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 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

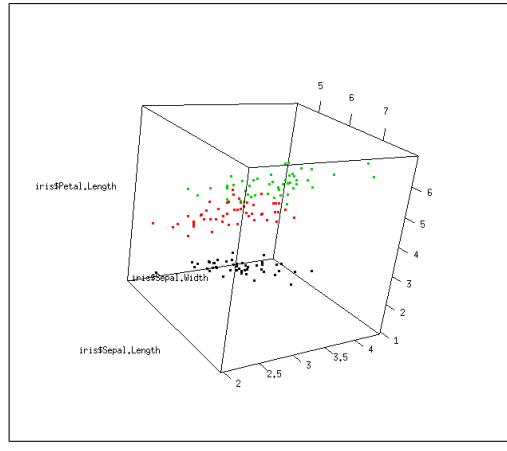

Рис. 7.1. Отображение многомерных данных с помощью пакета **RGL**.

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)
```

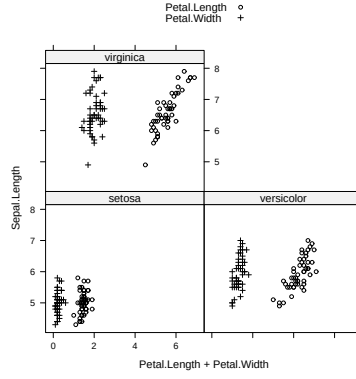

Рис. 7.2. Отображение многомерных данных с помощью пакета **lattice**.

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)
```

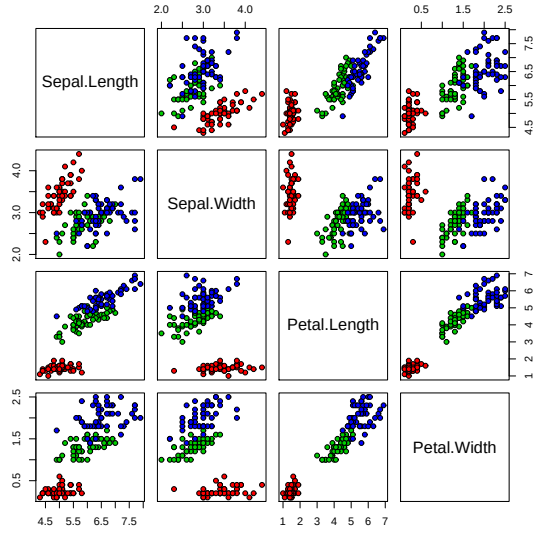

Рис. 7.3. Отображение многомерных данных с помощью пакета **pairs**.

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.

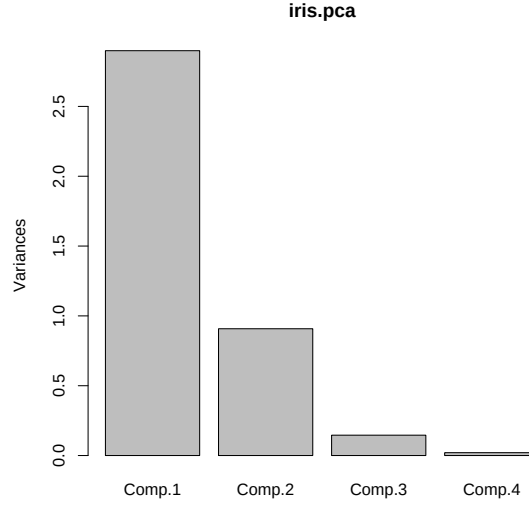

Рис. 7.4. Анализ главных компонент (служебный график)

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)
```

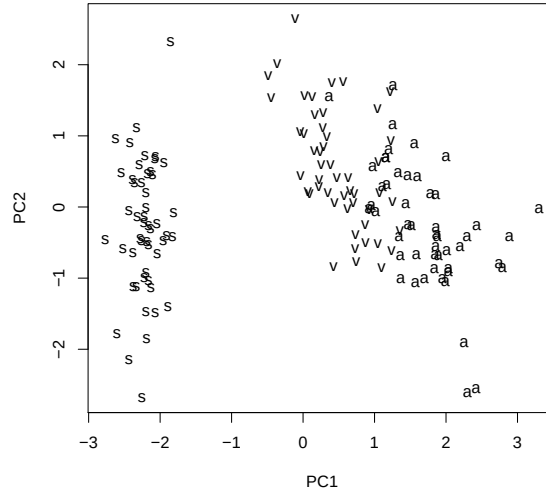

Рис. 7.5. Анализ главных компонент

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

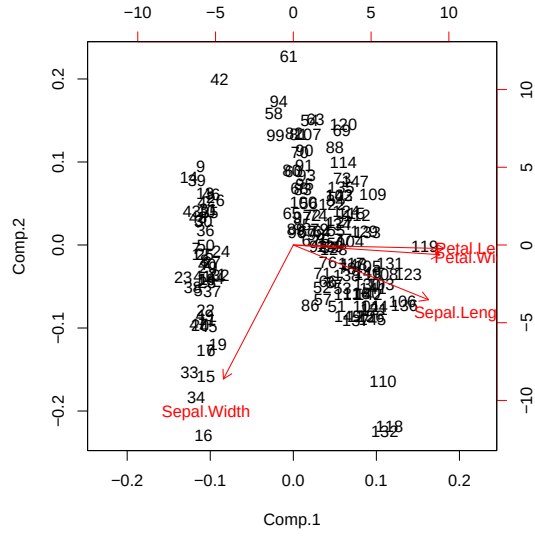

Рис. 7.6. Действие функции biplot

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
```

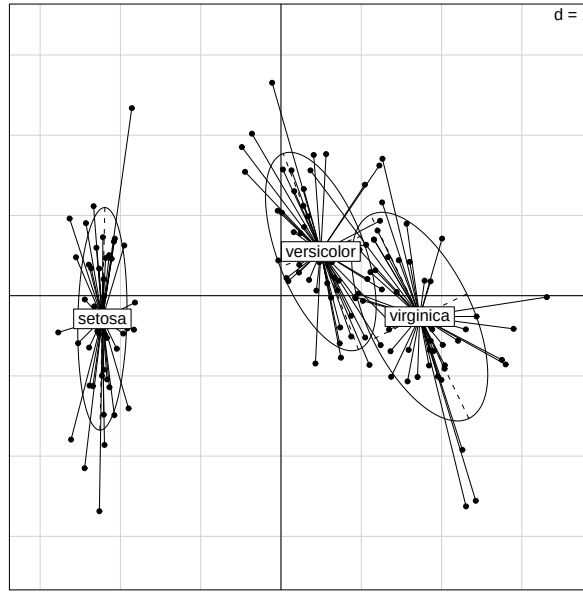

Рис. 7.7. Анализ главных компонент с помощью пакета **ade4**

```
                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")
```

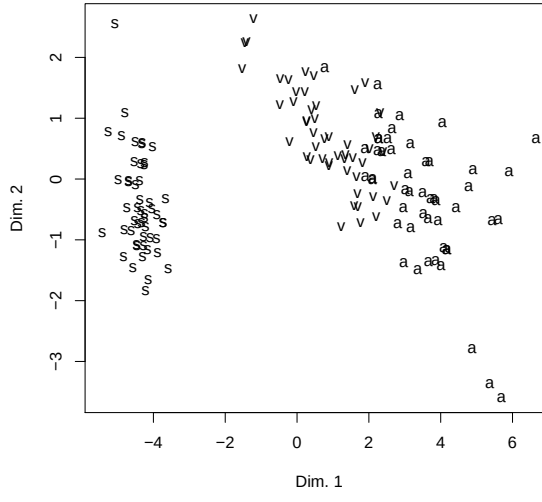

Рис. 7.8. Многомерное шкалирование

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(),

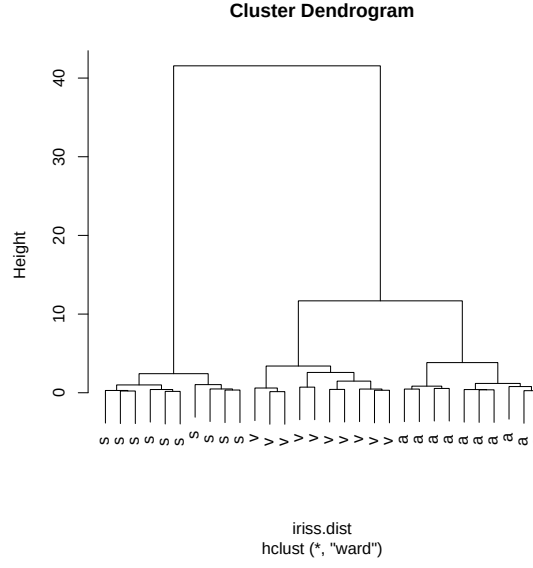

Рис. 7.9. Кластерная дендрограмма

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,

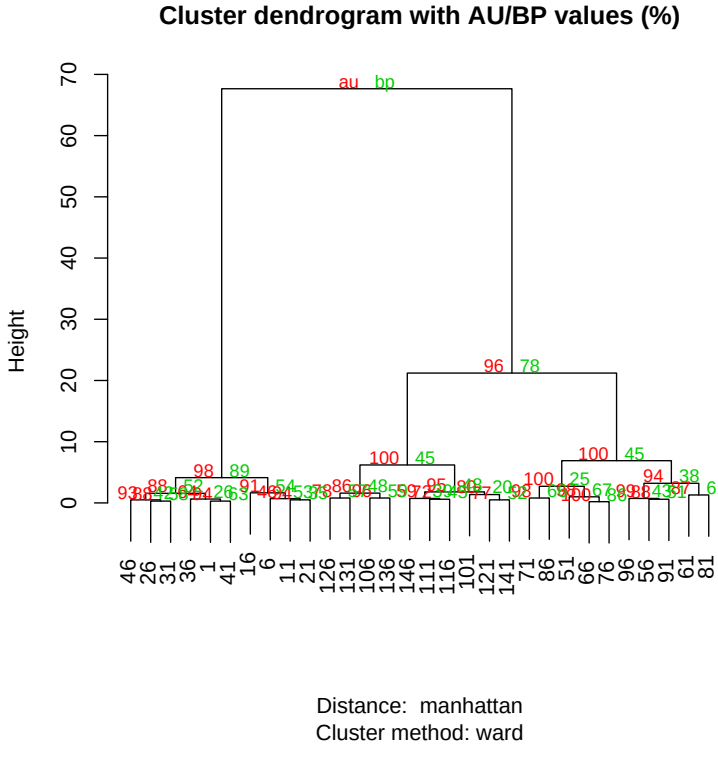

Рис. 7.10. Пакет **pvclust**

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]),
```

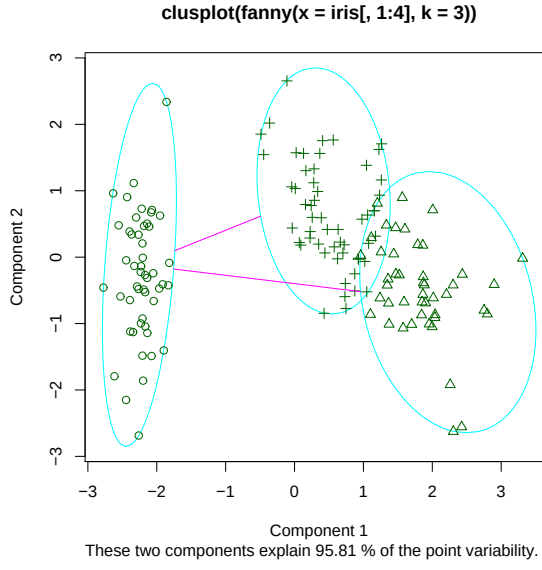

Рис. 7.11. Пакет **cluster**

```
+     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
```

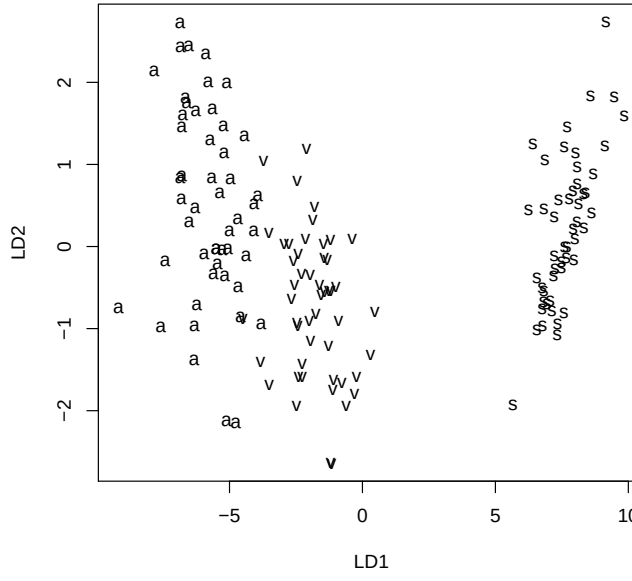

Рис. 7.12. Линейный дискриминантный анализ

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.

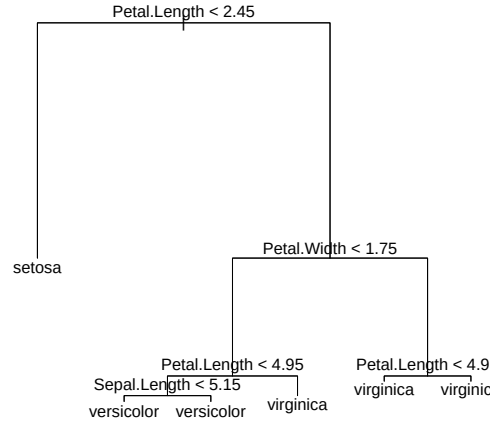

Рис. 7.13. Дерево решений

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.

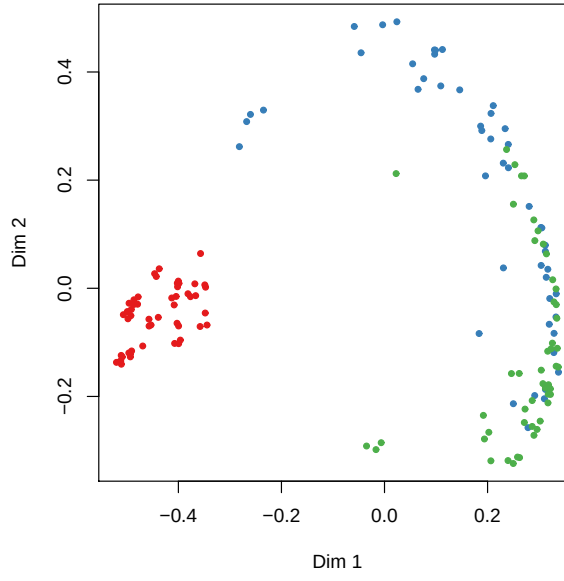

Рис. 7.14. Метод Random Forest

► 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.