

Тем не менее реализация его предоставлена функцией pearson.test(). У этой функции есть булевская опция `adjusted`, которая позволяет внести поправки в p-значение из-за наличия двух неизвестных параметров. Рекомендуемая последовательность действий такая: получить два p-значения, одно, соответствующее `adjusted=TRUE`, второе — `adjusted=FALSE`. Истинное p-значение обычно находится между. Кроме того, полезно поварьировать объем выборки и посмотреть, насколько сильно меняется p-значение. Если влияние объёма выборки сильное, то от использования критерия стоит отказаться во избежание ошибок.

```
> pearson.test(rnorm(50, mean = 6, sd = 4));

        Pearson chi-square normality test

data:  rnorm(50, mean = 6, sd = 4)
P = 5.2, p-value = 0.6356

> pearson.test(runif(50, min = -1, max = 1));

        Pearson chi-square normality test

data:  runif(50, min = -1, max = 1)
P = 7.6, p-value = 0.3692
```

► Теперь мы можем не только отличать нормальное распределения от «не нормального», но и сравнивать разные распределения. Это одна из самых первых ступенек на пути понимания сути данных, которые волей не волей приходится собирать для познания природы абсолютно любых явлений.

5.3. Взаимосвязь случайных величин

Одной из основных задач статистики является изучение зависимостей между данными. Под словом «зависимость» следует понимать зависимость в самом широком её смысле. Не однозначную функциональную зависимость, когда имея только один набор данным становится возможным полностью определить другой набор, а гораздо более общий случай, когда по одному набору данных можно получить хоть какую-нибудь информацию о втором.

5.3.1. Корреляция

Начнём всё же с функциональной зависимости, как наиболее просто формализуемой. Классическим инструментом для измерения *линейной зависимости* между двумя наборами данных является коэффициент корреляции. Коэффициент корреляции — это числовая величина, находящаяся в интервале от -1 до $+1$. Чем она больше по модулю (т. е. ближе к $+1$ или -1), тем выше линейная связь между наборами данных. Знак коэффициента корреляции показывает, в одном ли направлении изменяются наборы данных. Если один из наборов возрастает, а второй убывает, то коэффициент корреляции отрицателен, а если оба набора одновременно возрастают или убывают, то коэффициент корреляции положителен. Значение коэффициента корреляции по модулю равное 1 соответствует точной линейной зависимости между двумя наборами данных. Линейная зависимость является самой любимой у экспериментаторов всех мастей.

► Обратите внимание, что значение коэффициента корреляции близкое к нулю *не означает* независимости наборов данных. Коэффициент корреляции — это мера линейной зависимости, поэтому этот факт означает лишь отсутствие линейной зависимости, но не исключает любой другой. Отсутствие линейной зависимости равносильно независимости только для нормально распределённых выборок, факт нормальности, естественно, надо проверять отдельно.

Для вычисления коэффициента корреляции в R реализована функция cor:

```
> cor(5:15, 7:17)
[1] 1
> cor(5:15, c(7:16, 23))
[1] 0.9375093
```

В самом её простейшем случае ей передаются два аргумента (векторы одинаковой длины). Кроме того, возможен вызов функции с одним аргументом, в качестве которого может выступать матрица или набор данных (data frame). В этом случае функция cor вычисляет так называемую *корреляционную матрицу*, составленную из коэффициентов корреляций между столбцами матрицы или набора данных, взятых попарно:

```
>  cor(longley)

             GNP.deflator       GNP Unemployed Armed.Forces
GNP.deflator    1.0000000 0.9915892  0.6206334    0.4647442
GNP             0.9915892 1.0000000  0.6042609    0.4464368
Unemployed      0.6206334 0.6042609  1.0000000   -0.1774206
Armed.Forces    0.4647442 0.4464368 -0.1774206    1.0000000
Population      0.9791634 0.9910901  0.6865515    0.3644163
Year            0.9911492 0.9952735  0.6682566    0.4172451
Employed        0.9708985 0.9835516  0.5024981    0.4573074
               Population      Year  Employed
GNP.deflator    0.9791634 0.9911492 0.9708985
GNP             0.9910901 0.9952735 0.9835516
Unemployed      0.6865515 0.6682566 0.5024981
Armed.Forces    0.3644163 0.4172451 0.4573074
Population      1.0000000 0.9939528 0.9603906
Year            0.9939528 1.0000000 0.9713295
Employed        0.9603906 0.9713295 1.0000000
```

Если все данные присутствуют, то всё просто, но что делать, когда есть пропущенные наблюдения? Есть несколько способов, как вычислить корреляционную матрицу в этом случае. Для этого в команде cor есть опция `use`. По-умолчанию она равна `all.obs`, что при наличии хотя бы одного пропущенного наблюдения приводит к ошибки исполнения cor. Если опцию `use` приравнять значению `complete.obs`, то из данных до вычисления корреляционной матрицы удаляются

все наблюдения, в которых есть хотя бы один пропуск. Может оказаться так, что пропуски раскиданы по исходному набору данных достаточно хаотично и их много, так что после построчного удаления от матрицы фактически ничего не остаётся. В таком случае поможет попарное удаление пропусков, то есть удаляются строчки с пропусками не из всей матрицы сразу, а только лишь из двух столбцов непосредственно перед вычислением коэффициента корреляции. Для этого опцию `use` следует приравнять значению `pairwise.complete.obs`.

► В последнем случае следует принимать во внимание то, что коэффициенты корреляции вычисляются по *разному* количеству наблюдений и сравнивать их друг с другом может быть опасно.

```
> cor(swiss)
                 Fertility Agriculture Examination
Fertility        1.0000000  0.35307918  -0.6458827
Agriculture      0.3530792  1.00000000  -0.6865422
Examination     -0.6458827 -0.68654221   1.0000000
Education       -0.6637889 -0.63952252   0.6984153
Catholic         0.4636847  0.40109505  -0.5727418
Infant.Mortality 0.4165560 -0.06085861  -0.1140216
                   Education   Catholic Infant.Mortality
Fertility        -0.66378886  0.4636847  0.41655603
Agriculture      -0.63952252  0.4010951 -0.06085861
Examination       0.69841530 -0.5727418 -0.11402160
Education         1.00000000 -0.1538589 -0.09932185
Catholic         -0.15385892  1.0000000  0.17549591
Infant.Mortality -0.09932185  0.1754959  1.00000000

> # Создаём копию данных.
> swissNA <- swiss
> # Удаляем некоторые данные.
> swissNA[1,2] <- swissNA[7,3] <- swissNA[25,5] <- NA
> cor(swissNA)
Ошибка в cor(swissNA) : пропущенные наблюдения в cov/cor

> cor(swissNA, use = "complete")
                 Fertility Agriculture Examination
Fertility        1.0000000  0.37821953  -0.6548306
Agriculture      0.3782195  1.00000000  -0.7127078
Examination     -0.6548306 -0.71270778   1.0000000
Education       -0.6742158 -0.64337782   0.6977691
Catholic         0.4772298  0.40148365  -0.6079436
Infant.Mortality 0.3878150 -0.07168223  -0.1071005
                   Education   Catholic Infant.Mortality
Fertility        -0.67421581  0.4772298  0.38781500
```

```
Agriculture       -0.64337782  0.4014837 -0.07168223
Examination        0.69776906 -0.6079436 -0.10710047
Education          1.00000000 -0.1701445 -0.08343279
Catholic          -0.17014449  1.0000000  0.17221594
Infant.Mortality  -0.08343279  0.1722159  1.00000000

> cor(swissNA, use = "pairwise")

                  Fertility Agriculture Examination
Fertility         1.0000000  0.39202893  -0.6531492
Agriculture       0.3920289  1.00000000  -0.7150561
Examination      -0.6531492 -0.71505612   1.0000000
Education        -0.6637889 -0.65221506   0.6992115
Catholic          0.4723129  0.41520069  -0.6003402
Infant.Mortality  0.4165560 -0.03648427  -0.1143355
                    Education   Catholic Infant.Mortality
Fertility         -0.66378886  0.4723129  0.41655603
Agriculture       -0.65221506  0.4152007 -0.03648427
Examination        0.69921153 -0.6003402 -0.11433546
Education          1.00000000 -0.1791334 -0.09932185
Catholic          -0.17913339  1.0000000  0.18503786
Infant.Mortality  -0.09932185  0.1850379  1.00000000
```

Есть ещё два момента, на которые стоит обратить внимание. Первый — это ранговый коэффициент корреляции Спирмена (Spearman) ρ. Коэффициент ρ отражает меру монотонной зависимости и является более робастным (то есть он менее подвержен влиянию случайных «выбросов» в данных). Он полезен в случае, когда набор данных не получен выборкой из двумерного нормального распределения. Для подсчёта ρ достаточно приравнять опцию `method` значению `spearman`:

```
> x <- rexp(50);
> cor(x, log(x), method="spearman")
[1] 1
```

Можно сравнить, насколько сильно отличаются обычный коэффициента корреляции от коэффициента корреляции Спирмена:

```
> clP <- cor(longley)
> clS <- cor(longley, method = "spearman")
> i <- lower.tri(clP)
> cor(cbind(P = clP[i], S = clS[i]))
         P        S
P 1.000000 0.980239
S 0.980239 1.000000
```

Второй момент — это проверка гипотезы о значимости коэффициента корреляции. Это равносильно проверке гипотезы о равенстве нулю коэффициента корреляции. Если гипотеза отвергается, то влияние одного набора данных на другой считается *значимым*. Для проверки гипотезы используется функция cor.test:

```
> x <- rnorm(50)
> y <- rnorm(50, mean = 2, sd = 1);
> # Тестируем независимые данные.
> cor.test(x,y)

        Pearson's product-moment correlation

data:  x and y
t = 0.2496, df = 48, p-value = 0.804
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.2447931  0.3112364
sample estimates:
       cor
0.03600814

> # Тестируем линейно зависимые данные.
> cor.test(x, 2*x);

        Pearson's product-moment correlation

data:  x and 2 * x
t = Inf, df = 48, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 1 1
sample estimates:
cor
  1
```

Видно, что в первом случае гипотеза о равенстве нулю коэффициента корреляции не отвергается, что соответствует исходным данным. Во втором случае вызов был осуществлён с заведомо линейно-зависимыми аргументами и критерий отвергает гипотезу о равенстве нулю коэффициента корреляции с большим уровнем надёжности. Кроме непосредственно p-значения функция выводит оценку коэффициента корреляции и доверительный интервал для него. Выставить доверительный уровень для него можно с помощью опции `conf.level`. Также при помощи опции `method` можно выбирать, относительно какого коэффициента корреляции (простого или рангового) проводить проверку гипотезы значимости.

Следует признать, что смотреть на матрицу, полную чисел, не очень удобно. В **R** есть несколько способов, с помощью которых можно визуализировать корреляционную матрицу.

Первый способ — это использовать функцию symnum, которая выведет матрицу по-прежнему в текстовом виде, но все числа будут заменены на буквы, в зависимости от того, какому диапазону принадлежало значение:

```
> symnum(cor(longley))
             GNP. GNP U A P Y E
GNP.deflator 1
GNP          B    1
Unemployed   ,    ,   1
Armed.Forces .    .     1
Population   B    B   , . 1
Year         B    B   , . B 1
Employed     B    B   . . B B 1
attr(,"legend")
[1] 0 ‘ ’ 0.3 ‘.’ 0.6 ‘,’ 0.8 ‘+’ 0.9 ‘*’ 0.95 ‘B’ 1
```

Эта функция имеет большое количество разнообразных настроек, но по умолчанию они все выставлены в значения, оптимальные для отображения корреляционных матриц.

Второй способ — это графическое представление корреляционных коэффициентов. Идея проста: нужно разбить область от -1 до $+1$ на отдельные диапазоны, назначить каждому свой цвет, а затем всё это отобразить. Для этого следует воспользоваться функциями image и axis (рис. 5.3):

```
> C <- cor(longley)
> image(1:ncol(C), 1:nrow(C), C, col = rainbow(12),
+       axes = FALSE, xlab = "", ylab = "")
> # Подписи к осям.
>  axis(1, at = 1:ncol(C), labels=colnames(C))
>  axis(2, at = 1:nrow(C), labels=rownames(C), las = 2)
```

Ещё один интересный способ представления корреляционной матрицы предоставляется пакетом **ellipse**. В этом случае значения коэффициентов корреляции рисуются в виде эллипсов, отражающих форму плотности двумерного нормального распределения с данным значением корреляции между компонентами. Чем ближе значение коэффициента корреляции к $+1$ или -1 — тем более вытянутым становится эллипс. Наклон эллипса отражает знак. Для получения изображения необходимо вызвать функцию plotcorr (рис. 5.4):

```
> # Устанавливаем библиотеку.
> install.packages(pkgs=c("ellipse"))
> # Загружаем.
```

Рис. 5.3. Графическое представление корреляционной матрицы.

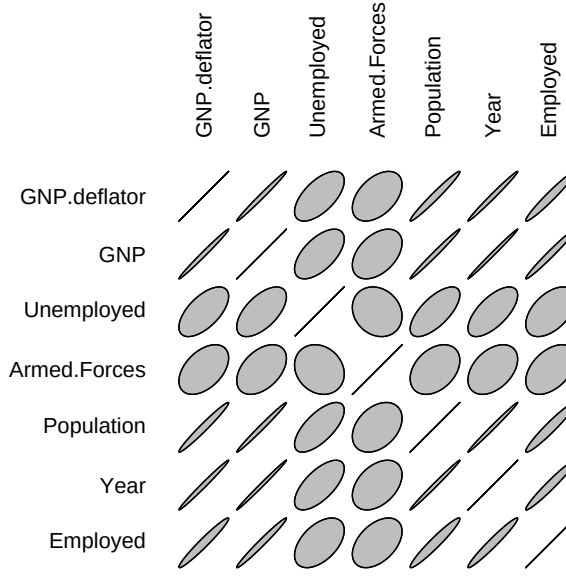

Рис. 5.4. Результат работы команды plotcorr из пакета **ellipse**.

```
> library(ellipse)
> # Используем.
> plotcorr(cor(longley))
```

5.3.2. Таблицы сопряжённости

Таблицы сопряжённости (contigency tables) — это удобный способ изображения категориальных переменных и исследования зависимостей между ними. Таблица сопряжённости представляет собой таблицу, ячейки которой индексируются градациями участвующих факторов, а числовое значение ячейки — количество наблюдений с данными градациями факторов. Построить таблицу сопряжённости можно с помощью функции table. В качестве аргументов ей нужно передать факторы, на основе которых будет строиться таблица сопряжённости:

```
> #  Таблица сопряжённости для выборки, имеющей
> # распределение Пуассона (n = 100, λ = 5).
> table(rpois(100,5))

 0  2  3  4  5  6  7  8  9 10 11
 1  7 18 17 22 13 13  4  1  1  3

> with(airquality, table(cut(Temp, quantile(Temp)), Month))

         Month
           5  6  7  8  9
  (56,72] 24  3  0  1 10
  (72,79]  5 15  2  9 10
  (79,85]  1  7 19  7  5
  (85,97]  0  5 10 14  5
```

R использует «честное» представление для трёх- и более мерных таблиц сопряжённости, то есть каждый фактор получает по своему измерению. Однако, это не очень удобно при выводе подобных таблиц на печать или сравнении с таблицами в литературе. Традиционно для этого используются «плоские» таблицы сопряжённости, когда все факторы, кроме одного, объединяются в один «многомерный» фактор и именно градации такого фактора используются при построении таблицы сопряжённости. Построить плоскую таблицу сопряжённости можно с помощью функцией ftable:

```
> ftable(Titanic, row.vars = 1:3)

                    Survived  No Yes
Class Sex    Age
1st   Male   Child            0   5
```

```
              Adult          118  57
      Female Child            0   1
              Adult           4 140
2nd   Male   Child            0  11
              Adult          154  14
      Female Child            0  13
              Adult           13  80
3rd   Male   Child           35  13
              Adult          387  75
      Female Child           17  14
              Adult           89  76
Crew  Male   Child            0   0
              Adult          670 192
      Female Child            0   0
              Adult           3  20
```

Опция `row.vars` позволяет указать номера переменных в наборе данных, которые следует объединить в один единый фактор, градации которого и будут индексировать строки таблицы сопряжённости. Опция `col.vars` проделывает то же самое, но для столбцов таблицы.

Функцию table можно использовать и для других целей. Самое простое — это подсчёт частот. Например, можно считать пропуски:

```
> d <- factor(rep(c("A","B","C"), 10), levels=c("A","B","C","D","E"))
> is.na(d) <- 3:4
> table(factor(d, exclude = NULL))

   A    B    C <NA>
   9   10    9    2
```

Функция mosaicplot позволяет получить графическое изображение таблицы сопряжённости[2] (рис. 5.5):

```
> mosaicplot(Titanic, main = "Survival␣on␣the␣Titanic", color = TRUE)
```

При помощи функции chisq.test[3] можно проверить гипотезу о независимости двух факторов. Например, проверим гипотезу о независимыми цвет глаз и волос:

```
> x <- margin.table(HairEyeColor, c(1, 2))
> chisq.test(x)
```

[2]Если стандартной функции plot передать в качестве аргумента таблицу сопряжённости, то вызывается именно эта функция

[3]Того же эффекта можно добиться если функции summary передать таблицу сопряжённости в качестве аргумента.

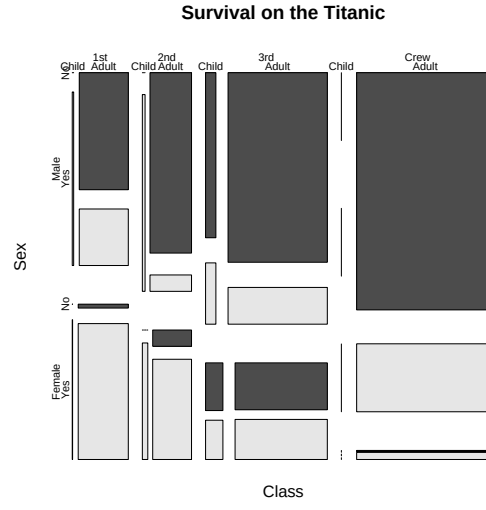

Рис. 5.5. Графическое представление таблицы сопряжённости.

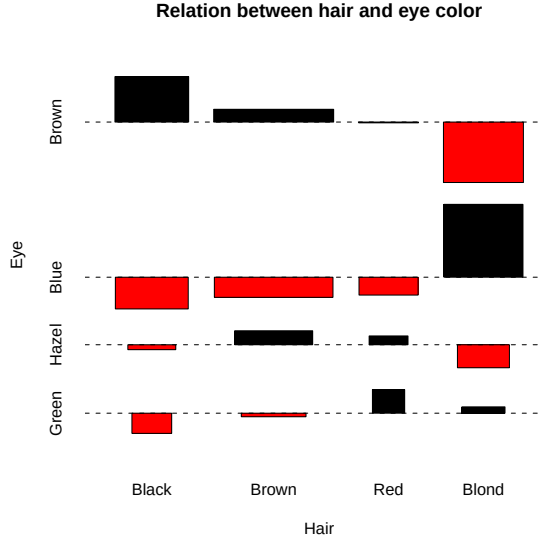

Рис. 5.6. Сравнение цвета глаз и волос с помощью функции assocplot.

```
        Pearson's Chi-squared test

data:  x
X-squared = 138.2898, df = 9, p-value < 2.2e-16
```

Набор данных `HairEyeColor` — это многомерная таблица сопряжённости. Здесь для суммирования частот по всем кроме двух «измерениям» использовалась функция margin.table. Таким образом в результате была получена двумерная таблица сопряжённости.

Чтобы зависимости между градациями двух факторов изобразить графически можно воспользоваться функцией assocplot. На рис. 5.6 показаны отклонения ожидаемых (при предположении независимости факторов) частот от наблюдаемых величин. Высота прямоугольника показывает абсолютную величину этого отклонения, а положение — знак отклонения. Ширина прямоугольника отображает собственно, саму ожидаемую величину значения в ячейка, но она не так информативна для предварительного анализа.

```
> x <- margin.table(HairEyeColor, c(1, 2))
> assocplot(x, main = "Relation␣between␣hair␣and␣eye␣color")
```

Здесь отчётливо видно, что, для людей со светлыми волосами характерен голубой цвет глаз и совсем не характерен карий цвет, а для обладателей чёрных волос ситуация в точности обратная.

► Поиск зависимостей между различными наборами данных — это любимое занятий человека разумного. Одно из самых больших интеллектуальных удовольствий — это обнаружение новой, ранее неизвестной, зависимости. Приятнее всего, если эта зависимость линейная.

Оглавление

Глава 6

Графический интерфейс к R

Работа с **R** ориентирована на консоль, но и мышь иногда тоже хочется чем нибудь занять, поэтому в этой статье будут рассмотрены разные дополнения к **R** с графическим интерфейсом, которые могут оказаться полезными в тяжёлом деле анализа данных.

6.1. Может лучше скажем мыши нет?

Прежде чем начать повествование хотелось бы сказать пару слов в пользу консольного интерфейса. Анализ данных — это творческий процесс. Ничего так этот процесс не продвигает, как неспешный вдумчивый ввод команд с клавиатуры. Естественно, этому вводу должен предшествовать интервал времени целиком и полностью посвящённый чтению документации, книг и статей по теме проблемы. Менюшки и кнопочки отвлекают, создавая иллюзию простоты творческого процесса, требуя нажать их немедленно и посмотреть что случится. Как правило, ничего при этом не случается, то есть всё равно придётся взять книгу в руки, а затем подумать.

Консольный интерфейс в **R**, идеален. Он предоставляет пользователю историю команд, позволяет по TAB дополнить их, сохраняет информацию и объекты между сессиями (при условии если пользователь это захочет, естественно). Нужно поработать на удалённом компьютере? Нет проблем с консольным интерфейсом. А если воспользоваться программой **screen** (`man screen` — много интересного), то можно не бояться «разорванных» сессий и случайно закрытых терминалов.

Вы всё ещё хотите использовать GUI при анализе данных? Ну что же, дело ваше, но помните, что Вас предупреждали. ☺

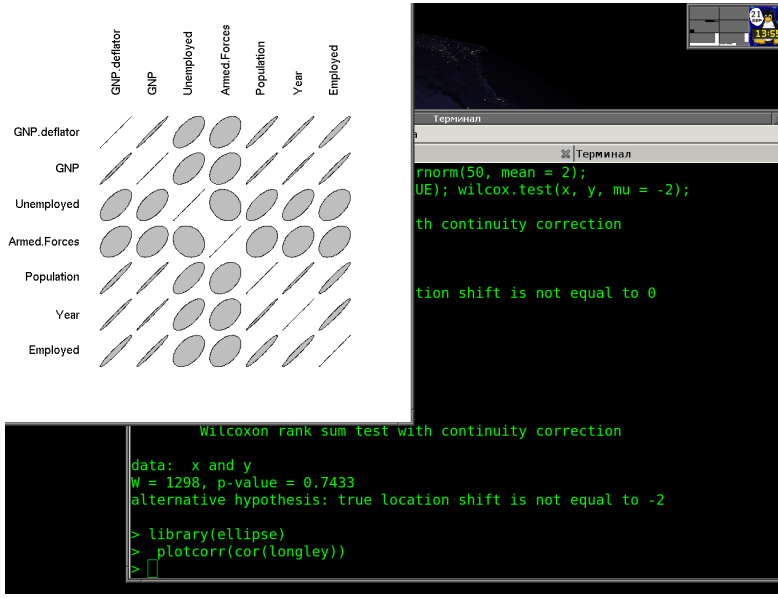

Рис. 6.1. Просто консоль

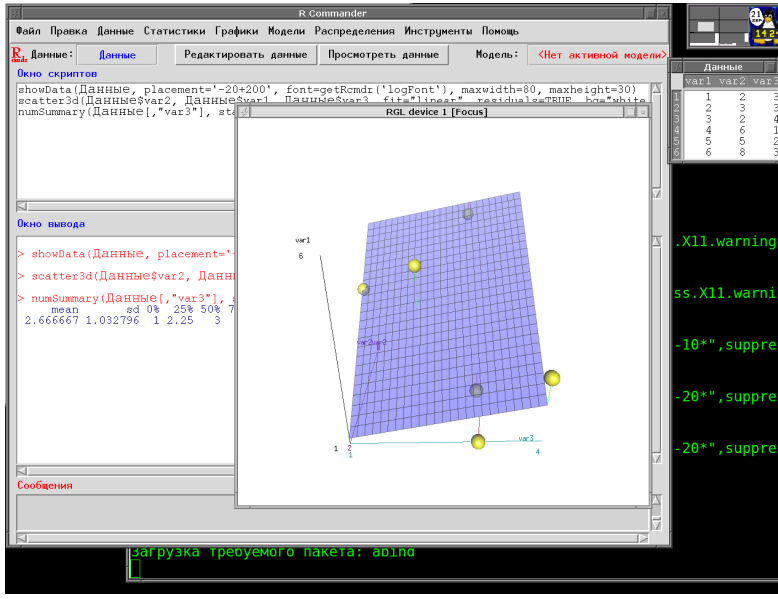

Рис. 6.2. **R Commander**

6.2. R Commander

R Commander или **Rcmdr** — это платформонезависимый графический интерфейс к **R**, написанный на Tcl/Tk. Домашняя страничка проекта располагается по адресу `http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/`.

Автор **R Commander** Джон Фокс (John Fox) признаётся, что если говорить о программах статистического анализа, то он не является фанатом интерфейса, состоящего из меню и диалогов. С его точки зрения **R Commander** полезен в основном для образовательных целей при введении в **R**, а также в отдельных очень редких случаях для быстрого анализа. Одна из основополагающих целей, преследуемых при создания интерфейса **R Commander**, являлась мягким переводом пользователя в консоль, где можно автоматизировать свои действия более глобально,

Пакет распространяется под лицензией GNU/GPL, и поэтому доступен во всех стандартных дистрибутивах GNU/Linux. Например, для установки в Ubuntu 8.04 достаточно выполнить команду:

```
% sudo aptitude install r-cran-rcmdr
```

Пакет присутствует также и на CRAN, поэтому его установку можно произвести и силами **R**:

```
> install.packages("Rcmdr", dependencies=TRUE)
```

Обратите внимание на значение опции `dependencies`. **R Commander** зависит от довольно большого числа пакетов. Кроме стандартных **r-base** библиотек **R Commander** предполагает что в системе уже установлены пакеты **abind**, **aplpack**, **car**, **effects**, **lmtest**, **multcomp**, **relimp**, **rgl** и (только для Windows) **RODBC**, поэтому до установке самой программы необходимо убедиться, что все эти пакеты уже стоят. Установка **R Commander** с помощью системных программ установки **aptitude**/**apt-get**, как и установка опции `dependencies` в состояние `TRUE` позволяет не беспокоиться об этих проблемах. Все необходимые пакеты будут скачены и установлены в нужном порядке.

После установки для запуска **R Commander** в сессии **R** следует выполнить команду:

```
> library(Rcmdr)
Загрузка требуемого пакета: tcltk
Загружаю интерфейс Tcl/Tk... готово
Загрузка требуемого пакета: car

Версия R Commander 1.3-9
```

При запуске **R Commander** открывается окно, разделённое на **Окно Скриптов** (Script Window), **Окно Вывода** (Output Window), информационное окно **Сообщения** (Messages) и снабжённое довольно «развесистым» меню. Многие

действия в **R Commander** можно выполнять через меню, которое достаточно легко настраивается, например, с помощью редактирования текстового файла `Rcmdr-menus.txt`. Так же можно вводить команды в **Окне Скриптов**, то есть консоль никуда не делась. При создании графиков они появляются в отдельных окнах, как и в случае **R**.

R Commander имеет русскую локализацию которая активизируется автоматически если выставлена русская локаль:

```
% locale
LANG=ru_RU.UTF-8
```

Основная проблема русской локализации — это кириллические шрифты, которые выставляются по умолчанию. Поэтому, если предпочтительно использовать локализованный интерфейс, то перед загрузкой **R Commander** ему с помощью команды options следует передать примерно следующие опции:

```
> options(Rcmdr=list(
+  default.font="-rfx-fixed-medium-r-normal-*-20*",
+  suppress.X11.warnings=TRUE))
```

Растровый шрифт семейства rfx от Дмитрия Болхивитянова (20 в конце строки — это размер шрифта по умолчанию), который указывается в качестве `default.font`, идёт в составе пакета xfonts-bolkhov. Вторая опция `suppress.X11.warnings` подавляет надоедливые сообщения при создании новых графических окон.

Выйти из **R Commander** можно через меню **Файл→Выйти**, при этом можно одновременно закрыть и сессию **R**, а можно выйти только из **R Commander**. Если сессия **R** осталась открытой, то повторный запуск **R Commander** выполняется с помощью команды:

```
> Commander()
```

Для вводного ознакомления с возможностями **R Commander** следует прочитать текст `Getting-Started-with-the-Rcmdr.pdf` получить доступ к которому можно через меню **Помощь→Введение в R Commander**.

Если по какой-то причине было принято решение анализировать данные исключительно с помощью **R Commander**, то можно сделать так, чтобы при запуске **R** эта графическая оболочка загружалась автоматически. Для этого в файл для пользовательских настроек `~/.Rprofile` достаточно добавить следующие строки:

```
message("Запуск R Commander");
old <- getOption("defaultPackages");
options(defaultPackages = c(old, "Rcmdr"));
```

Глобальная переменная `defaultPackages` содержит информацию об автоматически загружаемых модулей при старте **R**.

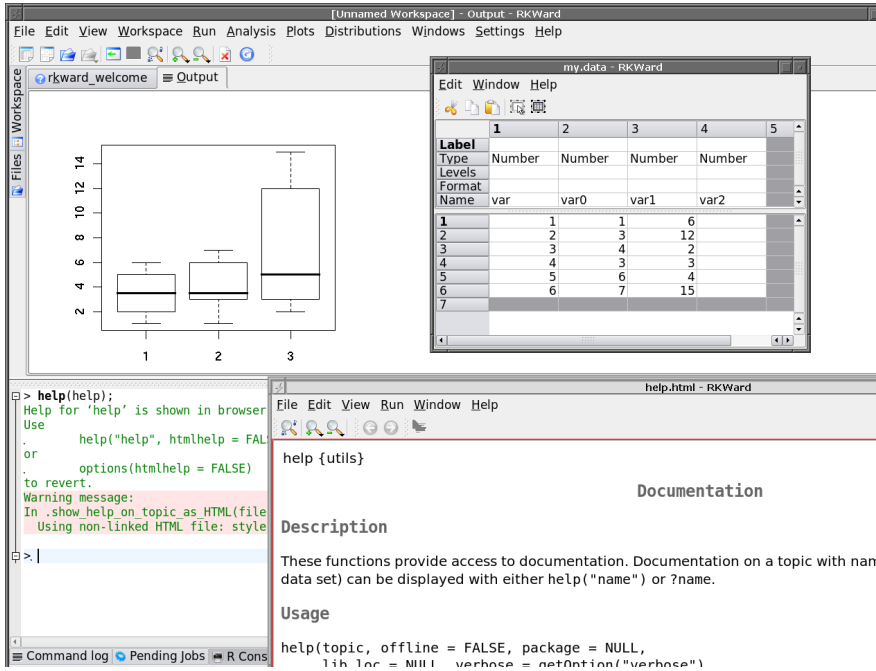

Рис. 6.3. **RKWard** — **R** и KDE

6.3. RKWard

RKWard (`http://rkward.sourceforge.net`) — это довольно удобный интерфейс к языку программирования **R**. Разработчики **RKWard** старались совместить мощь **R** с простотой использования, подобной предоставляемой коммерческими статистическими пакетами (такими, как, например, SPSS или Statistica).

Стоит отметить, что это им вполне удалось. Для начинающих пользователей **RKWard** предоставляет широкие возможности по выбору многих стандартных процедур статистического анализа по принципу «point-and-click»: для этого достаточно выбрать соответствующий пункт меню.

Для продвинутого пользователя **RKWard** предлагает достаточно удобный редактор кода с подсветкой, автоматической расстановкой отступов, автодополнением — теми вещами, без которых в настоящее время не обходится ни одна среда программирования. Привычная консоль **R** также никуда не делась, она доступна в любое время на вкладке «R Console».

Авторы **RKWard** взяли курс на как можно более полную интеграцию функций **R** в графическую среду: присутствует браузер текущего окружения, редактор данных. Есть менеджер пакетов, умеющий не только их устанавливать, но и следить за обновлениями, прозрачная интеграция со справочной системой. Кроме

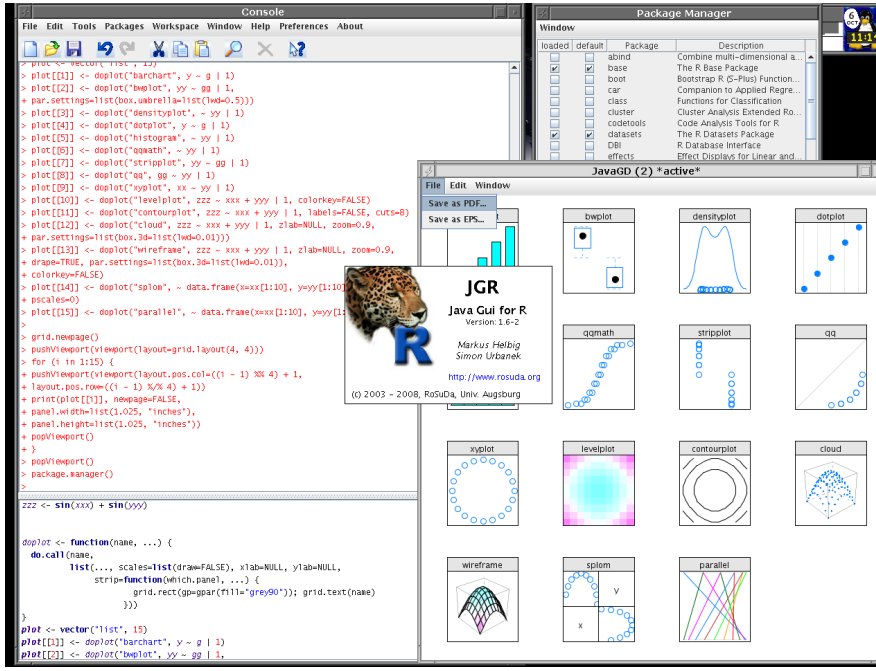

Рис. 6.4. **JGR** — интерфейс рисует Java.

того, **RKWard** умеет «перехватывать» создание графических окон и добавлять к ним чрезвычайно удобные функции типа сохранения содержимого графического окна в файл одного из стандартных форматов, поддерживаемых **R** (PDF, EPS, JPEG, PNG).

Интерфейс **RKWard** чрезвычайно гибок: пользователь может его расширять за счёт написания собственных плагинов (собственно, все встроенные средства анализа, доступные сразу же после запуска из меню — это тоже плагины, но только написанные авторами **RKWard**).

Программа использует библиотеки KDE (есть и для KDE 4).

6.4. JGR

Предпочитаете, чтобы интерфейс рисовался с помощью Java? Тогда **JGR** (**J**ava **G**UI для **R**) — это решение. **JGR** (произносится как Ягуар) впервые был представлен общественности в 2004 году и до сих пор развивается и поддерживается. Пакет **JGR** распространяется под лицензией GPL2 и его домашняя страничка доступна по адресу `http://jgr.markushelbig.org/JGR.html`.

JGR был «made on Mac» со всеми вытекающими, но под Linux он тоже работает. Для использования **JGR** необходимо установить Sun Java(TM) Development Kit (JDK).

```
> # Установка Sun JDK
> apt-get install sun-java6-jdk
> # Выбор Sun JDK в качестве основного из имеющихся альтернатив
> sudo update-java-alternatives -s java-6-sun
> # Настройка Java окружения для R
> sudo  R CMD javareconf
```

После установки и настройки Java-окружения следует запустить сессию **R** и выполнить следующие действия:

```
> install.packages('JGR')
# В качестве зеркала выбираю Швейцарию
> library(JGR)
> JGR()
Starting JGR run script. This can be done from the shell as well,
just run /home/user/R/i486/2.6/JGR/scripts/run
```

Всё, **JGR** теперь запущен. После установки **JGR** так же можно запустить с помощью скрипта, путь до которого указывается при запуске JGR. В приведённом выше примере `/home/user/R/i486/2.6/JGR/scripts/run`.

В **JGR** есть встроенный текстовый редактор, который конечно, не является полноценной заменой Emacs/vim, но в нём есть подсветка синтаксиса и завершение команд по TAB. В **JGR** встроена гипертекстовая помощь, простенькая электронная таблица, добавлена возможность управлять объектами в том числе и с помощью мышки, а так же есть графический интерфейс для установки и загрузки R-пакетов. С учётом того, что этот GUI может запускаться везде, где есть Java, то на него сто́ит обратить внимание.

6.5. SciViews-K

В мире универсальных редакторов и IDE достаточно известна платформа разработки Komodo от ActiveState. Она довольно хорошо себя зарекомендовала, как удобная среда для языков с динамической типизацией: PHP, Python, Perl, Ruby.

SciViews-K (`http://www.sciviews.org/SciViews-K/index.html`) — это молодой и динамично развивающийся проект (первый релиз был не так давно — в июне 2008 года), добавляющий **R** к семейству языков, поддерживаемых свободным редактором с открытым исходным кодом (лицензии как у Mozilla) Komodo Edit (`http://www.activestate.com/Products/komodo_ide/komodo_edit.mhtml`).

На настоящий момент возможности **SciViews-K** довольно скромны: более чем стандартный редактор кода с подсветкой, интеграция с командной строкой **R** и

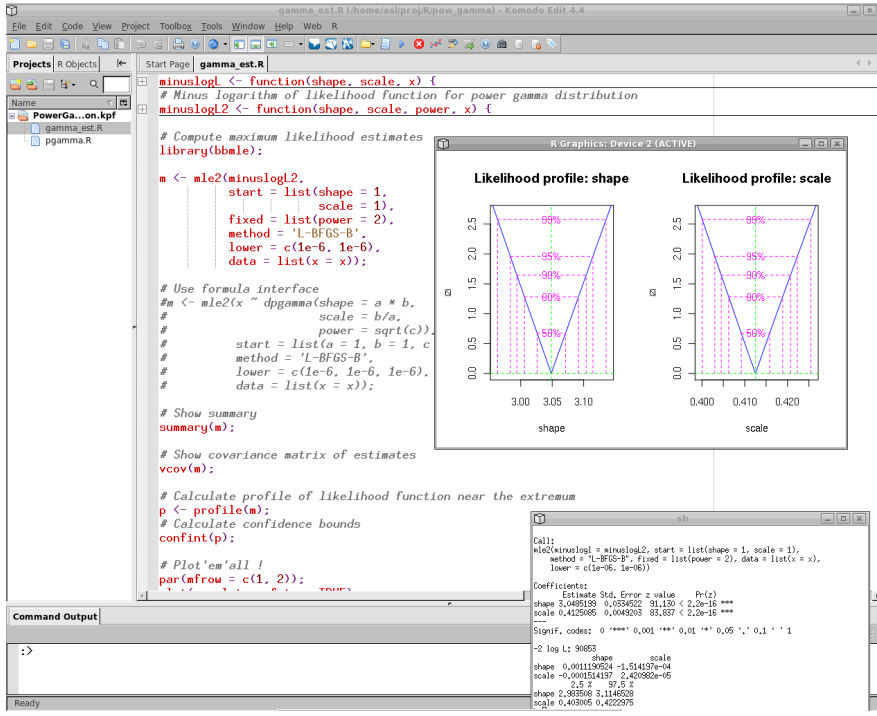

Рис. 6.5. **SciViews-K** — расширение для Komodo Edit.

системой справки. Кроме того, доступен просмотр текущего окружения и редактор наборов данных.

Поскольку **SciViews-K** является расширением для редактора Komodo Edit, то его установка производится в два приема: сначала надо установить собственно сам Komodo Edit, а потом уже установить **SciViews-K** как расширение к нему. Кроме того, при первом запуске **R** из под **SciViews-K** будет установлен пакет SciViews вместе с зависимостями.

6.6. Rattle

Rattle (`http://rattle.togaware.com/`) — это сокращение, которое расшифровывается как «the **R** **A**nalytical **T**ool **T**o **L**earn **E**asily» (легкая в освоении среда анализа **R**). По уверениям разработчиков **Rattle** он предназначен для интеллектуального анализа данных (data mining), иными словами для выявление скрытых закономерностей или взаимосвязей между переменными в больших массивах необработанных данных. **Rattle** — это среда для «разглядывания» данных человеком и программа всячески человеку в этом деле помогает. Альтернативное

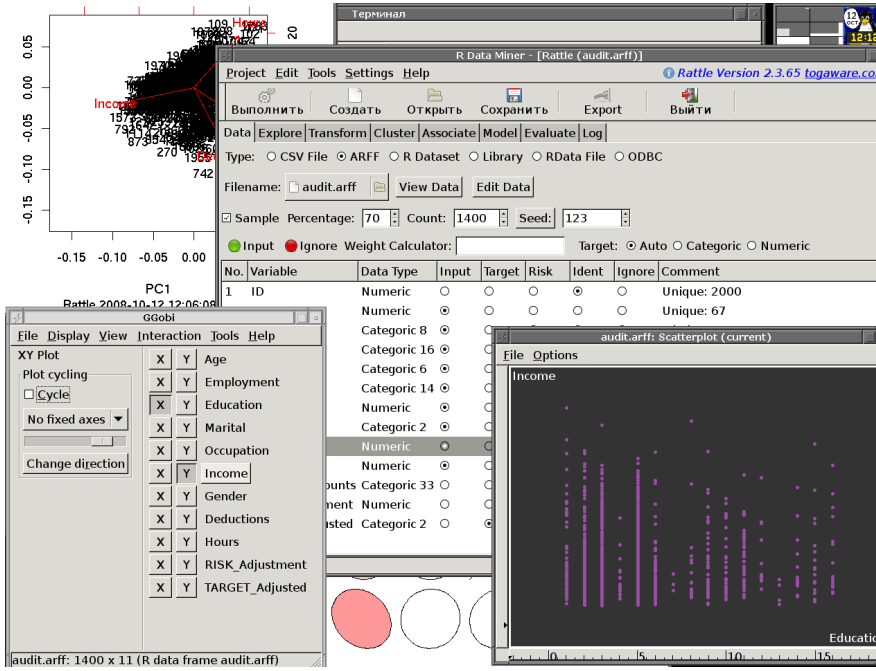

Рис. 6.6. **Rattle** и анализ данных в нём.

название пакета «A GNOME Data Miner Built on **R** » намекает, что этот пакет интегрирован в графическую среду GNOME.

Для установки **Rattle** необходимо чтобы уже присутствовали пакеты r-base-core, ggobi (визуализация данных) и libglade2-dev:

```
> apt-get install r-base-core ggobi libglade2-dev
```

После этого в консоли **R** следует выполнить команду:

```
> install.packages("rattle", dependencies=TRUE)
```

и откинуться на спинку кресла. Из-за большого количества зависимостей установка занимает приличный промежуток времени.

Запуск GUI производится, как обычно:

```
> library(rattle)
> rattle()
```

Простой интерфейс анализа позволяет использовать **Rattle** для обучения основам анализа. Программа активно развивается.

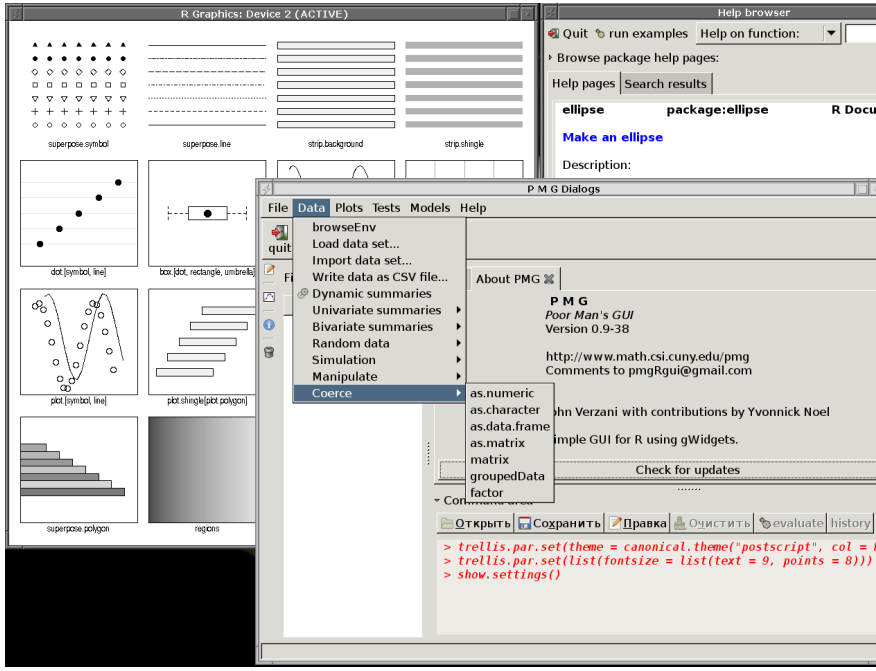

Рис. 6.7. **PMG** — GUI «для бедных».

6.7. PMG

Нужен графический интерфейс, но нет особого желания разбираться в «навороченных» GUI, тогда, возможно, имеет смысл обратить внимание на пакет **PMG** (**P**oor **M**an's **G**ui `http://wiener.math.csi.cuny.edu/pmg`).

Установка проста, хотя у пакета есть множество зависимостей, и проводится силами **R**. Пакет **PMG** является кроссплатформенным. Для отрисовки графического интерфейса используется библиотека GTK через пакет **RGtk2**, который в свою очередь можно использовать для создания своего GUI.

```
> # Установка PMG
> install.packages("pmg", dep=TRUE)
> # Запуск PMG
> library(pmg)
```

Этому графическому интерфейсу определённо есть куда развиваться.

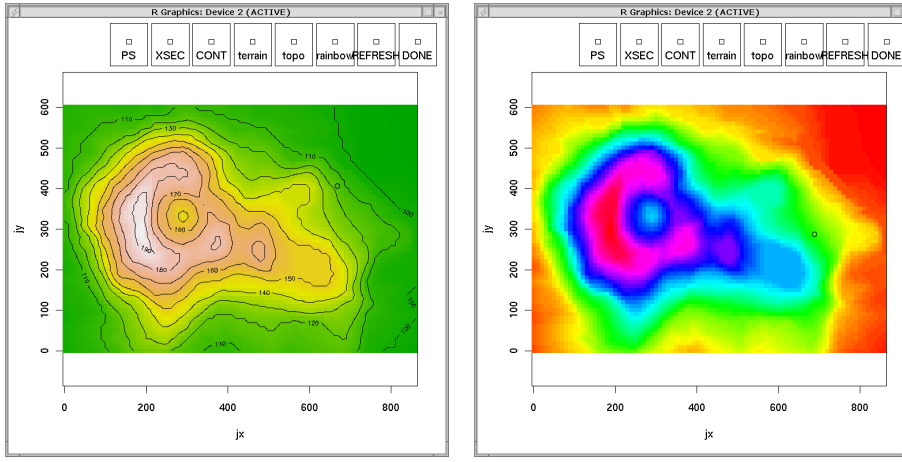

Рис. 6.8. **RPMG** — GUI «для действительно бедных».

6.8. RPMG

Что делать, если нет необходимости в сложной переливающейся графике, а нужно пара простейших интерактивных кнопочек на картинке и минимальное число зависимостей от других пакетов? В этом случае поможет **RPMG** — GUI для действительно бедных (**R**eally **P**oor **M**an's **G**UI), зато с его помощью легко можно организовать интерактивное графическое **R**-окно для личного пользования. На рис. 6.8 одно и тоже окно которое меняется в зависимости от активации кнопочек (набор прямоугольничков в верхней части экрана) с помощью мышки. Для установки пакета и демонстрации его возможностей следует выполнить следующие команды:

```
> install.packages("RPMG", dep=TRUE)
> library(RPMG)
> demo(RPMG)
```

Пакет не имеет домашней страницы, но его всегда можно скачать с CRAN: `http://probability.ca/cran/web/packages/RPMG/index.html`

6.9. RWeb

Нет желания давать студентам в руки ничего опаснее WEB браузера. Странное желание, но и оно для преподавания статистики с использованием **R** не помеха. Простой набор perl-скриптов **Rweb** (**R** Web Based Statistical Analysis) (`http://www.math.montana.edu/Rweb/`) можно использовать как отправную точку для реализации своего интерактивного Web-проекта. На сайте проекта есть

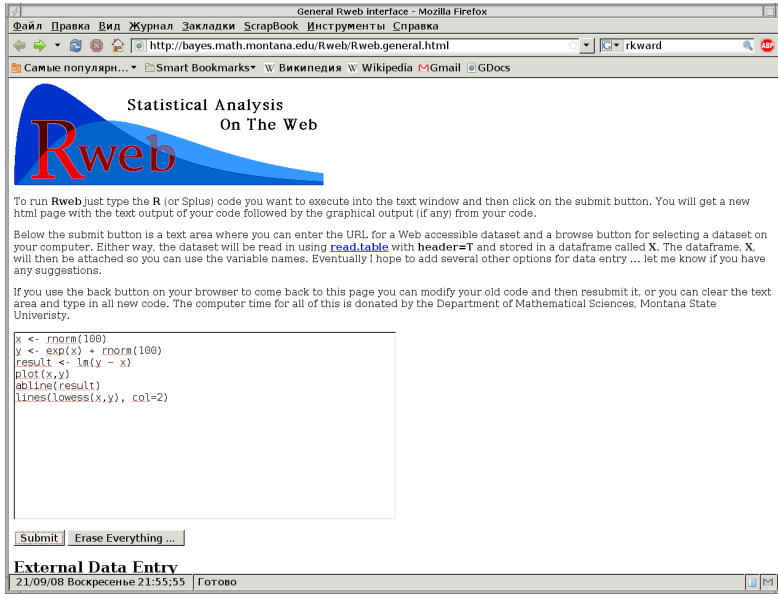

Рис. 6.9. Rweb (ввод данных)

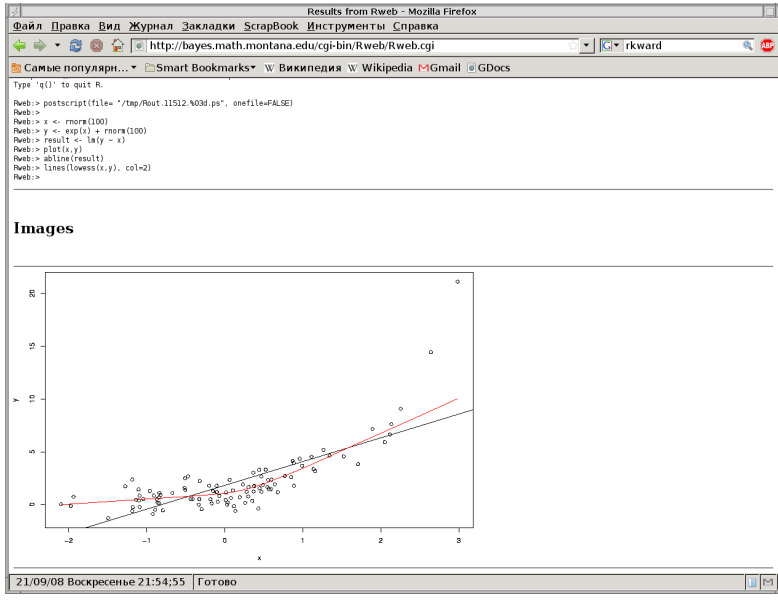

Рис. 6.10. Rweb (результат выполнение команд)

его рабочая демонстрация. Там можно прямо в окне браузера набрать команды **R** и рассмотреть результаты в том же окне. Проект давно не обновлялся, но его код вполне обозрим и его можно легко подстроить под свои нужды.

6.10. gnumeric

Gnumeric (`http://www.gnome.org/projects/gnumeric`) — это табличный процессор с открытым исходным кодом. Это то, что называется электронная таблица со всеми преимуществами и недостатками этого типа программ. Gnumeric поддерживает систему подключаемых модулей, позволяющая расширять функциональность. В 2003 году сотрудник Bell Labs Дункан Лэнг (Duncan Temple Lang) написал плагин, который позволяет вызвать прямо из Gnumeric любую функцию **R**. Этот плагин был назван без особых затей **RGnumeric** и взять его можно по адресу (`http://www.omegahat.org/RGnumeric/`).

К сожалению пакет для интеграции **R** с Gnumeric представляет из себя скорее демонстрацию возможностей, нежели законченный продукт. В рамках Google of Summer Code 2008 была попытка возродить идеи заложенные в этом пакете на новом уровне `http://www.r-project.org/SoC08/ideas.html`, но похоже, проект всё ещё ждёт своего автора. В любом случае исходники доступны и, хотя просто сама их сборка представляет из себя некоторое приключение, но истинные энтузиасты не ищут лёгких путей.

6.11. Emacs

Что это мы всё о GUI, да о GUI. Пора поговорить о **emacs**. Да, да про тот самый программный продукт в который встроена программа управления кофеваркой. Говорят она работает, причём очень не плохо. Посмотрим как **emacs** справляется с **R**.

ESS (`http://ess.r-project.org`) — это специализированная интерактивная **R**-среда или мода для GNU Emacs/XEmacs. **ESS** является сокращением от фразы «Emacs Speaks Statistics», которую можно перевести как «Emacs говорит на языке статистики». Пакет **ESS** поддерживает не только систему статистического анализа **R**, но и другие диалекты языка S (S 3/4, S-PLUS 3/4/5/6/7), а так же SAS, XLispStat, Stata и BUGS. При установке в пакета **ESS**, например, так:

```
> sudo aptitude install ess
```

и добавлении строчки для инициализации **ESS** в `~/.emacs`

```
;; подключаем ESS
(require 'ess-site)
```

редактору Emacs становятся доступны две дополнительные моды:

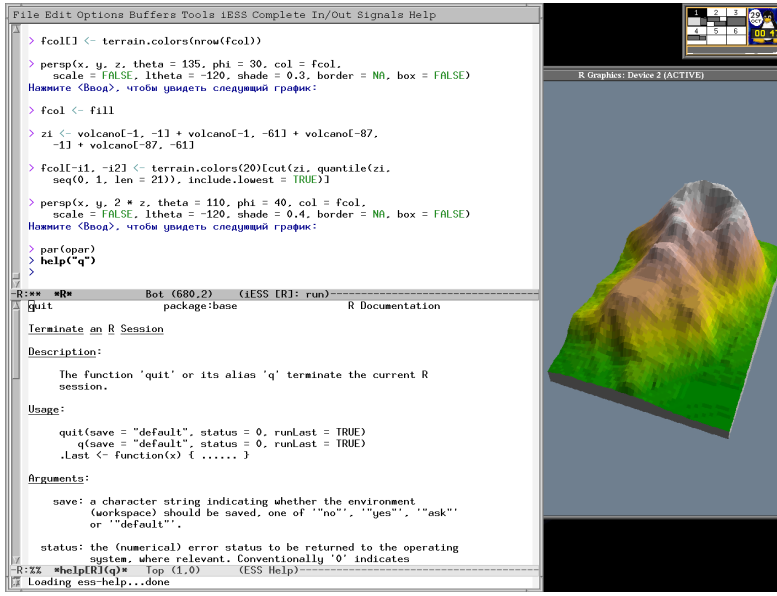

Рис. 6.11. **ESS** делает **R** частью emacs.

1) мода ESS, которая предназначена для редактирования исходных файлов с использованием команд **R** (включая дополнительную поддержку для редактирование Rnw-файлов), и

2) мода iESS — замена обычной **R** консоли.

Интерпретатор **R** запускается, как это не странно с помощью команды R: **Alt**+**x**, а затем R и перевод строки. При запуске спрашивается о местоположении рабочей директории для запускаемой сессии **R**, а затем отображается стандартное **R** приглашение. Можно параллельно запустить насколько сессий, также есть возможность запустить **R** -сессию на удалённом компьютере (команда ess-remote). Работа в этой моде почти ничем не отличается от работе в консоли, за исключением наличия стандартных возможностей редактирования текста в Emacs и дополнительного вспомогательном меню iESS. Из этого меню можно получить доступ к таким интересным возможностям как, например, редактирование объектов **R** (**^c ^d**). Emacs — это прежде всего текстовый редактор, поэтому подобные особенности делают анализ данных комфортнее.

ESS-мода инициализируется автоматически при редактировании файлов с расширением `R`. Emacs предоставляет пользователю подсветку синтаксиса, выравнивание исходного кода по TAB, исполнение отдельных строк, фрагментов и всего файла в интерпретаторе **R**. Дополнительное меню **ESS** позволяет получить доступ к этим возможностям. Редактирование Rnw-файлов (смесь LaTeX

и **R** команд для быстрого составления отчётов с помощью пакета **Sweave** — подробнее об этом было рассказано в разделе «Мастера отчётов» в главе «Данные и графики») чуть более мудрёно: мода **ESS** активируется только когда курсор оказывается внутри фрагментов с **R** -командами.

Если есть навыки работы в Emacs, то **ESS** — это верный выбор.

6.12. Ещё редакторы и среды

R широко известен хоть и в довольно узких кругах, поэтому почти все уважающие себя текстовые редакторы или среды разработки поддерживают его в той или иной степени. Кроме уже упомянутого Emacs к ним относится и Vim, и jEdit, и bluefish, и SciTE, и, естественно, монстр Eclipse имеет специальный плагин для поддержки **R** (`http://www.walware.de/goto/statet`). В этой главе рассказ коснулся только свободных пакетов, но наверняка не сложно будет найти поддержку и в коммерческих аналогах.

Оглавление

Глава 7

Интеллектуальный анализ данных или Data Mining

Фраза «data mining» всё чаще и чаще располагается на обложках книг по анализу данных, не говоря уж о росте упоминаний об этом методе во всеядном интернете. Говорят даже, что эпоха статистики одной-двух переменных закончилась, и наступило новое время — время интеллектуального анализа данных (data mining) больших и сверхбольших массивов данных.

На самом деле под методами «data mining» подразумеваются любые методы, как визуальные, так и аналитические, позволяющие «нащупать» структуру в данных, особенно в данных большого размера. В данных же небольшого размера структуру часто можно найти и не прибегая к специальным методам, например, просто поглядев на распределение того единственного параметра который изучается.

Данные для анализа используются, как правило, многомерные, то есть такие, которые можно представить в виде таблицы из нескольких колонок-переменных. Поэтому более традиционным названием для этих методов является «многомерный анализ», или «многомерная статистика», но «data mining» звучит, конечно, серьёзнее. Кроме многомерности и большого размера (сотни, а то и тысячи строк и столбцов), используемые данные отличаются ещё и тем, что переменные в них могут быть совершенно разных типов (качественные, балльные, счётные, непрерывные), причём даже «типичные» для статистики непрерывные числовые переменные вполне могут не подчиняться заранее известным законам распределения, то есть могут не быть параметрическими.

Сами по себе, многомерные методы делятся на визуализационные методы и методы классификации с обучением. В первом случае результат можно анализировать в основном зрительно, а во втором — возможна статистическая проверка

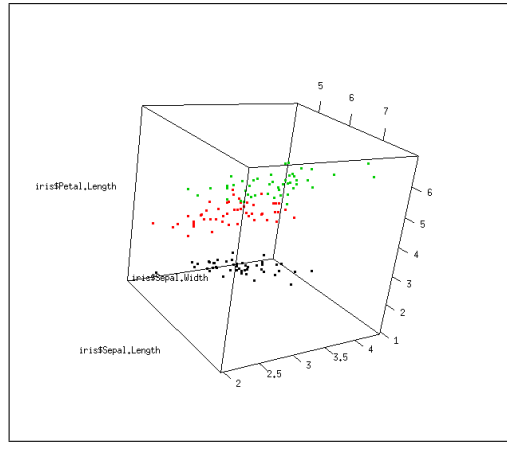

Рис. 7.1. Отображение многомерных данных с помощью пакета **RGL**.

результатов. Разумеется, граница между этими группами не резкая, но для удобства мы будем рассматривать их именно в этом порядке.

7.1. Графический анализ многих переменных

Самое простое, что можно сделать с многомерными данными — это построить график. Разумеется, для того, чтобы построить график предварительно надо свести всё разнообразие многомерных данных к двум, или в крайнем случае, к трём измерениям. Эта операция называется «сокращением размерности». Если переменных уже три и все три переменные непрерывные, то с представлением данных замечательно справится **RGL**, написанный с использованием OpenGL, и позволяющий трёхмерную визуализацию.

► Для примеров анализа в этой главе мы будем использовать встроенные в **R** данные `iris`. Это данные заимствованны из работы знаменитого математика (и биолога) Р. Фишера. Они описывают разнообразие нескольких признаков трёх видов ирисов (многолетние корневищные растения, относящиеся к семейству Касатиковых или Ирисовых). Эти данные состоят из 5 переменных (колонок), причём последняя колонка — это название вида.

А теперь визуализируем четыре из пяти колонок `iris` при помощи пакета **RGL**:

```
> # Инициализация RGL
> library(rgl)
> # Sepal - чашелистик, Petal - лепесток, Species - вид
> plot3d(iris$Sepal.Length, iris$Sepal.Width,
+        iris$Petal.Length, col=iris$Species, size=3)
```

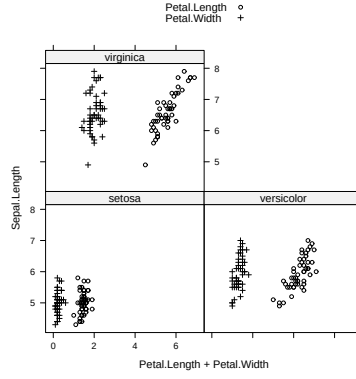

Рис. 7.2. Отображение многомерных данных с помощью пакета **lattice**.

Размер появившегося окна и проекцию отображения данных можно и нужно менять с помощью мышки.

Сразу видно, что один из видов (Iris setosa) хорошо отличается от двух других по признаку длины лепестков (Petal.Length). Кстати, в том что мы «визуализировали 4 признака», не было оговоркой, так как вид ириса — это тоже признак, закодированный в данном случае цветом.

Для трёхмерной визуализации данных можно обойтись и без OpenGL, например, при помощи пакета **scatterplot3d**. Но, по хорошему, трёхмерными графиками лучше не злоупотреблять. Очень часто двумерные проекции трёхмерных объектов не раскрывают, а «затемняют» суть явления. Правда, в случае **RGL** это компенсируется возможностью свободно менять «точку обзора».

Есть и более специализированные системы для визуализации многомерных данных без снижения размерности. Среди них можно отметить пакет **rggobi**, основанный на системе **Ggobi**. В этой главе мы его рассматривать не будем, так как, строго говоря, использование «постороннего программного обеспечения» уже выводит нас за рамки **R**.

Ещё один способ визуализации многомерных данных — это построение графиков-таблиц. Здесь **R** обладает колоссальными возможностями, которые предоставляются пакетом **lattice**. Пакет **lattice** предназначен для так называемой Trellis-графики. Долгое время Trellis-графика была общепризнанной изюминкой S-PLUS, а теперь эта изюминка доступа и в **R**. Вот как можно визуализировать четыре признака ирисов:

```
> # Инициализация lattice
> library(lattice)
> xyplot(Sepal.Length ~ Petal.Length + Petal.Width | Species,
+        data = iris, auto.key=TRUE)
```

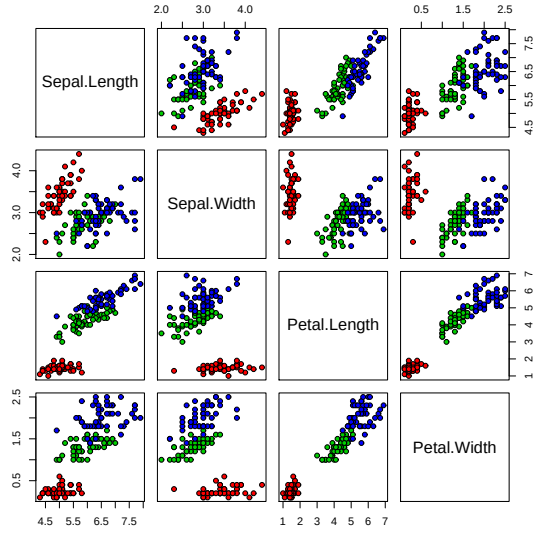

Рис. 7.3. Отображение многомерных данных с помощью пакета **pairs**.

В результате получилось графическое представление зависимости длины чашелистиков, как от длины, так и от ширины лепестков для каждого из трёх видов.

Можно обойтись и без **lattice**, так как есть несколько Trellis-подобных графиков доступных прямо в базовой поставке **R**. Самый простой из них — это pairs():

```
> pairs(iris[1:4], pch = 21,
+  bg = c("red", "green3", "blue")[unclass(iris$Species)])
```

Здесь мы получили зависимость значения каждого признака от каждого признака, причём заодно и «покрасили» точки в цвета видов. Таким образом, нам удалось визуализировать сразу *пять* переменных.

7.2. Сокращение размерности

Перейдём теперь к методам сокращения размерности. Самый распространённый из них — это «анализ главных компонент». Суть анализа главных компонент заключается в том, что все признаки-колонки преобразуются в компоненты, причём наибольшую информацию о разнообразии объектов несёт первая компонента, вторая несет меньше информации, третья — ещё меньше и так далее.

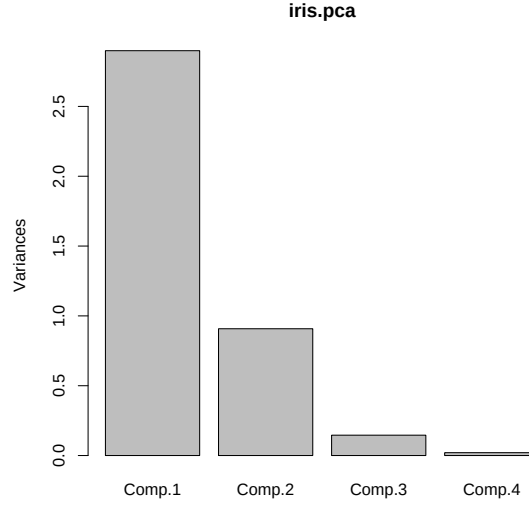

Рис. 7.4. Анализ главных компонент (служебный график)

Таким образом, хотя компонент получается столько же, сколько изначальных признаков, но в первых двух-трёх из них сосредоточена почти вся нужная нам информация, поэтому их можно использовать для визуализации данных на плоскости. Обычно используется первая и вторая (реже первая и третья) компоненты. Компоненты часто называют «факторами», и это порождает некоторую путаницу с похожим на анализ главных компонент «факторным анализом», преследующим, однако совсем другие цели.

Вот как делается анализ главных компонент на наших данных про ирисы:

```
> iris.pca <- princomp(scale(iris[,1:4]))
```

Мы использовали функцию scale() для того, чтобы привести все четыре переменные к одному масштабу (эта функция по умолчанию вычитает из данных среднее и делит их на квадрат среднего значения), поскольку переменные, варьирующие в разных масштабах, способны исказить результат анализа. В **R** реализован и другой метод анализа главных компонент, основанный на иных преобразованиях матрицы, и вызываемый функцией prcomp().

Выведем служебный график, показывающий относительные вклады каждого компонента в общий разброс данных:

```
> plot(iris.pca)
```

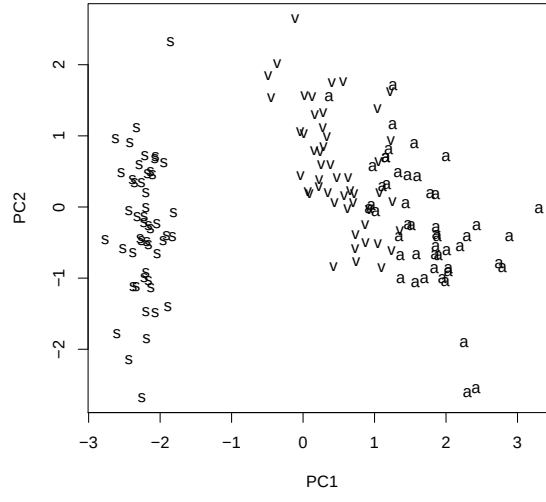

Рис. 7.5. Анализ главных компонент

На графике (рис. 7.4) хорошо видно, что компонент четыре, как и признаков, но в отличие от первоначальных признаков наибольший вклад вносят первые два компонента. Вместо графика можно получить тоже самое в текстовом виде, набрав:

```
> summary(iris.pca)
Importance of components:
                          Comp.1    Comp.2     Comp.3      Comp.4
Standard deviation     1.7026571 0.9528572 0.38180950 0.143445939
Proportion of Variance 0.7296245 0.2285076 0.03668922 0.005178709
Cumulative Proportion  0.7296245 0.9581321 0.99482129 1.000000000
```

Теперь перейдём к собственно визуализации:

```
> iris.p <- predict(iris.pca)
> plot(iris.p[,1:2], type="n", xlab="PC1", ylab="PC2")
> text(iris.p[,1:2],
+      labels=abbreviate(iris[,5],1, method="both.sides"))
```

На рис. 7.5 представлено разнообразие ирисов. Получилось, что Iris setosa (маркируется буквой «s») сильно отличается от двух остальных видов, Iris versicolor («v») и Iris virginica («a»). Функция predict() позволяет расположить исходные

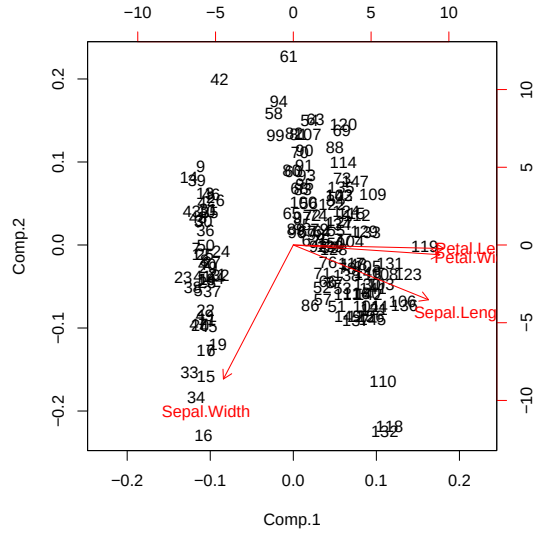

Рис. 7.6. Действие функции biplot

случаи (строки) в пространстве вновь найденных компонент. Функция abbreviate() «умным» образом сокращает названия до одной буквы.

Иногда для анализа данных бывает полезной и функция biplot():

```
> biplot(iris.pca)
```

Результирующий график позволяет понять, насколько силён вклад каждого из четырёх исходных признаков в первые две компоненты. В данном случае хорошо видно, что признаки длины и ширины лепестков (в отличие от признаков длины и ширины чашелистиков) вносят гораздо больший вклад в первую компоненту, которая собственно, и «различает» виды. К сожалению, графиком, выдаваемым при помощи biplot(), довольно трудно «управлять», поэтому часто предпочтительнее воспользоваться выводом функция loadings():

```
>  loadings(iris.pca)

Loadings:
             Comp.1 Comp.2 Comp.3 Comp.4
Sepal.Length  0.521 -0.377  0.720  0.261
Sepal.Width  -0.269 -0.923 -0.244 -0.124
Petal.Length  0.580        -0.142 -0.801
Petal.Width   0.565        -0.634  0.524
```

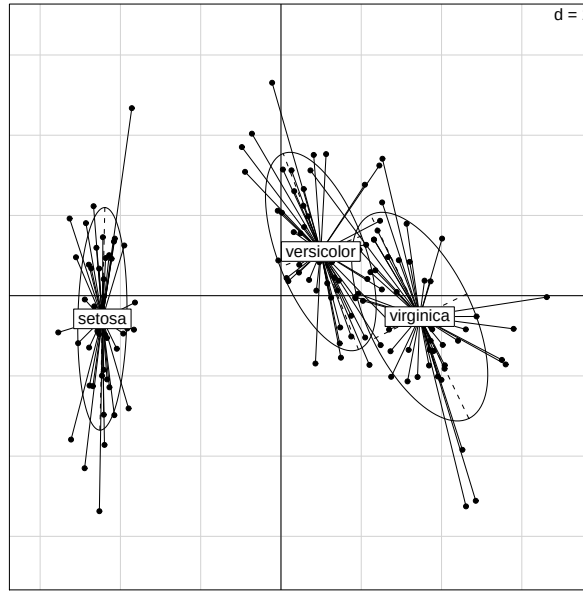

Рис. 7.7. Анализ главных компонент с помощью пакета **ade4**

```
                Comp.1 Comp.2 Comp.3 Comp.4
SS loadings       1.00   1.00   1.00   1.00
Proportion Var    0.25   0.25   0.25   0.25
Cumulative Var    0.25   0.50   0.75   1.00
```

Эта функция выводит фактически те же самые данные, что и biplot(), но немного подробнее.

Пакеты **ade4** и **vegan** реализуют множество вариаций анализа главных компонент, но самое главное то, что они содержат гораздо больше возможностей для визуализации. Например, вот как можно проанализировать те же данные по ирисам с помощью пакета **ade4**:

```
> # Установка пакета ade4
> install.packages("ade4", dependencies=TRUE)
> # Загрузка пакета
> library(ade4)
> iris.dudi < − dudi.pca(iris[,1:4], scannf=FALSE)
> s.class(iris.dudi$li, iris[,5])
```

Очевидно, что на рис. 7.7 различия между ирисами видны яснее. Более того, с помощью **ade4** можно проверить качество разрешения между классами (в данном случае видами ирисов):

```
> iris.between <- between(iris.dudi, iris[,5], scannf=FALSE)
> randtest(iris.between)
Monte-Carlo test
Call: randtest.between(xtest = iris.between)

Observation: 0.7224358

Based on 999 replicates
Simulated p-value: 0.001
Alternative hypothesis: greater

     Std.Obs  Expectation     Variance
6.868114e+01 1.374496e-02 1.064728e-04
```

Из распечатки видно, что Классы (виды ирисов) различаются хорошо (0.7 близко 1) и стабильно. Вот этот метод уже действительно близок к «настоящей статистике», нежели к типичной визуализации данных.

7.3. Классификация без обучения

Ещё одним способом снижения размерности является ординация (упорядочивание, или классификация без обучения), проводимая на основании заранее вычисленных значений сходства между всеми парами объектов (строк). В результате этой процедуры получается квадратная матрица расстояний, диагональ которой обычно состоит из нулей (расстояние между объектом и им же самим равно нулю). За десятилетия развития этой области статистики придуманы сотни коэффициентов сходства, из которых наиболее употребительными являются евклидово и квартальное (манхеттеновское). Эти коэффициенты применимы, в основном, к непрерывным переменным. Балльные и бинарные переменные в общем случае требуют других коэффициентов, но в пакете **cluster** реализована функция daisy(), которая способная распознавать тип переменной и применять соответствующие коэффициенты, а в пакете **vegan** реализовано множество дополнительных коэффициентов сходства.

Вот как можно построить матрицу сходства (лучше её всё-таки называть матрицей различий, поскольку в её ячейках стоят именно расстояния) для наших ирисов:

```
> library(cluster)
> iris.dist <- daisy(iris[,1:4], metric="manhattan")
```

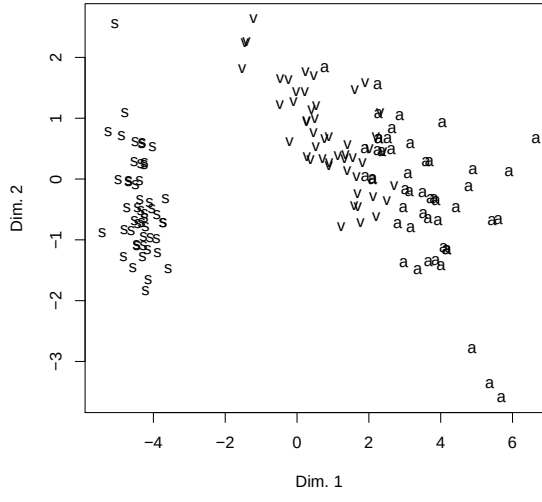

Рис. 7.8. Многомерное шкалирование

С полученной таким образом матрицей можно делать довольно многое. Одно из самых простых её применений является «многомерное шкалирование» или иначе «анализ главных координат» (это название применяют в основном для метрических вариантов этого метода).

Попробуем разобраться в сути метода «многомерного шкалирование». Допустим, в результате долгой и изнурительной последовательности действий были измерены по карте расстояние между десятком городов, результаты сохранены, а карта была неожиданно потеряна. Теперь перед исследователями стоит задача: восстановить карту взаимного расположения городов, зная только расстояния между ними. Именно такую задачу и решает многомерное шкалирование. Причём это отнюдь не метафора. Можно набрать example(cmdscale) и посмотреть на примере 21 европейского города как подобное вычисляется на самом деле. Для наших же ирисов многомерное шкалирование можно применить так:

```
> iris.c <− cmdscale(iris.dist)
> plot(iris.c[,1:2], type="n", xlab="Dim.␣1", ylab="Dim.␣2")
> text(iris.c[,1:2],
+      labels=abbreviate(iris[,5],1, method="both.sides"))
```

Как видно из рис. 7.8, результат очень похож на результат анализа главных компонент, что неудивительно, так как внутренняя структура данных (которую нам и надо найти в процессе «data mining») не изменилась. Кроме cmdscale(),

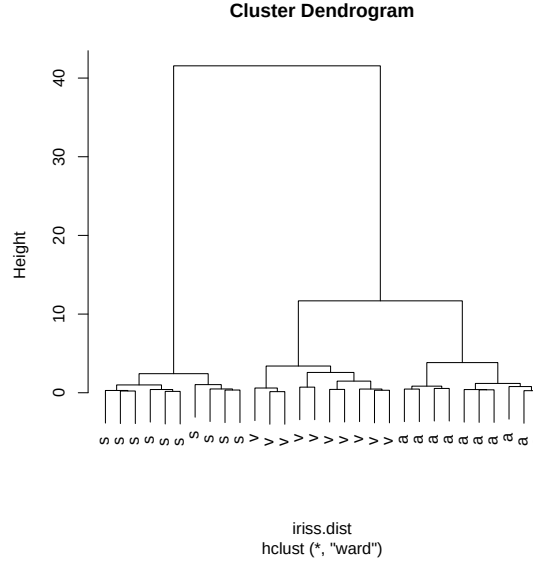

Рис. 7.9. Кластерная дендрограмма

советуем обратить внимание на непараметрический вариант этого метода, осуществляемый при помощи функции isoMDS().

7.4. Кластерный анализ

Ещё одним вариантом работы с матрицей различий является «кластерный анализ». Существует множество его разновидностей, причём наиболее употребительными являются иерархические методы, которые вместо уже привычных нам двумерных графиков производят «полуторамерные» деревья классификации, или дендрограммы.

```
> iriss <- iris[seq(1,nrow(iris),5),]
> iriss.dist <- daisy(iriss[,1:4])
> iriss.h <- hclust(iriss.dist, method="ward")
> plot(iriss.h,
+      labels=abbreviate(iriss[,5],1, method="both.sides"))
```

Для построения «дерева» была взята только каждая пятая строка данных, иначе ветки сидели бы слишком плотно. Это, кстати, недостаток иерархической кластеризации как метода визуализации данных. Метод Уорда («ward») даёт очень хорошо очерченные кластеры при условии, естественно, если их удаётся найти,

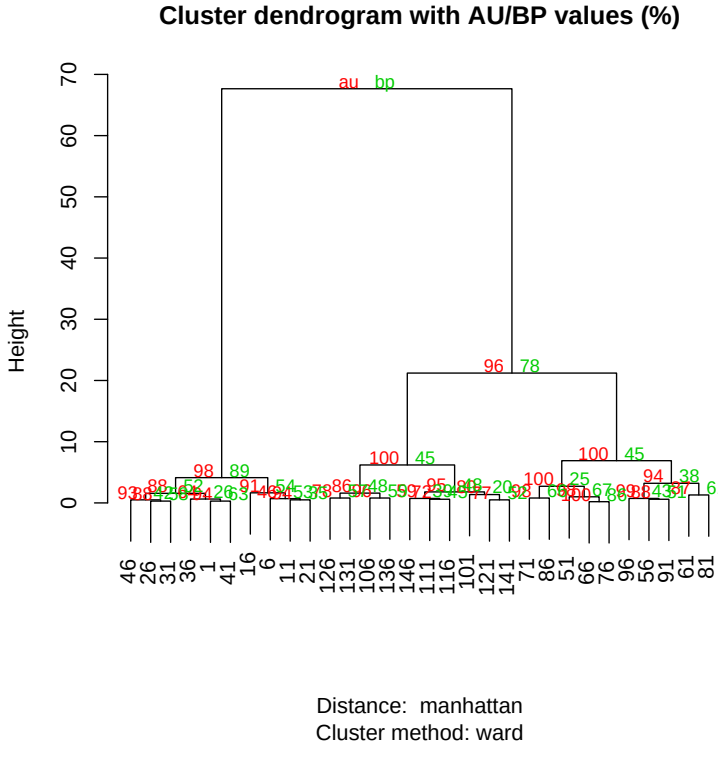

Рис. 7.10. Пакет **pvclust**

поэтому неудивительно, что в нашем случае (рис. 7.9) все три вида разделились. При этом отлично видно, что виды на «v» (versicilor и virginica) разделяются на более низком уровне (`Height ~ 12`), то есть сходство между ними сильнее, чем каждого из них с третьим видом.

Кластерный анализ этого типа весьма привлекателен тем, что даёт готовую классификацию. Однако не сто́ит забывать, что это «всего лишь визуализация». Насколько «хороши» получившиеся кластеры, проверить порой непросто, хотя и здесь существует множество методов. Один из них, так называемый «silhouette plot» реализован в пакете **cluster**. Для того чтобы увидеть этот метод в действии достаточно набрать example(agnes). Ещё один, очень «модный» сейчас метод, основанный на boostrap-репликации, реализован в пакете **pvclust**:

```
> install.packages("pvclust", dependencies=TRUE)
> library(pvclust)
> iriss.pv < − pvclust(t(iriss[,1:4]),
```

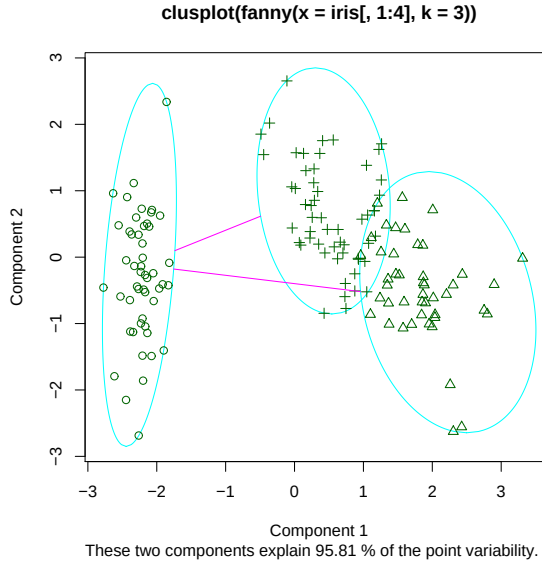

Рис. 7.11. Пакет **cluster**

```
+     method.dist="manhattan", method.hclust="ward", nboot=100)
> plot(iriss.pv, print.num=FALSE)
```

Красным цветом на графике 7.10 печатаются p-values, связанные с устойчивостью кластеров в процессе репликации исходных данных. Значения, близкие к 100, считаются «хорошими». В нашем случае мы видим как раз «хорошую» устойчивость получившихся основных трёх кластеров (разных видов ирисов), и неплохую устойчивость кластера, состоящего из двух видов на "v".

Кроме иерархических методов кластеризации, существуют и другие. Из них наиболее интересны так называемые fuzzy-методы, основанные на идее того, что каждый объект может принадлежать к нескольким кластерам сразу, но с разной «силой». Вот как реализуется такой метод в пакете **cluster**:

```
> library(cluster)
> iris.f <- fanny(iris[,1:4], 3)
> plot(iris.f, which=1)
> head(data.frame(sp=iris[,5], iris.f$membership))
      sp        X1         X2         X3
1 setosa 0.9142273 0.03603116 0.04974153
2 setosa 0.8594576 0.05854637 0.08199602
3 setosa 0.8700857 0.05463714 0.07527719
4 setosa 0.8426296 0.06555926 0.09181118
```

```
5 setosa 0.9044503 0.04025288 0.05529687
6 setosa 0.7680227 0.09717445 0.13480286
```

Подобные рис. 7.11 графики мы уже неоднократно видели. В нём нет ничего принципиально нового. А вот текстовый вывод интереснее. Для каждой строчки указан «membership» или показатель «силы», с которой данный элемент «притягивается» к каждому из трёх кластеров. Как видно, шестая особь, несмотря на то, что почти наверняка принадлежит к первому кластеру, тяготеет также и к третьему. Недостатком этого метода является необходимость заранее указывать количество получающихся кластеров.

Подобный метод реализован также и в пакете **e1071**. Функция называется cmeans(), но в этом случае вместо количества кластеров можно указать предполагаемые центры, вокруг которых будут группироваться элементы.

7.5. Классификация с обучением

Теперь обратимся к методам, которые лишь частично могут называться «визуализацией». В зарубежной литературе именно их принято называть «методами классификации». Для того, чтобы работать с этими методами, надо освоить технику «обучения». Как правило, выбирается часть данных с известной групповой принадлежностью. На основании анализа этой части, называемой «тренировочной выборкой», строится гипотеза о том, как должны распределяться по группам остальные, не классифицированные данные. Как правило, можно узнать, насколько хорошо работает та или иная гипотеза. Кроме того, методы классификации с обучением можно с успехом применять и для других целей, например, для выяснения важности признаков.

Один из самых простых методов в этой группе — это «линейный дискриминантный анализ». Его основной идеей является создание функций, которые на основании линейных комбинаций значений признаков (это и есть классификационная гипотеза) «сообщают», куда нужно отнести данную особь. Воспользуемся этим методом для выяснения структуры данных ирисов:

```
> iris.train <- iris[seq(1,nrow(iris),5),]
> iris.unknown <- iris[-seq(1,nrow(iris),5),]
> library(lda.cv)
> iris.lda <- lda(scale(iris.train[,1:4]), iris.train[,5])
> iris.ldap <- predict(iris.lda, iris.unknown[,1:4])$class
> table(iris.ldap, iris.unknown[,5])

iris.ldap    setosa versicolor virginica
  setosa          0          0         0
  versicolor     34          0         0
  virginica       6         40        40
```

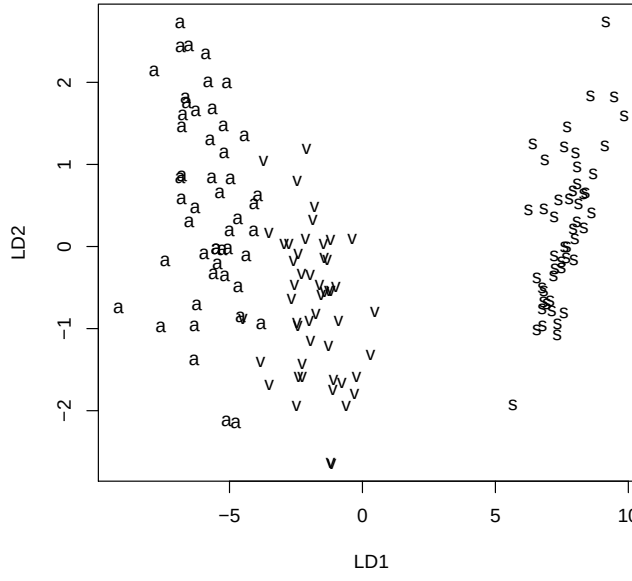

Рис. 7.12. Линейный дискриминантный анализ

На выходе получился довольно забавный результат: наша тренировочная выборка привела к построению гипотезы, по которой все virginica и versicolor (а также часть setosa) попали в одну группу. Это говорит не только о близости видов на "v но также и о недостаточной «тренировке».

Линейный дискриминантный анализ можно использовать и для визуализации данных, например, так:

```
> iris.lda2 <− lda(scale(iris[,1:4]), iris[,5])
> iris.ldap2 <− predict(iris.lda2, dimen=2)$x
> plot(iris.ldap2, type="n", xlab="LD1", ylab="LD2")
> text(iris.ldap2, labels=abbreviate(iris[,5], 1,
+      method="both.sides"))
```

Здесь мы в качестве тренировочной выборки использовалась вся выборка целиком. Как видно на рис. 7.12, виды на "v"разделились лучше, чем в предыдущих примерах, поскольку дискриминантный анализ склонен переоценивать различия между группами. Это свойство, а также жёсткая параметричность метода привели к тому, что в настоящее время этот тип анализа используется всё реже.

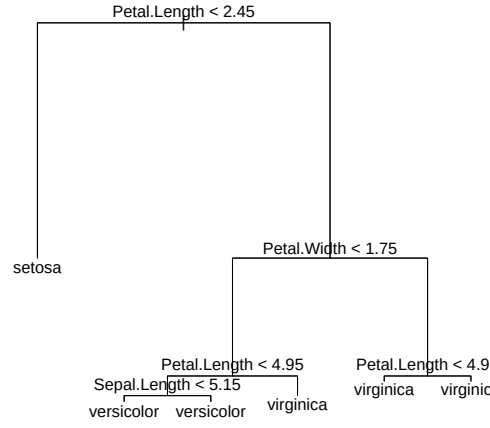

Рис. 7.13. Дерево решений

На замену дискриминантному анализу приходит множество других методов со схожим принципом работы. Одними из самых оригинальных алгоритмов являются так называемые «деревья классификации», или «деревья решений». Они позволяют выяснить, какие именно показатели могут быть использованы для разделения объектов на заранее заданные группы. В результате строится ключ, в котором на каждой ступени объекты делятся на две группы:

```
> install.packages("tree")
> library(tree)
> iris.tree <- tree(iris[,5] ~ ., iris[,-5])
> plot(iris.tree)
> text(iris.tree)
```

Здесь опять была использована в качестве тренировочной вся выборка (чтобы посмотреть на пример частичной тренировочной выборки, можно набрать ?predict.tree). В результате получился график (рис. 7.13), очень похожий на так называемые «определительные таблицы», по которым биологи определяют виды организмов. Из графика рис. 7.13 легко понять, что к setosa относятся все ирисы, у которых длина лепестков меньше 2.45 (читать график надо влево), а из оставшихся ирисов те, у которых ширина лепестков меньше 1.75, а длина меньше 4.95, относятся к versicolor. Все остальные ирисы относятся к virginica.

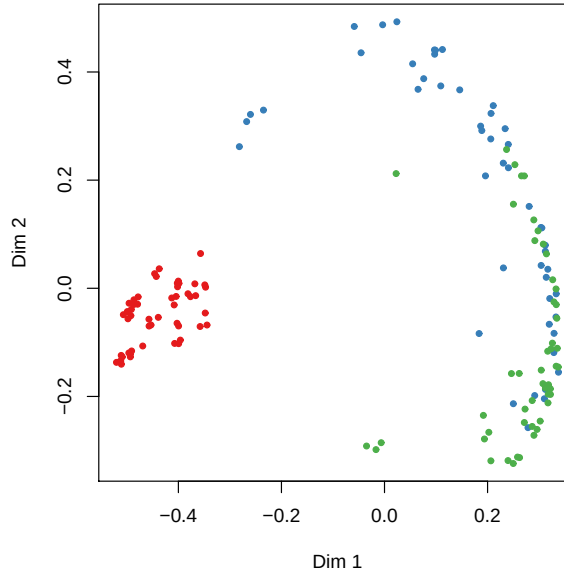

Рис. 7.14. Метод Random Forest

► Деревья классификации реализованы и в пакете **rpart**. Создатели **R** рекомендуют использовать именно его вместо **tree**.

Есть еще один, набирающий сейчас всё большую популярность, метод, идеологически близкий к деревьям классификации, который называется «Random Forest», поскольку основой метода является производство большого количества классификационных «деревьев».

```
> library(randomForest)
> set.seed(17)
> iris.rf <- randomForest(iris.train[,5] ~ ., data=iris.train[,1:4])
> iris.rfp <- predict(iris.rf, iris.unknown[,1:4])
> table(iris.rfp, iris.unknown[,5])

iris.rfp     setosa versicolor virginica
  setosa         40          0         0
  versicolor      0         39         9
  virginica       0          1        31
```

Здесь очень заметна значительно более высокая эффективность этого метода по сравнению с линейным дискриминантным анализом. Кроме того, Random Forest позволяет выяснить значимость (importance) каждого признака, а также дистанции между всеми объектами тренировочной выборки (proximity), которые затем можно использовать для кластеризации или многомерного шкалирования. Наконец, этот метод позволяет «чистую визуализацию» данных, то есть он может работать как метод классификации без обучения:

```
> set.seed(17)
> iris.urf <- randomForest(iris[,-5])
> MDSplot(iris.urf, iris[,5])
```

Великое множество методов «data mining», разумеется, не реально охватить в небольшой статье. Однако и нельзя не упомянуть ещё об одном современном методе, основанном на идее вычисления параметров гиперплоскости, разделяющей различные группы в многомерном пространстве признаков — «Support Vector Machines».

```
> library(e1071)
> iris.svm <- svm(Species ~ ., data = iris.train)
> iris.svmp <- predict(iris.svm, iris[,1:4])
> table(iris.svmp, iris[,5])

iris.svmp    setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         50        14
  virginica       0          0        36
```

Этот метод изначально разрабатывался для случая бинарной классификации, однако в **R** его можно использовать, и для большего числа групп. Работает он эффективнее дискриминантного анализа, хотя и не так точно, как Random Forest.

В заключении хотелось бы отметить, что настоящая статья ни в коем случае не заменяет больших книг и справочников, написанных по теме «data mining». Здесь мы ставили целью лишь дать примерное представление о многообразии методов анализа многомерных данных, и наиболее распространенных способах решения основных проблем, возникающих при поиске порядка в больших массивах информации.