R, Quick start to data analysis

Alex Shlemov Anton Korobeynikov

15 декабря 2014 г.

Содержание

1 Справка, workspaces, запуск скриптов, пакеты

1.1 Справка

```
help(package = package_name) # Справка по пакету
> help(package = lattice)

?function_name # Справка по функции
> ?ls

?"keyword" # Справка по ключевому слову
> ?"for"
> ?"+"
> ?"["
```

```
> ?"[[<-"
??pattern # Поиск по справке
> ??glm
apropos("pattern") # Возвращает найденные имена функций, подходящие под
    шаблон
> apropos("GLM")
```

1.2 Переменные, рабочие пространства (workspaces), история команд, выход

```
ls() # Возвращает вектор из имен переменных в текущем scope, если запущен в
    терминале, то возвращает имена переменных из рабочего workspace
ls(all.values = TRUE) # Возвращает ВСЕ имена переменных текущего scope
    (включая начинающиеся с .)

rm(varname) # Удаляет переменную. Имя без кавычек
rm(list = ls(all.values = TRUE)) # При вызове из терминала -- чистит
    workspace

save.image(file = "workspace_file_name.rda") # сохраняет workspace (проще
    говоря, все переменные) в файл
load.image(file = "workspace_file_name.rda") # Загружает workspace из файла

history(max.show = Inf) # Показывает историю команд
savehistory(file = "history_file_name.R") # Сохраняет историю команд в файл

q() # Выход из R
q("no") # Выход из R без сохранения workspace (предпочтительнее)
```

1.3 Запуск скриптов

```
source("script_file_name.R") # Выполняет скрипт из файла
```

Также есть утилита `Rscript`, которая позволяет выполнить R-файл прямо из командной строки:

```
> Rscript script.R
```

Можно включить ее в shabang и сделать скрипт исполняемым файлом (в Unix):

script.R

```
#!/usr/bin/Rscript

args <- commandArgs(TRUE) # Получить аргументы командной строки в виде
    вектора строк
print(args)
```

после чего:

```
> chmod +x script.R
> ./script.R just command line args 3 14 15
[1] "just"    "command" "line"    "args"    "3"       "14"      "15"
```

Если есть необходимость в детальном разборе аргументов командной строки, не нужно писать свой ~~велосипед~~ парсер, есть пакеты `getopt` и `optparse`.

1.4 Пакеты

```
library("package_name") # Подключает установленный пакет.
# Кавычки можно опустить:
> library(lattice)

install.package("package_name") # Устанавливает пакет с зеркала CRAN
> install.packages("latticeExtra")
```

При первом запуске в сессии `R` предложит выбрать зеркало CRAN, достаточно выбрать "Cloud" (первое в списке). Обратите внимание, что в Unix пакеты скачиваются в виде исходников и собираются у Вас на машине, поэтому должен быть установлен компилятор C/C++/fortran и необходимые библиотеки (причем девелоперские версии, в пакетном менеджере они обычно имеют суффикс "-dev", например "libfftw3-dev"). Под Windows пакеты скачиваются уже собранными.

При необходимости, можно установить сторонние пакеты из исходников. Для этого удобно пользоваться пакетом `devtools`:

```
install.packages("devtools")
library(devtools)
install_github("asl/rssa")
# Аналогично:
install_git(...); install_bitbucket(...); install_url(...); install_local
   (...)
```

Здесь пакет в любом случае будет собираться из исходников, под Windows нужно устанавливать и настраивать весь toolchain (msys + mingw + девелоперские либы). Под Unix могут понадобится некоторые стандартные утилиты типа `curl` (как правило, они уже установлены).

2 Вектора, матрицы, массивы

2.1 Вектора, основные операции

Начнем с того, что в `R` нет "скалярных" значений, любое скалярное значение (число, строка) это вектор длины 1. Вектора бывают следующих типов: `numeric, complex, logical, character`, т.е. числовые, комплексные, булевские и строковые. Числовые вектора делятся на `integer` и `double`, но это деление исключительно внутреннее — при делении или выходе из диапазона целые числа автоматически приводятся к вещественному типу.

Создание и простейшая работа с векторами:

```
> v <- 1:10
> print(v)
[1]  1  2  3  4  5  6  7  8  9 10
> 10:1
[1] 10  9  8  7  6  5  4  3  2  1
> seq(1, 10, 2)
[1] 1 3 5 7 9
> seq(from = 10, by = 5, length.out = 6)
```

```
[1] 10 15 20 25 30 35

# Создание “пустых” векторов
> v <- numeric(10)
> v
[1] 0 0 0 0 0 0 0 0 0 0
> b <- logical(10)
> b
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> cplx <- complex(10)
> cplx
[1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
> ch <- character(10)
> ch
[1] "" "" "" "" "" "" "" "" "" ""

# Прочитать элемент
> v[2]
[1] 0
> ch[3]
[1] ""
> b[4]
[1] FALSE
> cplx[5]
[1] 0+0i

# Записать элемент
> v[6] <- 42
> ch[7] <- "Hello"
> b[8] <- TRUE
> i <- 9
> 4i + 3 -> cplx[i]

# Повторения
> rep(1:3, 5) # Последовательная склейка
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
> rep(1:3, each = 5) # И повтор каждого элемента
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3

# Конкатенация (склейка)
> c(1:5, 5:1, 3:4)
[1] 1 2 3 4 5 5 4 3 2 1 3 4
```

Немного служебных операций. Вывод:

```
> print(ch)
[1] ""      ""      ""      ""      ""      ""      "Hello" ""      ""
[10] ""
```

Если Вы работаете в командной сессии, то выводится результат каждой выполненной

команды. Но если Вы проводите какие-то действия в цикле, в функции, в вызываемом по `source()` или `Rscript` скрипте, то желаемый вывод необходимо делать явно.

Кстати говоря, если Вы работаете в командной строке, то переменная `.Last.value` всегда содержит результат последней команды:

```
> 2 + 2
[1] 4
> print(.Last.value)
[1] 4
```

Summary:

```
> summary(1:10)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
```

Вообще `summary()` (как, кстати, и `print()`) — это полиморфные функции, для каждого типа объекта они определены по-своему. Для числовых векторов `summary()` выводит квантили и среднее. Незамысловато, но бывает полезно.

Длина вектора:

```
> length(v)
[1] 10
> length(v) <- 5
> v
[1] 0 0 0 0 0
> length(v) <- 10
> v
[1]  0  0  0  0  0 NA NA NA NA NA
```

Функция `length()` работает и на присваивание. При попытке увеличить длину вектора новые элементы получают значение `NA`, т.е. пропущенное значение.

Тип вектора:

```
> mode(v) # Логический тип (mode)
[1] "numeric"
> storage.mode(v) # Хранимый тип. Нужен редко, в основном, если хочется
   передать указатель на объект “наружу”
[1] "double"
```

Обе функции работают на присваивание, изменяя тип объекта.

Также можно совершить приведение типа с помощью функций `as.whatever()`:

```
> as.character(10)
[1] "10"
> as.logical(10)
[1] TRUE
> as.numeric("33.5")
[1] 33.5
> as.integer("33.5")
[1] 33
> as.integer(33.5)
[1] 33
```

Все стандартные операции с векторами векторизованы, т.е. выполняются поэлементно:

```
> 1:10 + 10:1
[1] 11 11 11 11 11 11 11 11 11 11
> sin(1:10)
[1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243 -0.2794155
[7]  0.6569866  0.9893582  0.4121185 -0.5440211
```

При этом если в бинарной операции встречаются вектора неодинаковой длины, то используются так называемое переписывание (recycling), вектор меньшей длины автоматически повторяется нужное число раз:

```
> 1:10 + 1:5
[1]  2  4  6  8 10  7  9 11 13 15
```

При этом, если длина меньшего вектора не является делителем длины большей, будет выведено предупреждение (warning):

```
> 1:10 + 1:3
[1]  2  4  6  5  7  9  8 10 12 11
Warning message:
In 1:10 + 1:3 :
 longer object length is not a multiple of shorter object length
```

Обычно меньший вектор имеет длину 1 и такой проблемы не возникает:

```
> (1:10)^2
[1]   1   4   9  16  25  36  49  64  81 100
```

Кстати, степень имеет более высокий приоритет, чем ::

```
> 1:3^2
[1] 1 2 3 4 5 6 7 8 9
```

Полезные векторизованные функции:

```
a + b, a - b, a * b, a / b # 4 действия арифметики
a ^ b # степень
a %/% b, a %% b # целочисленное деление и взятие остатка

exp(x), log(x) # экспонента и логарифм

abs(x) # Модуль

Re(z), Im(z), Conj(z), Mod(z), Arg(z) # Вещественная и мнимая часть,
    комплексное сопряжение, модуль и аргумент

cos(x), sin(x), tan(x), acos(x), asin(x), atan(x), atan2(y, x) #
    Тригонометрия

x == y, x != y, x > y, x >= y, etc # поэлементные сравнения
> 1:10 > 5
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
x & y, x | y, xor(x, y) # булевские поэлементные операции
x && y, x || y # булевские операции для векторов длины 1, вычисляемые по
    короткой схеме
```

Агрегирующие функции. Наряду с поэлементной векторизацией (любители функционального программирования назвали бы ее “map”) есть функции, сопоставляющие вектору единичное значение (любители ФП назвали бы это “reduce”). Вот примеры таких функций:

```
sum(x), prod(x) # Сумма и произведение всех элементов
max(x), min(x), which.max(), which.min() # Максимум-минимум и индекс
    максимального и минимального элемента

mean(), sd(), cov(), cor(), median(), mad(), quantile() # Статистические
    функции

all(x), any(x) # Логические функции, возвращают TRUE, если все (или хотя бы
    один) из элементов вектора истина
```

2.2 Вектора, доступ к элементам (subscripting, индексная техника)

Доступ к элементам вектора осуществляется с помощью оператора “[” (“subscript”). Доступ работает как на чтение, так и на запись:

```
x[?]
x[?] <- y
```

Чтение возвращает подвектор (возможно, что пустой). При записи подвектор перезаписывается значениями из вектора, стоящего в правой части (y). Если длины перезаписываемого подвектора и правой части не совпадают, применяется переписывание (если количество заменяемых значений не делится на количество новых, то выводится предупреждение).

Что может стоять внутри “[ ]”?

2.2.1 Числовой вектор индексов

Все нецелые значения приводятся к целым (отбрасывается дробная часть). Нули отбрасываются. Для положительных индексов возвращаются соответсвующие элементы (нумерация от единицы!!!):

```
> v <- c("a", "b", "c", "d", "e", "f", "g", "h")
> v[c(1, 3, 5.9)]
[1] "a" "c" "e"
> v[c(1, 3, 5.9)] <- "X"
> v
[1] "X" "b" "X" "d" "X" "f" "g" "h"
```

Для отрицательных возвращаются все элементы, кроме названных:

```
> v[-c(2, 4, 7.7)]
[1] "X" "X" "X" "f" "h"
```

```
> v[-c(2, 4, 7.7)] <- Y
Error: object 'Y' not found
> v[-c(2, 4, 7.7)] <- "Y"
> v
[1] "Y" "b" "Y" "d" "Y" "Y" "g" "Y"
```

Смешивать отицательные и положительные индексы нельзя. На чтение положительные индексы можно дублировать:

```
> v
[1] "Y" "b" "Y" "d" "Y" "Y" "g" "Y"
> v[c(1, 1, 1, 2)]
[1] "Y" "Y" "Y" "b"
```

На запись тоже можно, но в таком случае элемент с повторенным индексом будет перезаписан несколько раз и в итоге в нем окажется последний записанный элемент:[1]

```
> v[c(1, 1, 1)] <- c("X", "Y", "Z")
> v
[1] "Z" "b" "c"
```

2.2.2 Логический вектор-маска

Выбираются элементы, соответствующие TRUE. Если вектор недостаточной длины, используется переписывание (если длина маски не делит длину вектора, то выведется соответствующее предупреждение). Если вектор-маска больше длины вектора, то вектор удлиняется до необходимой длины и дополняется пропусками (NA).

```
> v <- 1:10
> v[c(TRUE, FALSE)] # Выбрать четные элементы
[1] 1 3 5 7 9
> v[c(TRUE, FALSE)] <- 42 # Заменить четные элементы
> v
[1] 42  2 42  4 42  6 42  8 42 10
```

В основном, в качестве логической маски используются выражения-“запросы”:

```
> v[v > 6] <- 0 # Заменить элементы > 6
> v
[1] 0 2 0 4 0 6 0 0 0 0
```

Тут нет никакой магии — v > 6 возвращает логический вектор.

2.2.3 Строковый вектор имен

Для того, чтобы обращаться к элементам вектора по именам, необходимо эти имена назначить. У каждого вектора есть возможность установить атрибут names — строковый вектор такой же длины, как и сам вектор:

```
> v <- 1:3
> names(v) <- c("a", "b", "c")
> v
```

[1] Лично я считаю, что использовать повторные индексы на запись — очень скверная идея.

```
a b c
1 2 3
> names(v)
[1] "a" "b" "c"
```

Вектор стал именованным. Теперь если передать в качестве индекса строковый вектор, будут выбраны соответствующие элементы:

```
> v[c("a", "b")]
a b
1 2
> v[c("a", "b")] <- 42
> v
 a  b  c
42 42  3
```

2.3 Матрицы и массивы

2.3.1 Создание и размерность матриц

Матрица создается с помощью одноименной команды:

```
> m <- matrix(1:9, 3, 3)
> m
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Матрицы в `R` представляют собой вектор (с разверткой FORTRAN-style, т.е. по столбцам) со специальным атрибутом размерности:

```
> dim(m)
[1] 3 3
> length(m)
[1] 9
> dim(m) <- c(1, 9)
> m
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    2    3    4    5    6    7    8    9
```

Как видите, атрибут доступен на запись, единственное, необходимо, чтобы `prod(dim(x)) = length(x)`.

Также есть функции `nrow()` и `ncol()`, возвращают число строк и столбцов соответственно.

2.3.2 Операции с матрицами

Так как матрицы являются векторами, для них можно делать те же операции, что и для векторов; при этом размерность будет сохраняться:

```
> m <- matrix(1:9, 3, 3)
> sin(m)
```

```
          [,1]       [,2]      [,3]
[1,] 0.8414710 -0.7568025 0.6569866
[2,] 0.9092974 -0.9589243 0.9893582
[3,] 0.1411200 -0.2794155 0.4121185
> m + m
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
> m ^ 2
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
> m * 2
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
> m * m
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
> m > 10
      [,1]  [,2]  [,3]
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE FALSE
[3,] FALSE FALSE FALSE
> m > 5
      [,1]  [,2] [,3]
[1,] FALSE FALSE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE  TRUE TRUE
```

Обратите внимание, что произведение матриц — поэлеметное. Если мы хотим получить обычное операторное произведение, следует использовать `%*%`:

```
> m %*% m
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

Вектор без атрибута размерности считается вектор-столбцом, но при умножении вектора на матрицу слева вектор автоматически транспонируется:

```
> m
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

```
[3,]    3    6    9
> m %*% 1:3
     [,1]
[1,]   30
[2,]   36
[3,]   42
> 1:3 %*% m
     [,1] [,2] [,3]
[1,]   14   32   50
```

Обратите внимание, что умножение "$*$" это поэлеметное умножение каждого столбца на вектор:

```
> m * 1:3
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    4   10   16
[3,]    9   18   27
```

2.3.3 Функции для работы с матрицами

:

```
solve(m) # обратная матрица
solve(m, y) # m^{-1}y, но вычисляется устойчивее
t(m) # транспонирование

qr(m), eigen(m), svd(m), chol(m) # классические матричные разложения (QR,
   EVD, SVD и разложение Холецкого)

crossprod(x, y) # x^T y, но вычисляется немного быстрее
tcrossprod(x, y) # xy^T, аналогично
crossprod(x), tcrossprod(x) # умножение саму на себя: x^T x и xx^T

diag(m) # для матрицы, возвращает вектор главной диагонали, при этом
   доступна на запись
> m <- matrix(1:9, 3, 3)
> diag(m)
[1] 1 5 9
> diag(m) <- -diag(m)
> m
     [,1] [,2] [,3]
[1,]   -1    4    7
[2,]    2   -5    8
[3,]    3    6   -9
diag(n) # для числа -- возвращает единичную матрицу порядка n
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
```

```
[3,]    0    0    1
```

Слейка матриц:

```
cbind(a, b, c, ...) # Склейка матриц по столбцам: [a : b : c : ...]
rbind(a, b, c, ...) # Склейка матриц по строкам (вертикально)
```

Если вектор (не матрицу) передать на вход `cbind()`, то он будет рассматриваться как столбец, а если `rbind()` — то как строка. При этом для векторов и матриц работают правила переписывания.

Сумма и среднее по строкам и столбцам:

```
> m <- matrix(1:9, 3, 3)
> rowMeans(m)
[1] 4 5 6
> colMeans(m)
[1] 2 5 8
> rowSums(m)
[1] 12 15 18
> colSums(m)
[1]  6 15 24
```

2.3.4 Многомерные массивы

Также, кроме матриц присутствуют и многомерные массивы (тензоры) `array()`:

```
> a <- array(1:8, dim = c(2, 2, 2))
> a
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

2.4 Матрицы и массивы, доступ к элементам

Обсудим обращение к элементам матриц и многомерных массивов. Аналогично векторам, обращение возможно как на чтение, так и на запись

2.4.1 Обращение как к вектору

И матрица, и массив являются вектором, следовательно, для них работают те же методы индексирования, что и для векторов, при этом. напоминаю, матрица укладывается в вектор по столбцам. На практике, пожалуй, из этого может быть полезна только техника “логических запросов” типа:

```
m[m > 10]
m[m < 0] <- 0
```

2.4.2 Обращение к декартовому произведению измерений

Для обращения к матрице можно использовать двухиндексную технику (а для обращения к массивам — r-индексную, где r — количество измерений):

```
m[i, j]
a[i, j, k]
```

где `i, j, k` могут быть числовыми, логическими или строковыми векторами. Результатом будет подмассив той же структуры (подвыборка произойдет независимо по всем измерениям).

```
> m <- matrix(1:9, 3, 3)
> m
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> m[c(TRUE, FALSE, TRUE), -1] # Выбрать 1 и 3 строки и отбросить 1 столбец
     [,1] [,2]
[1,]    4    7
[2,]    6    9
```

Чтобы иметь возможность обращаться к строкам и столбцам матрицы по именам, нужно задать атрибуты `colnames` и `rownames` (а в случае массива — атрибут `dimnames`):

```
> m <- matrix(1:9, 3, 3)
> rownames(m) <- c("a", "b", "c")
> colnames(m) <- c("x", "y", "z")
> m[c("a", "c"), c("y", "y", "x")]
  y y x
a 4 4 1
c 6 6 3

> a <- array(1:8, dim = c(2, 2, 2))
> dimnames(a) <- list(c("a", "b"), c("i", "j"), c("x", "y"))
> a
, , x

  i j
a 1 3
b 2 4

, , y

  i j
a 5 7
b 6 8
```

```
> a["a", "j", "y"]
[1] 7
```

Нужно отметить две тонкости. Во-первых, один или несколько индексов можно опускать, это будет означать выбор всего диапазона. Во-вторых, если в результате выбора полученный массив будет иметь меньшую размерность, чем исходный (например, выбираем строку из матрицы), то вырожденные измерения автоматически "схлопнутся" (drop):

```
m <- matrix(1:9, 3, 3)
m

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

m[1, ]

## [1] 1 4 7

m[, 1]

## [1] 1 2 3
```

В большинстве случаев это удобно: когда мы извлекаем строку или столбец, более приятно получать вектор, а не длинную матрицу. Но иногда требуется, чтобы подмассив имел строго такую же размерность, как и исходный массив. Чтобы избежать схлопывания и получить массив той же размерности нужно явно указать:

```
m <- matrix(1:9, 3, 3)
m[1, , drop = FALSE]

##      [,1] [,2] [,3]
## [1,]    1    4    7

m[, 1, drop = FALSE]

##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

2.4.3 Обращение по многомерному индексу

Можно передать в `[ ]` матрицу из r столбцов и n строк, где r — число измерений (2 для матрицы). В результате каждая строка будет рассматриваться как набор координат выбираемого элемента и результатом будет вектор длины n:

```
> m <- matrix(1:9, 3, 3)
> m[cbind(1:ncol(m), ncol(m):1)] # Антидиагональ
[1] 7 5 3
```

3 Списки

Список — это вектор, который может хранить элементы различных типов. В отличие от Python, нет возможности создать рекурсивный список (так как копирование всегда происходит по значению).

3.1 Создание списка, склейка, повторение

```
l <- list(a = 1, b = "string", f = q)  # Может хранить объекты разных типов
l

## $a
## [1] 1
##
## $b
## [1] "string"
##
## $f
## function (save = "default", status = 0, runLast = TRUE)
## .Internal(quit(save, status, runLast))
## <bytecode: 0x312db30>
## <environment: namespace:base>

l <- list(a = 1, 2)  # Не обязательно все элементы должны иметь имена
l

## $a
## [1] 1
##
## [[2]]
## [1] 2
```

```
l <- as.list(1:3)
l
l1 <- list(1, "A")
l2 <- list("b", 10)
c(l1, l2)  # Списки можно склеивать
rep(l1, 5)  # И повторять
```

3.2 Обращение к элементам

3.2.1 Взятие подсписка

Для списков оператор `[` работает также, как и для векторов, только возвращается не подвектор, а подсписок:

```
l <- list(a = 1, b = "string", d = TRUE)
l[1:2]

## $a
## [1] 1
##
## $b
## [1] "string"

l[-2]

## $a
## [1] 1
##
## $d
## [1] TRUE

l[c("a", "b")]

## $a
## [1] 1
##
## $b
## [1] "string"

l[1:2] <- list(5, "char")
l

## $a
## [1] 5
##
## $b
## [1] "char"
##
## $d
## [1] TRUE
```

3.2.2 Взятие элемента

Оператор `[[` позволяет обратиться к элементу:

```
l[[1]]

## [1] 5

l[["d"]] <- list(42)
l

## $a
## [1] 5
##
## $b
## [1] "char"
##
## $d
## $d[[1]]
## [1] 42
```

Также к элементам списка можно обращаться через оператор $:

```
ll <- list(a = 1, b = 2, "ccc")
ll$a

## [1] 1

ll$b <- 42
ll$c  # При чтении достаточно уникального префикса

## NULL

ll

## $a
## [1] 1
##
## $b
## [1] 42
##
## [[3]]
## [1] "ccc"

ll$c <- 42  # А при записи будет создан элемент с переданным именем
ll

## $a
## [1] 1
##
## $b
## [1] 42
##
```

```
## [[3]]
## [1] "ccc"
##
## $c
## [1] 42
```

Присваивание элементу значения NULL удаляет элемент:

```
l

## $a
## [1] 5
##
## $b
## [1] "char"
##
## $d
## $d[[1]]
## [1] 42

l[[1]] <- NULL
l$d <- NULL
l

## $b
## [1] "char"
```

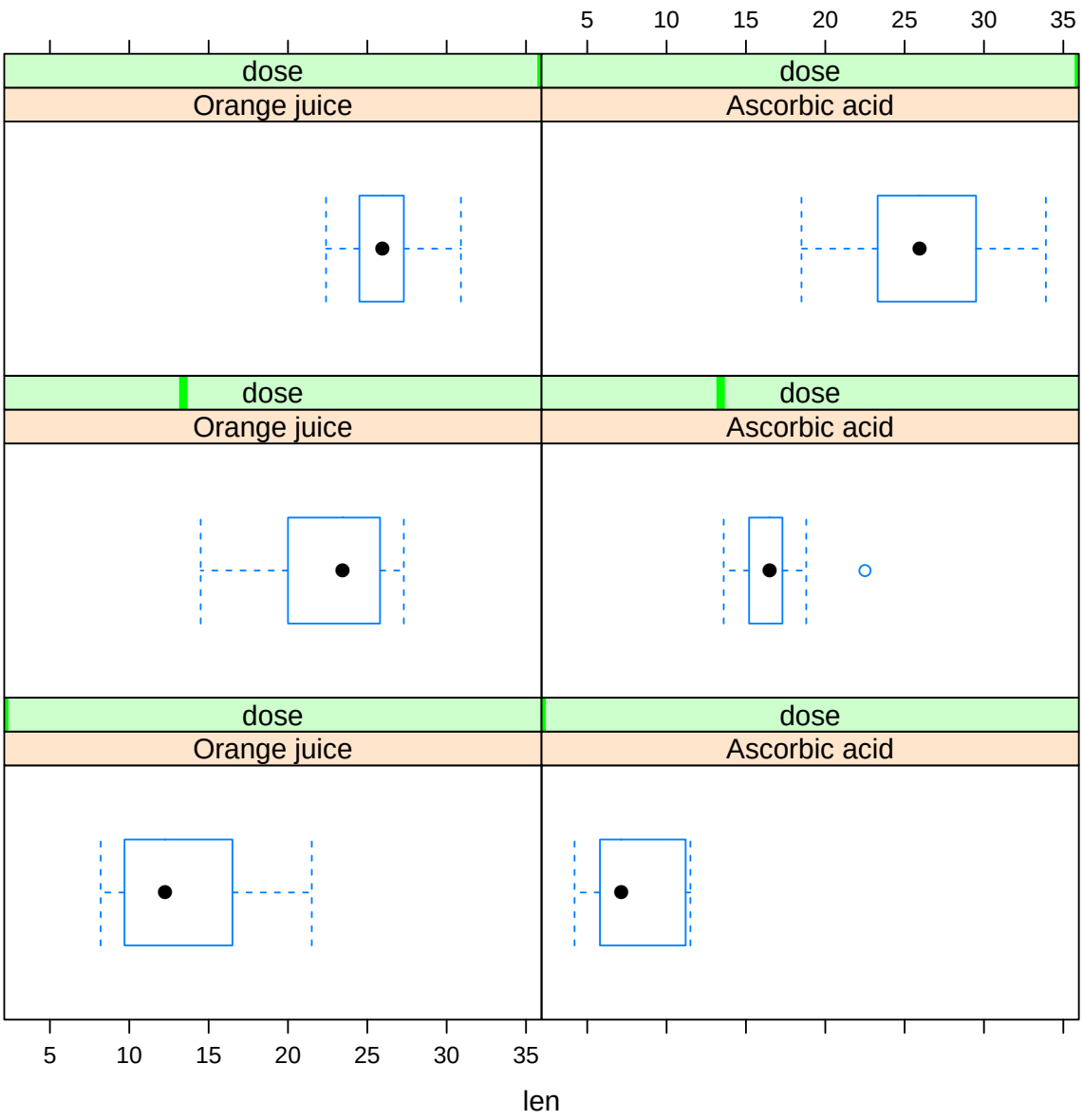
Если Вам по каким-то причинам надо положить NULL в список, то это делается так:

```
l[1] <- list(NULL)
```

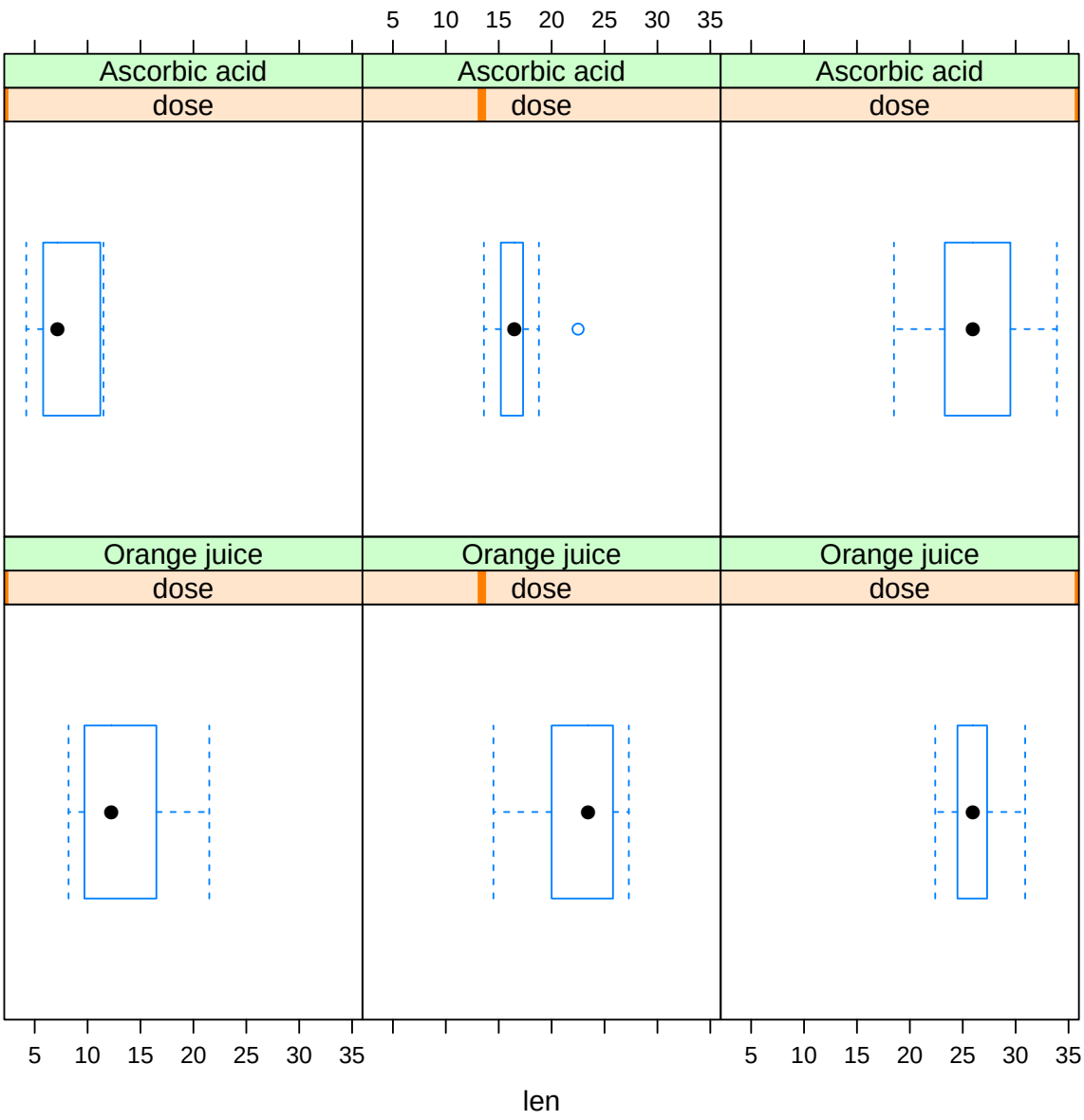
4 Материалы с занятия 3 октября

4.1 Toothgrowth

```
library(lattice)
library(latticeExtra)

## Loading required package: RColorBrewer

library(MASS)
tooth <- read.table("toothgrowth.txt")
tooth$supp <- factor(tooth$supp, labels = c("Orange juice",
    "Ascorbic acid"))
tooth$supp <- factor(tooth$supp, levels = c("Orange juice",
    "Ascorbic acid"))
bwplot(~len | supp * dose, data = tooth)
```




```
bwplot(~len | dose * supp, data = tooth)
```




```
contrasts(tooth$supp)

##                Ascorbic acid
## Orange juice               0
## Ascorbic acid              1

contrasts(tooth$supp) <- contr.sum
contrasts(tooth$supp)

##                [,1]
## Orange juice      1
## Ascorbic acid    -1

l <- lm(len ~ supp + dose, data = tooth)
summary(l)
```

```
##
## Call:
## lm(formula = len ~ supp + dose, data = tooth)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.600 -3.700  0.373  2.116  8.800
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.4225     1.1599   6.399 3.17e-08 ***
## supp1         1.8500     0.5468   3.383   0.0013 **
## dose          9.7636     0.8768  11.135 6.31e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.236 on 57 degrees of freedom
## Multiple R-squared:  0.7038,Adjusted R-squared:  0.6934
## F-statistic: 67.72 on 2 and 57 DF,  p-value: 8.716e-16

l <- lm(len ~ supp * dose, data = tooth)
summary(l)

##
## Call:
## lm(formula = len ~ supp * dose, data = tooth)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.2264 -2.8462  0.0504  2.2893  7.9386
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.4225     1.1182   6.638 1.37e-08 ***
## supp1         4.1275     1.1182   3.691 0.000507 ***
## dose          9.7636     0.8453  11.551  < 2e-16 ***
## supp1:dose   -1.9521     0.8453  -2.309 0.024631 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.083 on 56 degrees of freedom
## Multiple R-squared:  0.7296,Adjusted R-squared:  0.7151
## F-statistic: 50.36 on 3 and 56 DF,  p-value: 6.521e-16

tooth$dose <- factor(tooth$dose, ordered = TRUE)
contrasts(tooth$dose)
```

```
##                  .L         .Q
## [1,] -7.071068e-01  0.4082483
## [2,] -7.850462e-17 -0.8164966
## [3,]  7.071068e-01  0.4082483
```

```r
contrasts(tooth$dose) <- contr.helmert
contrasts(tooth$dose)
```

```
##     [,1] [,2]
## 0.5   -1   -1
## 1      1   -1
## 2      0    2
```

```r
l <- lm(len ~ supp * dose, data = tooth)
summary(l)
```

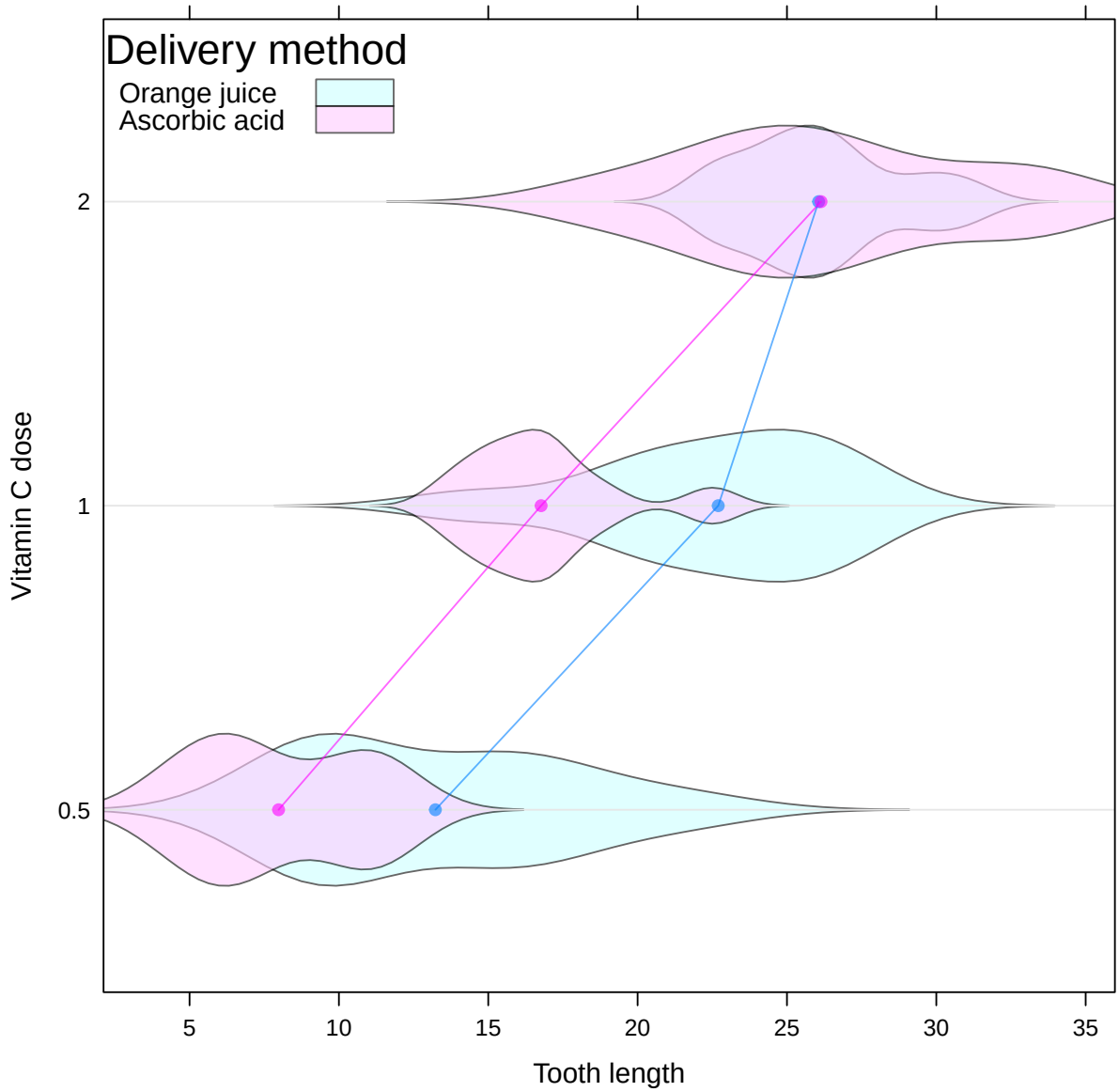
```
##
## Call:
## lm(formula = len ~ supp * dose, data = tooth)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -8.20  -2.72  -0.27   2.65   8.27
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  18.8133     0.4688  40.130  < 2e-16 ***
## supp1         1.8500     0.4688   3.946 0.000231 ***
## dose1         4.5650     0.5742   7.951 1.19e-10 ***
## dose2         3.6433     0.3315  10.990 2.17e-15 ***
## supp1:dose1   0.1700     0.5742   0.296 0.768308
## supp1:dose2  -0.9450     0.3315  -2.851 0.006166 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.631 on 54 degrees of freedom
## Multiple R-squared:  0.7937,Adjusted R-squared:  0.7746
## F-statistic: 41.56 on 5 and 54 DF,  p-value: < 2.2e-16
```

```r
stepAIC(l)
```

```
## Start:  AIC=160.43
## len ~ supp * dose
##
##             Df Sum of Sq    RSS    AIC
## <none>                   712.11 160.43
## - supp:dose  2    108.32 820.43 164.93
##
```

```
## Call:
## lm(formula = len ~ supp * dose, data = tooth)
##
## Coefficients:
## (Intercept)        supp1        dose1        dose2
##      18.813        1.850        4.565        3.643
## supp1:dose1  supp1:dose2
##       0.170       -0.945

l.lin <- lm(len ~ supp + dose, data = tooth)
# the smaller AIC/BIC, the better the fit
AIC(l, l.lin)

##       df      AIC
## l      7 332.7056
## l.lin  5 337.2013

BIC(l, l.lin)

##       df     BIC
## l      7 347.366
## l.lin  5 347.673

anova(l, l.lin)

## Analysis of Variance Table
##
## Model 1: len ~ supp * dose
## Model 2: len ~ supp + dose
##   Res.Df    RSS Df Sum of Sq     F  Pr(>F)
## 1     54 712.11
## 2     56 820.43 -2   -108.32 4.107 0.02186 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

tooth.agg <- aggregate(subset(tooth, select = len),
    list(supp = tooth$supp, dose = tooth$dose),
    mean)
dp <- dotplot(factor(dose) ~ len, groups = supp,
    data = tooth.agg, auto.key = list(title = "Delivery",
        corner = c(0, 1)), type = "b", xlab = "mean(tooth length)",
    ylab = "Vitamin C dose", par.settings = simpleTheme(pch = 19))
vp <- bwplot(factor(dose) ~ len, groups = supp,
    data = tooth, panel = function(...) {
        panel.superpose(..., col = trellis.par.get("superpose.polygon")$col,
            panel.groups = panel.violin)
    }, auto.key = list(title = "Delivery method",
        corner = c(0, 1), points = FALSE,
```

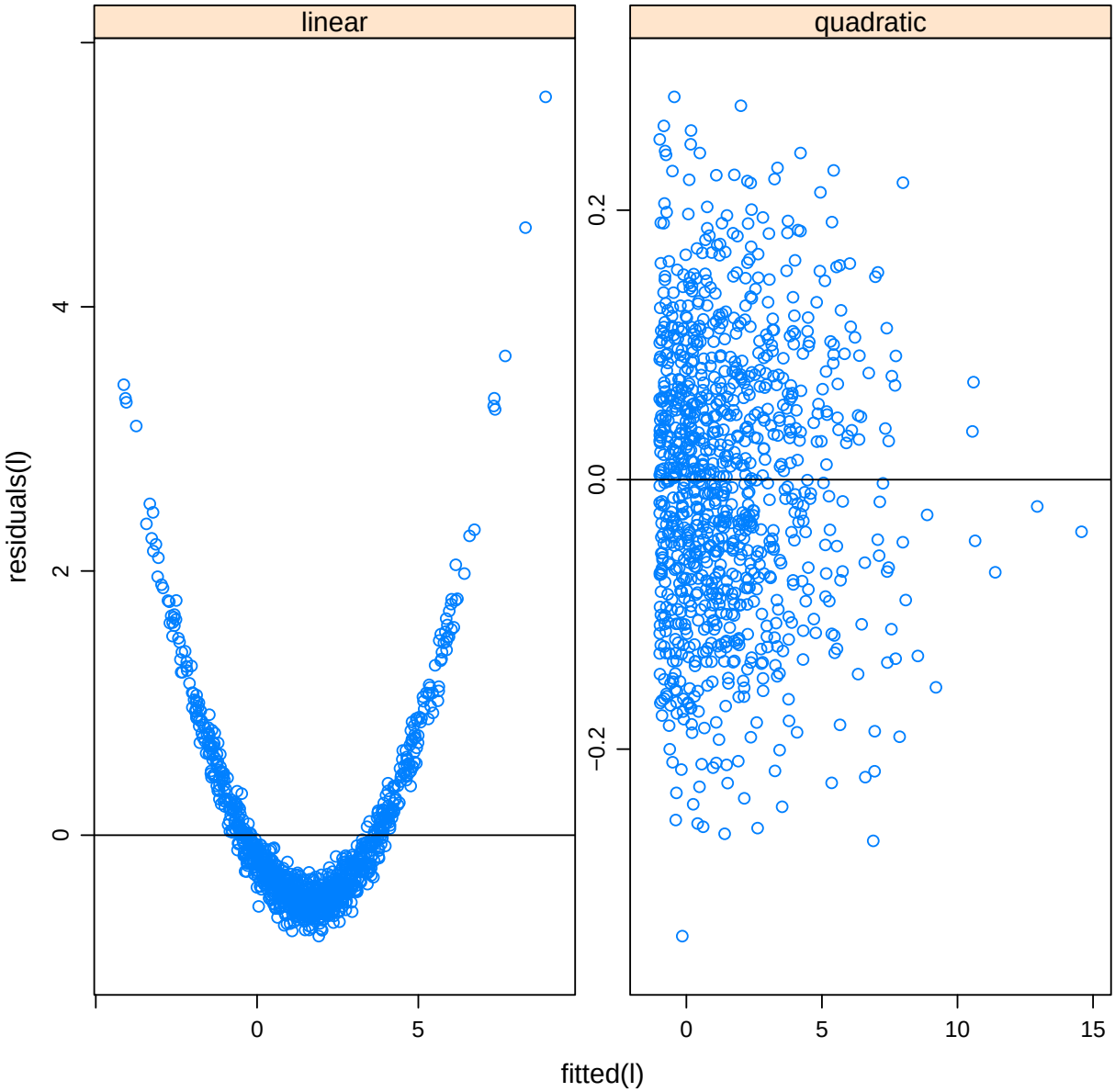
```
        lines = FALSE, rectangles = TRUE),
    xlab = "Tooth length", ylab = "Vitamin C dose",
    par.settings = simpleTheme(alpha = 0.6,
        pch = 19))
vp + dp
```



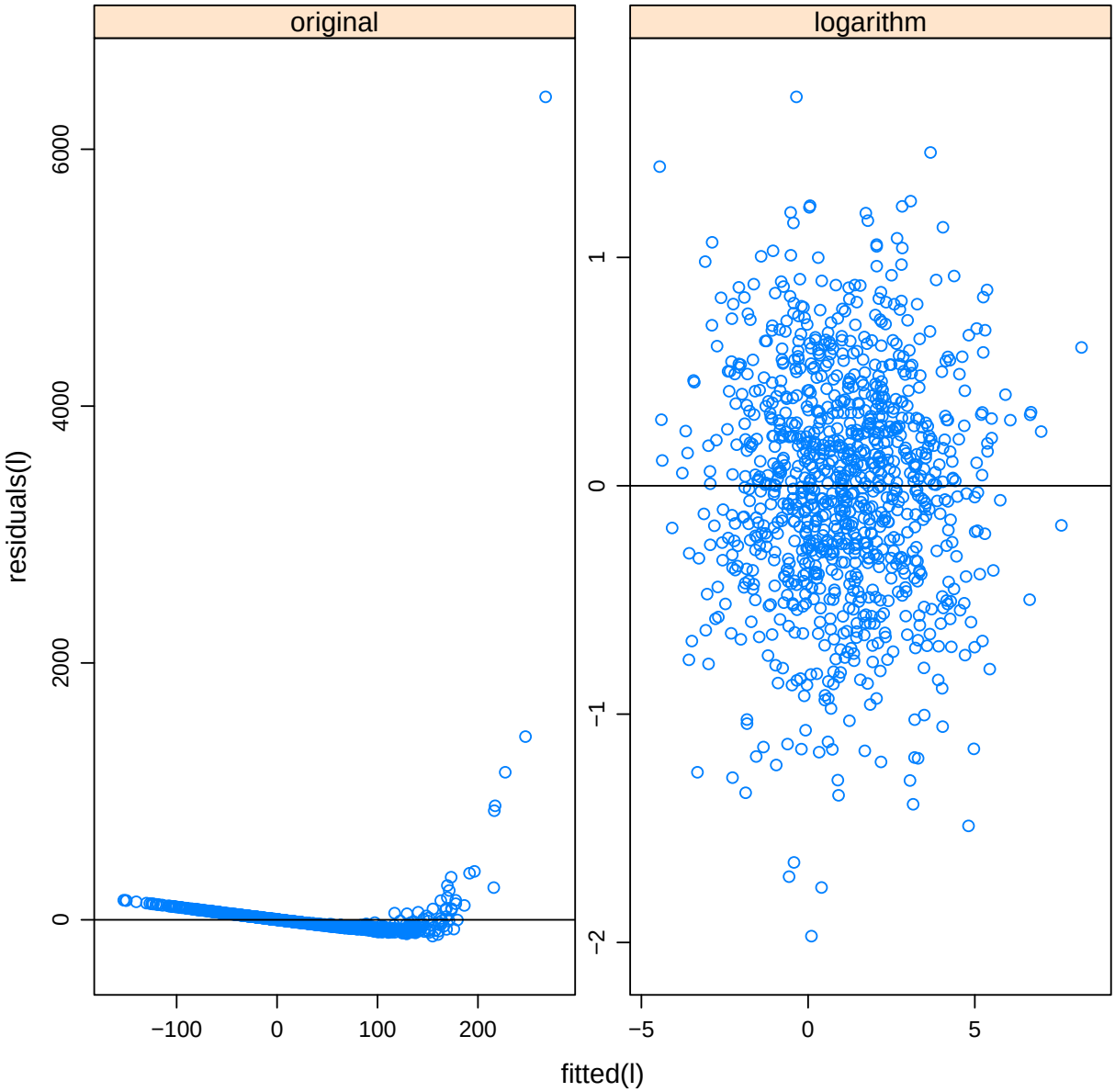
4.2 Графики residuals-vs-fitted

```
library(lattice)
library(latticeExtra)
library(MASS)
panel <- function(...) {
    panel.xyplot(...)
```

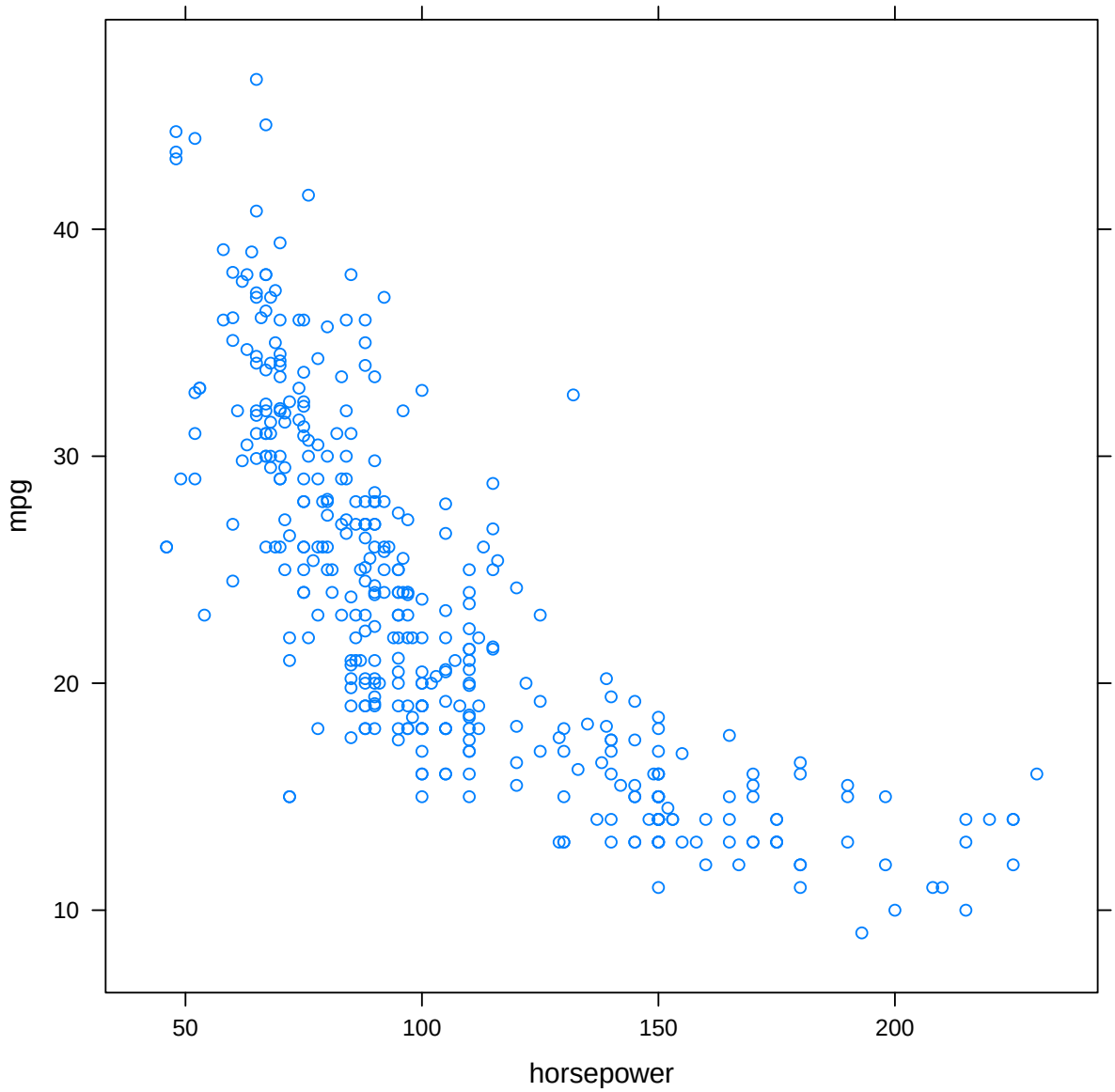
```r
    panel.lmline(...)
}
N <- 1000
x <- rnorm(N)
beta0 <- 1
beta1 <- 2
beta1_2 <- 0.5
y <- beta0 + beta1 * x + beta1_2 * x^2 +
    rnorm(N, sd = 0.1)
df <- data.frame(y = y, x = x)
l <- lm(y ~ x, data = df)
l2 <- lm(y ~ poly(x, degree = 2), data = df)
p1 <- xyplot(residuals(l) ~ fitted(l), panel = panel)
p2 <- xyplot(residuals(l2) ~ fitted(l2),
    panel = panel)
plot(c(linear = p1, quadratic = p2))
```



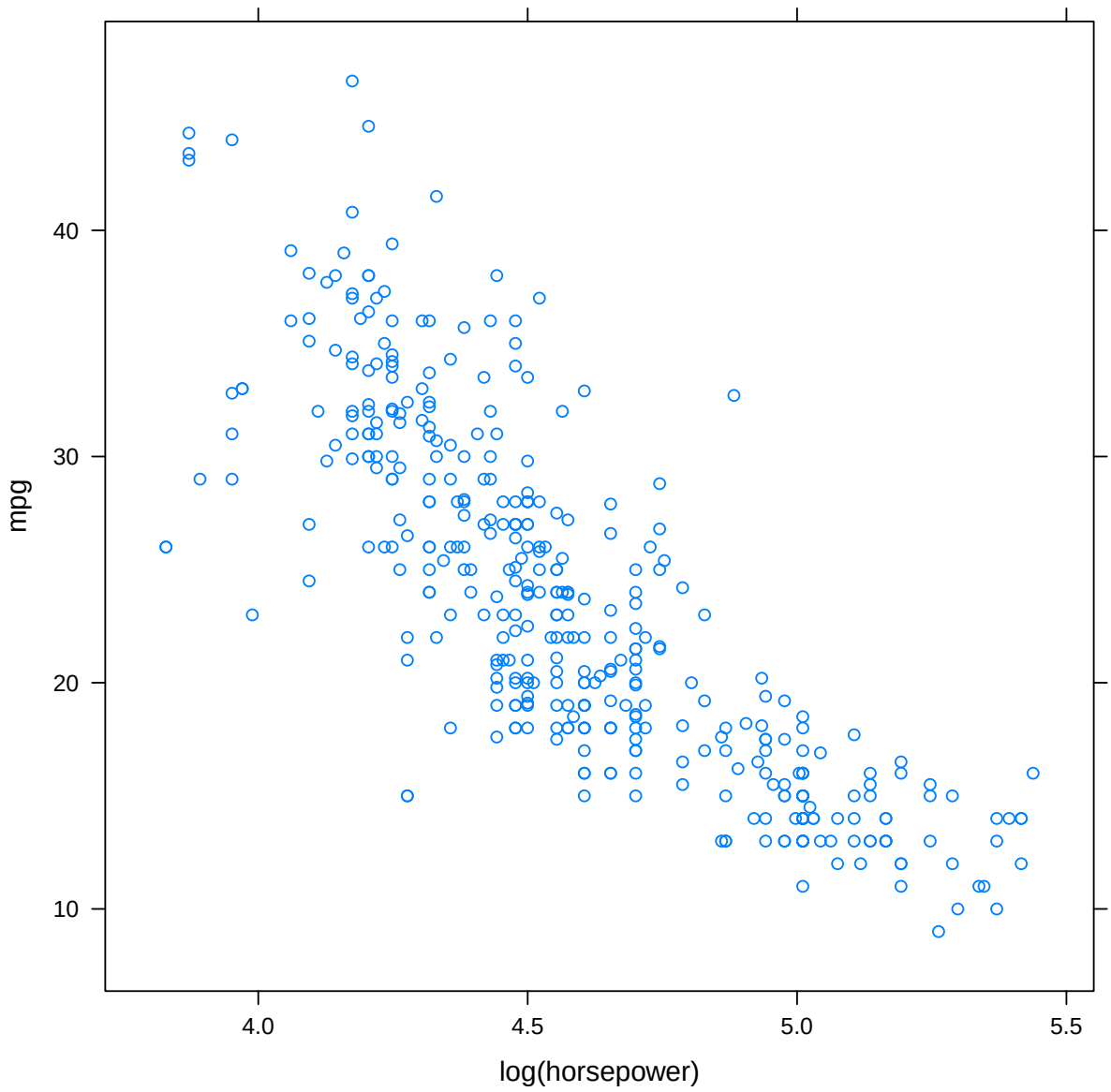

```
y <- exp(beta0 + beta1 * x + rnorm(N, sd = 0.5))
df <- data.frame(y = y, x = x)
l <- lm(y ~ x, data = df)
l2 <- lm(log(y) ~ x, data = df)
p1 <- xyplot(residuals(l) ~ fitted(l), panel = panel)
p2 <- xyplot(residuals(l2) ~ fitted(l2),
    panel = panel)
plot(c(original = p1, logarithm = p2))
```




```
Auto <- read.table("Auto.data", header = TRUE,
    na.strings = "?")
Auto <- na.omit(Auto)
xyplot(mpg ~ horsepower, data = Auto)
```




```
xyplot(mpg ~ log(horsepower), data = Auto)
```




```
l1 <- lm(mpg ~ horsepower, data = Auto)
l2 <- lm(mpg ~ poly(horsepower, degree = 2),
    data = Auto)
llog <- lm(mpg ~ log(horsepower), data = Auto)
l5 <- lm(mpg ~ poly(horsepower, degree = 5),
    data = Auto)
l6 <- lm(mpg ~ poly(horsepower, degree = 6),
    data = Auto)
summary(l1)

##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -13.5710  -3.2592  -0.3435   2.7630  16.9240
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.935861   0.717499   55.66   <2e-16 ***
## horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.906 on 390 degrees of freedom
## Multiple R-squared:  0.6059,Adjusted R-squared:  0.6049
## F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16
```

```r
summary(llog)
```

```
##
## Call:
## lm(formula = mpg ~ log(horsepower), data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.2299  -2.7818  -0.2322   2.6661  15.4695
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     108.6997     3.0496   35.64   <2e-16 ***
## log(horsepower) -18.5822     0.6629  -28.03   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.501 on 390 degrees of freedom
## Multiple R-squared:  0.6683,Adjusted R-squared:  0.6675
## F-statistic: 785.9 on 1 and 390 DF,  p-value: < 2.2e-16
```

```r
summary(l5)
```

```
##
## Call:
## lm(formula = mpg ~ poly(horsepower, degree = 5), data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.4326  -2.5285  -0.2925   2.1750  15.9730
##
## Coefficients:
##                                Estimate Std. Error t value
```

```
## (Intercept)                      23.4459     0.2185 107.308
## poly(horsepower, degree = 5)1 -120.1377     4.3259 -27.772
## poly(horsepower, degree = 5)2   44.0895     4.3259  10.192
## poly(horsepower, degree = 5)3   -3.9488     4.3259  -0.913
## poly(horsepower, degree = 5)4   -5.1878     4.3259  -1.199
## poly(horsepower, degree = 5)5   13.2722     4.3259   3.068
##                                 Pr(>|t|)
## (Intercept)                      < 2e-16 ***
## poly(horsepower, degree = 5)1  < 2e-16 ***
## poly(horsepower, degree = 5)2  < 2e-16 ***
## poly(horsepower, degree = 5)3  0.36190
## poly(horsepower, degree = 5)4  0.23117
## poly(horsepower, degree = 5)5  0.00231 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.326 on 386 degrees of freedom
## Multiple R-squared:  0.6967,Adjusted R-squared:  0.6928
## F-statistic: 177.4 on 5 and 386 DF,  p-value: < 2.2e-16
```

```r
summary(l6)
```

```
##
## Call:
## lm(formula = mpg ~ poly(horsepower, degree = 6), data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.571  -0.269   2.209  15.362
##
## Coefficients:
##                                 Estimate Std. Error t value
## (Intercept)                      23.4459     0.2177 107.715
## poly(horsepower, degree = 6)1 -120.1377     4.3096 -27.877
## poly(horsepower, degree = 6)2   44.0895     4.3096  10.231
## poly(horsepower, degree = 6)3   -3.9488     4.3096  -0.916
## poly(horsepower, degree = 6)4   -5.1878     4.3096  -1.204
## poly(horsepower, degree = 6)5   13.2722     4.3096   3.080
## poly(horsepower, degree = 6)6   -8.5462     4.3096  -1.983
##                                 Pr(>|t|)
## (Intercept)                      < 2e-16 ***
## poly(horsepower, degree = 6)1  < 2e-16 ***
## poly(horsepower, degree = 6)2  < 2e-16 ***
## poly(horsepower, degree = 6)3  0.36008
## poly(horsepower, degree = 6)4  0.22941
## poly(horsepower, degree = 6)5  0.00222 **
## poly(horsepower, degree = 6)6  0.04807 *
```

```
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.31 on 385 degrees of freedom
## Multiple R-squared:  0.6998,Adjusted R-squared:  0.6951
## F-statistic: 149.6 on 6 and 385 DF,  p-value: < 2.2e-16
```

```
AIC(l1, l2, llog, l5, l6)
```

```
##      df      AIC
## l1    3 2363.324
## l2    4 2274.354
## llog  3 2295.760
## l5    7 2268.663
## l6    8 2266.680
```

```
BIC(l1, l2, llog, l5, l6)
```

```
##      df      BIC
## l1    3 2375.237
## l2    4 2290.239
## llog  3 2307.674
## l5    7 2296.462
## l6    8 2298.450
```

```
anova(l1, l2)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, degree = 2)
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    390 9385.9
## 2    389 7442.0  1    1943.9 101.61 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(l1, l5)
```

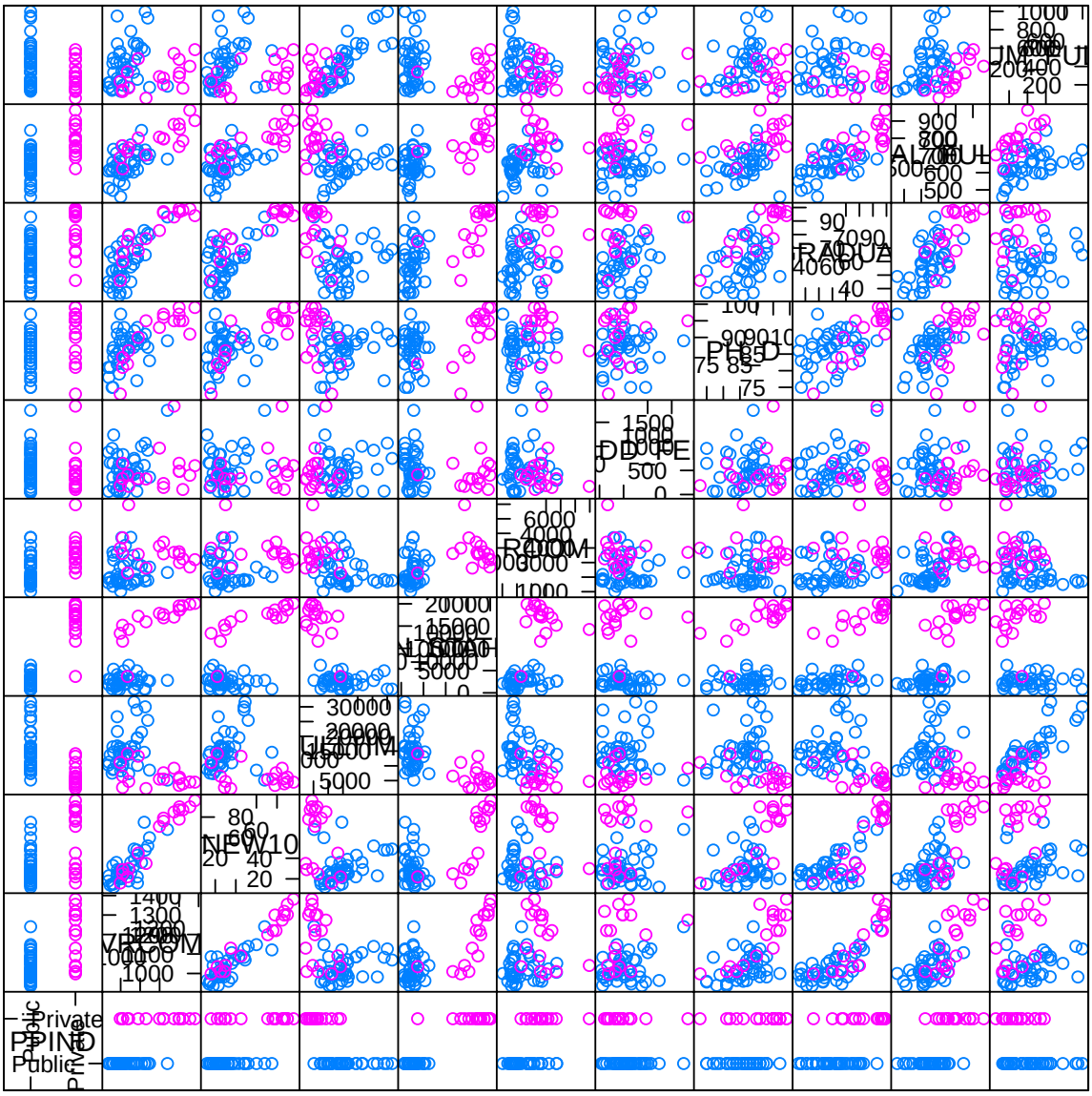
```
## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, degree = 5)
##   Res.Df    RSS Df Sum of Sq     F    Pr(>F)
## 1    390 9385.9
## 2    386 7223.4  4    2162.5 28.89 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
anova(l1, l6)

## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, degree = 6)
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    390 9385.9
## 2    385 7150.3  5    2235.6 24.074 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(l5, l6)

## Analysis of Variance Table
##
## Model 1: mpg ~ poly(horsepower, degree = 5)
## Model 2: mpg ~ poly(horsepower, degree = 6)
##   Res.Df    RSS Df Sum of Sq      F  Pr(>F)
## 1    386 7223.4
## 2    385 7150.3  1    73.038 3.9326 0.04807 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4.3 Линейная регрессия (Университеты)

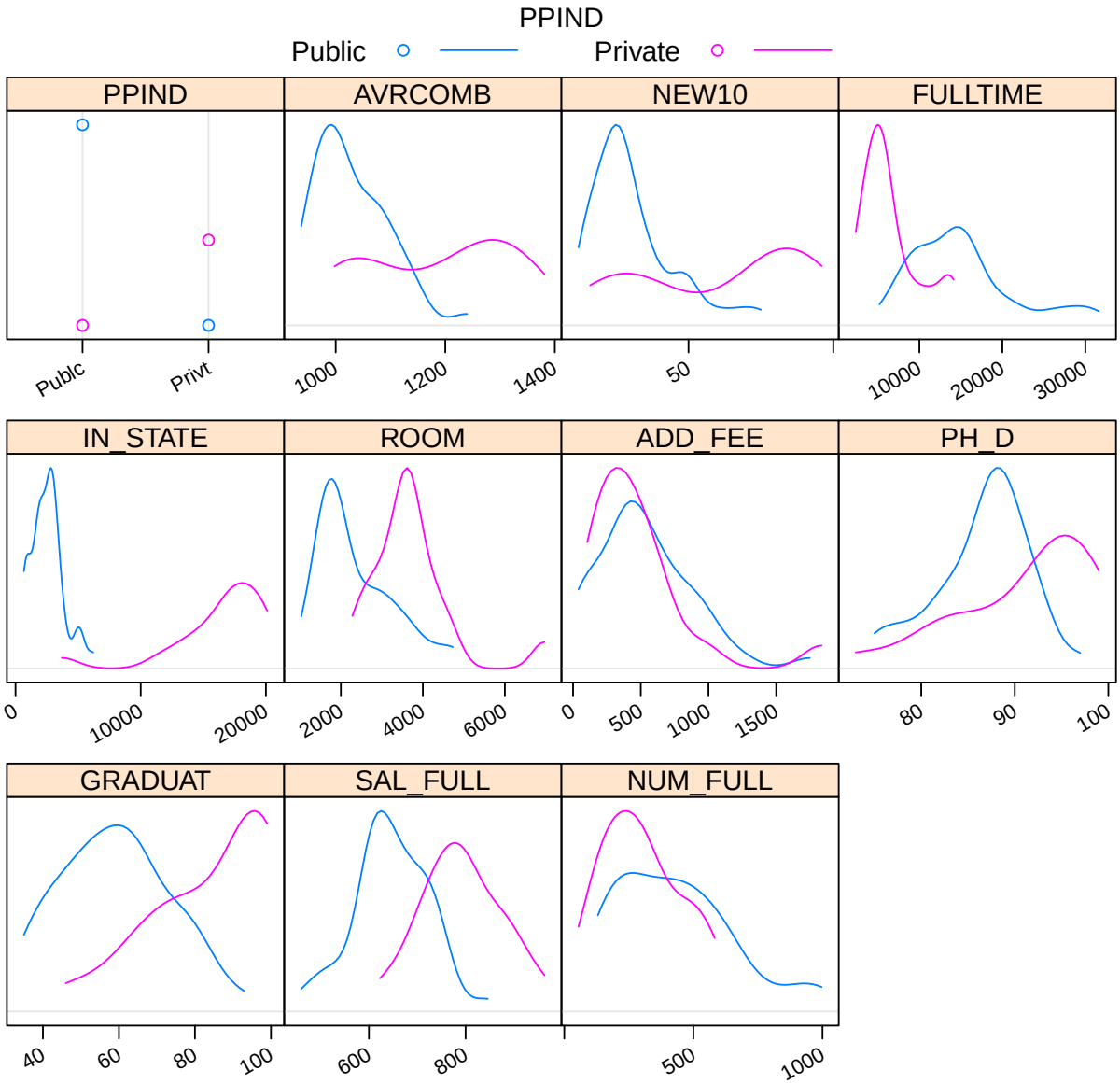
```r
library(MASS)
library(lattice)
library(latticeExtra)
library(latticist)

## Loading required package: vcd
## Loading required package: grid
##
## Attaching package: 'vcd'
##
## The following object is masked from 'package:latticeExtra':
##
##    rootogram

df <- read.csv2(file = "I.csv")
# Я установил пакет latticist, чтобы
# вдоволь налюбоваться на университеты Я
# решил, что переменных слишком много и
# оставил только по одной из каждого
```

```r
# класса Например, количество вечерников
# и очников --- явно характеристики
# одного и того же, поэтому оставим
# только одну из них Мы же помним, что
# брать сильно кореллированые признаки в
# модель --- дурной тон?)  Итааааааак,
# барабанная дробь, я решил оставить:
# PPIND - фактор, 1 - Государственный, 2
# - Частный университет.  AVRCOMB -
# средний средний балл на вступительных
# экзаменах (SAT, вроде нашего ЕГЭ).
# NEW10 - Это то, что будем
# аппроксимровать, задание у нас такое
# Это процент свежезачисленных
# студентов-отличников (Процент среди
# поступивших, тех, кто в H.S. входил в
# 10% лучших) В оригинале ''Pct. new
# students from top 10% of H.S. class''
# FULLTIME - Количество студентов-очников
# IN_STATE - Плата за обучение для
# местных ROOM - Плата за койку в общаге
# ADD_FEE - Дополнительные поборы (сверх
# платы за обучение, койку и учебные
# материалы) PH_D - Процент кандидатов
# наук среди педагогического состава.
# GRADUAT - Процент выпустившихся.
# Гы-гы=) SAL_FULL - Средняя зарплата
# полного профессора (full professor).
# NUM_FULL - Количетво этих самых полных
# профессоров Теперь нам надо обрезать и
# подправить исходный датафрейм и
# скормить его latticist.
# latticist(df)
# Отобрали признаки
df <- subset(df, select = c(PPIND, AVRCOMB,
    NEW10, FULLTIME, IN_STATE, ROOM, ADD_FEE,
    PH_D, GRADUAT, SAL_FULL, NUM_FULL))
# Сконвертировали тип Университета в
# фактор, так и вывод красивее, и в
# модели будет удобнее интерпретировать
df$PPIND <- factor(df$PPIND, labels = c("Public",
    "Private"))
df <- na.exclude(df)
splom(df, groups = df$PPIND)
```

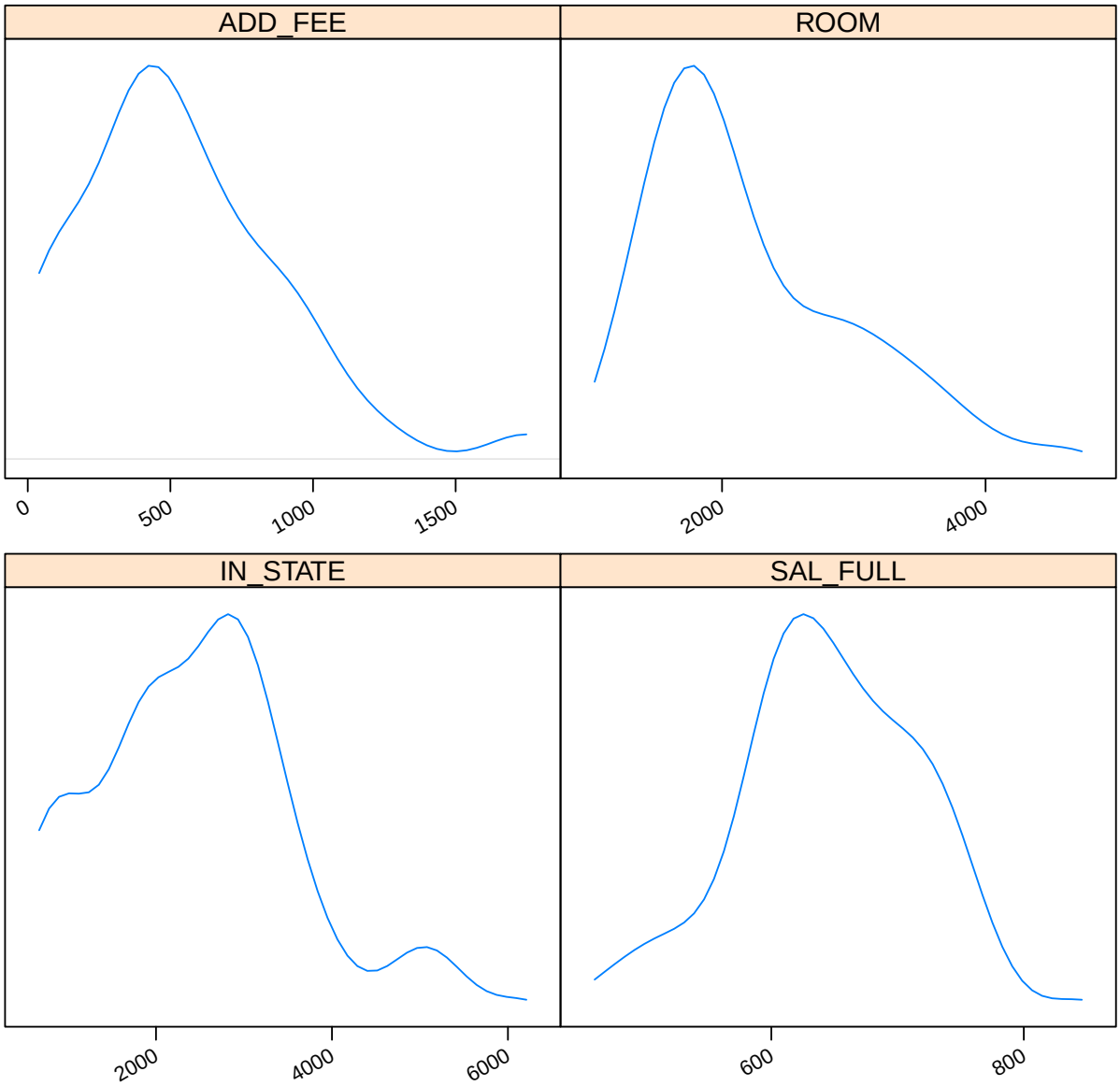


Scatter Plot Matrix

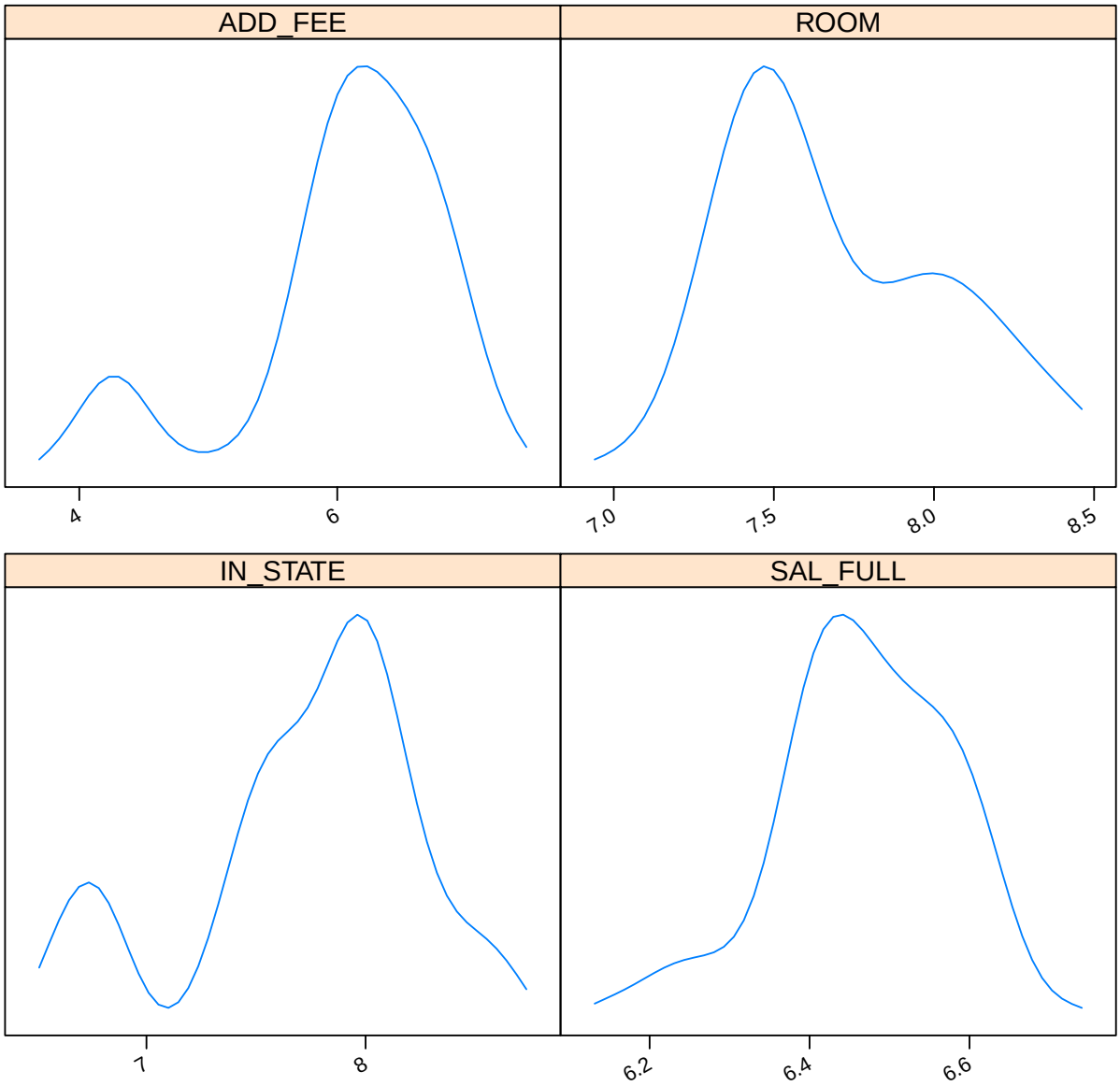
```
# Гвардия, в бой!  latticist(df)
# На этом графике хорошо видно, что
# данные неоднородны.  Нам надо будет
# выбрать, кого оставить --- частные или
# государственные университеты.  Я
# оставлю государственные, потому что их
# больше.
marginal.plot(df, data = df, groups = PPIND,
    auto.key = list(lines = TRUE, title = "PPIND",
        cex.title = 1, columns = 2))
```




```r
df.pub <- subset(df, PPIND == "Public")
# Теперь посмотрим на нормальность,
# может, что-то стоит
# прологарифмировать?...
# latticist(df.pub) Подозрение падает на
# денежные признаки. Они часто
# логнормальны.
marginal.plot(subset(df.pub, select = c(ADD_FEE,
    ROOM, IN_STATE, SAL_FULL)))
```




```
marginal.plot(log(subset(df.pub, select = c(ADD_FEE,
    ROOM, IN_STATE, SAL_FULL))))
```




```
# После логарифмирования появилась
# мультимодальность, хотя распределения
# стали на вид немного более
# симметричными.  Я все-таки хочу
# оставить логарифмирование, потому что
# это денежные признаки Но потом мы
# проверим и без него
# Итого
fit1 <- lm(NEW10 ~ AVRCOMB + FULLTIME + log(IN_STATE) +
    log(ROOM) + log(ADD_FEE) + log(SAL_FULL) +
    PH_D + GRADUAT + NUM_FULL, data = df.pub)
summary(fit1)

##
## Call:
```

```
## lm(formula = NEW10 ~ AVRCOMB + FULLTIME + log(IN_STATE) + log(ROOM) +
##     log(ADD_FEE) + log(SAL_FULL) + PH_D + GRADUAT + NUM_FULL,
##     data = df.pub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -22.770  -3.407   1.111   3.588  15.071
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -4.397e+01  8.490e+01  -0.518  0.60828
## AVRCOMB        1.469e-01  2.823e-02   5.203 1.31e-05 ***
## FULLTIME      -2.458e-04  3.797e-04  -0.647  0.52227
## log(IN_STATE) -9.057e+00  2.646e+00  -3.422  0.00181 **
## log(ROOM)     -4.011e+00  4.082e+00  -0.982  0.33375
## log(ADD_FEE)  -3.651e+00  1.628e+00  -2.242  0.03251 *
## log(SAL_FULL)  1.114e+01  1.459e+01   0.764  0.45090
## PH_D          -4.984e-01  2.842e-01  -1.754  0.08965 .
## GRADUAT        2.645e-01  1.469e-01   1.801  0.08173 .
## NUM_FULL       1.225e-02  1.172e-02   1.045  0.30431
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.755 on 30 degrees of freedom
## Multiple R-squared:  0.7874,Adjusted R-squared:  0.7236
## F-statistic: 12.35 on 9 and 30 DF,  p-value: 6.861e-08

# Имеем --- отличники прекрасно сдают
# экзамены и поступают туда, где меньше
# надо платить, меньше поборов, меньше
# кандидатов (sic!)  и больше процент
# успешно закончивших.  На самом деле,
# это несодержательно.  AVRCOMB ---
# абсолютно не нужен нам.  И так понятно,
# что отличники там, где отличники.  Если
# мы хотим получить действительно
# информативную модель и нетривиальные
# выводы, то из предикторов AVRCOMB имеет
# смысл убрать, иначе трактовка регрессии
# будет тавтологией
fit2 <- lm(NEW10 ~ FULLTIME + log(IN_STATE) +
    log(ROOM) + log(ADD_FEE) + log(SAL_FULL) +
    PH_D + GRADUAT + NUM_FULL, data = df.pub)
summary(fit2)

##
## Call:
```

```
## lm(formula = NEW10 ~ FULLTIME + log(IN_STATE) + log(ROOM) + log(ADD_FEE) +
##     log(SAL_FULL) + PH_D + GRADUAT + NUM_FULL, data = df.pub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -22.749  -5.841   0.533   5.365  23.062
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.531e+01  1.128e+02   0.402 0.690693
## FULLTIME      -3.024e-04  5.150e-04  -0.587 0.561361
## log(IN_STATE) -1.313e+01  3.430e+00  -3.829 0.000586 ***
## log(ROOM)     -2.312e+00  5.521e+00  -0.419 0.678299
## log(ADD_FEE)  -4.263e+00  2.204e+00  -1.934 0.062256 .
## log(SAL_FULL)  1.671e+01  1.974e+01   0.846 0.403798
## PH_D          -2.902e-01  3.818e-01  -0.760 0.452957
## GRADUAT        7.369e-01  1.566e-01   4.705    5e-05 ***
## NUM_FULL       1.917e-02  1.580e-02   1.213 0.234185
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.52 on 31 degrees of freedom
## Multiple R-squared:  0.5956,Adjusted R-squared:  0.4912
## F-statistic: 5.706 on 8 and 31 DF,  p-value: 0.0001718

# Уже на что-то похоже.  Поступают туда,
# где дешевле, где больше шанс
# выпуститься и меньше поборов.
# Попробуем уменьшить число признаков.
# Вручную по t-test и по Акаике
fit2.manual <- lm(NEW10 ~ log(IN_STATE) +
    log(ADD_FEE) + GRADUAT, data = df.pub)
summary(fit2.manual)

##
## Call:
## lm(formula = NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT,
##     data = df.pub)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.2271  -5.1813  -0.4403   6.7982  19.5760
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   113.0619    28.2689   4.000 0.000302 ***
## log(IN_STATE) -13.1875     3.2280  -4.085 0.000235 ***
```

```
## log(ADD_FEE)   -4.8304     2.1489  -2.248 0.030801 *
## GRADUAT         0.8187     0.1376   5.949 8.14e-07 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.53 on 36 degrees of freedom
## Multiple R-squared:   0.53,Adjusted R-squared:  0.4908
## F-statistic: 13.53 on 3 and 36 DF,  p-value: 4.559e-06

fit2.aic <- stepAIC(fit2)

## Start:  AIC=196.08
## NEW10 ~ FULLTIME + log(IN_STATE) + log(ROOM) + log(ADD_FEE) +
##     log(SAL_FULL) + PH_D + GRADUAT + NUM_FULL
##
##                 Df Sum of Sq    RSS    AIC
## - log(ROOM)      1     19.41 3451.4 194.31
## - FULLTIME       1     38.17 3470.2 194.52
## - PH_D           1     63.96 3496.0 194.82
## - log(SAL_FULL)  1     79.32 3511.3 195.00
## - NUM_FULL       1    162.97 3595.0 195.94
## <none>                       3432.0 196.08
## - log(ADD_FEE)   1    414.20 3846.2 198.64
## - log(IN_STATE)  1   1623.10 5055.1 209.57
## - GRADUAT        1   2450.49 5882.5 215.63
##
## Step:  AIC=194.31
## NEW10 ~ FULLTIME + log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) +
##     PH_D + GRADUAT + NUM_FULL
##
##                 Df Sum of Sq    RSS    AIC
## - FULLTIME       1     36.92 3488.4 192.73
## - log(SAL_FULL)  1     60.03 3511.5 193.00
## - PH_D           1     61.34 3512.8 193.01
## <none>                       3451.4 194.31
## - NUM_FULL       1    193.54 3645.0 194.49
## - log(ADD_FEE)   1    451.12 3902.6 197.22
## - log(IN_STATE)  1   1729.29 5180.7 208.55
## - GRADUAT        1   2629.10 6080.5 214.96
##
## Step:  AIC=192.73
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) + PH_D +
##     GRADUAT + NUM_FULL
##
##                 Df Sum of Sq    RSS    AIC
## - PH_D           1     42.66 3531.0 191.22
## - log(SAL_FULL)  1     65.23 3553.6 191.47
```

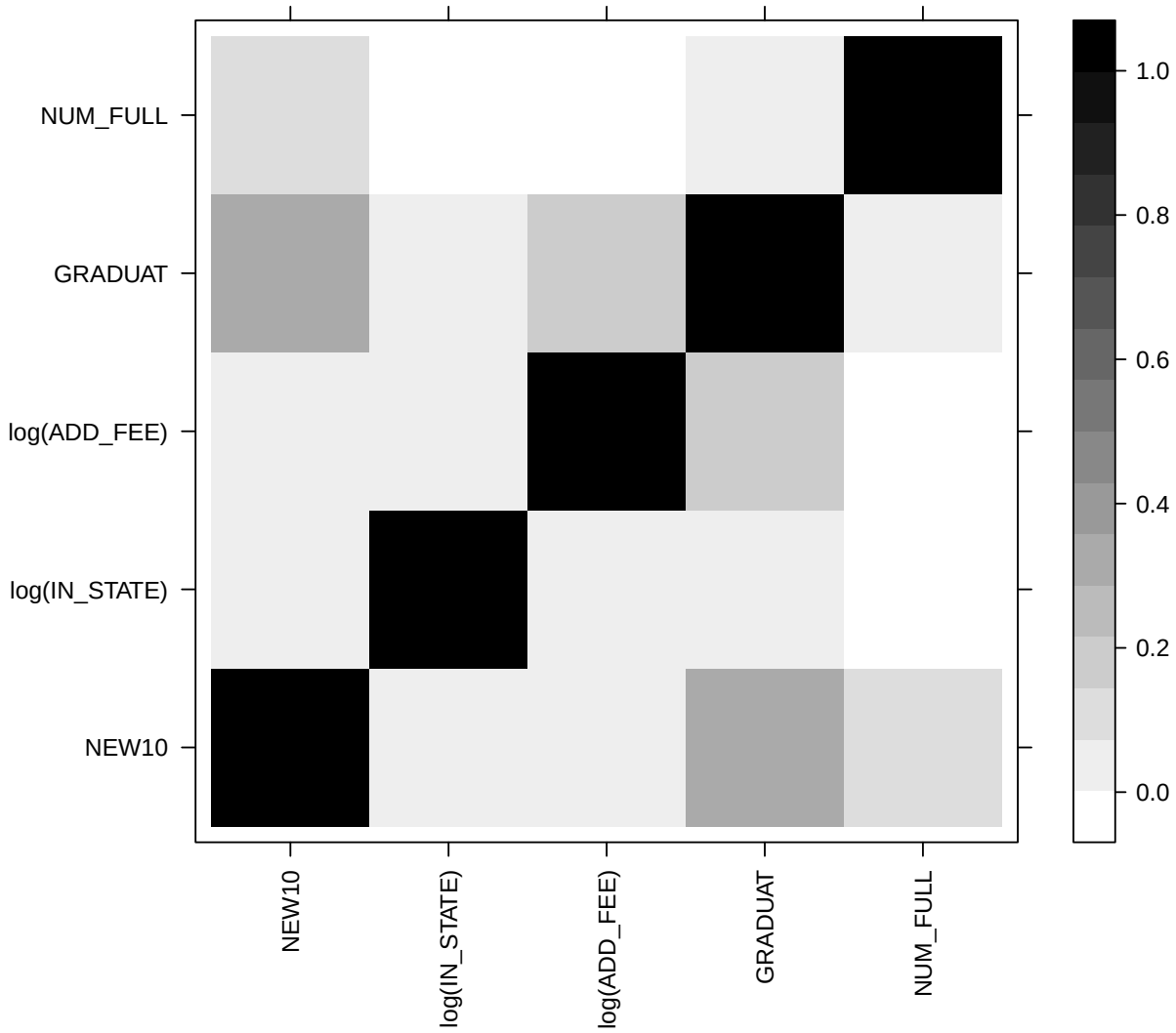
```
## <none>                        3488.4 192.73
## - NUM_FULL       1    250.29 3738.6 193.50
## - log(ADD_FEE)   1    433.03 3921.4 195.41
## - log(IN_STATE)  1   1702.06 5190.4 206.63
## - GRADUAT        1   2592.19 6080.5 212.96
##
## Step:  AIC=191.22
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) + GRADUAT +
##     NUM_FULL
##
##                 Df Sum of Sq    RSS    AIC
## - log(SAL_FULL)  1     59.98 3591.0 189.89
## <none>                        3531.0 191.22
## - NUM_FULL       1    212.33 3743.3 191.55
## - log(ADD_FEE)   1    458.20 3989.2 194.10
## - log(IN_STATE)  1   1724.25 5255.3 205.12
## - GRADUAT        1   2550.89 6081.9 210.97
##
## Step:  AIC=189.89
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT + NUM_FULL
##
##                 Df Sum of Sq    RSS    AIC
## <none>                        3591.0 189.89
## - NUM_FULL       1     397.4 3988.4 192.09
## - log(ADD_FEE)   1     446.4 4037.4 192.58
## - log(IN_STATE)  1    1664.3 5255.3 203.12
## - GRADUAT        1    3291.4 6882.4 213.91
```

```r
summary(fit2.aic)
```

```
##
## Call:
## lm(formula = NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT +
##     NUM_FULL, data = df.pub)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22.9871  -5.6649  -0.5082   5.3076  23.9919
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   102.452608  27.733017   3.694 0.000748 ***
## log(IN_STATE) -12.574064   3.121998  -4.028 0.000288 ***
## log(ADD_FEE)   -4.344059   2.082633  -2.086 0.044348 *
## GRADUAT         0.765564   0.135166   5.664 2.14e-06 ***
## NUM_FULL        0.014231   0.007231   1.968 0.057016 .
## ---
## Signif. codes:
```

```
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.13 on 35 degrees of freedom
## Multiple R-squared:  0.5768,Adjusted R-squared:  0.5285
## F-statistic: 11.93 on 4 and 35 DF,  p-value: 3.23e-06

# Акаике предлагает оставить признак
# NUM_FULL.  Отличники поступают туда
# еще, где народу побольше.
# Посравниваю-ка я модели...
AIC(fit2.manual, fit2.aic)

##             df      AIC
## fit2.manual  5 307.6059
## fit2.aic     6 305.4073

# Неудивительно, потому что fit.aic
# построена по stepAIC()
anova(fit2.manual, fit2.aic)

## Analysis of Variance Table
##
## Model 1: NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT
## Model 2: NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT + NUM_FULL
##   Res.Df    RSS Df Sum of Sq      F  Pr(>F)
## 1     36 3988.4
## 2     35 3591.0  1    397.42 3.8735 0.05702 .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Посмотрим на корреляции признаков,
# возможно, некоторые признаки захочется
# удалить, потому что они сильно
# коррелируют с другими
cor(fit2.aic$model)

##                     NEW10 log(IN_STATE) log(ADD_FEE)
## NEW10          1.0000000   -0.25000604   0.11170368
## log(IN_STATE) -0.2500060    1.00000000  -0.15425172
## log(ADD_FEE)   0.1117037   -0.15425172   1.00000000
## GRADUAT        0.5449545    0.26450468   0.41064855
## NUM_FULL       0.3516683   -0.02384413  -0.02151739
##                 GRADUAT    NUM_FULL
## NEW10         0.5449545  0.35166826
## log(IN_STATE) 0.2645047 -0.02384413
## log(ADD_FEE)  0.4106486 -0.02151739
## GRADUAT       1.0000000  0.15156033
## NUM_FULL      0.1515603  1.00000000
```

```r
levelplot(cor(fit2.aic$model)^2, par.settings = list(regions = list(col = colorRamp
    scales = list(x = list(rot = 90)), xlab = "",
    ylab = "")
```




```r
summary(fit2.aic)

##
## Call:
## lm(formula = NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT +
##     NUM_FULL, data = df.pub)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -22.9871  -5.6649  -0.5082   5.3076  23.9919
##
```

```
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   102.452608  27.733017   3.694 0.000748 ***
## log(IN_STATE) -12.574064   3.121998  -4.028 0.000288 ***
## log(ADD_FEE)   -4.344059   2.082633  -2.086 0.044348 *
## GRADUAT         0.765564   0.135166   5.664 2.14e-06 ***
## NUM_FULL        0.014231   0.007231   1.968 0.057016 .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.13 on 35 degrees of freedom
## Multiple R-squared:  0.5768,Adjusted R-squared:  0.5285
## F-statistic: 11.93 on 4 and 35 DF,  p-value: 3.23e-06

# ADD_FEE и NUM_FULL у меня на большом
# подозрении, особенно первый.  Они
# малозначимы и сильно коррелируют с
# GRADUAT, велика вероятность, что они
# мусорные Используем CV сравнение
# train-test слишком неустойчиво себя
# ведет
l <- update(fit2.aic, . ~ . - log(ADD_FEE) -
    NUM_FULL)
library(e1071)
tune(lm, fit2.aic$call$formula, data = df.pub,
    tunecontrol = tune.control(sampling = "cross",
        cross = 35))

##
## Error estimation of 'lm' using 35-fold cross validation: 129.8278

tune(lm, l$call$formula, data = df.pub, tunecontrol = tune.control(sampling = "cros
    cross = 35))

##
## Error estimation of 'lm' using 35-fold cross validation: 140.6516

# Вывод --- все-таки выкидывать не стоило
# Попробуем нелогарифмировать признаки и
# сравним модели
fit.nolog <- lm(NEW10 ~ IN_STATE + ADD_FEE +
    GRADUAT + NUM_FULL, data = df.pub)
summary(fit.nolog)

##
## Call:
## lm(formula = NEW10 ~ IN_STATE + ADD_FEE + GRADUAT + NUM_FULL,
##     data = df.pub)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.0914  -4.6808   0.5468   3.5763  24.7934
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.866977   7.509751  -0.249 0.805118
## IN_STATE    -0.005280   0.001417  -3.726 0.000685 ***
## ADD_FEE     -0.005985   0.005486  -1.091 0.282694
## GRADUAT      0.726085   0.144096   5.039 1.43e-05 ***
## NUM_FULL     0.015331   0.007459   2.055 0.047376 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.45 on 35 degrees of freedom
## Multiple R-squared:  0.5498,Adjusted R-squared:  0.4984
## F-statistic: 10.69 on 4 and 35 DF,  p-value: 9.126e-06
```

```r
AIC(fit.nolog, fit2.aic)
```

```
##           df      AIC
## fit.nolog  6 307.8804
## fit2.aic   6 305.4073
```

```r
# Как видно, прологарифмировали мы
# все-таки не зря
```

4.3.1 Advertising, окончательный результат

```r
library(lattice)
library(latticeExtra)
library(MASS)
library(e1071)
Advertising <- read.csv("Advertising.csv")
Advertising$X <- NULL
l <- lm(Sales ~ TV + Radio + Newspaper, data = Advertising)
li <- lm(Sales ~ (TV + Radio + Newspaper)^2,
    data = Advertising)
ltvradio <- lm(Sales ~ TV + Radio + Newspaper +
    TV:Radio, data = Advertising)
laic <- stepAIC(li)
```

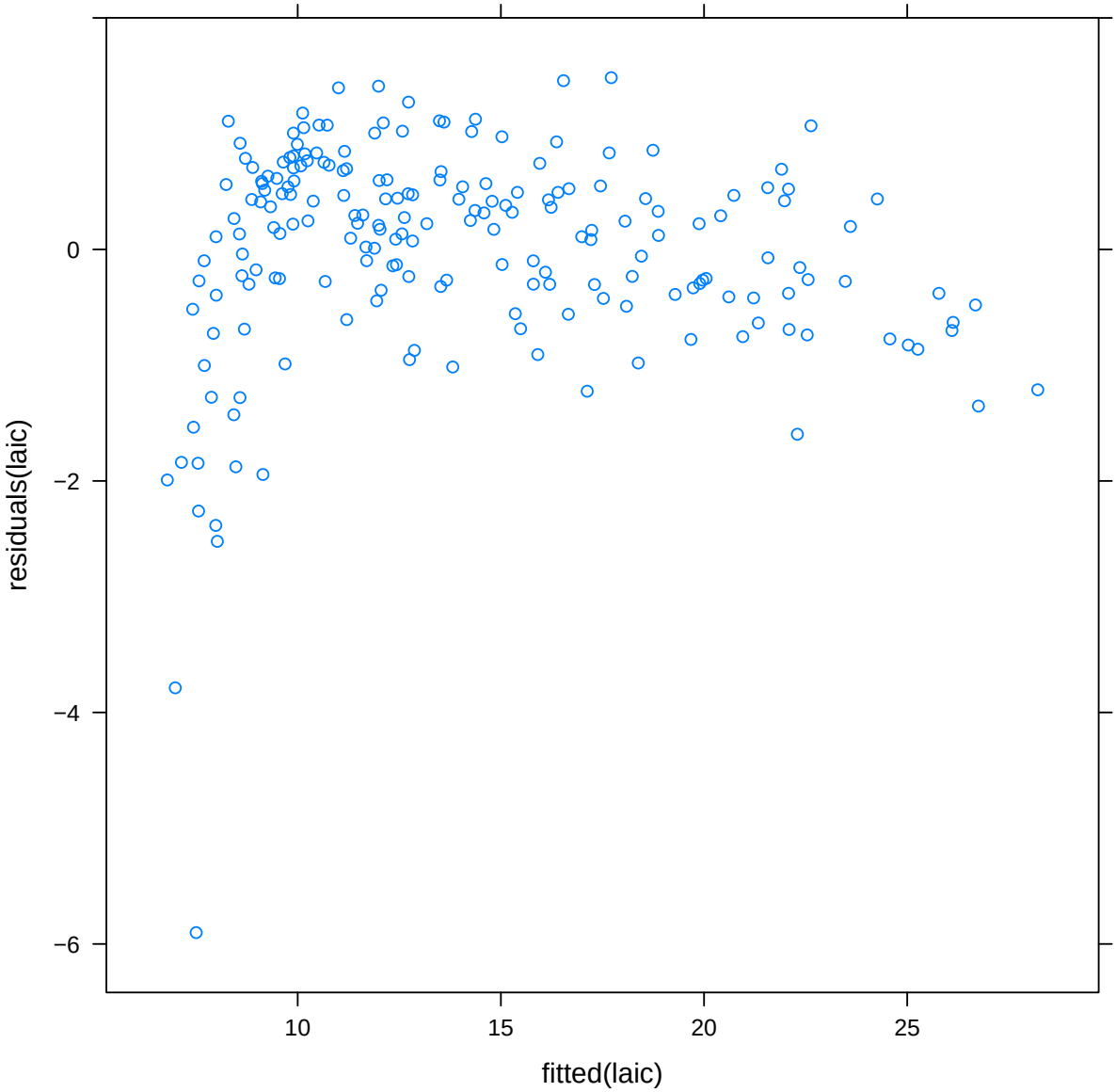
```
## Start:  AIC=-18.59
## Sales ~ (TV + Radio + Newspaper)^2
##
```

```
##                    Df Sum of Sq    RSS     AIC
## - Radio:Newspaper  1      0.19 170.12 -20.363
## <none>                         169.93 -18.586
## - TV:Newspaper     1      4.37 174.30 -15.511
## - TV:Radio         1    349.71 519.64 202.965
##
## Step:  AIC=-20.36
## Sales ~ TV + Radio + Newspaper + TV:Radio + TV:Newspaper
##
##                 Df Sum of Sq    RSS     AIC
## <none>                       170.12 -20.363
## - TV:Newspaper  1      4.19 174.31 -17.494
## - TV:Radio      1    352.83 522.95 202.234
```

```r
summary(laic)
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper + TV:Radio + TV:Newspaper,
##     data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.9019 -0.3818  0.1937  0.5741  1.4839
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.541e+00  2.652e-01  24.668   <2e-16 ***
## TV            2.035e-02  1.605e-03  12.675   <2e-16 ***
## Radio         2.018e-02  9.734e-03   2.073   0.0395 *
## Newspaper     1.342e-02  6.377e-03   2.105   0.0366 *
## TV:Radio      1.136e-03  5.664e-05  20.059   <2e-16 ***
## TV:Newspaper -7.719e-05  3.531e-05  -2.187   0.0300 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9364 on 194 degrees of freedom
## Multiple R-squared:  0.9686,Adjusted R-squared:  0.9678
## F-statistic:  1197 on 5 and 194 DF,  p-value: < 2.2e-16
```

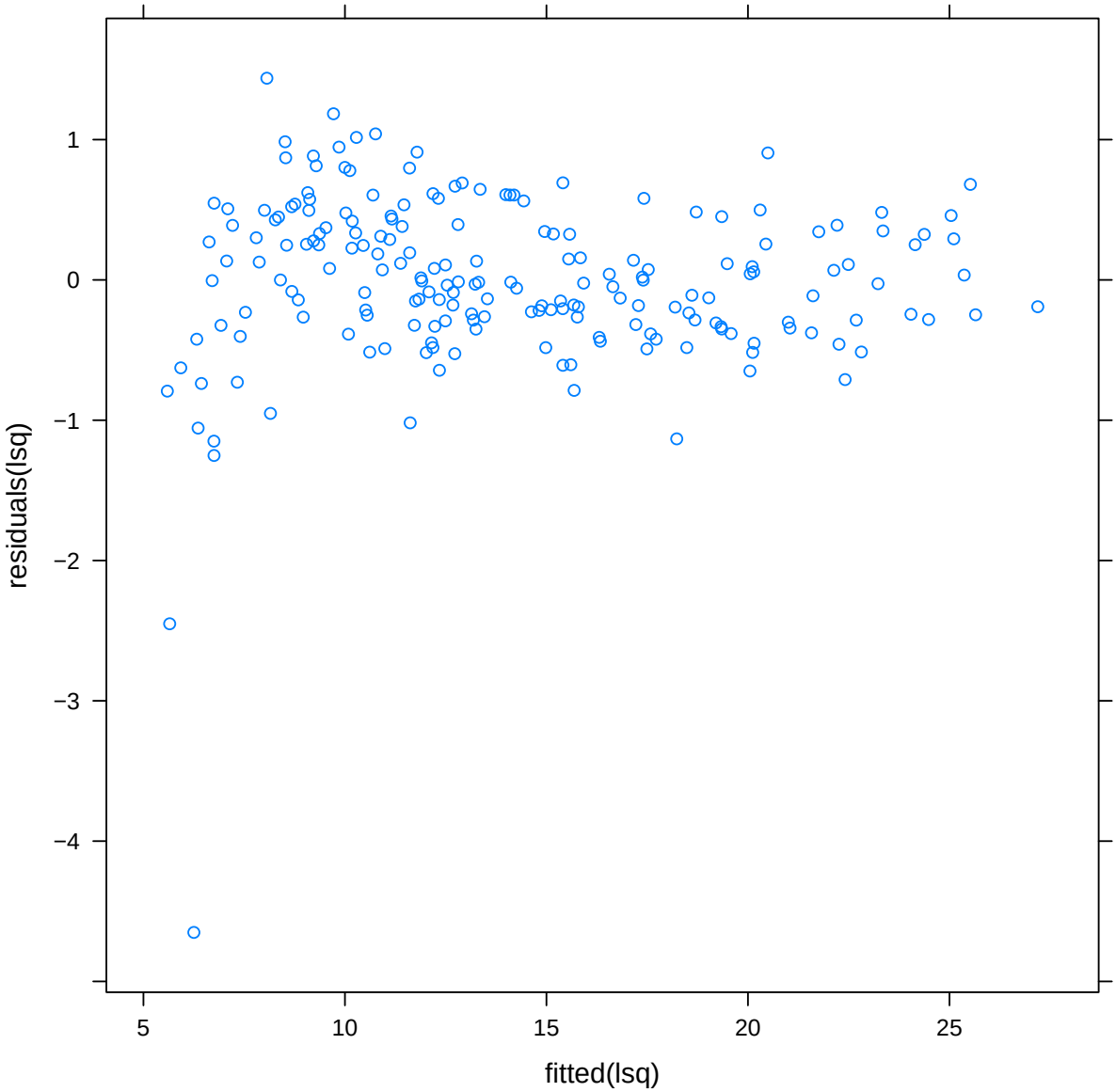
```r
xyplot(residuals(laic) ~ fitted(laic))
```




```
lsq <- lm(Sales ~ poly(TV, Radio, Newspaper,
    degree = 2), data = Advertising)
summary(lsq)

##
## Call:
## lm(formula = Sales ~ poly(TV, Radio, Newspaper, degree = 2),
##     data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.6507 -0.2941 -0.0060  0.3830  1.4378
##
## Coefficients:
##                                                Estimate
```

```
## (Intercept)                                      13.94105
## poly(TV, Radio, Newspaper, degree = 2)1.0.0  53.73032
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -9.97989
## poly(TV, Radio, Newspaper, degree = 2)0.1.0  40.10484
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 280.35751
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.29727
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   0.94373
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 -16.93577
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   5.31882
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.10711
##                                              Std. Error
## (Intercept)                                     0.04806
## poly(TV, Radio, Newspaper, degree = 2)1.0.0    0.62580
## poly(TV, Radio, Newspaper, degree = 2)2.0.0    0.63251
## poly(TV, Radio, Newspaper, degree = 2)0.1.0    0.67044
## poly(TV, Radio, Newspaper, degree = 2)1.1.0    9.65318
## poly(TV, Radio, Newspaper, degree = 2)0.2.0    0.64859
## poly(TV, Radio, Newspaper, degree = 2)0.0.1    0.74141
## poly(TV, Radio, Newspaper, degree = 2)1.0.1    8.84414
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   10.95328
## poly(TV, Radio, Newspaper, degree = 2)0.0.2    0.65997
##                                              t value
## (Intercept)                                  290.083
## poly(TV, Radio, Newspaper, degree = 2)1.0.0  85.858
## poly(TV, Radio, Newspaper, degree = 2)2.0.0 -15.778
## poly(TV, Radio, Newspaper, degree = 2)0.1.0  59.819
## poly(TV, Radio, Newspaper, degree = 2)1.1.0  29.043
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.458
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   1.273
## poly(TV, Radio, Newspaper, degree = 2)1.0.1  -1.915
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   0.486
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.162
##                                              Pr(>|t|)
## (Intercept)                                    <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.0.0    <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)2.0.0    <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.1.0    <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.1.0    <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.2.0     0.647
## poly(TV, Radio, Newspaper, degree = 2)0.0.1     0.205
## poly(TV, Radio, Newspaper, degree = 2)1.0.1     0.057 .
## poly(TV, Radio, Newspaper, degree = 2)0.1.1     0.628
## poly(TV, Radio, Newspaper, degree = 2)0.0.2     0.871
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6203 on 190 degrees of freedom
```

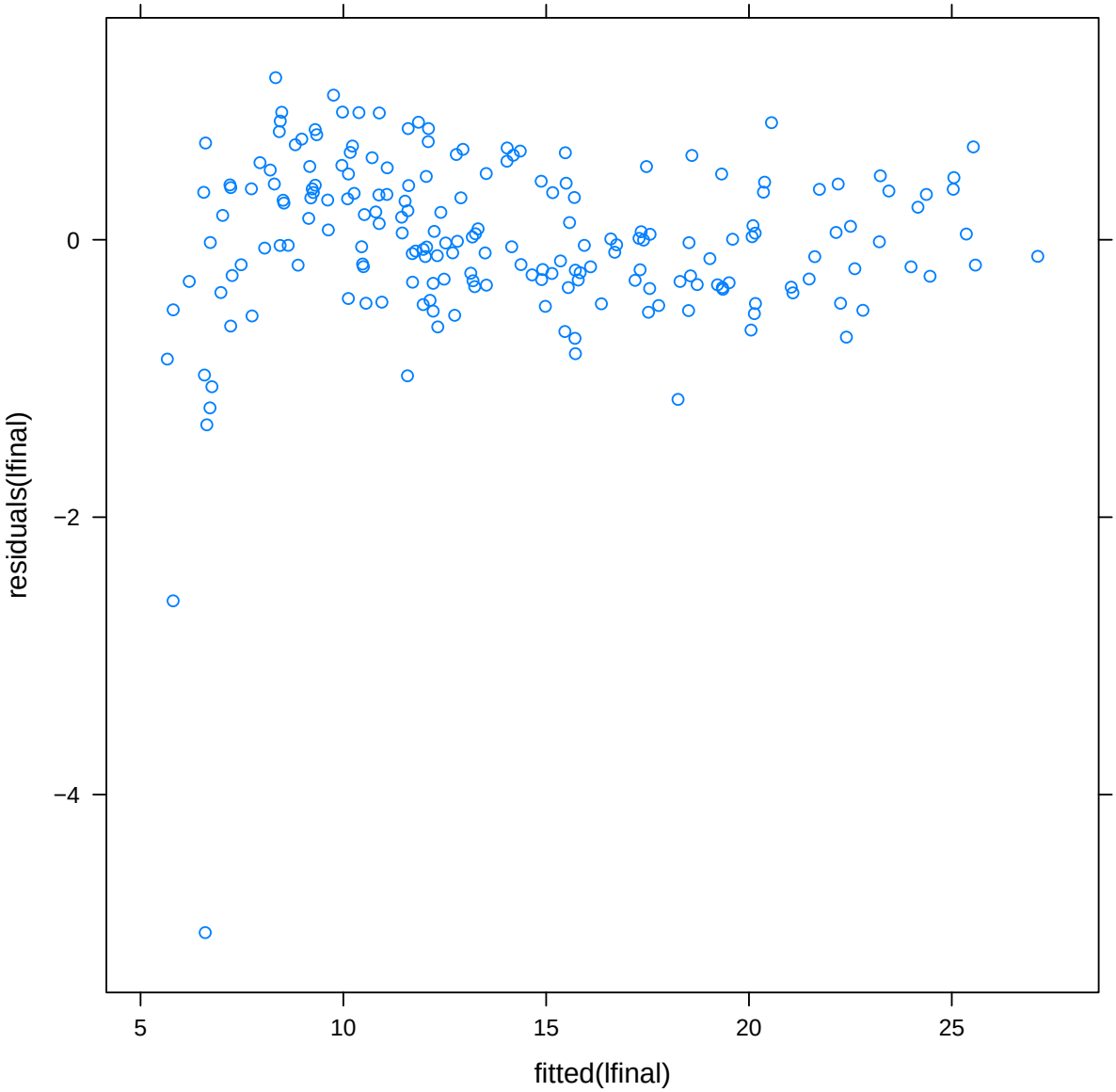
```
## Multiple R-squared:  0.9865,Adjusted R-squared:  0.9859
## F-statistic:  1543 on 9 and 190 DF,  p-value: < 2.2e-16

xyplot(residuals(lsq) ~ fitted(lsq))
```




```
lsqaic <- stepAIC(lsq)

## Start:  AIC=-181.3
## Sales ~ poly(TV, Radio, Newspaper, degree = 2)
##
##                                          Df Sum of Sq
## <none>
## - poly(TV, Radio, Newspaper, degree = 2)  9      5344
##                                            RSS    AIC
## <none>                                    73.1 -181.3
```

```
## - poly(TV, Radio, Newspaper, degree = 2) 5417.1  661.8

summary(lsqaic)

##
## Call:
## lm(formula = Sales ~ poly(TV, Radio, Newspaper, degree = 2),
##     data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.6507 -0.2941 -0.0060  0.3830  1.4378
##
## Coefficients:
##                                              Estimate
## (Intercept)                                  13.94105
## poly(TV, Radio, Newspaper, degree = 2)1.0.0  53.73032
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -9.97989
## poly(TV, Radio, Newspaper, degree = 2)0.1.0  40.10484
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 280.35751
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.29727
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   0.94373
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 -16.93577
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   5.31882
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.10711
##                                             Std. Error
## (Intercept)                                    0.04806
## poly(TV, Radio, Newspaper, degree = 2)1.0.0    0.62580
## poly(TV, Radio, Newspaper, degree = 2)2.0.0    0.63251
## poly(TV, Radio, Newspaper, degree = 2)0.1.0    0.67044
## poly(TV, Radio, Newspaper, degree = 2)1.1.0    9.65318
## poly(TV, Radio, Newspaper, degree = 2)0.2.0    0.64859
## poly(TV, Radio, Newspaper, degree = 2)0.0.1    0.74141
## poly(TV, Radio, Newspaper, degree = 2)1.0.1    8.84414
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   10.95328
## poly(TV, Radio, Newspaper, degree = 2)0.0.2    0.65997
##                                             t value
## (Intercept)                                 290.083
## poly(TV, Radio, Newspaper, degree = 2)1.0.0  85.858
## poly(TV, Radio, Newspaper, degree = 2)2.0.0 -15.778
## poly(TV, Radio, Newspaper, degree = 2)0.1.0  59.819
## poly(TV, Radio, Newspaper, degree = 2)1.1.0  29.043
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.458
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   1.273
## poly(TV, Radio, Newspaper, degree = 2)1.0.1  -1.915
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   0.486
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.162
##                                             Pr(>|t|)
```

```
## (Intercept)                                      <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.0.0   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)2.0.0   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.1.0   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.1.0   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.2.0    0.647
## poly(TV, Radio, Newspaper, degree = 2)0.0.1    0.205
## poly(TV, Radio, Newspaper, degree = 2)1.0.1    0.057 .
## poly(TV, Radio, Newspaper, degree = 2)0.1.1    0.628
## poly(TV, Radio, Newspaper, degree = 2)0.0.2    0.871
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6203 on 190 degrees of freedom
## Multiple R-squared:  0.9865,Adjusted R-squared:  0.9859
## F-statistic:  1543 on 9 and 190 DF,  p-value: < 2.2e-16

lfinal <- lm(Sales ~ (TV + Radio)^2 + I(TV^2),
    data = Advertising)
summary(lfinal)

##
## Call:
## lm(formula = Sales ~ (TV + Radio)^2 + I(TV^2), data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.9949 -0.2969 -0.0066  0.3798  1.1686
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.137e+00  1.927e-01  26.663  < 2e-16 ***
## TV           5.092e-02  2.232e-03  22.810  < 2e-16 ***
## Radio        3.516e-02  5.901e-03   5.959 1.17e-08 ***
## I(TV^2)     -1.097e-04  6.893e-06 -15.920  < 2e-16 ***
## TV:Radio     1.077e-03  3.466e-05  31.061  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6238 on 195 degrees of freedom
## Multiple R-squared:  0.986,Adjusted R-squared:  0.9857
## F-statistic:  3432 on 4 and 195 DF,  p-value: < 2.2e-16

xyplot(residuals(lfinal) ~ fitted(lfinal))
```




```
tune(lm, lfinal$call$formula, data = Advertising,
    tunecontrol = tune.control(sampling = "fix"))

##
## Error estimation of 'lm' using fixed training/validation set: 0.6253161

tune(lm, l$call$formula, data = Advertising,
    tunecontrol = tune.control(sampling = "fix"))

##
## Error estimation of 'lm' using fixed training/validation set: 3.791728

tune(lm, ltvradio$call$formula, data = Advertising,
    tunecontrol = tune.control(sampling = "fix"))
```

```
##
## Error estimation of 'lm' using fixed training/validation set: 1.1799

tune(lm, li$call$formula, data = Advertising,
    tunecontrol = tune.control(sampling = "fix"))

##
## Error estimation of 'lm' using fixed training/validation set: 1.202255
```

5 Материалы с занятия 10 октября

Вспомогательнй код для классификации

```
library(MASS)  # AIC(), BIC(), lda(), qda()
library(lattice)  # xyplot(), densityplot()
library(latticeExtra)  # layer()

## Loading required package: RColorBrewer

library(ROCR)  # performance(), prediction()

## Loading required package: gplots
##
## Attaching package: 'gplots'
##
## The following object is masked from 'package:stats':
##
##    lowess
##
## Loading required package: methods

# library(caret) # specificity(),
# sensitivity()
library(nnet)  # multinom()
library(e1071)  # naiveBayes(), tune()
specificity <- caret:::specificity
sensitivity <- caret:::sensitivity
ROC <- function(predicted, actual, ...) {
    pred <- prediction(predicted, as.numeric(actual))
    roc <- performance(pred, measure = "tpr",
        x.measure = "fpr", ...)
    roc
}
xyplot.performance <- function(x, ...) {
    xyplot(x@y.values[[1]] ~ x@x.values[[1]],
        xlab = x@x.name, ylab = x@y.name,
        type = "l", ...) + layer_(abline(a = 0,
        b = 1, col = "red"))
```

```
}
AUC <- function(predicted, actual, ...) {
    pred <- prediction(predicted, as.numeric(actual))
    perf <- performance(pred, measure = "auc",
        ...)
    perf@y.values[[1]]
}
roc.opt <- function(predicted, actual, cutoff = NULL,
    measure = c("mean", "max", "err")) {
    pred <- prediction(predicted, as.numeric(actual))
    perf <- performance(pred, measure = "fpr",
        x.measure = "fnr")
    measure <- match.arg(measure)
    fpr <- perf@y.values[[1]]
    fnr <- perf@x.values[[1]]
    npos <- pred@n.pos[[1]]
    nneg <- pred@n.neg[[1]]
    err <- (fpr * nneg + fnr * npos)/(npos +
        nneg)
    error.rate <- switch(measure, mean = (fpr +
        fnr)/2, max = pmax(fpr, fnr), err = err)
    if (is.null(cutoff)) {
        i <- which.min(error.rate)
    } else {
        i <- which.min(abs(perf@alpha.values[[1]] -
            cutoff))
    }
    list(cutoff = perf@alpha.values[[1]][i],
        fpr = fpr[i], fnr = fnr[i], err = err[i],
        error.rate = error.rate[i])
}
simple.predict.glm <- function(x, newdata,
    ...) {
    response <- predict(x, newdata, type = "response",
        ...)
    factor(levels(x$model[, 1])[1 + as.integer(response >
        0.5)])
}
my.predict.glm <- function(x, newdata = x$data,
    ..., measure = "max") {
    opt <- roc.opt(fitted(x), as.numeric(x$model[,
        1]), measure = measure)
    cutoff <- opt$cutoff
    factor(as.integer(predict(x, newdata = newdata,
        type = "response") > cutoff), labels = levels(x$model[,
        1]))
}
error.fun.max <- function(true, predicted) {
```

```
    1 - min(sensitivity(predicted, true),
        specificity(predicted, true))
}
error.fun.mean <- function(true, predicted) {
    1 - mean(sensitivity(predicted, true),
        specificity(predicted, true))
}
my.lda <- function(x, data, ...) {
    out <- lda(x, data, ...)
    out$data <- data
    out
}
my.qda <- function(x, data, ...) {
    out <- qda(x, data, ...)
    out$data <- data
    out
}
simple.predict.da <- function(...) predict(...)$class
my.predict.da <- function(x, newdata, cutoff.data = x$data,
    ..., measure = "max") {
    response <- model.frame(x$terms, cutoff.data)[,
        1]
    opt <- roc.opt(predict(x, cutoff.data)$posterior[,
        2], as.numeric(response), measure = measure)
    cutoff <- opt$cutoff
    factor(as.integer(predict(x, newdata = newdata)$posterior[,
        2] > cutoff), labels = levels(response))
}
```

5.1 LDA и tune

```
library(MASS)
library(lattice)
library(latticeExtra)
library(ROCR)
library(e1071)
ld <- lda(Species ~ ., data = iris)
ld

## Call:
## lda(Species ~ ., data = iris)
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
```

```
## Group means:
##            Sepal.Length Sepal.Width Petal.Length
## setosa            5.006       3.428        1.462
## versicolor        5.936       2.770        4.260
## virginica         6.588       2.974        5.552
##            Petal.Width
## setosa           0.246
## versicolor       1.326
## virginica        2.026
##
## Coefficients of linear discriminants:
##                     LD1         LD2
## Sepal.Length  0.8293776  0.02410215
## Sepal.Width   1.5344731  2.16452123
## Petal.Length -2.2012117 -0.93192121
## Petal.Width  -2.8104603  2.83918785
##
## Proportion of trace:
##    LD1    LD2
## 0.9912 0.0088
```

```
train.idx <- sample(nrow(iris), size = nrow(iris) *
    0.6)
iris.train <- iris[train.idx, ]
iris.test <- iris[-train.idx, ]
ld <- lda(Species ~ ., data = iris.train,
    prior = c(1/3, 1/3, 1/3))
ld
```

```
## Call:
## lda(Species ~ ., data = iris.train, prior = c(1/3, 1/3, 1/3))
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length
## setosa         5.025000    3.389286     1.446429
## versicolor     5.923529    2.773529     4.247059
## virginica      6.550000    3.000000     5.532143
##            Petal.Width
## setosa       0.2464286
## versicolor   1.3117647
## virginica    2.0357143
##
## Coefficients of linear discriminants:
##                     LD1        LD2
```

```
## Sepal.Length  0.6535275 -0.6210853
## Sepal.Width   1.9036507 -1.6906044
## Petal.Length -2.3435547  1.6694124
## Petal.Width  -2.8281153 -3.8409118
##
## Proportion of trace:
##    LD1    LD2
## 0.9902 0.0098
```

```
table(predicted = predict(ld, iris.test)$class,
    actual = iris.test$Species)
```

```
##             actual
## predicted    setosa versicolor virginica
##   setosa         22          0         0
##   versicolor      0         15         0
##   virginica       0          1        22
```

```
table(predicted = predict(ld, iris.test,
    prior = c(0.5, 0.25, 0.25))$class, actual = iris.test$Species)
```

```
##             actual
## predicted    setosa versicolor virginica
##   setosa         22          0         0
##   versicolor      0         15         0
##   virginica       0          1        22
```

```
cm <- table(predicted = predict(ld, iris.test)$class,
    actual = iris.test$Species)
chisq.test(cm)
```

```
## Warning in chisq.test(cm): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  cm
## X-squared = 113.8043, df = 4, p-value < 2.2e-16
```

```
# test-train
tune(lda, Species ~ ., prior = c(1/3, 1/3,
    1/3), data = iris, predict.func = function(...) predict(...)$class,
    tunecontrol = tune.control(sampling = "fix",
        fix = 2/3))
```

```
##
## Error estimation of 'lda' using fixed training/validation set: 0
```

```r
simple.predict.da <- function(...) predict(...)$class
# bootstrap
tune(lda, Species ~ ., prior = c(1/3, 1/3,
    1/3), data = iris, predict.func = simple.predict.da,
    tunecontrol = tune.control(sampling = "bootstrap",
        nboot = 10, boot.size = 9/10))

##
## Error estimation of 'lda' using bootstrapping: 0.02151559

# leave-one-out
tune(lda, Species ~ ., prior = c(1/3, 1/3,
    1/3), data = iris, predict.func = simple.predict.da,
    tunecontrol = tune.control(sampling = "cross",
        cross = nrow(iris)))

##
## Error estimation of 'lda' using leave-one-out: 0.02

# cross-validationi (default)
tn <- tune(lda, Species ~ ., prior = c(1/3,
    1/3, 1/3), data = iris, predict.func = simple.predict.da,
    tunecontrol = tune.control(sampling = "cross",
        cross = 10))
tn$best.model

## Call:
## best.tune(lda, train.x = Species ~ ., data = iris, predict.func = simple.predict
##     tunecontrol = tune.control(sampling = "cross", cross = 10),
##     prior = c(1/3, 1/3, 1/3))
##
## Prior probabilities of groups:
##     setosa versicolor  virginica
##  0.3333333  0.3333333  0.3333333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length
## setosa            5.006       3.428        1.462
## versicolor        5.936       2.770        4.260
## virginica         6.588       2.974        5.552
##            Petal.Width
## setosa           0.246
## versicolor       1.326
## virginica        2.026
##
## Coefficients of linear discriminants:
##                     LD1         LD2
## Sepal.Length  0.8293776  0.02410215
## Sepal.Width   1.5344731  2.16452123
```

```
## Petal.Length -2.2012117 -0.93192121
## Petal.Width  -2.8104603  2.83918785
##
## Proportion of trace:
##    LD1    LD2
## 0.9912 0.0088
```

```r
tn$performance
```

```
##   dummyparameter error dispersion
## 1              0  0.02 0.03220306
```

```r
# tn$train.ind
# Naive Bayes
nb <- naiveBayes(Species ~ ., data = iris)
tn.nb <- tune(naiveBayes, Species ~ ., data = iris)
# multinomial regression
mln <- multinom(Species ~ ., data = iris,
    trace = FALSE)
tn.mln <- tune(multinom, Species ~ ., data = iris,
    trace = FALSE)
summary(mln)
```

```
## Call:
## multinom(formula = Species ~ ., data = iris, trace = FALSE)
##
## Coefficients:
##            (Intercept) Sepal.Length Sepal.Width
## versicolor    18.69037    -5.458424   -8.707401
## virginica    -23.83628    -7.923634  -15.370769
##            Petal.Length Petal.Width
## versicolor     14.24477   -3.097684
## virginica      23.65978   15.135301
##
## Std. Errors:
##            (Intercept) Sepal.Length Sepal.Width
## versicolor    34.97116     89.89215    157.0415
## virginica     35.76649     89.91153    157.1196
##            Petal.Length Petal.Width
## versicolor     60.19170    45.48852
## virginica      60.46753    45.93406
##
## Residual Deviance: 11.89973
## AIC: 31.89973
```

```r
mln.aic <- stepAIC(mln)
```

```
## Start:  AIC=31.9
## Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
```

```
##
##                Df    AIC
## - Sepal.Length  2 29.267
## - Sepal.Width   2 31.498
## <none>            31.900
## - Petal.Width   2 39.773
## - Petal.Length  2 41.915
##
## Step:  AIC=29.27
## Species ~ Sepal.Width + Petal.Length + Petal.Width
##
##                Df    AIC
## <none>            29.267
## - Sepal.Width   2 32.579
## - Petal.Length  2 39.399
## - Petal.Width   2 43.516
```

```
summary(mln.aic)
```

```
## Call:
## multinom(formula = Species ~ Sepal.Width + Petal.Length + Petal.Width,
##     data = iris, trace = FALSE)
##
## Coefficients:
##            (Intercept) Sepal.Width Petal.Length Petal.Width
## versicolor    14.15646   -17.32240     14.09906   -2.695628
## virginica    -36.44078   -25.70717     21.98210   18.765796
##
## Std. Errors:
##            (Intercept) Sepal.Width Petal.Length Petal.Width
## versicolor    29.66211    47.48205     68.57820    39.08345
## virginica     32.18618    48.00257     68.76678    39.75433
##
## Residual Deviance: 13.26653
## AIC: 29.26653
```

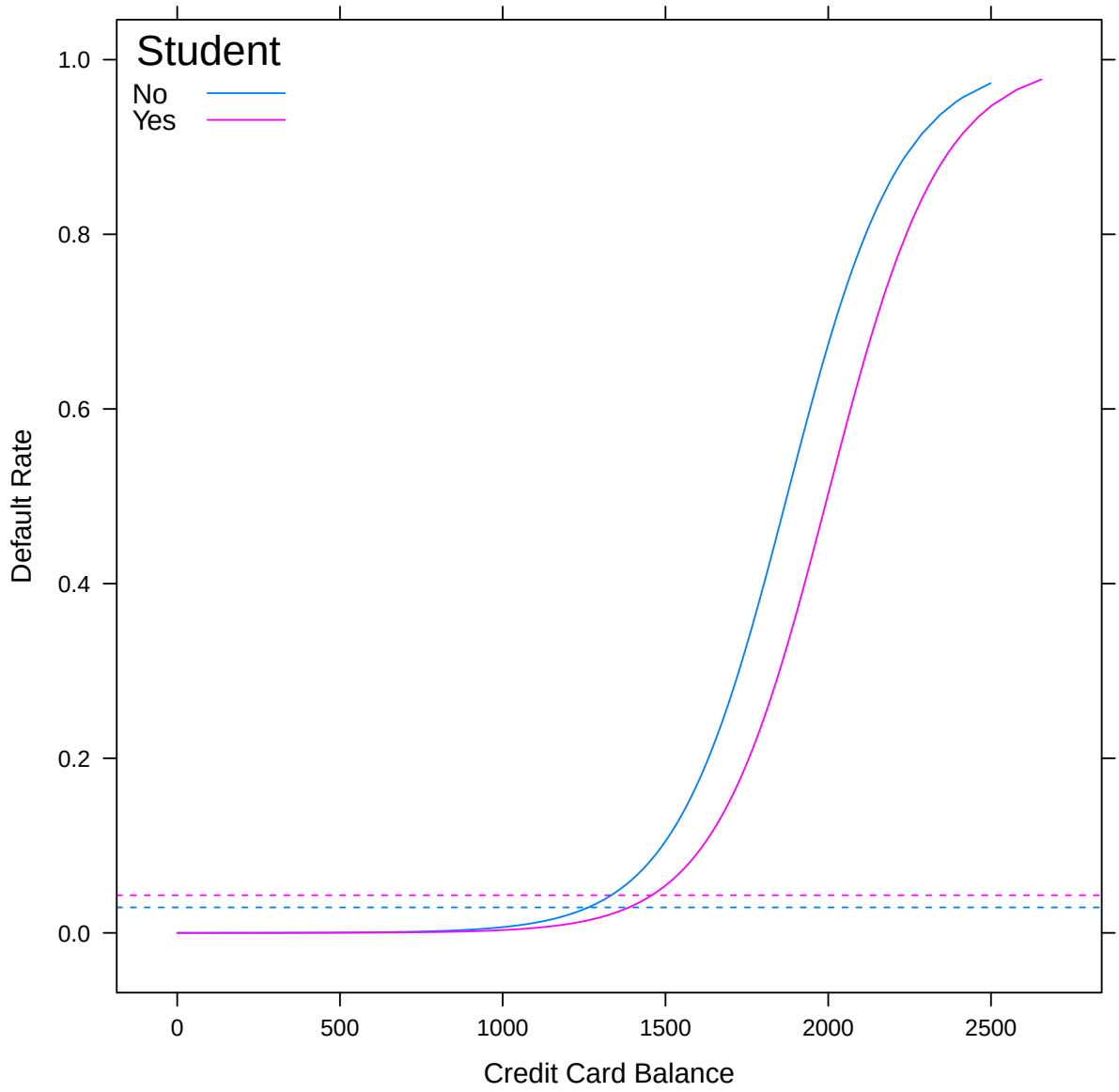
5.2 Default

```
library(MASS)
library(lattice)
library(latticeExtra)
library(ISLR)
```

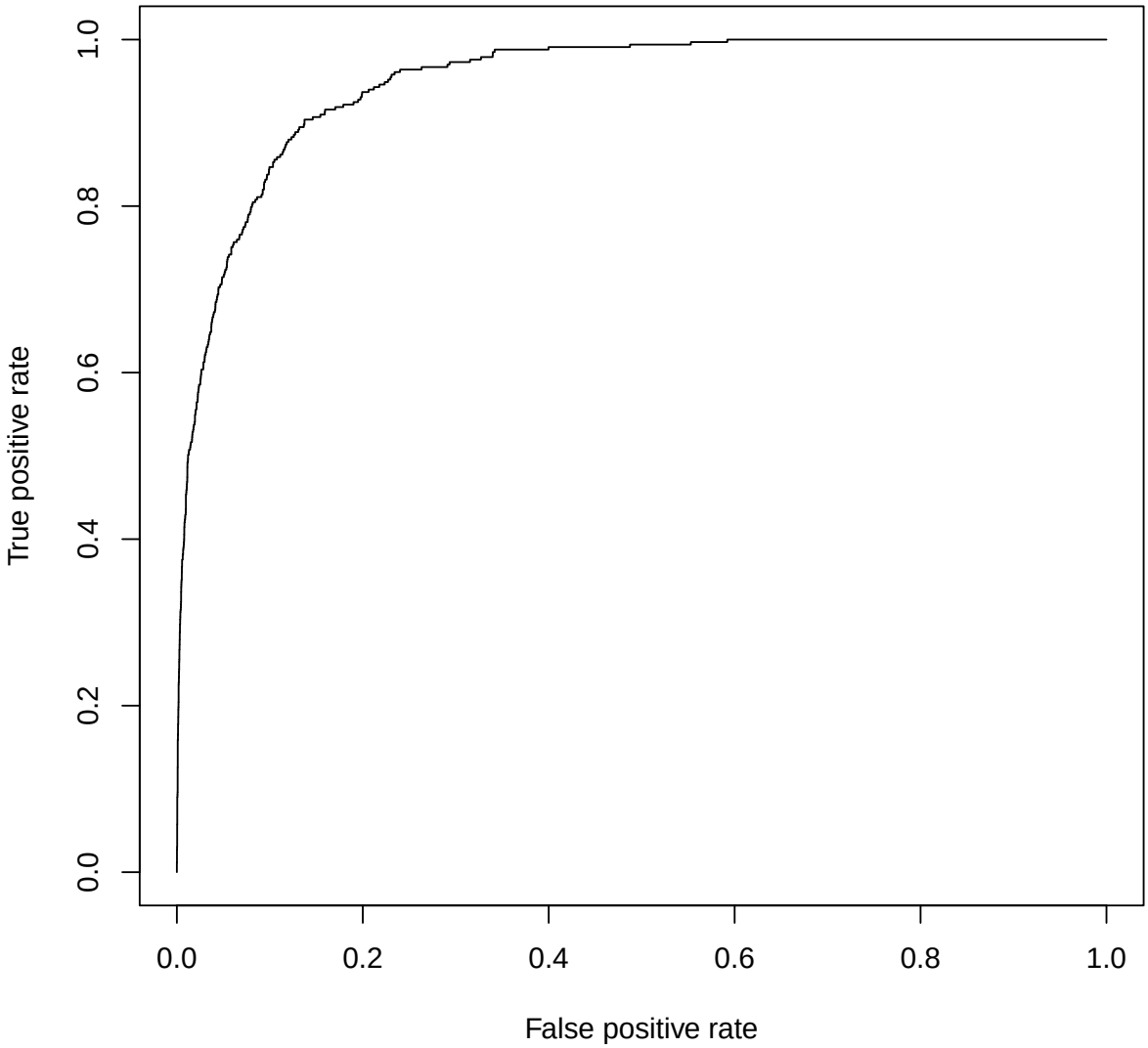
```
##
## Attaching package: 'ISLR'
##
## The following object is masked _by_ '.GlobalEnv':
```

```
##
##    Auto
##
## The following object is masked from 'package:vcd':
##
##    Hitters
```

```r
source("class.R")
data(Default)
gl <- glm(default ~ balance + student, data = Default,
    family = binomial(link = "logit"))
Default.sorted <- Default[order(Default$balance),
    ]
xyplot(predict(gl, Default.sorted, type = "response") ~
    balance, groups = student, data = Default.sorted,
    type = "l", auto.key = list(corner = c(0,
        1), title = "Student", lines = TRUE,
        points = FALSE), xlab = "Credit Card Balance",
    ylab = "Default Rate") + layer_(panel.superpose(x = Default.sorted$default ==
    "Yes", panel.groups = function(x, y,
    ...) panel.abline(h = mean(x), ...),
    lty = "dashed", ...))
```




```
roc <- ROC(predict(gl, Default), Default$default)
plot(roc)
```




```
AUC(predict(gl, Default), Default$default)

## [1] 0.9495476

error.fun.auc <- function(true, predicted) {
    -AUC(predicted, true)
}
tune(glm, default ~ ., data = Default, family = binomial(link = "logit"),
    tunecontrol = tune.control(error.fun = error.fun.auc))

##
## Error estimation of 'glm' using 10-fold cross validation: -0.9490864

table(predicted = my.predict.glm(gl, measure = "max"),
    actual = Default$default)
```

```
##          actual
## predicted   No  Yes
##       No  8507   41
##       Yes 1160  292
```

```r
table(predicted = my.predict.glm(gl, measure = "mean"),
    actual = Default$default)
```

```
##          actual
## predicted   No  Yes
##       No  8340   33
##       Yes 1327  300
```

```r
table(predicted = my.predict.glm(gl, measure = "err"),
    actual = Default$default)
```

```
##          actual
## predicted   No  Yes
##       No  9613  209
##       Yes   54  124
```

```r
train <- sample(nrow(Default), size = 0.6 *
    nrow(Default))
gl <- glm(default ~ balance + student, data = Default,
    subset = train, family = binomial(link = "logit"))
table(predicted = my.predict.glm(gl, Default[-train,
    ], measure = "max"), actual = Default[-train,
    ]$default)
```

```
##          actual
## predicted   No  Yes
##       No  3387   12
##       Yes  480  121
```

```r
table(predicted = my.predict.glm(gl, Default[-train,
    ], measure = "mean"), actual = Default[-train,
    ]$default)
```

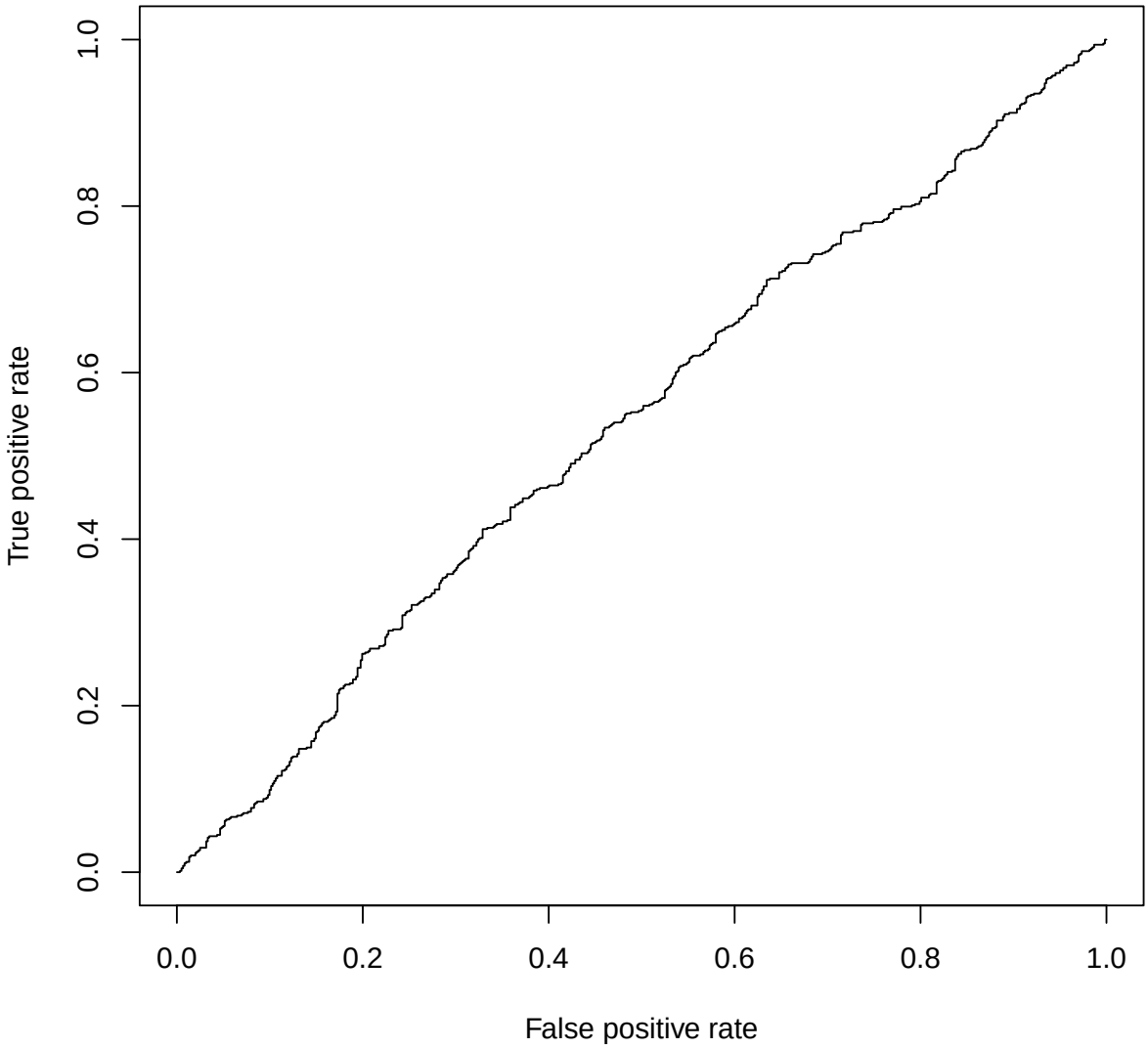
```
##          actual
## predicted   No  Yes
##       No  3282   11
##       Yes  585  122
```

```r
table(predicted = my.predict.glm(gl, Default[-train,
    ], measure = "err"), actual = Default[-train,
    ]$default)
```

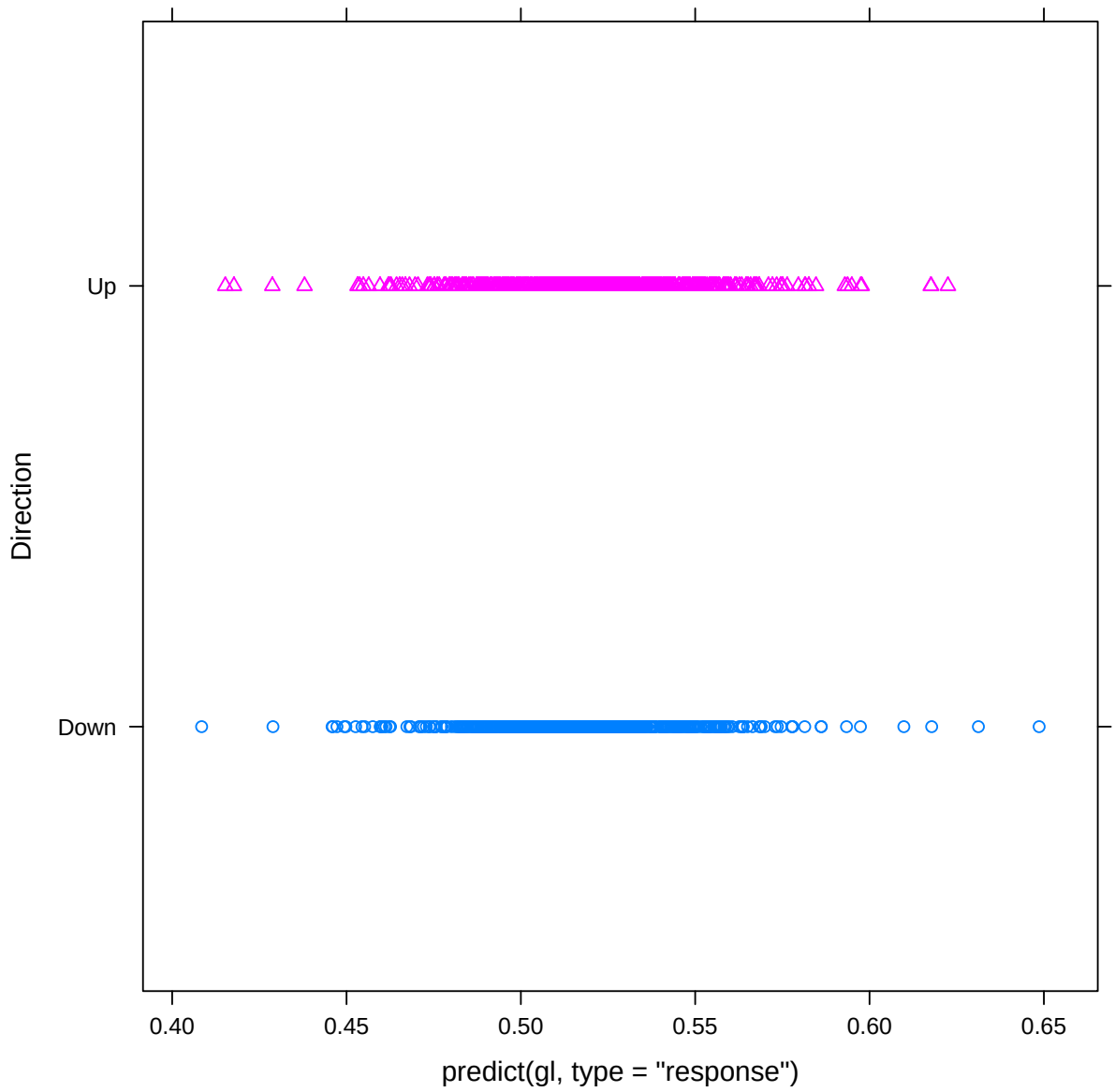
```
##          actual
## predicted   No  Yes
##       No  3839   83
##       Yes   28   50
```

5.3 Smarket

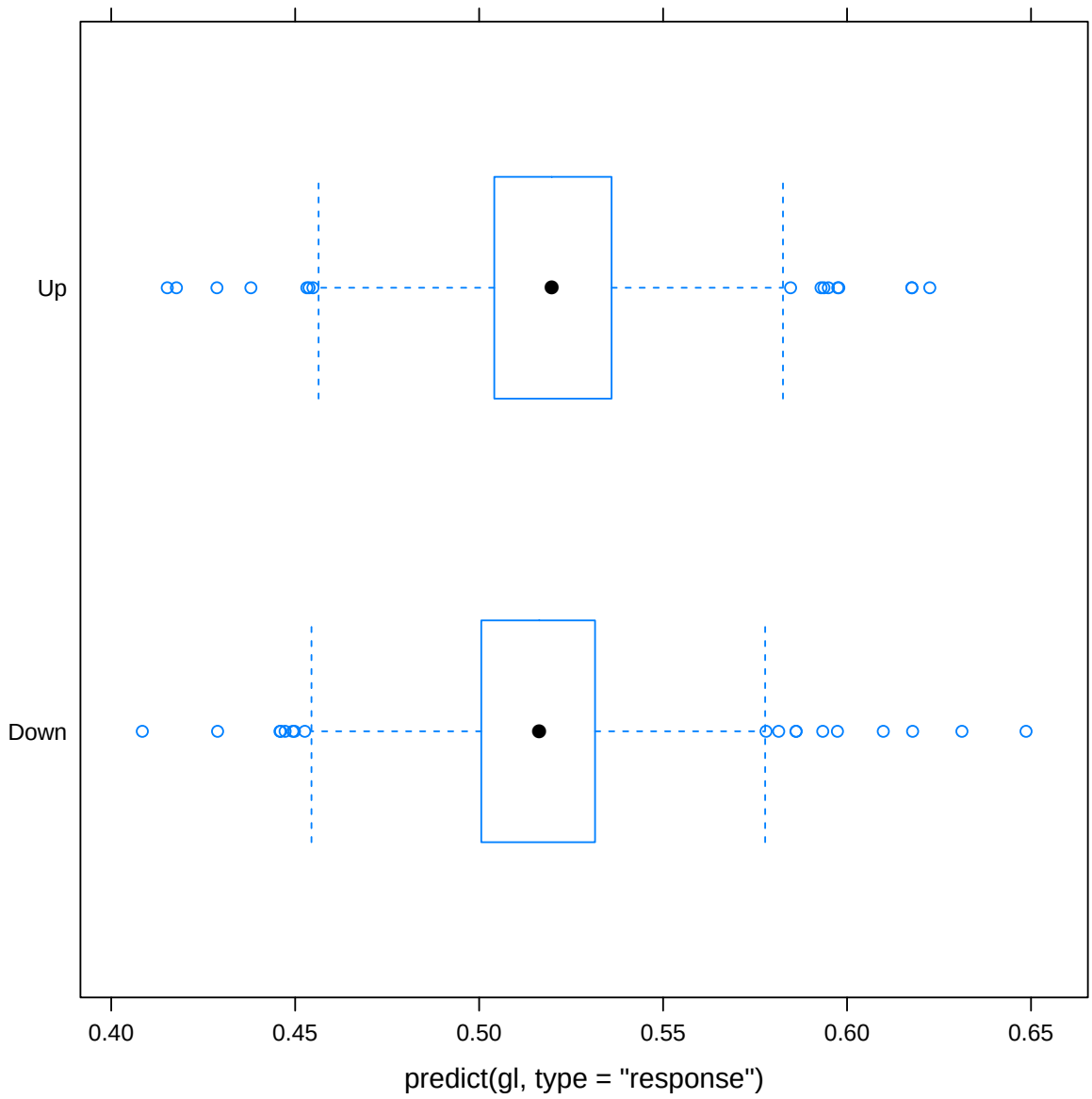
```r
library(ISLR)
library(MASS)
data(Smarket)
gl <- glm(Direction ~ Volume + Lag1 + Lag2 +
    Lag3 + Lag4 + Lag5, data = Smarket, family = binomial(link = "logit"))
summary(gl)

##
## Call:
## glm(formula = Direction ~ Volume + Lag1 + Lag2 + Lag3 + Lag4 +
##     Lag5, family = binomial(link = "logit"), data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.446  -1.203   1.065   1.145   1.326
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Volume       0.135441   0.158360   0.855    0.392
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3

source("class.R")
roc <- ROC(predict(gl), Smarket$Direction)
plot(roc)
```



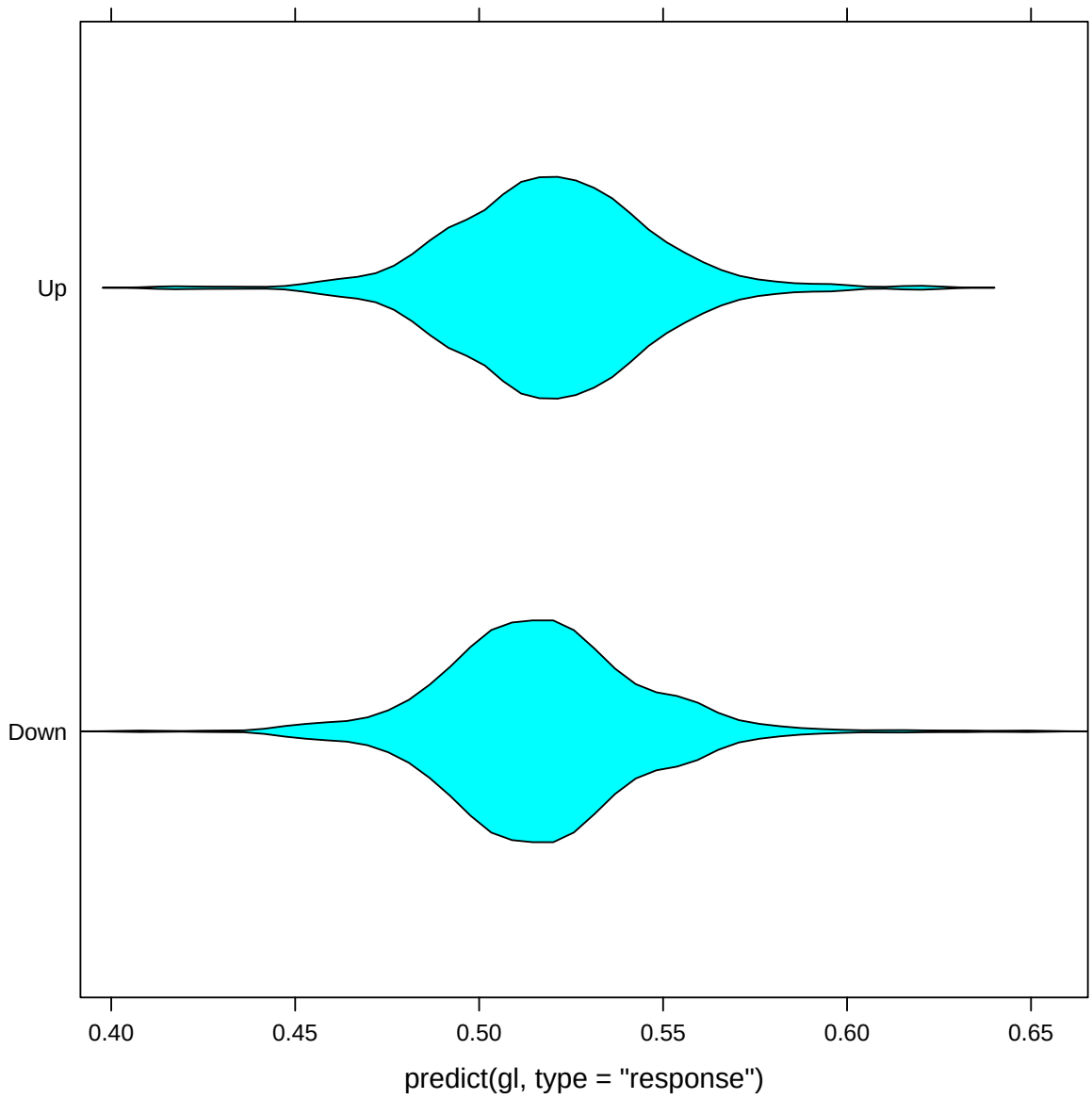

```
AUC(predict(gl), Smarket$Direction)

## [1] 0.5387341

xyplot(Direction ~ predict(gl, type = "response"),
    data = Smarket, groups = Direction, par.settings = simpleTheme(pch = 1:2))
```




```
bwplot(Direction ~ predict(gl, type = "response"),
    data = Smarket)
```

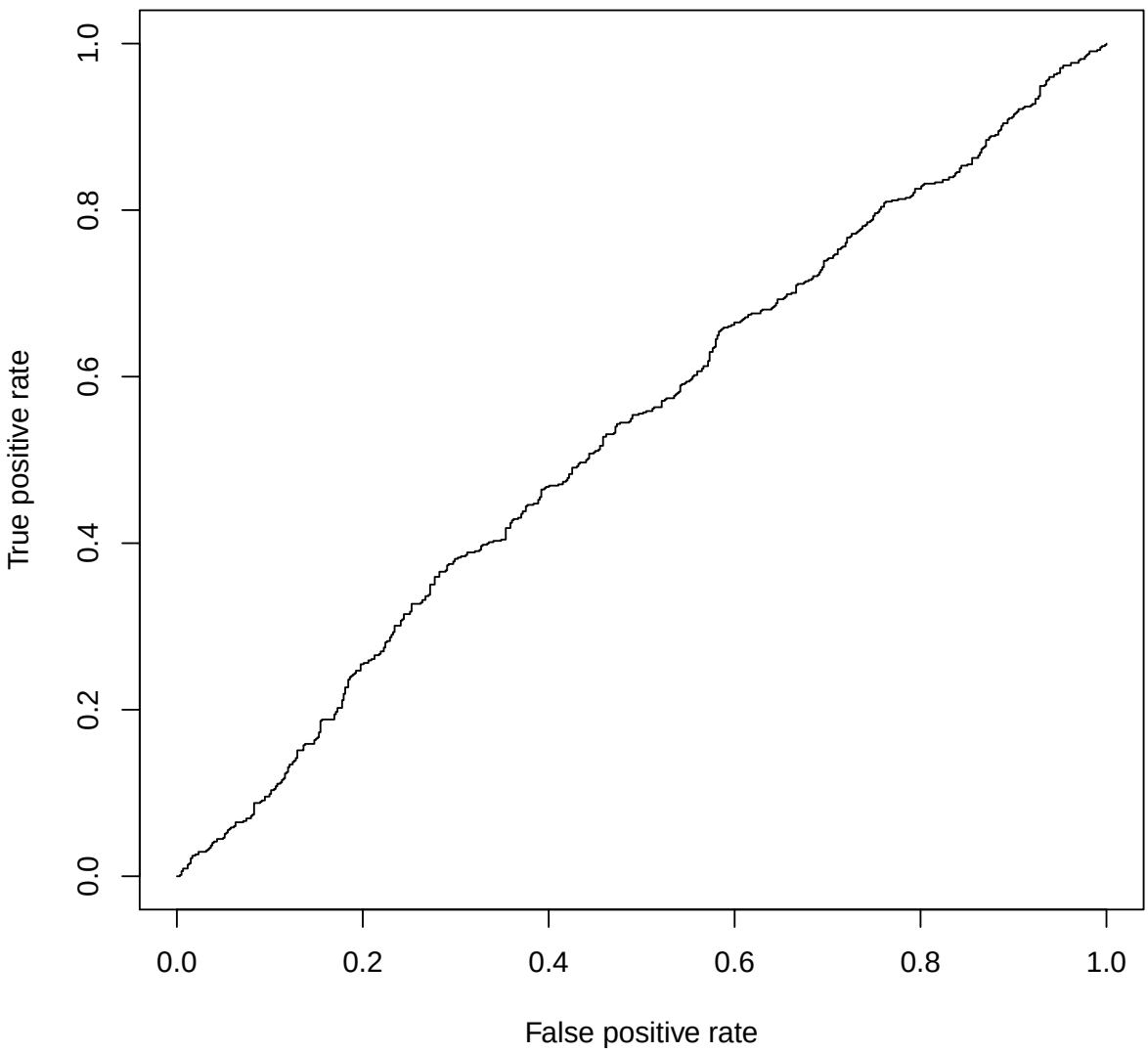



```
bwplot(Direction ~ predict(gl, type = "response"),
    data = Smarket, panel = panel.violin)
```

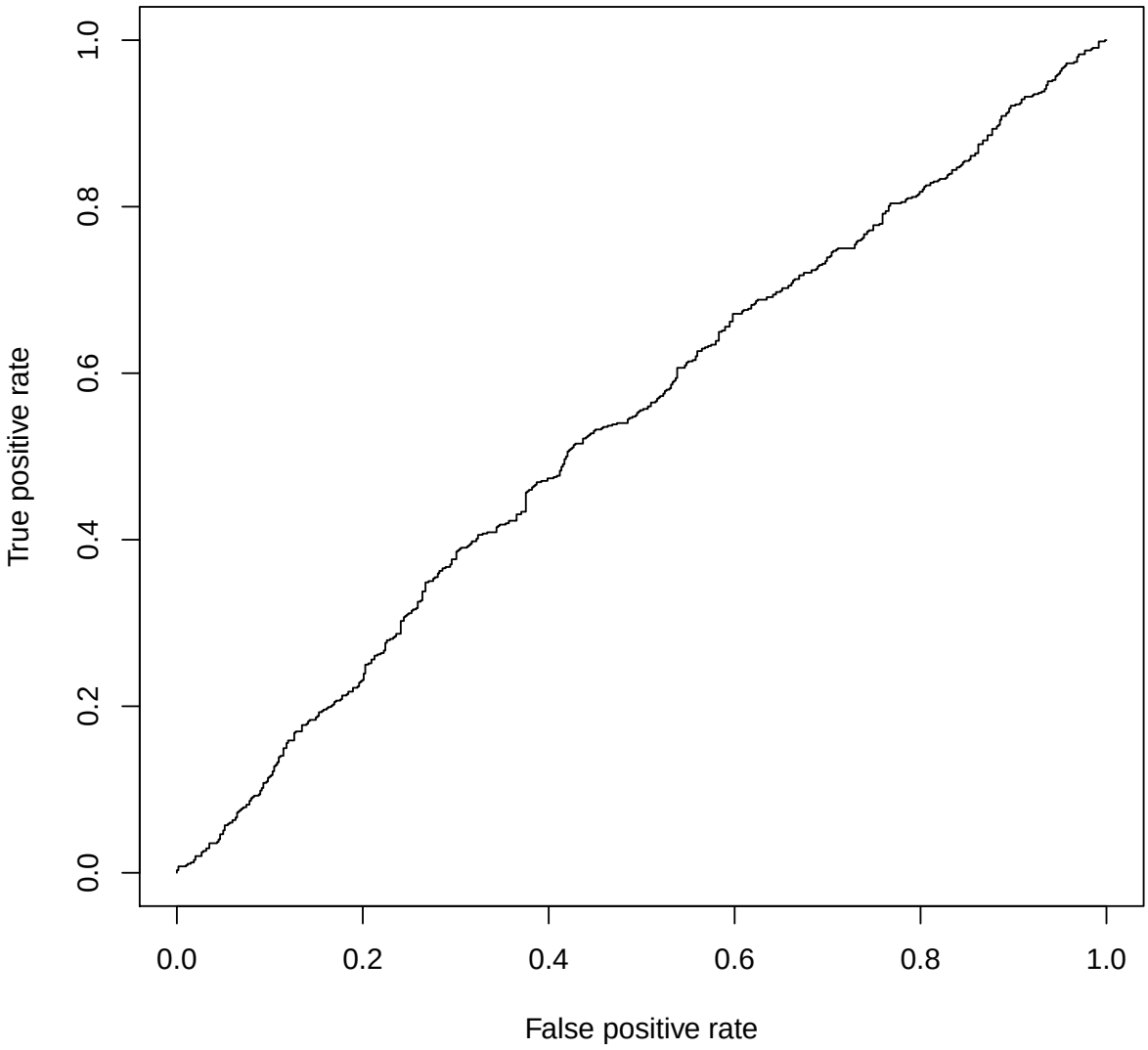



```
gl <- glm(Direction ~ Volume + Lag1 + Lag2,
    data = Smarket, family = binomial(link = "logit"))
summary(gl)

##
## Call:
## glm(formula = Direction ~ Volume + Lag1 + Lag2, family = binomial(link = "logit"
##     data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.452  -1.203   1.068   1.146   1.331
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
```

```
## (Intercept) -0.12058    0.24018  -0.502    0.616
## Volume       0.13184    0.15799   0.835    0.404
## Lag1        -0.07326    0.05017  -1.460    0.144
## Lag2        -0.04279    0.05006  -0.855    0.393
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.7  on 1246  degrees of freedom
## AIC: 1735.7
##
## Number of Fisher Scoring iterations: 3

roc <- ROC(predict(gl), Smarket$Direction)
plot(roc)
```



```r
AUC(predict(gl), Smarket$Direction)

## [1] 0.537096

gl <- glm(Direction ~ poly(Volume, Lag1,
    Lag2, degree = 2), data = Smarket, family = binomial(link = "logit"))
roc <- ROC(predict(gl), Smarket$Direction)
plot(roc)
```




```r
AUC(predict(gl), Smarket$Direction)

## [1] 0.5401183

library(e1071)
tn.glm <- tune(glm, Direction ~ poly(Volume,
```

```r
    Lag1, Lag2, degree = 2), data = Smarket,
    family = binomial(link = "logit"), predict.func = simple.predict.glm,
    tunecontrol = tune.control(cross = 100))
tn.mglm <- tune(glm, Direction ~ poly(Volume,
    Lag1, Lag2, degree = 2), data = Smarket,
    family = binomial(link = "logit"), predict.func = function(...) my.predict.glm(
        measure = "err"), tunecontrol = tune.control(cross = 100))
tn.glm

##
## Error estimation of 'glm' using 100-fold cross validation: 0.5097436

tn.mglm

##
## Error estimation of 'glm' using 100-fold cross validation: 0.5048077

tn.mglm$performances$dispersion

## [1] 0.163591

tn.qda <- tune(qda, Direction ~ Volume +
    Lag1 + Lag2, data = Smarket, predict.func = simple.predict.da,
    tunecontrol = tune.control(cross = 100))
tn.qda

##
## Error estimation of 'qda' using 100-fold cross validation: 0.5065385

Smarket.train <- subset(Smarket, Year <=
    2004)
Smarket.test <- subset(Smarket, Year > 2004)
qd <- qda(Direction ~ Lag1 + Lag2, data = Smarket.train)
cm.train <- table(actual = Smarket.train$Direction,
    predicted = predict(qd, Smarket.train)$class)
cm.test <- table(actual = Smarket.test$Direction,
    predicted = predict(qd, Smarket.test)$class)
chisq.test(cm.test)

##
##  Pearson's Chi-squared test with Yates' continuity
##  correction
##
## data:  cm.test
## X-squared = 5.6585, df = 1, p-value = 0.01737
```

5.4 banknote

```r
source("class.R")
banknote <- read.csv("banknote/data_banknote_authentication.txt",
    header = FALSE, comment.char = "#")
colnames(banknote) <- c("variance", "skewness",
    "curtosis", "entropy", "class")
banknote$class <- factor(banknote$class,
    labels = c("genuine", "forged"))
summary(banknote)

##     variance          skewness          curtosis
##  Min.   :-7.0421   Min.   :-13.773   Min.   :-5.2861
##  1st Qu.:-1.7730   1st Qu.: -1.708   1st Qu.:-1.5750
##  Median : 0.4962   Median :  2.320   Median : 0.6166
##  Mean   : 0.4337   Mean   :  1.922   Mean   : 1.3976
##  3rd Qu.: 2.8215   3rd Qu.:  6.815   3rd Qu.: 3.1793
##  Max.   : 6.8248   Max.   : 12.952   Max.   :17.9274
##     entropy            class
##  Min.   :-8.5482   genuine:762
##  1st Qu.:-2.4135   forged :610
##  Median :-0.5867
##  Mean   :-1.1917
##  3rd Qu.: 0.3948
##  Max.   : 2.4495

nb <- naiveBayes(class ~ ., data = banknote)
ld <- lda(class ~ ., data = banknote)
qd <- qda(class ~ ., data = banknote)
gl <- glm(class ~ ., data = banknote, family = binomial(link = "logit"))

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

table(predicted = predict(nb, banknote),
    actual = banknote$class)

##          actual
## predicted genuine forged
##   genuine     671    127
##   forged       91    483

tn <- tune(naiveBayes, class ~ ., data = banknote)
tn

##
## Error estimation of 'naiveBayes' using 10-fold cross validation: 0.1595843

tn$performances$dispersion

## [1] 0.03457557
```

```
tn.glm <- tune(glm, class ~ ., data = banknote,
    family = binomial(link = "logit"), predict.func = simple.predict.glm)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

tn.lda <- tune(lda, class ~ ., data = banknote,
    predict.func = simple.predict.da)
tn.qda <- tune(qda, class ~ ., data = banknote,
    predict.func = simple.predict.da)
tn.nb <- tune(naiveBayes, class ~ ., data = banknote)
tn.mglm <- tune(glm, class ~ ., data = banknote,
    family = binomial(link = "logit"), predict.func = function(...) my.predict.glm(
        measure = "err"))

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

tn.mlda <- tune(my.lda, class ~ ., data = banknote,
    predict.func = function(...) my.predict.da(...,
        measure = "err"))
tn.mqda <- tune(my.qda, class ~ ., data = banknote,
    predict.func = function(...) my.predict.da(...,
        measure = "err"), tunecontrol = tune.control(cross = 100))
tn.lda

##
## Error estimation of 'lda' using 10-fold cross validation: 0.02331535

tn.qda
```

```
##
## Error estimation of 'qda' using 10-fold cross validation: 0.01457738
```

```
tn.glm
```

```
##
## Error estimation of 'glm' using 10-fold cross validation: 0.01239289
```

```
tn.nb
```

```
##
## Error estimation of 'naiveBayes' using 10-fold cross validation: 0.1625304
```

```
tn.mlda
```

```
##
## Error estimation of 'my.lda' using 10-fold cross validation: 0.008753835
```

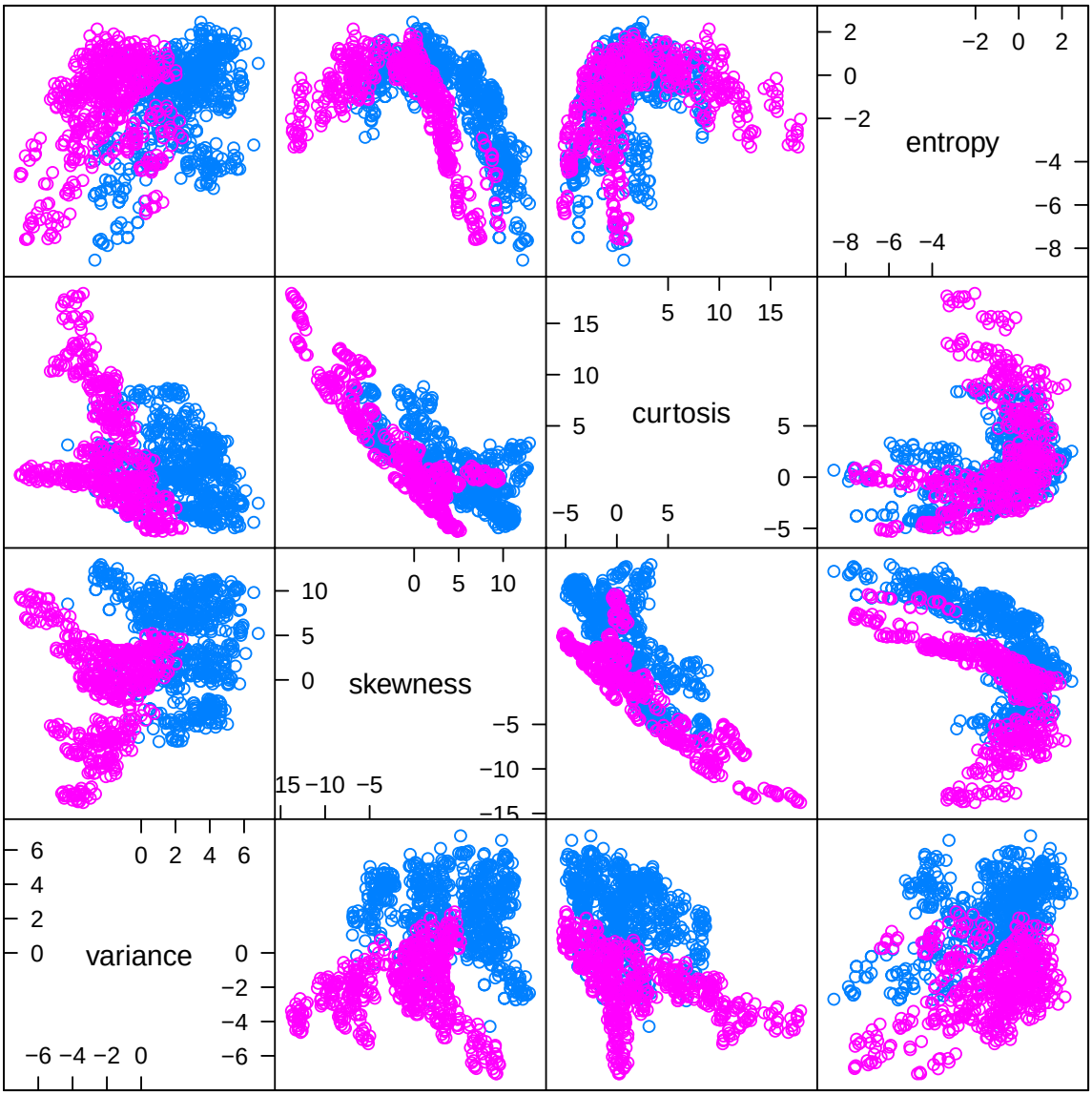
```
tn.mqda
```

```
##
## Error estimation of 'my.qda' using 100-fold cross validation: 0.0007692308
```

```
summary(tn.mglm$best.model)
```

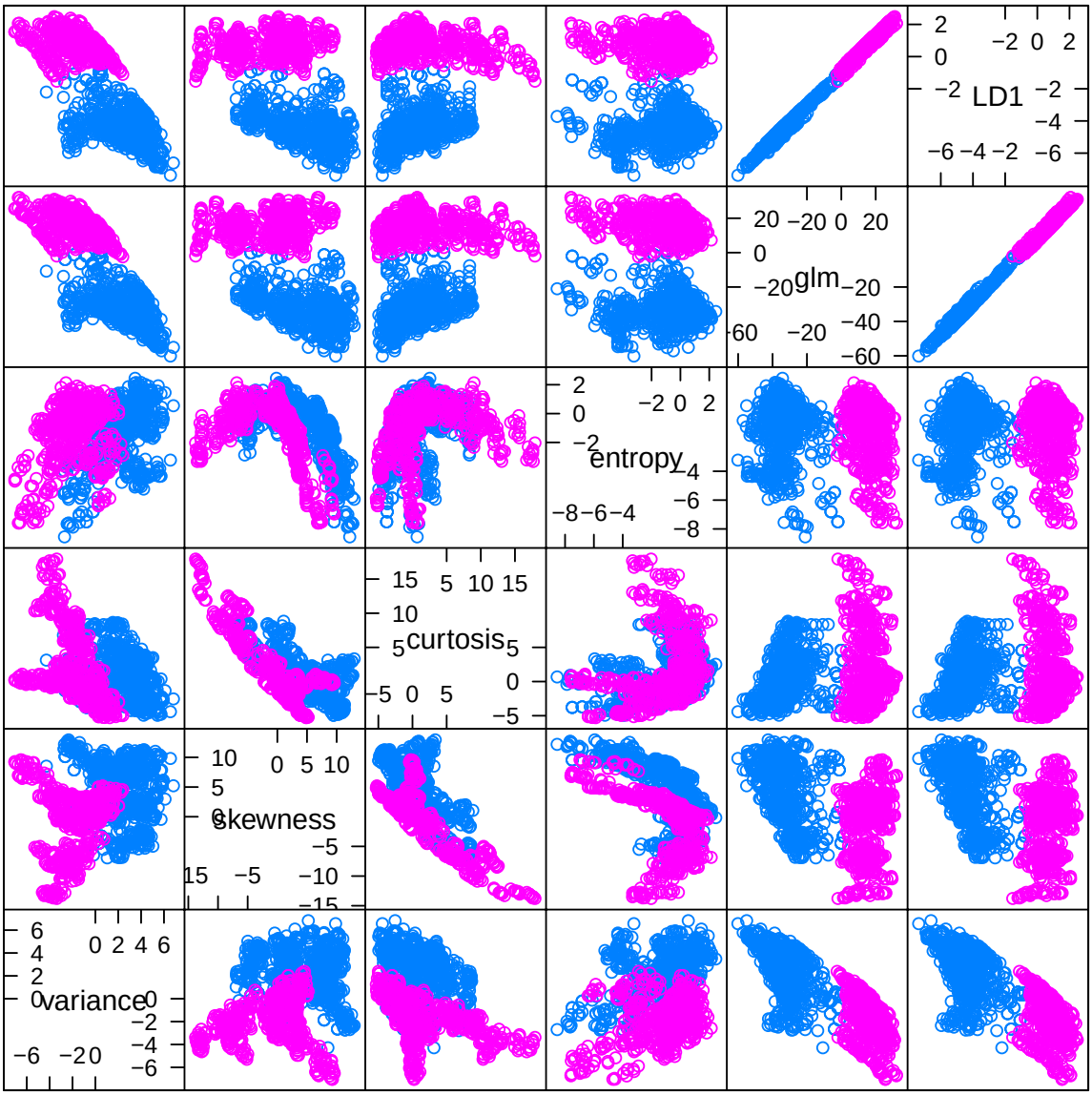
```
##
## Call:
## best.tune(method = glm, train.x = class ~ ., data = banknote,
##     predict.func = function(...) my.predict.glm(..., measure = "err"),
##     family = binomial(link = "logit"))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.70001   0.00000   0.00000   0.00029   2.24614
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   7.3218     1.5589   4.697 2.64e-06 ***
## variance     -7.8593     1.7383  -4.521 6.15e-06 ***
## skewness     -4.1910     0.9041  -4.635 3.56e-06 ***
## curtosis     -5.2874     1.1612  -4.553 5.28e-06 ***
## entropy      -0.6053     0.3307  -1.830   0.0672 .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1885.122  on 1371  degrees of freedom
## Residual deviance:   49.891  on 1367  degrees of freedom
## AIC: 59.891
##
## Number of Fisher Scoring iterations: 12
```

```r
summary(tn.mqda$best.model)

##         Length Class      Mode
## prior    2     -none-     numeric
## counts   2     -none-     numeric
## means    8     -none-     numeric
## scaling 32     -none-     numeric
## ldet     2     -none-     numeric
## lev      2     -none-     character
## N        1     -none-     numeric
## call     6     -none-     call
## terms    3     terms      call
## xlevels  0     -none-     list
## data     5     data.frame list

splom(subset(banknote, select = -class),
    groups = banknote$class)
```



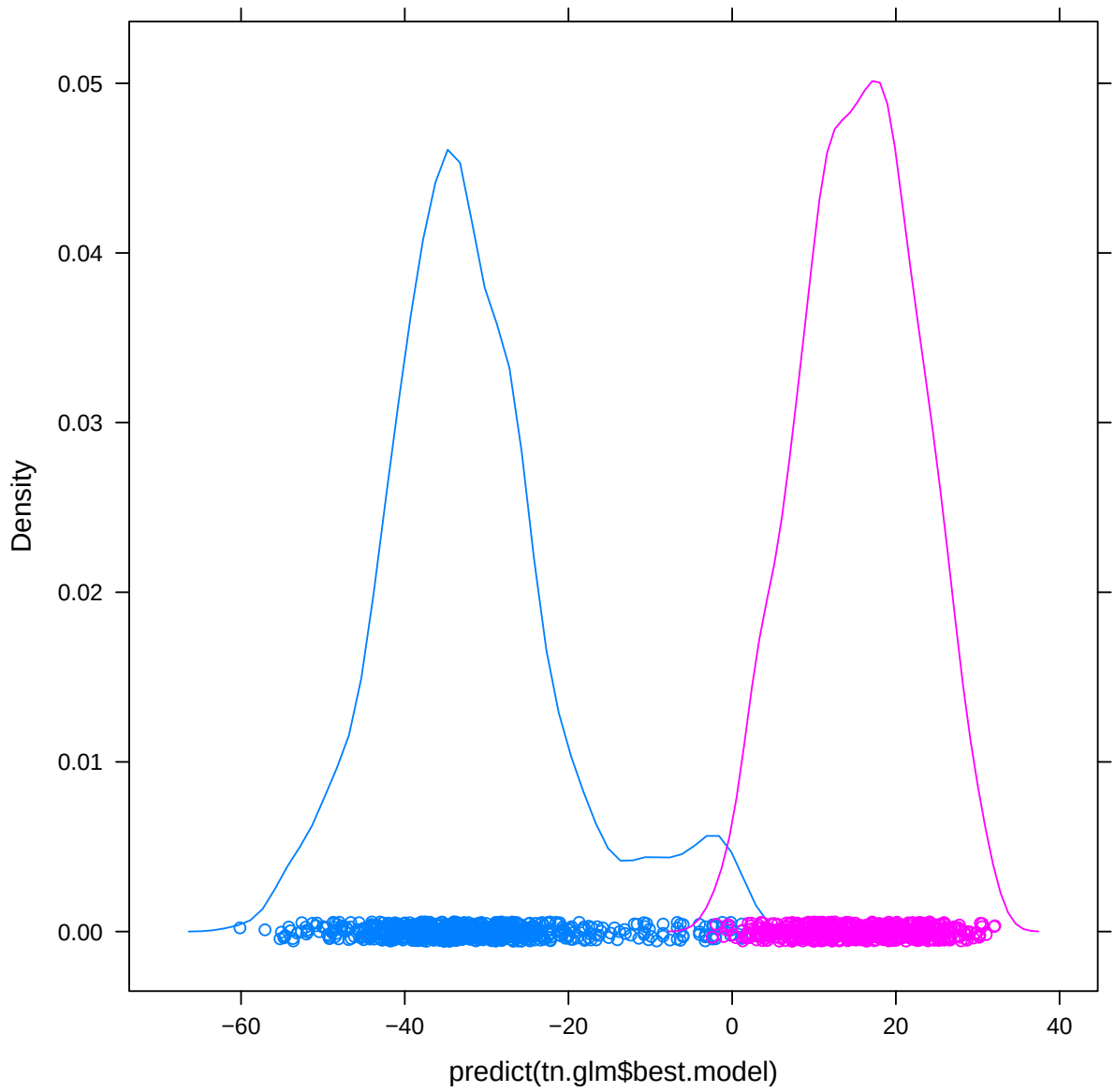
Scatter Plot Matrix

```
splom(cbind(subset(banknote, select = -class),
    glm = predict(tn.mglm$best.model), lda = as.matrix(subset(banknote,
        select = -class)) %*% tn.mlda$best.model$scaling),
    groups = banknote$class)
```

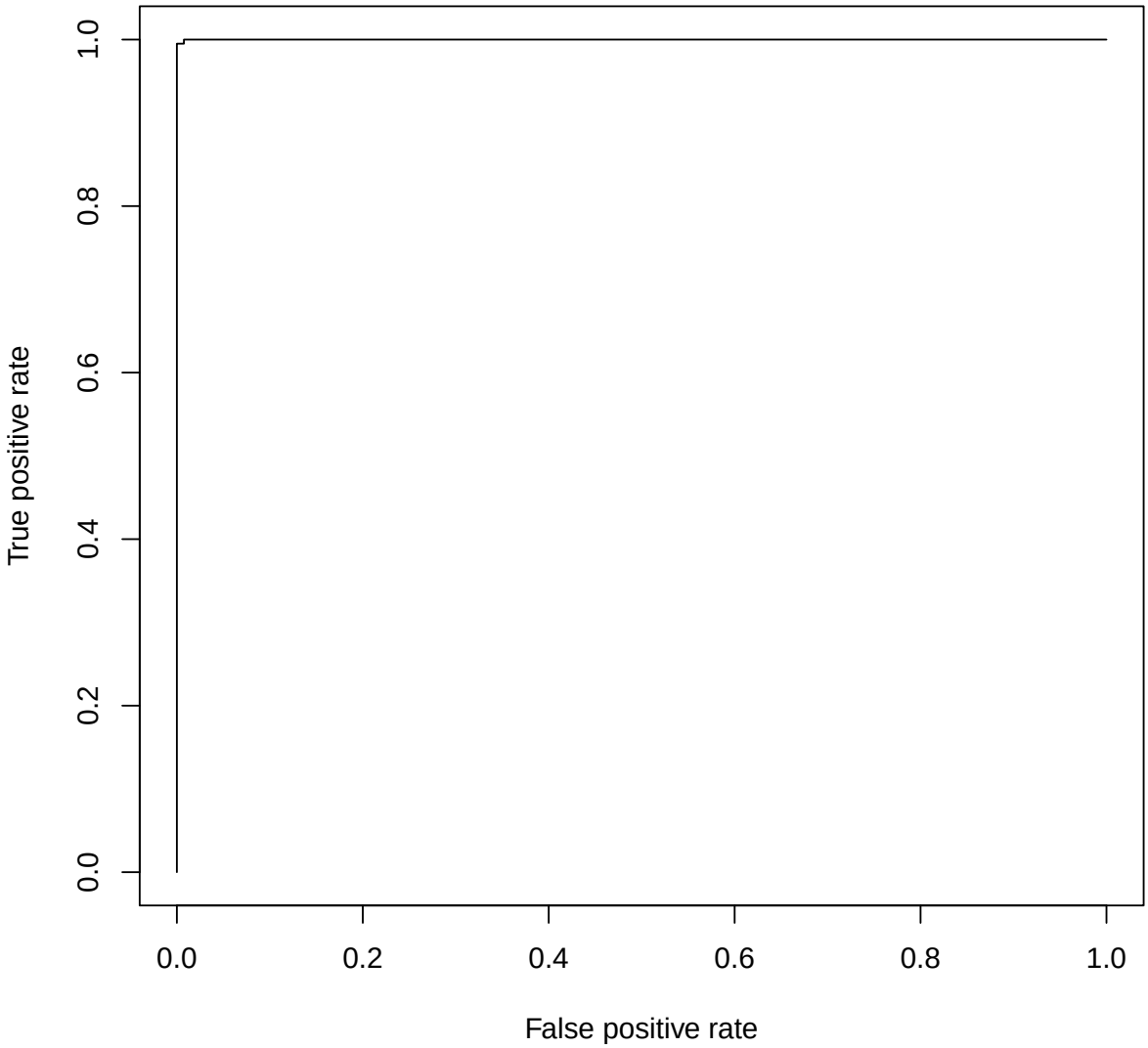


Scatter Plot Matrix

```
densityplot(~predict(tn.glm$best.model),
    groups = banknote$class)
```




```
train <- sample(nrow(banknote), size = 0.66 *
    nrow(banknote))
gl <- glm(class ~ ., data = banknote, subset = train,
    family = binomial)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

roc <- ROC(predict(gl, banknote[-train, ],
    type = "response"), as.numeric(banknote[-train,
    ]$class))
plot(roc)
```



6 Материалы с занятия 17 октября

6.1 Bootstrap and CV

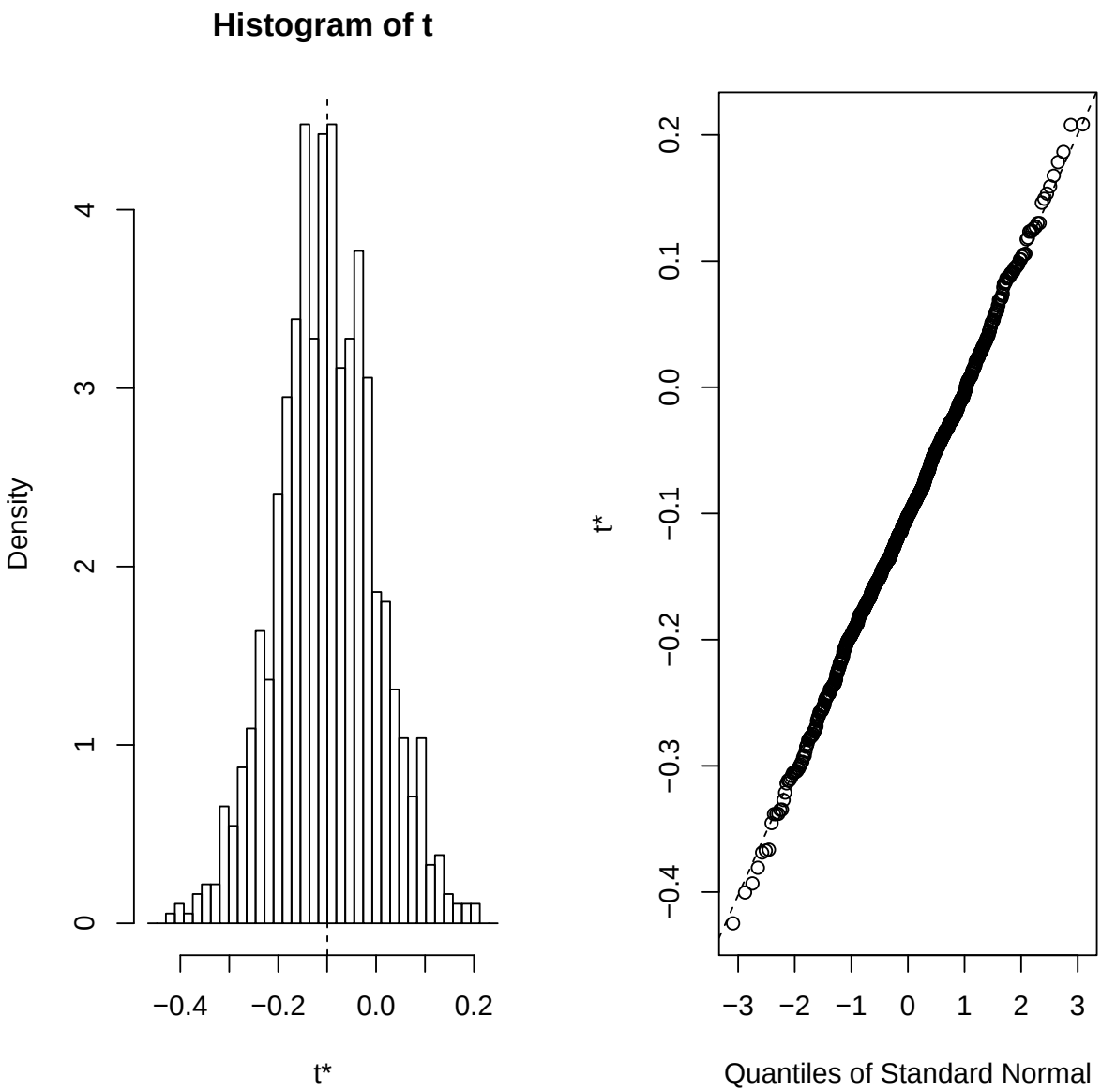
```
library(MASS)
library(lattice)
library(latticeExtra)
library(e1071)
library(boot)

##
## Attaching package: 'boot'
##
```

```
## The following object is masked from 'package:lattice':
##
##    melanoma

library(mvtnorm)
source("class.R")
N <- 100
x <- rnorm(N)
b.np <- boot(x, function(data, subset) mean(data[subset]),
    R = 999)
boot.ci(b.np)

## Warning in boot.ci(b.np): bootstrap variances needed for studentized intervals
```

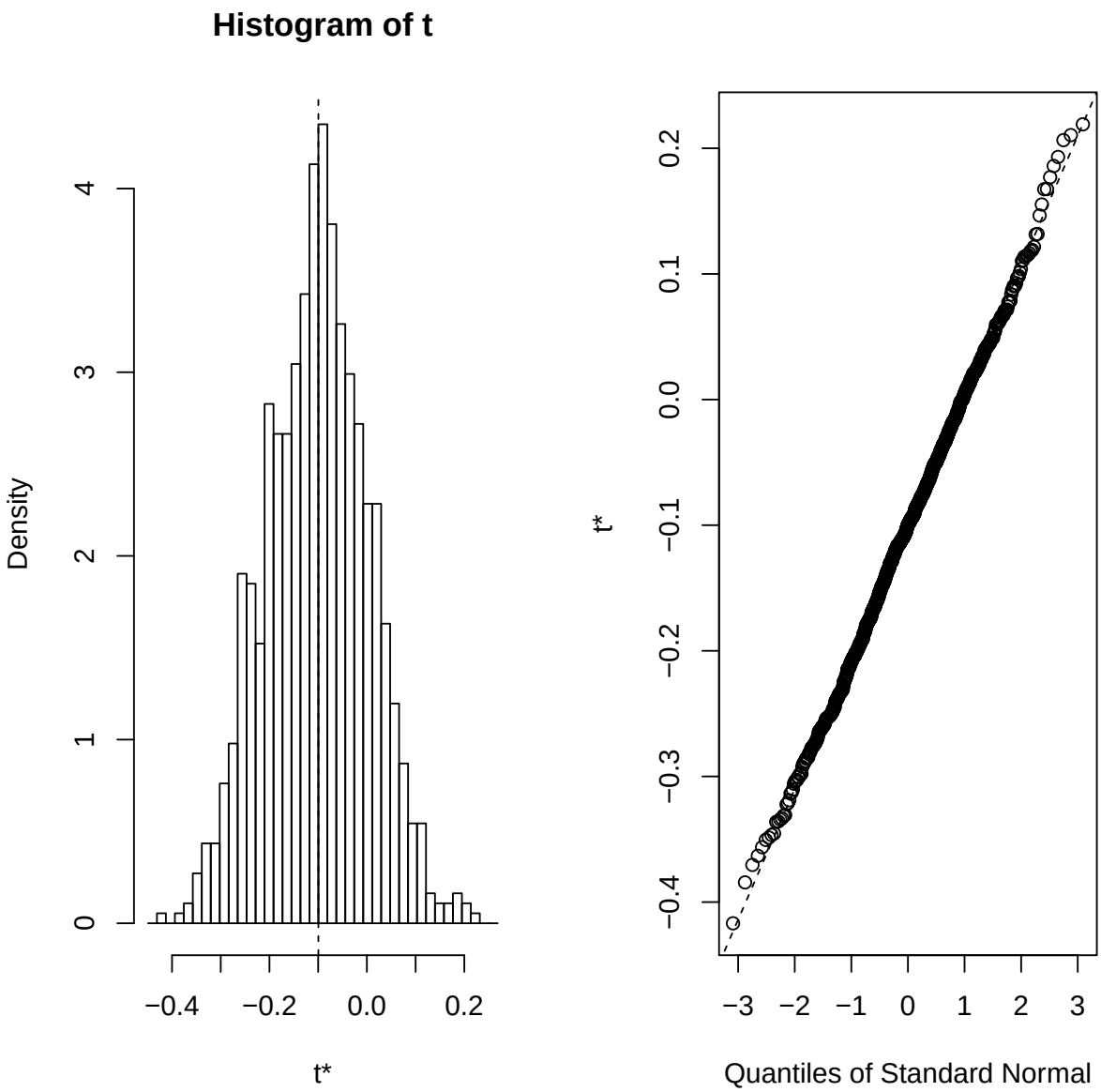
```
boot.ci(b.np)$normal

## Warning in boot.ci(b.np): bootstrap variances needed for studentized intervals
```

```
t.test(x)
```

```r
b.p <- boot(x, mean, R = 999, sim = "parametric",
    ran.gen = function(data, pars) rnorm(length(data),
        mean = pars$mean, sd = pars$sd),
    mle = list(mean = mean(x), sd = sd(x)))
boot.ci(b.p, type = "norm")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.p, type = "norm")
##
## Intervals :
## Level      Normal
## 95%   (-0.3009,  0.1067 )
## Calculations and Intervals on Original Scale

boot.ci(b.p, type = "perc")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.p, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   (-0.3028,  0.0983 )
## Calculations and Intervals on Original Scale

plot(b.np)
```

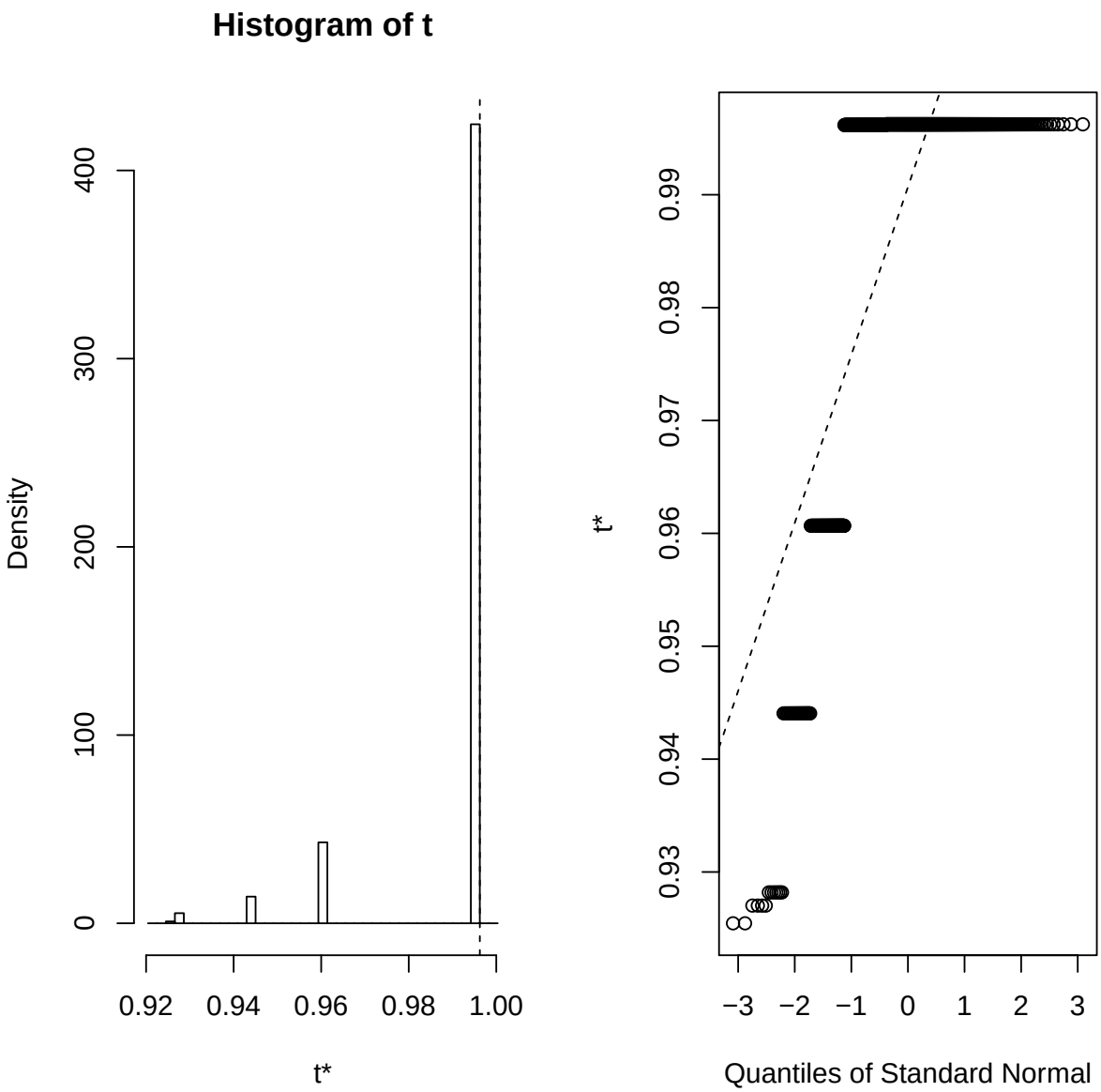

Histogram of t
Density
0
1
2
3
4
−0.4
−0.2
0.0
0.2
t*
t*
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
−3
−2
−1
0
1
2
3
Quantiles of Standard Normal

```
plot(b.p)
```



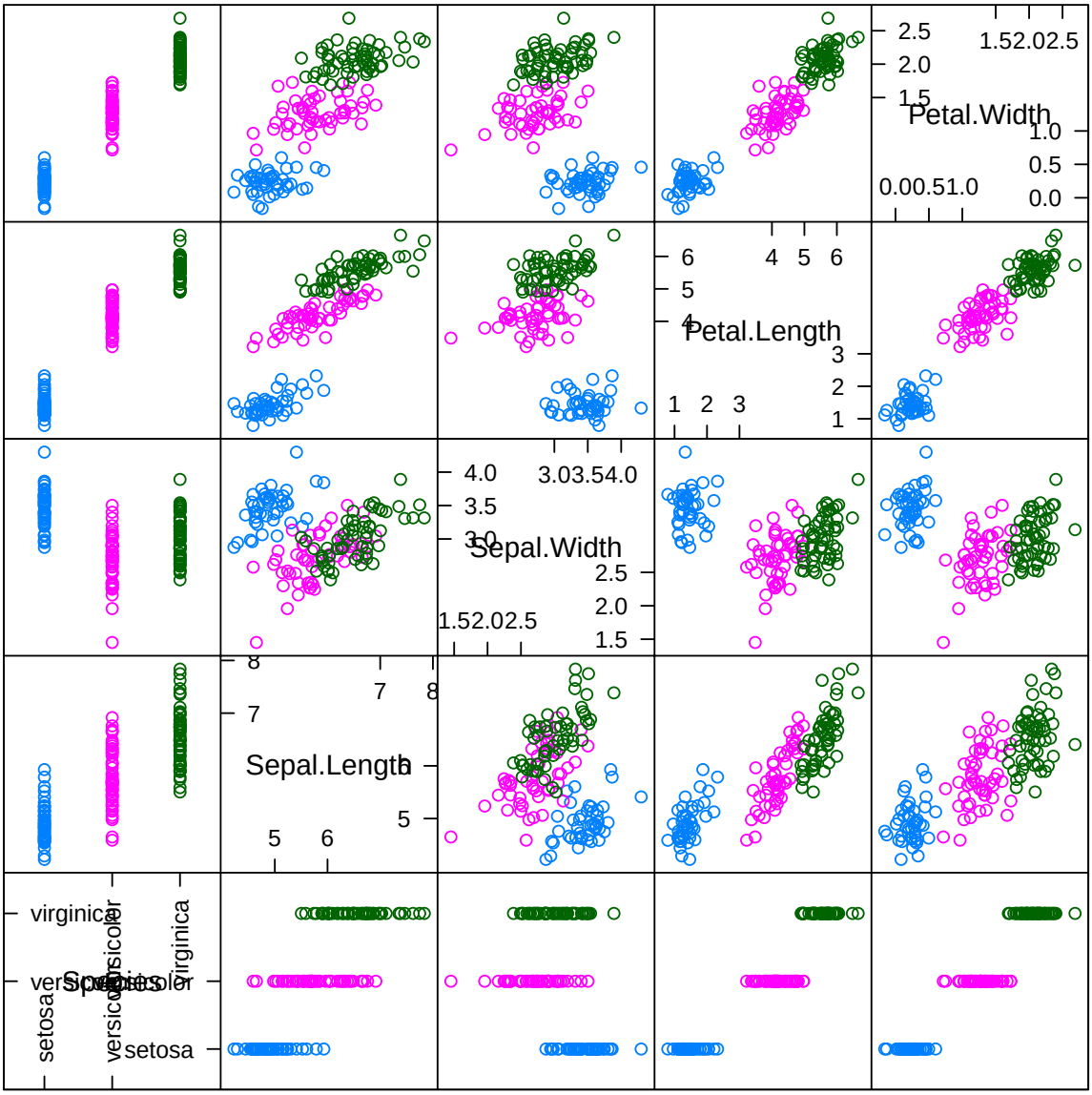

```
x <- runif(N)
b.np <- boot(x, function(x, subset) max(x[subset]),
    R = 999)
b.p <- boot(x, max, R = 999, sim = "parametric",
    ran.gen = function(data, pars) runif(length(data),
        min = pars$min, max = pars$max),
    mle = list(min = min(x), max = max(x)))
boot.ci(b.np)

## Warning in boot.ci(b.np): bootstrap variances needed for studentized intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
```

```
## boot.ci(boot.out = b.np)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.9726,  1.0310 )   ( 0.9962,  1.0484 )
##
## Level     Percentile            BCa
## 95%   ( 0.9441,  0.9962 )   ( 0.9282,  0.9962 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable

boot.ci(b.p, type = "norm")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.p, type = "norm")
##
## Intervals :
## Level      Normal
## 95%   ( 0.9865,  1.0263 )
## Calculations and Intervals on Original Scale

boot.ci(b.p, type = "perc")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.p, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   ( 0.9585,  0.9960 )
## Calculations and Intervals on Original Scale

plot(b.np)
```



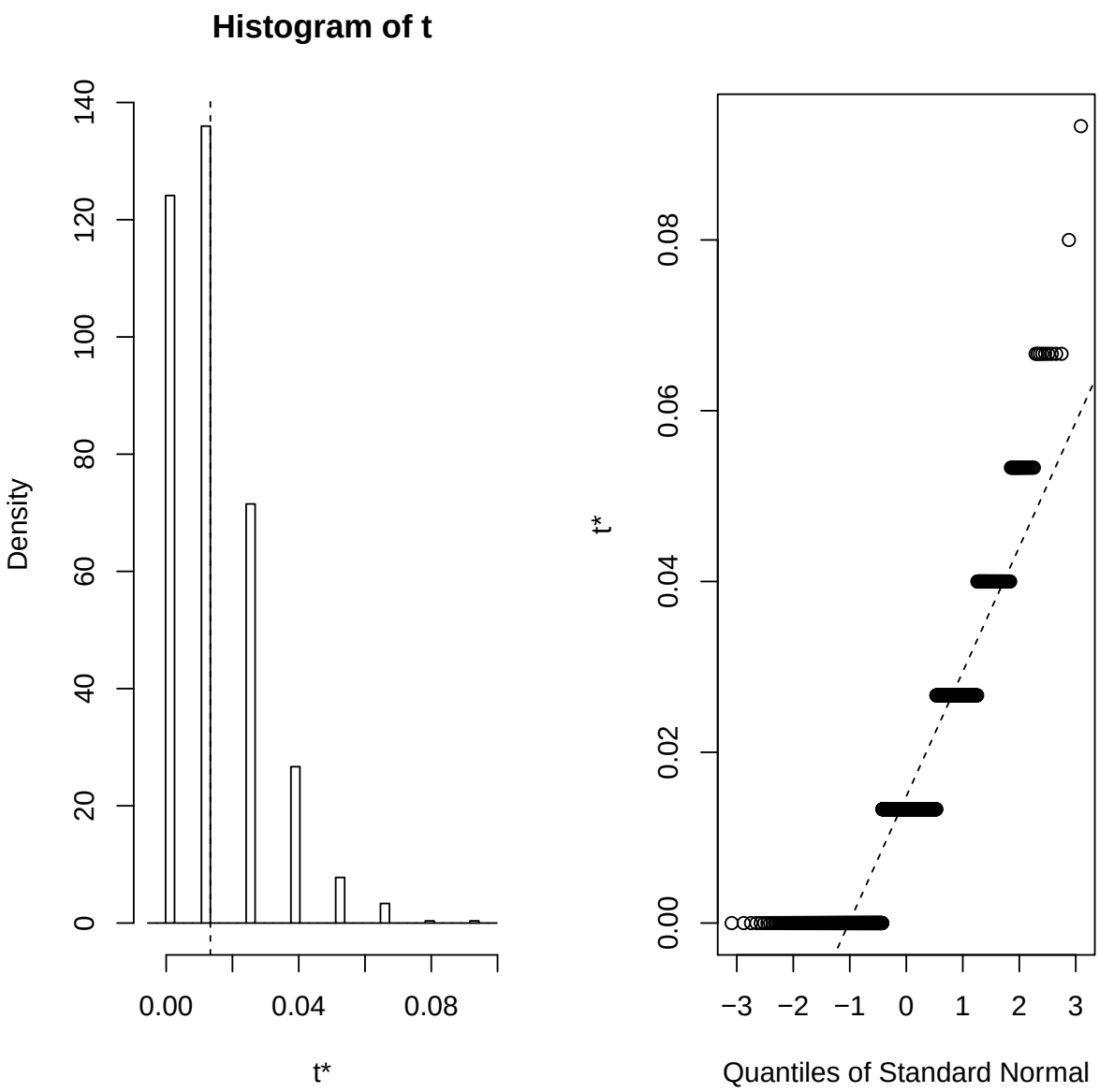

```
lda.model <- function(data, groups) {
    data <- as.matrix(data)
    means <- aggregate(data, list(groups = groups),
        mean)
    data <- data - as.matrix(means[match(groups,
        means$groups), -1])
    list(cov = cov(data), means = means[,
        -1, drop = FALSE])
}
model <- lda.model(subset(iris, select = -Species),
    iris$Species)
make.data <- function(data, groups, lda.model,
    size = nrow(data), groups.name = "Species") {
    ind <- sample(seq_along(levels(groups)),
        size = size, replace = TRUE)
```

```
    res <- data.frame(name = factor(levels(groups)[ind],
        levels = levels(groups)))
    names(res) <- groups.name
    mx <- lda.model$means[ind, ]
    mx <- mx + rmvnorm(nrow(mx), sigma = lda.model$cov)
    colnames(mx) <- colnames(lda.model$cov)
    res <- cbind(res, as.data.frame(mx))
    rownames(res) <- NULL
    res
}
res <- make.data(iris, iris$Species, model)
splom(res, groups = res$Species)
```

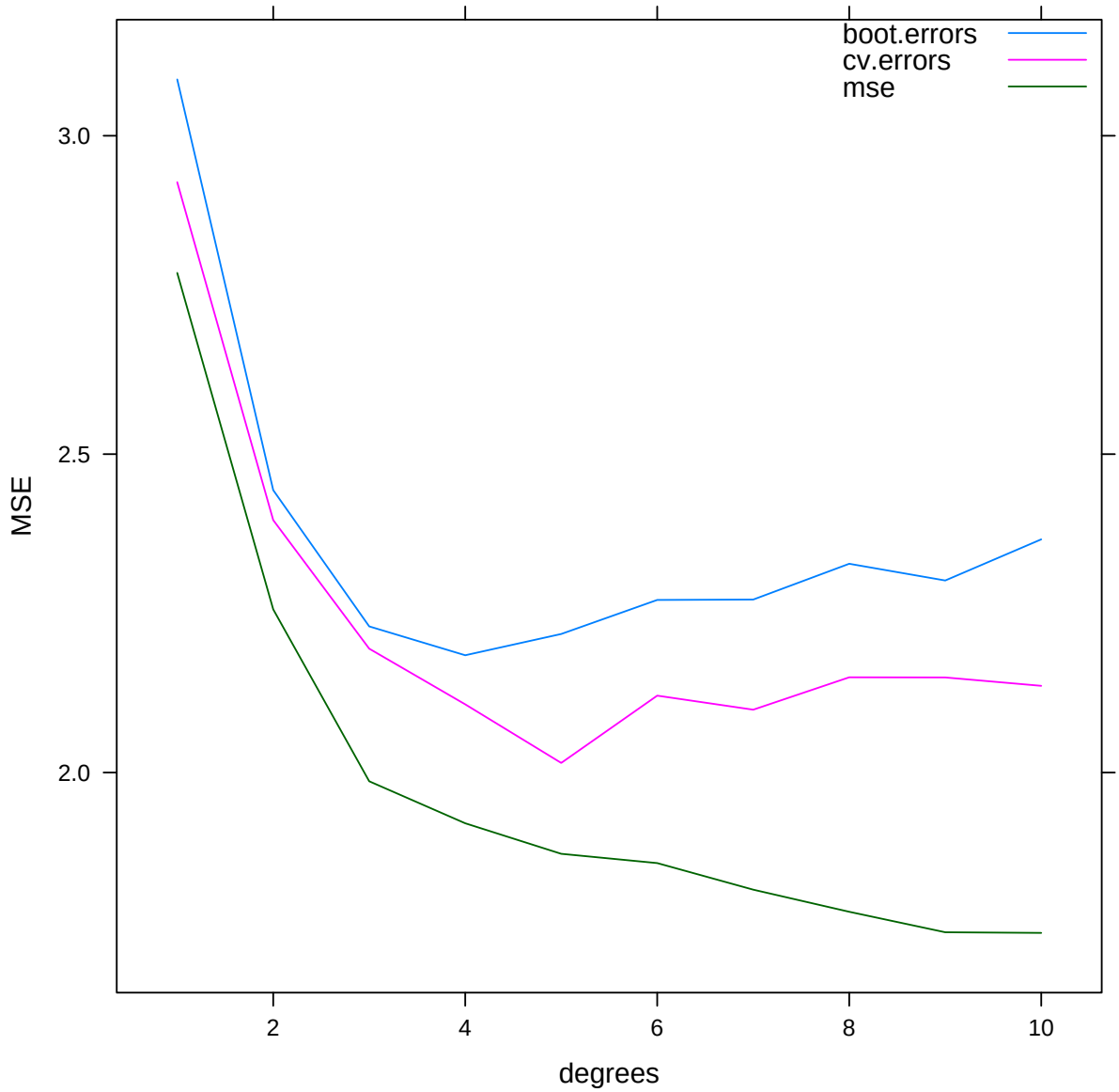


Scatter Plot Matrix

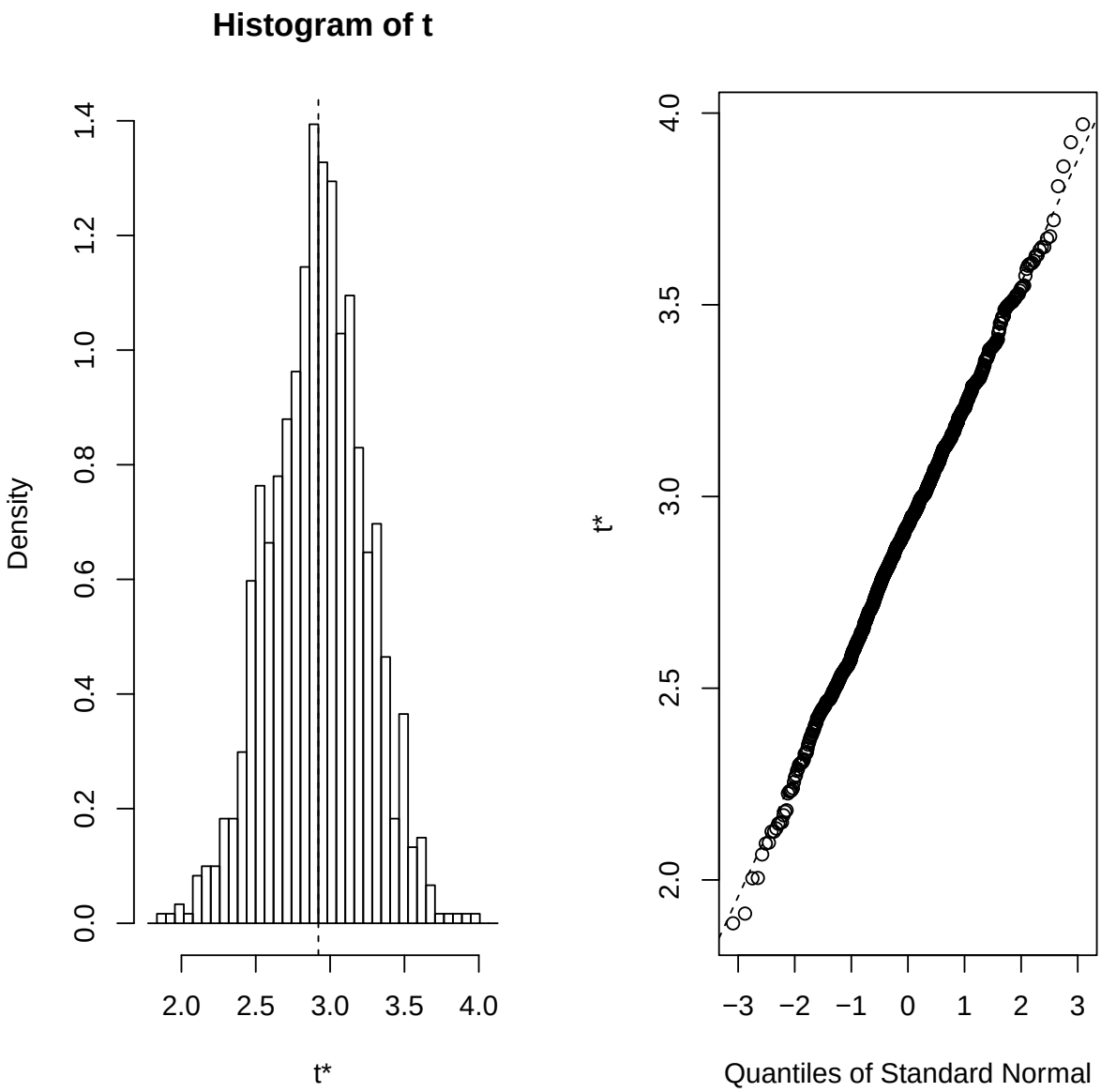
```r
b <- boot(iris, function(data) {
    tn <- tune(my.lda, Species ~ ., data = data,
        predict.func = simple.predict.da,
        tunecontrol = tune.control(sampling = "fix",
            fix = 1/2))
    tn$best.performance
}, R = 999, sim = "parametric", ran.gen = function(data,
    mle, ..., size = 300) make.data(data,
    mle$groups, mle$lda.model, ...), mle = list(groups = iris$Species,
    lda.model = lda.model(subset(iris, select = -Species),
        groups = iris$Species)))
plot(b)
```



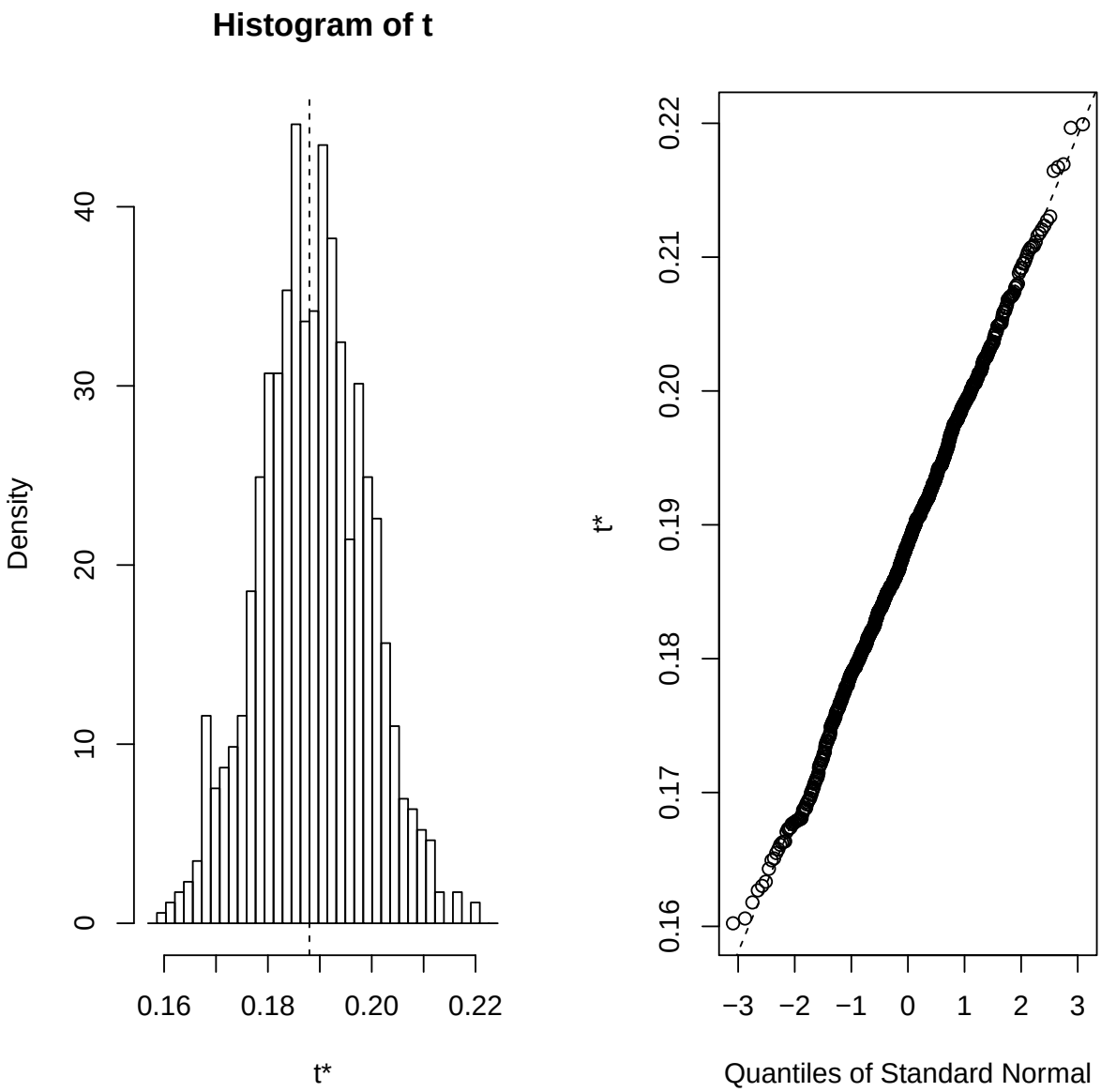
```r
boot.ci(b, type = "perc")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, type = "perc")
##
## Intervals :
## Level     Percentile
## 95%   ( 0.0000,  0.0533 )
## Calculations and Intervals on Original Scale

Advertising <- read.csv("Advertising.csv")
Advertising$X <- NULL
degrees <- 1:10
tns.cv <- list()
tns.boot <- list()
mse <- rep(NA_real_, length(degrees))
for (i in seq_along(degrees)) {
    model <- Sales ~ Radio + poly(TV, degree = degrees[i])
    tns.cv[[i]] <- tune(lm, model, data = Advertising,
        tunecontrol = tune.control(sampling = "cross",
            cross = 15))
    tns.boot[[i]] <- tune(lm, model, data = Advertising,
        tunecontrol = tune.control(sampling = "boot",
            nboot = 100))
    mse[i] <- mean(residuals(lm(model, data = Advertising))^2)
}
cv.errors <- sapply(tns.cv, function(el) el$performance$error)
boot.errors <- sapply(tns.boot, function(el) el$performance$error)
degrees[which.min(cv.errors)]

## [1] 5

degrees[which.min(boot.errors)]

## [1] 4

xyplot(boot.errors + cv.errors + mse ~ degrees,
    type = "l", auto.key = list(corner = c(1,
        1), lines = TRUE, points = FALSE),
    ylab = "MSE")
```



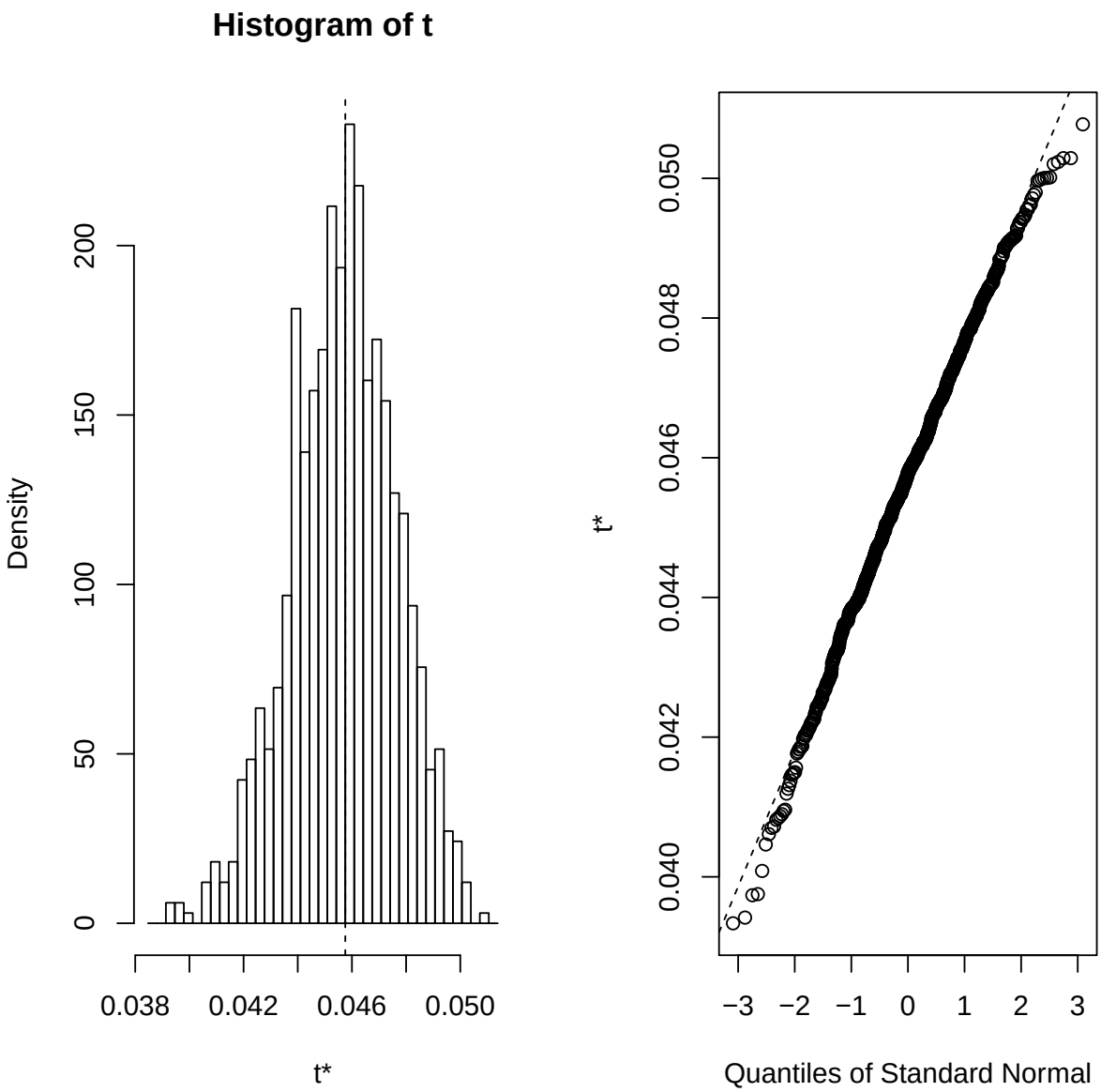

```
b <- boot(Advertising, function(...) coef(lm(Sales ~
    Radio + TV, ...)), R = 999)
plot(b, index = 1)
```




```
plot(b, index = 2)
```




```
plot(b, index = 3)
```




```
boot.ci(b, index = 1)

## Warning in boot.ci(b, index = 1): bootstrap variances needed for studentized
intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, index = 1)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 2.297,  3.553 )   ( 2.314,  3.558 )
##
```

```
## Level     Percentile            BCa
## 95%   ( 2.285,  3.529 )   ( 2.274,  3.528 )
## Calculations and Intervals on Original Scale

boot.ci(b, index = 1)$percent

## Warning in boot.ci(b, index = 1): bootstrap variances needed for studentized
intervals

##      conf
## [1,] 0.95 25 975 2.284511 3.528545

boot.ci(b, index = 2)$percent

## Warning in boot.ci(b, index = 2): bootstrap variances needed for studentized
intervals

##      conf
## [1,] 0.95 25 975 0.1679166 0.208791

boot.ci(b, index = 3)$percent

## Warning in boot.ci(b, index = 3): bootstrap variances needed for studentized
intervals

##      conf
## [1,] 0.95 25 975 0.04176539 0.04934465

l <- lm(Sales ~ Radio + TV, data = Advertising)
confint(l)

##                  2.5 %     97.5 %
## (Intercept) 2.34034299 3.50185683
## Radio       0.17213877 0.20384969
## TV          0.04301292 0.04849671
```

```
library(MASS)
library(e1071)
source("class.R")
library(nnet)
fertility <- read.csv("fertility/fertility_Diagnosis.txt",
    comment.char = "#", header = FALSE)
names(fertility) <- c("Season", "Age", "ChildishDesiases",
    "Trauma", "Surgical", "Fevers", "Alcohol",
    "Smoking", "Sitting", "Diagnosis")
fertility$Season <- factor(fertility$Season,
    labels = c("winter", "spring", "summer",
        "fall"))
contrasts(fertility$Season) <- contr.sum
```

```r
fertility$Age <- 18 + 18 * fertility$Age
fertility$ChildishDesiases <- factor(fertility$ChildishDesiases,
    labels = c("yes", "no"))
contrasts(fertility$ChildishDesiases) <- c(1,
    0)
fertility$Trauma <- factor(fertility$Trauma,
    labels = c("yes", "no"))
contrasts(fertility$Trauma) <- c(1, 0)
fertility$Surgical <- factor(fertility$Surgical,
    labels = c("yes", "no"))
contrasts(fertility$Surgical) <- c(1, 0)
fertility$Fevers <- factor(fertility$Fevers,
    labels = c("less3", "more3", "no"), ordered = TRUE)
fertility$Fevers <- factor(fertility$Fevers,
    levels = rev(levels(fertility$Fevers)),
    ordered = TRUE)
contrasts(fertility$Fevers) <- contr.helmert
fertility$Alcohol <- factor(fertility$Alcohol,
    labels = c("sevday", "evday", "sevweek",
        "evweek", "never"), ordered = TRUE)
fertility$Alcohol <- factor(fertility$Alcohol,
    levels = rev(levels(fertility$Alcohol)),
    ordered = TRUE)
contrasts(fertility$Alcohol) <- contr.helmert
fertility$Smoking <- factor(fertility$Smoking,
    labels = c("never", "occasional", "daily"),
    ordered = TRUE)
contrasts(fertility$Smoking) <- contr.helmert
fertility$Sitting <- fertility$Sitting *
    16
fertility$Diagnosis <- factor(fertility$Diagnosis,
    labels = c("Normal", "Altered"))
gl <- glm(Diagnosis ~ ., data = fertility,
    family = binomial)
summary(gl)

##
## Call:
## glm(formula = Diagnosis ~ ., family = binomial, data = fertility)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1621  -0.4872  -0.2585  -0.1271   2.8269
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -20.9269   678.7346  -0.031   0.9754
## Season1             -1.4493     1.0583  -1.370   0.1708
```

```
## Season2            -0.3051     0.6931  -0.440   0.6598
## Season3             0.9589     1.2124   0.791   0.4290
## Age                 0.3677     0.2162   1.701   0.0890 .
## ChildishDesiases1  -0.5696     1.0441  -0.546   0.5854
## Trauma1             1.6410     0.8572   1.914   0.0556 .
## Surgical1          -0.3057     0.7979  -0.383   0.7017
## Fevers1             0.2473     0.5347   0.462   0.6438
## Fevers2             0.5369     0.3723   1.442   0.1493
## Alcohol1            0.9215     0.5298   1.740   0.0819 .
## Alcohol2            0.4349     0.3150   1.381   0.1674
## Alcohol3           -3.3031   599.8863  -0.006   0.9956
## Alcohol4           -2.4536   494.6789  -0.005   0.9960
## Smoking1           -0.1733     0.5031  -0.345   0.7304
## Smoking2            0.1632     0.3036   0.537   0.5910
## Sitting             0.2050     0.1511   1.357   0.1747
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 73.385  on 99  degrees of freedom
## Residual deviance: 55.134  on 83  degrees of freedom
## AIC: 89.134
##
## Number of Fisher Scoring iterations: 15
```

```r
gl <- glm(Diagnosis ~ Trauma + Age + Alcohol,
    data = fertility, family = binomial)
summary(gl)
```

```
##
## Call:
## glm(formula = Diagnosis ~ Trauma + Age + Alcohol, family = binomial,
##     data = fertility)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2511  -0.5020  -0.3513  -0.2706   2.4948
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -14.7395   678.7121  -0.022   0.9827
## Trauma1       1.3935     0.7439   1.873   0.0610 .
## Age           0.2066     0.1590   1.300   0.1937
## Alcohol1      0.8010     0.4381   1.828   0.0675 .
## Alcohol2      0.2621     0.2646   0.991   0.3218
## Alcohol3     -3.2959   599.8862  -0.005   0.9956
```

```
## Alcohol4     -2.2602   494.6788  -0.005   0.9964
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 73.385  on 99  degrees of freedom
## Residual deviance: 63.914  on 93  degrees of freedom
## AIC: 77.914
##
## Number of Fisher Scoring iterations: 15
```

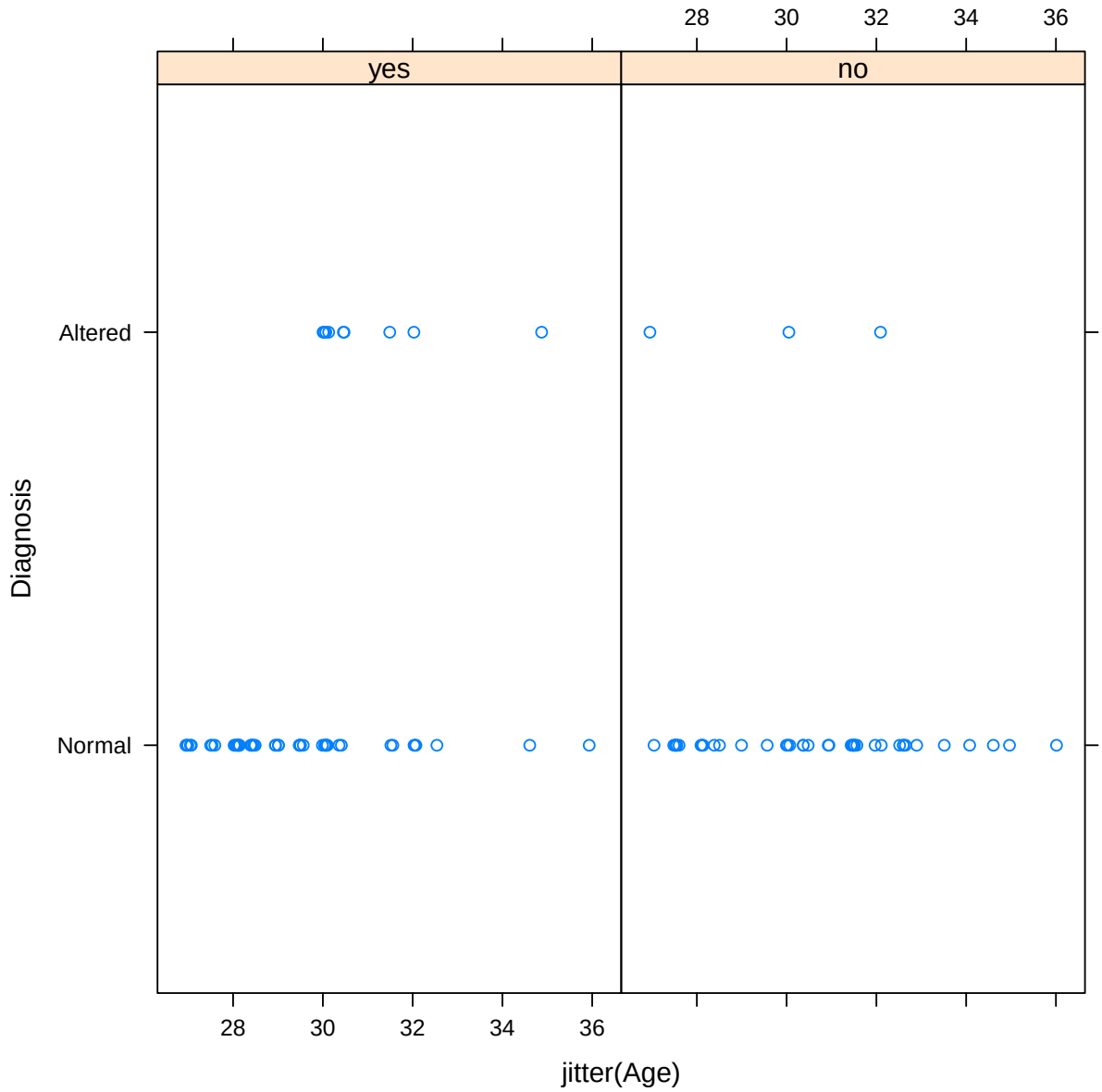
```
gl.aic <- stepAIC(gl)
```

```
## Start:  AIC=77.91
## Diagnosis ~ Trauma + Age + Alcohol
##
##           Df Deviance    AIC
## - Alcohol  4   69.022 75.022
## - Age      1   65.616 77.616
## <none>         63.914 77.914
## - Trauma   1   67.938 79.938
##
## Step:  AIC=75.02
## Diagnosis ~ Trauma + Age
##
##          Df Deviance    AIC
## <none>        69.022 75.022
## - Age     1   71.280 75.280
## - Trauma  1   72.118 76.118
```

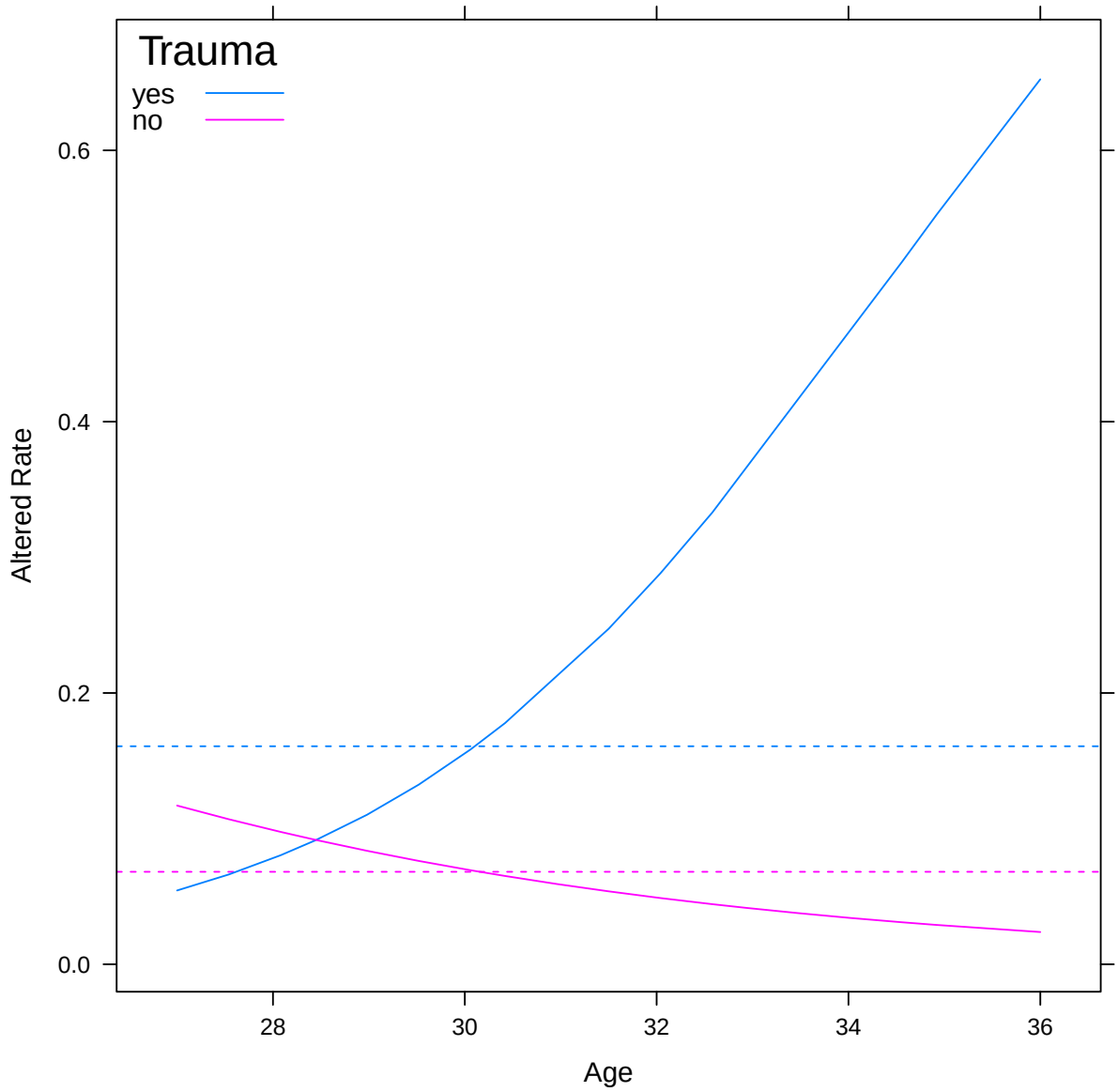
```
summary(gl.aic)
```

```
##
## Call:
## glm(formula = Diagnosis ~ Trauma + Age, family = binomial, data = fertility)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0360  -0.5396  -0.4415  -0.3155   2.6531
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -9.3009     4.5276  -2.054   0.0399 *
## Trauma1       1.2102     0.7336   1.650   0.0990 .
## Age           0.2152     0.1422   1.513   0.1302
## ---
## Signif. codes:
```

```
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 73.385  on 99  degrees of freedom
## Residual deviance: 69.022  on 97  degrees of freedom
## AIC: 75.022
##
## Number of Fisher Scoring iterations: 5

gl <- glm(Diagnosis ~ Age * Trauma, data = fertility,
    family = binomial)
xyplot(Diagnosis ~ jitter(Age) | Trauma,
    data = fertility, auto.key = list(corner = c(0,
        1)))
```



```r
gl <- glm(Diagnosis ~ Age * Trauma, data = fertility,
    family = binomial(link = "logit"))
fertility.sorted <- fertility[order(fertility$Age),
    ]
xyplot(predict(gl, fertility.sorted, type = "response") ~
    Age, groups = Trauma, data = fertility.sorted,
    type = "l", auto.key = list(corner = c(0,
        1), title = "Trauma", lines = TRUE,
        points = FALSE), xlab = "Age", ylab = "Altered Rate") +
    layer_(panel.superpose(x = fertility$Diagnosis ==
        "Altered", panel.groups = function(x,
        y, ...) panel.abline(h = mean(x),
        ...), lty = "dashed", groups = fertility$Trauma,
        ...))
```



```r
gl <- glm(Diagnosis ~ Sitting, data = fertility,
    family = binomial)
summary(gl)
```

```
##
## Call:
## glm(formula = Diagnosis ~ Sitting, family = binomial, data = fertility)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.5602  -0.5135  -0.4967  -0.4902   2.1151
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.14628    0.74487  -2.881  0.00396 **
## Sitting      0.02335    0.10174   0.230  0.81844
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 73.385  on 99  degrees of freedom
## Residual deviance: 73.333  on 98  degrees of freedom
## AIC: 77.333
##
## Number of Fisher Scoring iterations: 4
```

```r
ld <- lda(Diagnosis ~ Age * Trauma, data = fertility)
table(predicted = predict(ld, fertility)$class,
    actual = fertility$Diagnosis)
```

```
## Warning in predict.lda(ld, fertility): variable names in 'newdata' do not match
those in 'object'
```

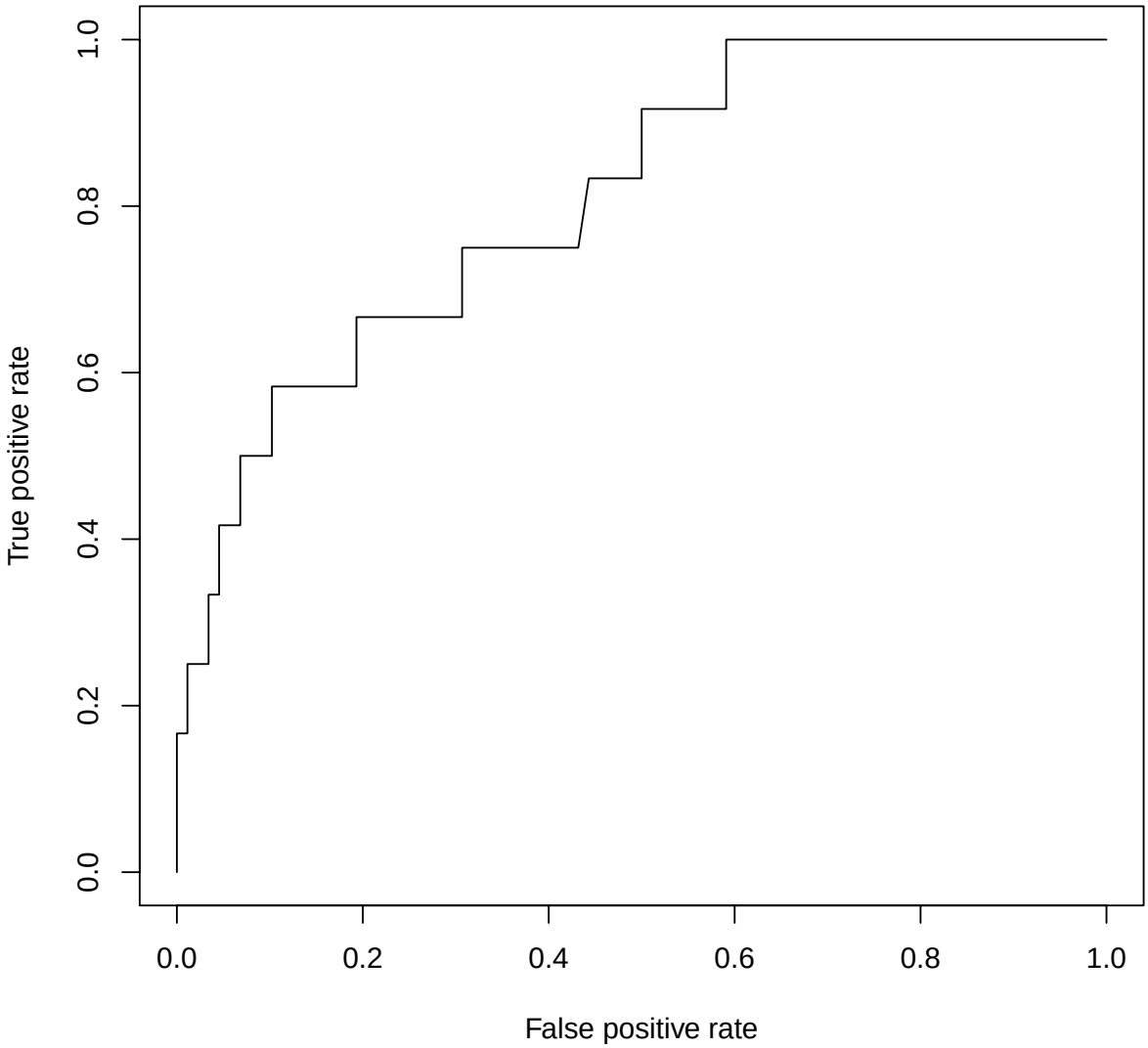
```
##          actual
## predicted Normal Altered
##   Normal      82      12
##   Altered      6       0
```

```r
qd <- qda(Diagnosis ~ Age * Trauma, data = fertility)
table(predicted = predict(qd, fertility)$class,
    actual = fertility$Diagnosis)
```

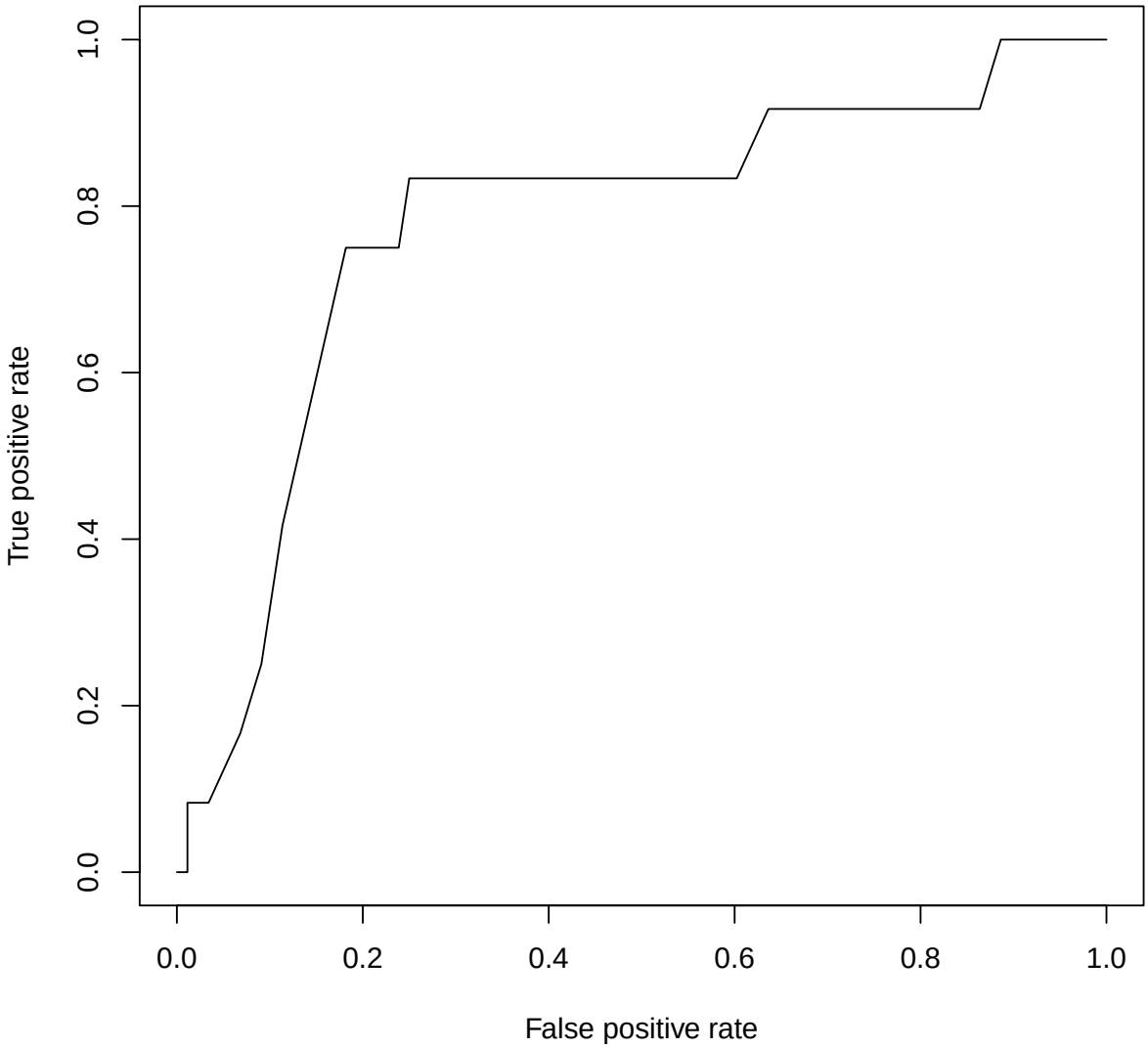
```
## Warning in predict.qda(qd, fertility): variable names in 'newdata' do not match
those in 'object'
```

```
##          actual
## predicted Normal Altered
##   Normal      76      11
##   Altered     12       1
```

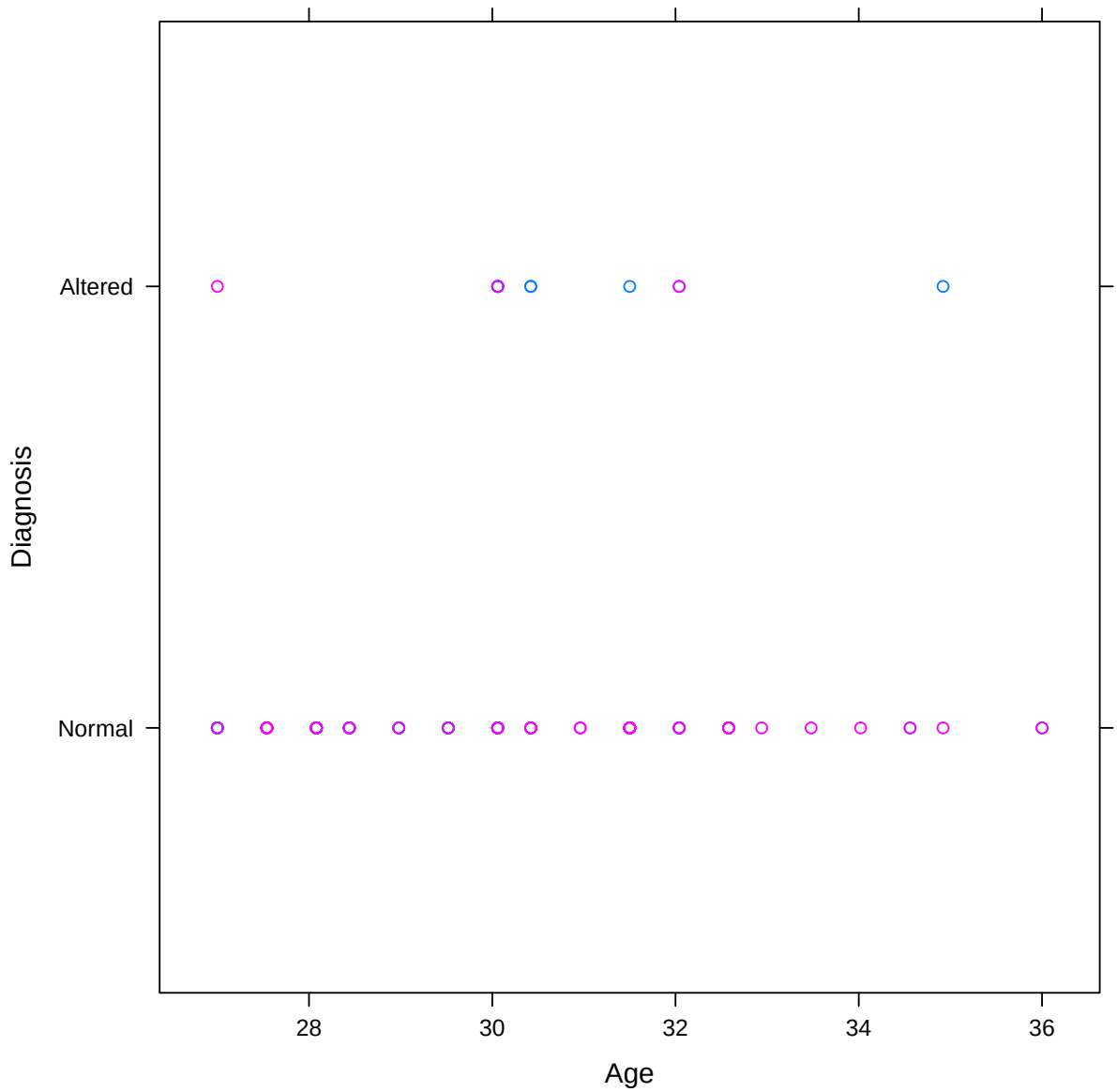
```r
ld <- lda(Diagnosis ~ Age + Trauma, data = fertility)
table(predicted = predict(ld, fertility)$class,
    actual = fertility$Diagnosis)

## Warning in predict.lda(ld, fertility): variable names in 'newdata' do not match
those in 'object'

##          actual
## predicted Normal Altered
##   Normal      88      12
##   Altered      0       0

tune(glm, data = fertility, Diagnosis ~ Age *
    Trauma, family = binomial, tunecontrol = tune.control(sampling = "boot"),
    predict.func = my.predict.glm)

##
## Error estimation of 'glm' using bootstrapping: 0.2421375

nb <- naiveBayes(Diagnosis ~ ., data = fertility)
table(predicted = predict(nb, fertility),
    actual = fertility$Diagnosis)

##          actual
## predicted Normal Altered
##   Normal      88      11
##   Altered      0       1

pred.nb <- predict(nb, fertility, type = "raw")[,
    2]
pred.glm <- predict(glm(Diagnosis ~ Age *
    Trauma, data = fertility, family = binomial))
plot(ROC(pred.nb, fertility$Diagnosis))
```



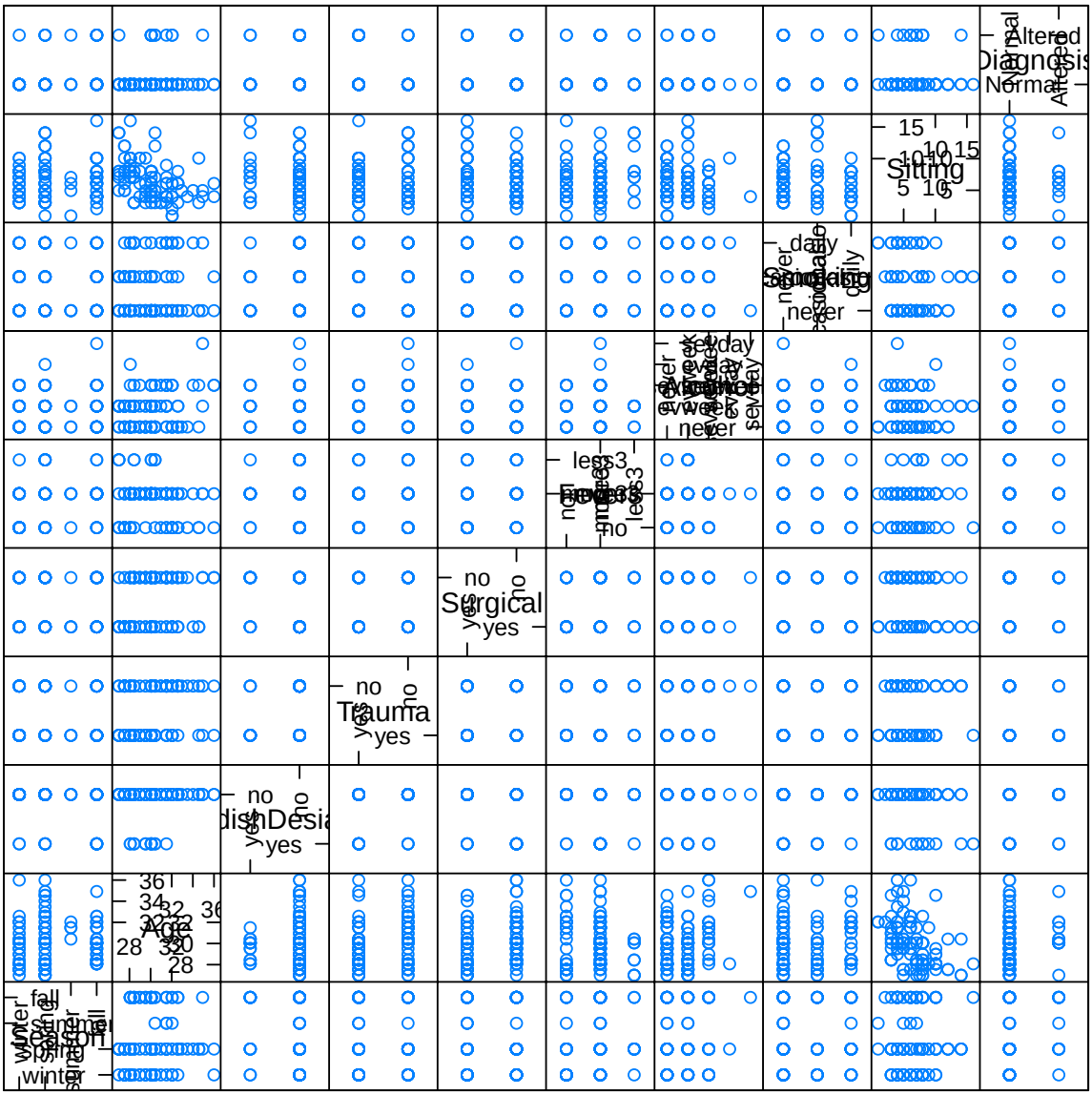

```
plot(ROC(pred.glm, fertility$Diagnosis))
```




```
xyplot(Diagnosis ~ Age, groups = Trauma,
    data = fertility)
```




```
gl <- glm(Diagnosis ~ Age + Trauma + Alcohol +
    Sitting + Season, data = fertility, family = binomial)
library(lattice)
splom(fertility)
```



Scatter Plot Matrix

7 Материалы с занятия 24 октября

```
read_chunk("PCA/PCAfncs.R")
read_chunk("PCA/PCA.R")
read_chunk("PCA/PCAmnist.R")
```

7.1 My PCA functions('PCAfncs.R')

```
library(svd)
logweightsplot <- function(pc, ncomp = 50,
    ...) {
```

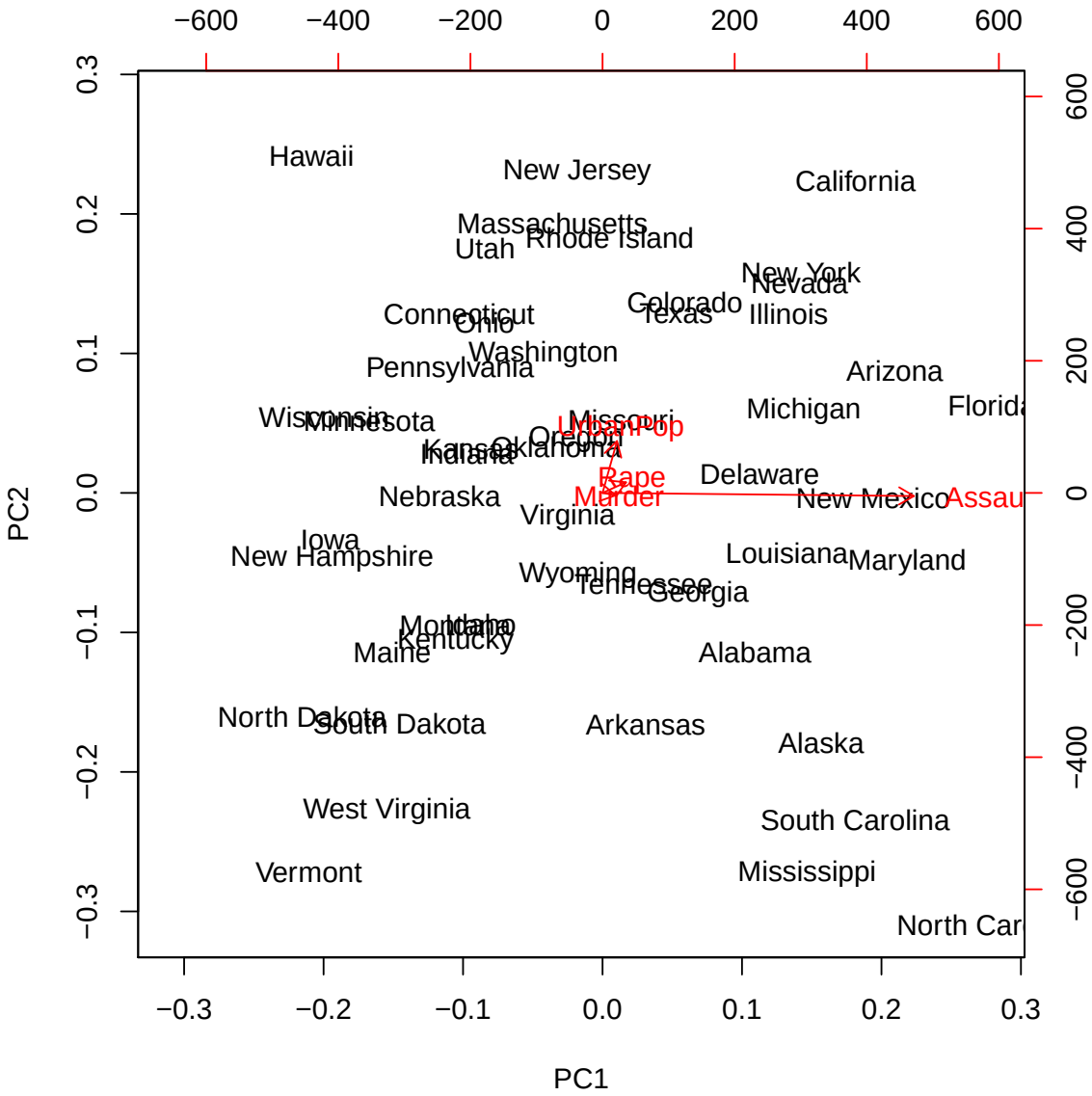
```
    w <- colSums(pc$x^2)
    w <- w[seq_len(min(ncomp, length(w)))]
    xyplot(log(w) ~ seq_along(w), ...)
}
cumweightsplot <- function(pc, ncomp = 50,
    ...) {
    w <- colSums(pc$x^2)
    w <- w/sum(w)
    w <- w[seq_len(min(ncomp, length(w)))]
    xyplot(cumsum(w) ~ seq_along(w), ...)
}
pcawrap <- function(learner, x, data = NULL,
    subset = NULL, ..., ncomp, center = TRUE,
    scale = TRUE) {
    if (is.null(data))
        data <- parent.frame()
    mf <- model.frame(x, data = data)
    if (!is.null(subset))
        mf <- mf[subset, ]
    response <- mf[, 1]
    predictors <- mf[, -1, drop = FALSE]
    pca <- prcomp(predictors, scale = scale,
        center = center, ncomp = ncomp)
    pca.data <- as.data.frame(predict(pca)[,
        seq_len(ncomp), drop = FALSE], predictors)
    pca.data$response <- response
    model <- learner(response ~ ., data = pca.data,
        ...)
    res <- list(pca = pca, model = model,
        formula = x, data = data, terms = attr(mf,
            "terms"), ncomp = ncomp)
    class(res) <- "pcawrap"
    res
}
predict.pcawrap <- function(object, newdata = object$data,
    ...) {
    mf <- model.frame(delete.response(object$terms),
        data = newdata)
    predictors <- as.data.frame(predict(object$pca,
        mf)[, seq_len(object$ncomp), drop = FALSE])
    predict(object$model, newdata = predictors,
        ...)
}
prcomp.default <- function(x, retx = TRUE,
    center = TRUE, scale. = FALSE, tol = NULL,
    ncomp = 50, use.robust.scaling = FALSE,
    use.robust.cov = FALSE, ...) {
    scale.mm <- function(x, center = TRUE,
```

```r
        scale = TRUE) {
        if (isTRUE(center))
            center <- apply(x, 2, median)
        if (isTRUE(scale))
            scale <- apply(x, 2, mad)
        base::scale(x, center = center, scale = scale)
    }
    x <- as.matrix(x)
    if (use.robust.scaling) {
        x <- scale.mm(x, center = center,
            scale = scale.)
    } else {
        x <- scale(x, center = center, scale = scale.)
    }
    cen <- attr(x, "scaled:center")
    sc <- attr(x, "scaled:scale")
    if (any(sc == 0))
        stop("cannot rescale a constant/zero column to unit variance")
    if (!use.robust.cov) {
        if (min(dim(x)) < 50) {
            s <- svd(x, nu = 0, nv = ncomp)
        } else {
            require("svd")
            s <- propack.svd(x, neig = ncomp)
        }
    } else {
        cov <- MASS::cov.rob(x)$cov
        if (ncol(cov) > 50) {
            v <- trlan.eigen(cov, neig = ncomp)$u
        } else {
            v <- eigen(cov, symmetric = TRUE)$vectors
        }
        d <- sqrt(colSums((x %*% v)^2))
        s <- list(d = d, v = v)
    }
    s$d <- s$d/sqrt(max(1, nrow(x) - 1))
    if (!is.null(tol)) {
        rank <- sum(s$d > (s$d[1L] * tol))
        if (rank < ncol(x)) {
            s$v <- s$v[, 1L:rank, drop = FALSE]
            s$d <- s$d[1L:rank]
        }
    }
    dimnames(s$v) <- list(colnames(x), paste0("PC",
        seq_len(ncol(s$v))))
    r <- list(sdev = s$d, rotation = s$v,
        center = if (is.null(cen)) FALSE else cen,
        scale = if (is.null(sc)) FALSE else sc)
```

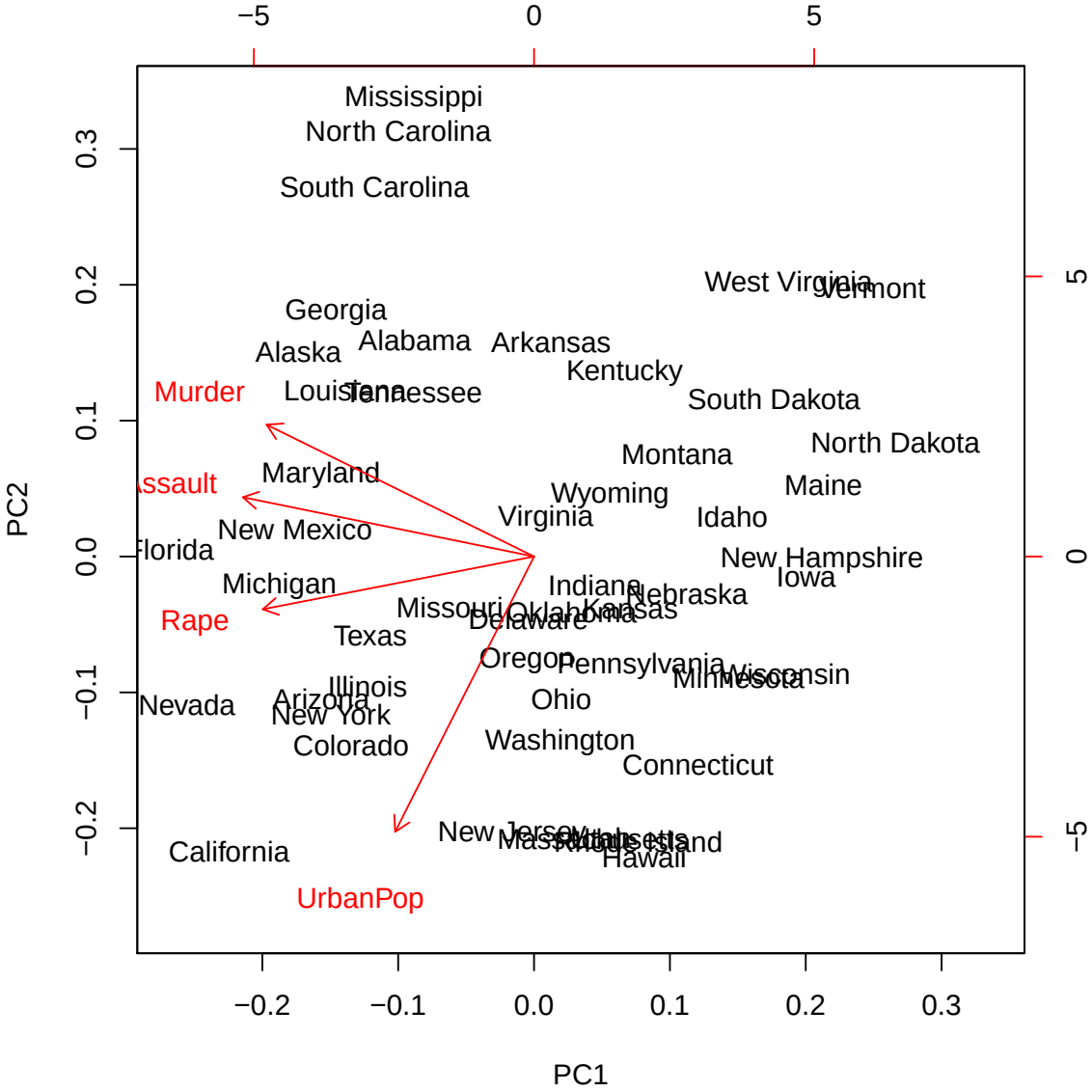
```r
    if (retx)
        r$x <- x %*% s$v
    class(r) <- "prcomp"
    r
}
unlockBinding("prcomp.default", env = loadNamespace("stats"))
assign("prcomp.default", prcomp.default,
    envir = loadNamespace("stats"))
lockBinding("prcomp.default", env = loadNamespace("stats"))
```

7.2 Basic PCA

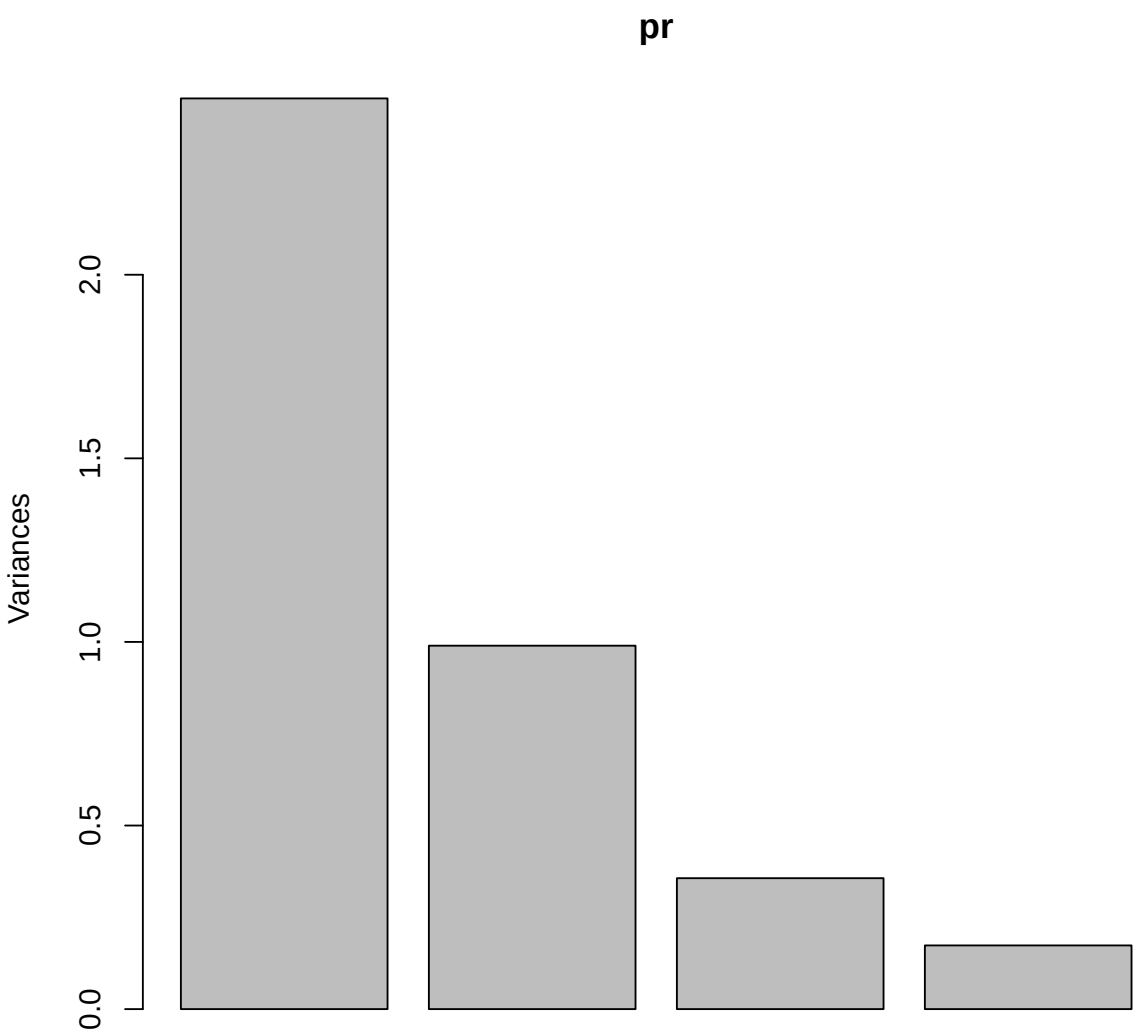
```r
library(MASS)
library(lattice)
library(latticeExtra)
library(e1071)
source("PCA/PCAfncs.R")
pr <- prcomp(USArrests)  # inappropriate, need scaling
biplot(pr)
```



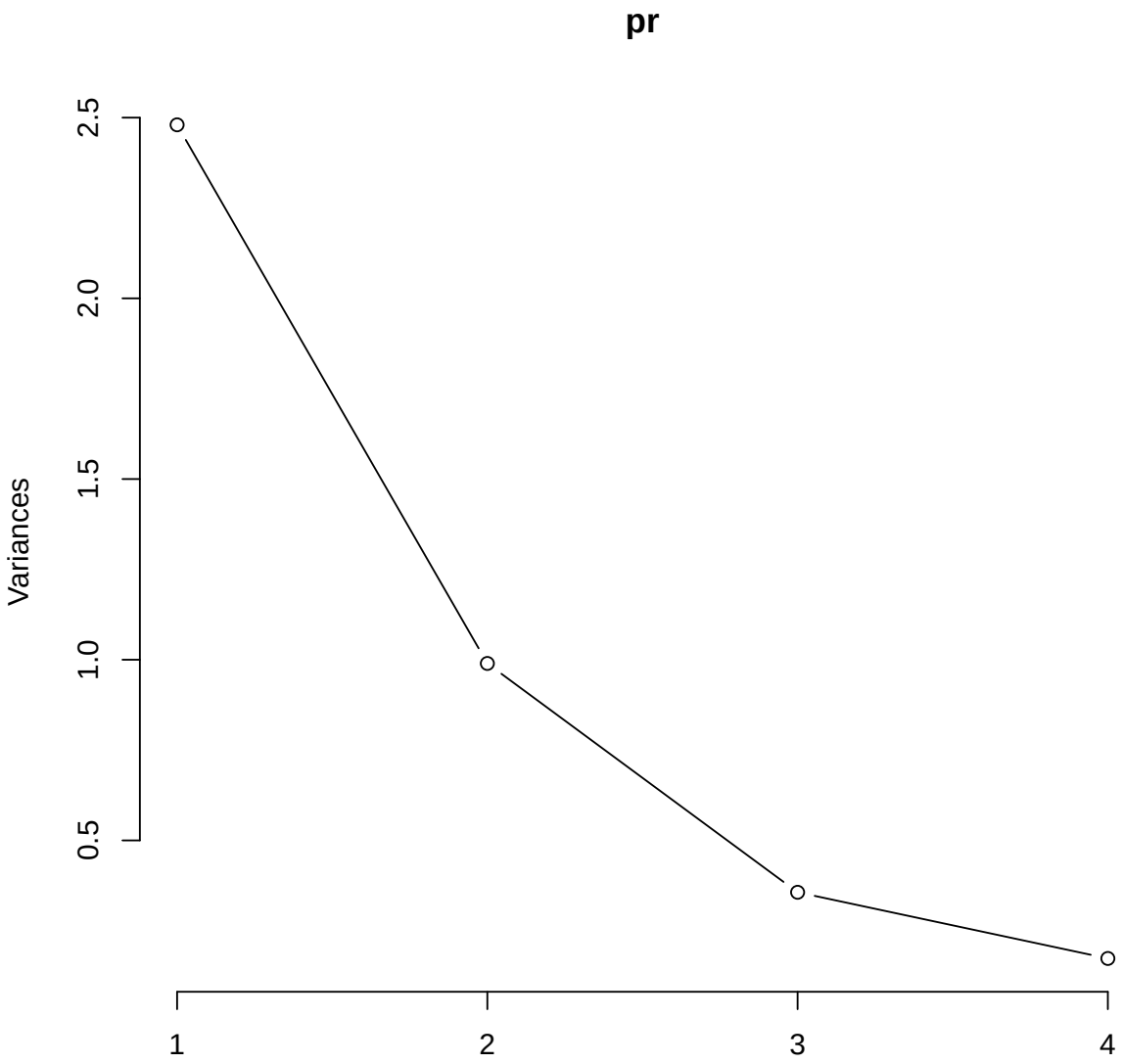

```
pr <- prcomp(USArrests, scale = TRUE)  # OK
biplot(pr)
```



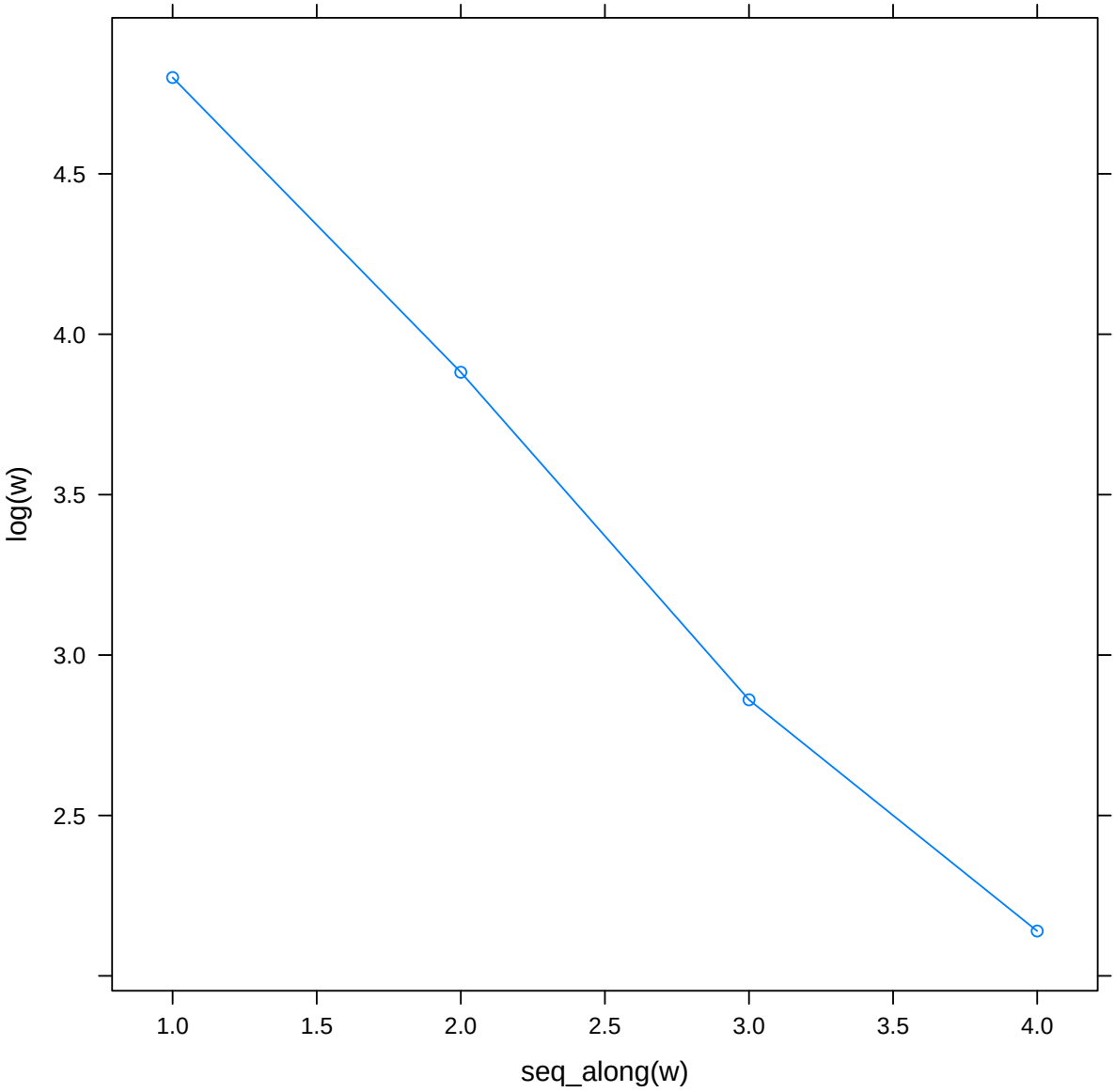

```
pr <- prcomp(~., data = USArrests, scale = TRUE)
plot(pr)
```



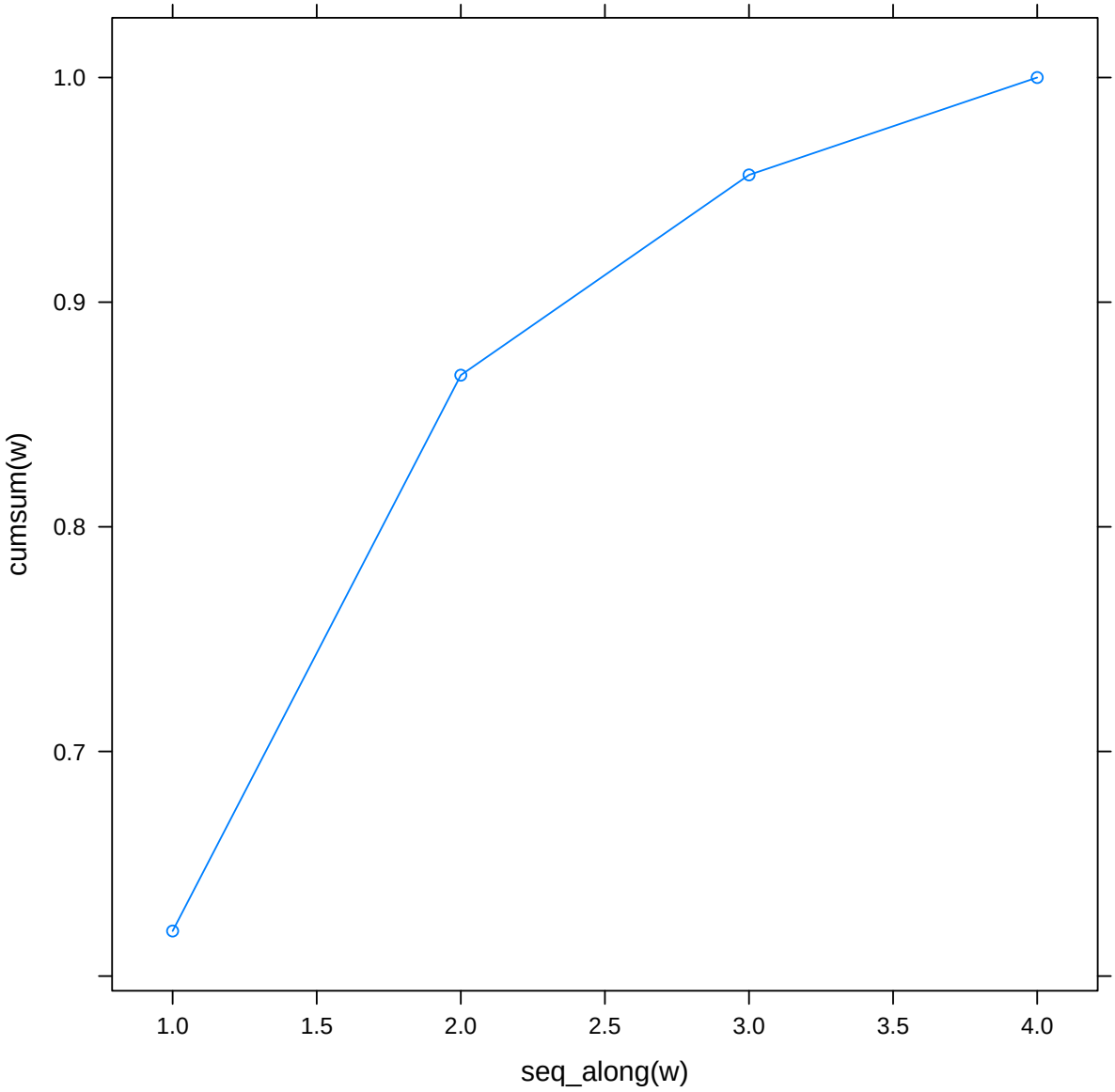

```r
summary(pr)

## Importance of components:
##                           PC1    PC2     PC3     PC4
## Standard deviation     1.5749 0.9949 0.59713 0.41645
## Proportion of Variance 0.6201 0.2474 0.08914 0.04336
## Cumulative Proportion  0.6201 0.8675 0.95664 1.00000

screeplot(pr, type = "lines")
```



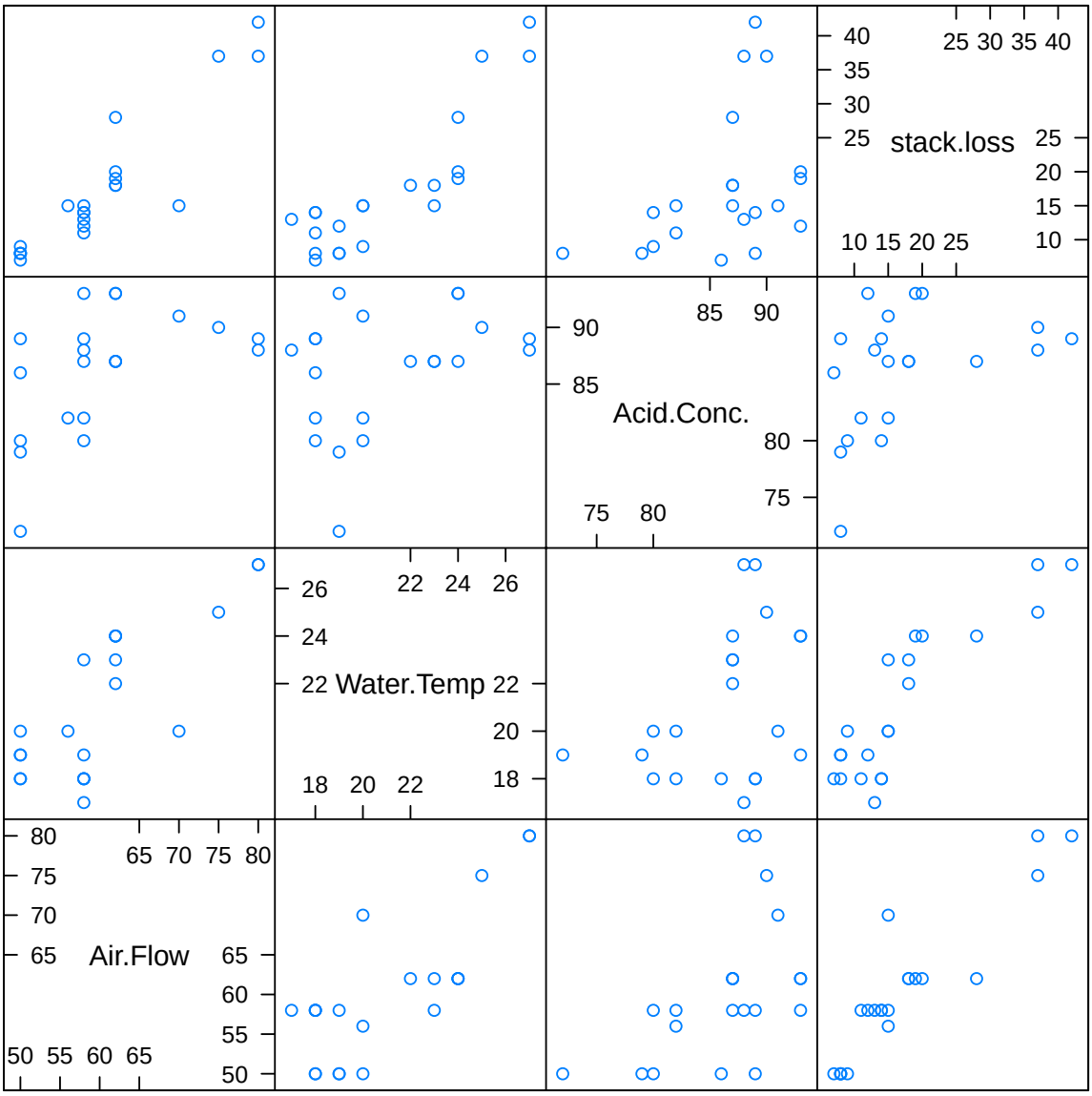

```
logweightsplot(pr, type = "b")
```




```
cumweightsplot(pr, type = "b")
```

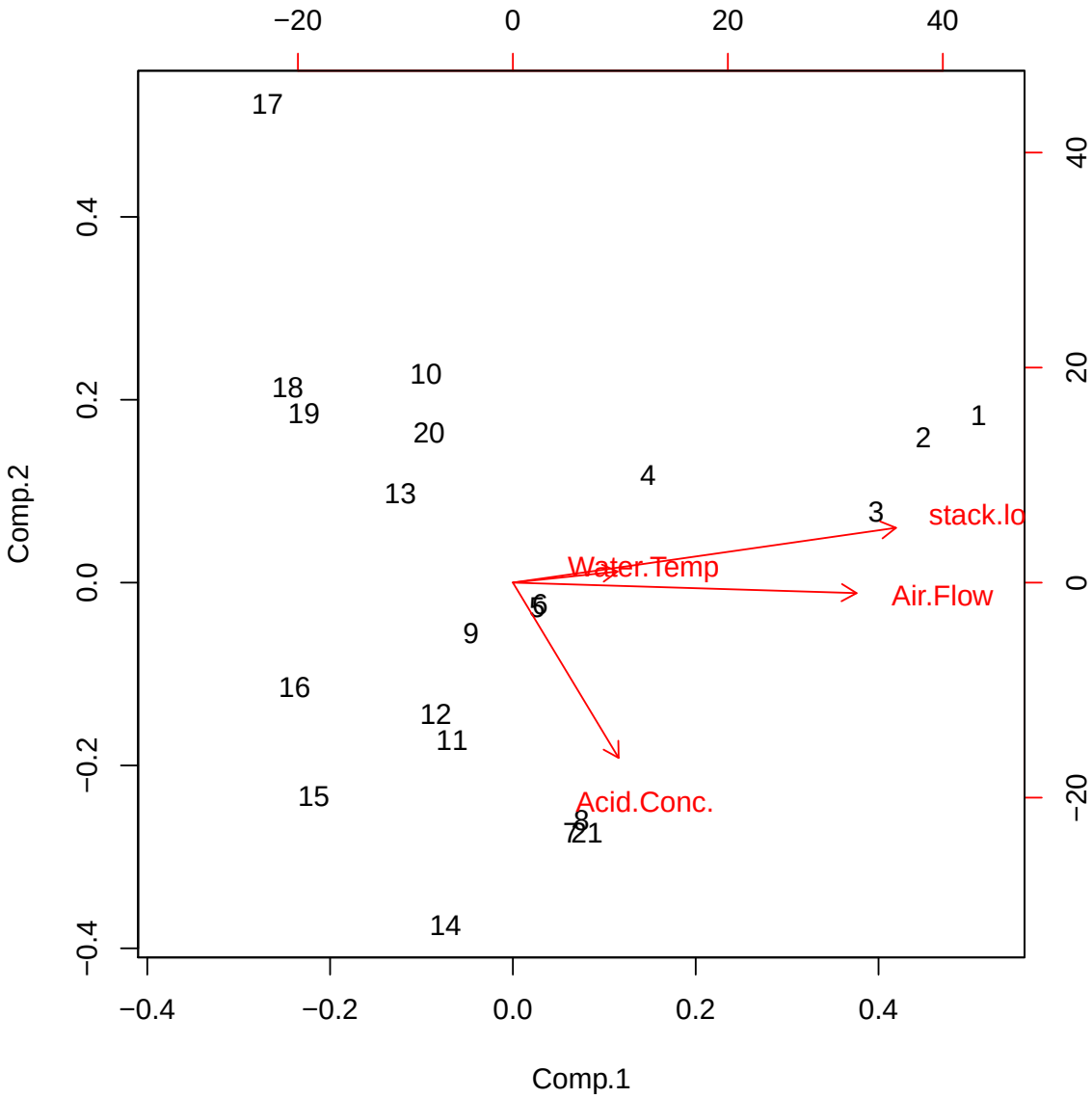



```
# princomp () is yet another one Robust
# PCA
splom(stackloss)
```

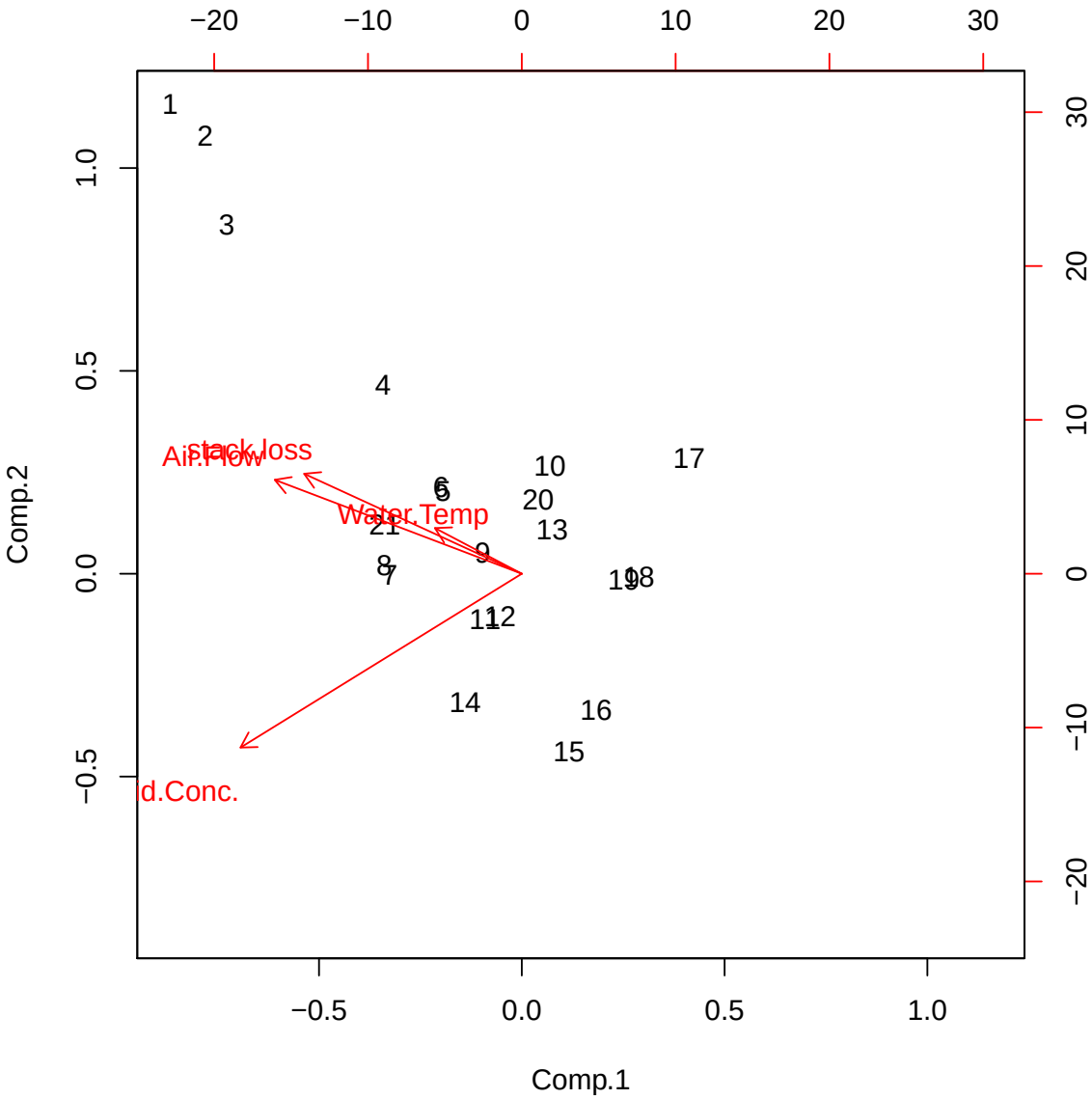


Scatter Plot Matrix

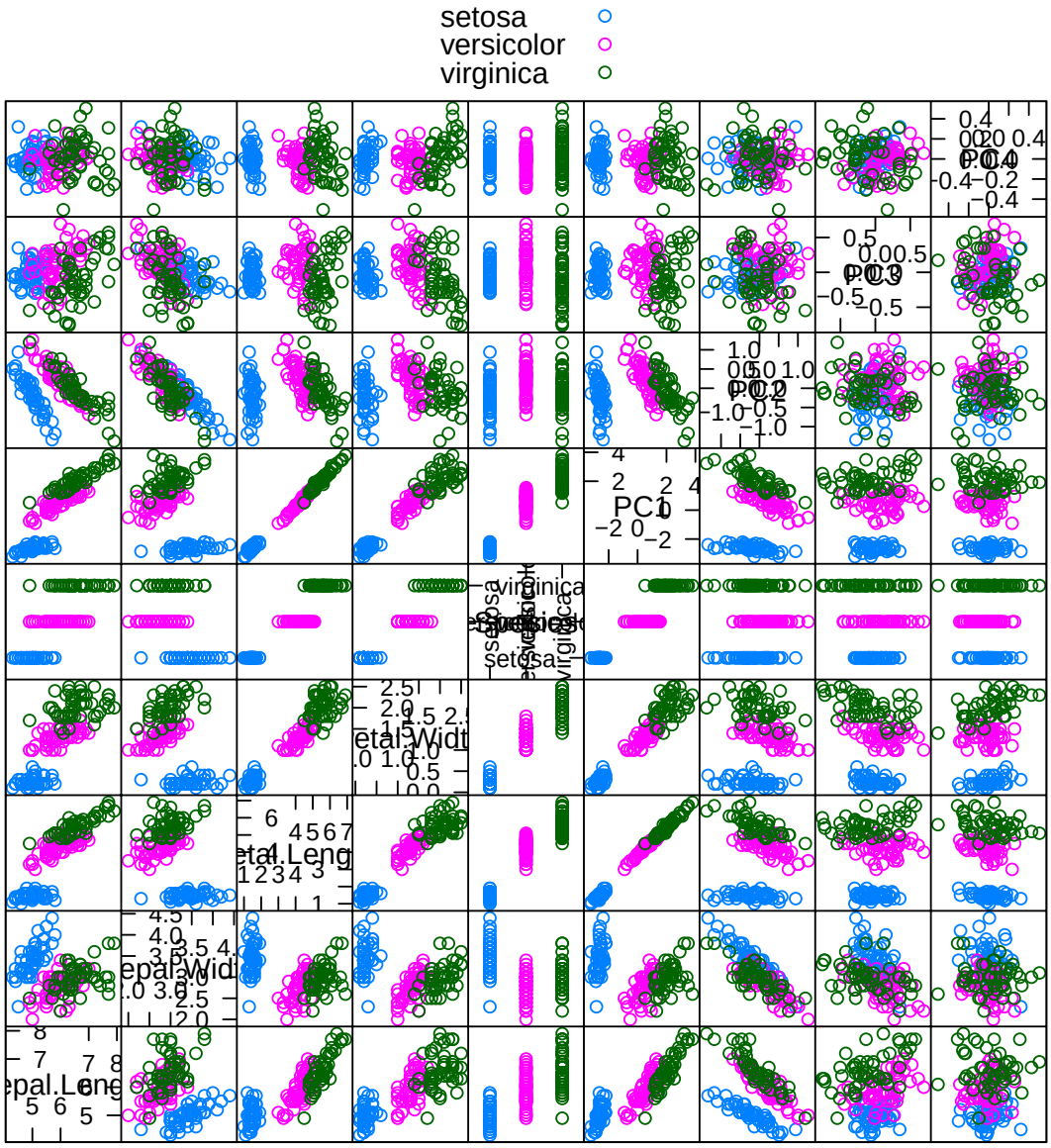
```
pc <- princomp(stackloss)
pc.rob <- princomp(stackloss, covmat = MASS::cov.rob(stackloss))
biplot(pc)
```




```
biplot(pc.rob)
```

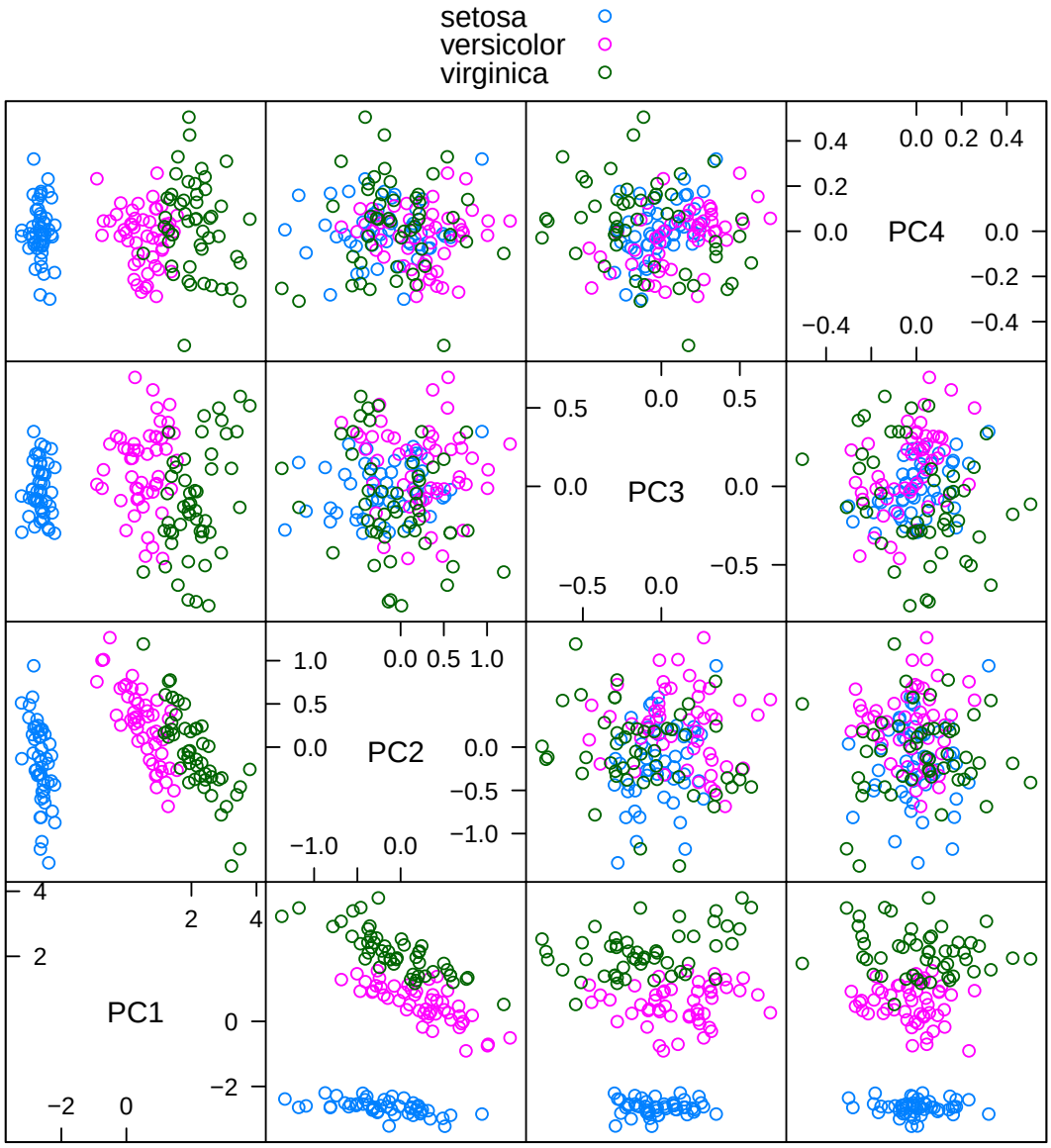



```
pc <- prcomp(~. - Species, data = iris)
splom(cbind(iris, predict(pc)), groups = iris$Species,
    auto.key = TRUE)
```



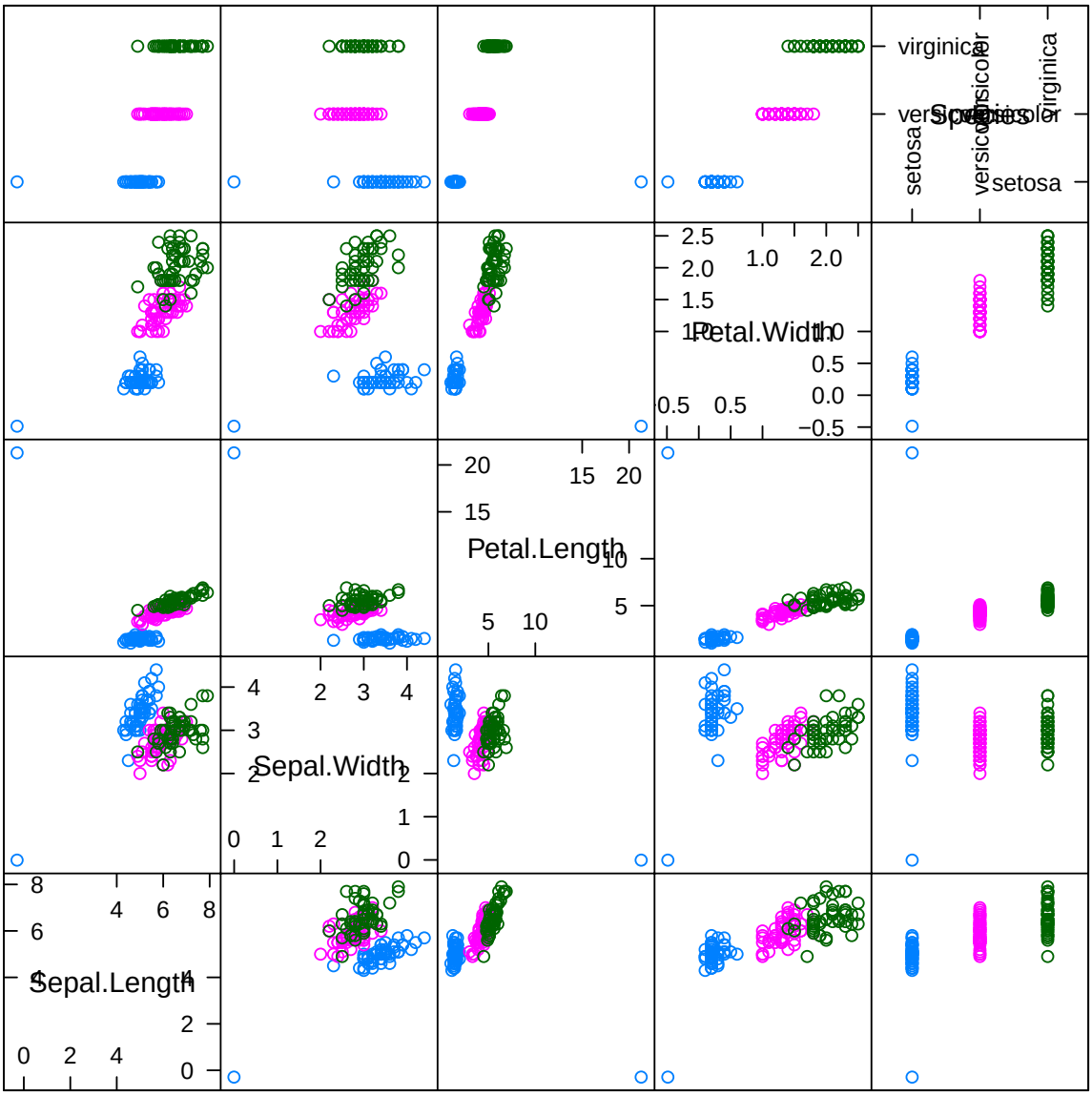
Scatter Plot Matrix

```
splom(predict(pc), groups = iris$Species,
    auto.key = TRUE)
```



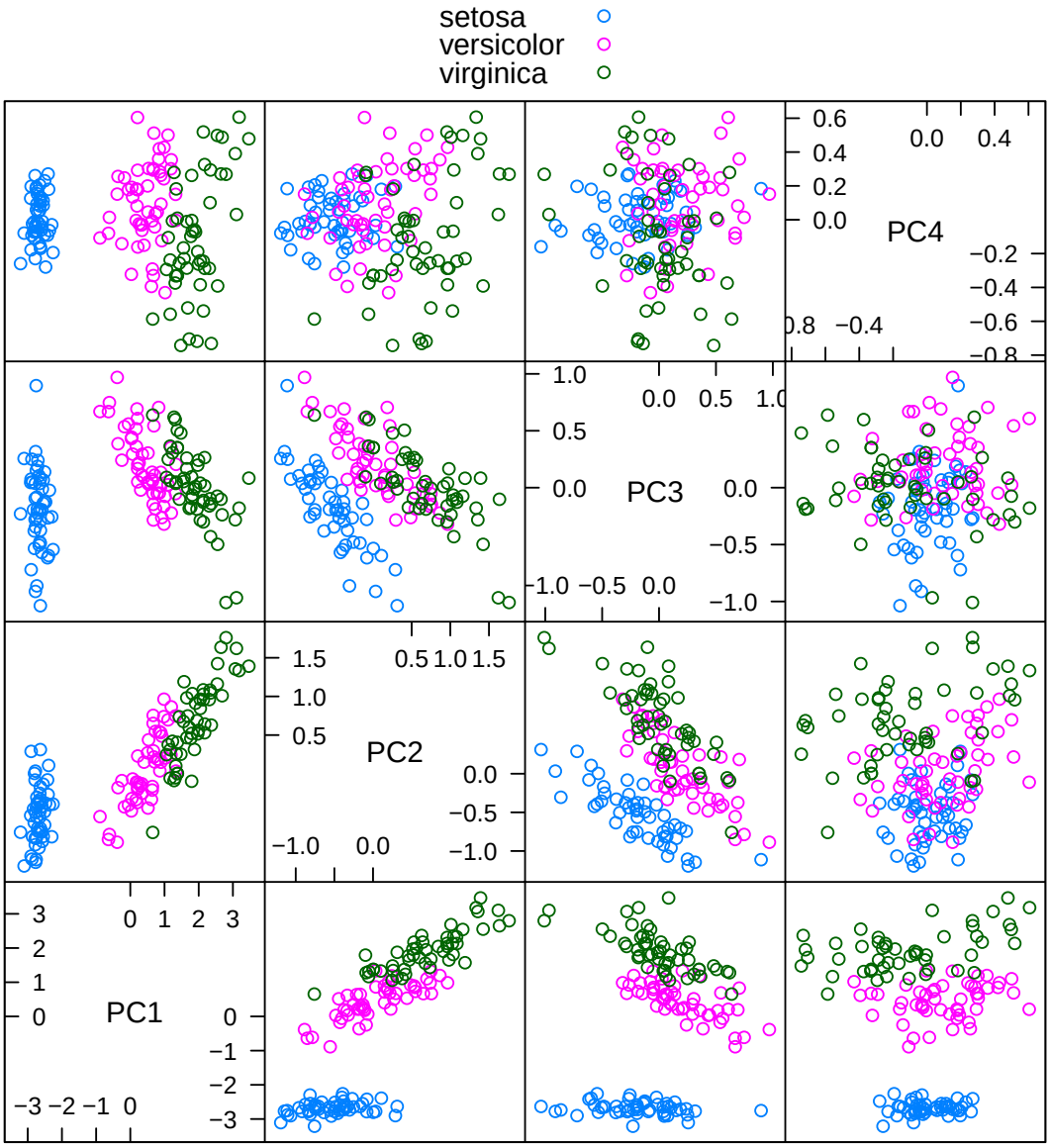
Scatter Plot Matrix

```
k <- 1
iris.out <- data.frame(Species = sample(iris$Species,
    size = k, replace = TRUE), Sepal.Length = rcauchy(k),
    Sepal.Width = rcauchy(k), Petal.Length = rcauchy(k),
    Petal.Width = rcauchy(k))
iris.spoiled <- rbind(iris, iris.out)
splom(iris.spoiled, groups = iris.spoiled$Species)
```



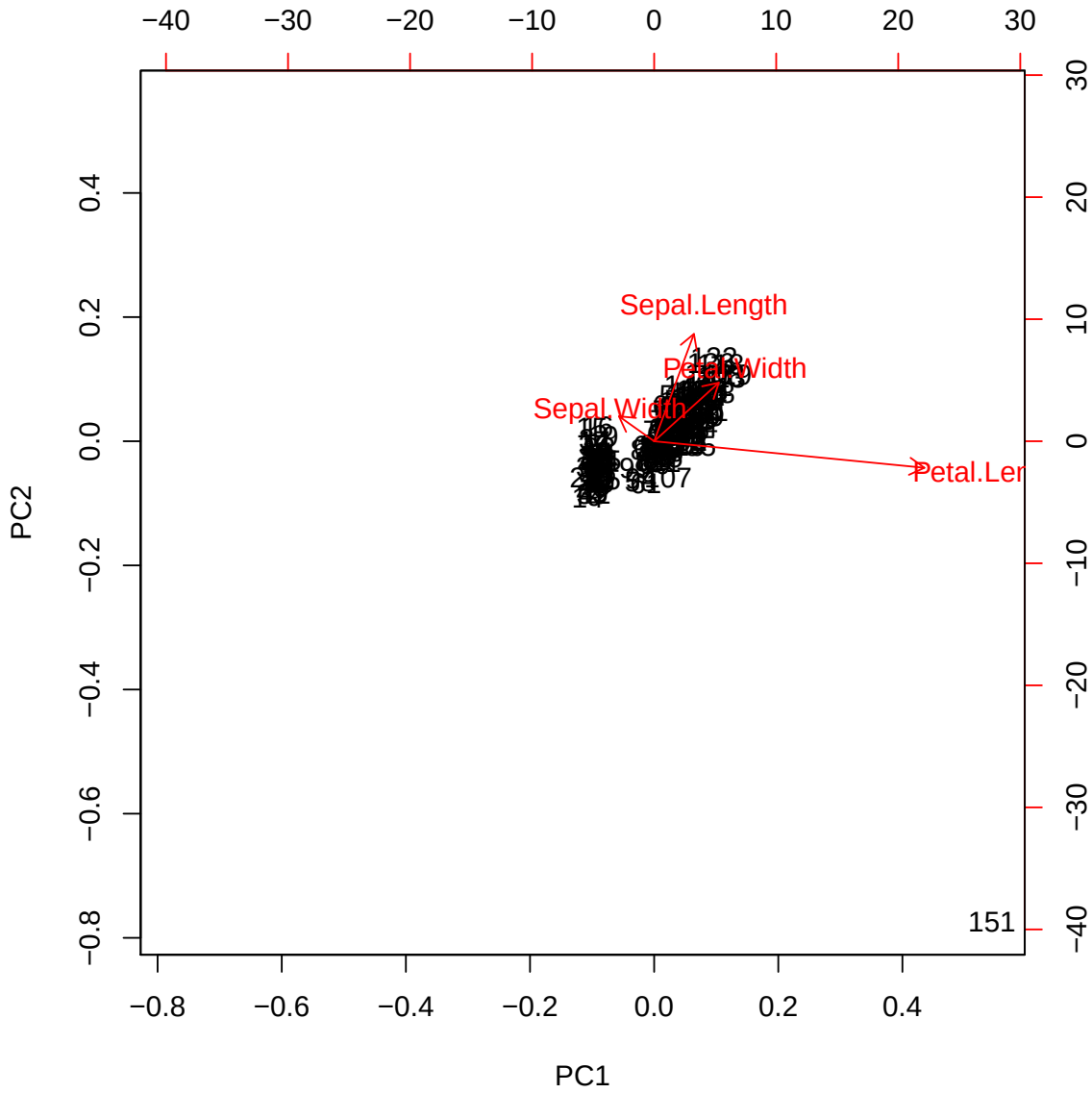
Scatter Plot Matrix

```
# pc <- princomp(iris.out[, -5]) pc.rob
# <- princomp(iris.out[, -5], covmat =
# MASS::cov.rob(iris.out[, -5]))
pc.rob <- prcomp(~. - Species, data = iris.spoiled,
    use.robust.scaling = TRUE, use.robust.cov = TRUE)
pc <- prcomp(~. - Species, data = iris.spoiled)
splom(predict(pc, iris), groups = iris$Species,
    auto.key = TRUE)
```

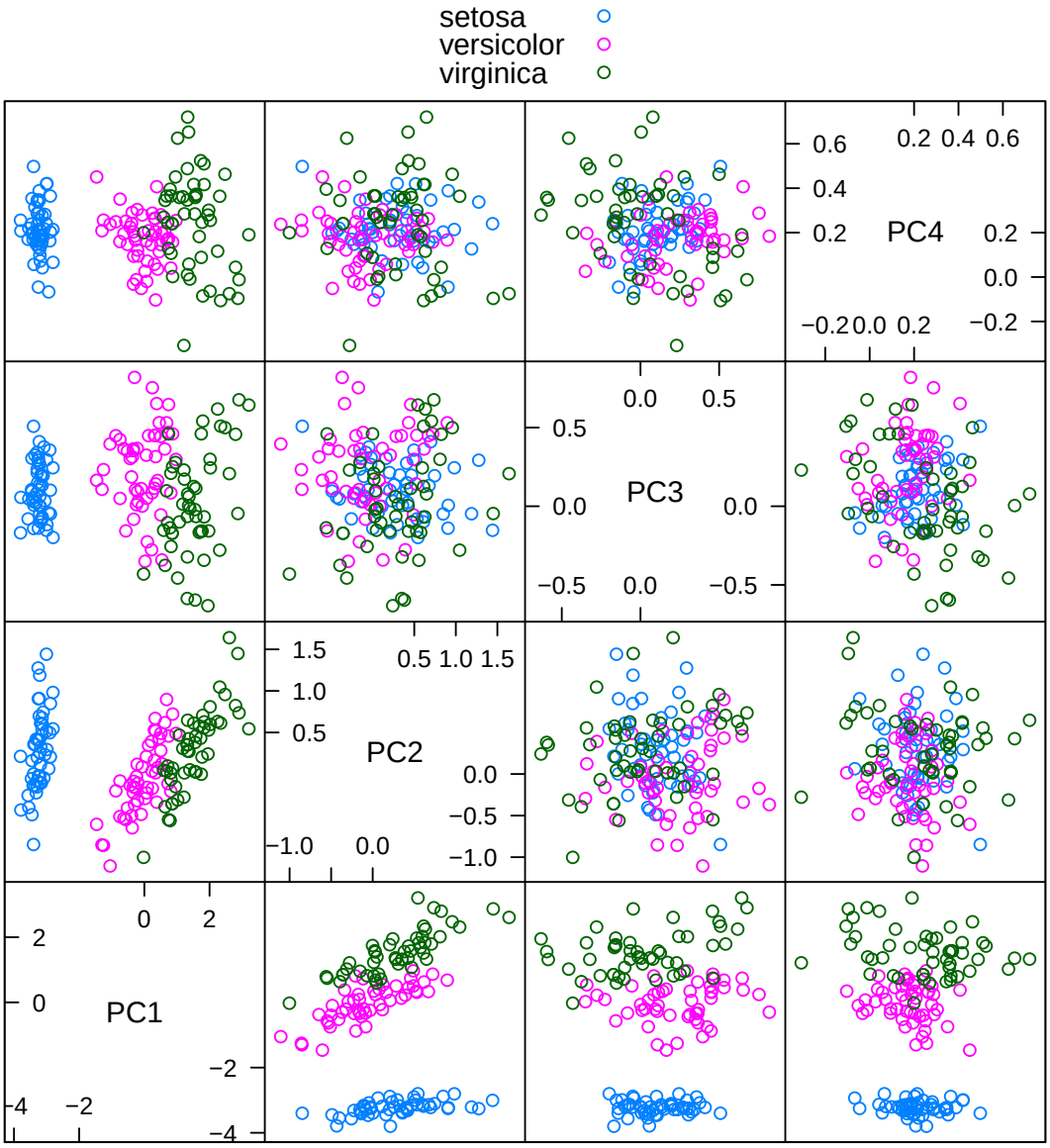


Scatter Plot Matrix

```
biplot(pc)
```

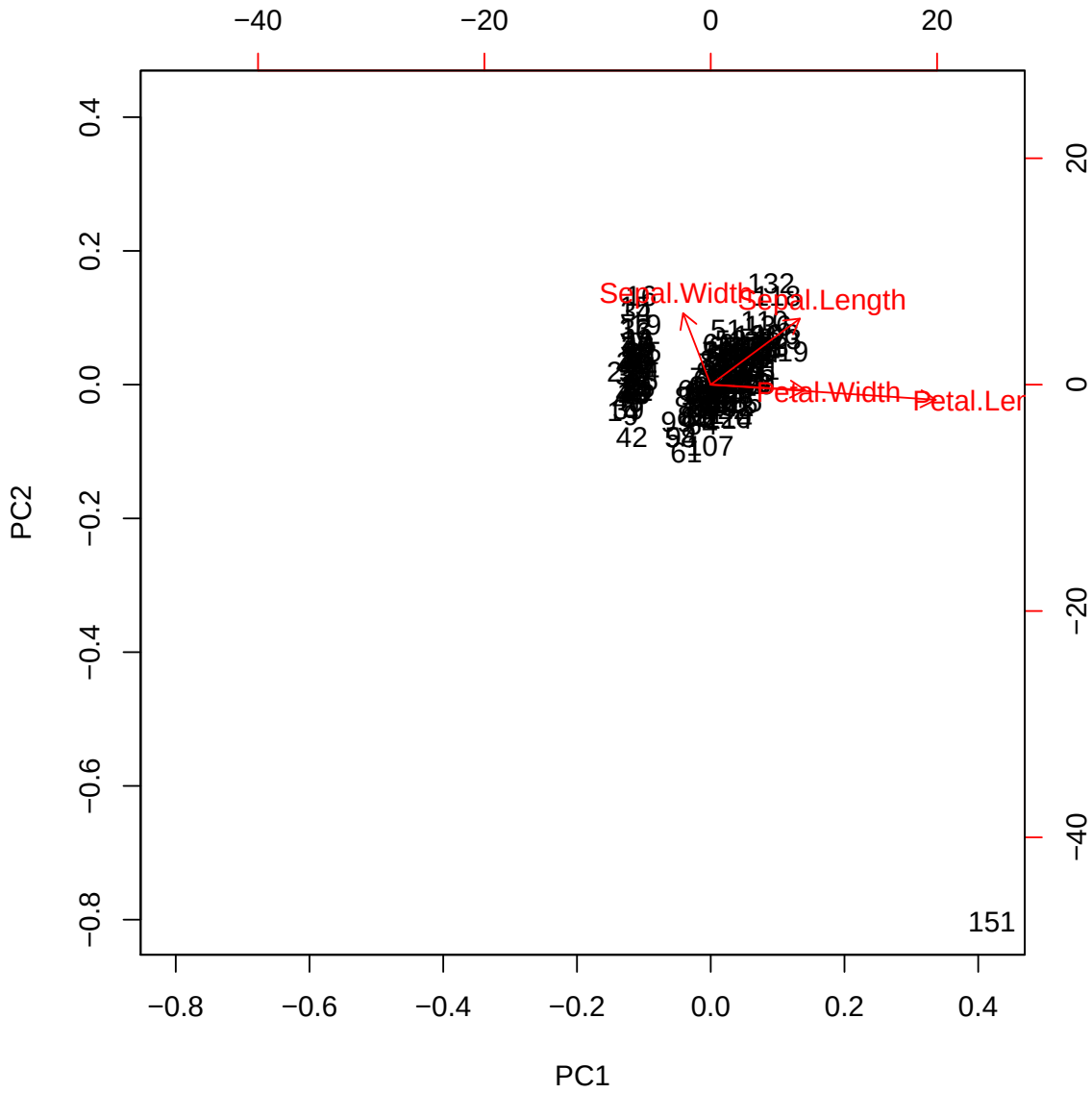



```
splom(predict(pc.rob, iris), groups = iris$Species,
    auto.key = TRUE)
```



Scatter Plot Matrix

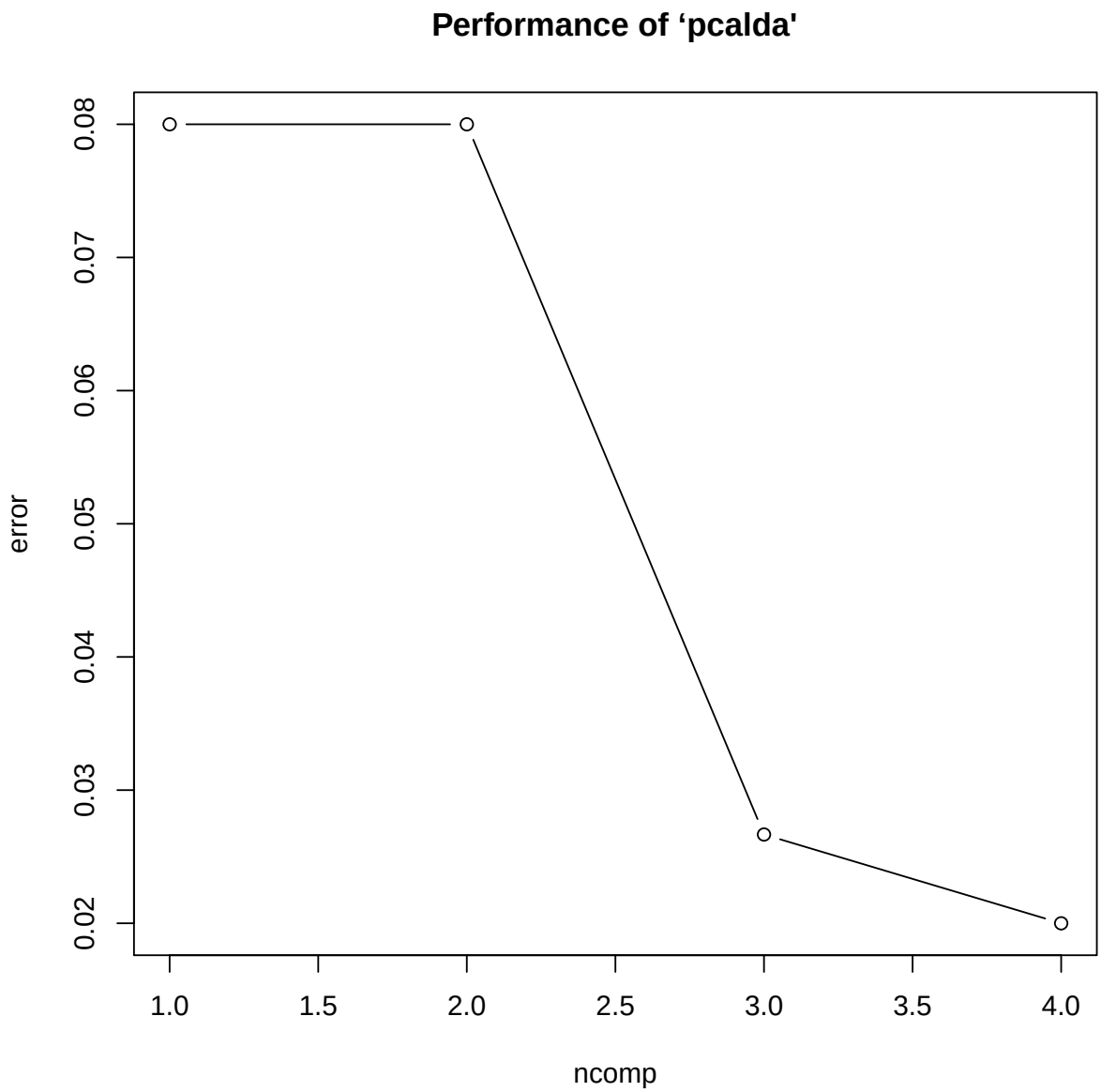
```
biplot(pc.rob)
```



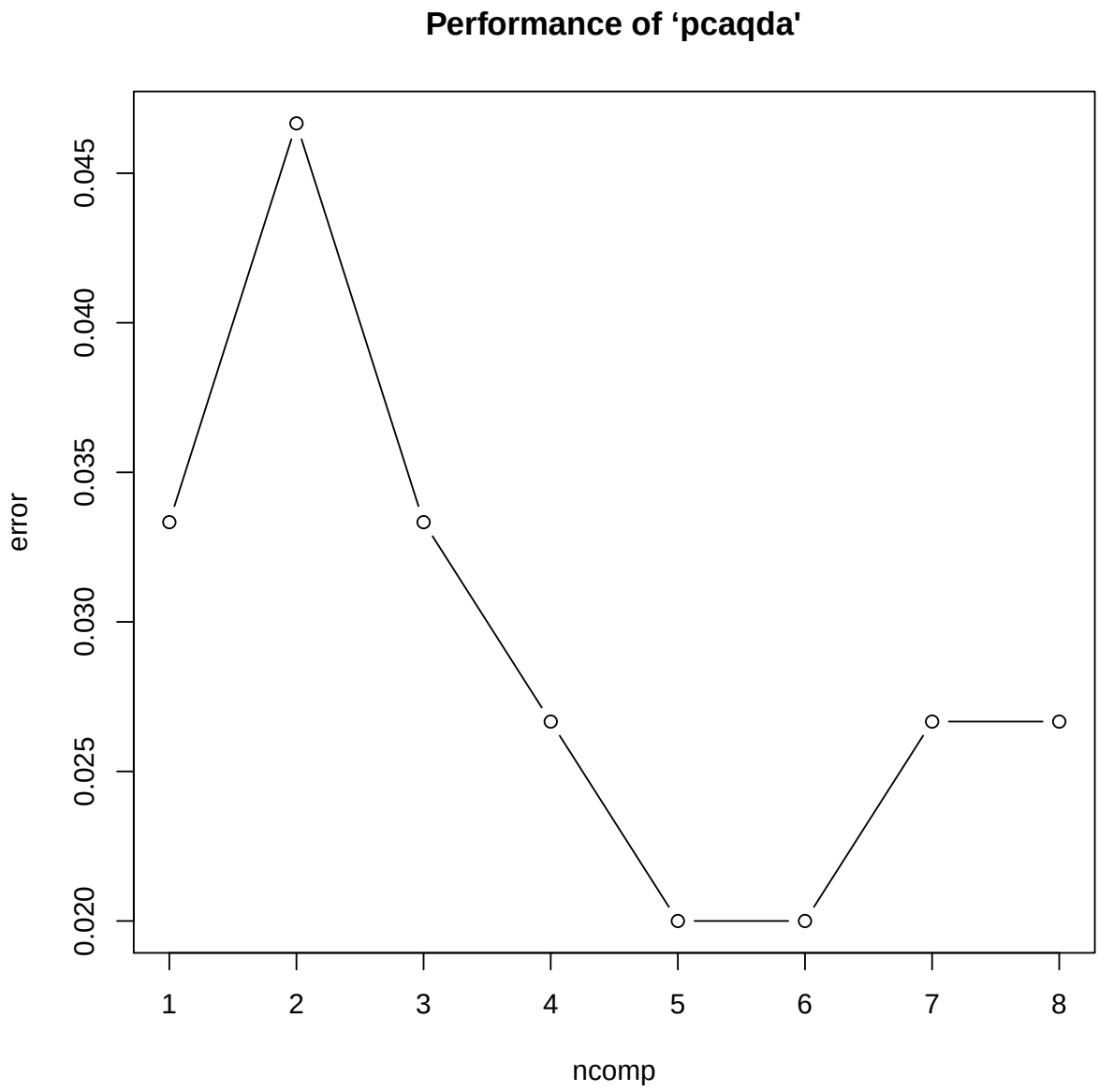
7.3 PCA-LDA for 'iris'

```
# pcalda test
train.idx <- sample(seq_len(nrow(iris)),
    size = 2/3 * nrow(iris))
iris.test <- iris[-train.idx, ]
iris.train <- iris[train.idx, ]
iris.test.Species <- iris.test$Species
iris.test$Species <- NULL
pcalda <- function(...) pcawrap(lda, ...)
predict.pcalda <- function(...) predict(...)$class
plda <- pcalda(Species ~ ., data = iris.train,
    ncomp = 1, scale = TRUE)
table(actual = iris.test.Species, predicted = predict(plda,
```

```
    iris.test)$class)

##             predicted
## actual       setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         14         2
##   virginica       0          1        23

tn <- tune(pcalda, Species ~ ., data = iris,
    ranges = list(ncomp = 1:4), predict.func = predict.pcalda,
    tunecontrol = tune.control(cross = nrow(iris)))
summary(tn)

##
## Parameter tuning of 'pcalda':
##
## - sampling method: leave-one-out
##
## - best parameters:
##  ncomp
##      4
##
## - best performance: 0.02
##
## - Detailed performance results:
##   ncomp      error dispersion
## 1     1 0.08000000  0.2722021
## 2     2 0.08000000  0.2722021
## 3     3 0.02666667  0.1616470
## 4     4 0.02000000  0.1404690

plot(tn)
```




```
pcaqda <- function(...) pcawrap(qda, ...)
predict.pcaqda <- function(...) predict(...)$class
sigma <- 0.05
iris$trash1 <- iris$Sepal.Length + iris$Petal.Length +
    rnorm(nrow(iris), sd = sigma)
iris$trash2 <- iris$Sepal.Length - iris$Petal.Length +
    rnorm(nrow(iris), sd = sigma)
iris$trash3 <- iris$Sepal.Length + 2 * iris$Petal.Length +
    rnorm(nrow(iris), sd = sigma)
iris$trash4 <- iris$Sepal.Length - 2 * iris$Petal.Length +
    rnorm(nrow(iris), sd = sigma)
tn <- tune(pcaqda, Species ~ ., data = iris,
    ranges = list(ncomp = 1:8), predict.func = predict.pcaqda,
    tunecontrol = tune.control(cross = nrow(iris)))
plot(tn)
```

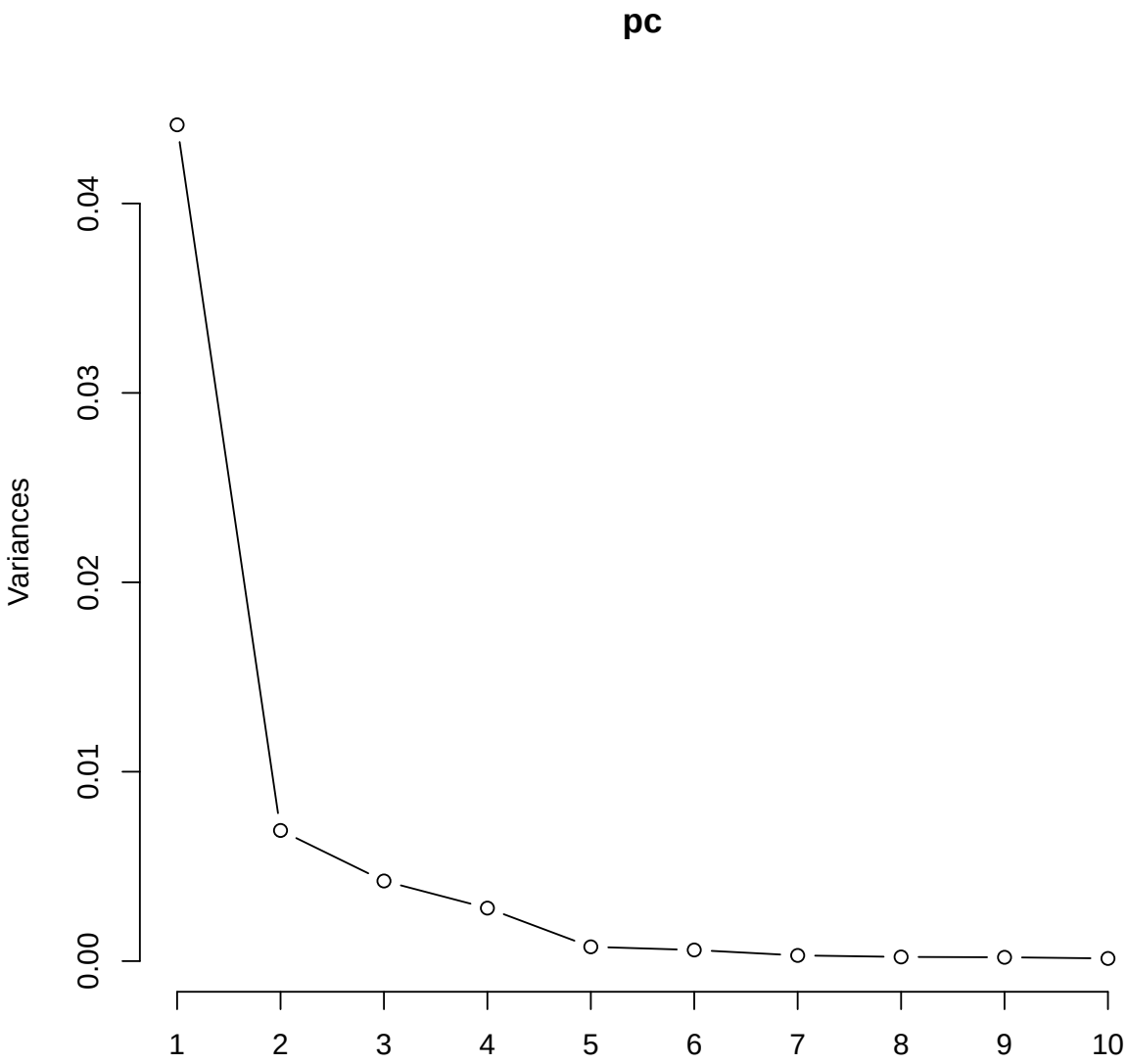


7.4 PCA-LM for 'gasoline'

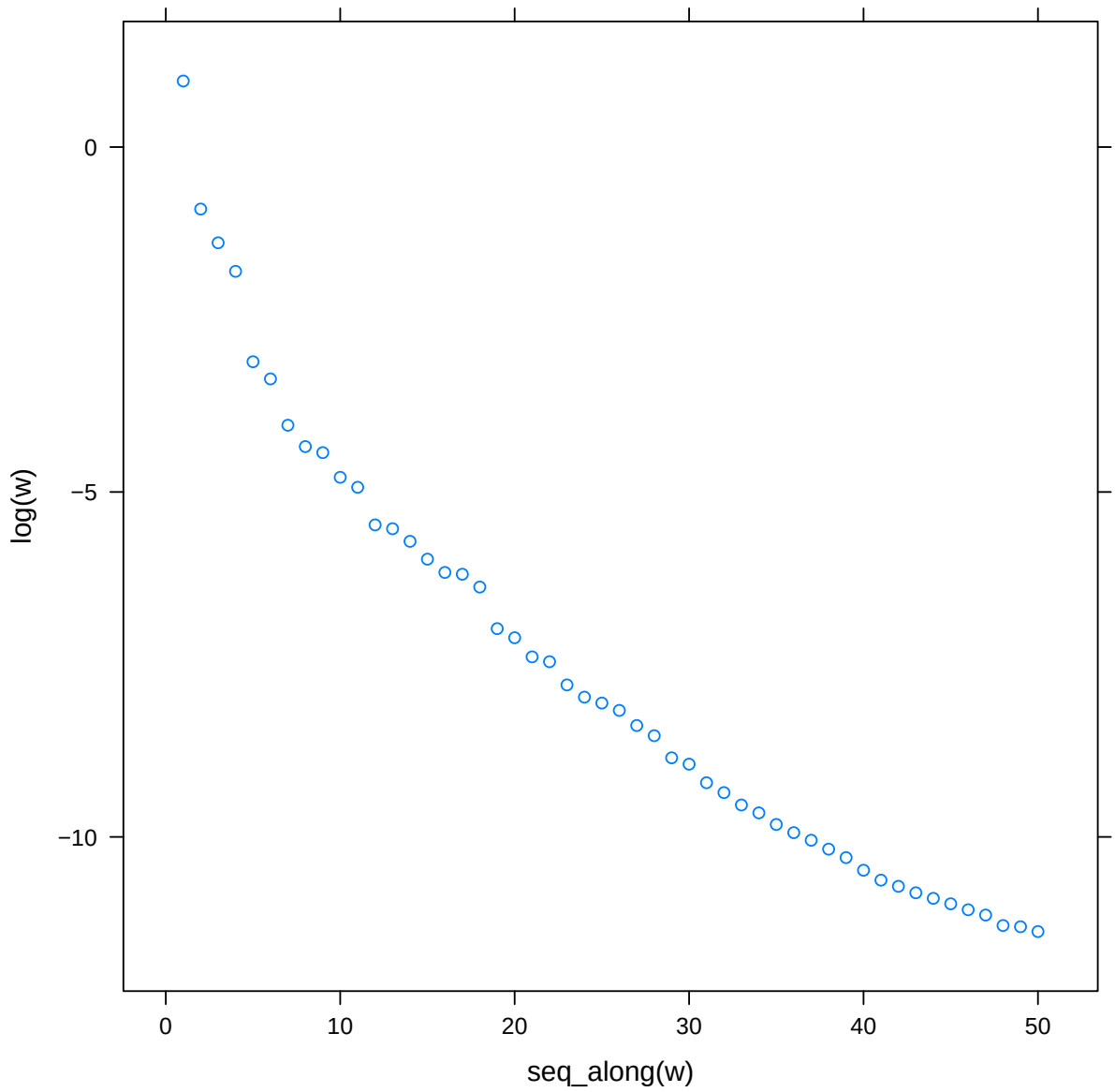
```
library(lattice)
library(latticeExtra)
library(pls)
library(MASS)
library(e1071)
data(gasoline)
names(gasoline)

## [1] "octane" "NIR"

dim(gasoline)
```

```
## [1] 60  2

class(gasoline$NIR) <- NULL
colnames(gasoline$NIR) <- paste("S", seq(900,
    1700, 2), sep = "")
gasoline <- cbind(subset(gasoline, select = octane),
    as.data.frame(gasoline$NIR))
dim(gasoline)

## [1]  60 402

l <- lm(octane ~ ., data = gasoline)
coef(l)[1:70]

## (Intercept)        S900        S902        S904        S906
##    483.1856 -24260.3171 -18267.4559  15989.9042 -14110.4715
##        S908        S910        S912        S914        S916
##  21111.9165  17721.0489  -6133.9016   -486.7148   6511.4456
##        S918        S920        S922        S924        S926
##  -4253.3466 -32548.3782  24675.4370   1871.8213 -22699.8907
##        S928        S930        S932        S934        S936
##  37521.8093  -4189.9107  -1284.8007  26831.7811  24394.2217
##        S938        S940        S942        S944        S946
##   6157.5540 -62075.6004 -76762.3711  51875.9328 -20214.4881
##        S948        S950        S952        S954        S956
## -26627.6883  37529.8105 -15734.7199  87375.7664 -39180.4653
##        S958        S960        S962        S964        S966
## -23517.7801  21175.4795  28376.4367  56671.2136 -46634.0914
##        S968        S970        S972        S974        S976
## -17618.9349  76039.4972  -7797.7093 -68962.8609 -54896.1171
##        S978        S980        S982        S984        S986
##  18649.6597  22834.2249 -57093.0827  18209.8438  37029.9733
##        S988        S990        S992        S994        S996
##   8335.4738  17109.9673 -34620.8186 -91225.9688  96890.9404
##        S998       S1000       S1002       S1004       S1006
##  20509.7228  53535.2166   4488.3995  49531.3777 -65202.7703
##       S1008       S1010       S1012       S1014       S1016
##   9533.3868 -88366.9080  45037.8874  15693.9422 -28076.9493
##       S1018       S1020       S1022       S1024       S1026
##          NA          NA          NA          NA          NA
##       S1028       S1030       S1032       S1034       S1036
##          NA          NA          NA          NA          NA

tune(lm, octane ~ ., data = gasoline)

## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
```

```
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading
## Warning in predict.lm(model, if (!is.null(validation.x)) validation.x else
if (useFormula) data[-train.ind[[sample]], : prediction from a rank-deficient
fit may be misleading

##
## Error estimation of 'lm' using 10-fold cross validation: 178712.7

pc <- prcomp(~. - octane, data = gasoline)
plot(pc, type = "lines")
```

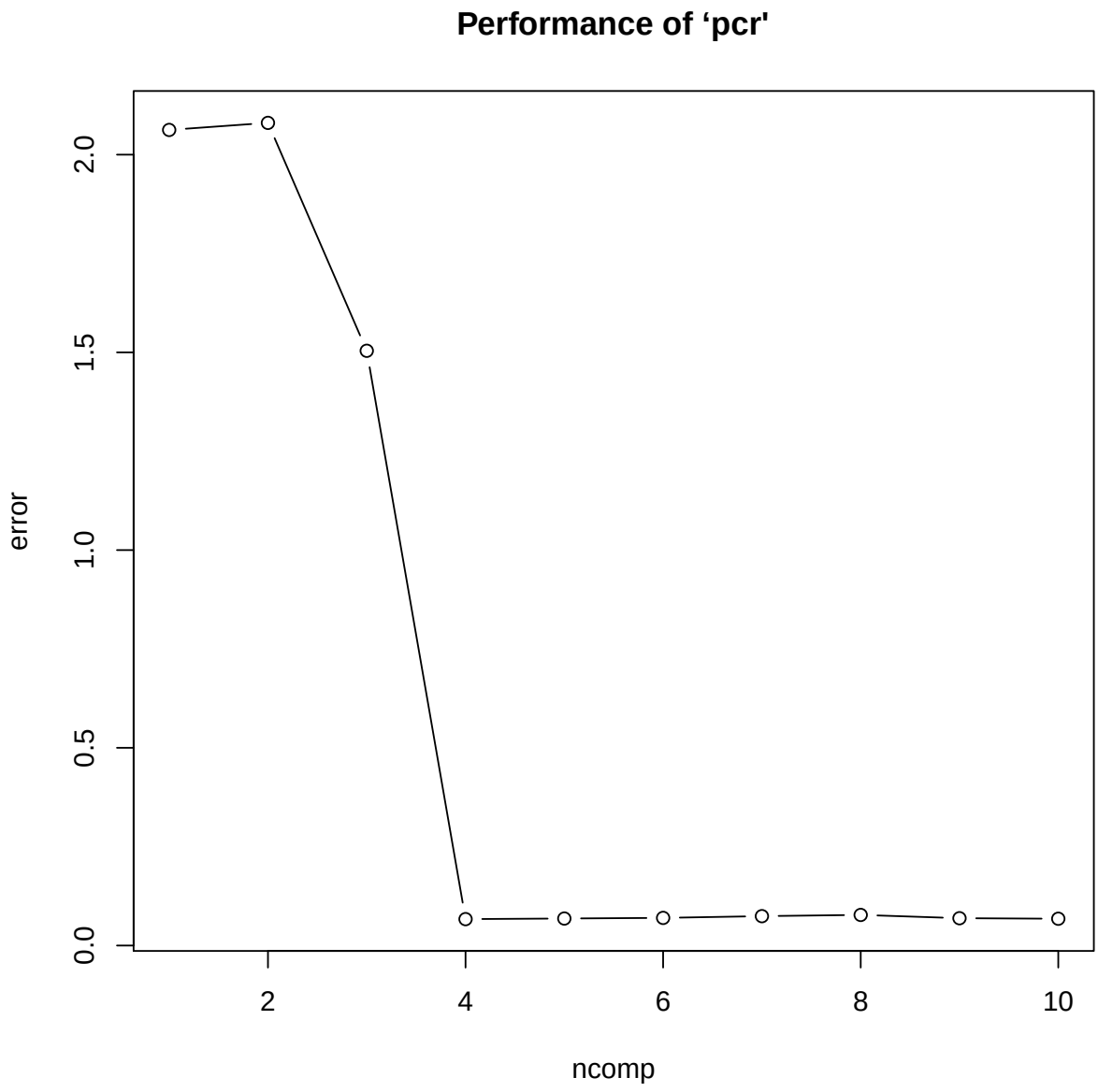



```
logweightsplot(pc)
```

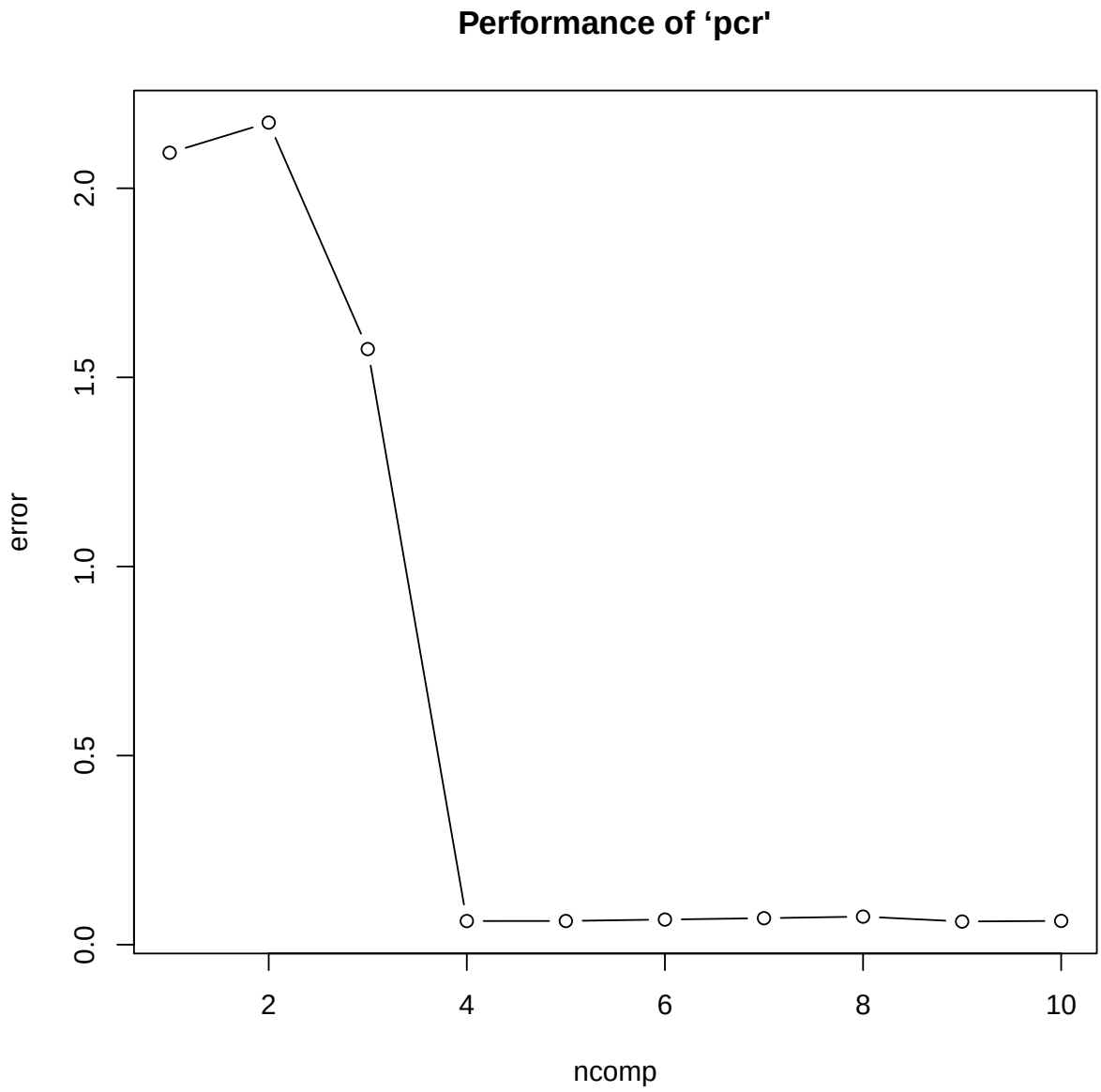



```r
pl <- pcr(octane ~ ., data = gasoline, ncomp = 6)
summary(pl)

## Data:  X dimension: 60 401
##  Y dimension: 60 1
## Fit method: svdpc
## Number of components considered: 6
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps
## X         72.57    83.90    90.86    95.46    96.70
## octane    18.99    19.62    46.50    97.69    97.78
##         6 comps
## X         97.66
## octane    97.79

# Many curses to package developers!
```

```r
my.predict.mvr <- function(object, newdata,
    ...) {
    predict(object, newdata, type = "response",
        ncomp = object$ncomp, ...)
}
tn <- tune(pcr, octane ~ ., data = gasoline,
    ranges = list(ncomp = 1:10), predict.func = my.predict.mvr,
    tunecontrol = tune.control(sampling = "cross"))
plot(tn)
```




```r
tn <- tune(pcr, octane ~ ., data = gasoline,
    ranges = list(ncomp = 1:10), predict.func = my.predict.mvr,
    tunecontrol = tune.control(sampling = "cross",
        cross = nrow(gasoline)))
plot(tn)
```



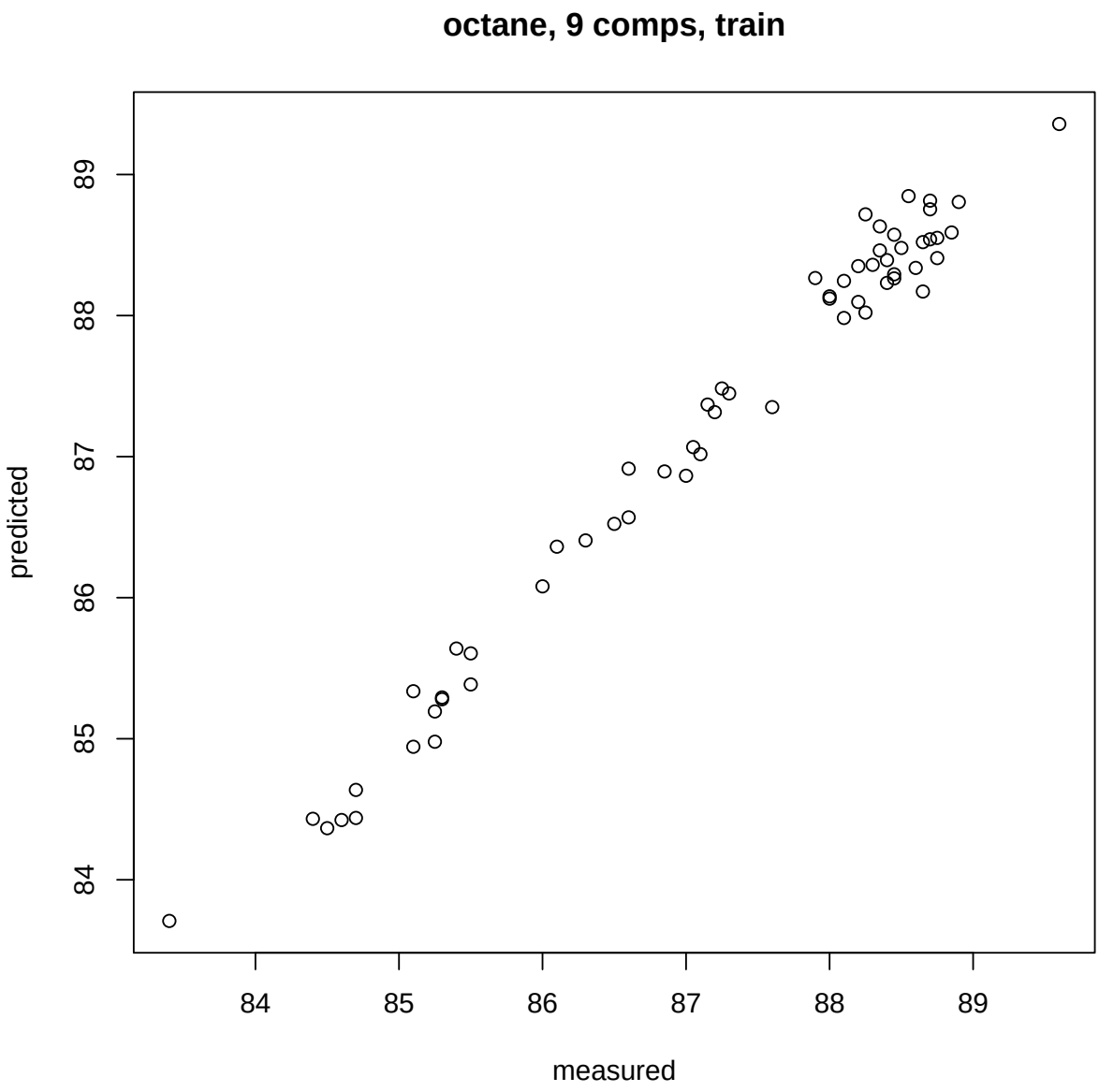

```
cv <- crossval(pl)
summary(cv)

## Data:  X dimension: 60 401
##  Y dimension: 60 1
## Fit method: svdpc
## Number of components considered: 6
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps
## CV           1.543    1.528    1.533    1.367   0.2543
## adjCV        1.543    1.520    1.525    1.371   0.2510
```

```
##        5 comps  6 comps
## CV      0.2566   0.2587
## adjCV   0.2539   0.2565
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps
## X         72.57    83.90    90.86    95.46    96.70
## octane    18.99    19.62    46.50    97.69    97.78
##         6 comps
## X         97.66
## octane    97.79
```

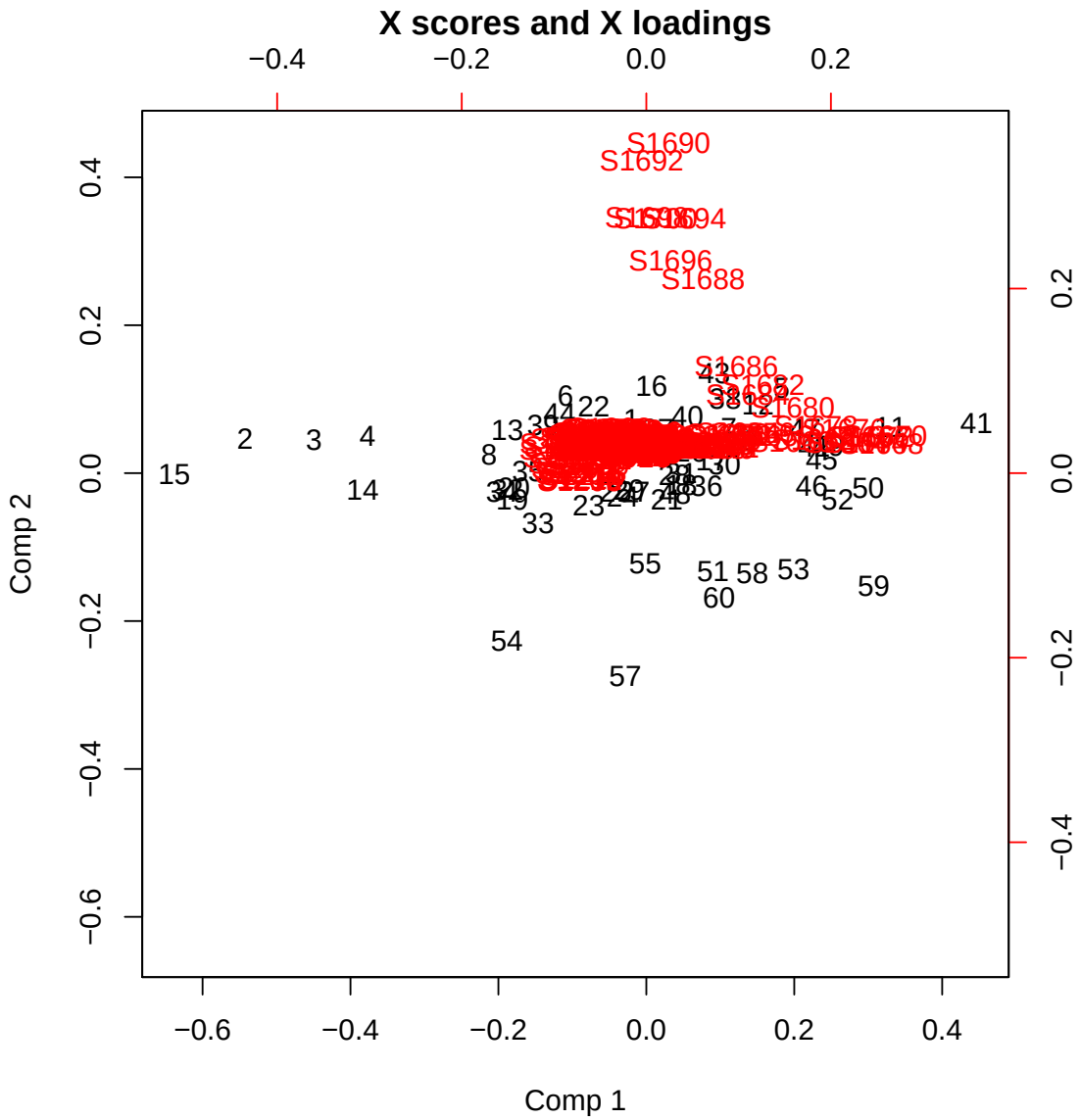
```
cv <- crossval(pl, segments = nrow(gasoline))
summary(cv)
```

```
## Data:  X dimension: 60 401
##  Y dimension: 60 1
## Fit method: svdpc
## Number of components considered: 6
##
## VALIDATION: RMSEP
## Cross-validated using 60 leave-one-out segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps
## CV           1.543    1.447    1.474    1.255   0.2501
## adjCV        1.543    1.446    1.474    1.255   0.2496
##        5 comps  6 comps
## CV      0.2503   0.2578
## adjCV   0.2500   0.2575
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps
## X         72.57    83.90    90.86    95.46    96.70
## octane    18.99    19.62    46.50    97.69    97.78
##         6 comps
## X         97.66
## octane    97.79
```

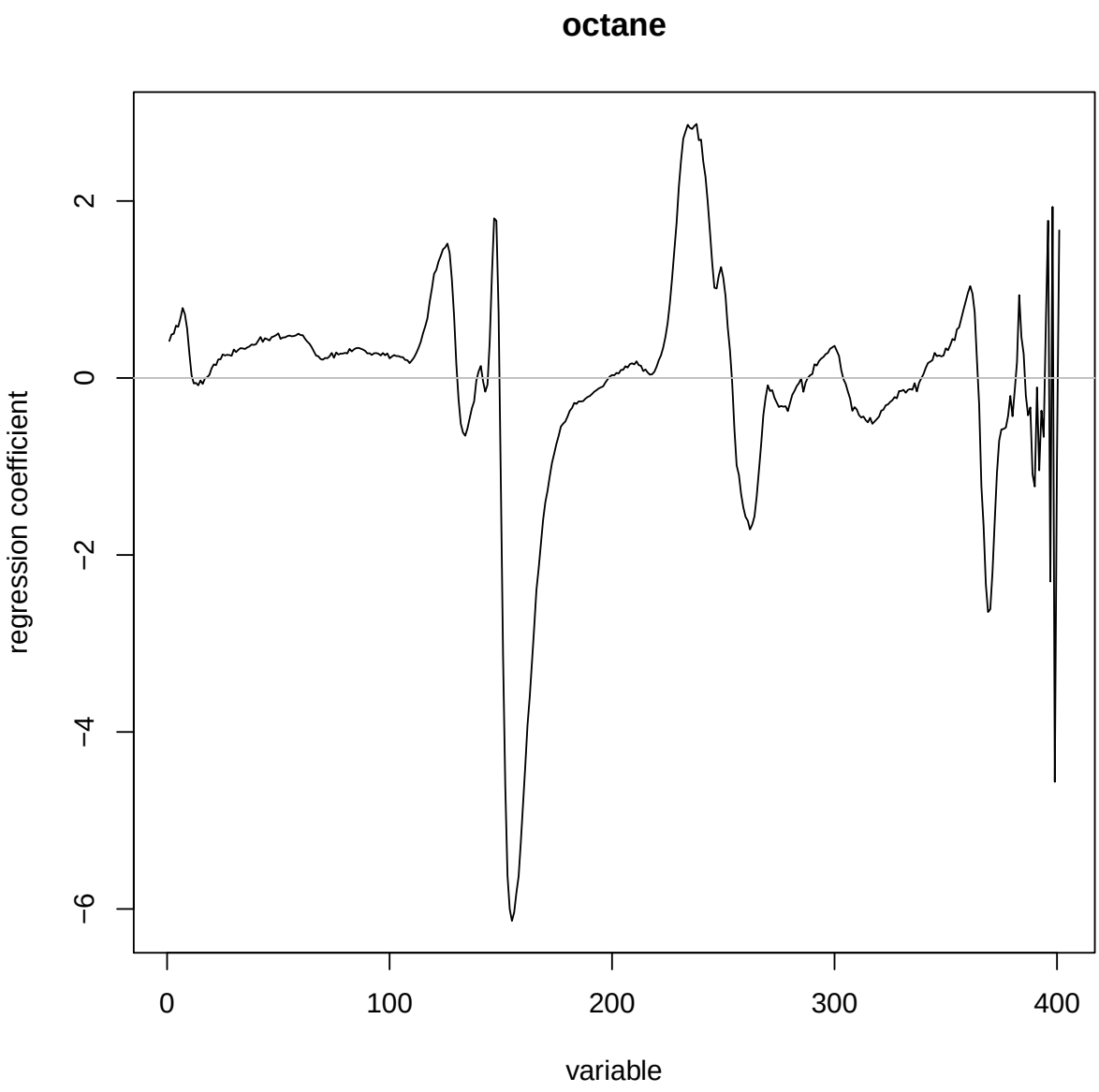
```
best.model <- tn$best.model
plot(best.model)
```



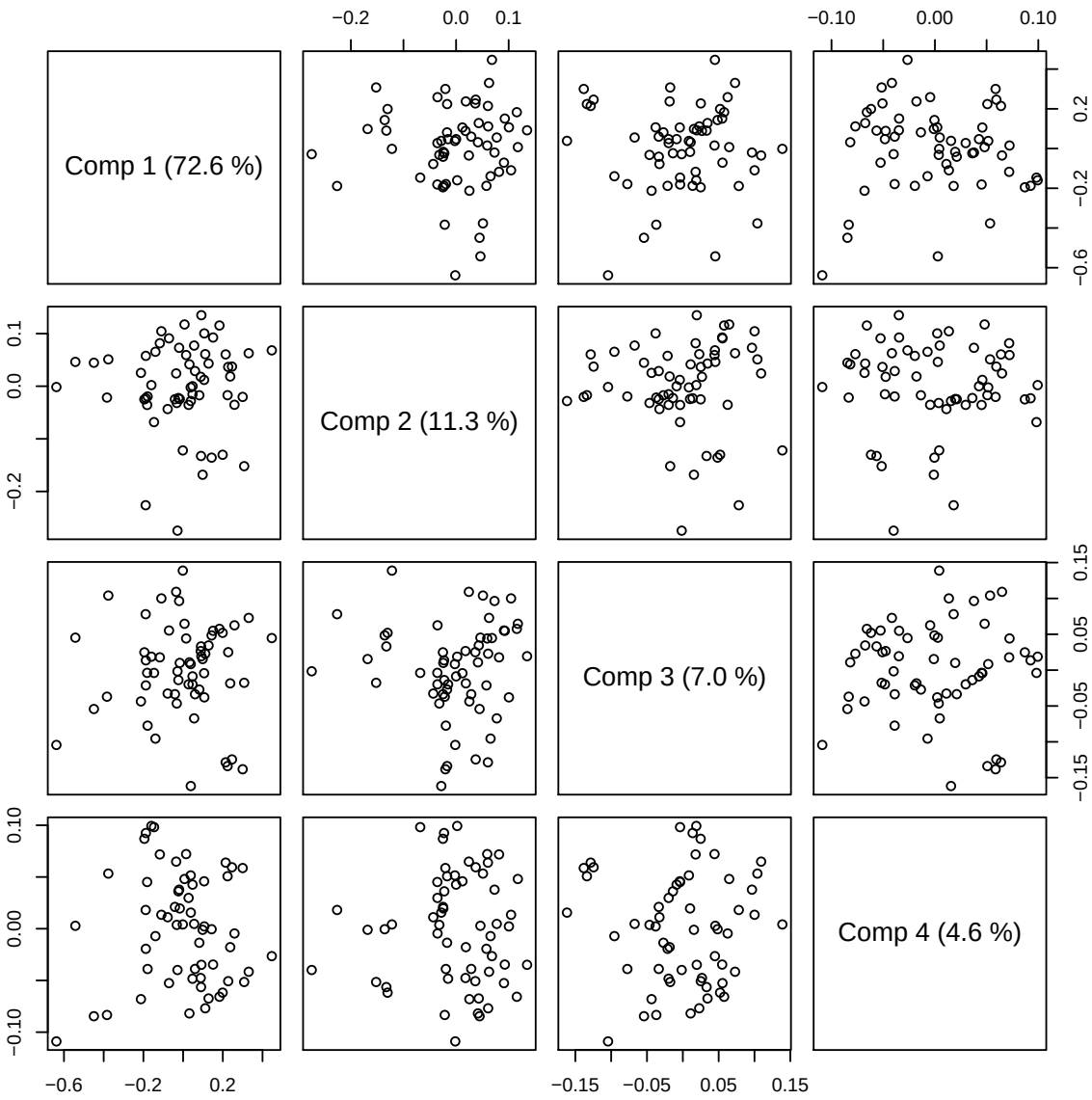

```
plot(best.model, plottype = "biplot")
```



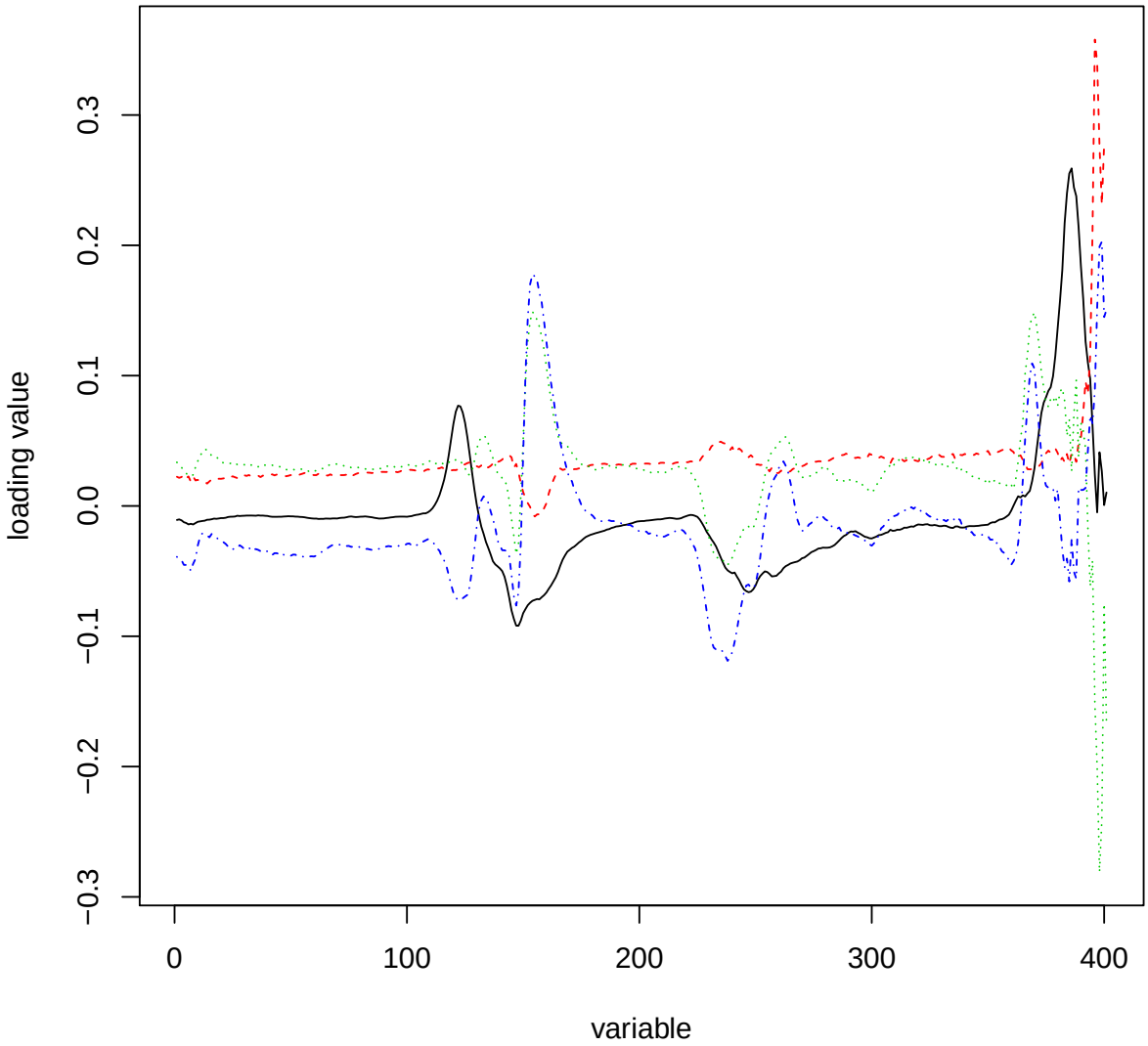

```
plot(best.model, plottype = "coefficients")
```



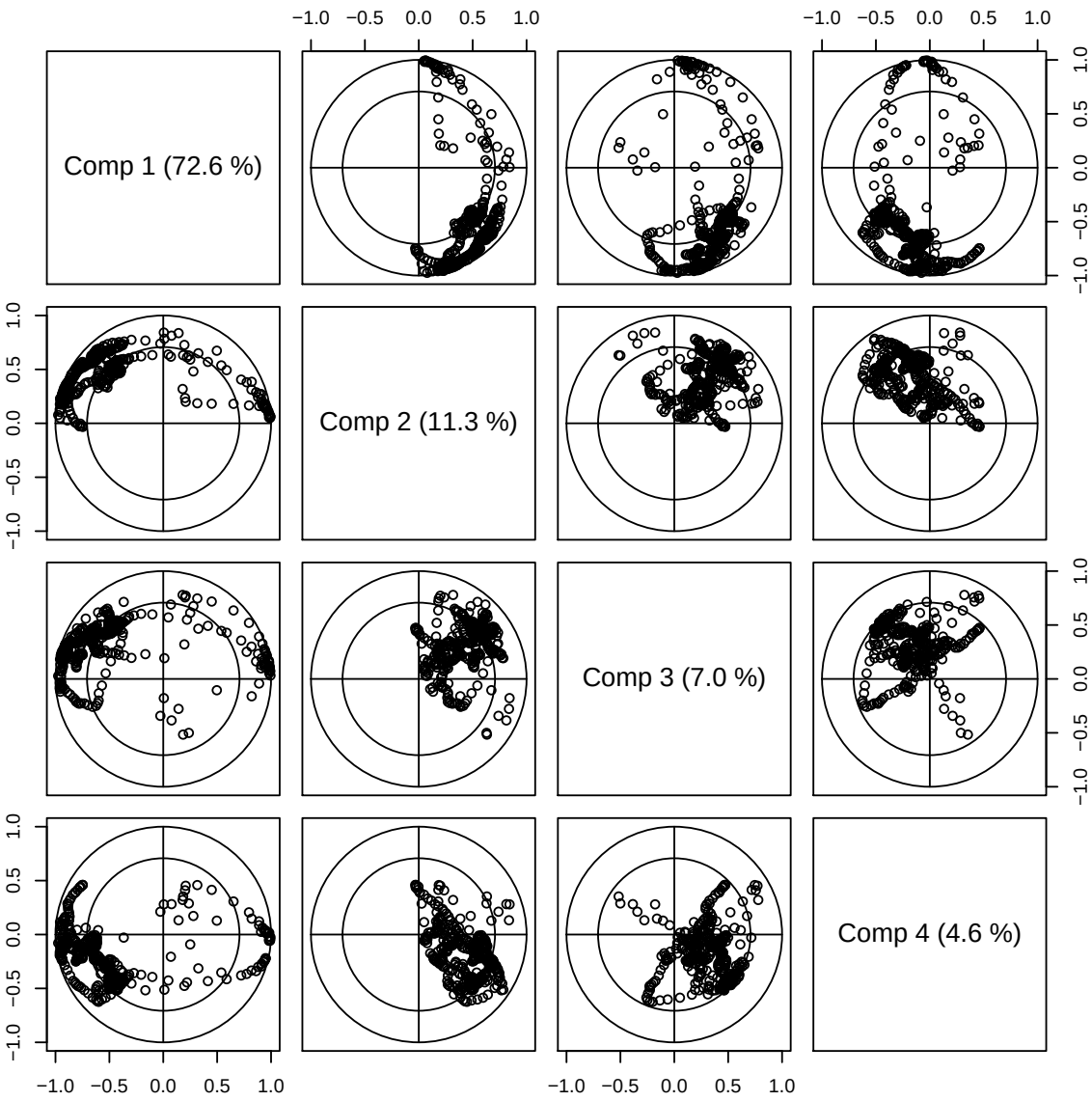

```
plot(best.model, plottype = "scores", comps = 1:4)
```




```r
plot(best.model, plottype = "loadings", comps = 1:4)
```



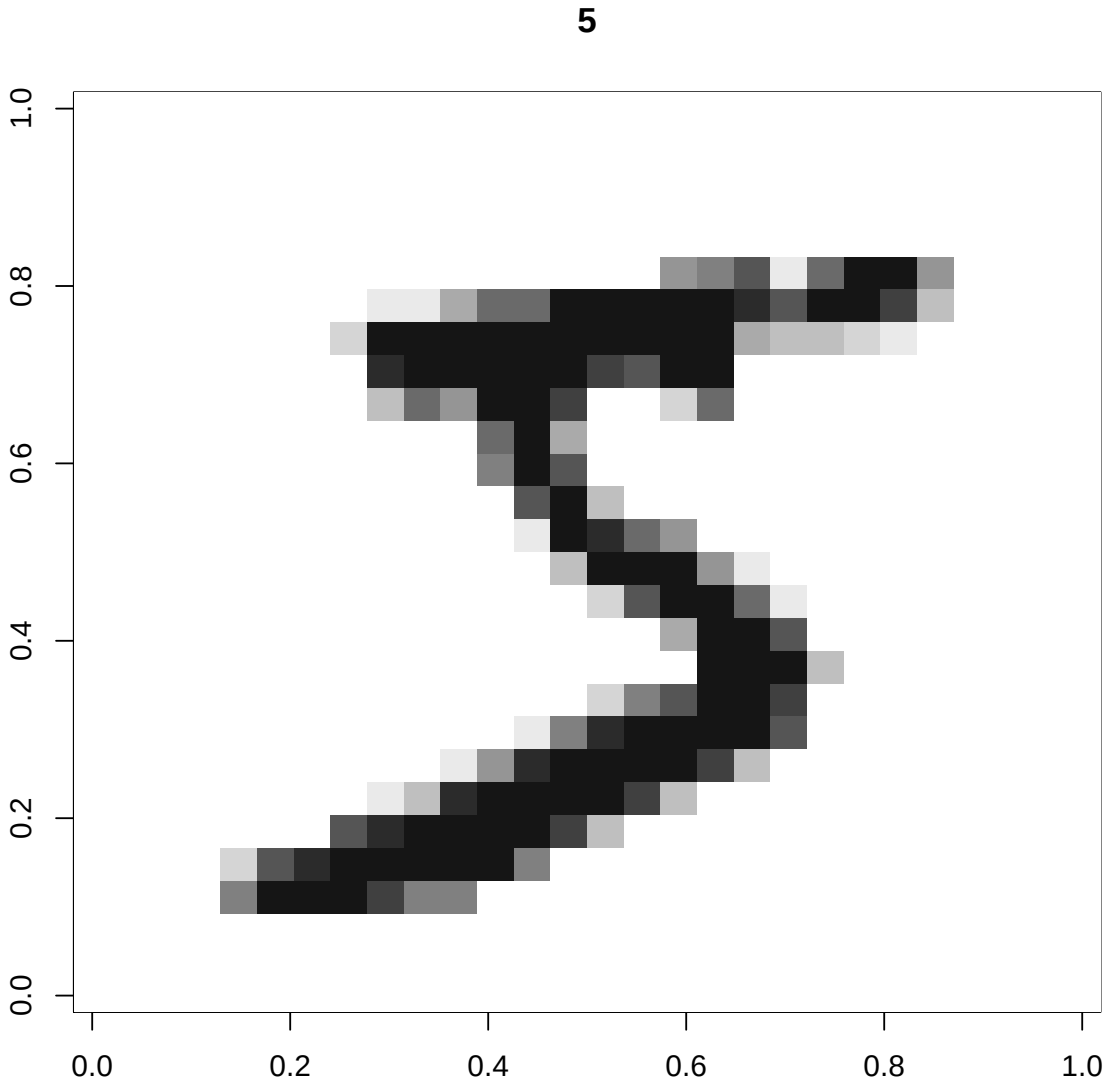

```
plot(best.model, plottype = "correlation",
    comps = 1:4)
```



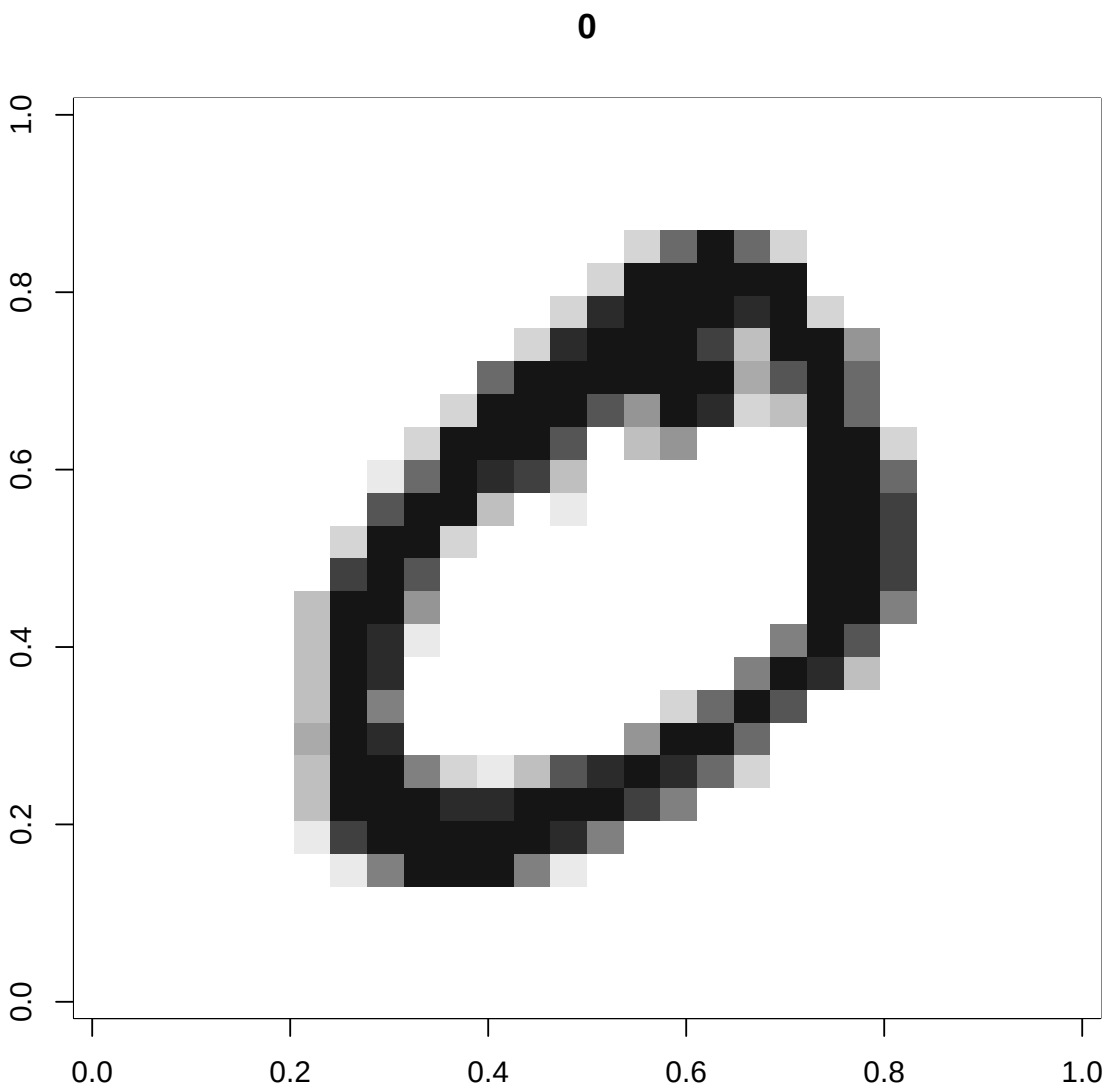
7.5 PCA-LDA/PCA-QDA for 'mnist'

```
rm(list = ls(all.names = TRUE))
library(e1071)
library(MASS)
source("PCA/PCAfncs.R")
load("mnist/mnist.rda")  # http://yann.lecun.com/exdb/mnist/
mnist.train$y <- factor(mnist.train$y)
pcalda <- function(...) pcawrap(lda, ...)
predict.pcalda <- function(...) predict(...)$class
show_digit <- function(arr784, col = gray(12:1/12),
    ...) {
    image(matrix(arr784, nrow = 28)[, 28:1],
        col = col, ...)
```

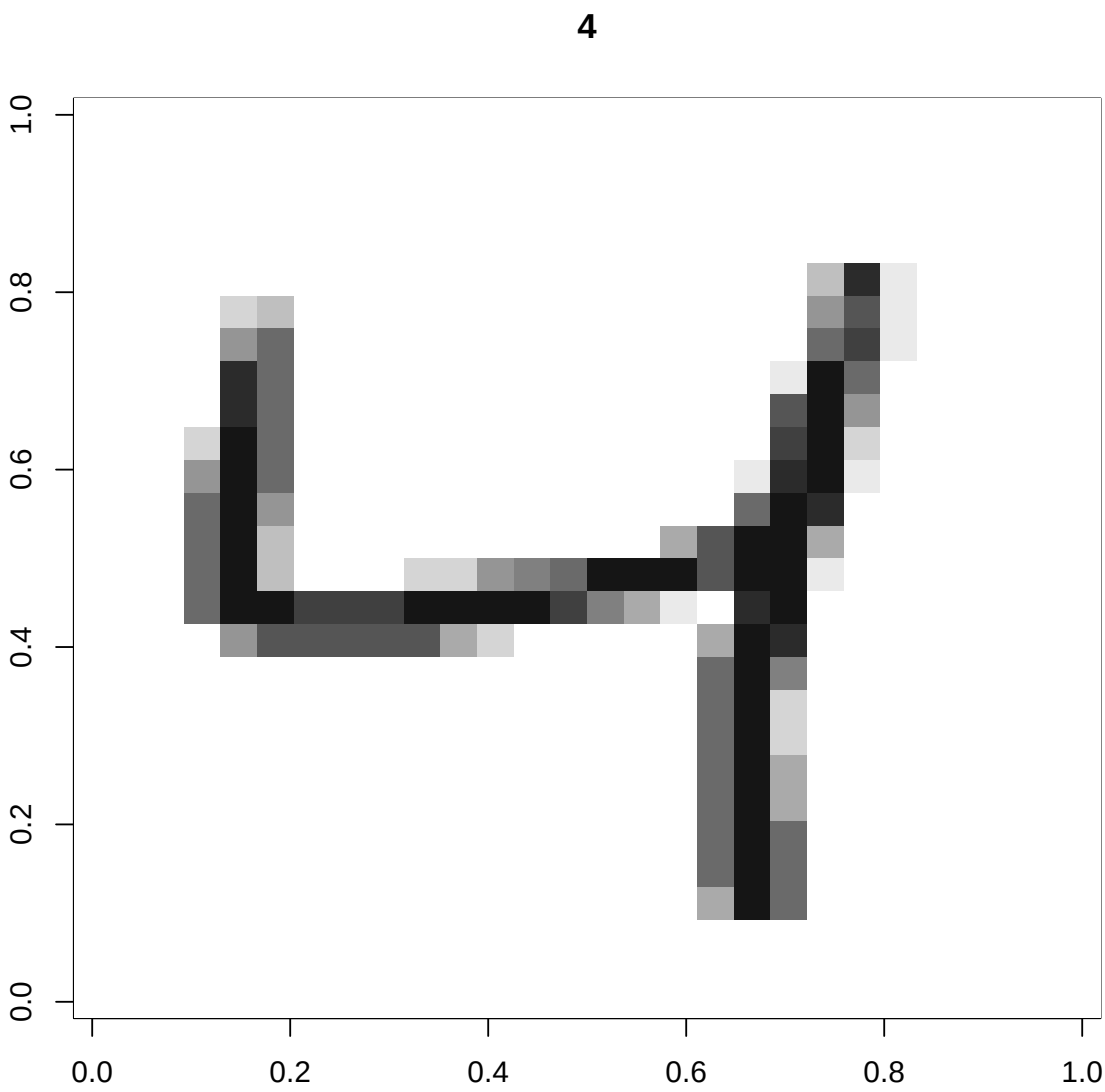
```
}
show_digit(as.matrix(mnist.train[1, -1]),
    main = mnist.train[1, 1])
```




```
show_digit(as.matrix(mnist.train[2, -1]),
    main = mnist.train[2, 1])
```




```
show_digit(as.matrix(mnist.train[3, -1]),
    main = mnist.train[3, 1])
```

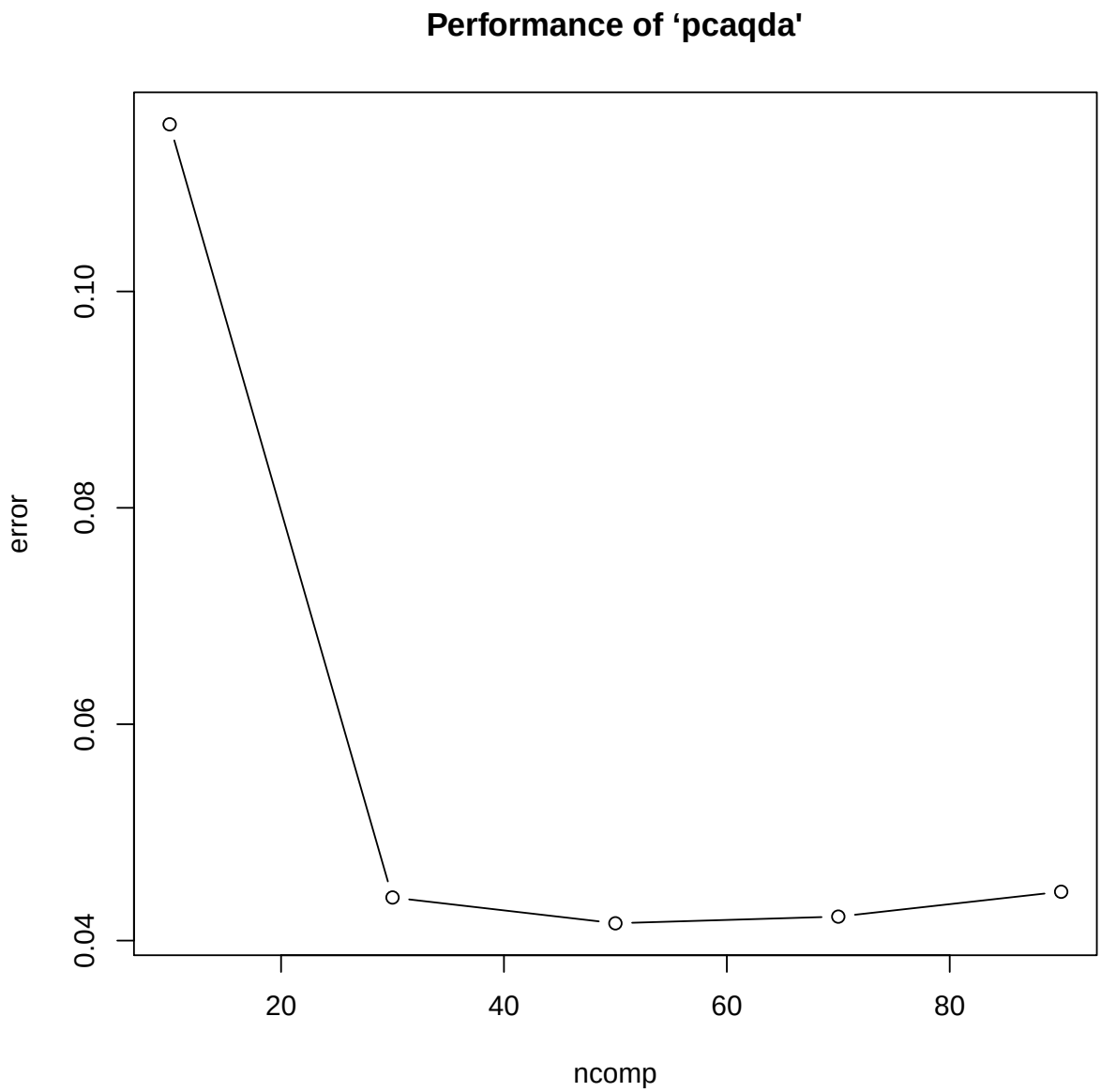



```
sds <- sapply(mnist.train, sd)
zero.sd <- names(mnist.train)[sds < 2]
mnist.train.nz <- mnist.train[, setdiff(names(mnist.train),
    zero.sd)]
tn.lda <- tune(lda, y ~ ., data = mnist.train.nz,
    predict.func = predict.pcalda, tunecontrol = tune.control(cross = 2))
summary(tn.lda)

##
## Error estimation of 'lda' using 2-fold cross validation: 0.1363833

table(actual = mnist.train$y, predicted = predict(tn.lda$best.model,
    mnist.train)$class)

##       predicted
```

```
## actual    0    1    2    3    4    5    6    7    8    9
##      0 5583    5   21   32   24  102   53    2   91   10
##      1    0 6460   37   21   10   38    6    8  151   11
##      2   57  199 4862  180  117   25  190   43  253   32
##      3   13   93  162 5189   25  225   21   91  173  139
##      4    5   56   31    2 5239   44   29    3   55  378
##      5   57   59   26  246   50 4466  112   29  239  137
##      6   59   55   61    4   80  142 5434    0   79    4
##      7   35  142   39   45  178   13    1 5238   29  545
##      8   32  334   45  196   75  267   29   13 4697  163
##      9   33   27   17   94  320   25    0  272   56 5105

tt <- table(actual = mnist.test$y, predicted = predict(tn.lda$best.model,
    mnist.test)$class)
print(tt)

##       predicted
## actual    0    1    2    3    4    5    6    7    8    9
##      0  940    0    1    4    2   13    9    1    9    1
##      1    0 1096    4    3    2    2    3    0   25    0
##      2   15   31  818   33   22    5   37    7   57    7
##      3    5    5   25  883    4   25    3   16   29   15
##      4    0   12    6    0  888    4    7    2   10   53
##      5    8    8    4   44   12  737   15    9   37   18
##      6   12    8   11    0   25   29  857    0   16    0
##      7    2   30   14   10   21    2    0  864    4   81
##      8    7   27    8   27   20   51   10    6  793   25
##      9    9    7    1   13   63    6    0   36   12  862

1 - sum(diag(tt))/sum(tt)

## [1] 0.1262
```

```
pcaqda <- function(...) pcawrap(qda, ...)
predict.pcaqda <- function(...) predict(...)$class


tn.qda <- tune(qda, y ~ ., data = mnist.train.nz,
               predict.func = predict.pcaqda,
               tunecontrol = tune.control(cross = 2))

## Error in qda.default(x, grouping, ...): rank deficiency in group 0

summary(tn.qda)

## Error in summary(tn.qda): object 'tn.qda' not found

tn.pcaqda <- tune(pcaqda, y ~ ., data = mnist.train,
                  scale = FALSE, center = TRUE,
```

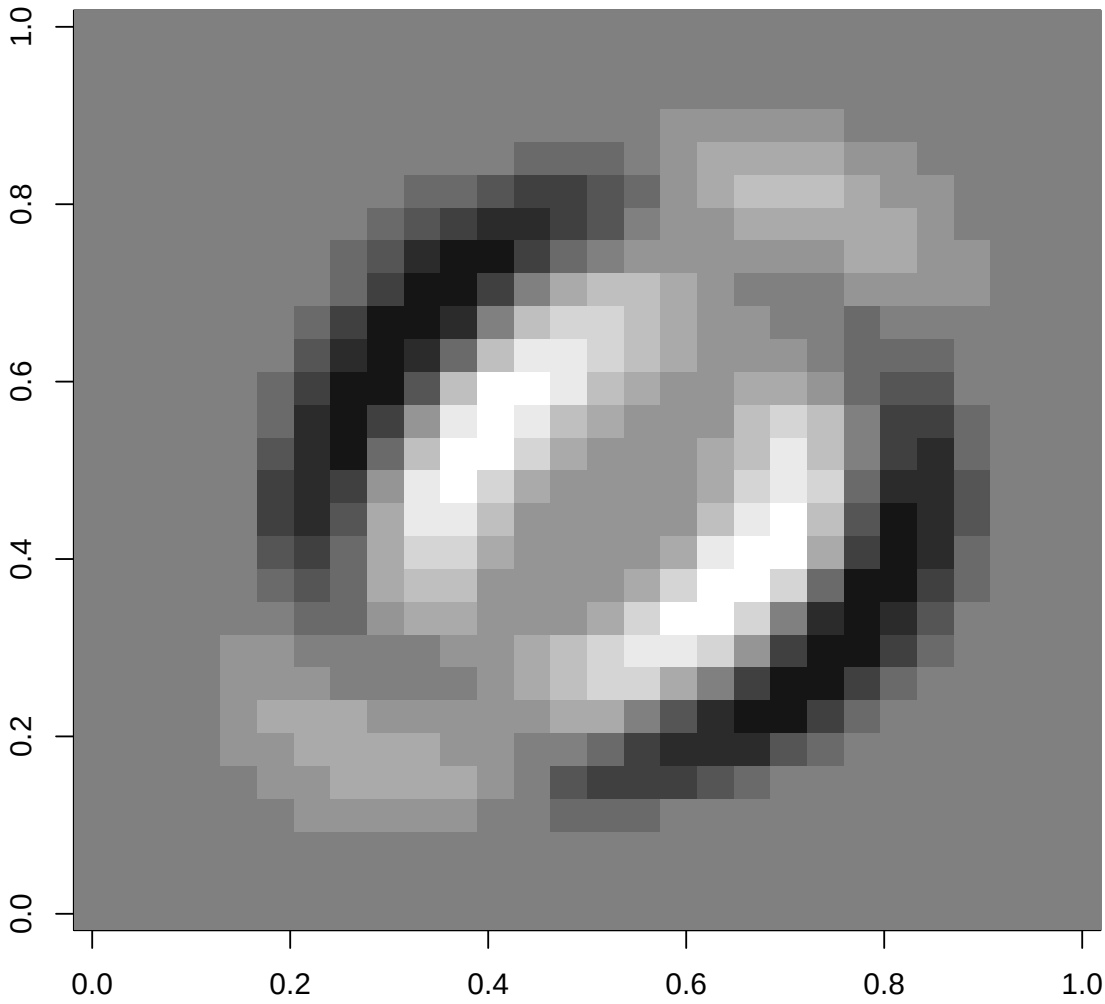
```r
                  # ranges = list(ncomp = c(1, 10, 20, 40, 50)),
                  ranges = list(ncomp = c(10, 30, 50, 70, 90)),
                  predict.func = predict.pcaqda,
                  tunecontrol = tune.control(cross = 2))
summary(tn.pcaqda)

##
## Parameter tuning of 'pcaqda':
##
## - sampling method: 2-fold cross validation
##
## - best parameters:
##  ncomp
##     50
##
## - best performance: 0.0416
##
## - Detailed performance results:
##   ncomp      error   dispersion
## 1    10 0.11545000 0.0012492220
## 2    30 0.04398333 0.0002592725
## 3    50 0.04160000 0.0008956686
## 4    70 0.04221667 0.0012963624
## 5    90 0.04451667 0.0007306770

plot(tn.pcaqda)
```



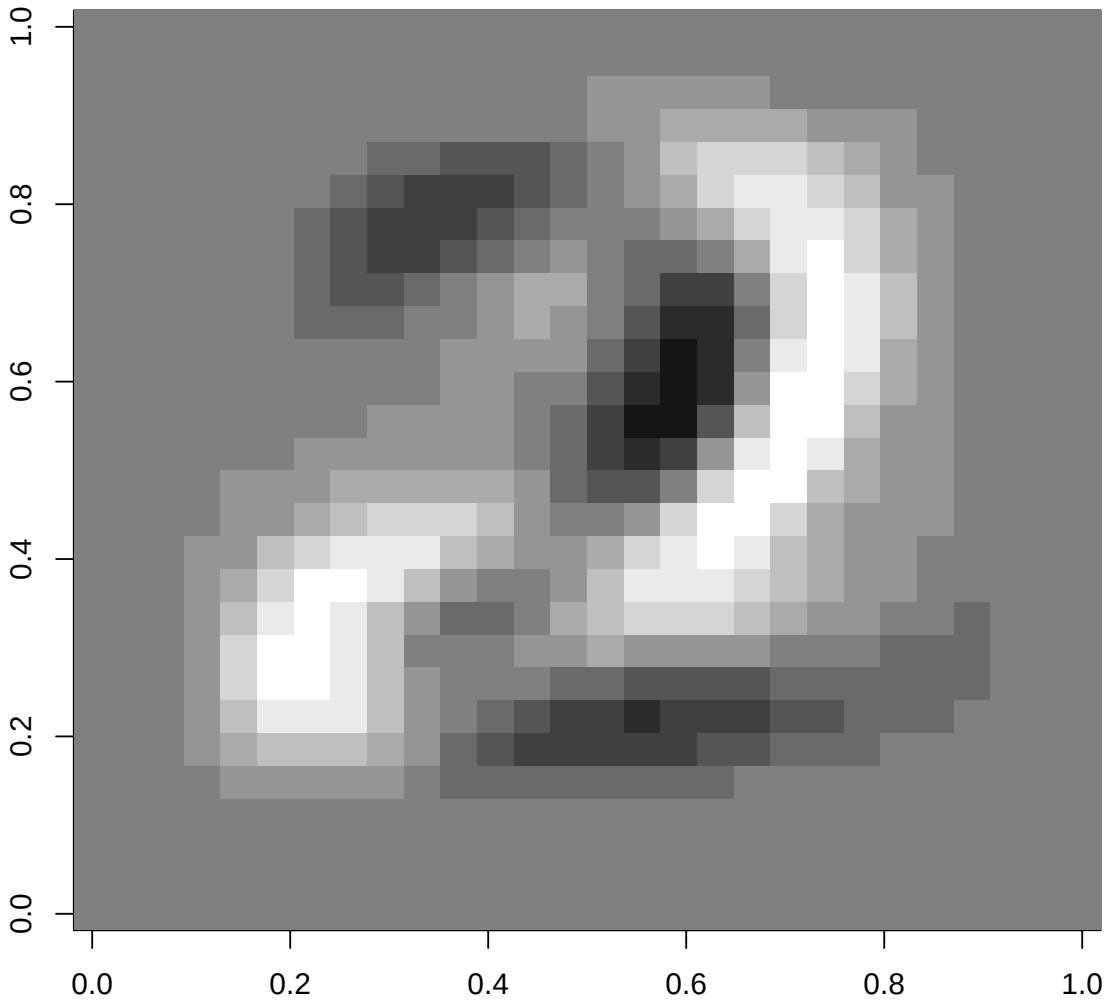

```
table(actual = mnist.train$y,
      predicted = predict(tn.pcaqda$best.model)$class)

##       predicted
## actual    0    1    2    3    4    5    6    7    8    9
##      0 5832    0   24    5    1   13    2    0   44    2
##      1    0 6437  106   14   31    0    2   13  136    3
##      2   13    1 5813   25   14    1    7   13   65    6
##      3    2    1   92 5824    4   51    0   19  116   22
##      4    6    1   20    3 5702    2   12   15   30   51
##      5   15    0    3   76    2 5218   17    0   75   15
##      6   29    3    7    2    6  108 5706    0   57    0
##      7   11    8   90   15   36   15    0 5935   56   99
##      8   12   24   40   71    8   40    2    4 5623   27
##      9   17    1   25   60   60   14    1   79   95 5597
```

```r
tt <- table(actual = mnist.test$y,
            predicted = predict(tn.pcaqda$best.model, mnist.test)$class)

print(tt)

##       predicted
## actual    0    1    2    3    4    5    6    7    8    9
##      0  970    0    1    0    0    2    1    1    5    0
##      1    0 1098   13    1    2    1    1    0   19    0
##      2    2    0 1000    3    3    0    2    2   20    0
##      3    2    0    9  970    0    5    0    2   18    4
##      4    1    0    4    0  964    0    3    2    2    6
##      5    2    0    1   18    0  860    2    0    9    0
##      6    8    1    2    0    4   12  924    0    7    0
##      7    1    2   28    1    3    2    0  959   14   18
##      8    4    0    8   10    1    5    1    2  938    5
##      9    5    0   10    6   11    2    0    6   16  953

1 - sum(diag(tt)) / sum(tt)

## [1] 0.0364
```

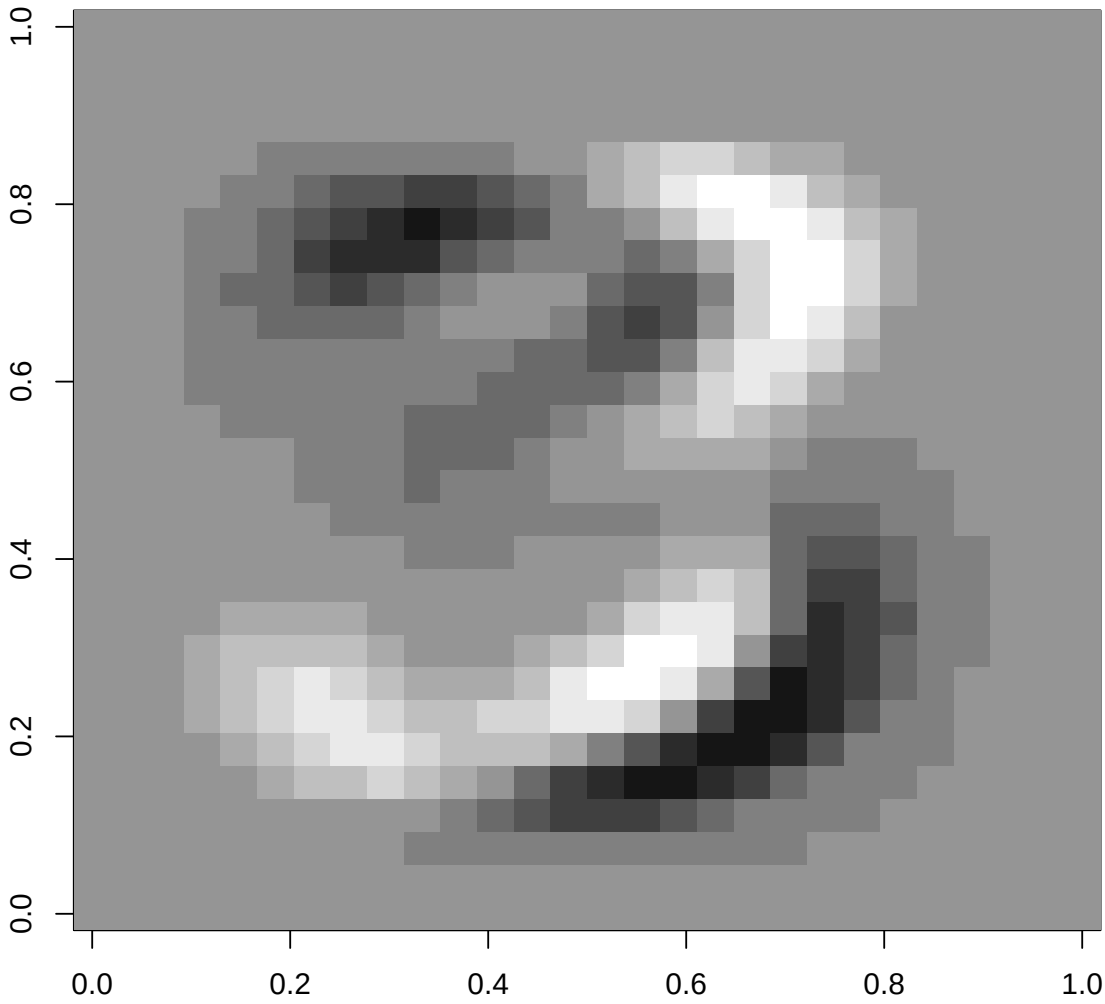
```r
prs <- by(mnist.train, mnist.train$y, function(df) {
    pr <- prcomp(~. - y, data = df, scale = FALSE,
        center = TRUE, ncomp = 3)
})
show_digit(prs[["0"]]$rotation[, 1])
```



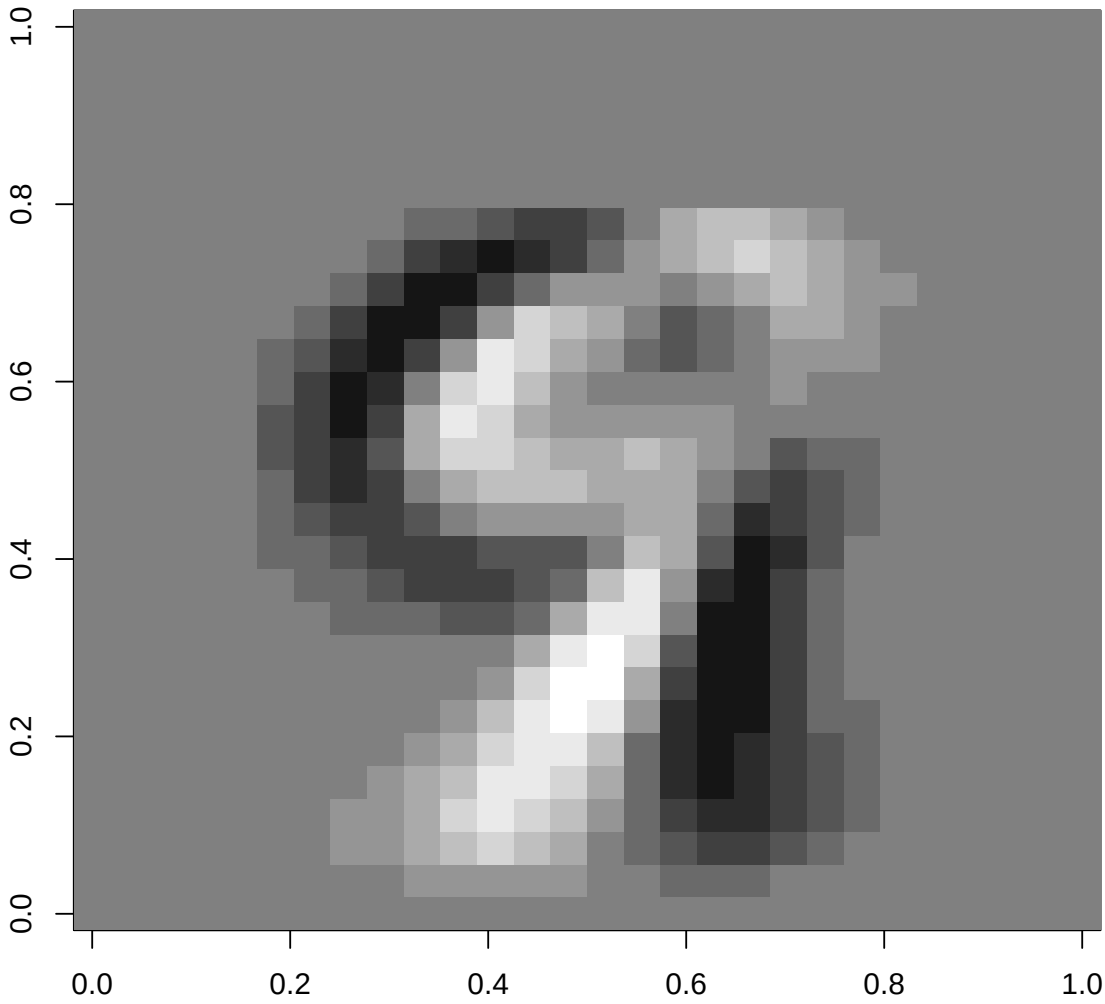

```
show_digit(prs[["2"]]$rotation[, 1])
```




```
show_digit(prs[["3"]]$rotation[, 1])
```




```
show_digit(prs[["9"]]$rotation[, 1])
```



8 Материалы с занятия 31 октября

```
read_chunk("clust/clust.R")
read_chunk("clust/usa.R")
```

8.1 k-means

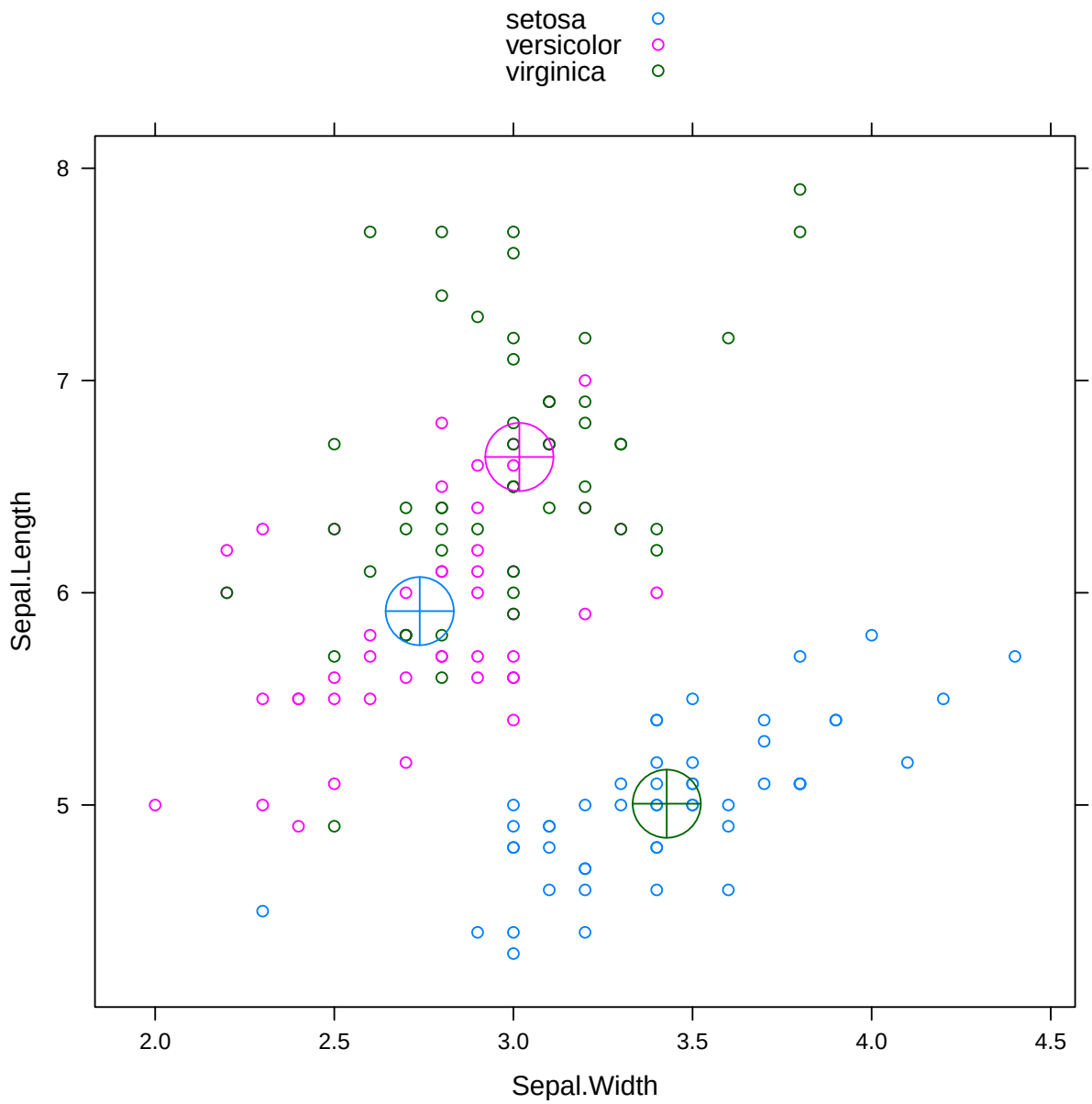
```
rm(list = ls(all.names = TRUE))
library(MASS)
library(cclust)  # cclust(), cclust.predict()
library(rattle)  # wine data
```

```r
## Rattle: A free graphical interface for data mining with R.
## Version 3.3.0 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

library(lattice)
library(latticeExtra)
# Hello, I'm iris dataset. Are you miss
# me?
kc <- kmeans(subset(iris, select = -Species),
    3, nstart = 10)
print(kc)

## K-means clustering with 3 clusters of sizes 62, 38, 50
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.901613    2.748387     4.393548    1.433871
## 2     6.850000    3.073684     5.742105    2.071053
## 3     5.006000    3.428000     1.462000    0.246000
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
##  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
##  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
##  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   3   3   3   3   3   1   1   2   1   1   1   1   1   1   1
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##   1   1   2   1   1   1   1   1   1   1   1   1   1   1   1
##  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105
##   1   1   1   1   1   1   1   1   1   1   2   1   2   2   2
## 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   2   1   2   2   2   2   2   2   1   1   2   2   2   2   1
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
##   2   1   2   1   2   2   1   1   2   2   2   2   2   1   2
## 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150
##   2   2   2   1   2   2   2   1   2   2   2   1   2   2   1
##
## Within cluster sum of squares by cluster:
## [1] 39.82097 23.87947 15.15100
##  (between_SS / total_SS =  88.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"
```

```
## [4] "withinss"     "tot.withinss" "betweenss"
## [7] "size"         "iter"         "ifault"

table(clust = kc$cluster, actual = iris$Species)

##      actual
## clust setosa versicolor virginica
##     1      0         48        14
##     2      0          2        36
##     3     50          0         0

# Mmm... Something wrong. Let's normalize
# predictors
iris.n <- scale(subset(iris, select = -Species),
    scale = TRUE, center = FALSE)
kc.n <- kmeans(iris.n, 3, nstart = 10)
table(clust = kc.n$cluster, actual = iris$Species)

##      actual
## clust setosa versicolor virginica
##     1      0          2        46
##     2     50          0         0
##     3      0         48         4

# Looks better
my.kmeans <- function(formula, centers, nstart = 10,
    data = NULL, subset = NULL, scale. = TRUE,
    ...) {
    cl <- match.call()
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data", "subset"),
        names(mf), 0L)
    mf <- mf[c(1L, m)]
    mf$drop.unused.levels <- TRUE
    mf[[1L]] <- quote(stats::model.frame)
    mf <- eval(mf, parent.frame())
    mt <- attr(mf, "terms")
    mm <- model.matrix(mt, mf)[, -1, drop = FALSE]
    mm <- scale(mm, scale = scale., center = FALSE)
    if (length(centers) > 1) {
        centers <- scale(centers, center = FALSE,
            scale = FALSE)
        attr(centers, "scaled:scale") <- attr(centers,
            "scaled:center") <- NULL
    }
    scale. <- attr(mm, "scaled:scale")
    if (is.null(scale.))
        scale. <- rep(1, ncol(mm))
    ccs <- lapply(seq_len(nstart), function(i.,
```

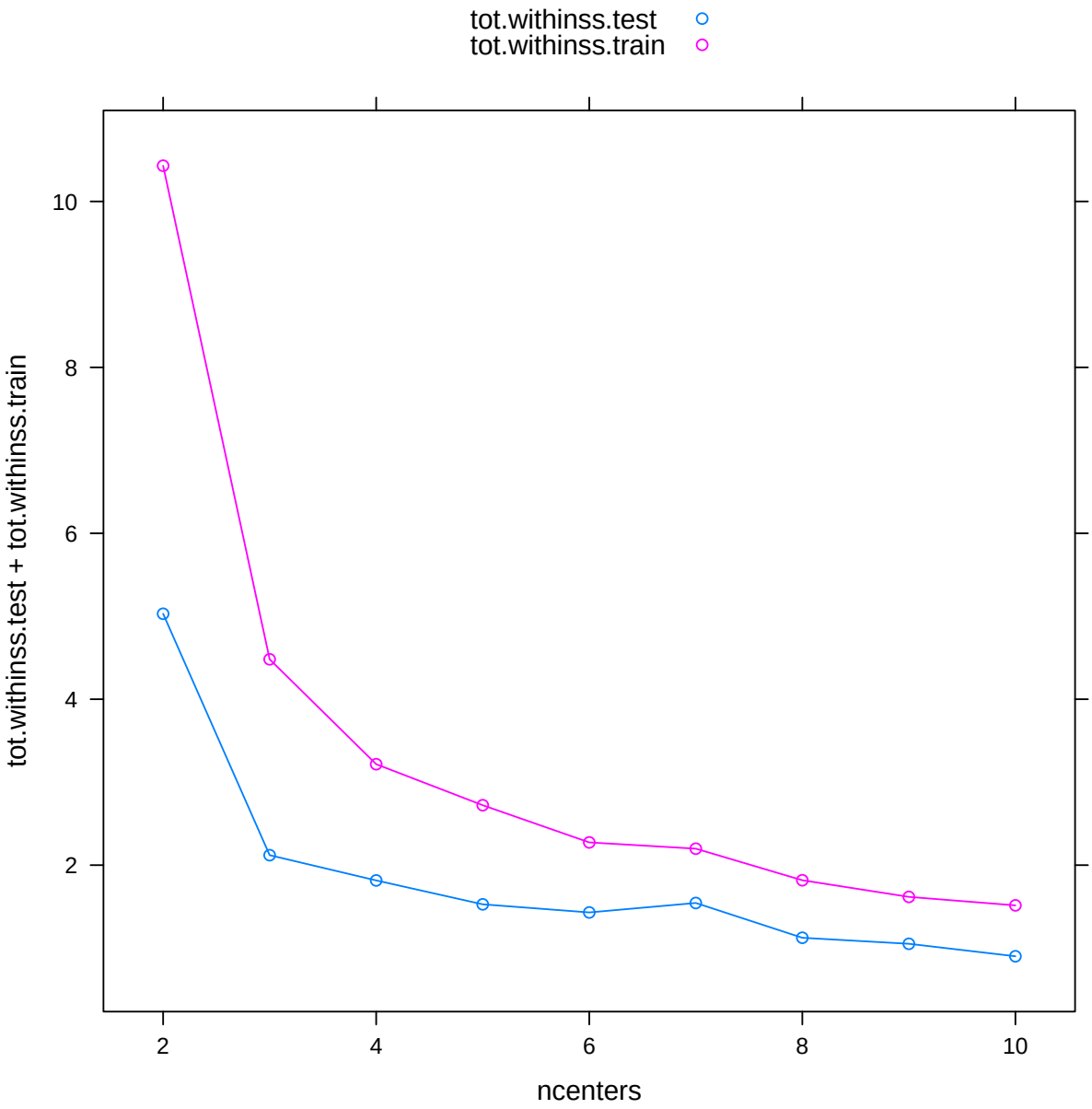
```
        ...) cclust(mm, centers = centers,
        ...), ...)
    withinsss <- sapply(ccs, function(cc) sum(cc$withinss))
    cc <- ccs[[which.min(withinsss)]]
    cc.centers <- scale(cc$centers, scale = 1/scale.,
        center = FALSE)
    attr(cc.centers, "scaled:scale") <- attr(centers,
        "scaled:center") <- NULL
    res <- list(model = cc, centers = cc.centers,
        scale = scale., formula = formula,
        data = data, terms = mt)
    class(res) <- "my.kmeans"
    res
}
predict.my.kmeans <- function(object, newdata = object$data,
    type = c("cluster", "distances", "model"),
    ...) {
    mt <- terms(object)
    mf <- model.frame(mt, data = newdata)
    mm <- model.matrix(mt, mf)[, -1, drop = FALSE]
    mm <- scale(mm, scale = object$scale,
        center = FALSE)
    pred <- predict(object$model, mm)
    type <- match.arg(type)
    if (identical(type, "cluster")) {
        pred$cluster
    } else if (identical(type, "model")) {
        withinss <- function(clobj, x) {
            retval <- rep(0, nrow(clobj$centers))
            x <- (x - clobj$centers[clobj$cluster,
                ])^2
            for (k in 1:nrow(clobj$centers)) {
                retval[k] <- sum(x[clobj$cluster ==
                  k, ])
            }
            retval
        }
        pred$withinss <- withinss(pred, mm)
        pred
    } else if (identical(type, "distances")) {
        dists <- function(clobj, x) {
            sapply(1:nrow(clobj$centers),
                function(k) {
                  rowSums(x - rep(clobj$centers[k,
                    ], each = nrow(x)))^2
                })
        }
        dists(pred, mm)
```

```
    }
}
kc <- my.kmeans(~., data = subset(iris, select = -Species),
    scale = TRUE, centers = 3)
table(my.kmeans = predict(kc), actual = iris$Species)

##          actual
## my.kmeans setosa versicolor virginica
##         1      0         48         4
##         2      0          2        46
##         3     50          0         0

xyplot(Sepal.Length ~ Sepal.Width, data = iris,
    groups = Species, auto.key = TRUE) +
    xyplot(Sepal.Length ~ Sepal.Width, data = as.data.frame(kc$centers),
        groups = 4:6, pch = 10, cex = 5)
```

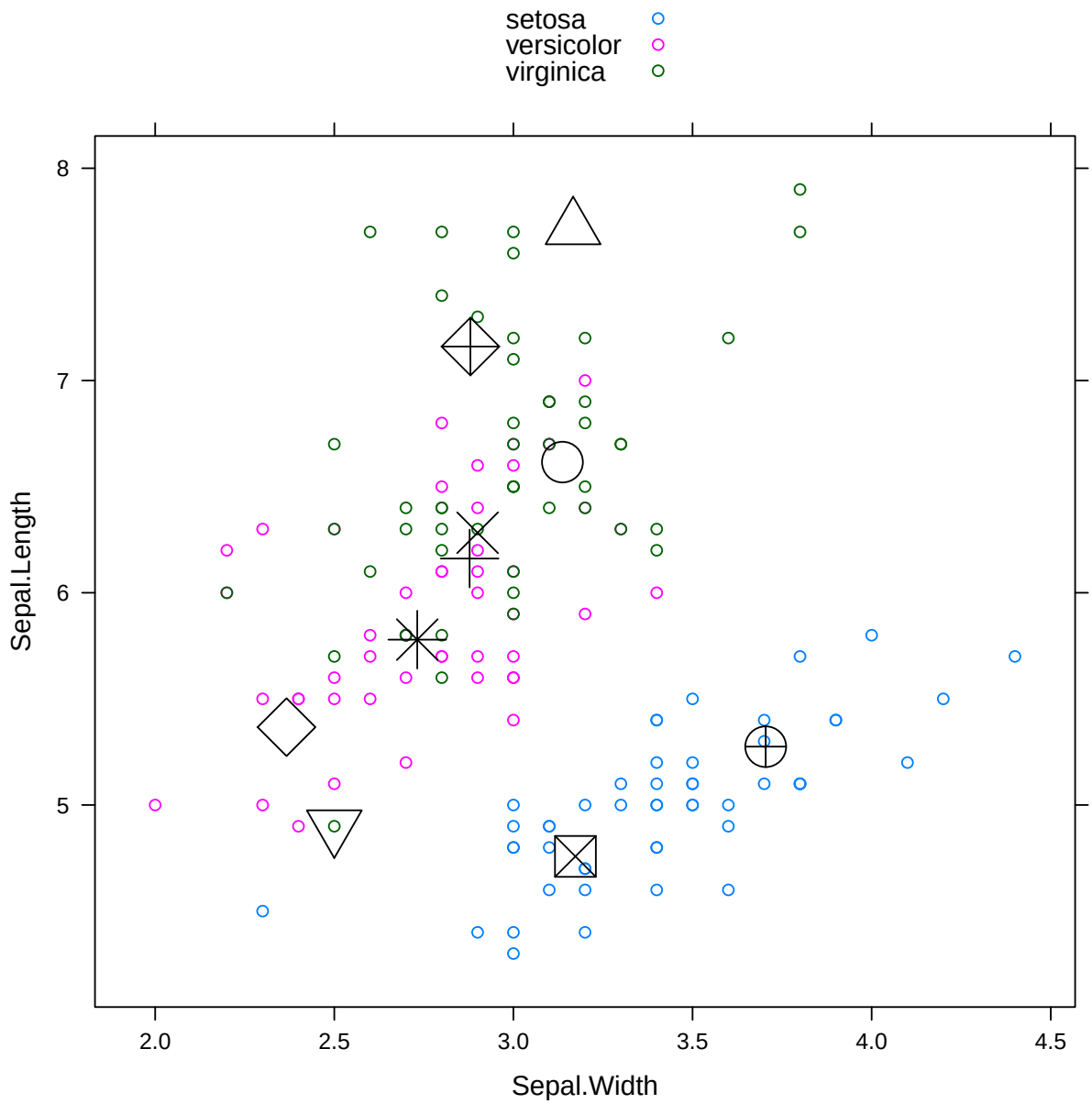



```
# test-train approach
train.idx <- sample(nrow(iris), size = 0.66 *
    nrow(iris))
iris.test <- iris[-train.idx, ]
iris.train <- iris[train.idx, ]
kc <- my.kmeans(~. - Species, data = iris,
    subset = train.idx, centers = 3)
kc <- my.kmeans(~. - Species, data = iris.train,
    centers = 3)
# Train
table(clust = predict(kc, iris.train), actual = iris.train$Species)

##      actual
## clust setosa versicolor virginica
##     1     30          0         0
```

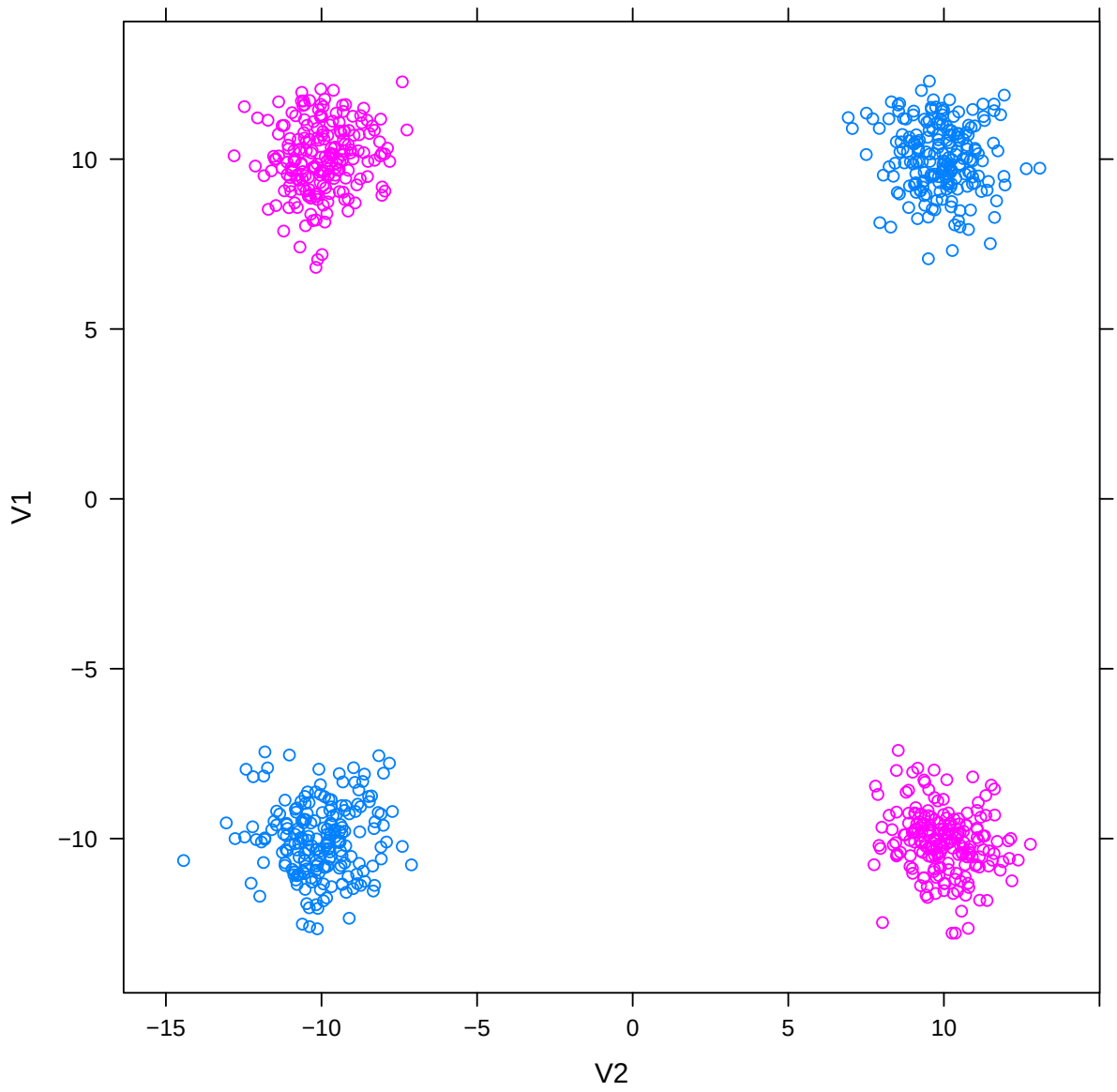
```
##     2      0         30         3
##     3      0          2        34

# Test
table(clust = predict(kc, iris.test), actual = iris[-train.idx,
    ]$Species)

##      actual
## clust setosa versicolor virginica
##     1     20          0         0
##     2      0         18         1
##     3      0          0        12

# Determine optimal number of clusters
kcs <- lapply(2:10, function(n) my.kmeans(~. -
    Species, data = iris, centers = n, subset = train.idx))
tot.withinss.train <- sapply(kcs, function(kc) sum(kc$model$withinss))
tot.withinss.test <- sapply(kcs, function(kc) sum(predict(kc,
    iris.test, type = "model")$withinss))
ncenters <- sapply(kcs, function(kc) kc$model$ncenters)
xyplot(tot.withinss.test + tot.withinss.train ~
    ncenters, type = "b", auto.key = TRUE)
```




```
kc <- my.kmeans(~. - Species, data = iris,
    centers = 10)
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
    groups = Species, auto.key = TRUE) +
    xyplot(Sepal.Length ~ Sepal.Width, data = as.data.frame(kc$centers),
        pch = 1:10, cex = 3, col = "black")
```

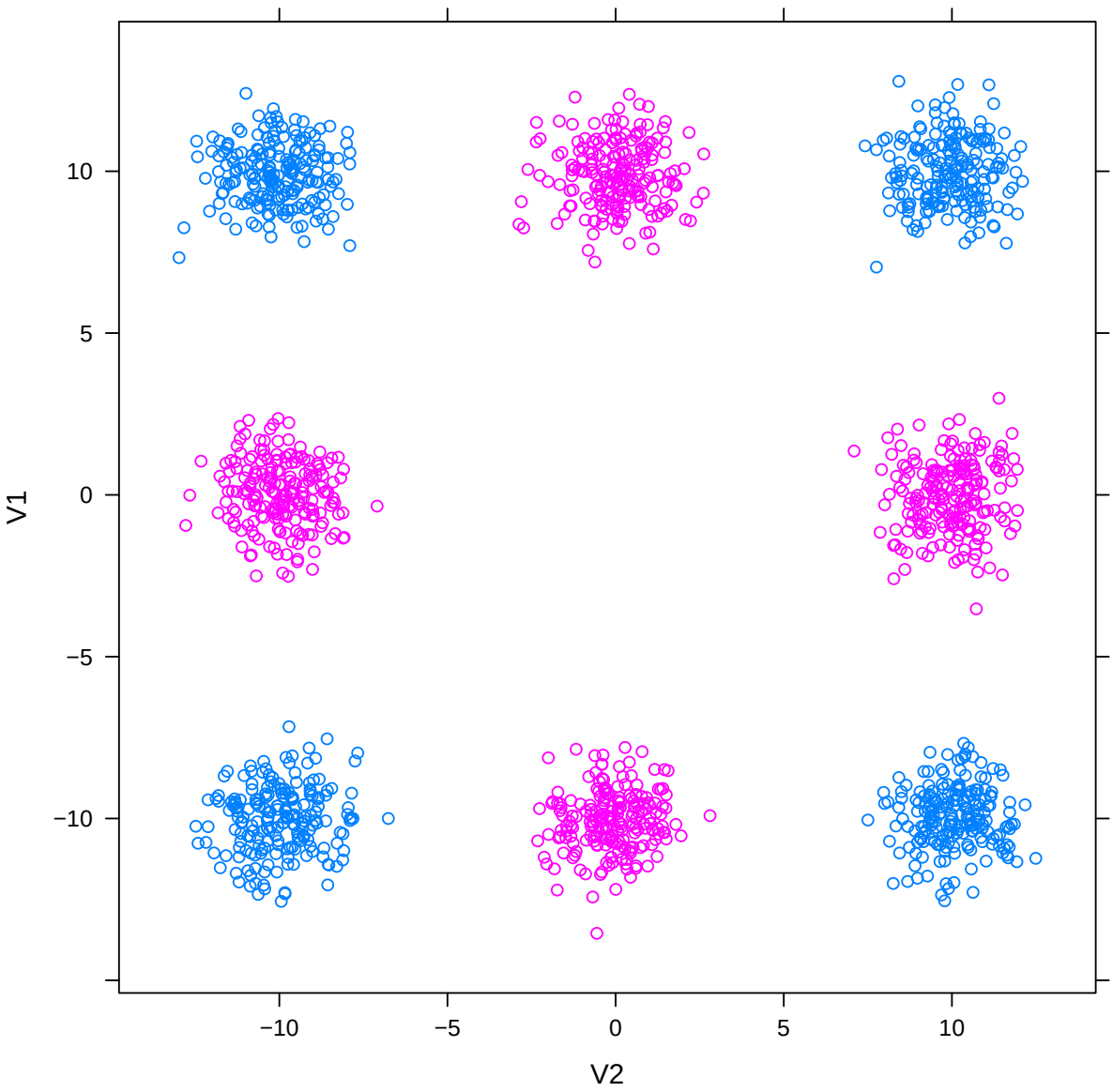



```r
## Clustering as classification
library(mvtnorm)
N <- 200
df <- rbind(rmvnorm(N, c(10, 10)), rmvnorm(N,
    c(-10, -10)), rmvnorm(N, c(10, -10)),
    rmvnorm(N, c(-10, 10)))
df <- as.data.frame(df)
df$class <- factor(rep(c("A", "B"), each = 2 *
    N))
xyplot(V1 ~ V2, groups = class, data = df)
```

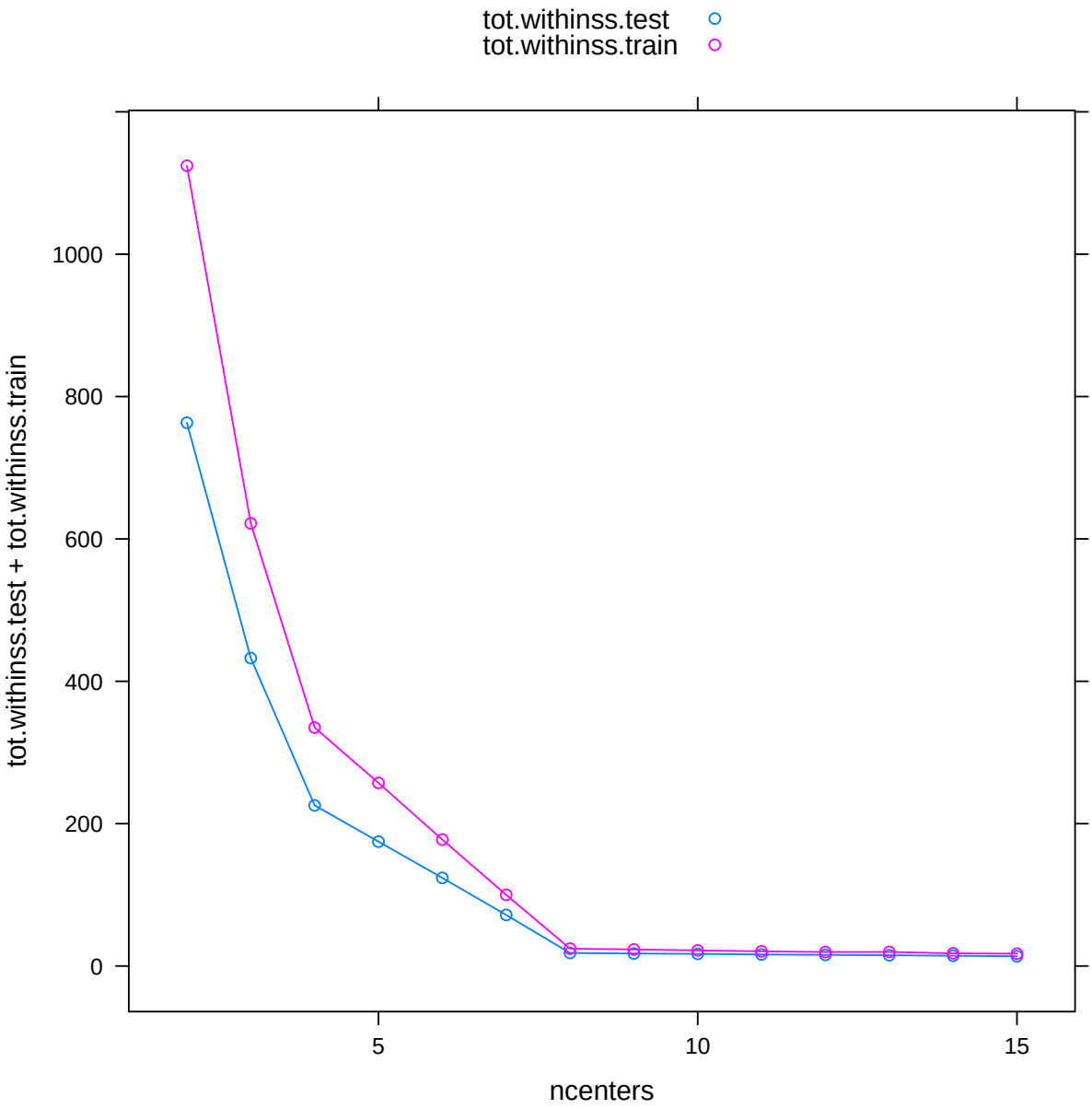



```
qd <- qda(class ~ ., data = df)
table(predicted = predict(qd, df)$class,
    actual = df$class)

##           actual
## predicted   A   B
##         A 400   0
##         B   0 400

ld <- lda(class ~ ., data = df)
table(predicted = predict(ld, df)$class,
    actual = df$class)

##           actual
## predicted   A   B
```

```
##          A 200 200
##          B 200 200

N <- 200
df <- rbind(rmvnorm(N, c(10, 10)), rmvnorm(N,
    c(10, -10)), rmvnorm(N, c(-10, -10)),
    rmvnorm(N, c(-10, 10)), rmvnorm(N, c(0,
        -10)), rmvnorm(N, c(0, 10)), rmvnorm(N,
        c(10, 0)), rmvnorm(N, c(-10, 0)))
df <- as.data.frame(df)
df$class <- factor(rep(c("A", "B"), each = 4 *
    N))
xyplot(V1 ~ V2, groups = class, data = df)
```



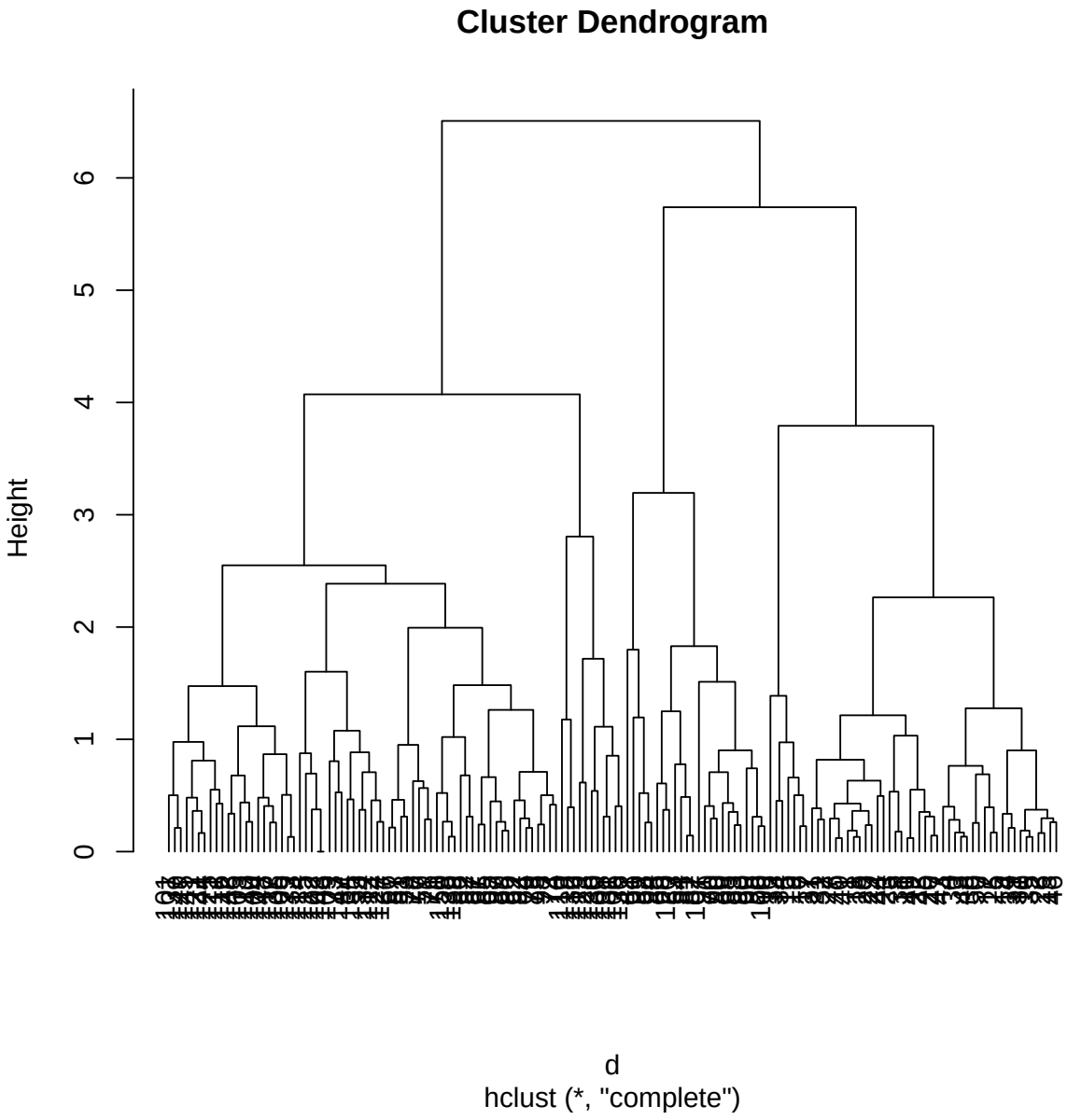
```r
qd <- qda(class ~ ., data = df)
table(predicted = predict(qd, df)$class,
    actual = df$class)

##          actual
## predicted   A   B
##         A 795  19
##         B   5 781

ld <- lda(class ~ ., data = df)
table(predicted = predict(ld, df)$class,
    actual = df$class)

##          actual
## predicted   A   B
##         A 400 403
##         B 400 397

train.idx <- sample(nrow(df), size = 0.6 *
    nrow(df))
# Determine optimal number of clusters
kcs <- lapply(2:15, function(n) my.kmeans(~. -
    class, data = df, centers = n, subset = train.idx))
tot.withinss.train <- sapply(kcs, function(kc) sum(kc$model$withinss))
tot.withinss.test <- sapply(kcs, function(kc) sum(predict(kc,
    df[-train.idx, ], type = "model")$withinss))
ncenters <- sapply(kcs, function(kc) kc$model$ncenters)
xyplot(tot.withinss.test + tot.withinss.train ~
    ncenters, type = "b", auto.key = TRUE)
```



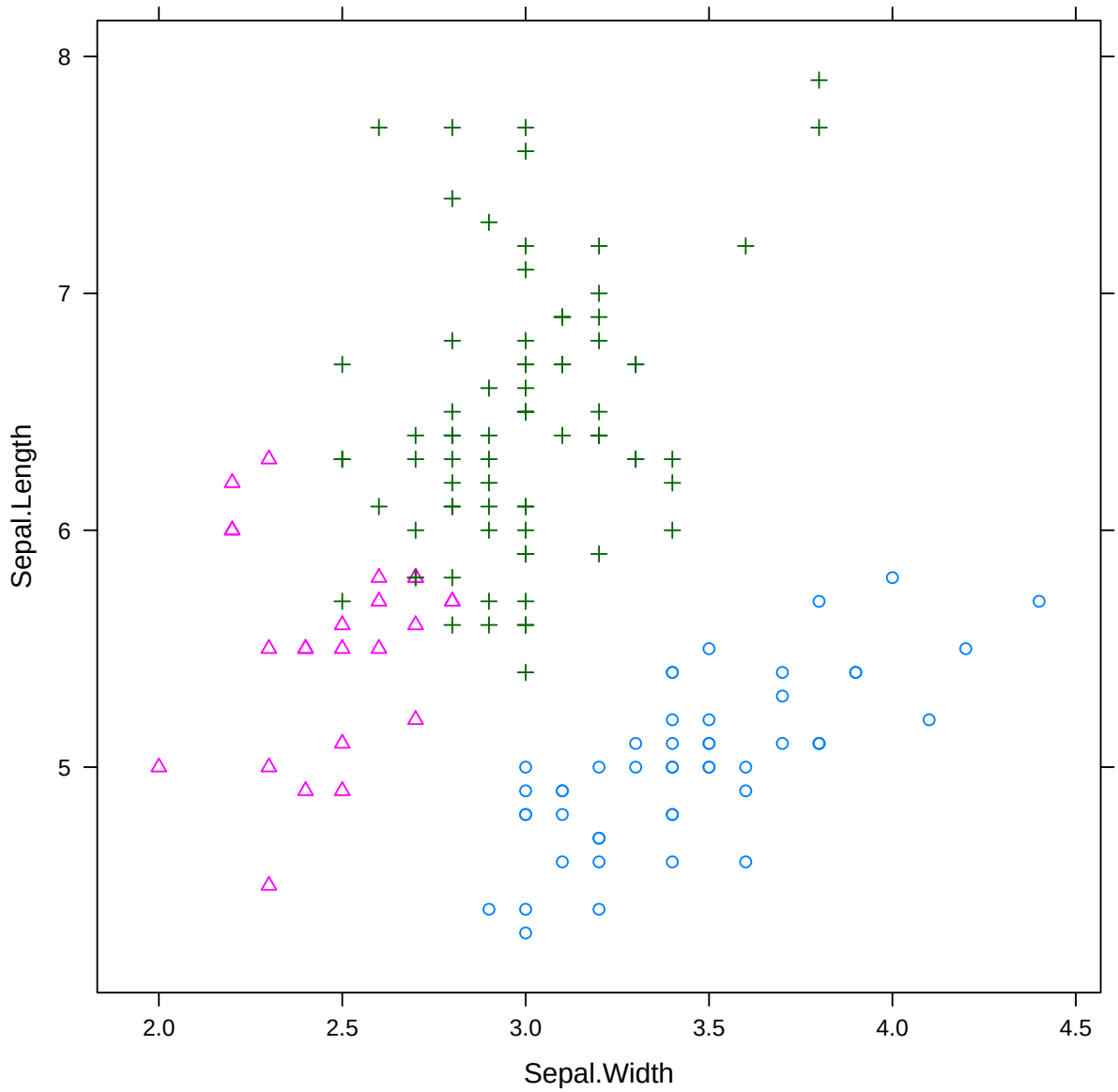

```
kc <- my.kmeans(~. - class, data = df, centers = 8,
    nstart = 25)
pred <- predict(kc)
tb <- table(cluser = pred, actual = df$class)
tb

##       actual
## cluser   A   B
##      1   0 200
##      2   0 200
##      3   0 200
##      4 200   0
##      5   0 200
##      6 200   0
##      7 200   0
```

```
##       8 200   0

tb <- tb/rowSums(tb)
## pred <- predict(kc, newdata)
pred <- colnames(tb)[max.col(tb[pred, ])]
head(pred)

## [1] "A" "A" "A" "A" "A" "A"
```

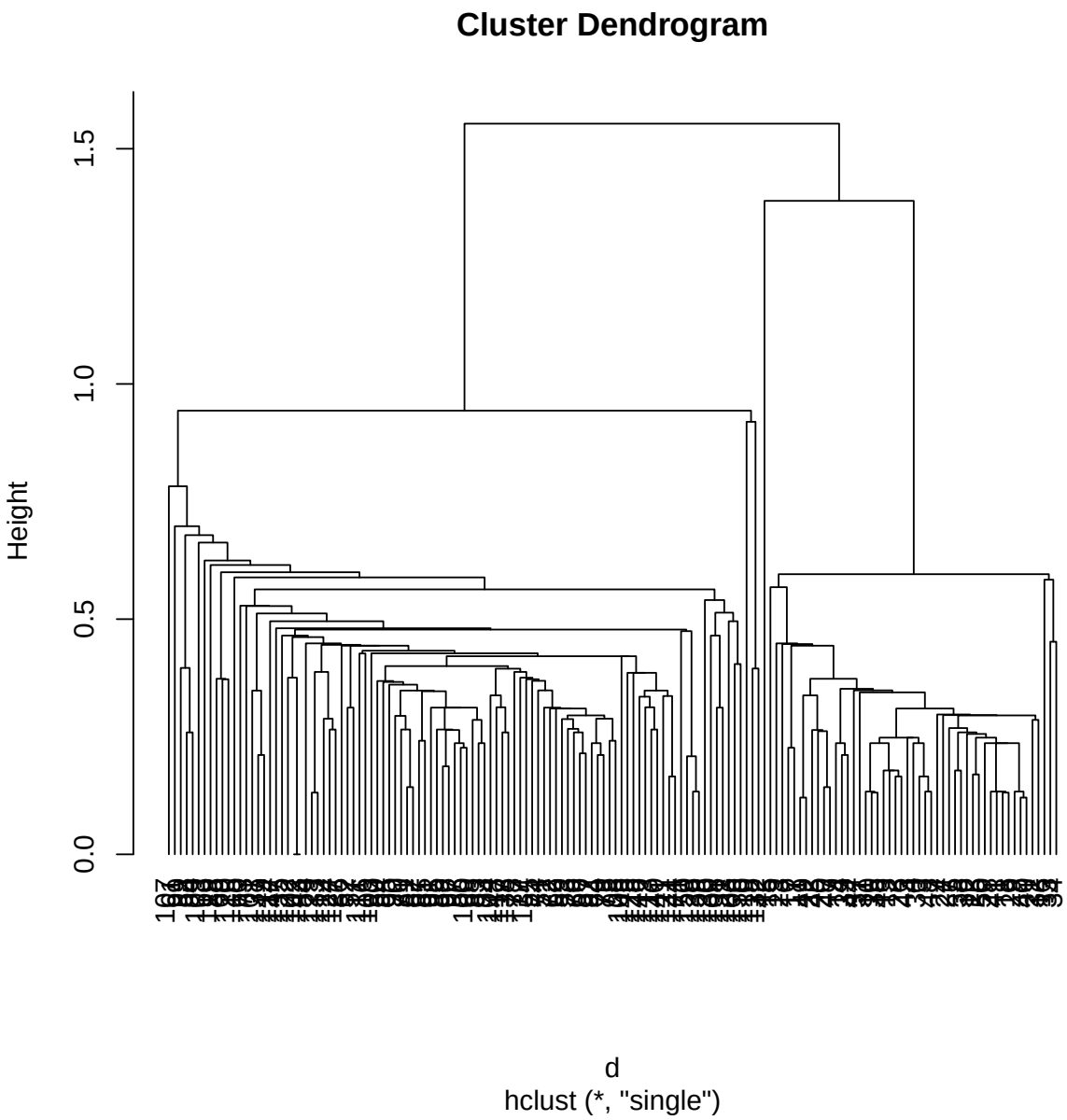
```
d <- dist(scale(subset(iris, select = -Species)))
h <- hclust(d, method = "complete")
plot(h, hang = -1)
```



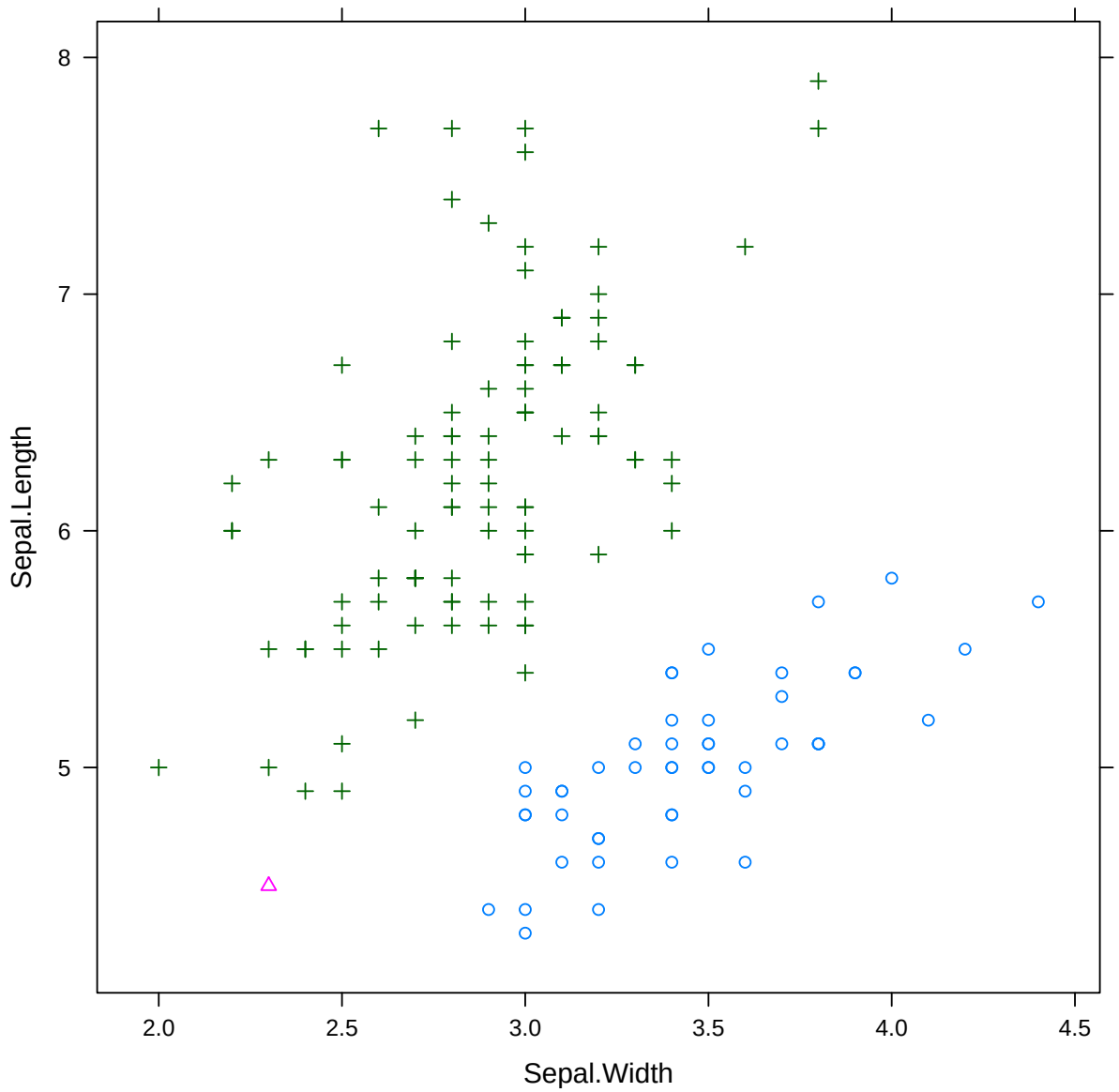
```r
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
    groups = cutree(h, k = 3), par.settings = simpleTheme(pch = 1:3))
```



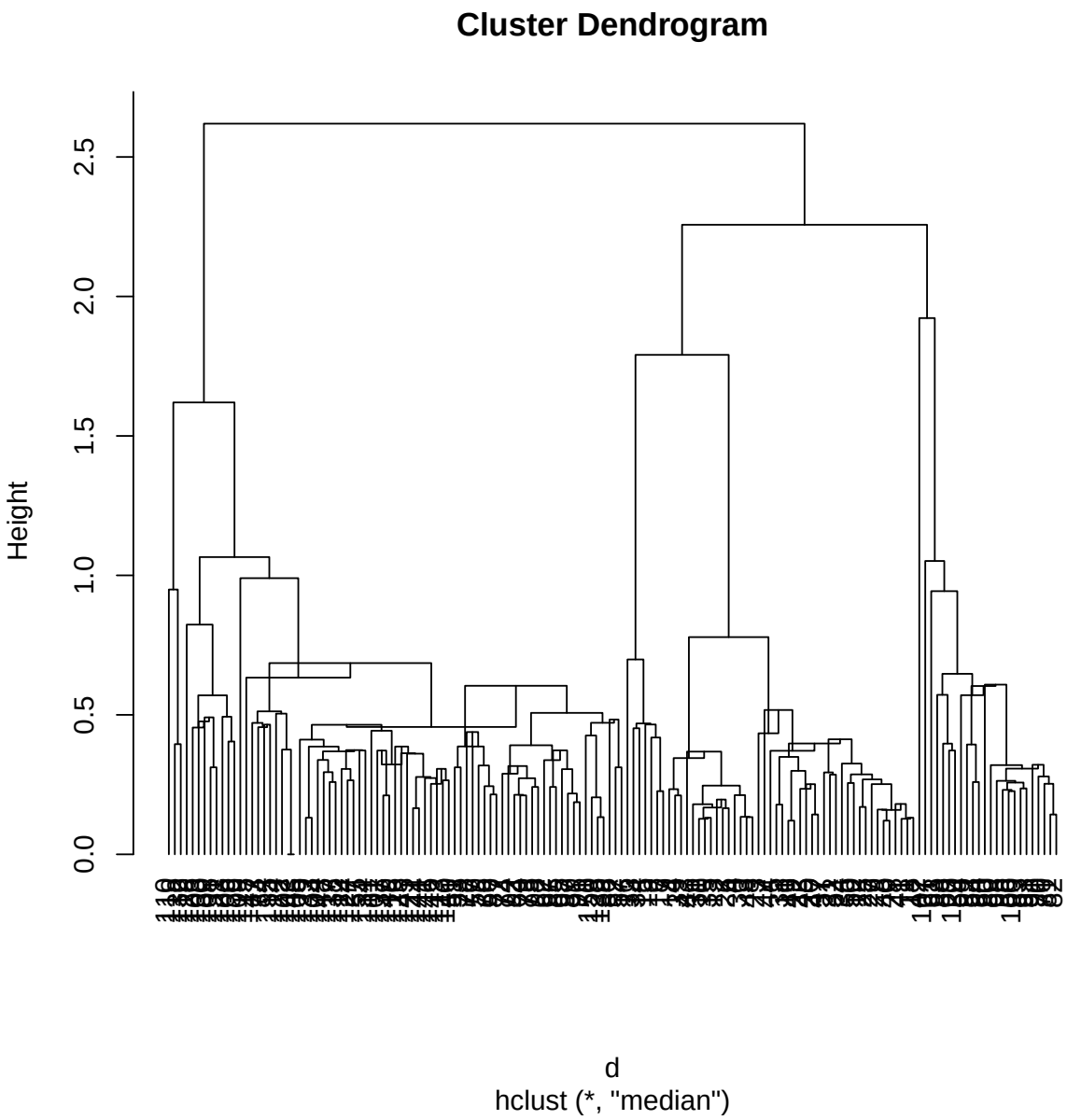

```r
h <- hclust(d, method = "single")
plot(h, hang = -1)
```



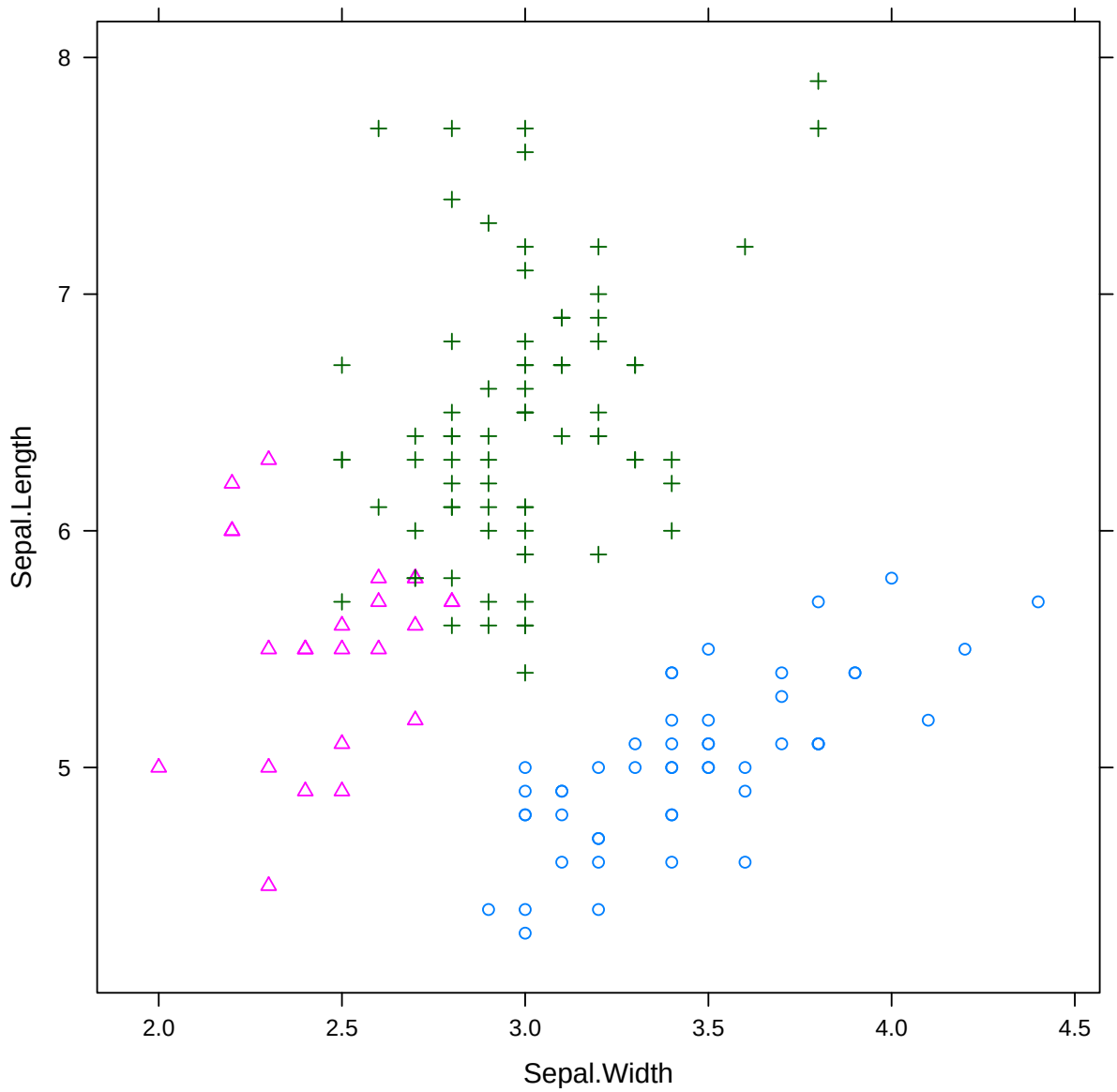

```
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
    groups = cutree(h, k = 3), par.settings = simpleTheme(pch = 1:3))
```




```
h <- hclust(d, method = "med")
plot(h, hang = -1)
```

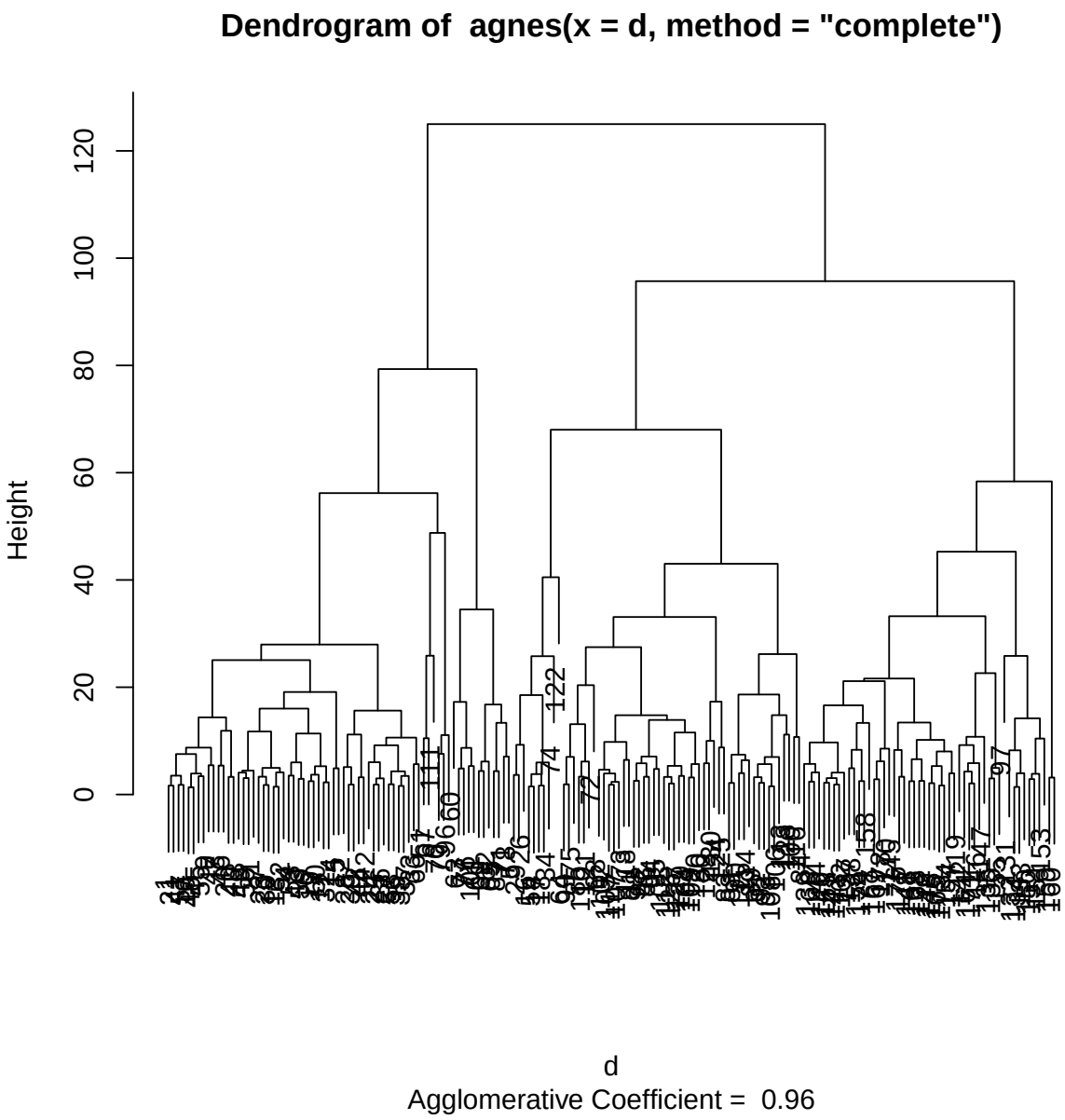



```
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
    groups = cutree(h, k = 3), par.settings = simpleTheme(pch = 1:3))
```




```r
library(cluster)  # agnes()
data(wine, package = "rattle")
head(wine)

##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols
## 1    1   14.23  1.71 2.43       15.6       127    2.80
## 2    1   13.20  1.78 2.14       11.2       100    2.65
## 3    1   13.16  2.36 2.67       18.6       101    2.80
## 4    1   14.37  1.95 2.50       16.8       113    3.85
## 5    1   13.24  2.59 2.87       21.0       118    2.80
## 6    1   14.20  1.76 2.45       15.2       112    3.27
##   Flavanoids Nonflavanoids Proanthocyanins Color  Hue
## 1       3.06          0.28            2.29  5.64 1.04
## 2       2.76          0.26            1.28  4.38 1.05
## 3       3.24          0.30            2.81  5.68 1.03
```

```
## 4        3.49          0.24            2.18  7.80 0.86
## 5        2.69          0.39            1.82  4.32 1.04
## 6        3.39          0.34            1.97  6.75 1.05
##   Dilution Proline
## 1     3.92    1065
## 2     3.40    1050
## 3     3.17    1185
## 4     3.45    1480
## 5     2.93     735
## 6     2.85    1450

d <- dist(scale(wine[, -1]))^2
h <- agnes(d, method = "complete")
plot(h, which.plot = 2)
```



```r
table(clust = cutree(h, k = 2), actual = wine$Type)
```

```
##      actual
## clust  1  2  3
##     1 51 18  0
##     2  8 53 48
```

```r
table(clust = cutree(h, k = 3), actual = wine$Type)
```

```
##      actual
## clust  1  2  3
##     1 51 18  0
##     2  8 50  0
##     3  0  3 48
```

```r
table(clust = cutree(h, k = 4), actual = wine$Type)
```

```
##      actual
## clust  1  2  3
##     1 51  6  0
##     2  8 50  0
##     3  0 12  0
##     4  0  3 48
```

```r
table(clust = cutree(h, k = 5), actual = wine$Type)
```

```
##      actual
## clust  1  2  3
##     1 51  6  0
##     2  8  2  0
##     3  0 12  0
##     4  0 48  0
##     5  0  3 48
```

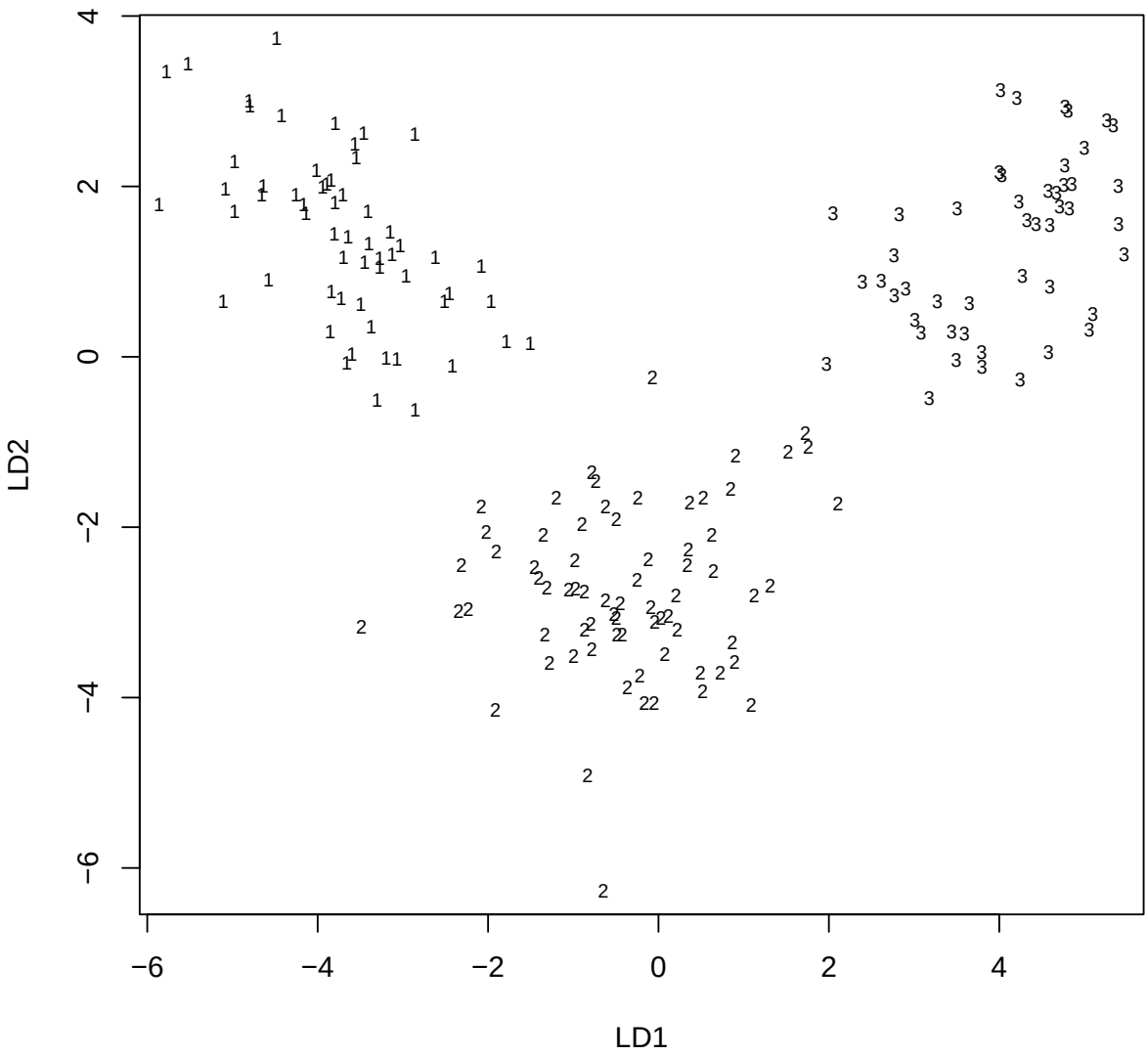
```r
table(clust = kmeans(scale(wine[, -1]), 3)$cluster,
    actual = wine$Type)
```

```
##      actual
## clust  1  2  3
##     1 59  3  0
##     2  0 65  0
##     3  0  3 48
```

```r
ld <- lda(Type ~ ., data = wine)
table(predicted = predict(ld, wine)$class,
    actual = wine$Type)
```

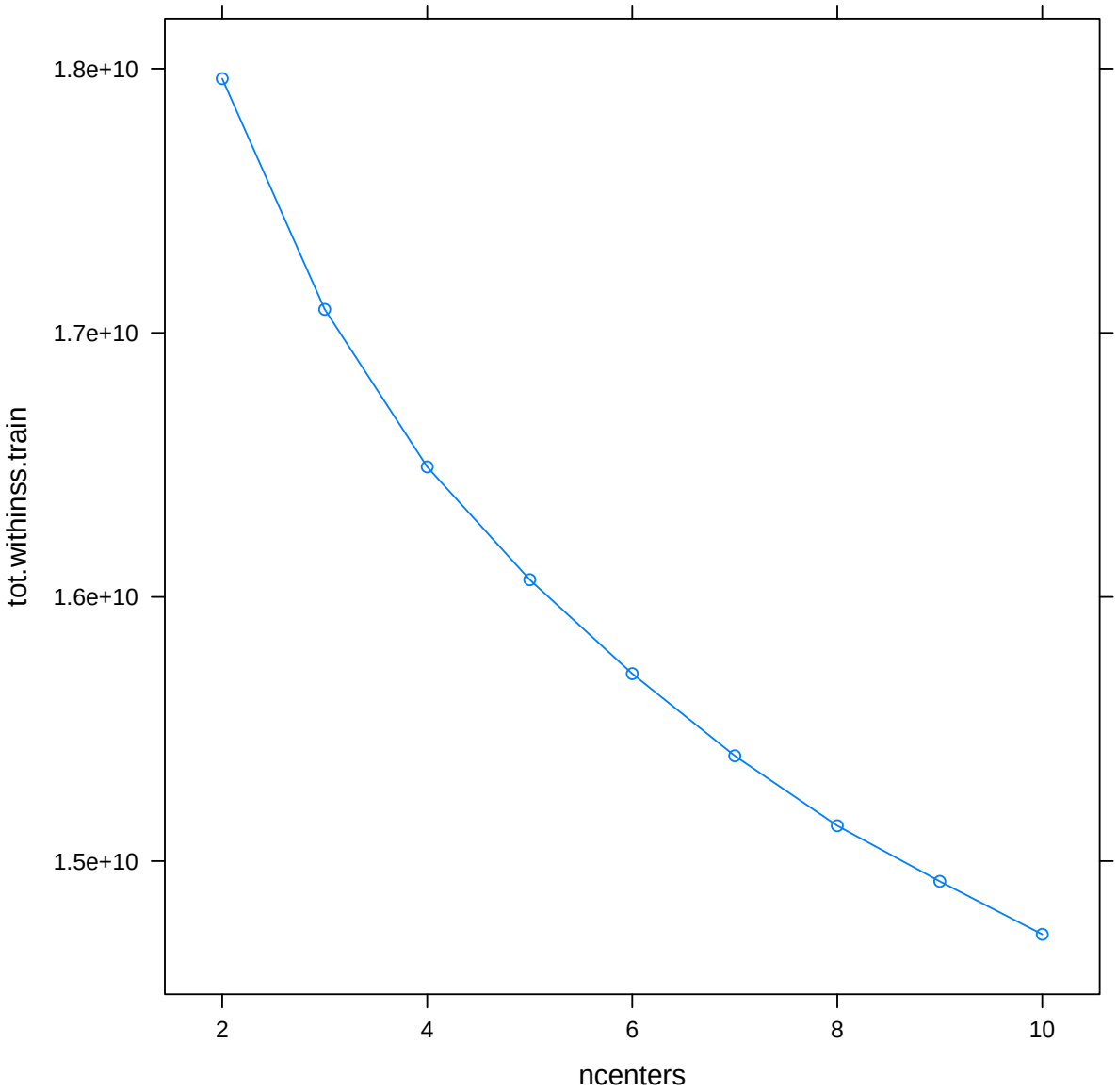
```
##          actual
## predicted  1  2  3
##         1 59  0  0
##         2  0 71  0
##         3  0  0 48
```

```r
plot(ld)
```

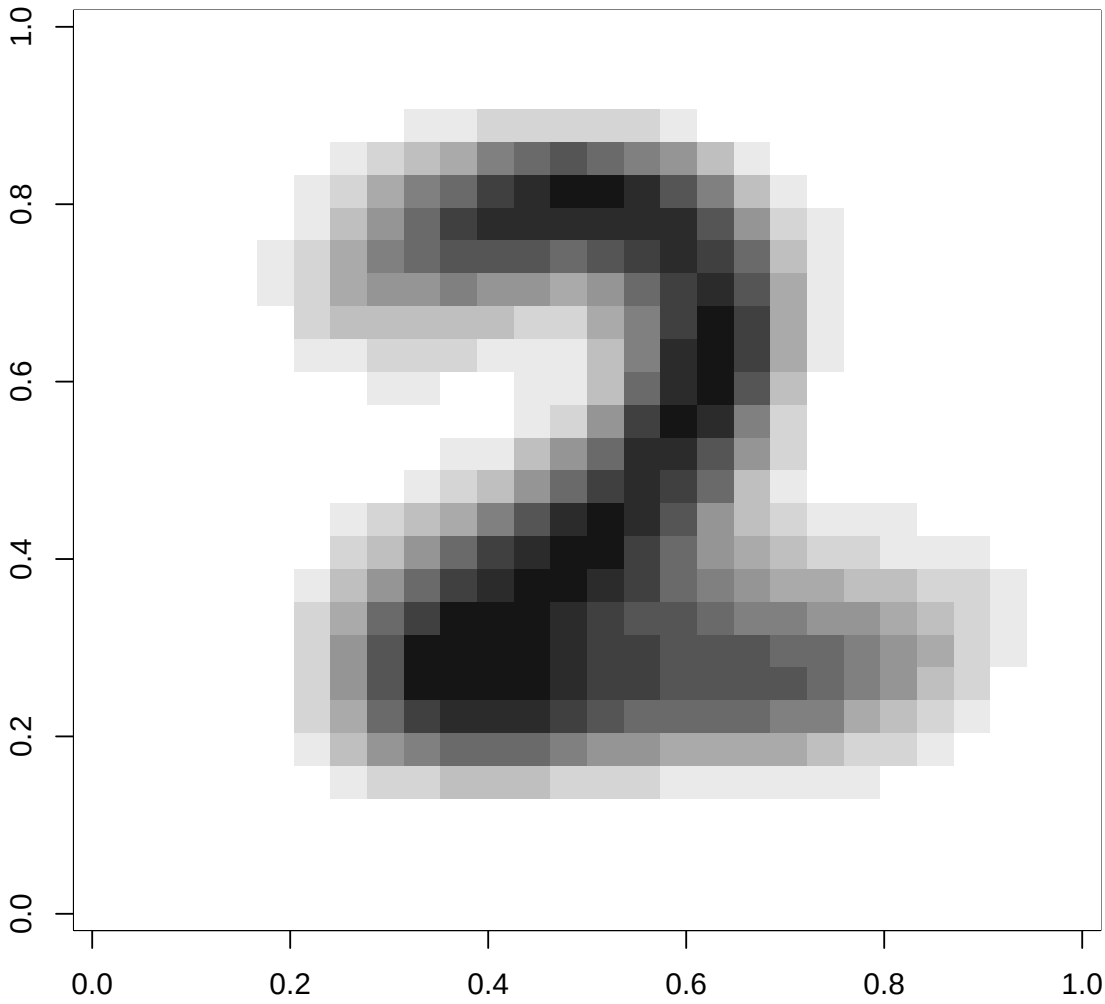



```
library(cclust)
load("mnist/mnist.rda")
twos <- X2 <- subset(mnist.train, y == 2,
    select = -y)
wssplot <- function(formula, data = NULL,
    n = 2:10, ...) {
    kcs <- lapply(n, function(n, ...) my.kmeans(formula,
        data = data, centers = n, ...), ...)
    tot.withinss.train <- sapply(kcs, function(kc) sum(kc$model$withinss))
    ncenters <- sapply(kcs, function(kc) kc$model$ncenters)
    xyplot(tot.withinss.train ~ ncenters,
        type = "b", auto.key = TRUE)
}
show_digit <- function(arr784, col = gray(12:1/12),
    ...) {
```

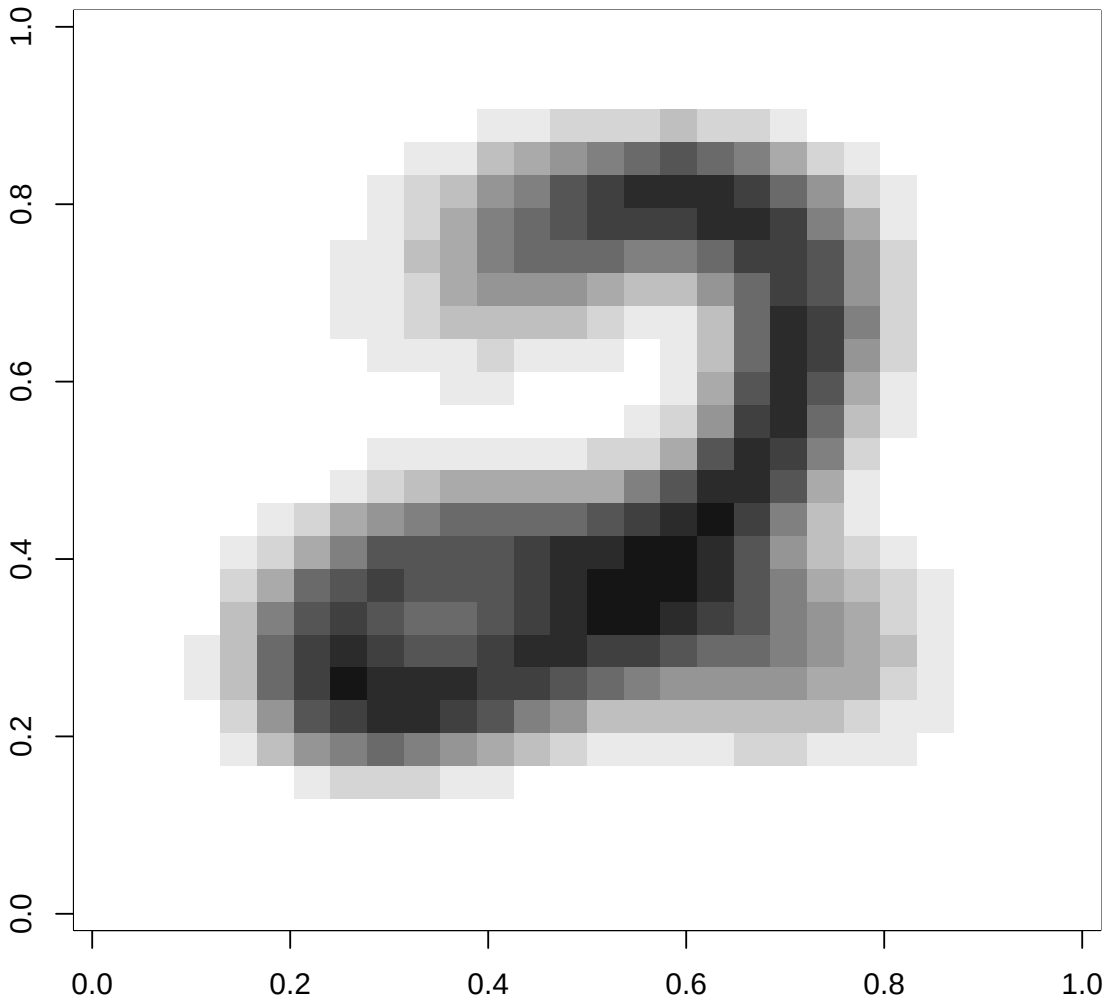
```
    image(matrix(arr784, nrow = 28)[, 28:1],
        col = col, ...)
}
wssplot(~., data = X2, scale = FALSE)
```




```
kc <- kmeans(X2, centers = 2)
agg <- aggregate(X2, list(cluster = kc$cluster),
    mean)
show_digit(as.matrix(agg[1, -1]))
```



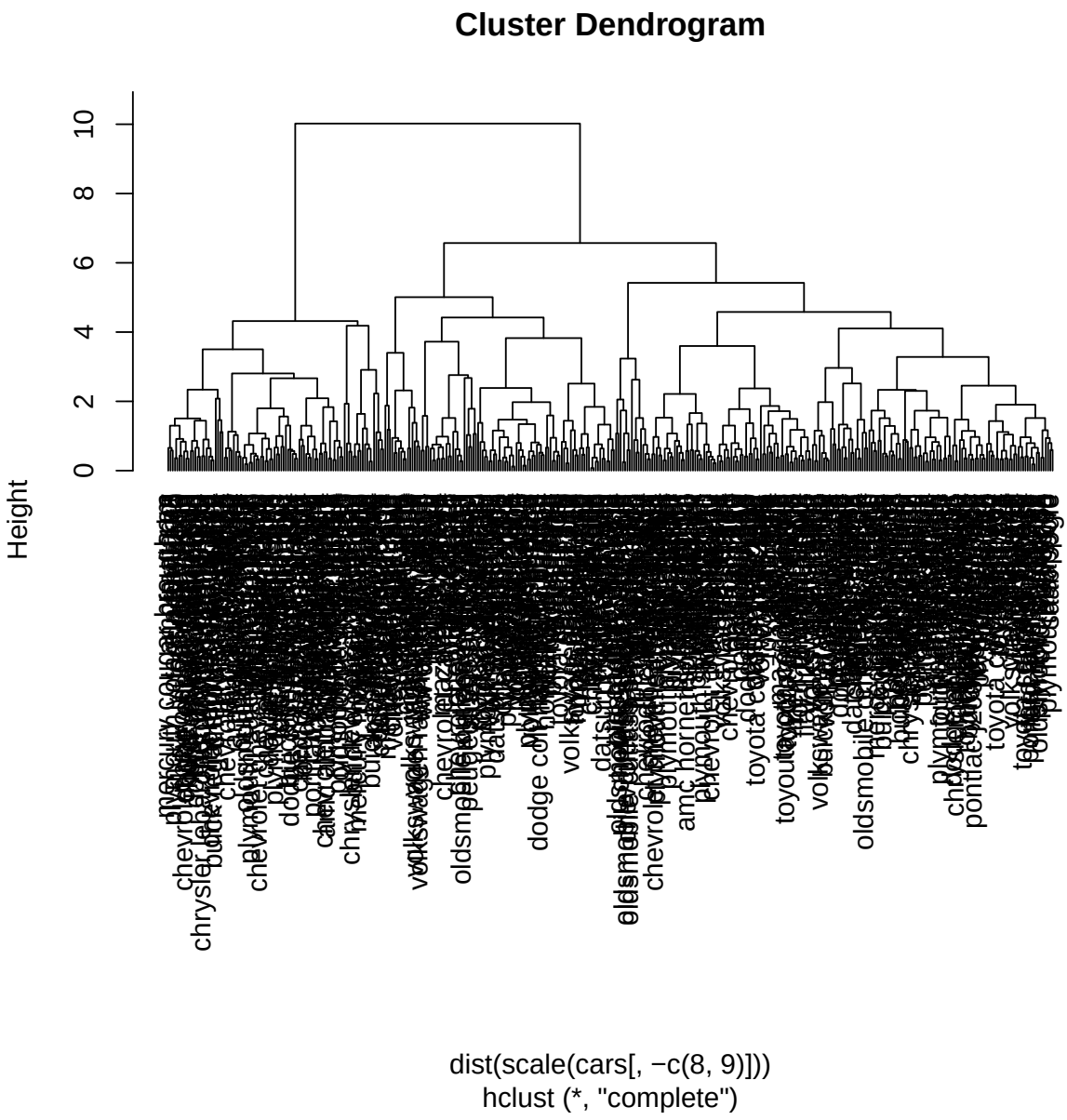

```
show_digit(as.matrix(agg[2, -1]))
```




```
cars <- read.table("clust/auto-mpg.tsv",
    header = TRUE, na.strings = "?", stringsAsFactors = FALSE)
cars$origin <- factor(cars$origin)
cars <- na.omit(cars)
head(cars)

##   mpg cylinders displacement horsepower weight acceleration
## 1  18         8          307        130   3504         12.0
## 2  15         8          350        165   3693         11.5
## 3  18         8          318        150   3436         11.0
## 4  16         8          304        150   3433         12.0
## 5  17         8          302        140   3449         10.5
## 6  15         8          429        198   4341         10.0
##   year origin                       name
## 1   70      1 chevrolet chevelle malibu
```

```
## 2   70      1          buick skylark 320
## 3   70      1         plymouth satellite
## 4   70      1              amc rebel sst
## 5   70      1                ford torino
## 6   70      1           ford galaxie 500
```

```
h <- hclust(dist(scale(cars[, -c(8, 9)])),
    method = "complete")
plot(h, hang = -1, labels = cars$name)
```




```
table(clust = cutree(h, k = 3), actual = cars$origin)

##       actual
## clust   1   2   3
##     1  96   0   0
```

```
##     2 121  38  35
##     3  28  30  44

cl <- data.frame(clust = cutree(h, k = 2),
    name = cars$name, hp = cars$horsepower)
cl[order(cl$clust), ]

##     clust                                name  hp
## 1       1            chevrolet chevelle malibu 130
## 2       1                    buick skylark 320 165
## 3       1                   plymouth satellite 150
## 4       1                        amc rebel sst 150
## 5       1                          ford torino 140
## 6       1                     ford galaxie 500 198
## 7       1                     chevrolet impala 220
## 8       1                    plymouth fury iii 215
## 9       1                     pontiac catalina 225
## 10      1                   amc ambassador dpl 190
## 11      1                  dodge challenger se 170
## 12      1                   plymouth 'cuda 340 160
## 13      1                chevrolet monte carlo 150
## 14      1              buick estate wagon (sw) 225
## 26      1                            ford f250 215
## 27      1                            chevy c20 200
## 28      1                           dodge d200 210
## 29      1                             hi 1200d 193
## 39      1                     chevrolet impala 165
## 40      1            pontiac catalina brougham 175
## 41      1                     ford galaxie 500 153
## 42      1                    plymouth fury iii 150
## 43      1                    dodge monaco (sw) 180
## 44      1             ford country squire (sw) 170
## 45      1                  pontiac safari (sw) 175
## 63      1                     chevrolet impala 165
## 64      1                     pontiac catalina 175
## 65      1                    plymouth fury iii 150
## 66      1                     ford galaxie 500 153
## 67      1                   amc ambassador sst 150
## 68      1                       mercury marquis 208
## 69      1                 buick lesabre custom 155
## 70      1           oldsmobile delta 88 royale 160
## 71      1               chrysler newport royal 190
## 73      1                     amc matador (sw) 150
## 74      1     chevrolet chevelle concours (sw) 130
## 75      1                ford gran torino (sw) 140
## 76      1       plymouth satellite custom (sw) 150
## 86      1                    buick century 350 175
## 87      1                          amc matador 150
```

```
## 88      1                     chevrolet malibu 145
## 89      1                     ford gran torino 137
## 90      1                dodge coronet custom 150
## 91      1             mercury marquis brougham 198
## 92      1            chevrolet caprice classic 150
## 93      1                             ford ltd 158
## 94      1             plymouth fury gran sedan 150
## 95      1         chrysler new yorker brougham 215
## 96      1             buick electra 225 custom 225
## 97      1              amc ambassador brougham 175
## 104     1                     chevrolet impala 150
## 105     1                         ford country 167
## 106     1               plymouth custom suburb 170
## 107     1             oldsmobile vista cruiser 180
## 116     1              chevrolet monte carlo s 145
## 117     1                  pontiac grand prix 230
## 122     1                    dodge dart custom 150
## 125     1                     oldsmobile omega 180
## 137     1                     ford gran torino 140
## 138     1             buick century luxus (sw) 150
## 139     1            dodge coronet custom (sw) 150
## 140     1                ford gran torino (sw) 140
## 141     1                     amc matador (sw) 150
## 157     1                     pontiac catalina 170
## 158     1                    chevrolet bel air 145
## 159     1                  plymouth grand fury 150
## 160     1                             ford ltd 148
## 166     1                  chevrolet monza 2+2 110
## 167     1                      ford mustang ii 129
## 188     1    chevrolet chevelle malibu classic 140
## 189     1               dodge coronet brougham 150
## 190     1                          amc matador 120
## 191     1                     ford gran torino 152
## 209     1           plymouth volare premier v8 150
## 213     1                     cadillac seville 180
## 214     1                            chevy c10 145
## 215     1                           ford f108 130
## 216     1                          dodge d100 150
## 222     1            chevrolet caprice classic 145
## 224     1                dodge monaco brougham 145
## 225     1              mercury cougar brougham 130
## 230     1                pontiac grand prix lj 180
## 231     1         chevrolet monte carlo landau 170
## 232     1                     chrysler cordoba 190
## 233     1                     ford thunderbird 149
## 251     1                       dodge diplomat 140
## 252     1                 mercury monarch ghia 139
## 263     1         chevrolet monte carlo landau 145
```

```
## 264     1      buick regal sport coupe (turbo) 165
## 265     1                          ford futura 139
## 266     1                      dodge magnum xe 140
## 287     1                      ford ltd landau 129
## 288     1                mercury grand marquis 138
## 290     1              buick estate wagon (sw) 155
## 291     1             ford country squire (sw) 142
## 293     1 chrysler lebaron town @ country (sw) 150
## 15      2                toyota corona mark ii  95
## 16      2                      plymouth duster  95
## 17      2                           amc hornet  97
## 18      2                        ford maverick  85
## 19      2                         datsun pl510  88
## 20      2         volkswagen 1131 deluxe sedan  46
## 21      2                         peugeot 504  87
## 22      2                         audi 100 ls  90
## 23      2                            saab 99e  95
## 24      2                            bmw 2002 113
## 25      2                         amc gremlin  90
## 30      2                         datsun pl510  88
## 31      2                  chevrolet vega 2300  90
## 32      2                        toyota corona  95
## 34      2                         amc gremlin 100
## 35      2            plymouth satellite custom 105
## 36      2            chevrolet chevelle malibu 100
## 37      2                      ford torino 500  88
## 38      2                         amc matador 100
## 46      2           amc hornet sportabout (sw) 110
## 47      2                  chevrolet vega (sw)  72
## 48      2                     pontiac firebird 100
## 49      2                         ford mustang  88
## 50      2                   mercury capri 2000  86
## 51      2                             opel 1900  90
## 52      2                         peugeot 304  70
## 53      2                           fiat 124b  76
## 54      2                  toyota corolla 1200  65
## 55      2                         datsun 1200  69
## 56      2                 volkswagen model 111  60
## 57      2                     plymouth cricket  70
## 58      2                toyota corona hardtop  95
## 59      2                   dodge colt hardtop  80
## 60      2                    volkswagen type 3  54
## 61      2                       chevrolet vega  90
## 62      2                  ford pinto runabout  86
## 72      2                      mazda rx2 coupe  97
## 77      2                      volvo 145e (sw) 112
## 78      2                  volkswagen 411 (sw)  76
## 79      2                     peugeot 504 (sw)  87
```

```
## 80      2                     renault 12 (sw)  69
## 81      2                     ford pinto (sw)  86
## 82      2                     datsun 510 (sw)  92
## 83      2          toyouta corona mark ii (sw)  97
## 84      2                     dodge colt (sw)  80
## 85      2             toyota corolla 1600 (sw)  88
## 98      2                    plymouth valiant 105
## 99      2                chevrolet nova custom 100
## 100     2                           amc hornet 100
## 101     2                        ford maverick  88
## 102     2                     plymouth duster  95
## 103     2              volkswagen super beetle  46
## 108     2                          amc gremlin 100
## 109     2                        toyota carina  88
## 110     2                       chevrolet vega  72
## 111     2                           datsun 610  94
## 112     2                            maxda rx3  90
## 113     2                           ford pinto  85
## 114     2                    mercury capri v6 107
## 115     2                 fiat 124 sport coupe  90
## 118     2                              fiat 128  49
## 119     2                           opel manta  75
## 120     2                           audi 100ls  91
## 121     2                          volvo 144ea 112
## 123     2                            saab 99le 110
## 124     2                       toyota mark ii 122
## 126     2                     plymouth duster  95
## 128     2                           amc hornet 100
## 129     2                       chevrolet nova 100
## 130     2                          datsun b210  67
## 131     2                           ford pinto  80
## 132     2                  toyota corolla 1200  65
## 133     2                       chevrolet vega  75
## 134     2   chevrolet chevelle malibu classic 100
## 135     2                          amc matador 110
## 136     2           plymouth satellite sebring 105
## 142     2                              audi fox  83
## 143     2                    volkswagen dasher  67
## 144     2                           opel manta  78
## 145     2                        toyota corona  52
## 146     2                           datsun 710  61
## 147     2                           dodge colt  75
## 148     2                              fiat 128  75
## 149     2                          fiat 124 tc  75
## 150     2                          honda civic  97
## 151     2                               subaru  93
## 152     2                             fiat x1.9  67
## 153     2              plymouth valiant custom  95
```

```
## 154     2                      chevrolet nova 105
## 155     2                     mercury monarch  72
## 156     2                       ford maverick  72
## 161     2                       buick century 110
## 162     2            chevroelt chevelle malibu 105
## 163     2                         amc matador 110
## 164     2                       plymouth fury  95
## 165     2                       buick skyhawk 110
## 168     2                      toyota corolla  75
## 169     2                          ford pinto  83
## 170     2                         amc gremlin 100
## 171     2                      pontiac astro  78
## 172     2                      toyota corona  96
## 173     2                   volkswagen dasher  71
## 174     2                          datsun 710  97
## 175     2                          ford pinto  97
## 176     2                   volkswagen rabbit  70
## 177     2                           amc pacer  90
## 178     2                          audi 100ls  95
## 179     2                        peugeot 504  88
## 180     2                        volvo 244dl  98
## 181     2                           saab 99le 115
## 182     2                    honda civic cvcc  53
## 183     2                            fiat 131  86
## 184     2                           opel 1900  81
## 185     2                            capri ii  92
## 186     2                         dodge colt  79
## 187     2                       renault 12tl  83
## 192     2                    plymouth valiant 100
## 193     2                      chevrolet nova 105
## 194     2                       ford maverick  81
## 195     2                          amc hornet  90
## 196     2                  chevrolet chevette  52
## 197     2                     chevrolet woody  60
## 198     2                           vw rabbit  70
## 199     2                        honda civic  53
## 200     2                      dodge aspen se 100
## 201     2                    ford granada ghia  78
## 202     2                  pontiac ventura sj 110
## 203     2                       amc pacer d/l  95
## 204     2                   volkswagen rabbit  71
## 205     2                        datsun b-210  70
## 206     2                      toyota corolla  75
## 207     2                          ford pinto  72
## 208     2                           volvo 245 102
## 210     2                        peugeot 504  88
## 211     2                      toyota mark ii 108
## 212     2                  mercedes-benz 280s 120
```

```
## 217     2                    honda accord cvcc  68
## 218     2             buick opel isuzu deluxe  80
## 219     2                       renault 5 gtl  58
## 220     2                   plymouth arrow gs  96
## 221     2               datsun f-10 hatchback  70
## 223     2          oldsmobile cutlass supreme 110
## 226     2                  chevrolet concours 110
## 227     2                       buick skylark 105
## 228     2              plymouth volare custom 100
## 229     2                        ford granada  98
## 234     2            volkswagen rabbit custom  78
## 235     2               pontiac sunbird coupe  88
## 236     2             toyota corolla liftback  75
## 237     2                 ford mustang ii 2+2  89
## 238     2                  chevrolet chevette  63
## 239     2                      dodge colt m/m  83
## 240     2                           subaru dl  67
## 241     2                   volkswagen dasher  78
## 242     2                          datsun 810  97
## 243     2                            bmw 320i 110
## 244     2                          mazda rx-4 110
## 245     2      volkswagen rabbit custom diesel  48
## 246     2                         ford fiesta  66
## 247     2                    mazda glc deluxe  52
## 248     2                      datsun b210 gx  70
## 249     2                    honda civic cvcc  60
## 250     2   oldsmobile cutlass salon brougham 110
## 253     2                  pontiac phoenix lj 105
## 254     2                    chevrolet malibu  95
## 255     2                ford fairmont (auto)  85
## 256     2                 ford fairmont (man)  88
## 257     2                    plymouth volare 100
## 258     2                        amc concord  90
## 259     2               buick century special 105
## 260     2                      mercury zephyr  85
## 261     2                        dodge aspen 110
## 262     2                     amc concord d/l 120
## 267     2                  chevrolet chevette  68
## 268     2                       toyota corona  95
## 269     2                          datsun 510  97
## 270     2                          dodge omni  75
## 271     2           toyota celica gt liftback  95
## 272     2                    plymouth sapporo 105
## 273     2               oldsmobile starfire sx  85
## 274     2                       datsun 200-sx  97
## 275     2                           audi 5000 103
## 276     2                         volvo 264gl 125
## 277     2                          saab 99gle 115
```

```
## 278     2                       peugeot 604sl 133
## 279     2                  volkswagen scirocco  71
## 280     2                      honda accord lx  68
## 281     2                    pontiac lemans v6 115
## 282     2                     mercury zephyr 6  85
## 283     2                      ford fairmont 4  88
## 284     2                     amc concord dl 6  90
## 285     2                        dodge aspen 6 110
## 286     2             chevrolet caprice classic 130
## 289     2                      dodge st. regis 135
## 292     2         chevrolet malibu classic (sw) 125
## 294     2                     vw rabbit custom  71
## 295     2                     maxda glc deluxe  65
## 296     2           dodge colt hatchback custom  80
## 297     2                        amc spirit dl  80
## 298     2                   mercedes benz 300d  77
## 299     2                    cadillac eldorado 125
## 300     2                          peugeot 504  71
## 301     2    oldsmobile cutlass salon brougham  90
## 302     2                    plymouth horizon  70
## 303     2                 plymouth horizon tc3  70
## 304     2                           datsun 210  65
## 305     2                   fiat strada custom  69
## 306     2                 buick skylark limited  90
## 307     2                   chevrolet citation 115
## 308     2             oldsmobile omega brougham 115
## 309     2                      pontiac phoenix  90
## 310     2                            vw rabbit  76
## 311     2                 toyota corolla tercel  60
## 312     2                   chevrolet chevette  70
## 313     2                           datsun 310  65
## 314     2                   chevrolet citation  90
## 315     2                        ford fairmont  88
## 316     2                          amc concord  90
## 317     2                          dodge aspen  90
## 318     2                            audi 4000  78
## 319     2                toyota corona liftback  90
## 320     2                            mazda 626  75
## 321     2                  datsun 510 hatchback  92
## 322     2                        toyota corolla  75
## 323     2                            mazda glc  65
## 324     2                           dodge colt 105
## 325     2                           datsun 210  65
## 326     2                 vw rabbit c (diesel)  48
## 327     2                   vw dasher (diesel)  48
## 328     2                  audi 5000s (diesel)  67
## 329     2                   mercedes-benz 240d  67
## 330     2                  honda civic 1500 gl  67
```

```
## 332     2                          subaru dl  67
## 333     2                    vokswagen rabbit  62
## 334     2                       datsun 280-zx 132
## 335     2                       mazda rx-7 gs 100
## 336     2                   triumph tr7 coupe  88
## 338     2                        honda accord  72
## 339     2                    plymouth reliant  84
## 340     2                       buick skylark  84
## 341     2              dodge aries wagon (sw)  92
## 342     2                  chevrolet citation 110
## 343     2                    plymouth reliant  84
## 344     2                      toyota starlet  58
## 345     2                      plymouth champ  64
## 346     2                    honda civic 1300  60
## 347     2                              subaru  67
## 348     2                      datsun 210 mpg  65
## 349     2                       toyota tercel  62
## 350     2                         mazda glc 4  68
## 351     2                  plymouth horizon 4  63
## 352     2                      ford escort 4w  65
## 353     2                      ford escort 2h  65
## 354     2                    volkswagen jetta  74
## 356     2                       honda prelude  75
## 357     2                      toyota corolla  75
## 358     2                        datsun 200sx 100
## 359     2                           mazda 626  74
## 360     2           peugeot 505s turbo diesel  80
## 361     2                        volvo diesel  76
## 362     2                     toyota cressida 116
## 363     2                   datsun 810 maxima 120
## 364     2                       buick century 110
## 365     2               oldsmobile cutlass ls 105
## 366     2                     ford granada gl  88
## 367     2              chrysler lebaron salon  85
## 368     2                  chevrolet cavalier  88
## 369     2            chevrolet cavalier wagon  88
## 370     2           chevrolet cavalier 2-door  88
## 371     2          pontiac j2000 se hatchback  85
## 372     2                      dodge aries se  84
## 373     2                     pontiac phoenix  90
## 374     2                ford fairmont futura  92
## 376     2                 volkswagen rabbit l  74
## 377     2                  mazda glc custom l  68
## 378     2                    mazda glc custom  68
## 379     2              plymouth horizon miser  63
## 380     2                      mercury lynx l  70
## 381     2                    nissan stanza xe  88
## 382     2                        honda accord  75
```

```
## 383     2                       toyota corolla  70
## 384     2                          honda civic  67
## 385     2                    honda civic (auto)  67
## 386     2                         datsun 310 gx  67
## 387     2                  buick century limited 110
## 388     2    oldsmobile cutlass ciera (diesel)  85
## 389     2            chrysler lebaron medallion  92
## 390     2                       ford granada l 112
## 391     2                     toyota celica gt  96
## 392     2                    dodge charger 2.2  84
## 393     2                     chevrolet camaro  90
## 394     2                      ford mustang gl  86
## 395     2                            vw pickup  52
## 396     2                        dodge rampage  84
## 397     2                          ford ranger  79
## 398     2                           chevy s-10  82
```

```
cl <- data.frame(clust = cutree(h, k = 4),
    name = cars$name, hp = cars$horsepower)
cl[order(cl$clust), ]
```

```
##     clust                                name  hp
## 1       1            chevrolet chevelle malibu 130
## 2       1                    buick skylark 320 165
## 3       1                   plymouth satellite 150
## 4       1                        amc rebel sst 150
## 5       1                          ford torino 140
## 6       1                     ford galaxie 500 198
## 7       1                     chevrolet impala 220
## 8       1                    plymouth fury iii 215
## 9       1                     pontiac catalina 225
## 10      1                   amc ambassador dpl 190
## 11      1                  dodge challenger se 170
## 12      1                   plymouth 'cuda 340 160
## 13      1                 chevrolet monte carlo 150
## 14      1              buick estate wagon (sw) 225
## 26      1                             ford f250 215
## 27      1                             chevy c20 200
## 28      1                            dodge d200 210
## 29      1                              hi 1200d 193
## 39      1                     chevrolet impala 165
## 40      1            pontiac catalina brougham 175
## 41      1                     ford galaxie 500 153
## 42      1                    plymouth fury iii 150
## 43      1                    dodge monaco (sw) 180
## 44      1             ford country squire (sw) 170
## 45      1                  pontiac safari (sw) 175
## 63      1                     chevrolet impala 165
```

```
## 64      1                     pontiac catalina 175
## 65      1                    plymouth fury iii 150
## 66      1                     ford galaxie 500 153
## 67      1                   amc ambassador sst 150
## 68      1                      mercury marquis 208
## 69      1                 buick lesabre custom 155
## 70      1           oldsmobile delta 88 royale 160
## 71      1               chrysler newport royal 190
## 73      1                     amc matador (sw) 150
## 74      1     chevrolet chevelle concours (sw) 130
## 75      1                ford gran torino (sw) 140
## 76      1       plymouth satellite custom (sw) 150
## 86      1                    buick century 350 175
## 87      1                          amc matador 150
## 88      1                     chevrolet malibu 145
## 89      1                     ford gran torino 137
## 90      1                 dodge coronet custom 150
## 91      1             mercury marquis brougham 198
## 92      1            chevrolet caprice classic 150
## 93      1                              ford ltd 158
## 94      1             plymouth fury gran sedan 150
## 95      1         chrysler new yorker brougham 215
## 96      1             buick electra 225 custom 225
## 97      1              amc ambassador brougham 175
## 104     1                     chevrolet impala 150
## 105     1                         ford country 167
## 106     1              plymouth custom suburb 170
## 107     1             oldsmobile vista cruiser 180
## 116     1              chevrolet monte carlo s 145
## 117     1                   pontiac grand prix 230
## 122     1                    dodge dart custom 150
## 125     1                     oldsmobile omega 180
## 137     1                     ford gran torino 140
## 138     1             buick century luxus (sw) 150
## 139     1            dodge coronet custom (sw) 150
## 140     1                ford gran torino (sw) 140
## 141     1                     amc matador (sw) 150
## 157     1                     pontiac catalina 170
## 158     1                    chevrolet bel air 145
## 159     1                  plymouth grand fury 150
## 160     1                              ford ltd 148
## 166     1                  chevrolet monza 2+2 110
## 167     1                      ford mustang ii 129
## 188     1    chevrolet chevelle malibu classic 140
## 189     1                dodge coronet brougham 150
## 190     1                          amc matador 120
## 191     1                     ford gran torino 152
## 209     1           plymouth volare premier v8 150
```

```
## 213     1                     cadillac seville 180
## 214     1                            chevy c10 145
## 215     1                            ford f108 130
## 216     1                           dodge d100 150
## 222     1            chevrolet caprice classic 145
## 224     1                dodge monaco brougham 145
## 225     1              mercury cougar brougham 130
## 230     1                pontiac grand prix lj 180
## 231     1         chevrolet monte carlo landau 170
## 232     1                     chrysler cordoba 190
## 233     1                     ford thunderbird 149
## 251     1                       dodge diplomat 140
## 252     1                 mercury monarch ghia 139
## 263     1         chevrolet monte carlo landau 145
## 264     1      buick regal sport coupe (turbo) 165
## 265     1                          ford futura 139
## 266     1                      dodge magnum xe 140
## 287     1                      ford ltd landau 129
## 288     1                mercury grand marquis 138
## 290     1              buick estate wagon (sw) 155
## 291     1             ford country squire (sw) 142
## 293     1 chrysler lebaron town @ country (sw) 150
## 15      2                toyota corona mark ii  95
## 16      2                      plymouth duster  95
## 17      2                           amc hornet  97
## 18      2                        ford maverick  85
## 19      2                         datsun pl510  88
## 21      2                          peugeot 504  87
## 22      2                          audi 100 ls  90
## 23      2                             saab 99e  95
## 24      2                             bmw 2002 113
## 25      2                          amc gremlin  90
## 30      2                         datsun pl510  88
## 31      2                   chevrolet vega 2300  90
## 32      2                        toyota corona  95
## 34      2                          amc gremlin 100
## 35      2            plymouth satellite custom 105
## 36      2            chevrolet chevelle malibu 100
## 37      2                      ford torino 500  88
## 38      2                          amc matador 100
## 46      2           amc hornet sportabout (sw) 110
## 48      2                     pontiac firebird 100
## 49      2                         ford mustang  88
## 50      2                    mercury capri 2000  86
## 51      2                            opel 1900  90
## 53      2                            fiat 124b  76
## 58      2                toyota corona hardtop  95
## 59      2                   dodge colt hardtop  80
```

```
## 62      2                  ford pinto runabout  86
## 72      2                      mazda rx2 coupe  97
## 77      2                      volvo 145e (sw) 112
## 80      2                      renault 12 (sw)  69
## 81      2                      ford pinto (sw)  86
## 82      2                      datsun 510 (sw)  92
## 83      2          toyouta corona mark ii (sw)  97
## 84      2                      dodge colt (sw)  80
## 85      2             toyota corolla 1600 (sw)  88
## 98      2                    plymouth valiant 105
## 99      2                chevrolet nova custom 100
## 100     2                           amc hornet 100
## 101     2                        ford maverick  88
## 102     2                      plymouth duster  95
## 108     2                          amc gremlin 100
## 111     2                           datsun 610  94
## 112     2                            maxda rx3  90
## 114     2                     mercury capri v6 107
## 115     2                 fiat 124 sport coupe  90
## 119     2                           opel manta  75
## 120     2                           audi 100ls  91
## 121     2                          volvo 144ea 112
## 123     2                            saab 99le 110
## 124     2                       toyota mark ii 122
## 126     2                      plymouth duster  95
## 128     2                           amc hornet 100
## 129     2                       chevrolet nova 100
## 131     2                           ford pinto  80
## 133     2                       chevrolet vega  75
## 134     2    chevrolet chevelle malibu classic 100
## 136     2           plymouth satellite sebring 105
## 142     2                             audi fox  83
## 143     2                    volkswagen dasher  67
## 144     2                           opel manta  78
## 147     2                           dodge colt  75
## 148     2                             fiat 128  75
## 149     2                          fiat 124 tc  75
## 150     2                          honda civic  97
## 151     2                               subaru  93
## 153     2              plymouth valiant custom  95
## 154     2                       chevrolet nova 105
## 165     2                        buick skyhawk 110
## 168     2                       toyota corolla  75
## 169     2                           ford pinto  83
## 170     2                          amc gremlin 100
## 171     2                        pontiac astro  78
## 172     2                        toyota corona  96
## 173     2                    volkswagen dasher  71
```

```
## 174     2                          datsun 710  97
## 175     2                          ford pinto  97
## 176     2                   volkswagen rabbit  70
## 177     2                           amc pacer  90
## 178     2                          audi 100ls  95
## 179     2                         peugeot 504  88
## 180     2                         volvo 244dl  98
## 181     2                           saab 99le 115
## 183     2                            fiat 131  86
## 184     2                           opel 1900  81
## 185     2                            capri ii  92
## 186     2                          dodge colt  79
## 187     2                        renault 12tl  83
## 192     2                    plymouth valiant 100
## 193     2                      chevrolet nova 105
## 194     2                       ford maverick  81
## 195     2                          amc hornet  90
## 198     2                           vw rabbit  70
## 200     2                       dodge aspen se 100
## 202     2                   pontiac ventura sj 110
## 203     2                        amc pacer d/l  95
## 204     2                   volkswagen rabbit  71
## 206     2                      toyota corolla  75
## 207     2                          ford pinto  72
## 208     2                           volvo 245 102
## 211     2                      toyota mark ii 108
## 212     2                   mercedes-benz 280s 120
## 218     2               buick opel isuzu deluxe  80
## 220     2                    plymouth arrow gs  96
## 226     2                   chevrolet concours 110
## 227     2                        buick skylark 105
## 228     2                plymouth volare custom 100
## 229     2                         ford granada  98
## 234     2              volkswagen rabbit custom  78
## 235     2                 pontiac sunbird coupe  88
## 236     2                toyota corolla liftback  75
## 237     2                    ford mustang ii 2+2  89
## 241     2                   volkswagen dasher  78
## 242     2                          datsun 810  97
## 243     2                            bmw 320i 110
## 244     2                          mazda rx-4 110
## 253     2                   pontiac phoenix lj 105
## 254     2                     chevrolet malibu  95
## 255     2                 ford fairmont (auto)  85
## 256     2                  ford fairmont (man)  88
## 257     2                      plymouth volare 100
## 258     2                          amc concord  90
## 259     2                 buick century special 105
```

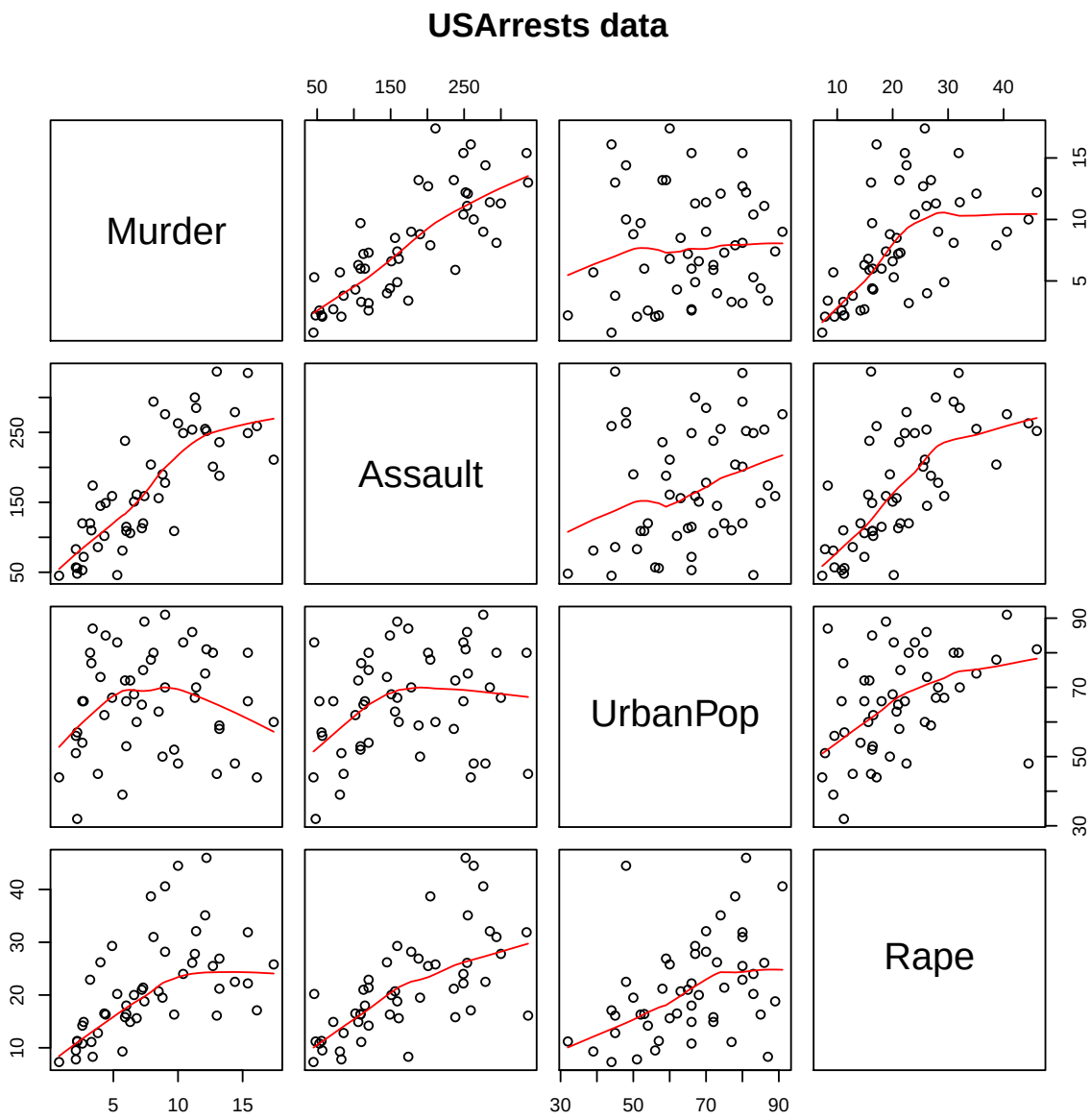
```
## 260     2                       mercury zephyr  85
## 261     2                          dodge aspen 110
## 262     2                      amc concord d/l 120
## 268     2                        toyota corona  95
## 269     2                           datsun 510  97
## 270     2                           dodge omni  75
## 271     2            toyota celica gt liftback  95
## 272     2                     plymouth sapporo 105
## 273     2                oldsmobile starfire sx  85
## 274     2                       datsun 200-sx  97
## 275     2                            audi 5000 103
## 276     2                          volvo 264gl 125
## 277     2                           saab 99gle 115
## 278     2                        peugeot 604sl 133
## 281     2                     pontiac lemans v6 115
## 282     2                     mercury zephyr 6  85
## 283     2                      ford fairmont 4  88
## 284     2                     amc concord dl 6  90
## 285     2                        dodge aspen 6 110
## 297     2                        amc spirit dl  80
## 306     2                 buick skylark limited  90
## 307     2                    chevrolet citation 115
## 308     2            oldsmobile omega brougham 115
## 309     2                      pontiac phoenix  90
## 314     2                    chevrolet citation  90
## 315     2                        ford fairmont  88
## 317     2                          dodge aspen  90
## 319     2                toyota corona liftback  90
## 324     2                           dodge colt 105
## 334     2                        datsun 280-zx 132
## 335     2                        mazda rx-7 gs 100
## 339     2                     plymouth reliant  84
## 340     2                        buick skylark  84
## 341     2                dodge aries wagon (sw)  92
## 342     2                    chevrolet citation 110
## 343     2                     plymouth reliant  84
## 358     2                         datsun 200sx 100
## 362     2                       toyota cressida 116
## 363     2                     datsun 810 maxima 120
## 364     2                        buick century 110
## 366     2                       ford granada gl  88
## 367     2                chrysler lebaron salon  85
## 371     2            pontiac j2000 se hatchback  85
## 372     2                        dodge aries se  84
## 373     2                       pontiac phoenix  90
## 374     2                  ford fairmont futura  92
## 381     2                     nissan stanza xe  88
## 382     2                         honda accord  75
```

```
## 387     2                buick century limited 110
## 389     2           chrysler lebaron medallion  92
## 390     2                       ford granada l 112
## 391     2                     toyota celica gt  96
## 392     2                    dodge charger 2.2  84
## 393     2                     chevrolet camaro  90
## 394     2                      ford mustang gl  86
## 396     2                        dodge rampage  84
## 20      3         volkswagen 1131 deluxe sedan  46
## 47      3                  chevrolet vega (sw)  72
## 52      3                          peugeot 304  70
## 54      3                  toyota corolla 1200  65
## 55      3                          datsun 1200  69
## 56      3                 volkswagen model 111  60
## 57      3                     plymouth cricket  70
## 60      3                    volkswagen type 3  54
## 61      3                       chevrolet vega  90
## 78      3                  volkswagen 411 (sw)  76
## 79      3                     peugeot 504 (sw)  87
## 103     3              volkswagen super beetle  46
## 109     3                        toyota carina  88
## 110     3                       chevrolet vega  72
## 113     3                           ford pinto  85
## 118     3                             fiat 128  49
## 130     3                          datsun b210  67
## 132     3                  toyota corolla 1200  65
## 145     3                       toyota corona  52
## 146     3                          datsun 710  61
## 152     3                            fiat x1.9  67
## 182     3                     honda civic cvcc  53
## 196     3                   chevrolet chevette  52
## 197     3                      chevrolet woody  60
## 199     3                          honda civic  53
## 205     3                         datsun b-210  70
## 210     3                          peugeot 504  88
## 217     3                    honda accord cvcc  68
## 219     3                        renault 5 gtl  58
## 221     3                datsun f-10 hatchback  70
## 238     3                   chevrolet chevette  63
## 239     3                       dodge colt m/m  83
## 240     3                            subaru dl  67
## 245     3      volkswagen rabbit custom diesel  48
## 246     3                          ford fiesta  66
## 247     3                     mazda glc deluxe  52
## 248     3                       datsun b210 gx  70
## 249     3                     honda civic cvcc  60
## 267     3                   chevrolet chevette  68
## 279     3                  volkswagen scirocco  71
```

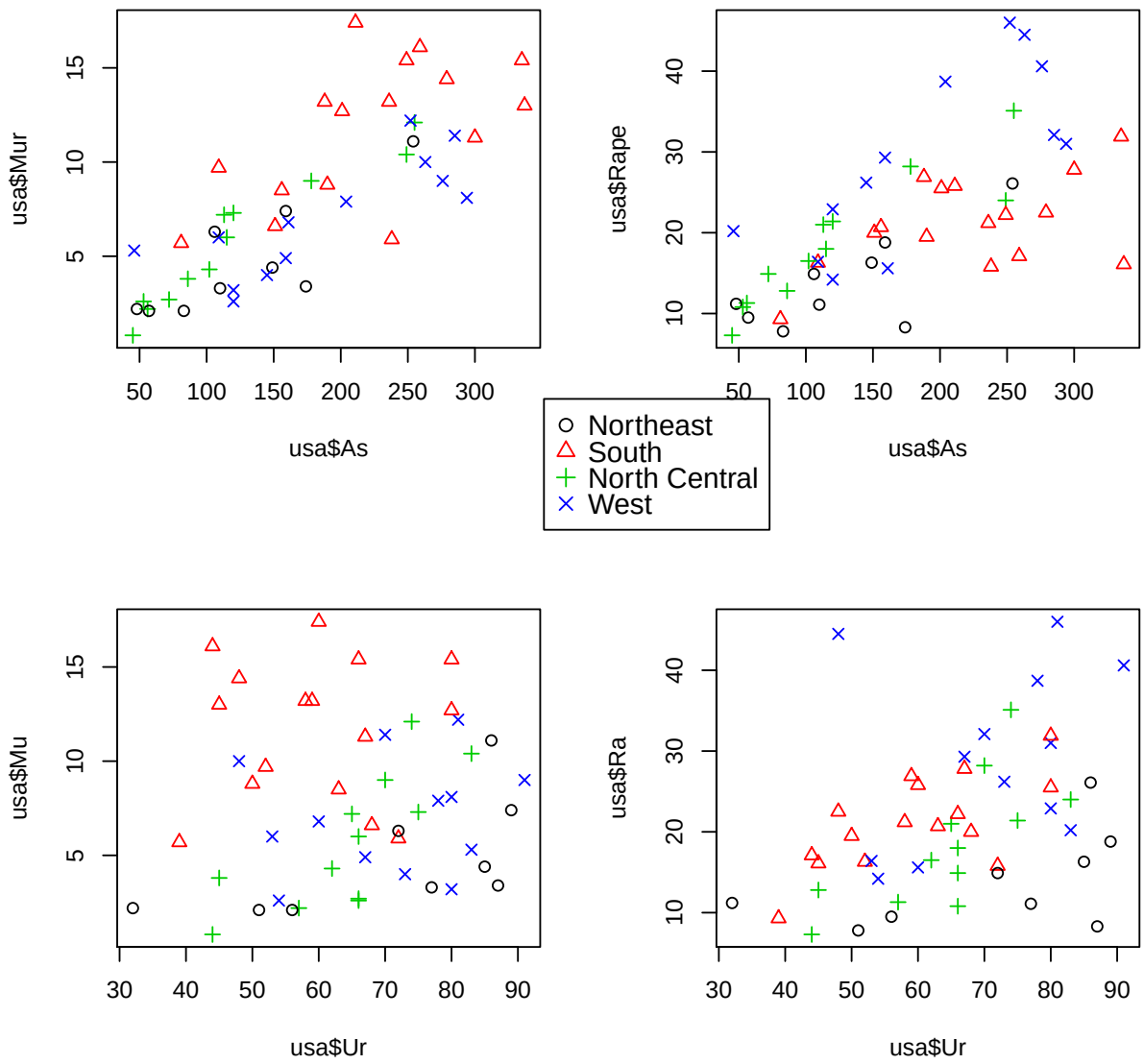
```
## 280     3                     honda accord lx  68
## 294     3                    vw rabbit custom  71
## 295     3                    maxda glc deluxe  65
## 296     3          dodge colt hatchback custom  80
## 298     3                   mercedes benz 300d  77
## 300     3                         peugeot 504  71
## 302     3                    plymouth horizon  70
## 303     3                 plymouth horizon tc3  70
## 304     3                          datsun 210  65
## 305     3                   fiat strada custom  69
## 310     3                           vw rabbit  76
## 311     3                toyota corolla tercel  60
## 312     3                   chevrolet chevette  70
## 313     3                          datsun 310  65
## 316     3                         amc concord  90
## 318     3                           audi 4000  78
## 320     3                           mazda 626  75
## 321     3                 datsun 510 hatchback  92
## 322     3                       toyota corolla  75
## 323     3                           mazda glc  65
## 325     3                          datsun 210  65
## 326     3                 vw rabbit c (diesel)  48
## 327     3                   vw dasher (diesel)  48
## 328     3                  audi 5000s (diesel)  67
## 329     3                   mercedes-benz 240d  67
## 330     3                  honda civic 1500 gl  67
## 332     3                           subaru dl  67
## 333     3                    vokswagen rabbit  62
## 336     3                   triumph tr7 coupe  88
## 338     3                        honda accord  72
## 344     3                      toyota starlet  58
## 345     3                      plymouth champ  64
## 346     3                    honda civic 1300  60
## 347     3                              subaru  67
## 348     3                      datsun 210 mpg  65
## 349     3                       toyota tercel  62
## 350     3                         mazda glc 4  68
## 351     3                   plymouth horizon 4  63
## 352     3                      ford escort 4w  65
## 353     3                      ford escort 2h  65
## 354     3                    volkswagen jetta  74
## 356     3                       honda prelude  75
## 357     3                      toyota corolla  75
## 359     3                           mazda 626  74
## 360     3            peugeot 505s turbo diesel  80
## 361     3                         volvo diesel  76
## 368     3                   chevrolet cavalier  88
## 369     3             chevrolet cavalier wagon  88
```

```
## 370     3            chevrolet cavalier 2-door  88
## 376     3                  volkswagen rabbit l  74
## 377     3                   mazda glc custom l  68
## 378     3                     mazda glc custom  68
## 379     3               plymouth horizon miser  63
## 380     3                       mercury lynx l  70
## 383     3                       toyota corolla  70
## 384     3                          honda civic  67
## 385     3                   honda civic (auto)  67
## 386     3                        datsun 310 gx  67
## 388     3    oldsmobile cutlass ciera (diesel)  85
## 395     3                             vw pickup  52
## 397     3                          ford ranger  79
## 398     3                           chevy s-10  82
## 135     4                          amc matador 110
## 155     4                      mercury monarch  72
## 156     4                        ford maverick  72
## 161     4                        buick century 110
## 162     4            chevroelt chevelle malibu 105
## 163     4                          amc matador 110
## 164     4                        plymouth fury  95
## 201     4                     ford granada ghia  78
## 223     4           oldsmobile cutlass supreme 110
## 250     4    oldsmobile cutlass salon brougham 110
## 286     4            chevrolet caprice classic 130
## 289     4                      dodge st. regis 135
## 292     4        chevrolet malibu classic (sw) 125
## 299     4                    cadillac eldorado 125
## 301     4    oldsmobile cutlass salon brougham  90
## 365     4                oldsmobile cutlass ls 105
```

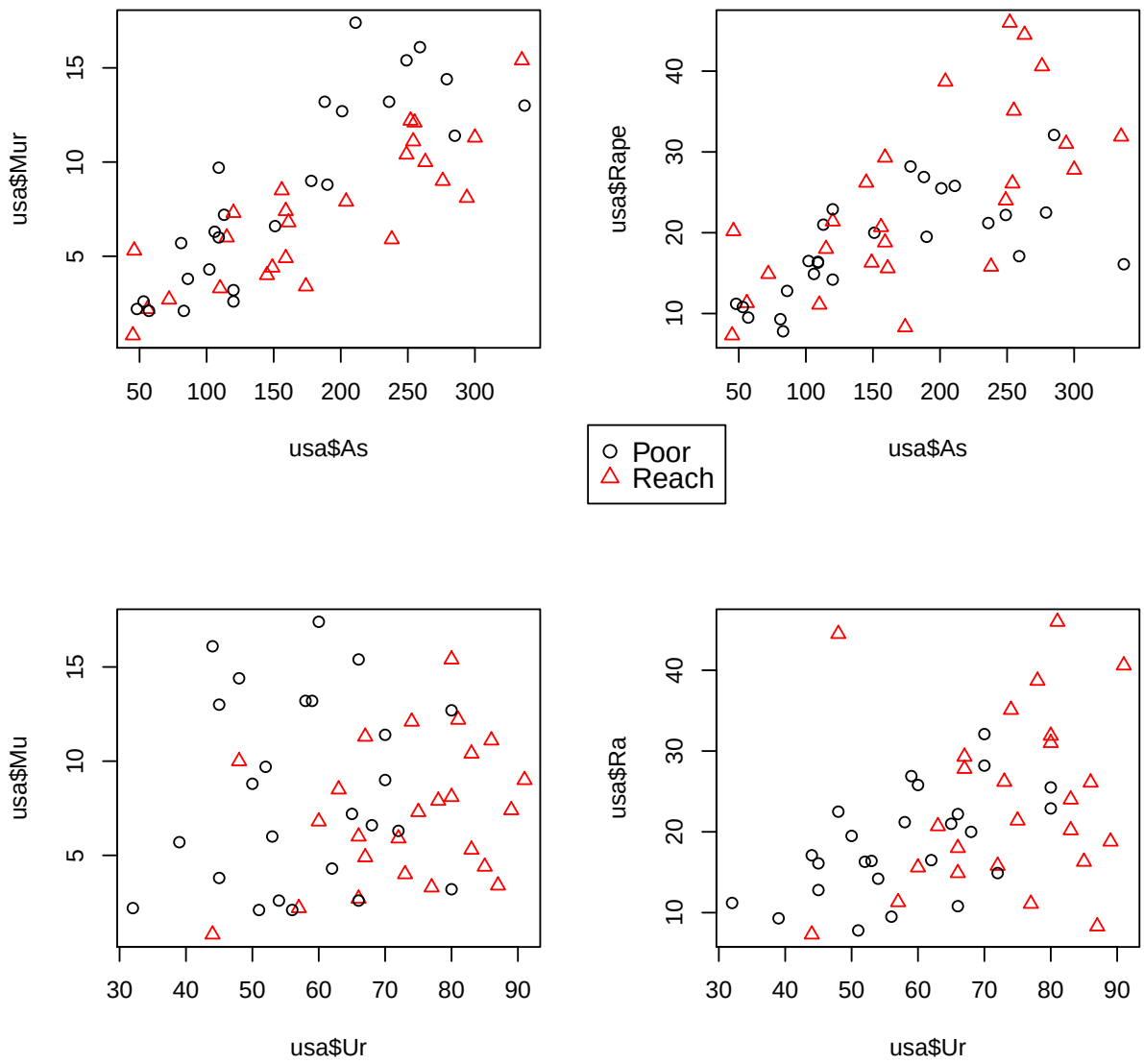
```r
usa <- cbind(state.division, state.region,
    state.x77[, "Income"], USArrests)
names(usa)[1:3] <- c("Division", "Region",
    "Income")
require(graphics)
pairs(USArrests, panel = panel.smooth, main = "USArrests data")
```



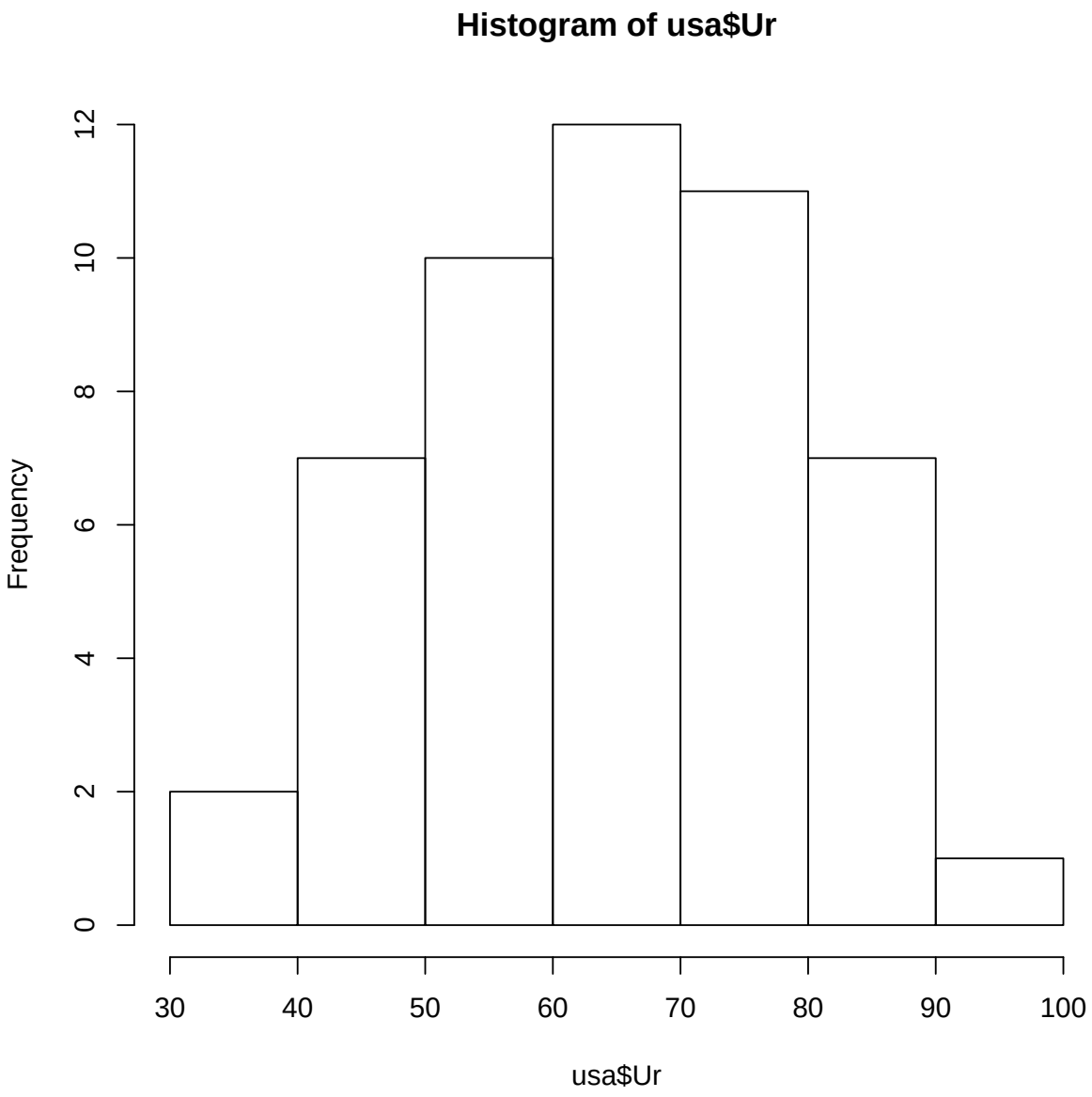

```
mInc <- median(usa$Income)
usa$IncF <- "Poor"
usa$IncF[usa$Income > mInc] <- "Reach"
usa$IncF <- factor(usa$IncF)
par(mfrow = c(2, 2))
plot(usa$As, usa$Mur, col = usa$Reg, pch = as.numeric(usa$Reg))
# identify(usa$As, usa$Mur);
plot(usa$As, usa$Rape, col = usa$Reg, pch = as.numeric(usa$Reg))
plot(usa$Ur, usa$Mu, col = usa$Reg, pch = as.numeric(usa$Reg))
plot(usa$Ur, usa$Ra, col = usa$Reg, pch = as.numeric(usa$Reg))
par(mfrow = c(1, 1))
legend("center", levels(usa$Reg), col = 1:nlevels(usa$Reg),
    pch = 1:nlevels(usa$Reg))
```



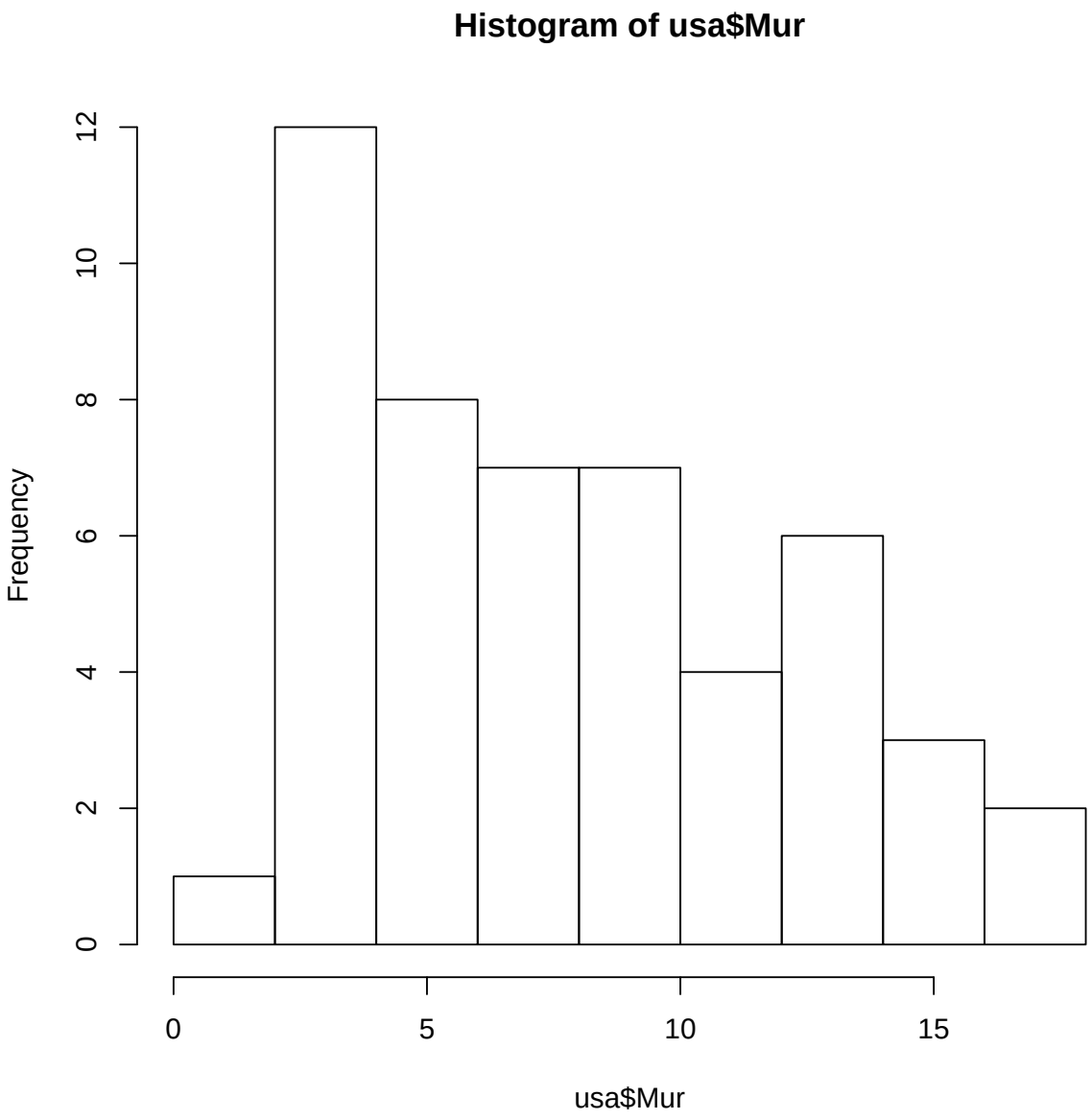

```
par(mfrow = c(2, 2))
plot(usa$As, usa$Mur, col = usa$IncF, pch = as.numeric(usa$IncF))
# identify(usa$As, usa$Mur);
plot(usa$As, usa$Rape, col = usa$IncF, pch = as.numeric(usa$IncF))
plot(usa$Ur, usa$Mu, col = usa$IncF, pch = as.numeric(usa$IncF))
plot(usa$Ur, usa$Ra, col = usa$IncF, pch = as.numeric(usa$IncF))
par(mfrow = c(1, 1))
legend("center", levels(usa$IncF), col = 1:nlevels(usa$IncF),
    pch = 1:nlevels(usa$IncF))
```


usa$Mur
usa$As
usa$Rape
usa$As
Poor
Reach
usa$Mu
usa$Ur
usa$Ra
usa$Ur

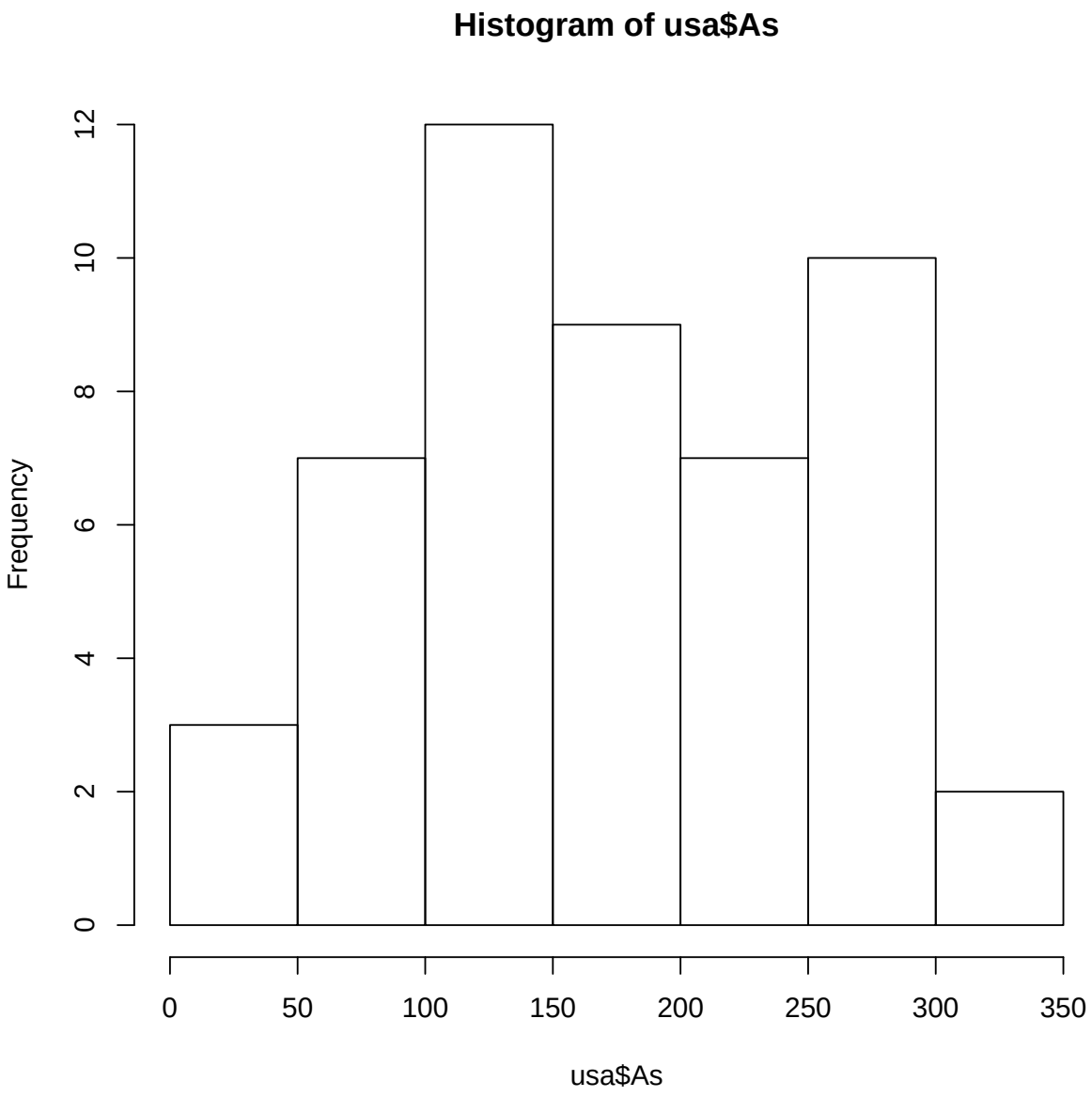
```
hist(usa$Ur)
```


Histogram of usa$Ur
Frequency
0
2
4
6
8
10
12
30
40
50
60
70
80
90
100
usa$Ur

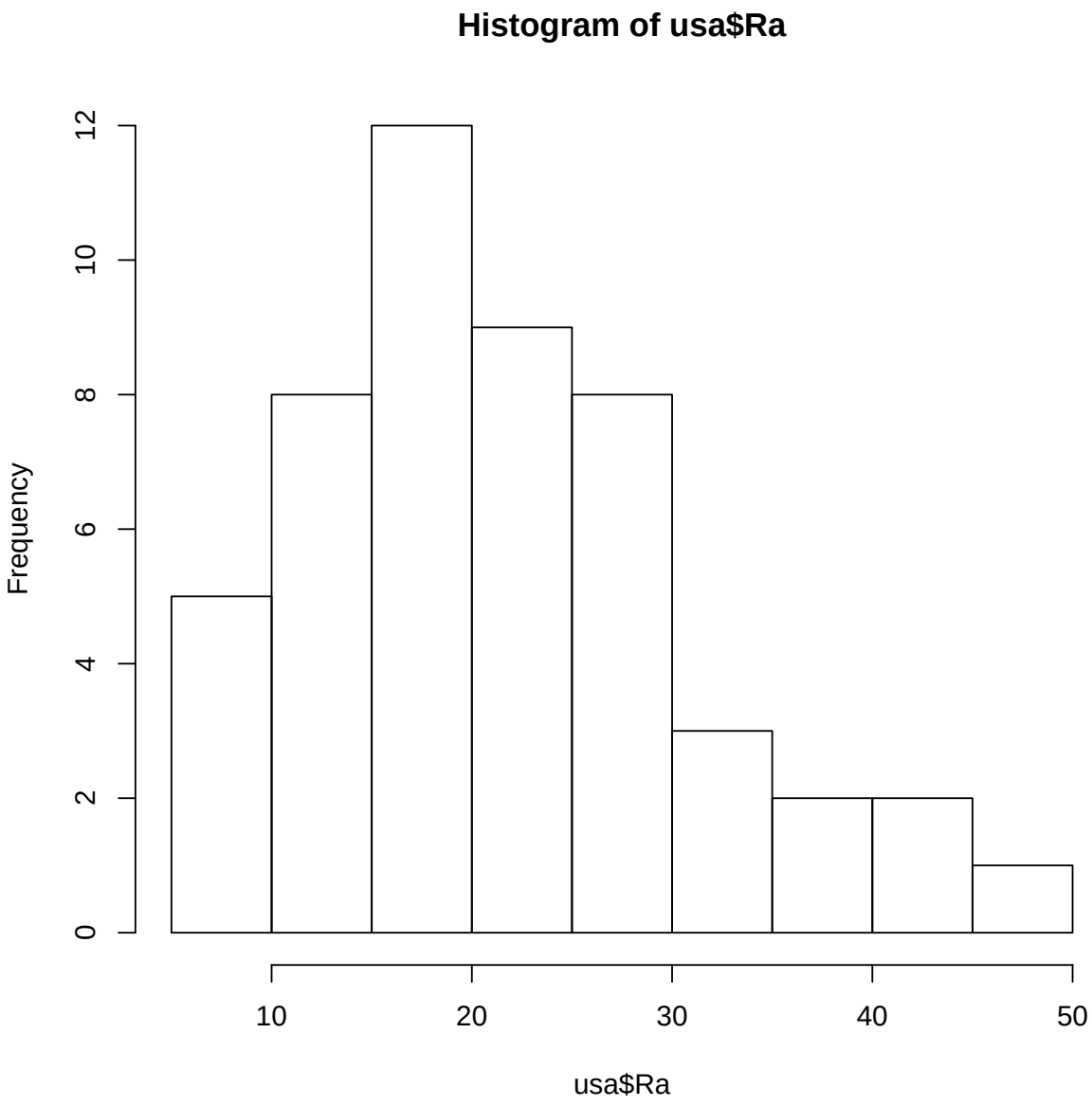
```
hist(usa$Mur)
```



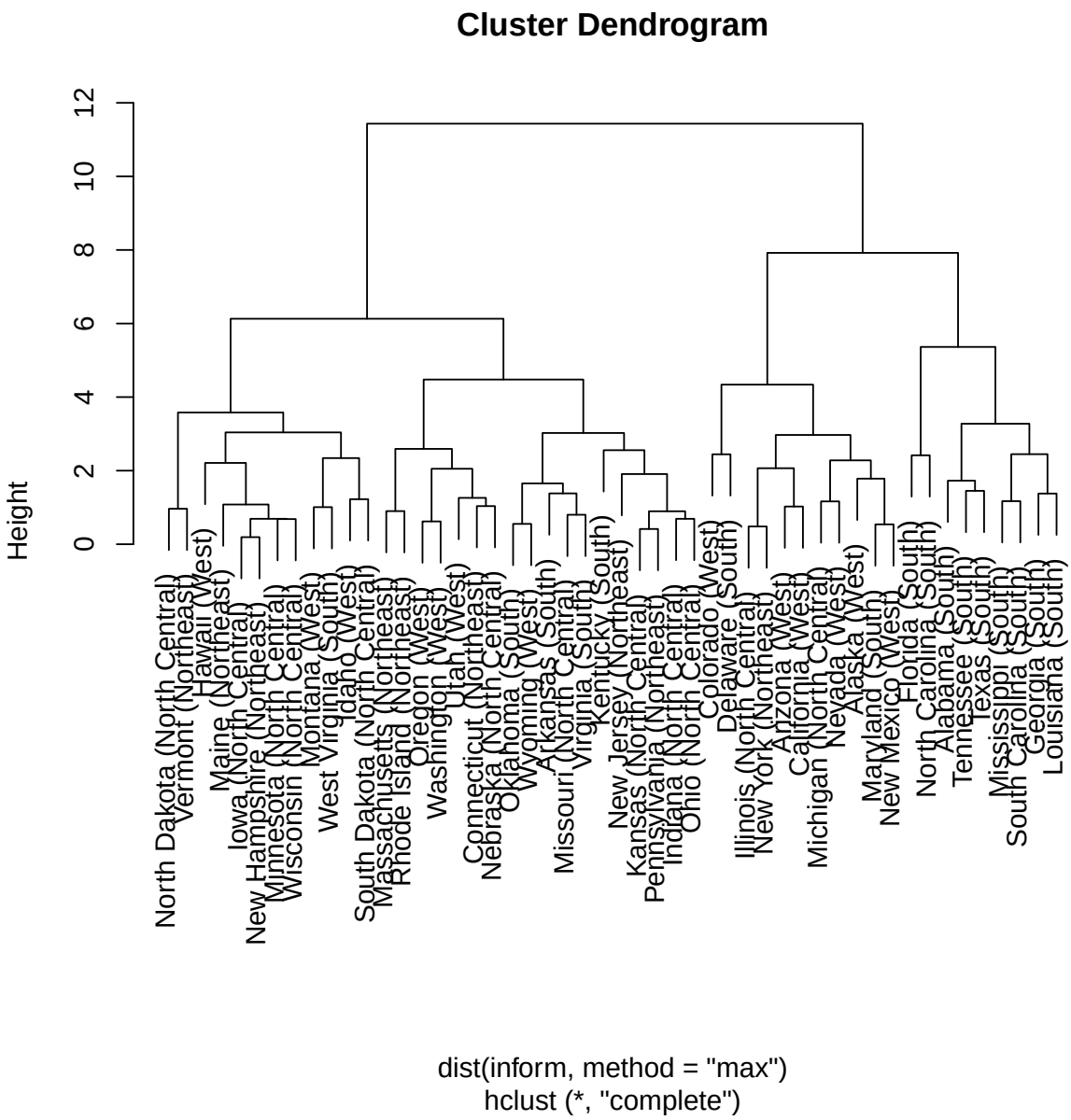

```
hist(usa$As)
```


Histogram of usa$As
Frequency
0
2
4
6
8
10
12
0
50
100
150
200
250
300
350
usa$As

```
hist(usa$Ra)
```




```
usa$Murder <- 3 * scale(usa$Murder)
usa$Assault <- 3 * scale(usa$Assault)
usa$Rape <- scale(usa$Rape)
usa$UrbanPop <- scale(usa$UrbanPop)
inform <- subset(usa, select = c(Murder,
    Assault, Rape, UrbanPop))
hc <- hclust(dist(inform, method = "max"),
    method = "com")
plot(hc, labels = paste(row.names(usa), " (",
    usa$Region, ")", sep = ""))
```



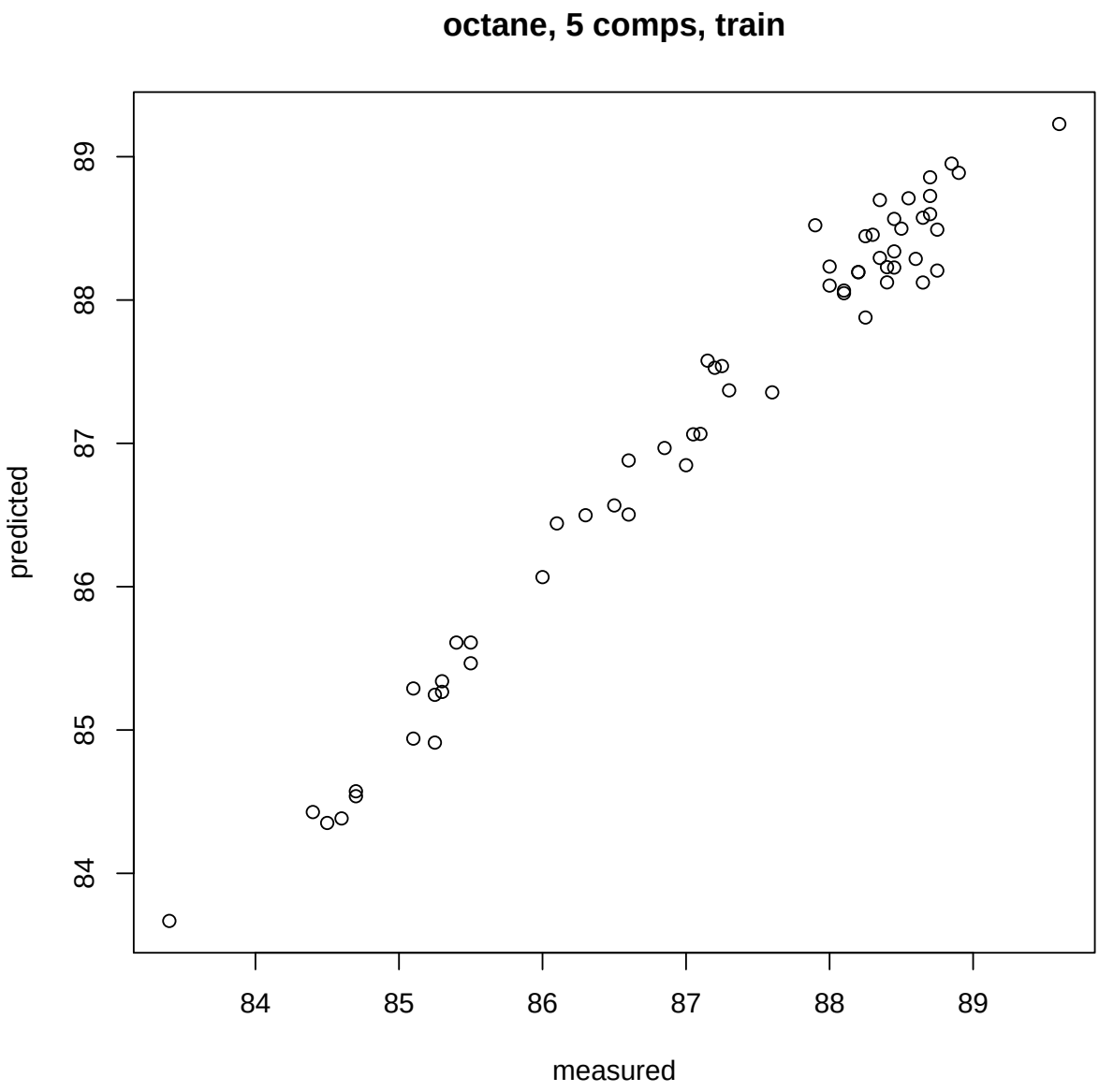

```
# plot(hc, labels=paste(row.names(usa), '
# (', usa$Income, ')', sep='')); 
```

9 Материалы с занятия 21 ноября

```
read_chunk("regularization/regularization.R")
```

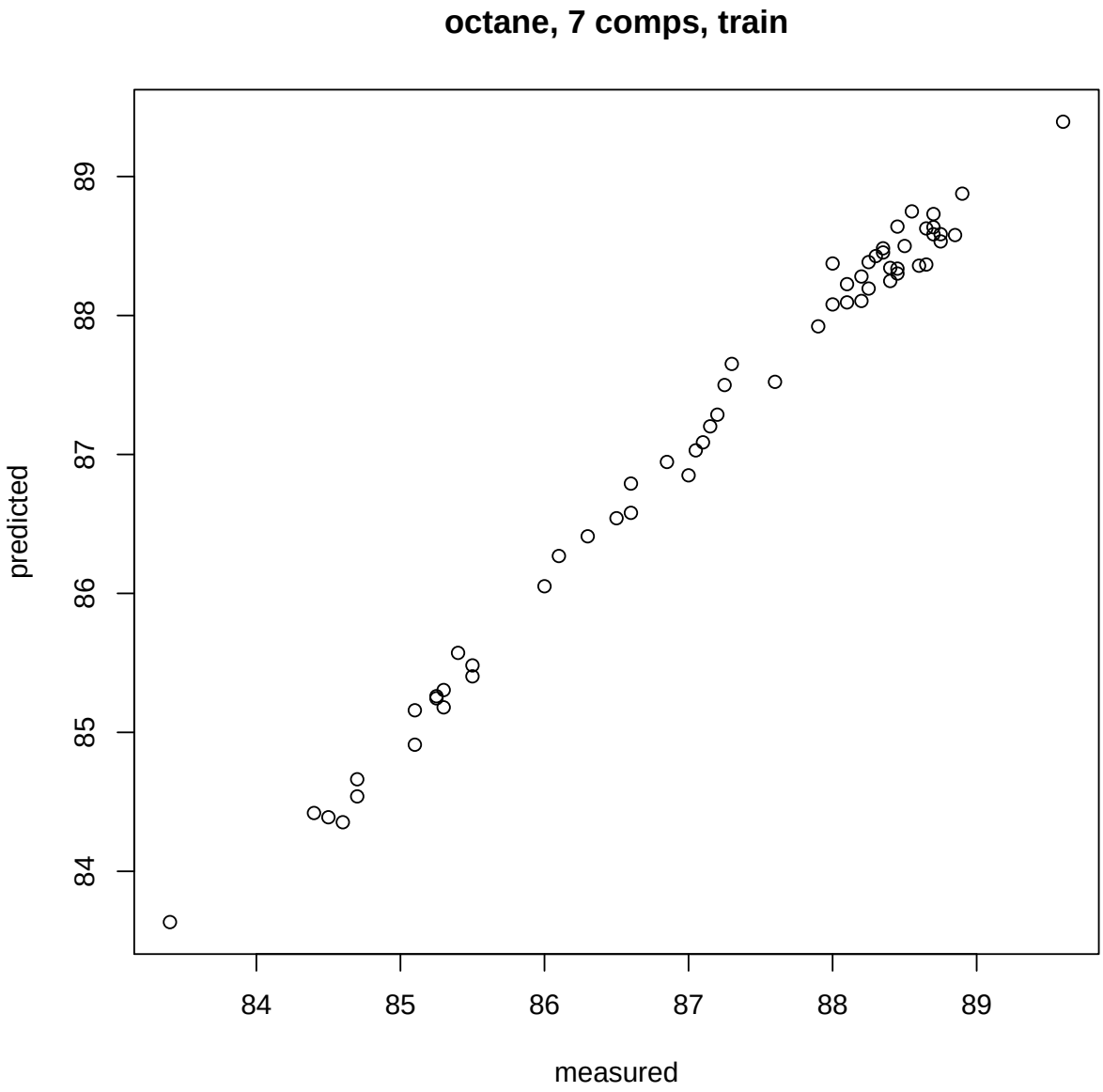
```
library(glmnet)
library(leaps)
# http://rstudio-pubs-static.s3.amazonaws.com/2897_9220b21cfc0c43a396ff9abf122bb351
# library(bestglm) library(meifly)
```

```r
library(pls)  # pls, pcr
library(lattice)
library(latticeExtra)
library(MASS)
library(e1071)
data(gasoline)
names(gasoline)

## [1] "octane" "NIR"

dim(gasoline)

## [1] 60  2

class(gasoline$NIR) <- NULL
colnames(gasoline$NIR) <- paste("S", seq(900,
    1700, 2), sep = "")
gasoline <- cbind(subset(gasoline, select = octane),
    as.data.frame(gasoline$NIR))
dim(gasoline)

## [1]  60 402

# Many curses to package developers!
my.predict.mvr <- function(object, newdata,
    ...) {
    as.vector(predict(object, newdata, type = "response",
        ncomp = object$ncomp, ...))
}
tn.pcr <- tune(pcr, octane ~ ., data = gasoline,
    ranges = list(ncomp = 1:10), predict.func = my.predict.mvr,
    tunecontrol = tune.control(sampling = "cross"))
print(tn.pcr)

##
## Parameter tuning of 'pcr':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  ncomp
##      5
##
## - best performance: 0.05946477

plot(tn.pcr$best.model)
```




```
tn.plsr <- tune(plsr, octane ~ ., data = gasoline,
    ranges = list(ncomp = 1:10), predict.func = my.predict.mvr,
    tunecontrol = tune.control(sampling = "cross"))
print(tn.plsr)

##
## Parameter tuning of 'plsr':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  ncomp
##      7
##
## - best performance: 0.05732338
```

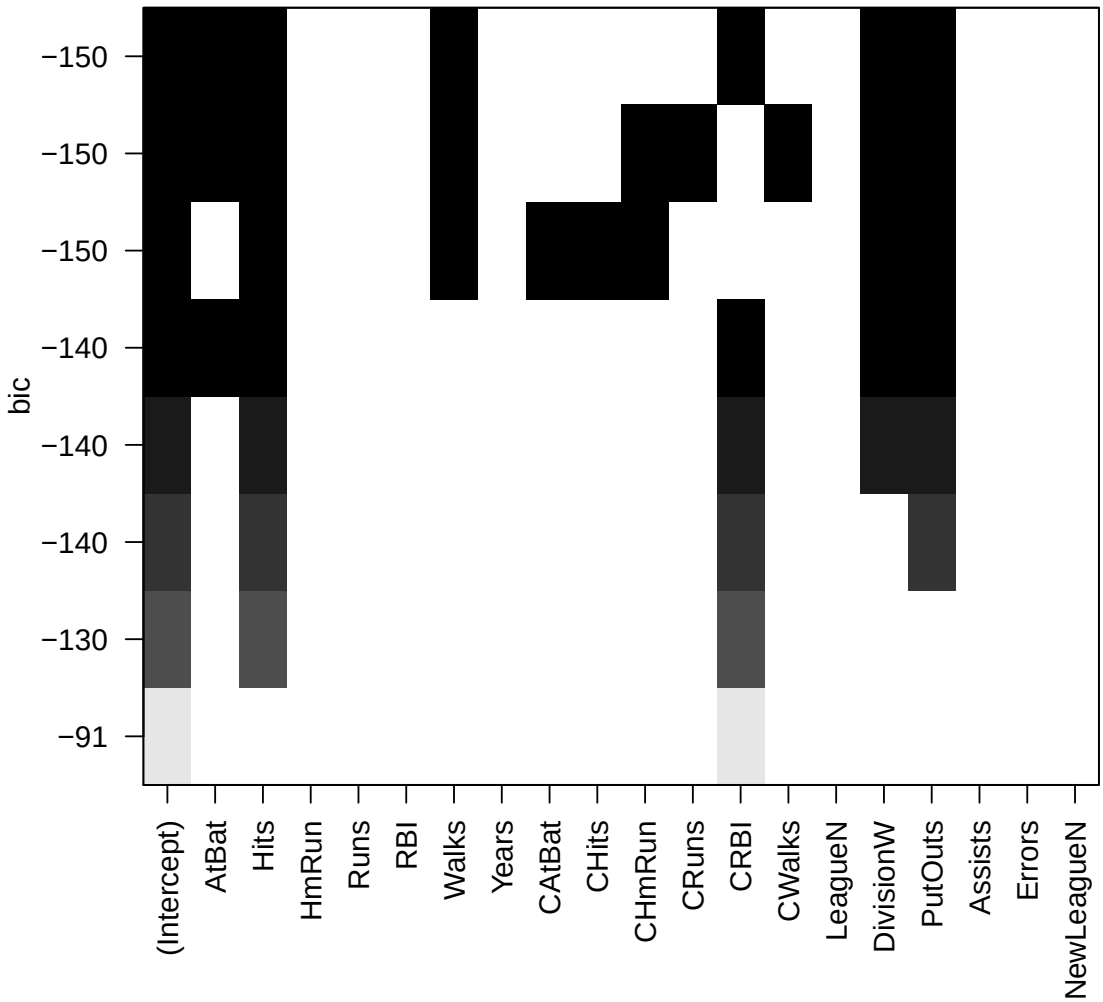
```r
plot(tn.plsr$best.model)
```



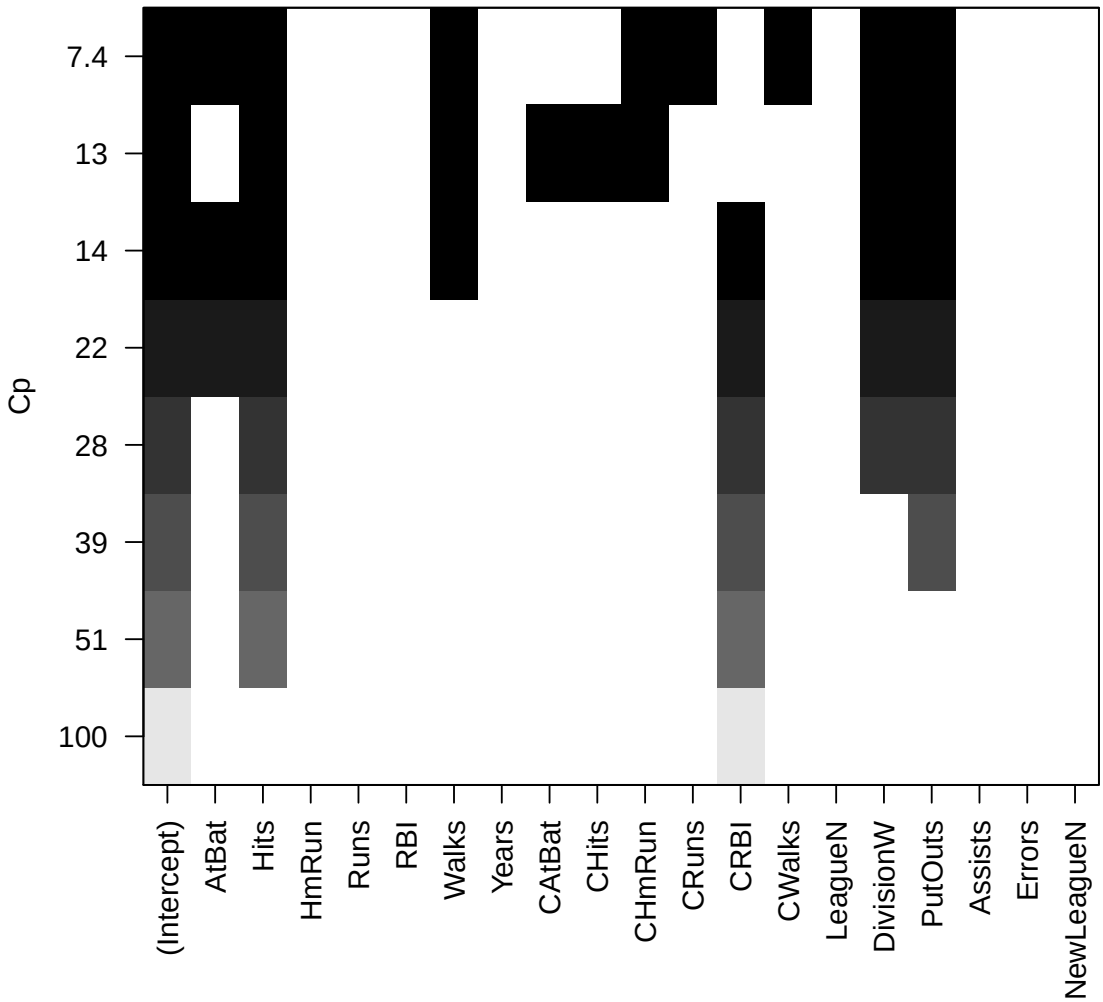

```r
library(ISLR)
data(Hitters, package = "ISLR")
names(Hitters)

##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"
##  [5] "RBI"       "Walks"     "Years"     "CAtBat"
##  [9] "CHits"     "CHmRun"    "CRuns"     "CRBI"
## [13] "CWalks"    "League"    "Division"  "PutOuts"
## [17] "Assists"   "Errors"    "Salary"    "NewLeague"

sum(is.na(Hitters$Salary))

## [1] 59
```

```r
Hitters <- na.omit(Hitters)
dim(Hitters)

## [1] 263  20

sum(is.na(Hitters))

## [1] 0

rl <- regsubsets(Salary ~ ., data = Hitters)
summary(rl)

## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = Hitters)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "
##          CHmRun CRuns CRBI CWalks LeagueN DivisionW PutOuts
## 1  ( 1 ) " "    " "   "*"  " "    " "     " "       " "
## 2  ( 1 ) " "    " "   "*"  " "    " "     " "       " "
```

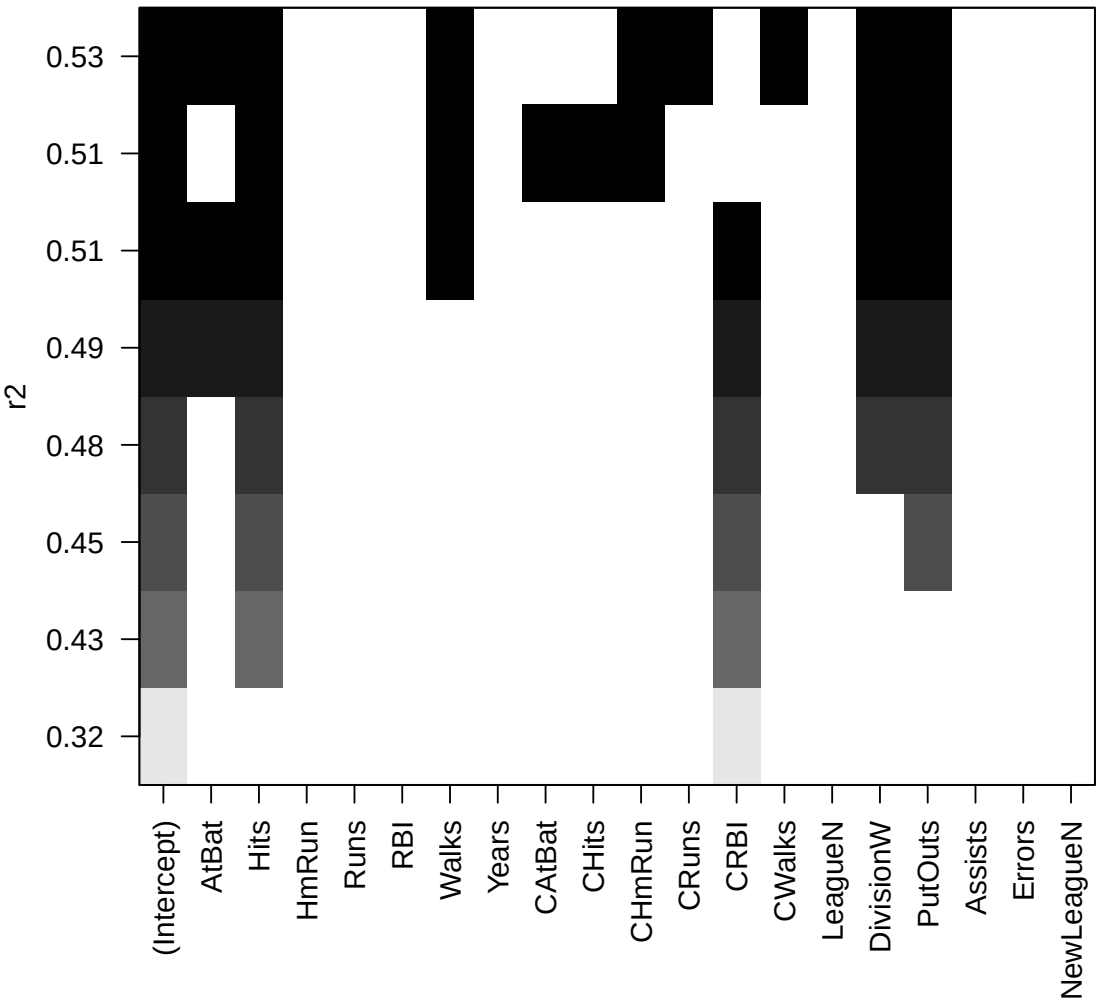
```
## 3  ( 1 ) " "    " "   "*"  " "    " "     " "       "*"
## 4  ( 1 ) " "    " "   "*"  " "    " "     "*"       "*"
## 5  ( 1 ) " "    " "   "*"  " "    " "     "*"       "*"
## 6  ( 1 ) " "    " "   "*"  " "    " "     "*"       "*"
## 7  ( 1 ) "*"    " "   " "  " "    " "     "*"       "*"
## 8  ( 1 ) "*"    "*"   " "  "*"    " "     "*"       "*"
##          Assists Errors NewLeagueN
## 1  ( 1 ) " "     " "    " "
## 2  ( 1 ) " "     " "    " "
## 3  ( 1 ) " "     " "    " "
## 4  ( 1 ) " "     " "    " "
## 5  ( 1 ) " "     " "    " "
## 6  ( 1 ) " "     " "    " "
## 7  ( 1 ) " "     " "    " "
## 8  ( 1 ) " "     " "    " "

plot(rl)
```



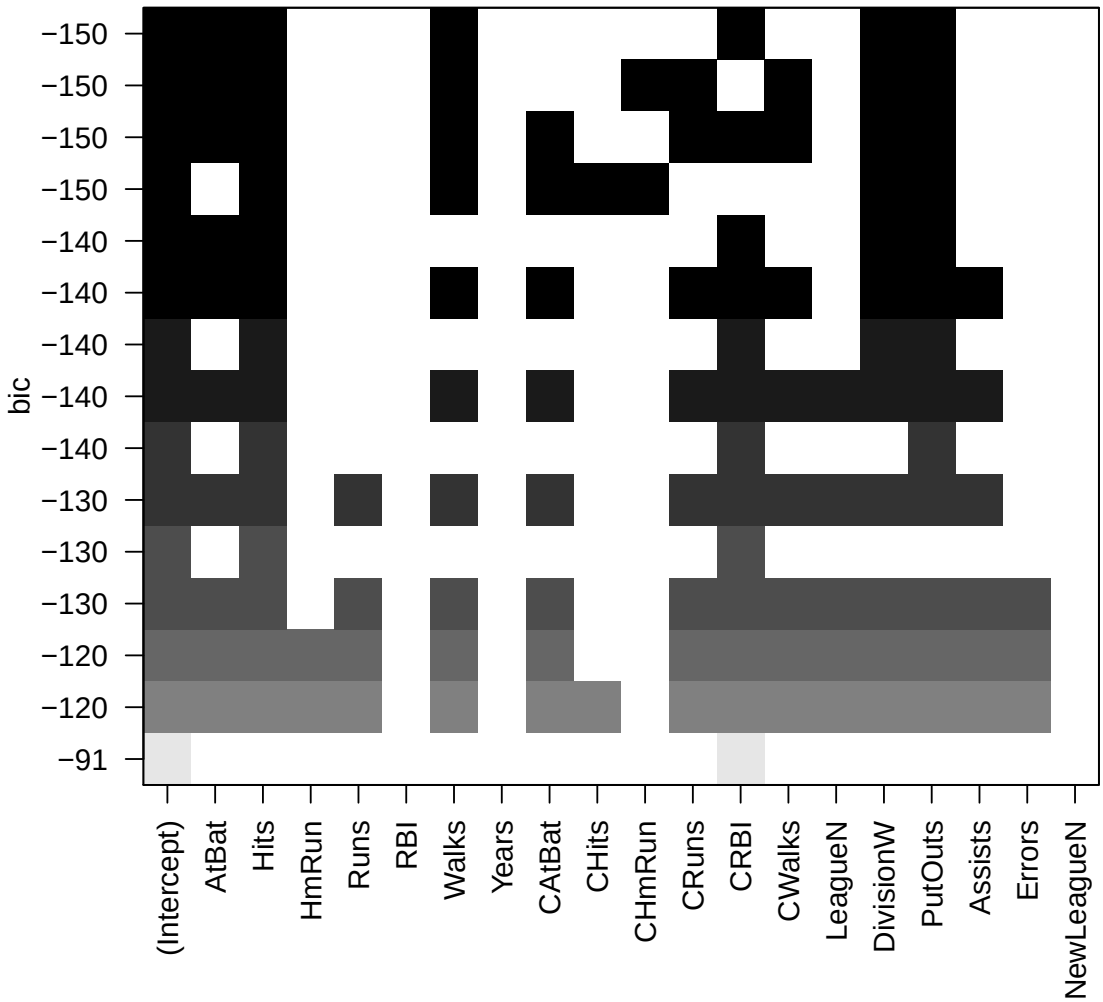

```r
plot(rl, scale = "Cp")
```



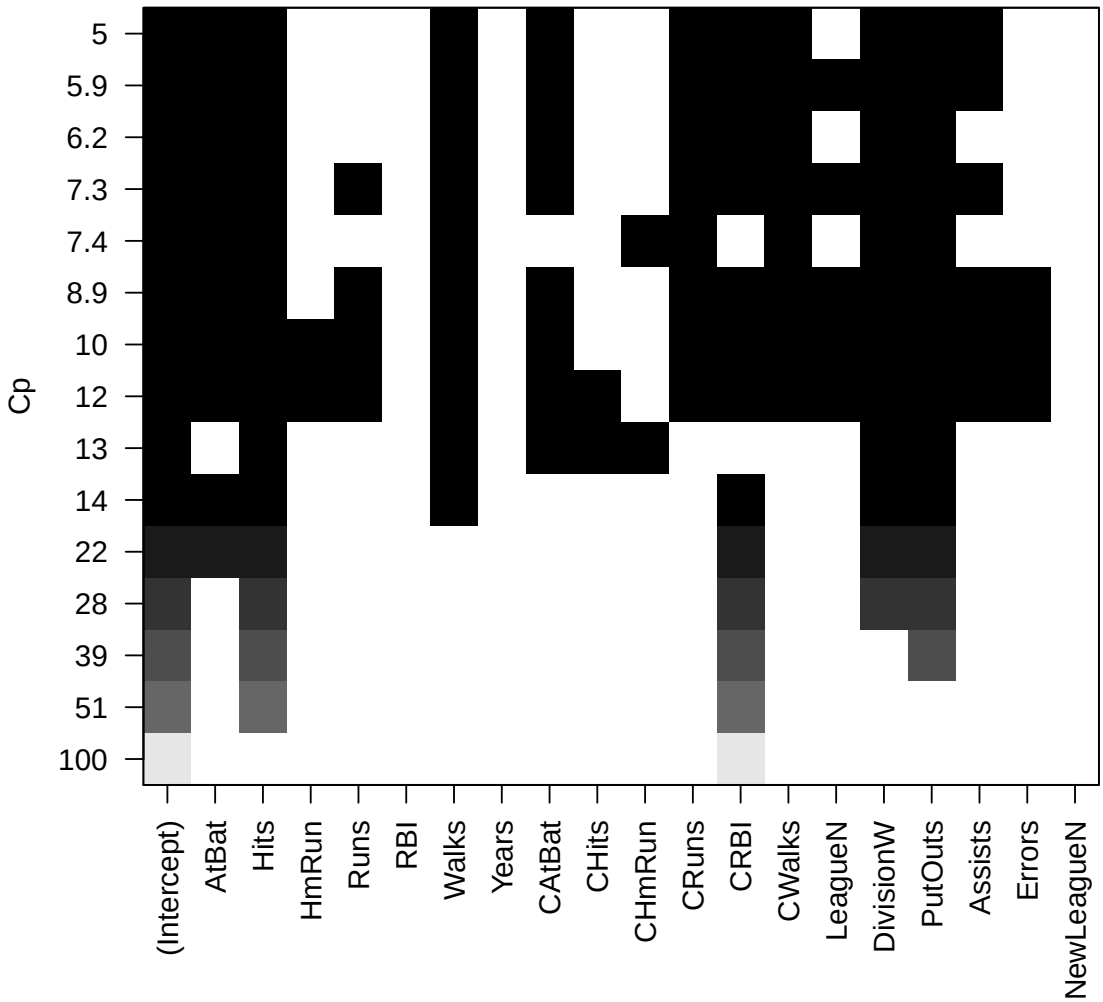

```
plot(rl, scale = "r2")
```




```
rl.summary <- summary(rl)
names(rl.summary)

## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"
## [7] "outmat" "obj"

rl <- regsubsets(Salary ~ ., data = Hitters,
    nvmax = 15)
plot(rl)
```

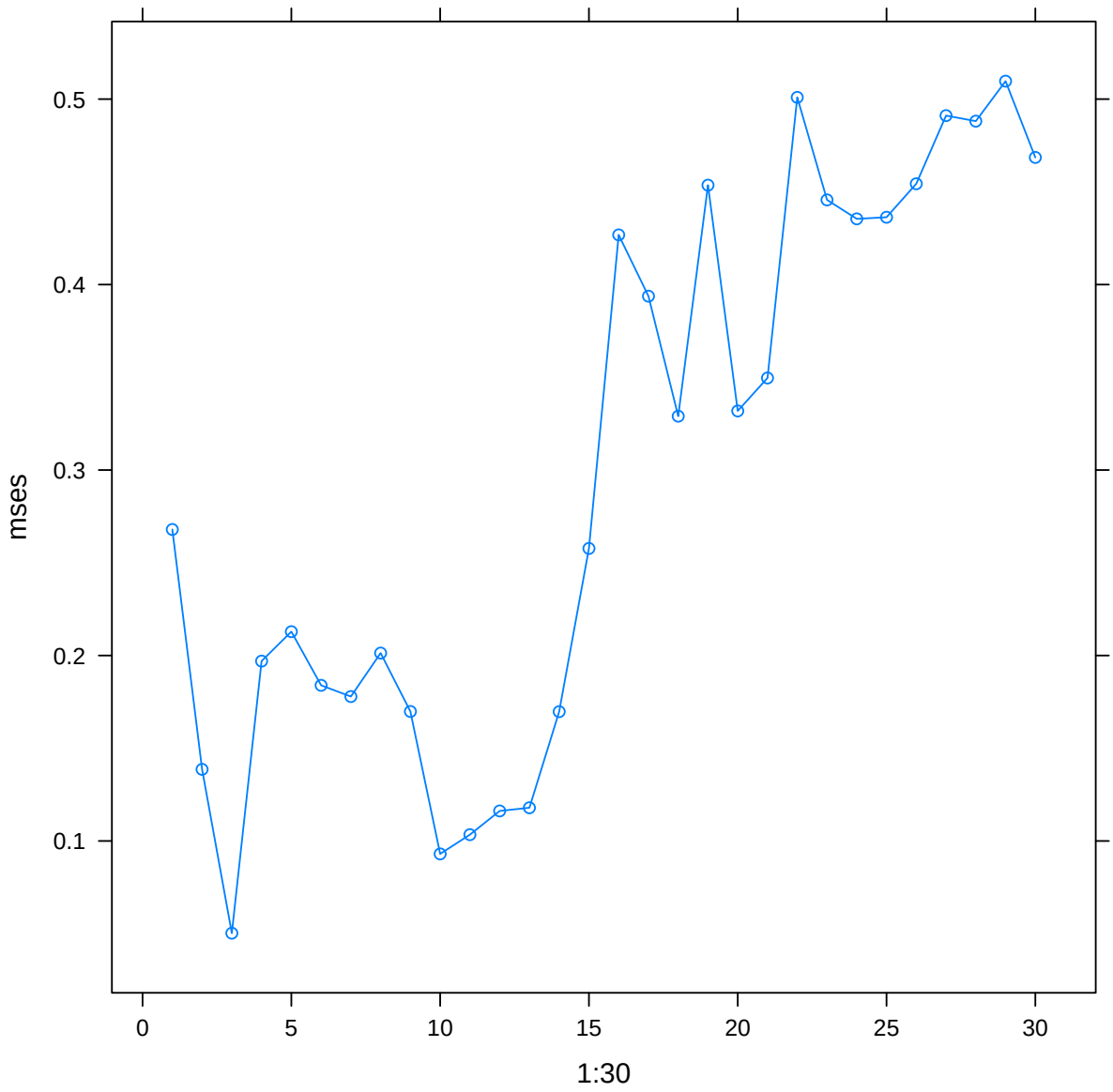



```
plot(rl, scale = "Cp")
```




```
coef(rl, 10)

##  (Intercept)        AtBat         Hits        Walks
##  162.5354420   -2.1686501    6.9180175    5.7732246
##       CAtBat        CRuns         CRBI       CWalks
##   -0.1300798    1.4082490    0.7743122   -0.8308264
##    DivisionW      PutOuts      Assists
## -112.3800575    0.2973726    0.2831680

predict.regsubsets <- function(object, newdata,
    id, ...) {
    form <- as.formula(object$call[[2]])
    mat <- model.matrix(form, newdata)
    coefi <- coef(object, id = id)
    xvars <- names(coefi)
```

```r
    mat[, xvars, drop = FALSE] %*% coefi
}
train.idx <- sample(nrow(gasoline), size = 0.66 *
    nrow(gasoline))
gasoline.train <- gasoline[train.idx, ]
gasoline.test <- gasoline[-train.idx, ]
rl <- regsubsets(octane ~ ., method = "forward",
    data = gasoline.train, nvmax = 30)

## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax, force.in
= force.in, : 363  linear dependencies found

mses <- sapply(1:30, function(id) {
    pred <- predict(rl, gasoline.test, id = id)
    mean((pred - gasoline.test$octane)^2)
})
xyplot(mses ~ 1:30, type = "b")
```




```
opt.age <- which.min(mses)
coefi.rl <- coefi <- coef(rl, opt.age)[-1]
coefi.rl

##     S1208     S1224     S1374
## -35.25926 -66.99336  66.46622

f <- as.formula(paste("octane", paste0(names(coefi),
    collapse = " + "), sep = " ~ "))
tn.rl <- tune(lm, f, data = gasoline)
print(tn.rl)

##
## Error estimation of 'lm' using 10-fold cross validation: 0.04787449

print(tn.pcr)
```

```
##
## Parameter tuning of 'pcr':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  ncomp
##      5
##
## - best performance: 0.05946477
```

```
print(tn.plsr)
```

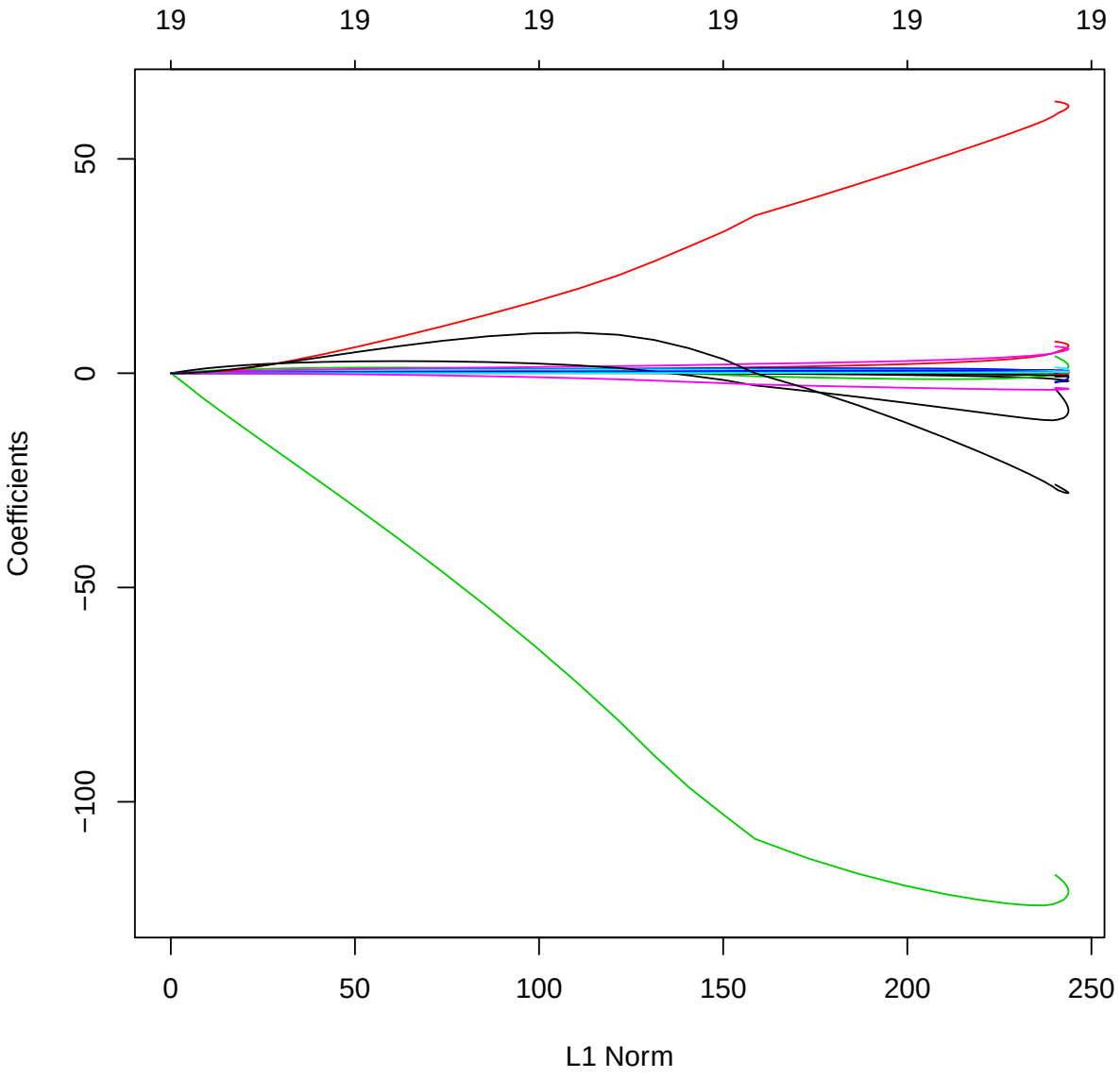
```
##
## Parameter tuning of 'plsr':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  ncomp
##      7
##
## - best performance: 0.05732338
```

```
library(glmnet)
x <- model.matrix(Salary ~ ., data = Hitters)[,
    -1, drop = FALSE]
y <- Hitters$Salary
grid <- 10^seq(10, -2, length.out = 100)
ridge.lm <- glmnet(x, y, alpha = 0, lambda = grid)
plot(ridge.lm)
predict(ridge.lm, s = 50, type = "coefficients")[1:20,
    ]
```

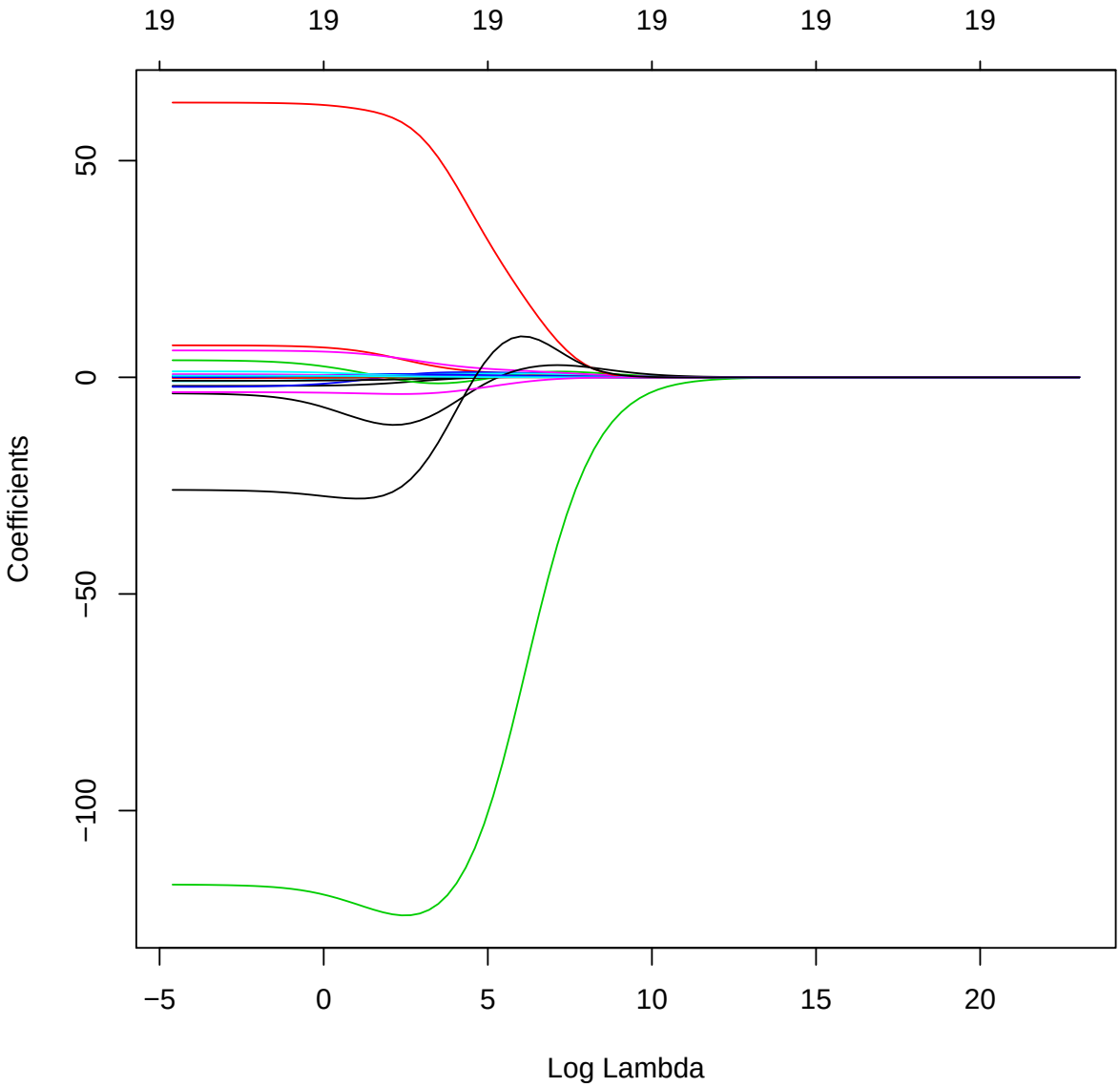
```
##   (Intercept)         AtBat          Hits         HmRun
##  4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00
##          Runs           RBI         Walks         Years
##  1.145892e+00  8.038292e-01  2.716186e+00 -6.218319e+00
##        CAtBat         CHits        CHmRun         CRuns
##  5.447837e-03  1.064895e-01  6.244860e-01  2.214985e-01
##          CRBI        CWalks       LeagueN     DivisionW
##  2.186914e-01 -1.500245e-01  4.592589e+01 -1.182011e+02
##       PutOuts       Assists        Errors    NewLeagueN
##  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

```
predict(ridge.lm, s = 50, type = "response")[1:20,
    ]
```

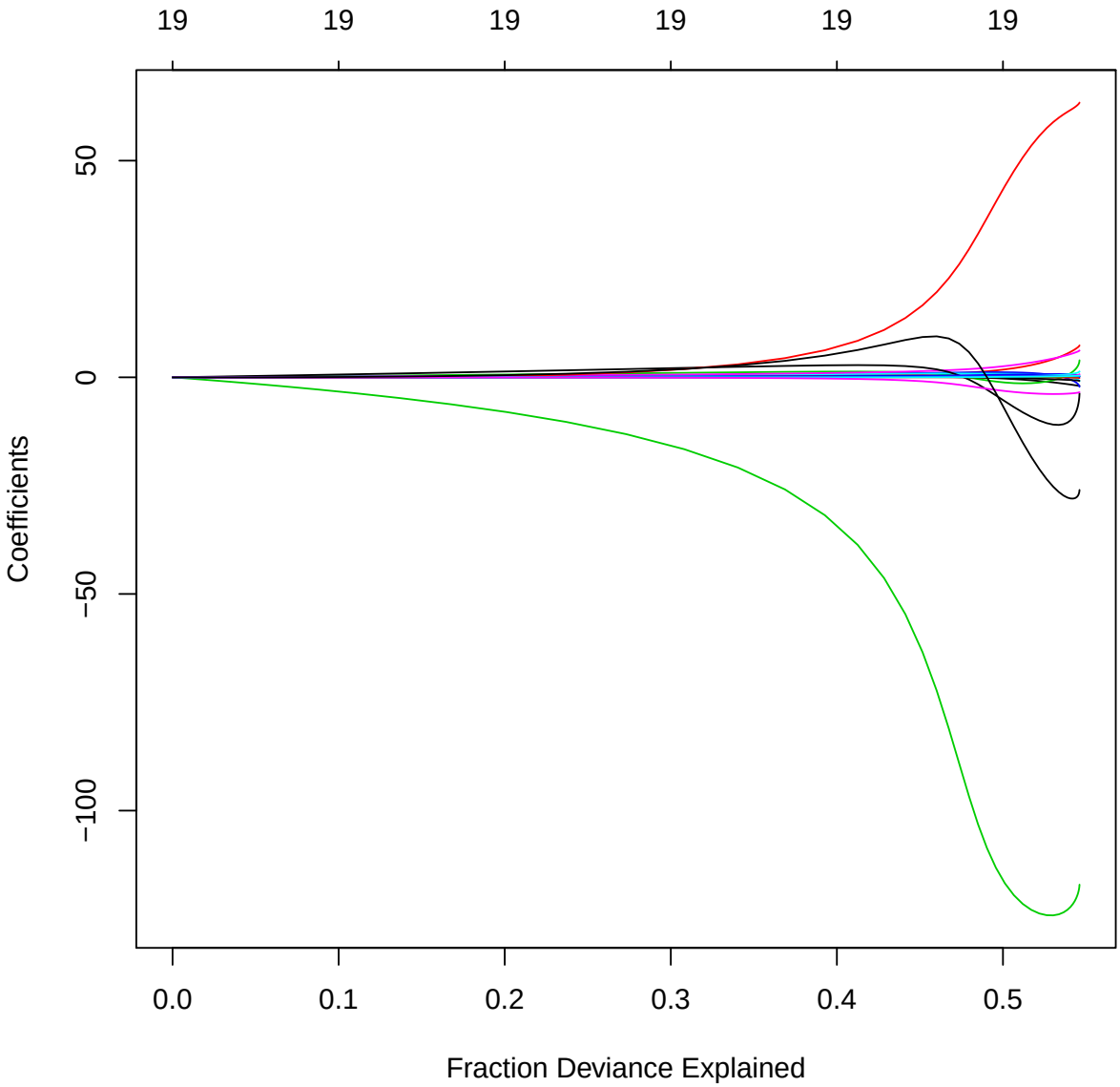
```
## Error in predict.glmnet(ridge.lm, s = 50, type = "response"): You need to supply
a value for 'newx'

my.glmnet <- function(x, data = NULL, subset = NULL,
    ...) {
    if (is.null(data))
        data <- parent.frame()
    mf <- model.matrix(x, data = data)
    mfr <- model.frame(x, data = data)
    response <- as.vector(mfr[, 1])
    if (!is.null(subset)) {
        response <- response[subset]
        mf <- mf[subset, , drop = FALSE]
    }
    predictors <- mf[, -1, drop = FALSE]
    model <- glmnet(predictors, response,
        ...)
    res <- list(model = model, formula = x,
        data = data, terms = attr(mf, "terms"))
    class(res) <- "my.glmnet"
    res
}
predict.my.glmnet <- function(object, newdata = object$data,
    s = object$model$lambda[1], ...) {
    mf <- model.matrix(object$formula, data = newdata)[,
        -1, drop = FALSE]
    .x <- as.matrix(mf)
    predict(object$model, newx = .x, type = "response",
        s = s, ...)
}
plot.my.glmnet <- function(x, ...) plot(x$model,
    ...)
coef.my.glmnet <- function(x, ...) coef(x$model,
    ...)
ridge <- my.glmnet(Salary ~ ., data = Hitters,
    alpha = 0, lambda = grid)
plot(ridge$model)
```



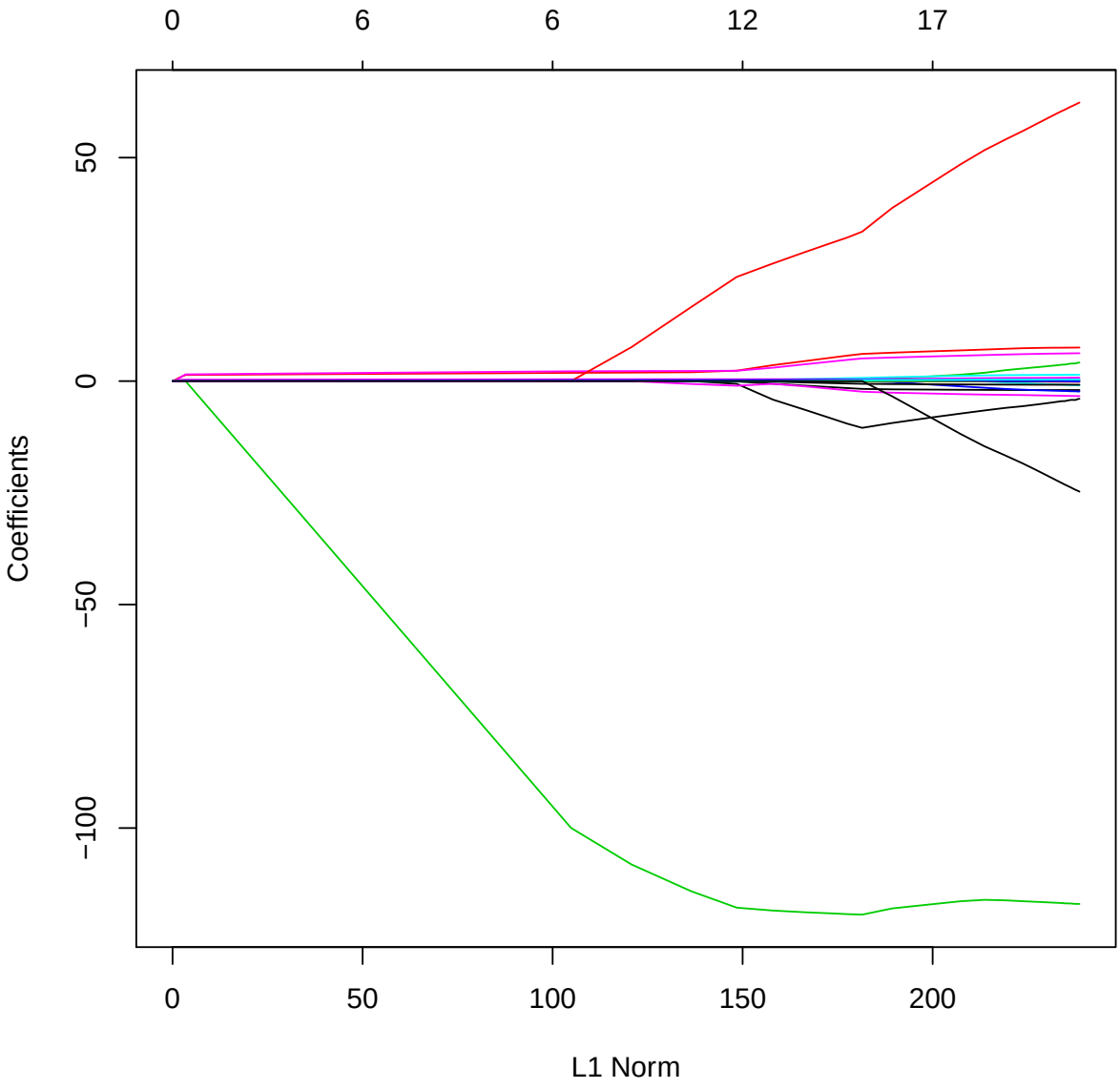

```
plot(ridge$model, xvar = "lambda")
```



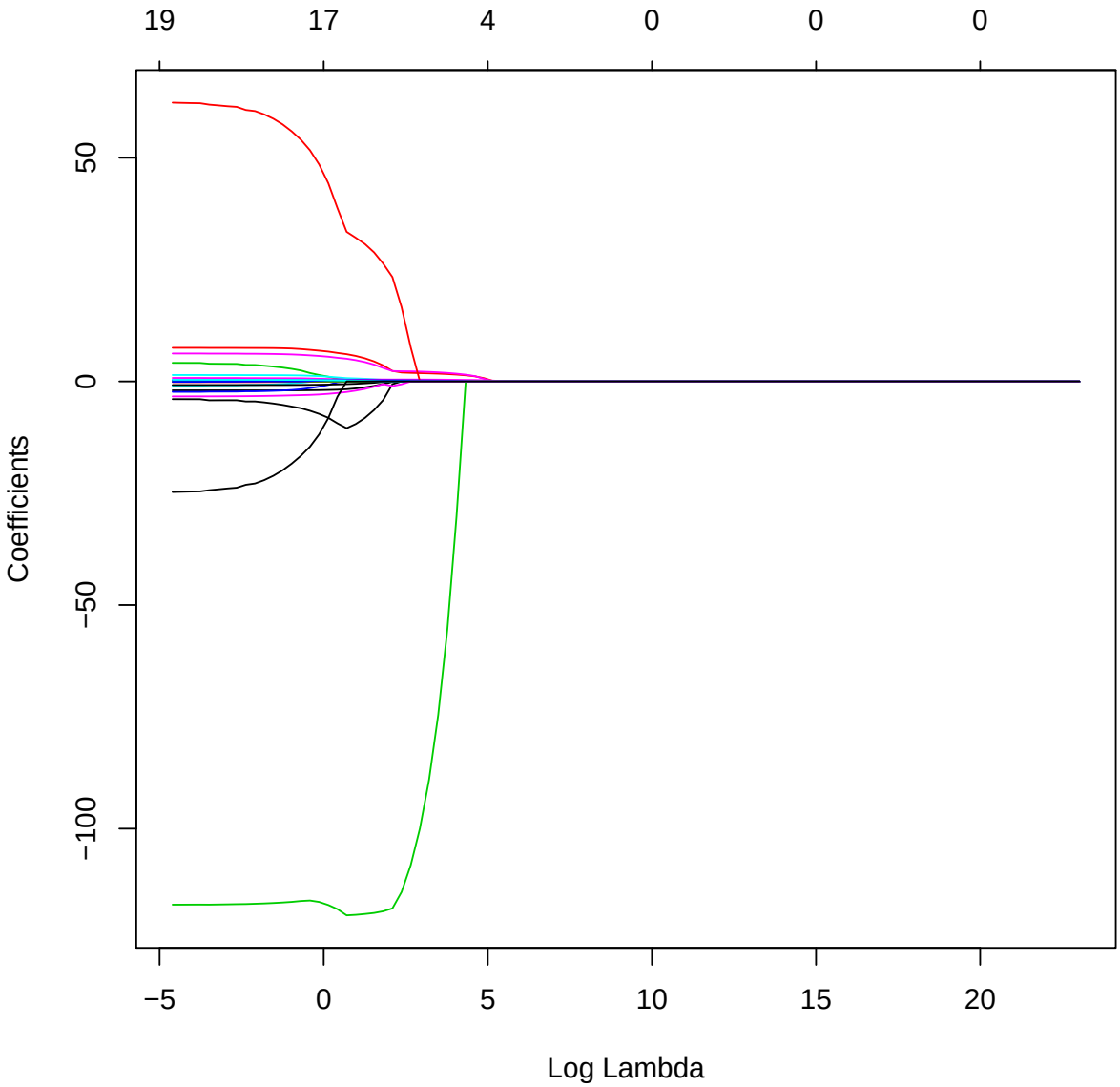

```
plot(ridge$model, xvar = "dev")
```



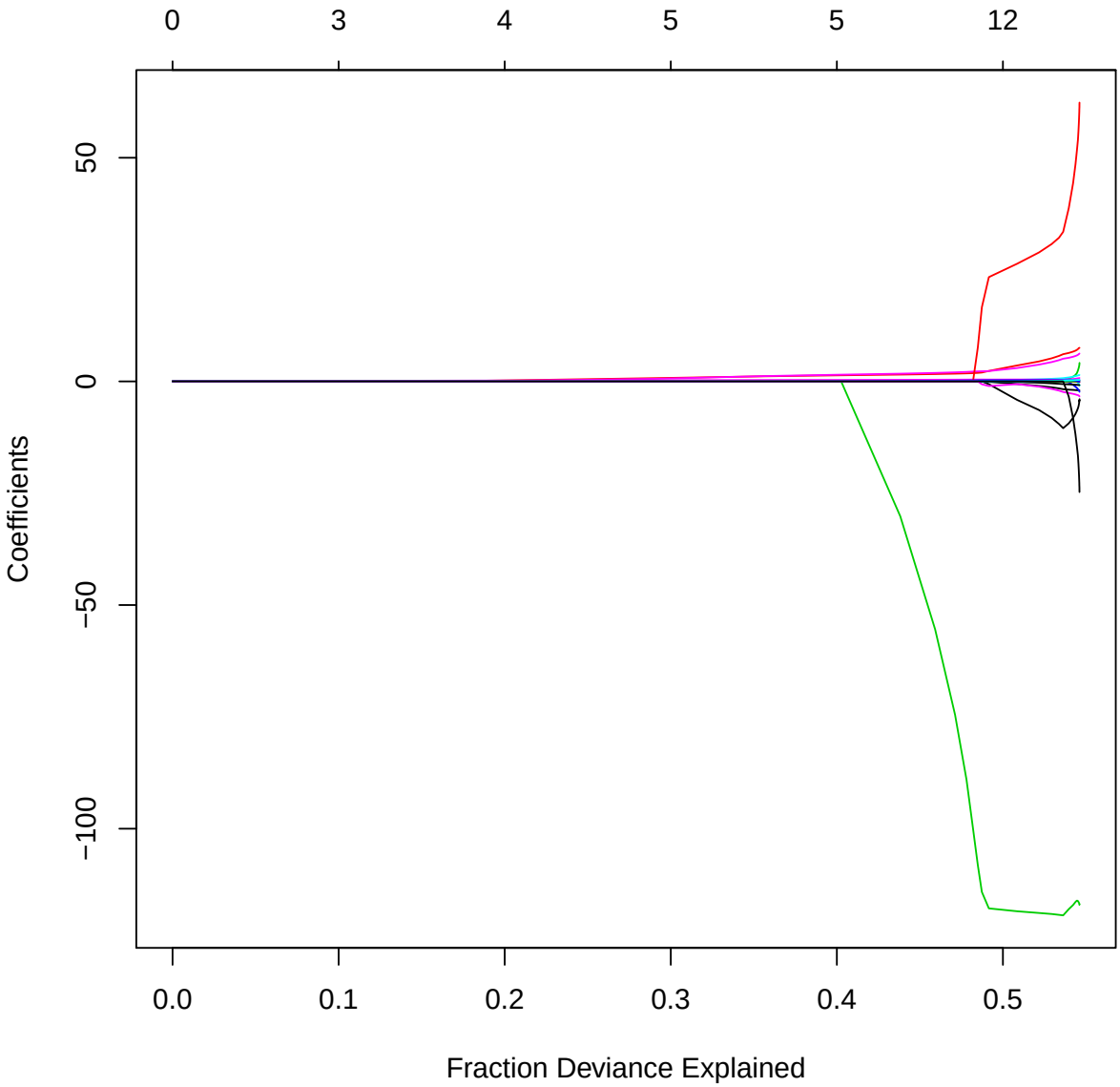

```
lasso <- my.glmnet(Salary ~ ., data = Hitters,
    alpha = 1, lambda = grid)
plot(lasso$model)
```




```
plot(lasso$model, xvar = "lambda")
```




```
plot(lasso$model, xvar = "dev")
```

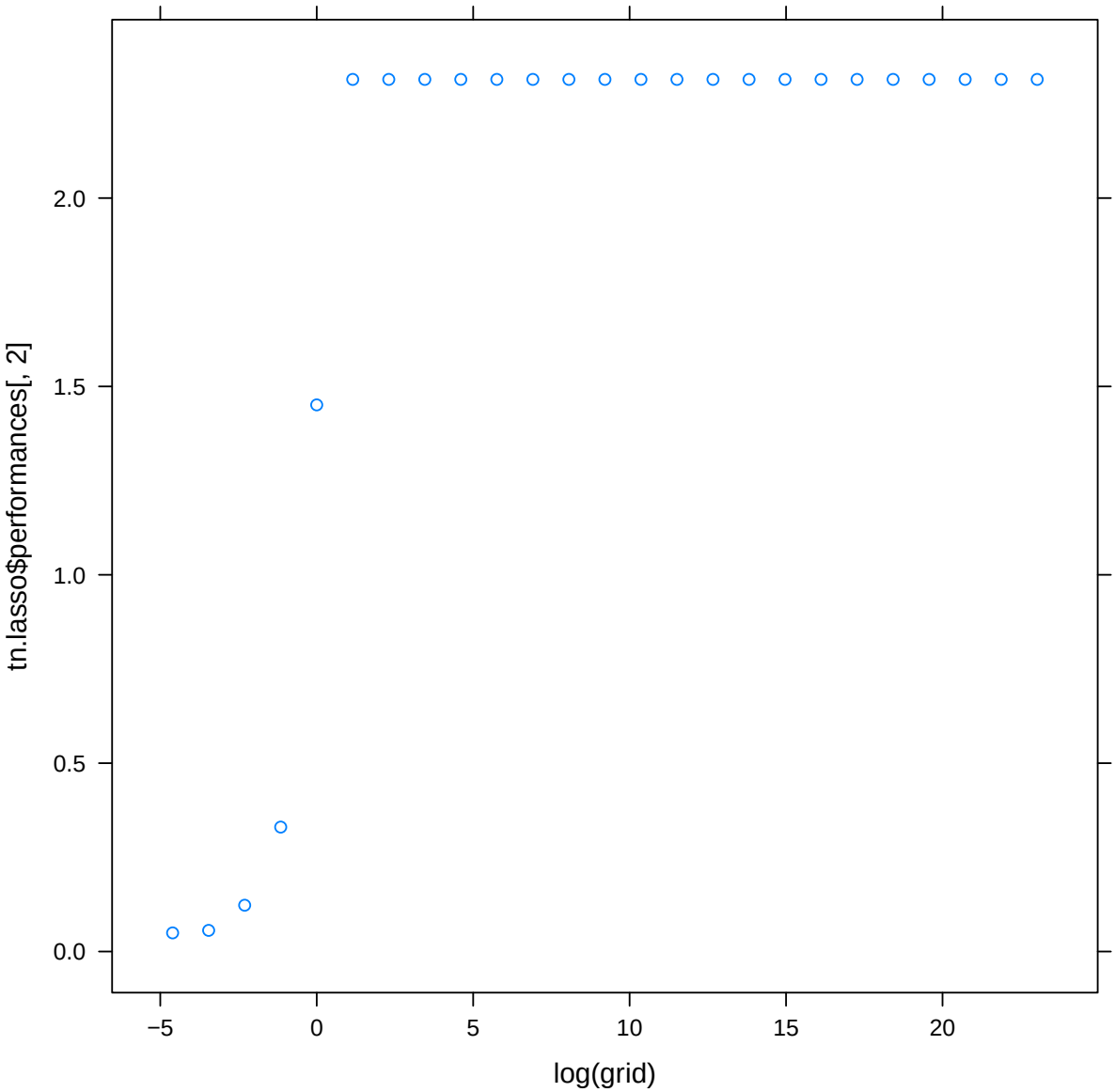



```
tn.lasso <- tune(my.glmnet, Salary ~ ., data = Hitters,
    alpha = 1, ranges = list(lambda = grid))
tn.lasso
```

```
##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##    lambda
##  2.656088
##
## - best performance: 114418
```

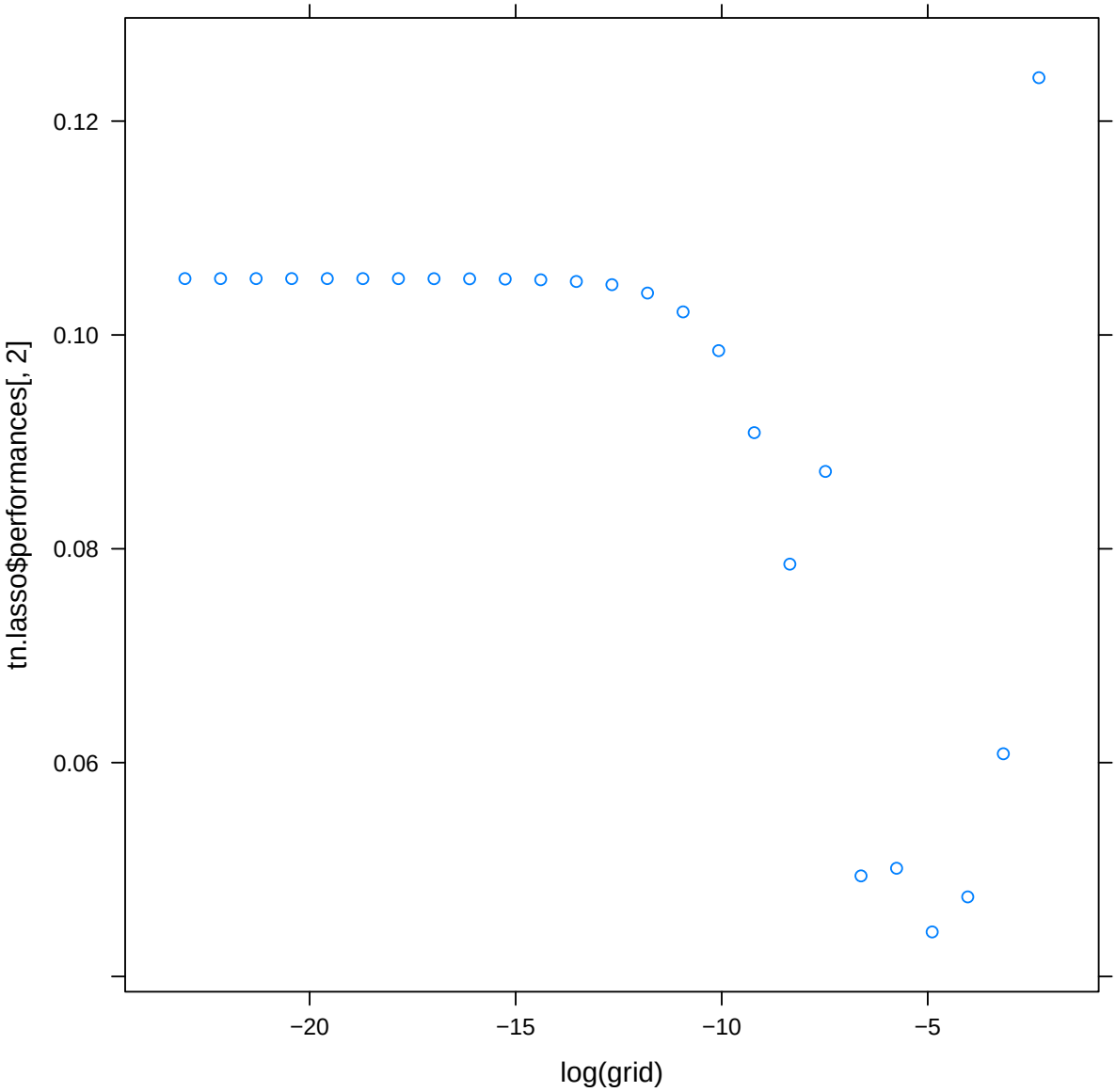
```
# gasoline again
```

```r
grid <- 10^seq(10, -2, length.out = 25)
tn.lasso <- tune(my.glmnet, octane ~ ., data = gasoline,
    alpha = 1, ranges = list(lambda = grid))
tn.lasso

##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  lambda
##    0.01
##
## - best performance: 0.04948989

xyplot(tn.lasso$performances[, 2] ~ log(grid))
```




```
# move grid
grid <- 10^seq(-1, -10, length.out = 25)
tn.lasso <- tune(my.glmnet, octane ~ ., data = gasoline,
    alpha = 1, ranges = list(lambda = grid))
tn.lasso

##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       lambda
##  0.007498942
##
```

```
## - best performance: 0.04416645

xyplot(tn.lasso$performances[, 2] ~ log(grid))
```




```
coefi.lasso.sp <- coef(tn.lasso$best.model$model)
coefi.lasso <- as.vector(as.matrix(coefi.lasso.sp))
names(coefi.lasso) <- rownames(coefi.lasso.sp)
coefi.lasso <- coefi.lasso[coefi.lasso !=
    0]
coefi.lasso

##  (Intercept)         S912         S914         S984
##  98.51731235   2.50389616   9.48472014   0.02908611
##         S992        S1206        S1214        S1218
##  10.08140814 -13.08513605  -7.17281422 -17.61653950
```

```
##        S1224        S1228        S1230        S1360
## -39.06493971 -12.81593338  -9.16021429  35.75121982
##        S1362        S1366        S1368        S1534
##  21.54848758   6.71229358   9.86548862  -0.84510338
##        S1542        S1632        S1634        S1638
##  -2.51020717  -7.15567837  -0.28996208  -6.15776471
##        S1684        S1686        S1688        S1690
##  -0.50788524  -1.64054708   0.56993497  -0.50046504
##        S1692
##  -1.08988221
```

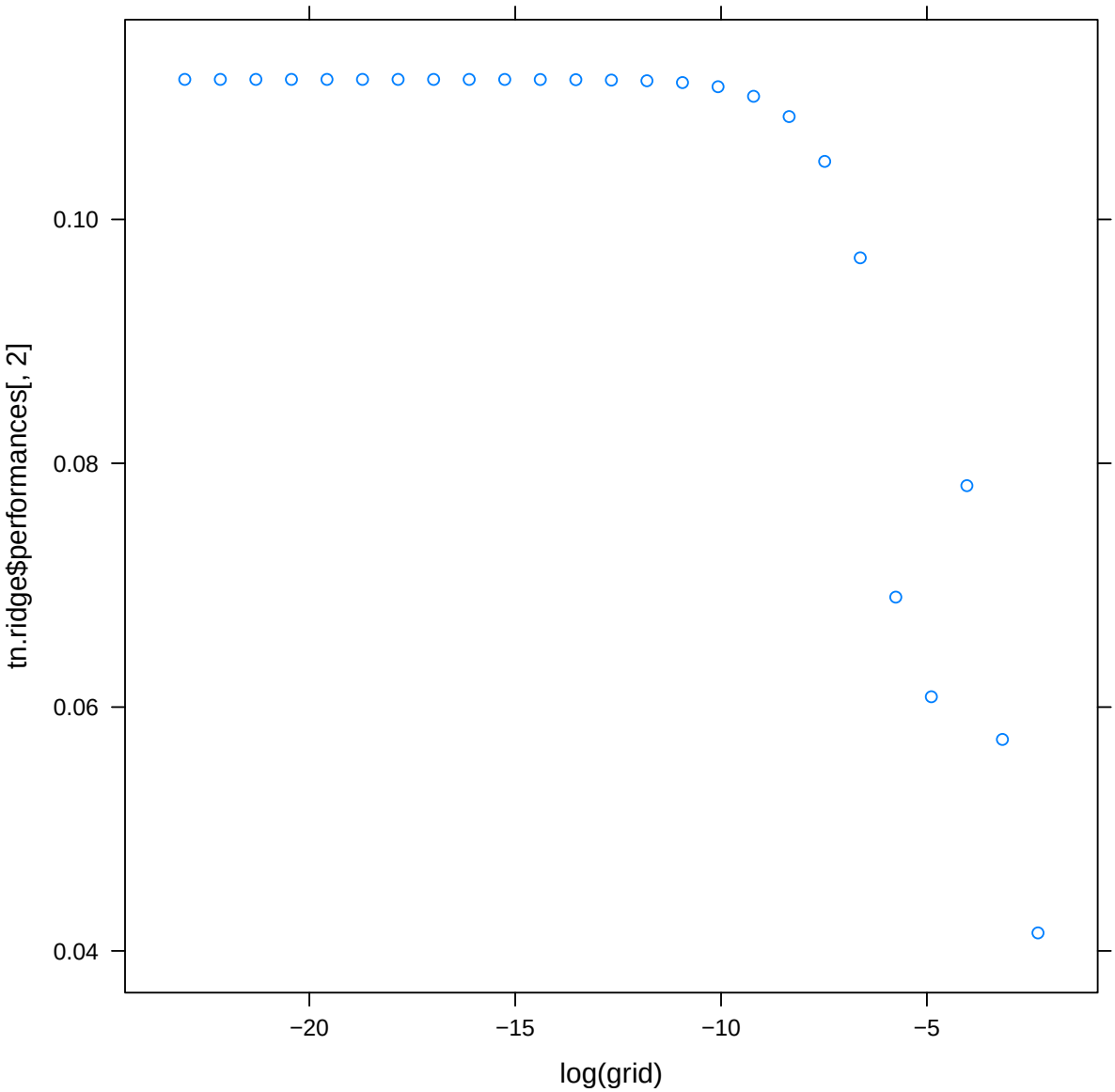
```
coefi.rl
```

```
##     S1208     S1224     S1374
## -35.25926 -66.99336  66.46622
```

```
# try ridge
grid <- 10^seq(-1, -10, length.out = 25)
tn.ridge <- tune(my.glmnet, octane ~ ., data = gasoline,
    alpha = 0, ranges = list(lambda = grid))
tn.ridge
```

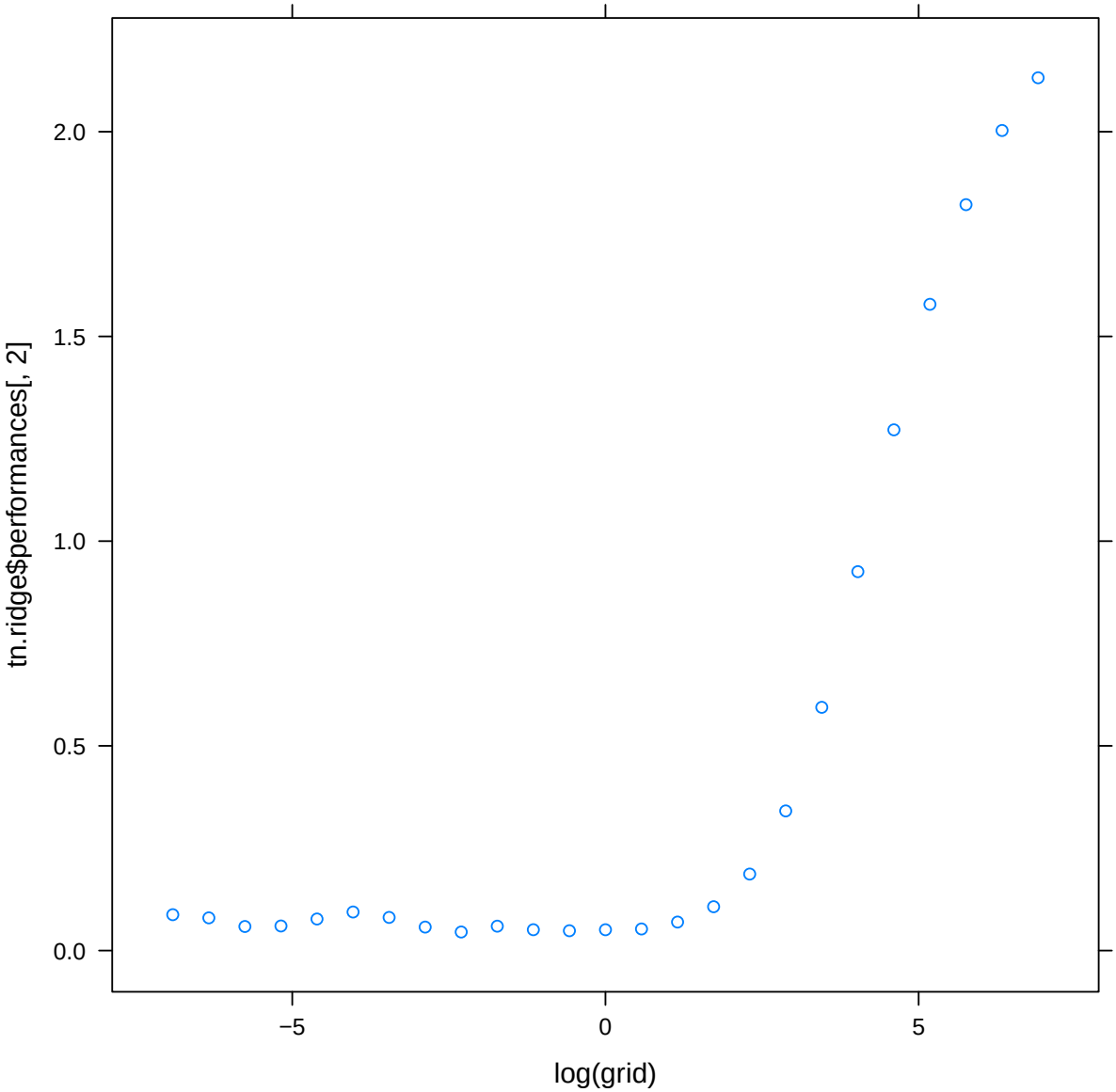
```
##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  lambda
##     0.1
##
## - best performance: 0.0414802
```

```
xyplot(tn.ridge$performances[, 2] ~ log(grid))
```



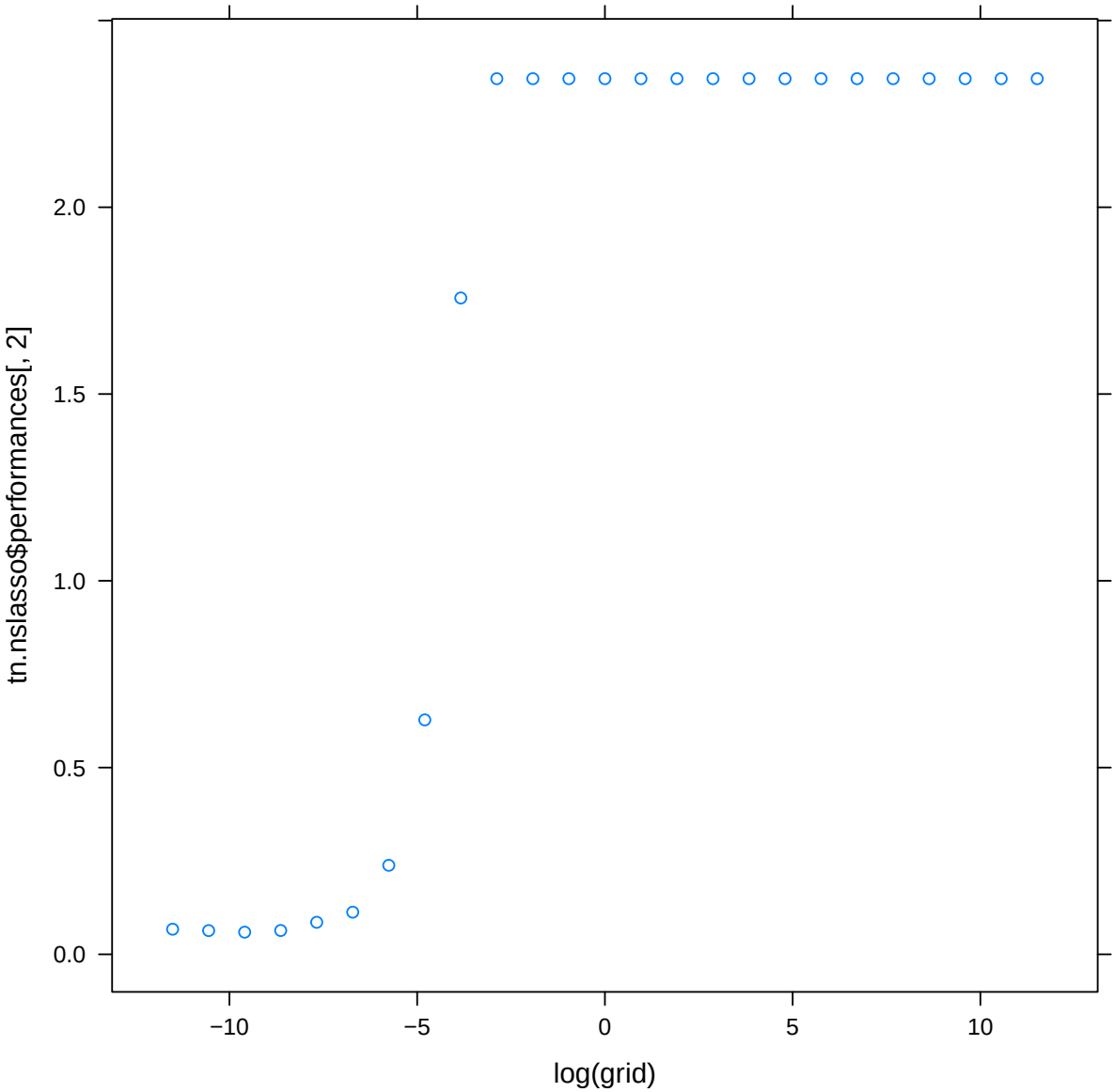

```
# move it!
grid <- 10^seq(0, 10, length.out = 25)
tn.ridge <- tune(my.glmnet, octane ~ ., data = gasoline,
    alpha = 0, ranges = list(lambda = grid))
# move it!
grid <- 10^seq(-3, 3, length.out = 25)
tn.ridge <- tune(my.glmnet, octane ~ ., data = gasoline,
    alpha = 0, ranges = list(lambda = grid))
tn.ridge

##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
```

```
## - best parameters:
##  lambda
##     0.1
##
## - best performance: 0.0453564

xyplot(tn.ridge$performances[, 2] ~ log(grid))
```



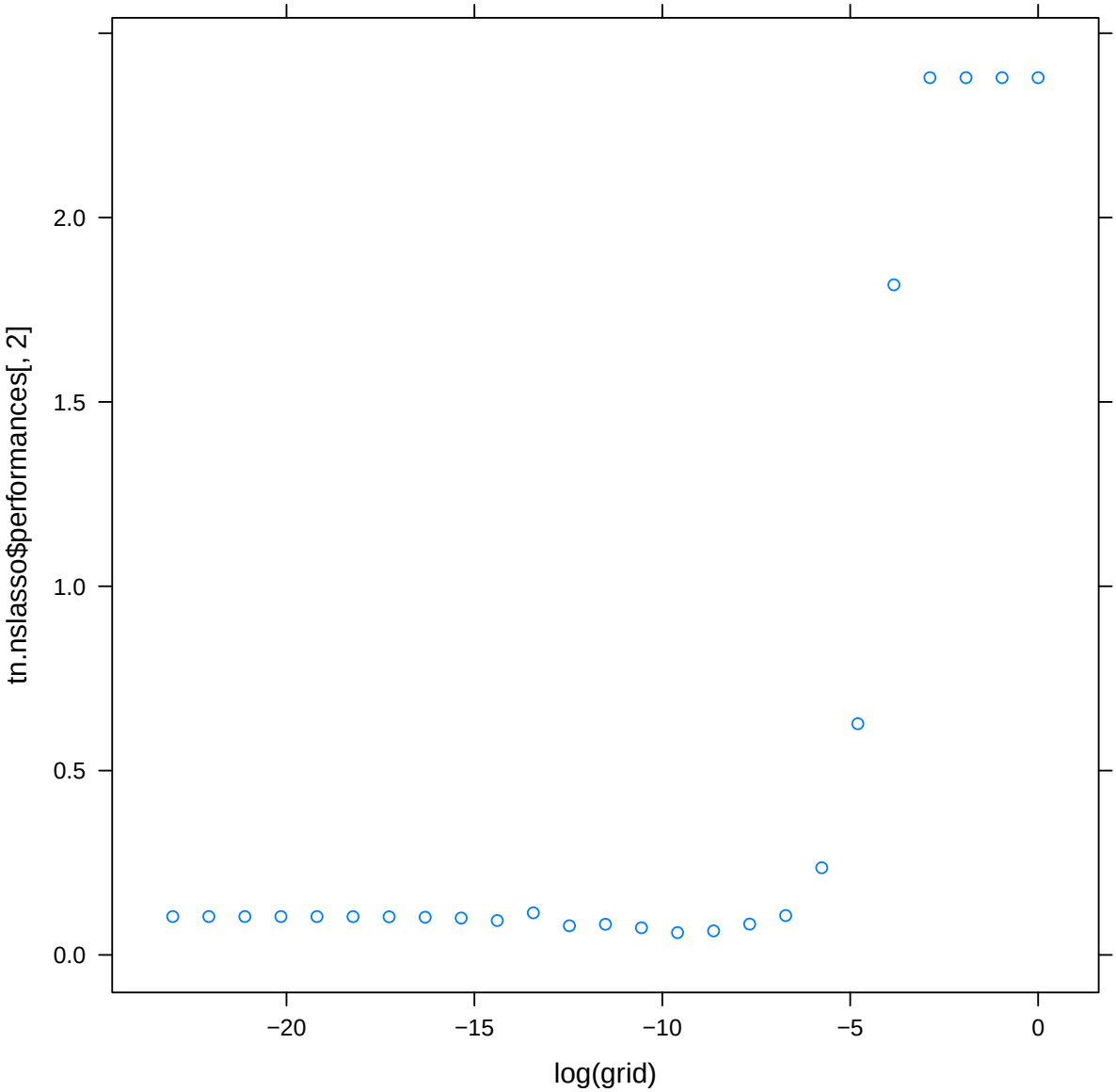

```
# don't standardize
grid <- 10^seq(-5, 5, length.out = 25)
tn.nslasso <- tune(my.glmnet, octane ~ .,
    data = gasoline, alpha = 1, standardize = FALSE,
    ranges = list(lambda = grid))
tn.nslasso

##
```

```
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        lambda
##  6.812921e-05
##
## - best performance: 0.05936071

xyplot(tn.nslasso$performances[, 2] ~ log(grid))
```




```
# move it
grid <- 10^seq(-10, 0, length.out = 25)
tn.nslasso <- tune(my.glmnet, octane ~ .,
```

```r
    data = gasoline, alpha = 1, standardize = FALSE,
    ranges = list(lambda = grid))
tn.nslasso

##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        lambda
##  6.812921e-05
##
## - best performance: 0.0604996

xyplot(tn.nslasso$performances[, 2] ~ log(grid))
```




```
print(tn.rl)

##
## Error estimation of 'lm' using 10-fold cross validation: 0.04787449

print(tn.pcr)

##
## Parameter tuning of 'pcr':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  ncomp
##      5
```

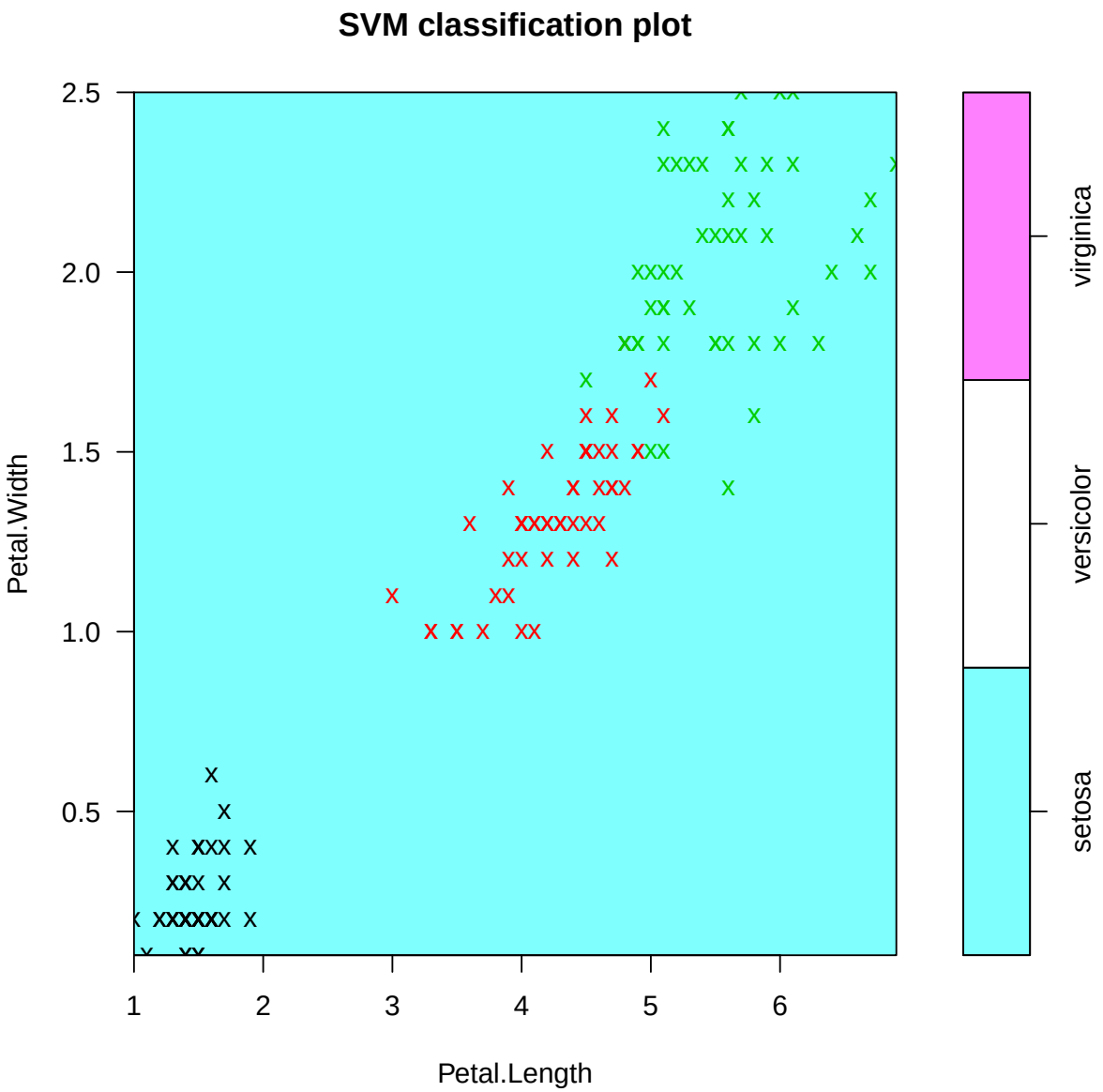
```
##
## - best performance: 0.05946477

print(tn.plsr)

##
## Parameter tuning of 'plsr':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  ncomp
##      7
##
## - best performance: 0.05732338

print(tn.lasso)

##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       lambda
##  0.007498942
##
## - best performance: 0.04416645

print(tn.nslasso)

##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        lambda
##  6.812921e-05
##
## - best performance: 0.0604996

print(tn.ridge)

##
## Parameter tuning of 'my.glmnet':
##
## - sampling method: 10-fold cross validation
##
```

```
## - best parameters:
##  lambda
##     0.1
##
## - best performance: 0.0453564
```

10 Материалы с занятия 28 ноября

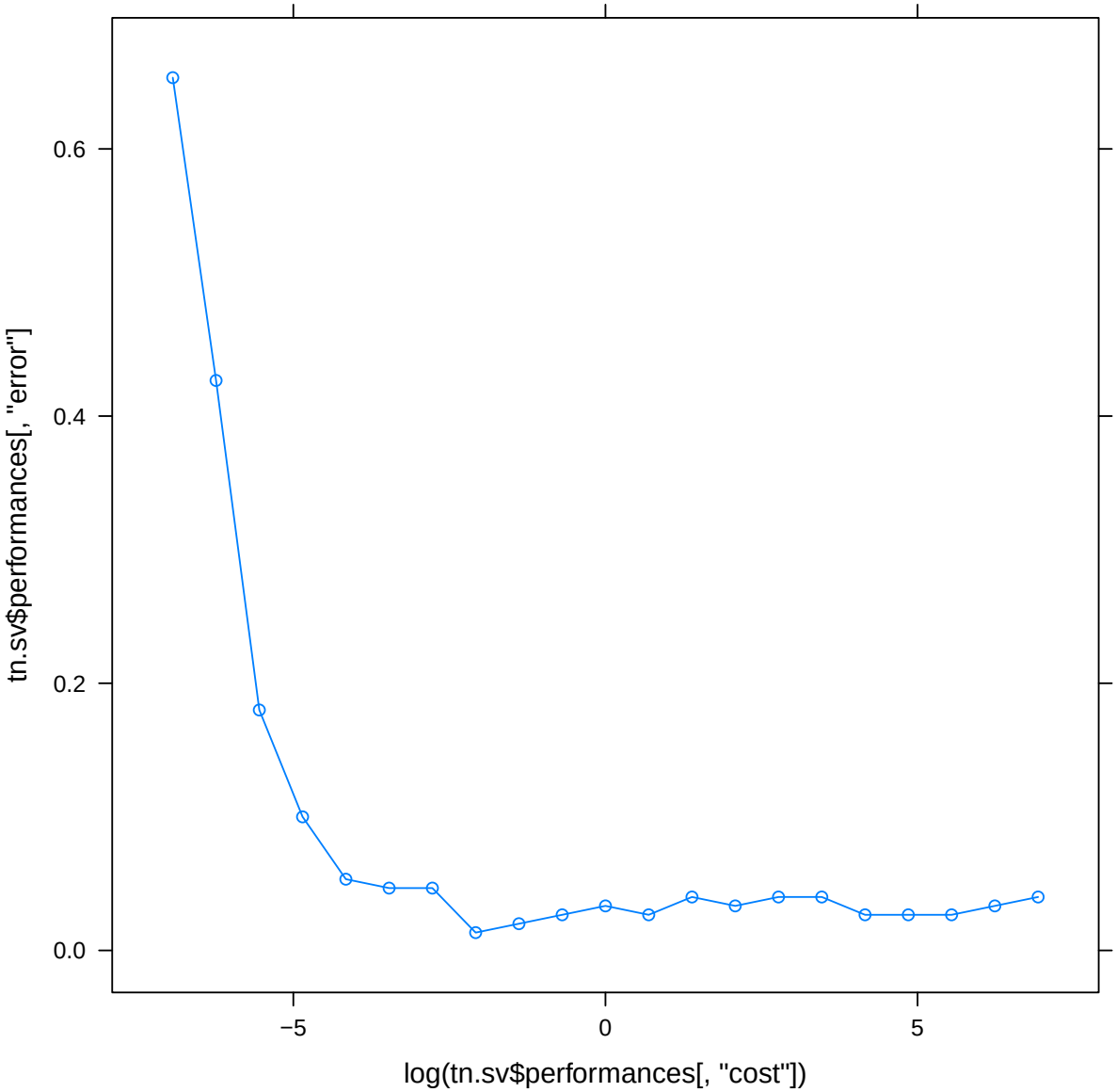
```
read_chunk("svm/svm.R")
read_chunk("svm/promoters.R")
read_chunk("svm/glaucomaM.R")
read_chunk("svm/artificial.R")
```

```
library(e1071)  # svm()
library(kernlab)  # ksvm()
library(klaR)  # svmlight ()
library(svmpath)  # svmpath()
library(MASS)
library(lattice)
library(latticeExtra)
# http://cran.r-project.org/web/packages/kernlab/vignettes/kernlab.pdf
# http://cbio.ensmp.fr/~jvert/svn/tutorials/practical/stringkernels/stringkernels_n
# http://www.jstatsoft.org/v15/i09/paper
sv <- svm(Species ~ ., data = iris, type = "C-classification",
    kernel = "linear", cost = 2^(-20))
print(sv)

##
## Call:
## svm(formula = Species ~ ., data = iris, type = "C-classification",
##     kernel = "linear", cost = 2^(-20))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  9.536743e-07
##       gamma:  0.125
##
## Number of Support Vectors:  150

plot(sv, iris, Petal.Width ~ Petal.Length,
    slice = list(Sepal.Width = 3, Sepal.Length = 4))
```



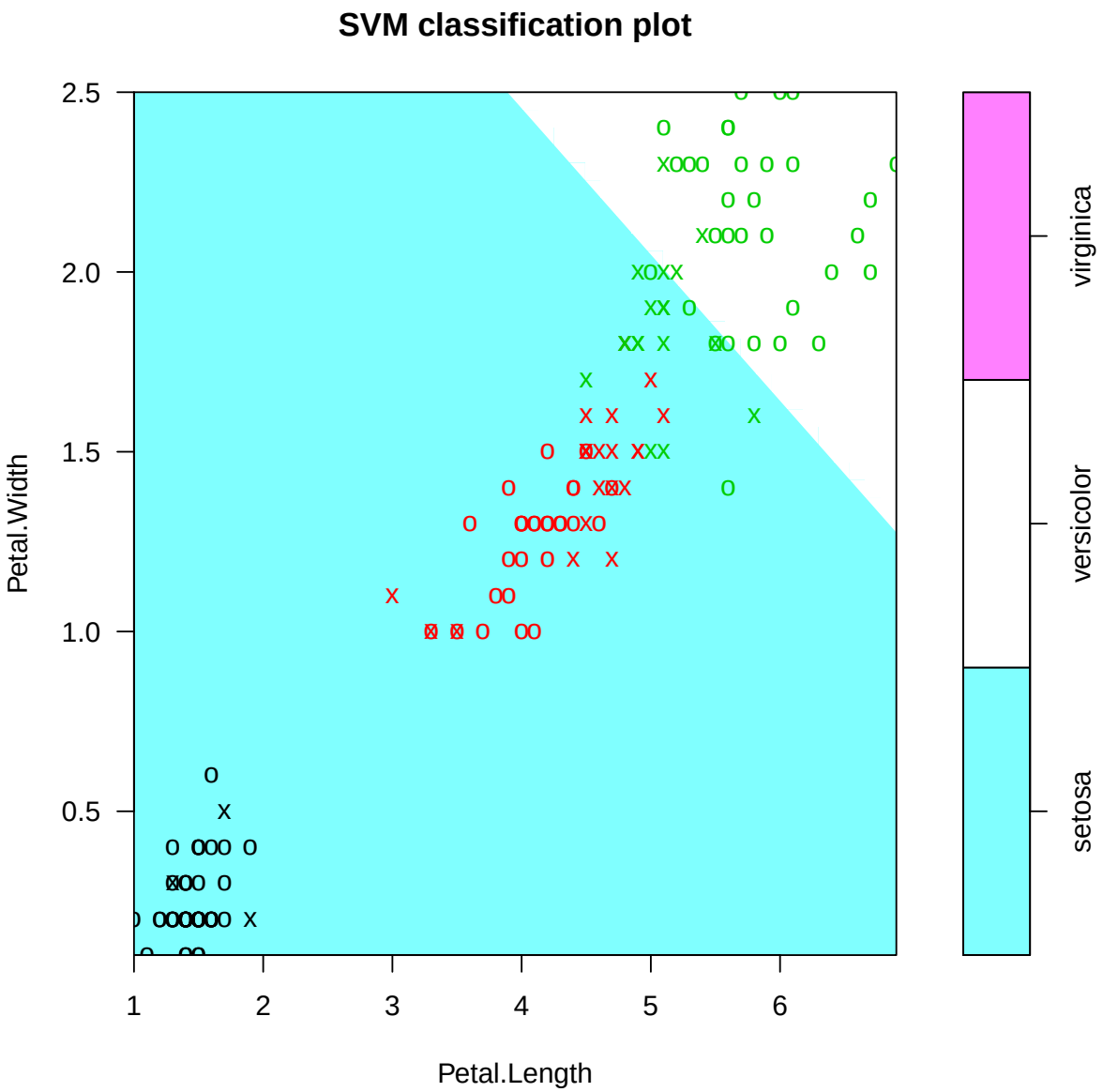

```
table(actual = iris$Species, predicted = predict(sv))

##              predicted
## actual        setosa versicolor virginica
##   setosa          50          0         0
##   versicolor       0         48         2
##   virginica        0         12        38

tn.sv <- tune(svm, Species ~ ., data = iris,
    type = "C-classification", kernel = "linear",
    cost = 2^(-20))
tn.sv

##
## Error estimation of 'svm' using 10-fold cross validation: 0.68
```

```r
# tn.sv <- tune(svm, Species ~ ., data =
# iris, type = 'C-classification', kernel
# = 'polynomial', degree = 1, ranges =
# list(cost = 2^(-10:10))) OR (better)
tn.sv <- tune.svm(Species ~ ., data = iris,
    type = "C-classification", kernel = "linear",
    cost = 2^(-10:10))
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.125
##
## - best performance: 0.01333333

xyplot(tn.sv$performances[, "error"] ~ log(tn.sv$performances[,
    "cost"]), type = "b")
```

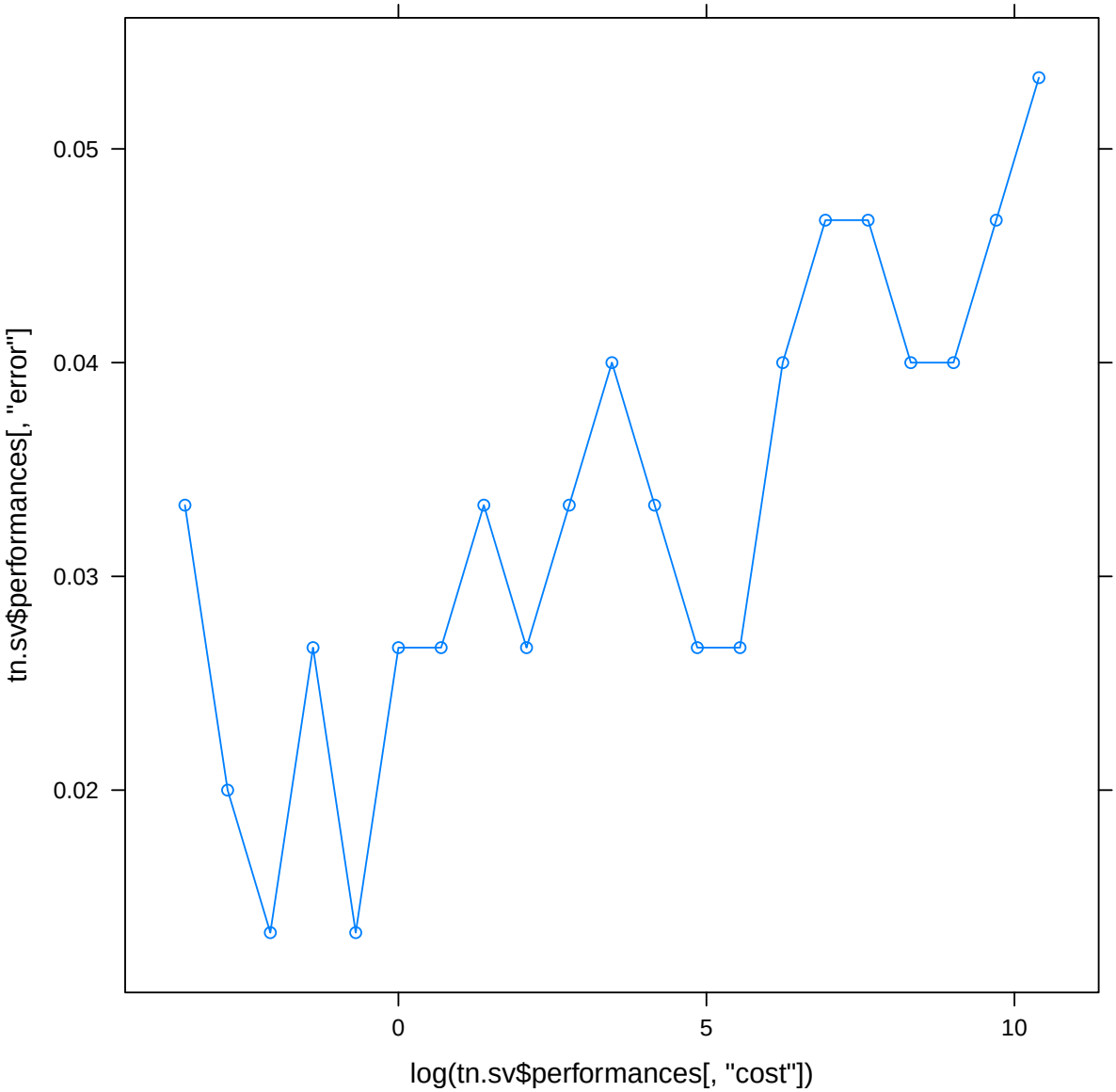



```
tn.sv <- tune.svm(Species ~ ., data = iris,
    type = "C-classification", kernel = "linear",
    tunecontrol = tune.control(cross = nrow(iris)),
    cost = 2^(-5:15))
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: leave-one-out
##
## - best parameters:
##   cost
##  0.125
##
```

```
## - best performance: 0.01333333

plot(tn.sv$best.model, iris, Petal.Width ~
    Petal.Length, slice = list(Sepal.Width = 3,
    Sepal.Length = 4))
```

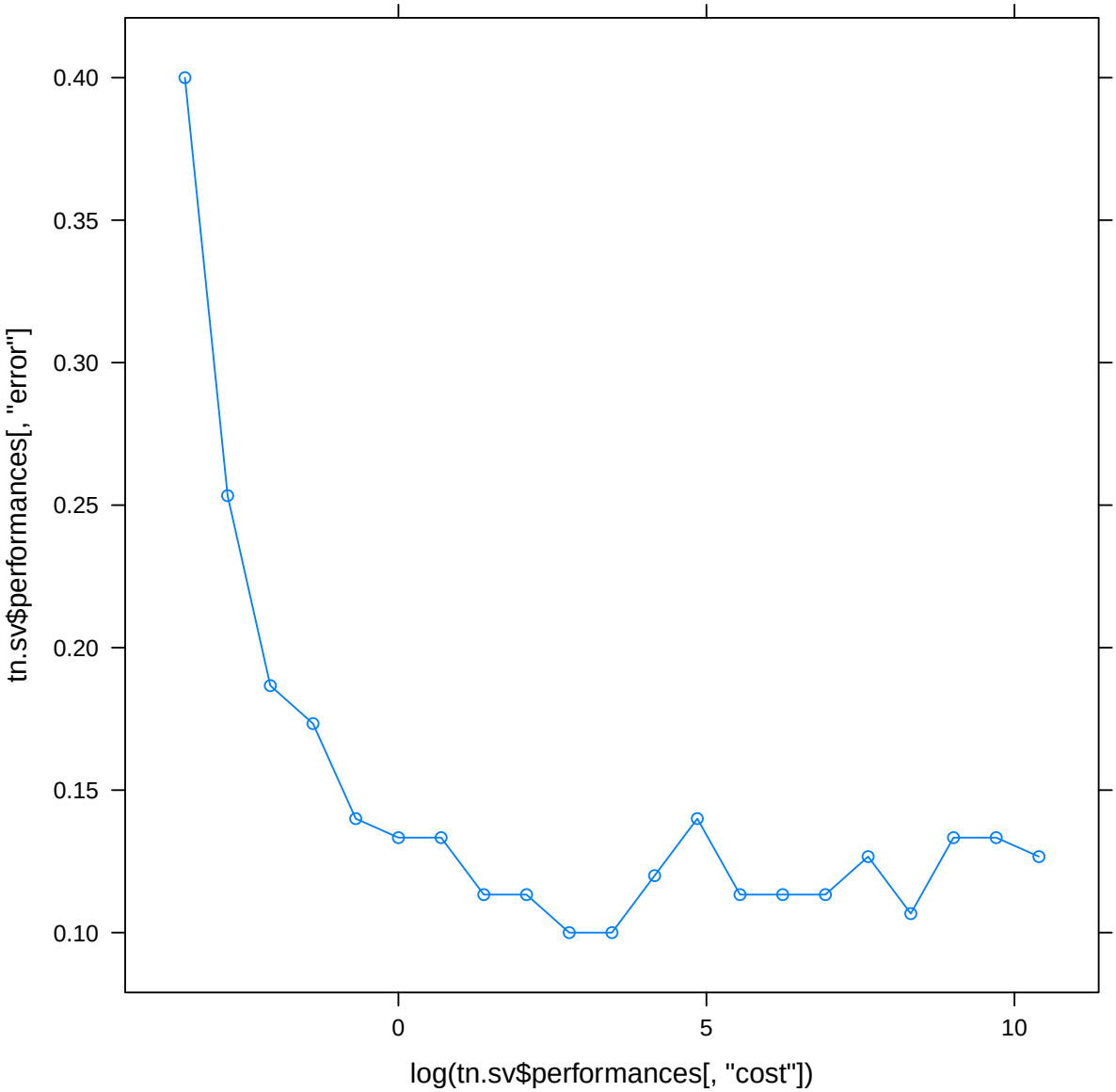



```
xyplot(tn.sv$performances[, "error"] ~ log(tn.sv$performances[,
    "cost"]), type = "b")
```



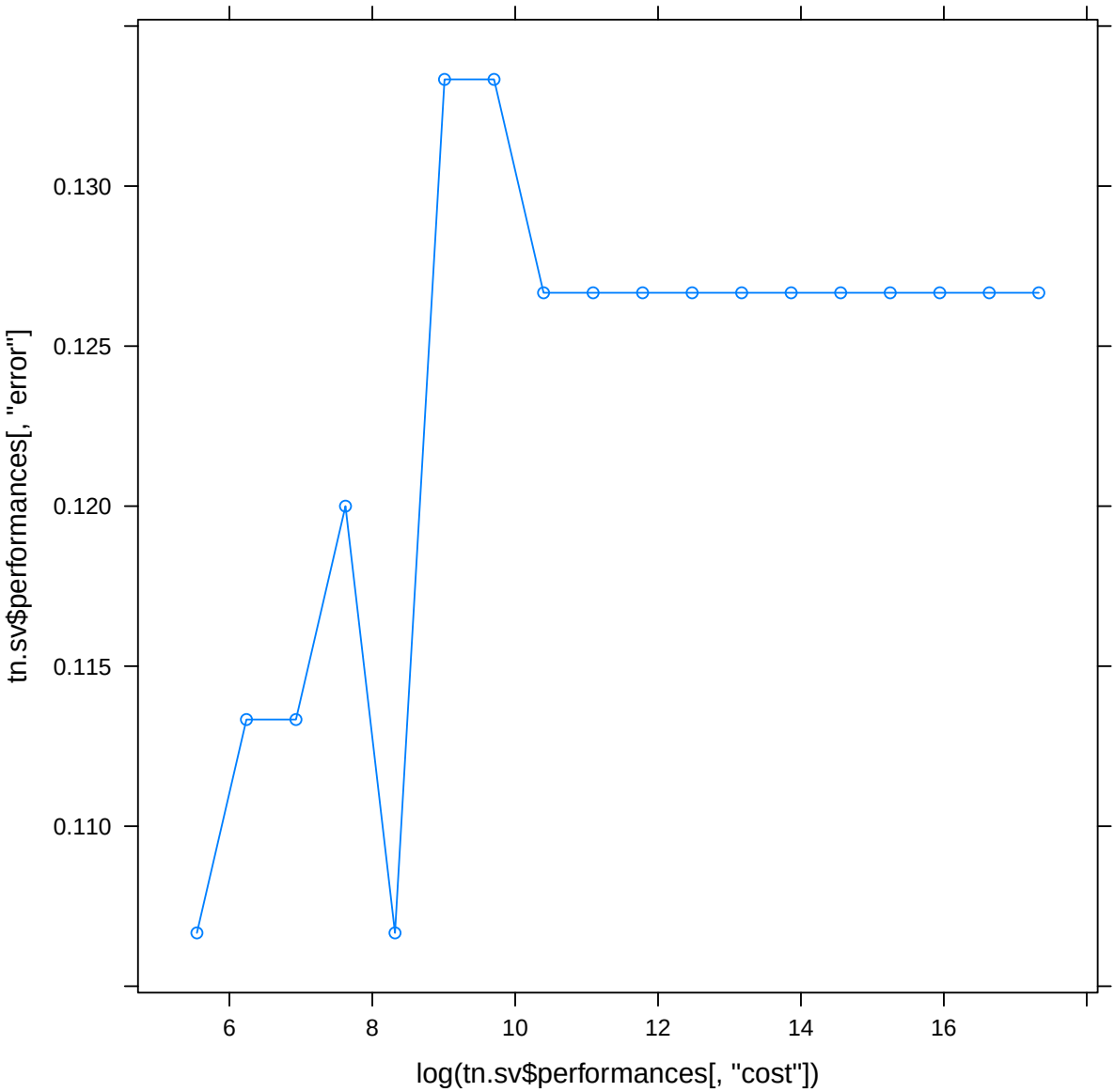

```
tn.sv$best.model

##
## Call:
## best.svm(x = Species ~ ., data = iris, cost = 2^(-5:15),
##     type = "C-classification", kernel = "linear",
##     tunecontrol = tune.control(cross = nrow(iris)))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.125
##       gamma:  0.125
##
```

```
## Number of Support Vectors:  43

# Try poly kernel
tn.sv <- tune.svm(Species ~ ., data = iris,
    type = "C-classification", kernel = "polynomial",
    tunecontrol = tune.control(cross = nrow(iris)),
    degree = 2, cost = 2^(-5:15))
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: leave-one-out
##
## - best parameters:
##  degree cost
##       2   16
##
## - best performance: 0.1

xyplot(tn.sv$performances[, "error"] ~ log(tn.sv$performances[,
    "cost"]), type = "b")
```



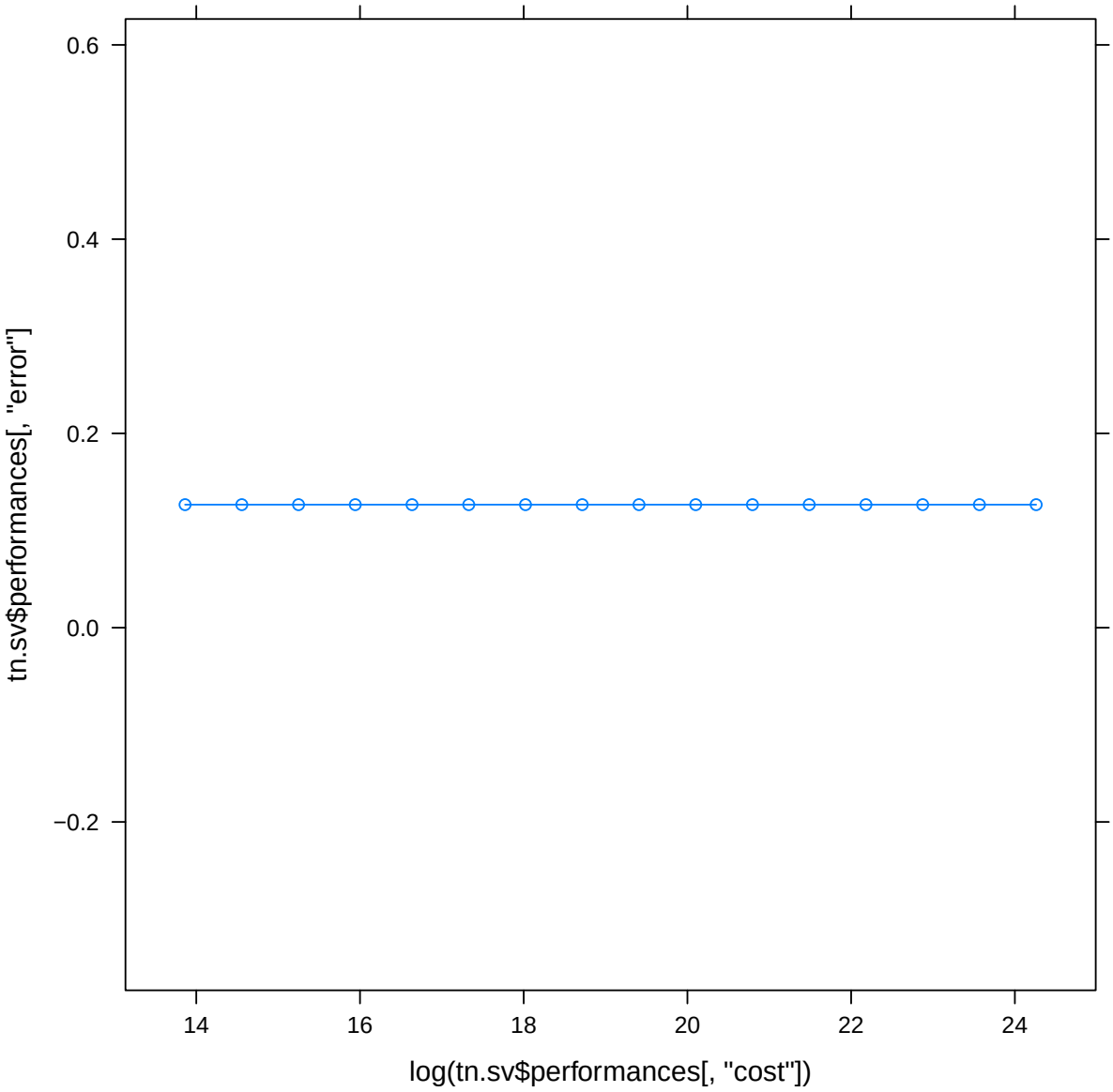

```
tn.sv <- tune.svm(Species ~ ., data = iris,
    type = "C-classification", kernel = "polynomial",
    tunecontrol = tune.control(cross = nrow(iris)),
    degree = 2, cost = 2^(8:25))
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: leave-one-out
##
## - best parameters:
##  degree cost
##       2  256
##
```

```
## - best performance: 0.1066667

tn.sv$best.model

##
## Call:
## best.svm(x = Species ~ ., data = iris, degree = 2,
##     cost = 2^(8:25), type = "C-classification", kernel = "polynomial",
##     tunecontrol = tune.control(cross = nrow(iris)))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  256
##      degree:  2
##       gamma:  0.125
##      coef.0:  0
##
## Number of Support Vectors:  44

xyplot(tn.sv$performances[, "error"] ~ log(tn.sv$performances[,
    "cost"]), type = "b")
```

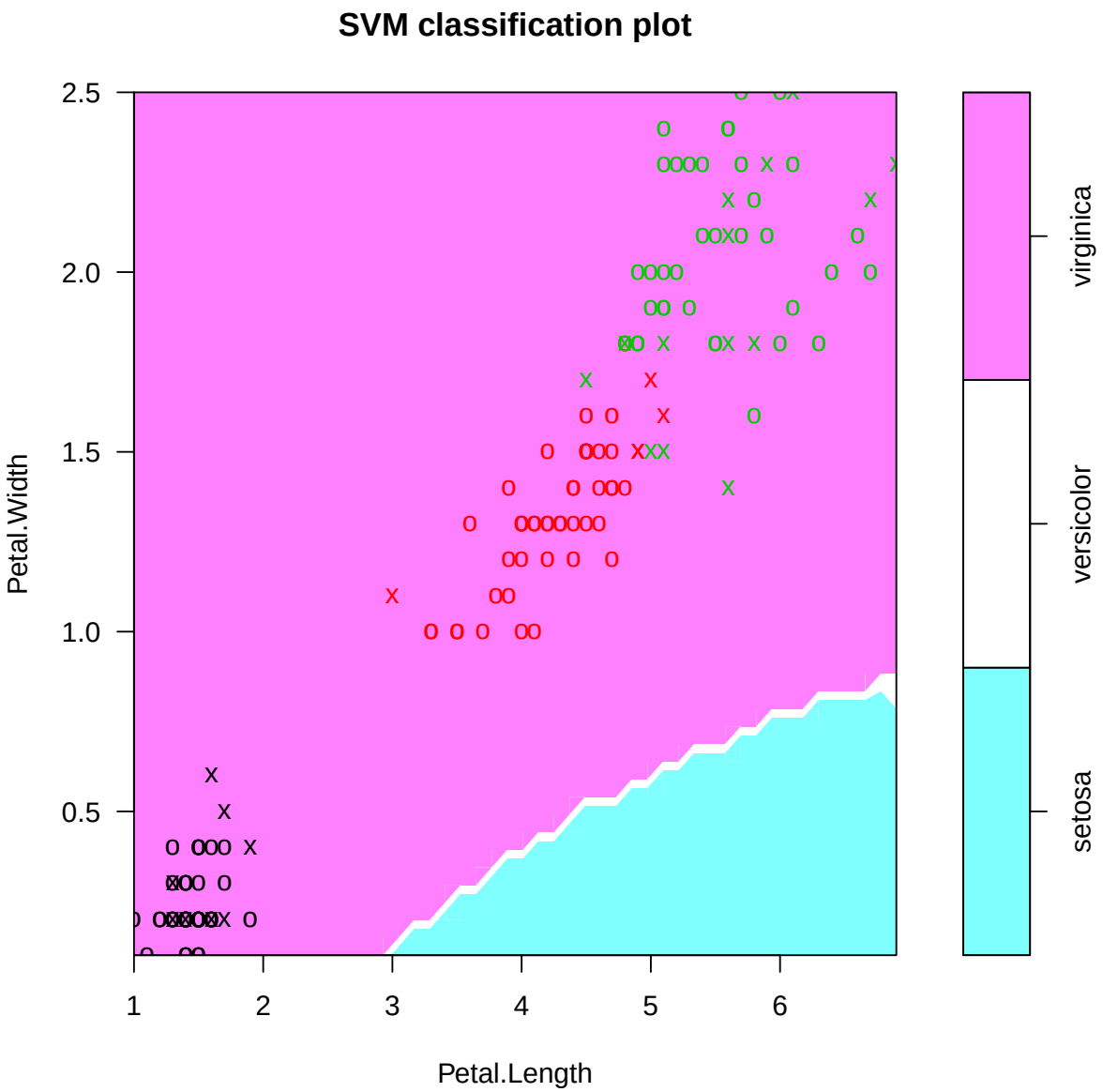



```
tn.sv <- tune.svm(Species ~ ., data = iris,
    type = "C-classification", kernel = "polynomial",
    tunecontrol = tune.control(cross = nrow(iris)),
    degree = 2, cost = 2^(20:35))
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: leave-one-out
##
## - best parameters:
##  degree    cost
##       2 1048576
##
```

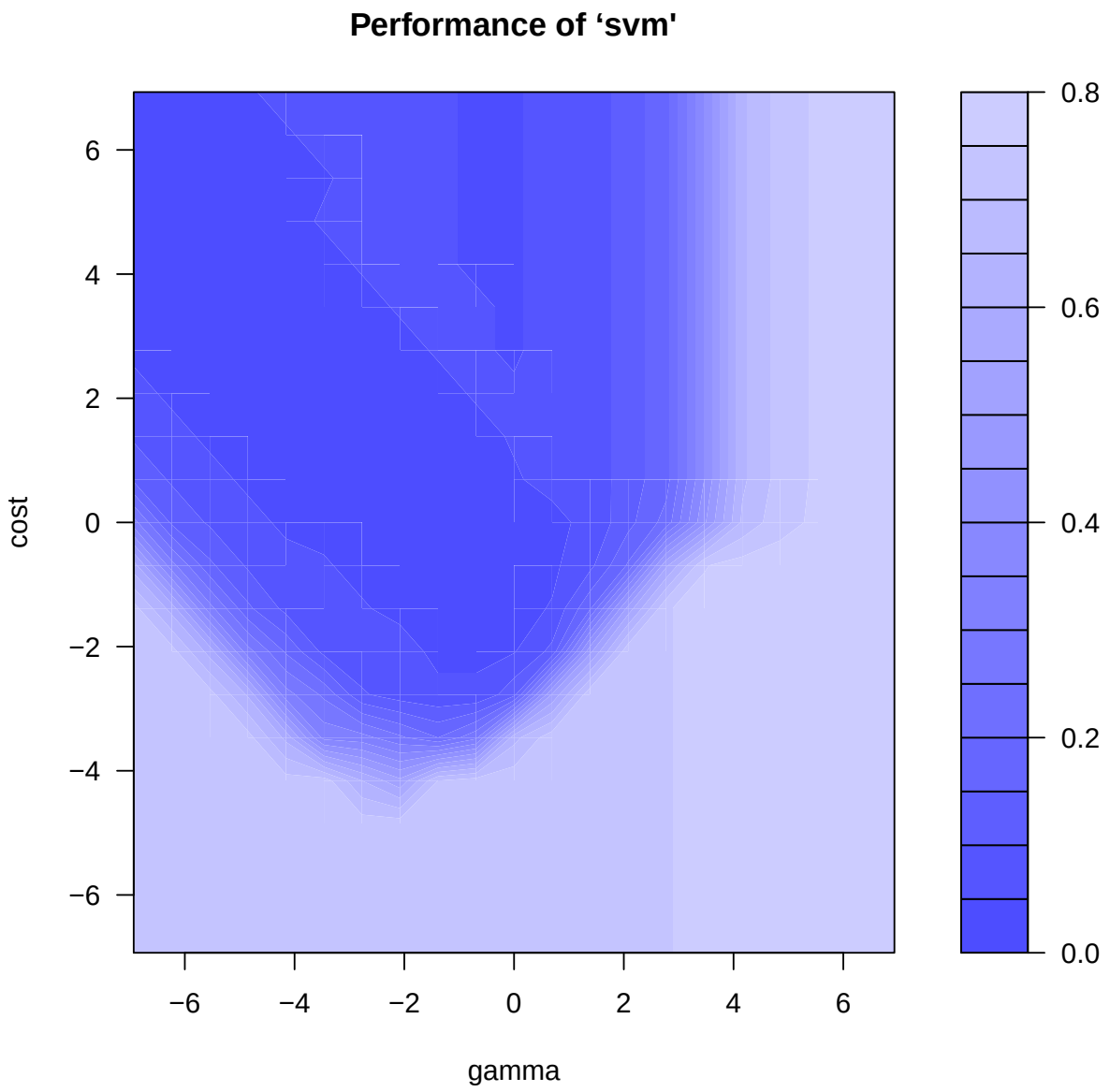
```
## - best performance: 0.1266667

xyplot(tn.sv$performances[, "error"] ~ log(tn.sv$performances[,
    "cost"]), type = "b")
```



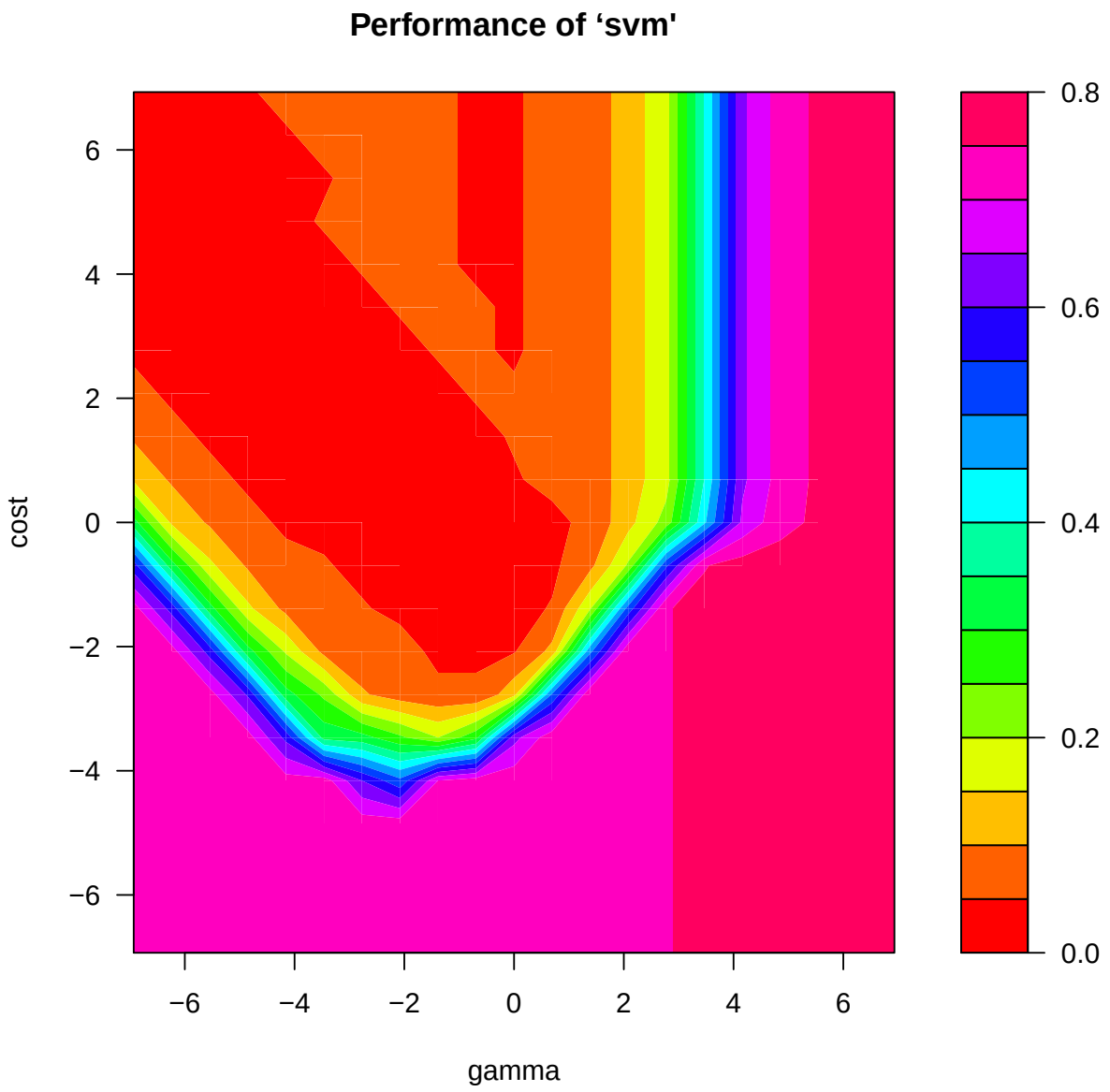

```
plot(tn.sv$best.model, iris, Petal.Width ~
    Petal.Length, slice = list(Sepal.Width = 3,
    Sepal.Length = 4))
```




```r
# Try gaussian kernel
tn.sv <- tune.svm(Species ~ ., data = iris,
    type = "C-classification", kernel = "radial",
    tunecontrol = tune.control(cross = 10),
    gamma = 2^(-10:10), cost = 2^(-10:10))
plot(tn.sv, transform.x = log, transform.y = log)
```




```
plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```




```
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      gamma cost
##  0.0078125   32
##
## - best performance: 0.01333333

tn.sv$best.model

##
```

```
## Call:
## best.svm(x = Species ~ ., data = iris, gamma = 2^(-10:10),
##     cost = 2^(-10:10), type = "C-classification",
##     kernel = "radial", tunecontrol = tune.control(cross = 10))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  32
##       gamma:  0.0078125
##
## Number of Support Vectors:  27
```

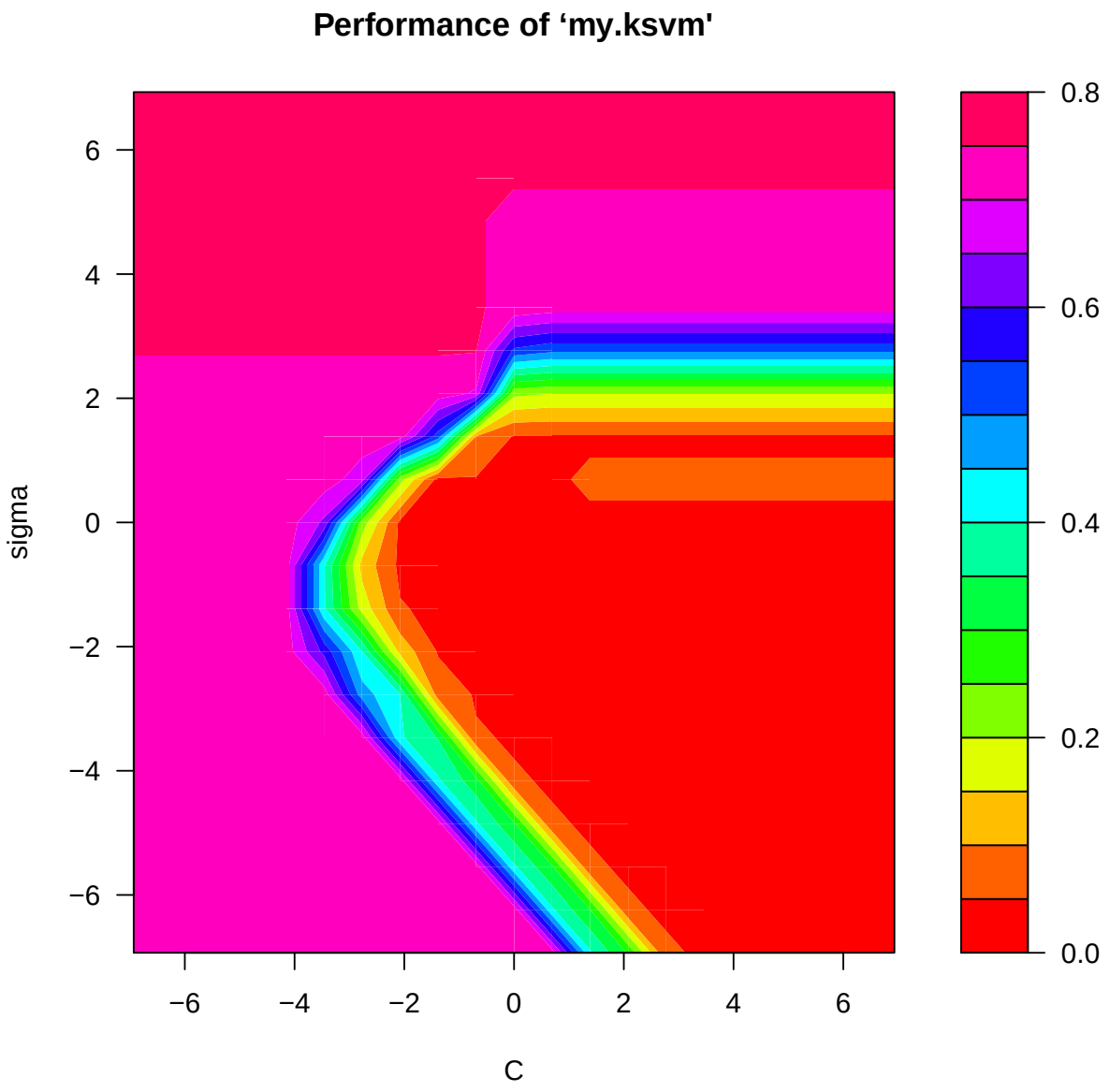
```
sv <- ksvm(Species ~ ., data = iris, type = "C-svc",
    C = 1, kernel = "laplacedot", kpar = list(sigma = 1))
table(actual = iris$Species, predicted = predict(sv))
```

```
##             predicted
## actual       setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         49         1
##   virginica       0          0        50
```

```
# Disable best model computation because
# of error in tune for S4-class learners
tn.sv <- tune(ksvm, Species ~ ., data = iris,
    type = "C-svc", C = 1, kernel = "laplacedot",
    kpar = list(sigma = 1), tunecontrol = tune.control(best.model = FALSE))
print(tn.sv)
```

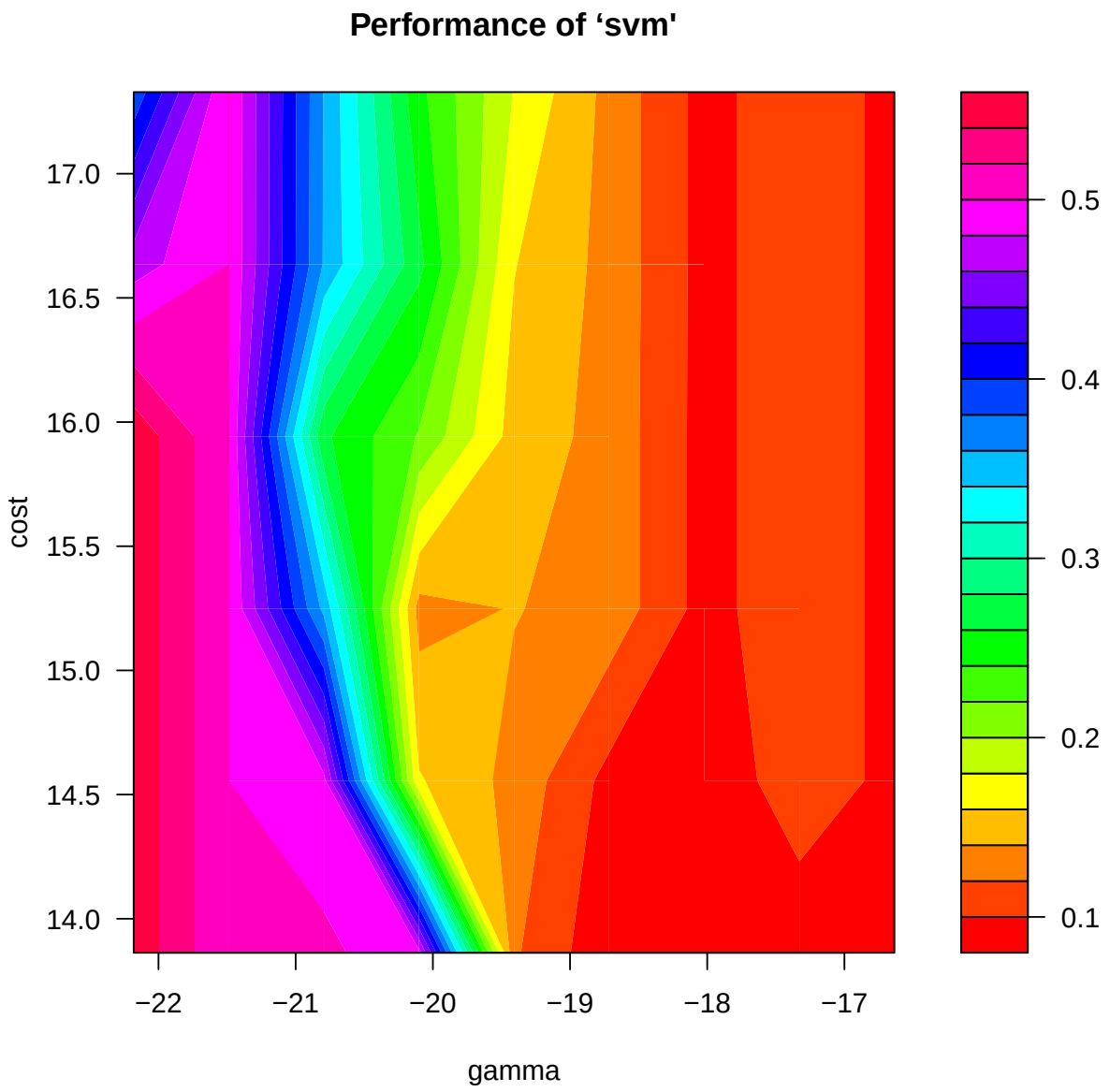
```
##
## Error estimation of 'ksvm' using 10-fold cross validation: 0.04
```

```
my.ksvm <- function(..., sigma) ksvm(...,
    kpar = list(sigma = sigma))
tn.sv <- tune(my.ksvm, Species ~ ., data = iris,
    type = "C-svc", ranges = list(C = 2^(-10:10),
        sigma = 2^(-10:10)), kernel = "laplacedot",
    tunecontrol = tune.control(best.model = FALSE))
plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```



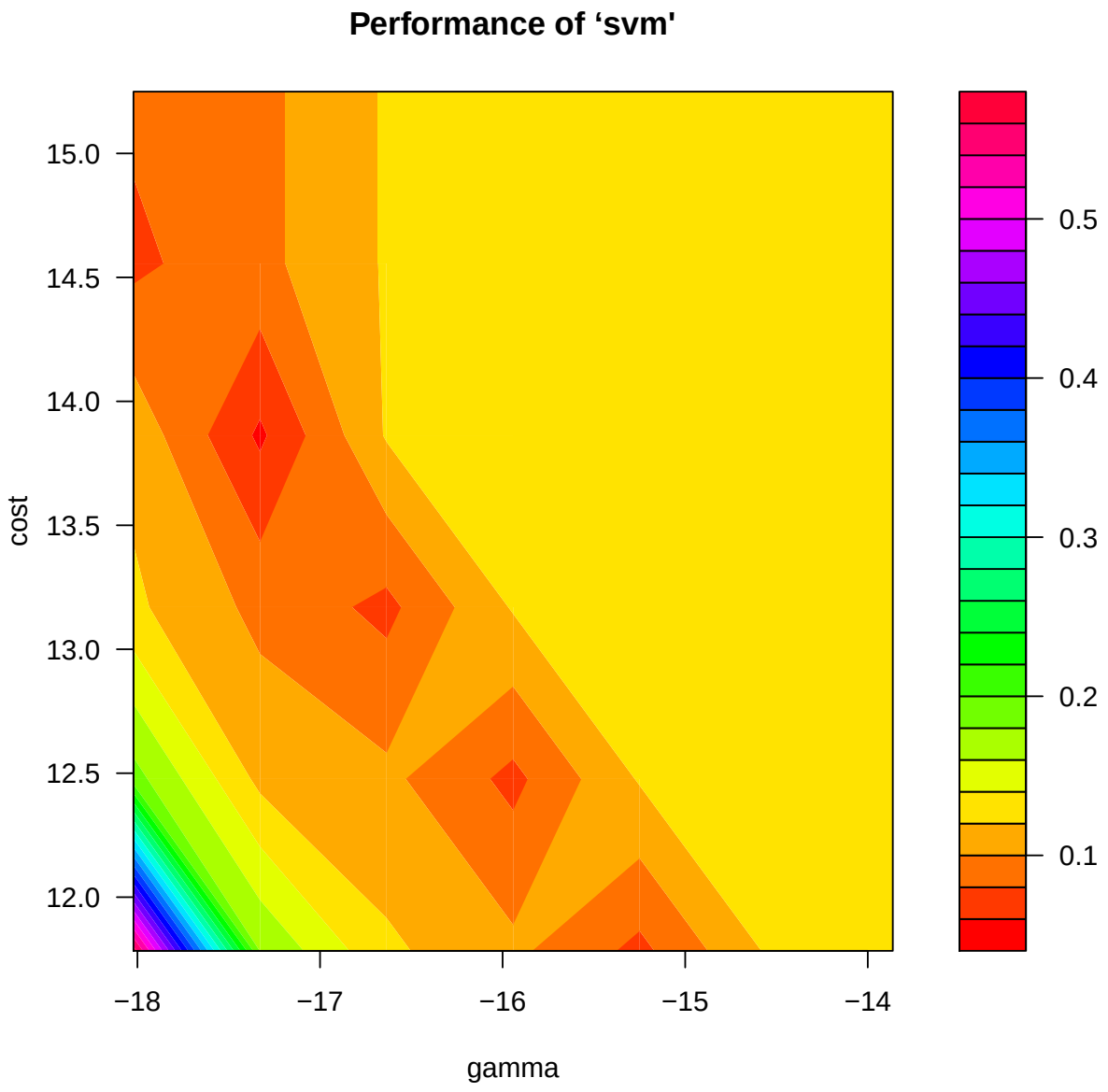

```
print(tn.sv)

##
## Parameter tuning of 'my.ksvm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##    C        sigma
##  256 0.0009765625
##
## - best performance: 0.02
```

```r
library(e1071)  # svm()
library(kernlab)  # ksvm()
library(klaR)  # svmlight ()
library(svmpath)  # svmpath()
library(MASS)
library(lattice)
library(latticeExtra)
promoters <- read.table(file = "svm/promoters.txt",
    comment.char = "#", stringsAsFactors = FALSE,
    header = FALSE, sep = ",", strip.white = TRUE,
    blank.lines.skip = TRUE)
names(promoters) <- c("ind", "names", "nuc")
promoters$ind <- factor(ifelse(promoters$ind ==
    "-", "Non-promoter", "Promoter"))
rownames(promoters) <- promoters$names
promoters$names <- NULL
promoters$nuc <- as.character(promoters$nuc)
nuc <- as.data.frame(strsplit(promoters$nuc,
    split = ""))
nuc <- as.data.frame(t(nuc))
promoters.factor <- cbind(nuc, ind = promoters$ind)
tn.sv <- tune.svm(ind ~ ., data = promoters.factor,
    type = "C-classification", gamma = 2^(-32:-24),
    cost = 2^(20:25))
plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```




```
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##          gamma    cost
##  1.490116e-08 2097152
##
## - best performance: 0.08272727

tn.sv <- tune.svm(ind ~ ., data = promoters.factor,
    type = "C-classification", gamma = 2^(-26:-20),
```

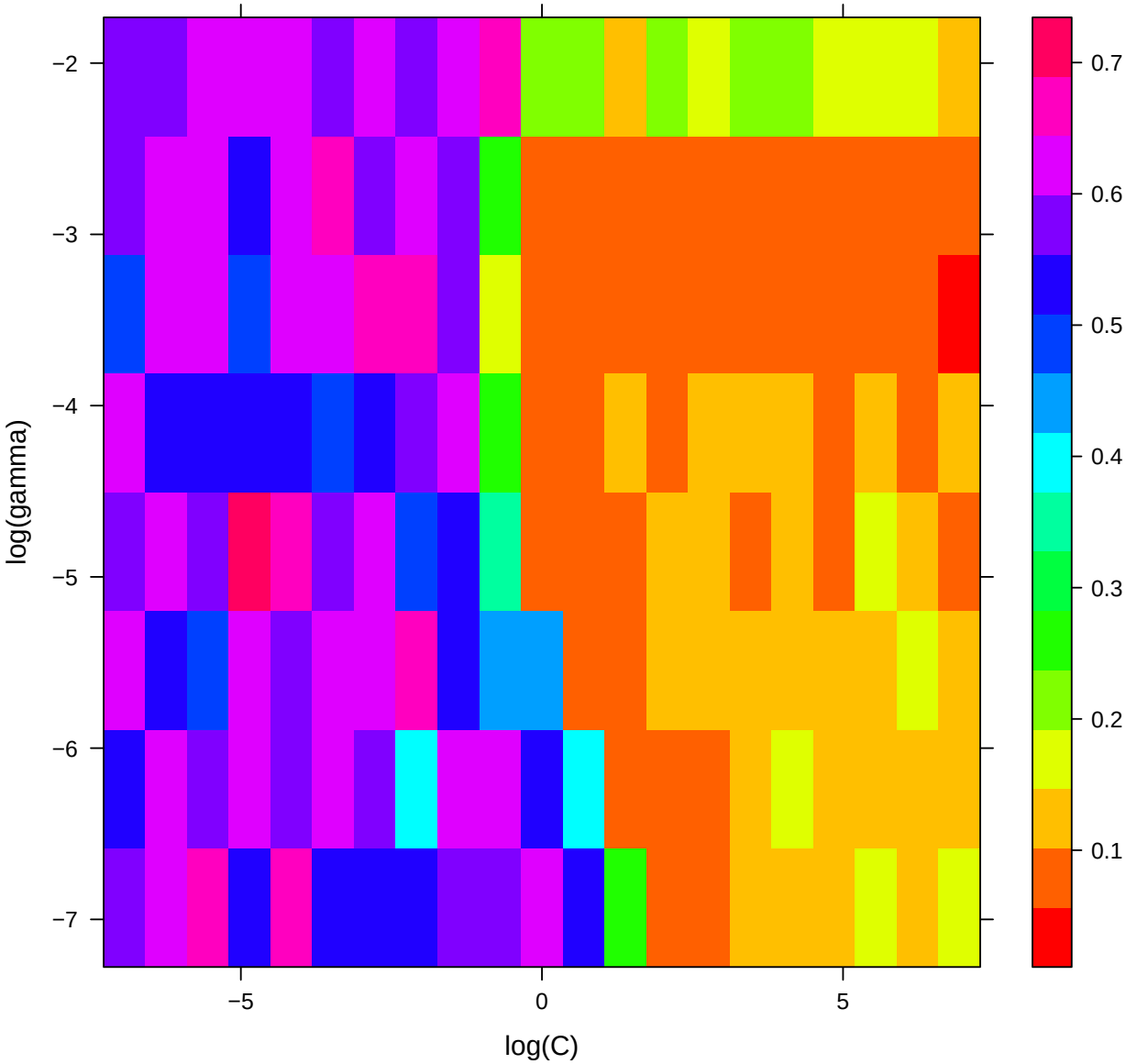
```
    cost = 2^(17:22))
plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```



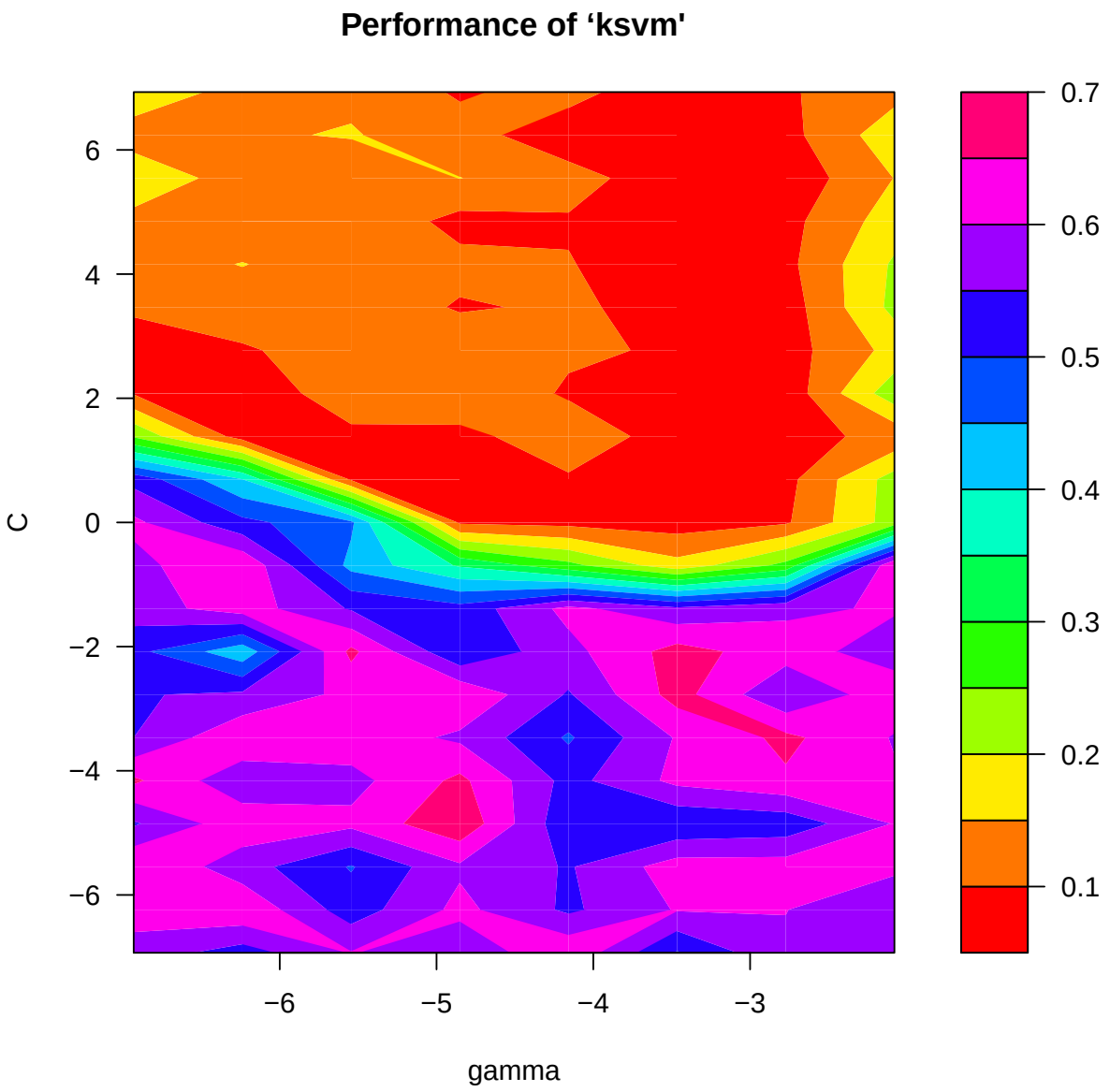

```
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##         gamma    cost
##  2.980232e-08 1048576
##
```

```
## - best performance: 0.05636364
```

```
library(stringdist)
kernel.gaussian.hamming <- function(sigma) {
    res <- function(x, y) exp(-sigma * stringdist(x,
        y, method = "hamming")^2)
    class(res) <- "kernel"
    res
}
kernel.laplace.hamming <- function(sigma) {
    res <- function(x, y) exp(-sigma * stringdist(x,
        y, method = "hamming"))
    class(res) <- "kernel"
    res
}
grid <- expand.grid(gamma = 2^(-10:-3), C = 2^(-10:10))
grid$error <- NA
for (i in seq_len(nrow(grid))) {
    gamma <- grid$gamma[i]
    C <- grid$C[i]
    km <- kernelMatrix(kernel.laplace.hamming(gamma),
        as.list(promoters$nuc))
    sv <- ksvm(km, y = promoters$ind, type = "C-svc",
        C = C, cross = 10)
    grid$error[i] <- sv@cross
}
min(grid$error)

## [1] 0.05545455

i <- which.min(grid$error)
grid$C[i]

## [1] 1024

grid$gamma[i]

## [1] 0.03125

levelplot(error ~ log(C) * log(gamma), data = grid,
    col.regions = rainbow)
```

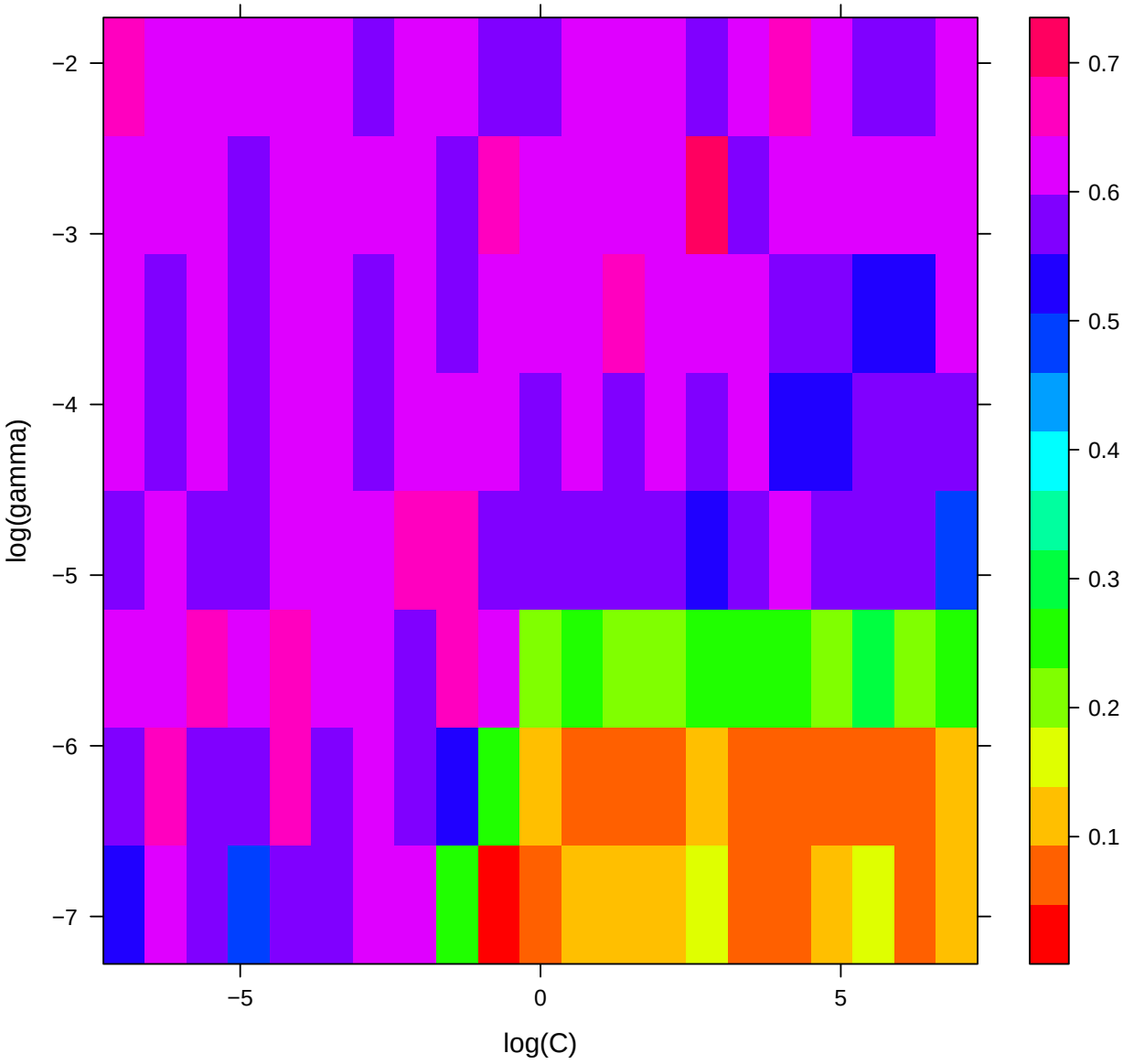



```
# Make fake `tune' object for pretty plot
make.tune <- function(x, method = "") {
    res <- list()
    res$method <- method
    x$dispersion <- 0
    res$performances <- x
    class(res) <- "tune"
    res
}
plot(make.tune(grid, method = "ksvm"), transform.x = log,
    transform.y = log, color.palette = rainbow)
```

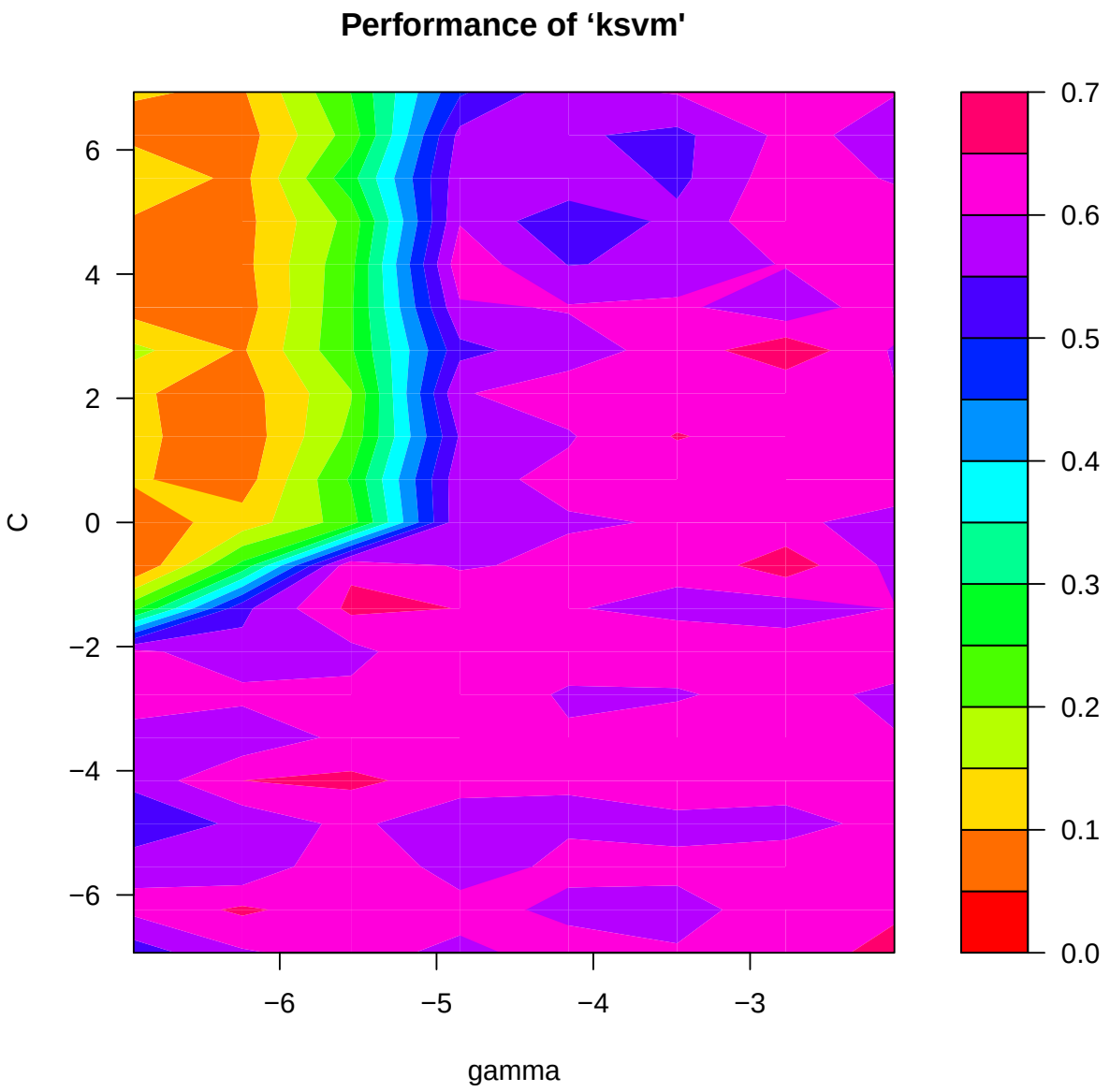



```
grid <- expand.grid(gamma = 2^(-10:-3), C = 2^(-10:10))
grid$error <- NA
for (i in seq_len(nrow(grid))) {
    gamma <- grid$gamma[i]
    C <- grid$C[i]
    km <- kernelMatrix(kernel.gaussian.hamming(gamma),
        as.list(promoters$nuc))
    sv <- ksvm(km, y = promoters$ind, type = "C-svc",
        C = C, cross = 10)
    grid$error[i] <- sv@cross
}
min(grid$error)

## [1] 0.04636364

i <- which.min(grid$error)
```

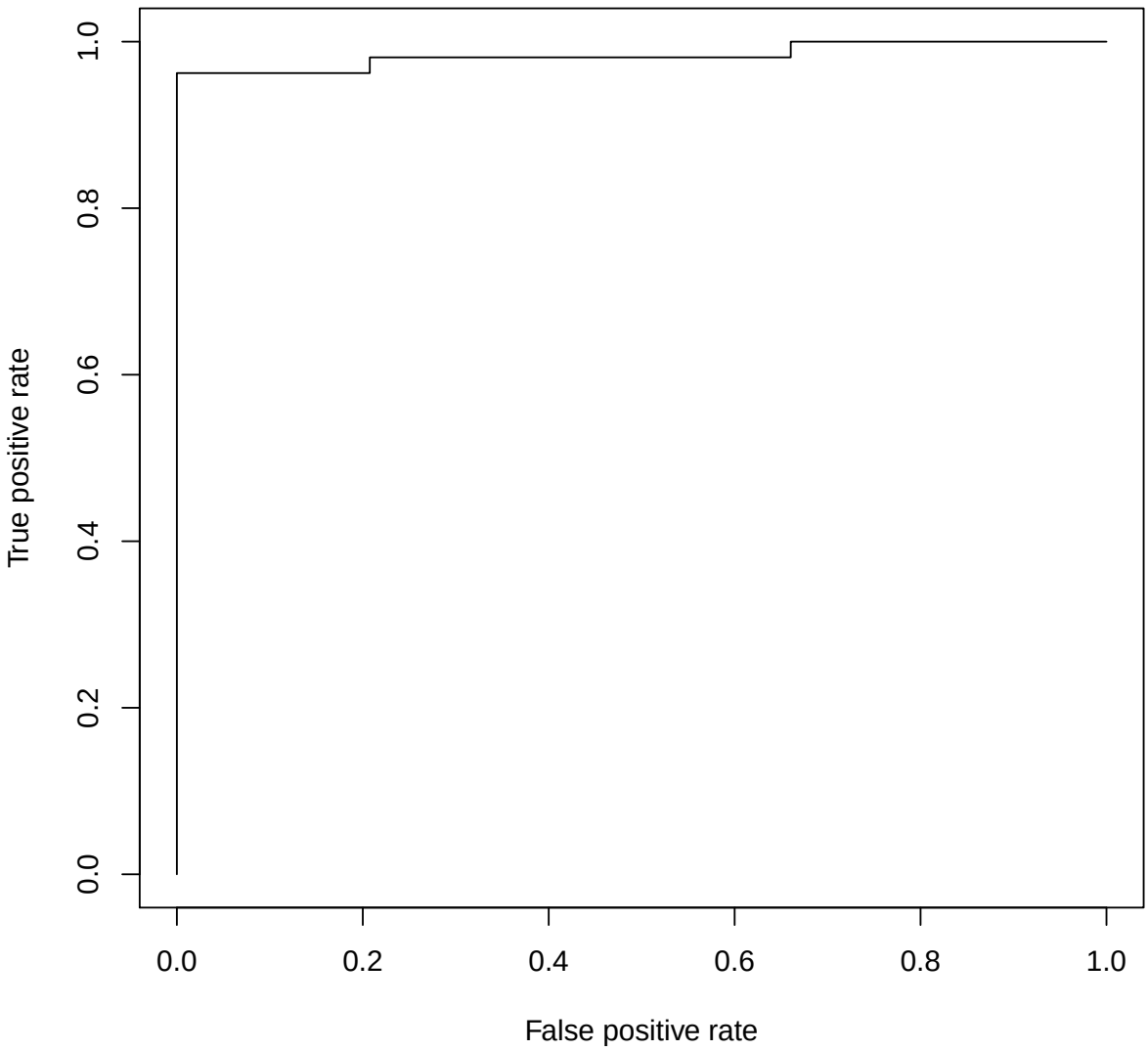
```r
grid$C[i]

## [1] 0.5

grid$gamma[i]

## [1] 0.0009765625

levelplot(error ~ log(C) * log(gamma), data = grid,
    col.regions = rainbow)
```




```r
plot(make.tune(grid, method = "ksvm"), transform.x = log,
    transform.y = log, color.palette = rainbow)
```

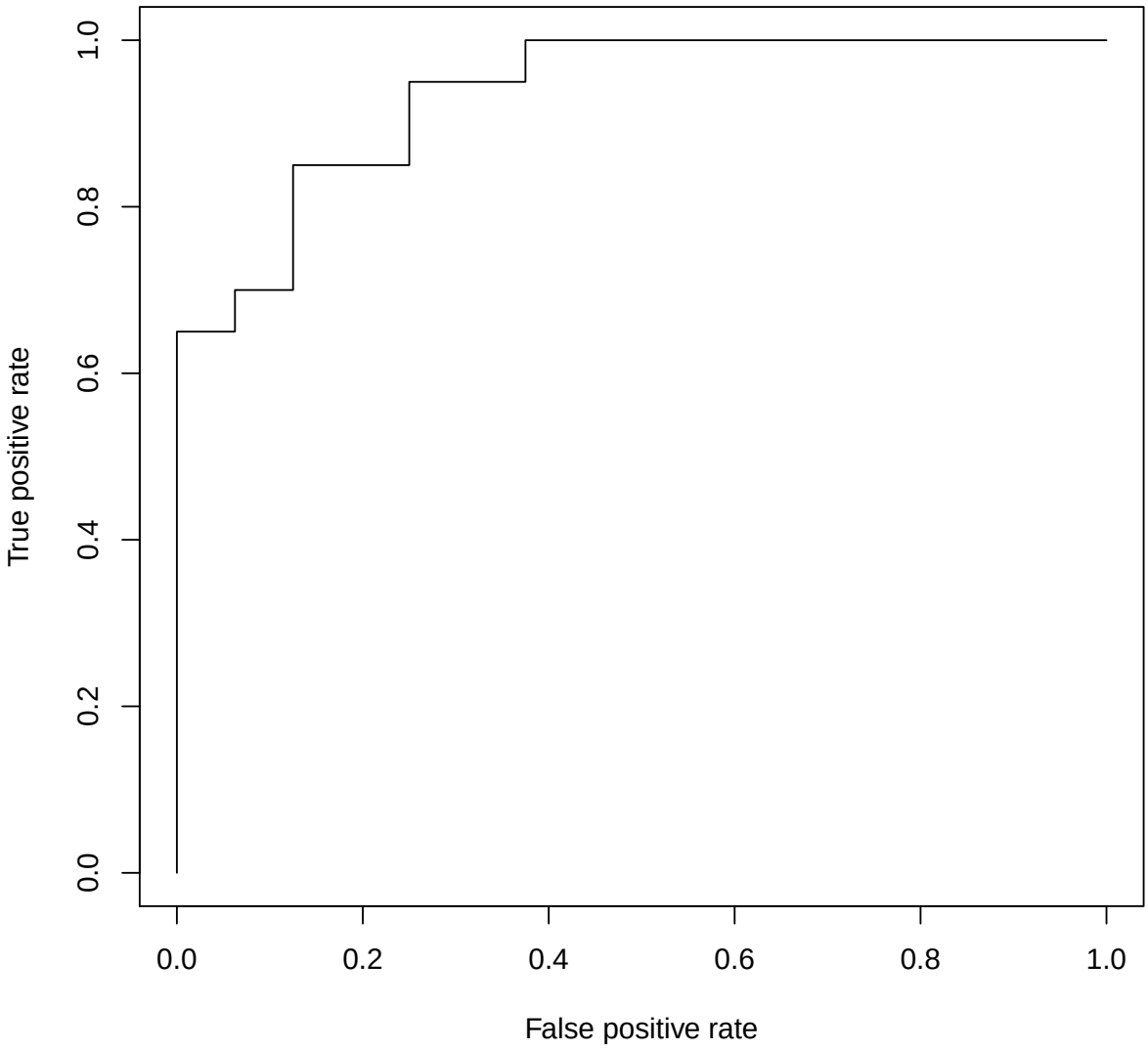



```
library(ROCR)
source("class.R")
i <- which.min(grid$error)
C <- grid$C[i]
gamma <- grid$gamma[i]
km <- kernelMatrix(kernel.gaussian.hamming(gamma),
    as.list(promoters$nuc))
# Add probabilities to model
sv <- ksvm(km, y = promoters$ind, type = "C-svc",
    C = C, prob.model = TRUE)
prob <- predict(sv, km, type = "prob")[,
    2]
pred <- predict(sv, km)
roc <- ROC(pred = prob, promoters$ind)
plot(roc)
```




```
table(actual = promoters$ind, predicted = pred)

##                 predicted
## actual           Non-promoter Promoter
##   Non-promoter             53        0
##   Promoter                  2       51

# Test-train ROC
train.idx <- sample(nrow(promoters), size = 2/3 *
    nrow(promoters))
train <- promoters[train.idx, ]
test <- promoters[-train.idx, ]
km <- kernelMatrix(kernel.gaussian.hamming(gamma),
```

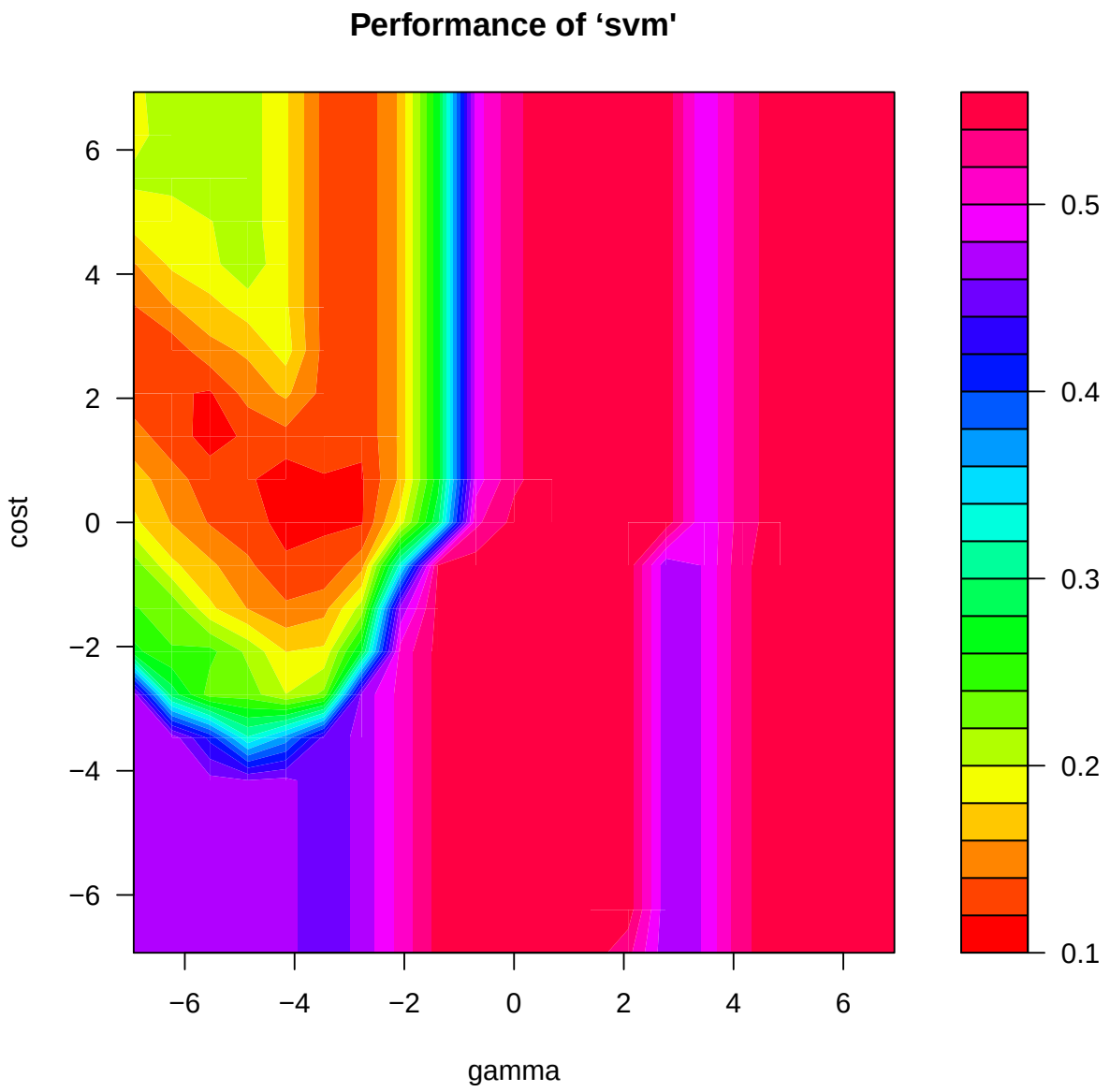
```
    as.list(train$nuc))
sv <- ksvm(km, y = train$ind, type = "C-svc",
    C = C, prob.model = TRUE)
km.test <- kernelMatrix(kernel.gaussian.hamming(gamma),
    as.list(test$nuc), as.list(train$nuc))
prob <- predict(sv, km.test, type = "prob")[,
    2]
pred <- predict(sv, km.test)
roc <- ROC(pred = prob, test$ind)
plot(roc)
```




```
table(actual = test$ind, predicted = pred)

##                 predicted
## actual           Non-promoter Promoter
```

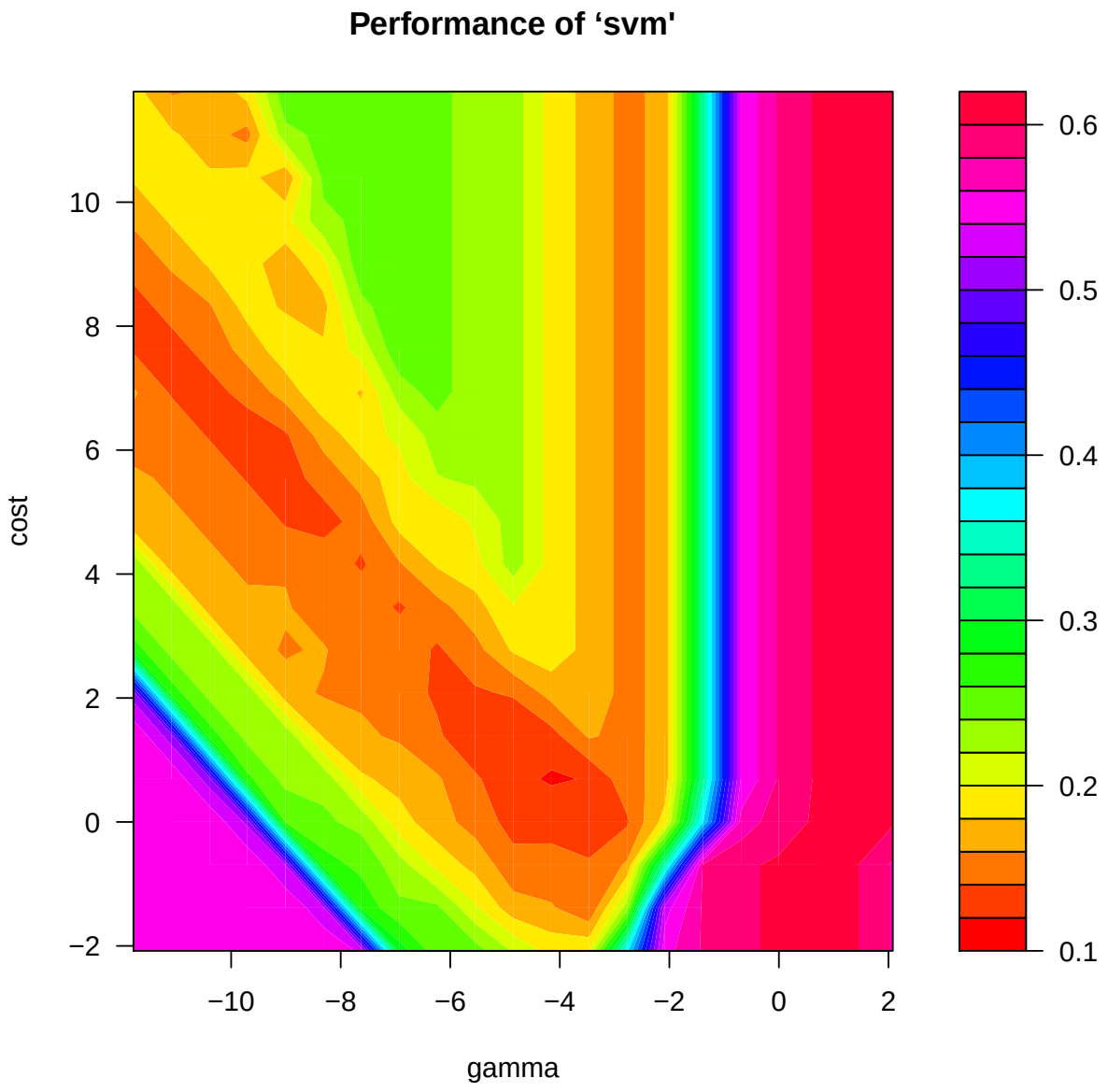
```
##   Non-promoter            16         0
##   Promoter                16         4
```

```
library(e1071)  # svm()
library(kernlab)  # ksvm()
library(klaR)  # svmlight ()
library(svmpath)  # svmpath()
library(MASS)
library(lattice)
library(latticeExtra)
glaucomaM <- read.table("svm/GlaucomaM.txt",
    stringsAsFactors = TRUE, header = TRUE,
    sep = " ", strip.white = TRUE, blank.lines.skip = TRUE)
table(glaucomaM$Class)

##
## glaucoma   normal
##       98       98

tn.sv <- tune.svm(Class ~ ., data = glaucomaM,
    type = "C-classification", cost = 2^(-10:10),
    gamma = 2^(-10:10))
plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```




```
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     gamma cost
##  0.015625    2
##
## - best performance: 0.1078947

tn.sv <- tune.svm(Class ~ ., data = glaucomaM,
    type = "C-classification", cost = 2^(-3:17),
```

```
    gamma = 2^(-17:3))
plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```

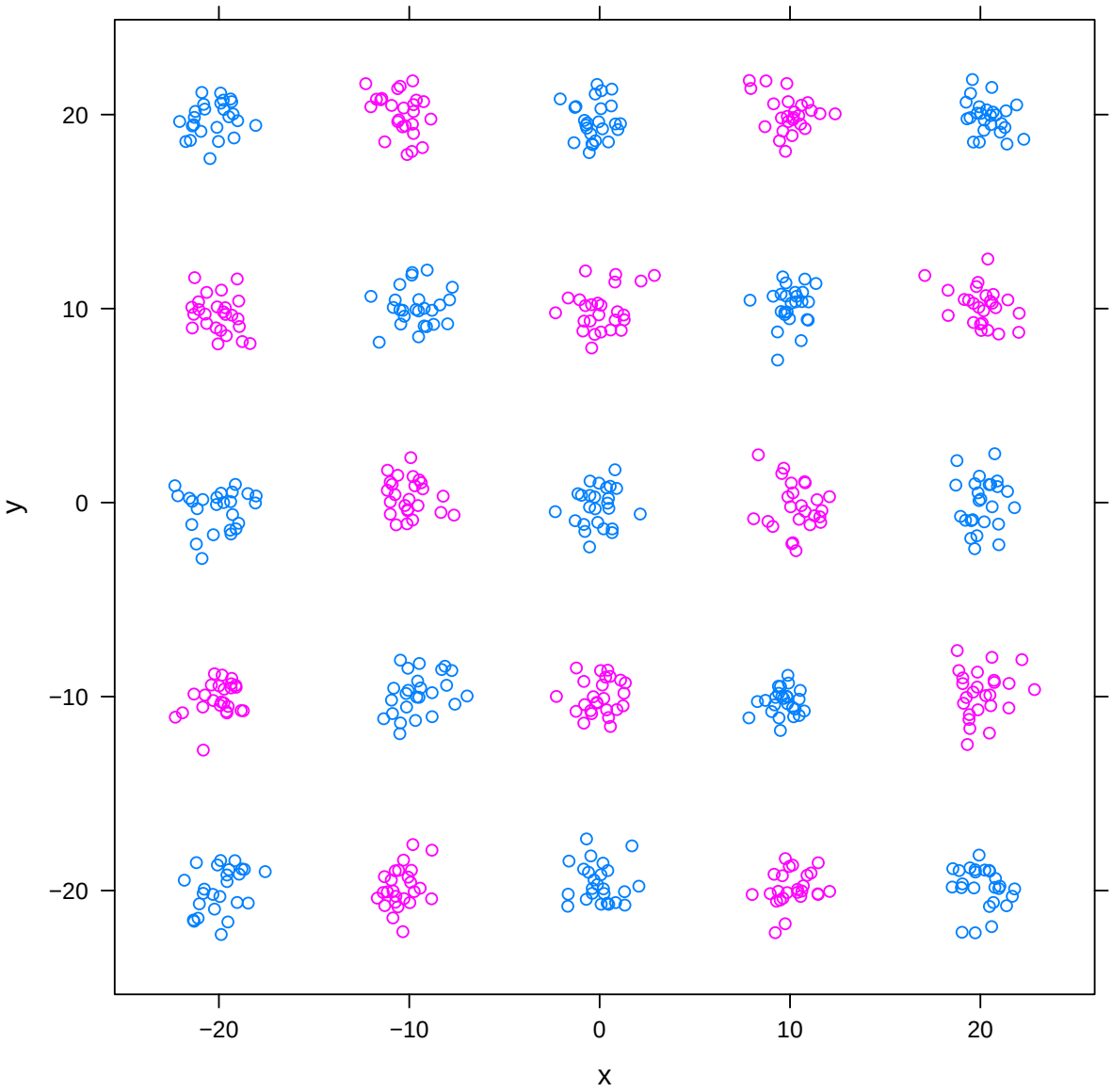



```
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     gamma cost
##  0.015625    2
##
```

```
## - best performance: 0.1168421
```

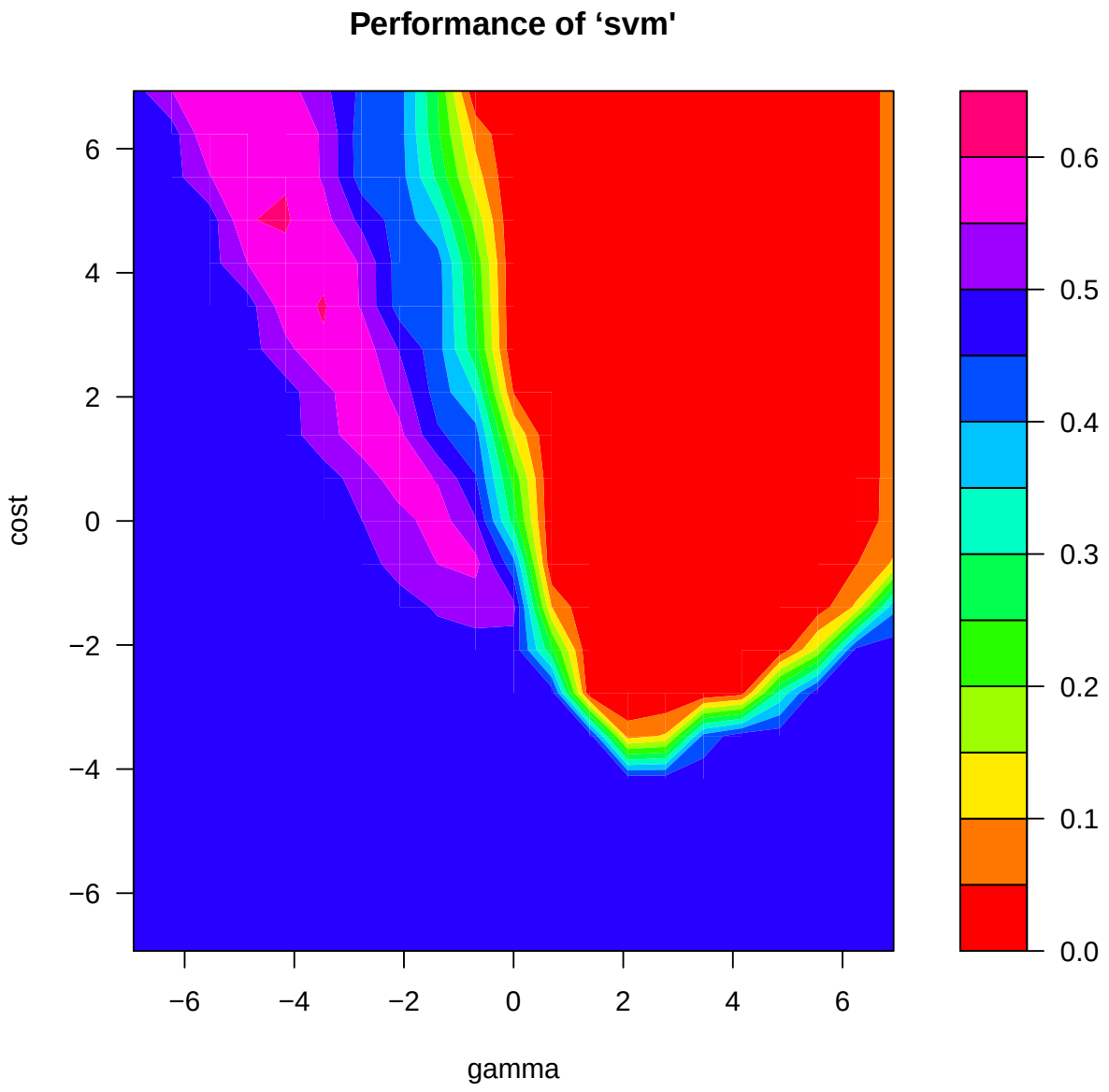
10.1 Модельные примеры SVM, не очень показательные

```
library(e1071)  # svm()
library(kernlab)  # ksvm()
library(MASS)
library(lattice)
library(latticeExtra)
## RBF-SVM is `prototype` method
library(mvtnorm)
grid <- expand.grid(x = c(-20, -10, 0, 10,
    20), y = c(-20, -10, 0, 10, 20))
grid$class <- factor(c("A", "B")[rep(1:2,
    nrow(grid))[1:nrow(grid)]])
N <- 25
df <- lapply(seq_len(nrow(grid)), function(i) {
    df <- rmvnorm(N, c(grid$x[i], grid$y[i]))
    df <- as.data.frame(df)
    names(df) <- c("x", "y")
    df$class <- grid$class[i]
    df
})
df <- do.call(rbind, df)
xyplot(y ~ x, groups = class, data = df)
```



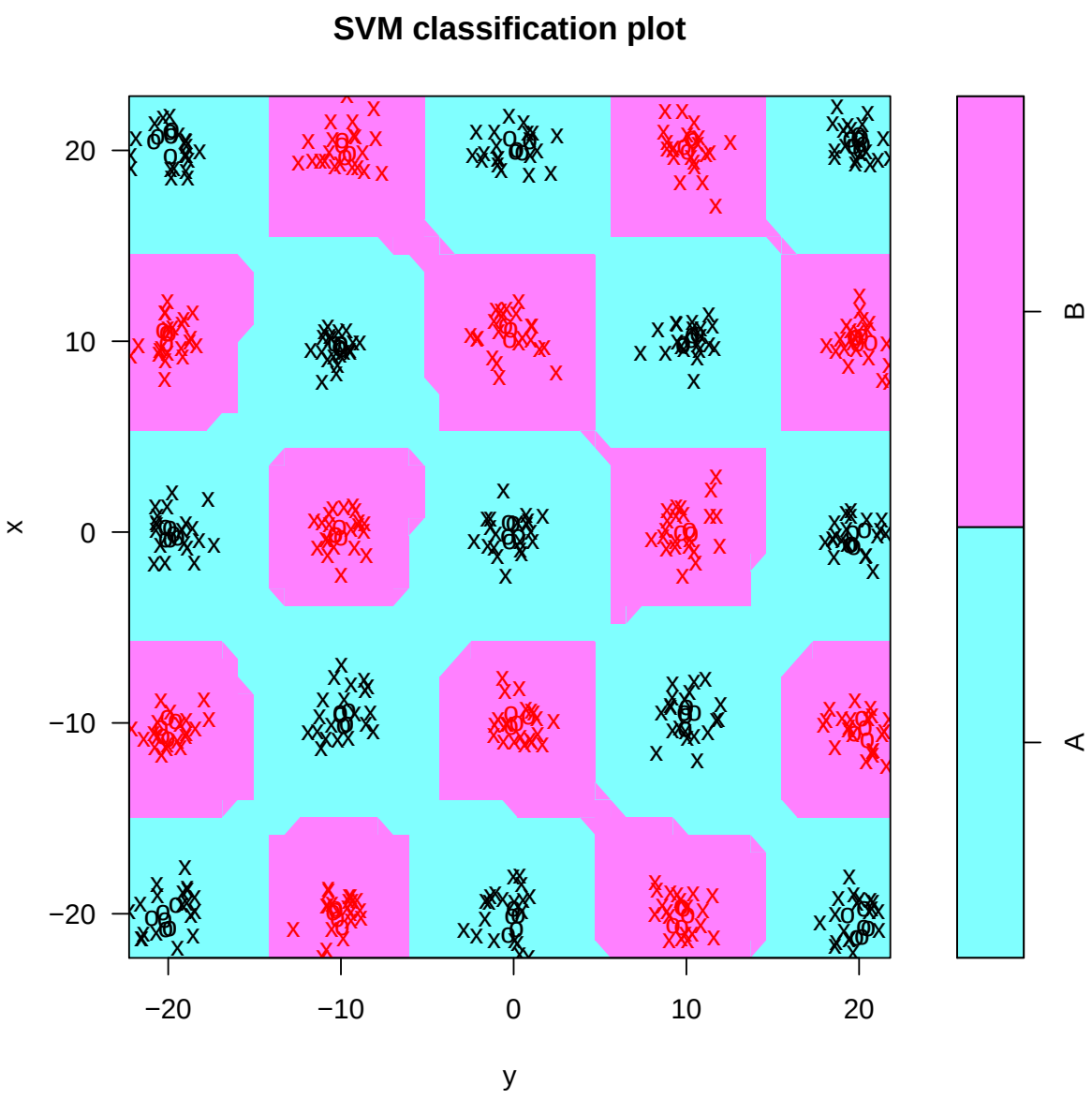

```
tn.sv <- tune.svm(class ~ ., data = df, kernel = "radial",
    cost = 2^(-10:10), gamma = 2^(-10:10),
    tunecontrol = tune.control(cross = 10))
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma   cost
##      8 0.0625
##
## - best performance: 0
```

```r
plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```



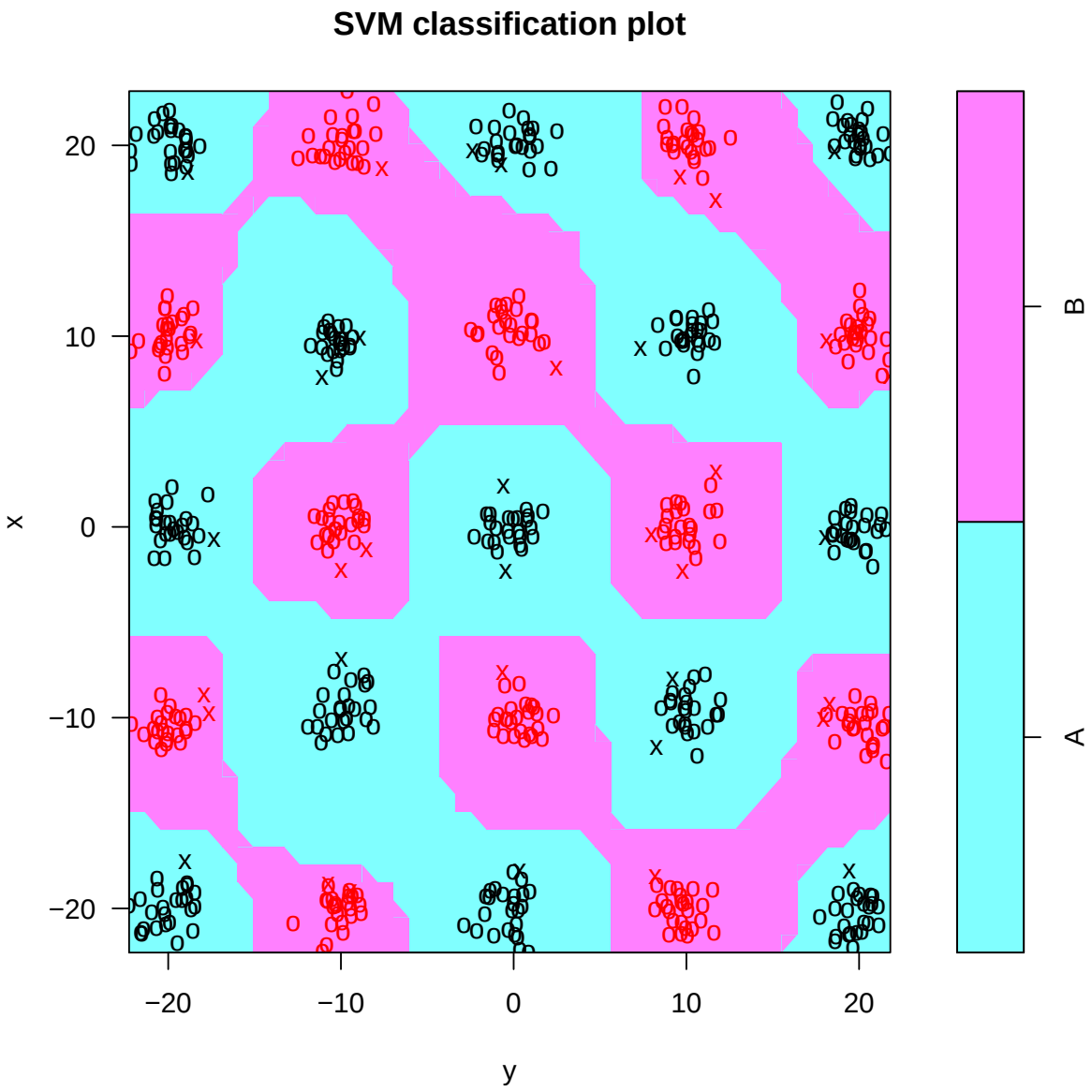

```r
tn.sv$best.model

##
## Call:
## best.svm(x = class ~ ., data = df, gamma = 2^(-10:10),
##     cost = 2^(-10:10), kernel = "radial", tunecontrol = tune.control(cross = 10)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.0625
```

```
##       gamma:  8
##
## Number of Support Vectors:  504

plot(tn.sv$best.model, data = df)
```



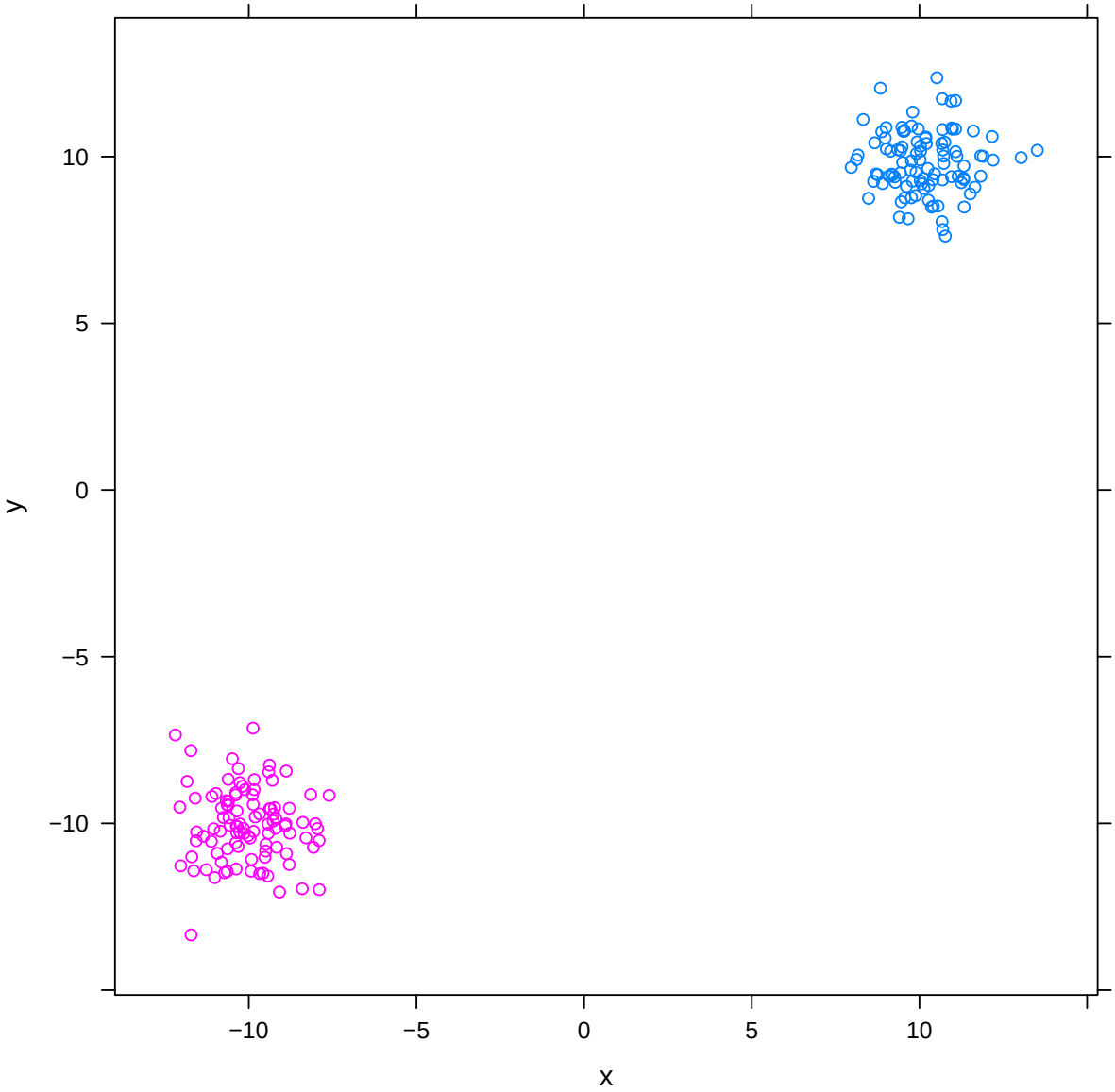

```
sv <- svm(class ~ ., data = df, kernel = "radial",
    cost = 1e+05, gamma = 1)
sv

##
## Call:
## svm(formula = class ~ ., data = df, kernel = "radial",
##     cost = 1e+05, gamma = 1)
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1e+05
##       gamma:  1
##
## Number of Support Vectors:  37

plot(sv, data = df)
```



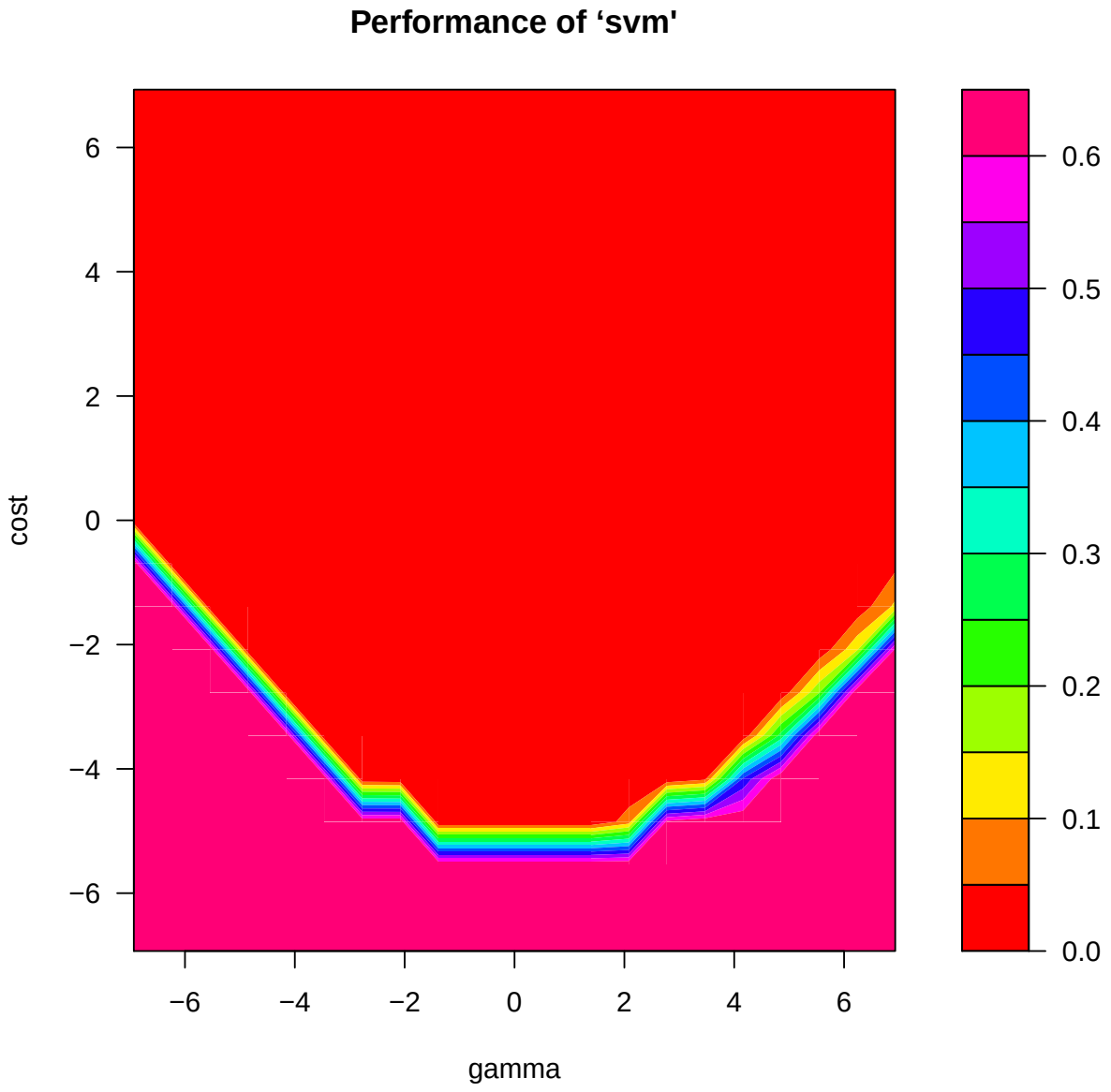

```
N <- 100
df <- rbind(rmvnorm(N, c(10, 10)), rmvnorm(N,
    c(-10, -10)))
df <- as.data.frame(df)
```

```r
names(df) <- c("x", "y")
df$class <- factor(rep(c("A", "B"), each = N))
xyplot(y ~ x, groups = class, data = df)
```




```r
tn.sv <- tune.svm(class ~ ., data = df, kernel = "radial",
    cost = 2^(-10:10), gamma = 2^(-10:10),
    tunecontrol = tune.control(cross = 10))
tn.sv

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
```

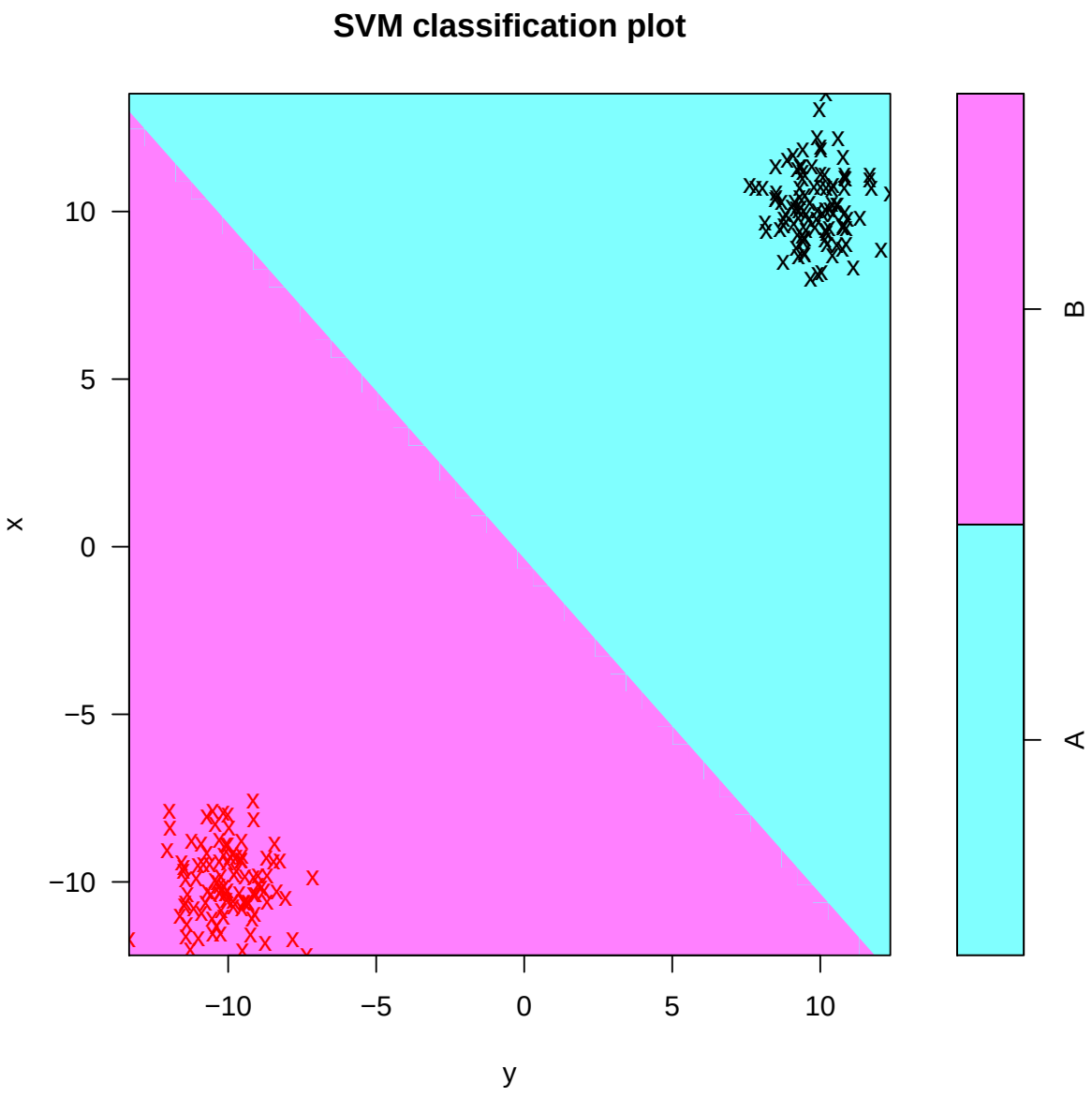
```
##  gamma      cost
##   0.25 0.0078125
##
## - best performance: 0

plot(tn.sv, transform.x = log, transform.y = log,
    color.palette = rainbow)
```




```
tn.sv$best.model

##
## Call:
## best.svm(x = class ~ ., data = df, gamma = 2^(-10:10),
##     cost = 2^(-10:10), kernel = "radial", tunecontrol = tune.control(cross = 10)
##
```

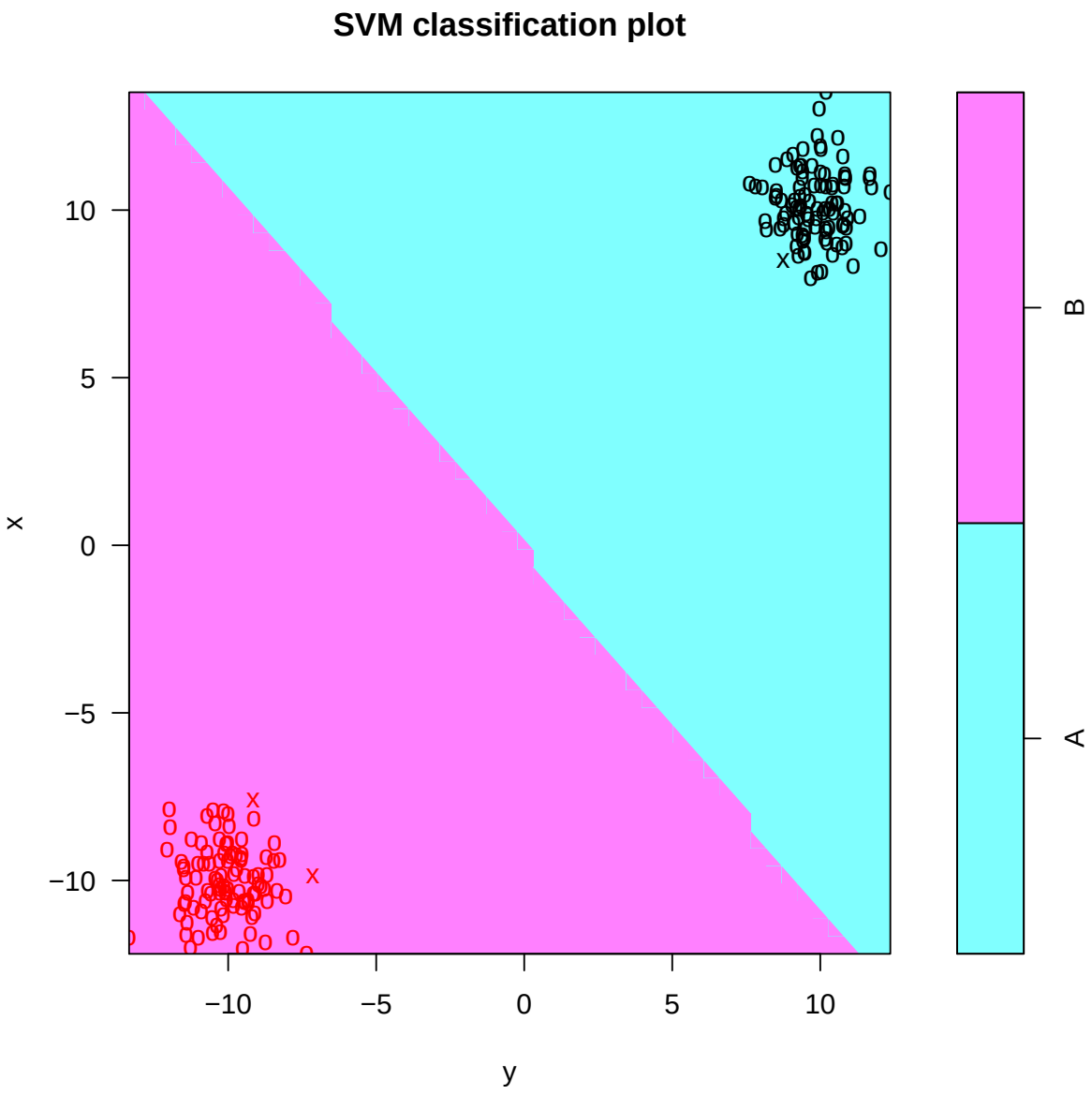
```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.0078125
##       gamma:  0.25
##
## Number of Support Vectors:  200
```

```r
plot(tn.sv$best.model, data = df)
```




```r
sv <- svm(class ~ ., data = df, kernel = "radial",
    cost = 1000, gamma = 0.1)
sv
```

```
##
## Call:
## svm(formula = class ~ ., data = df, kernel = "radial",
##     cost = 1000, gamma = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1000
##       gamma:  0.1
##
## Number of Support Vectors:  3

plot(sv, data = df)
```

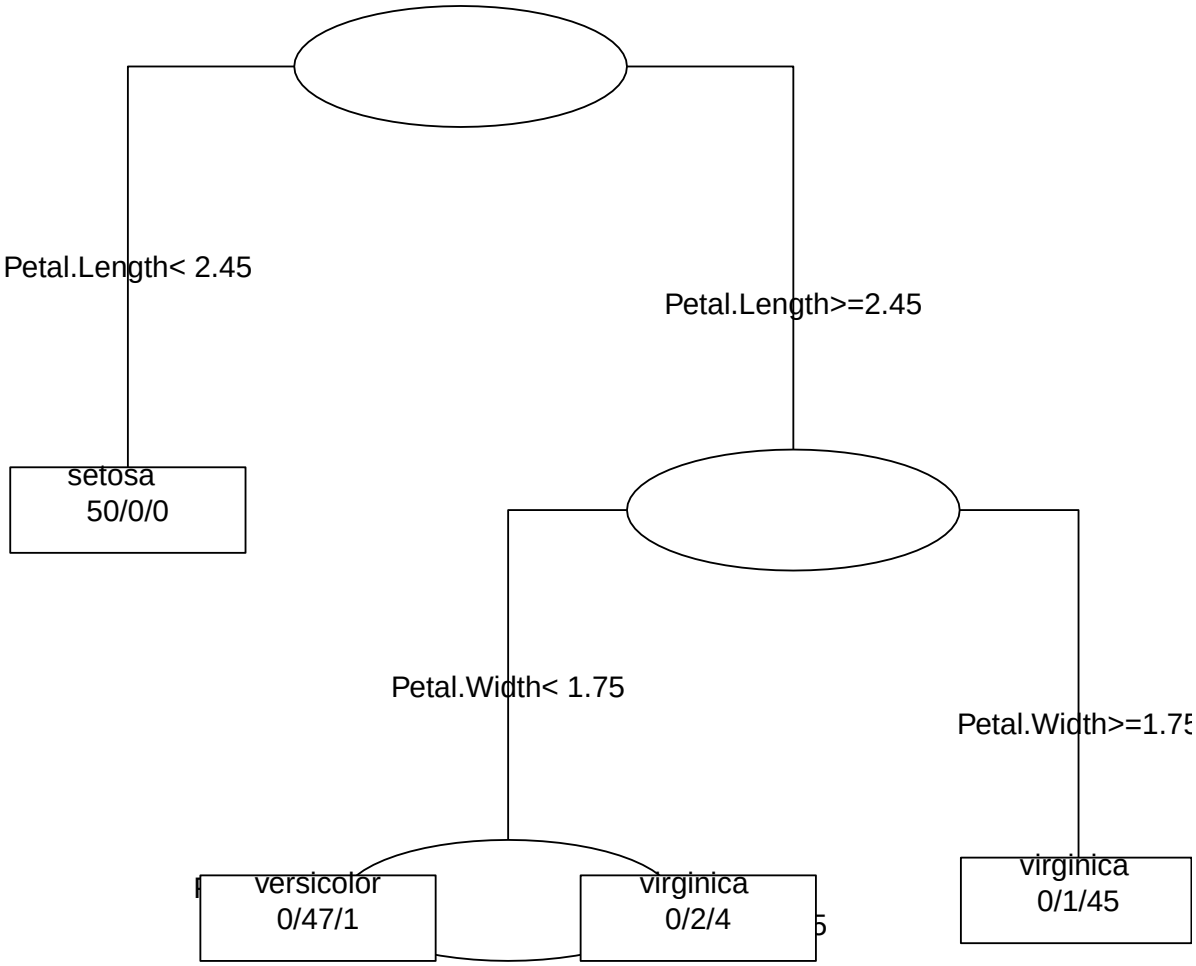


11 Материалы с занятия 5 декабря

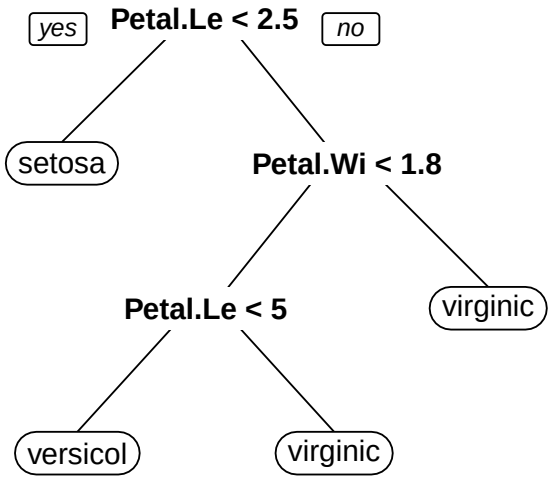
```
read_chunk("rpart/rpart.R")
read_chunk("rpart/rf.R")
```

11.1 CART

```
library(rpart)
library(rpart.plot)
library(party) # http://cran.r-project.org/web/packages/party/vignettes/party.pdf
library(e1071)
library(lattice)
library(latticeExtra)
library(pls)
library(MASS)


# iris!
rm(iris)
data(iris)

rp <- rpart(Species ~ ., data = iris,
            method = "class",
            model = TRUE, # Keep data in result
            parms = list(split = "information"), # or "gini"
            control = rpart.control(cp = 0, # improving treshold
                                    xval = 10, #C V-folds
                                    minsplit = 10,
                                    maxdepth = 30))

par(xpd = TRUE)
plot(rp, compress = TRUE)
text(rp, use.n = TRUE, fancy = TRUE)
```



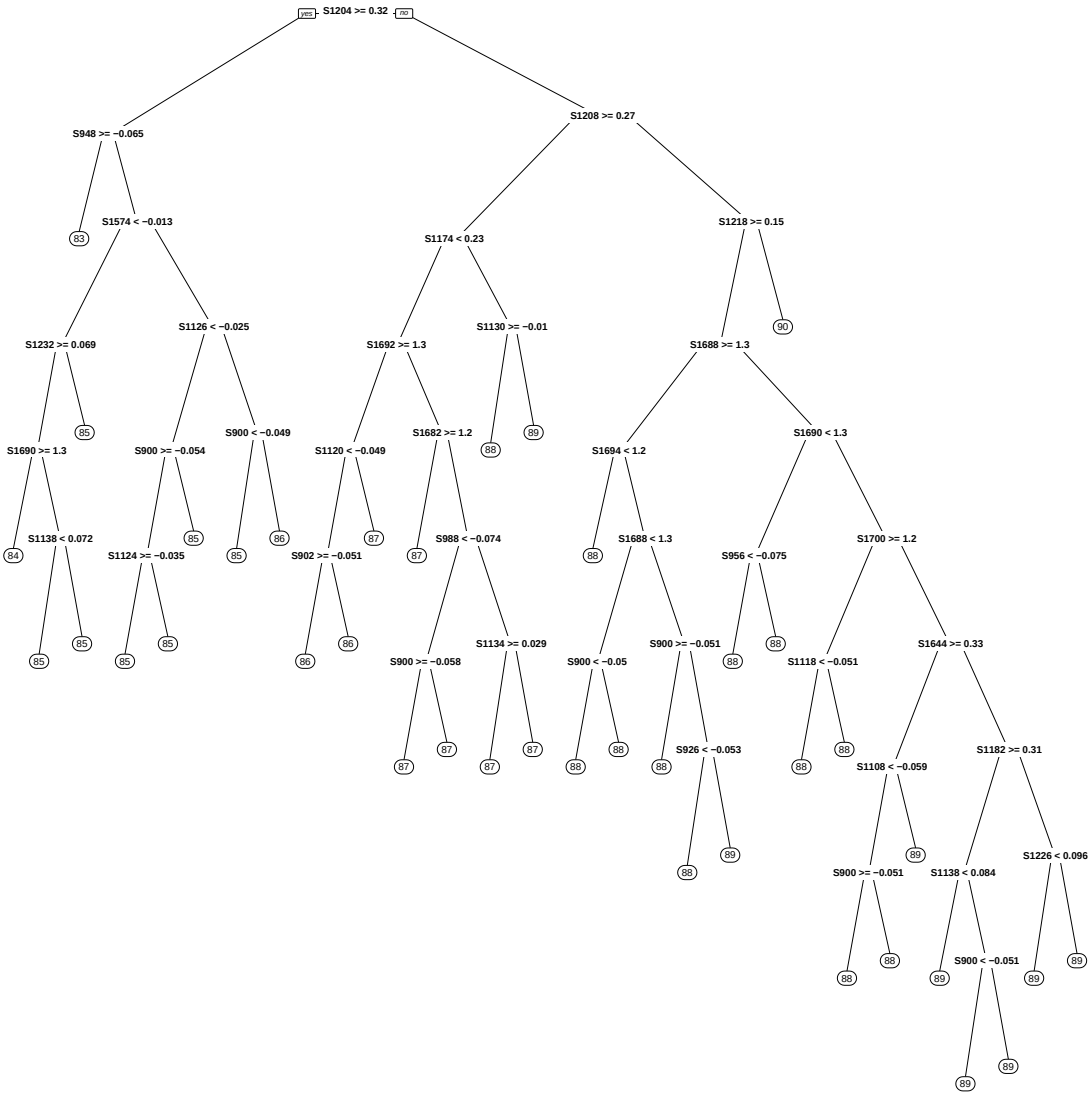

```
rpart.plot(rp)
```




```
rp

## n= 150
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
##    2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
##    3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
##      6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259)
##       12) Petal.Length< 4.95 48   1 versicolor (0.00000000 0.97916667 0.02083333
##       13) Petal.Length>=4.95 6   2 virginica (0.00000000 0.33333333 0.66666667)
##      7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087) *

tn.rp <- tune.rpart(Species ~ ., data = iris,
```

```
                      parms = list(split = "information"))
tn.rp

##
## Error estimation of 'rpart.wrapper' using 10-fold cross validation: 0.05333333
```

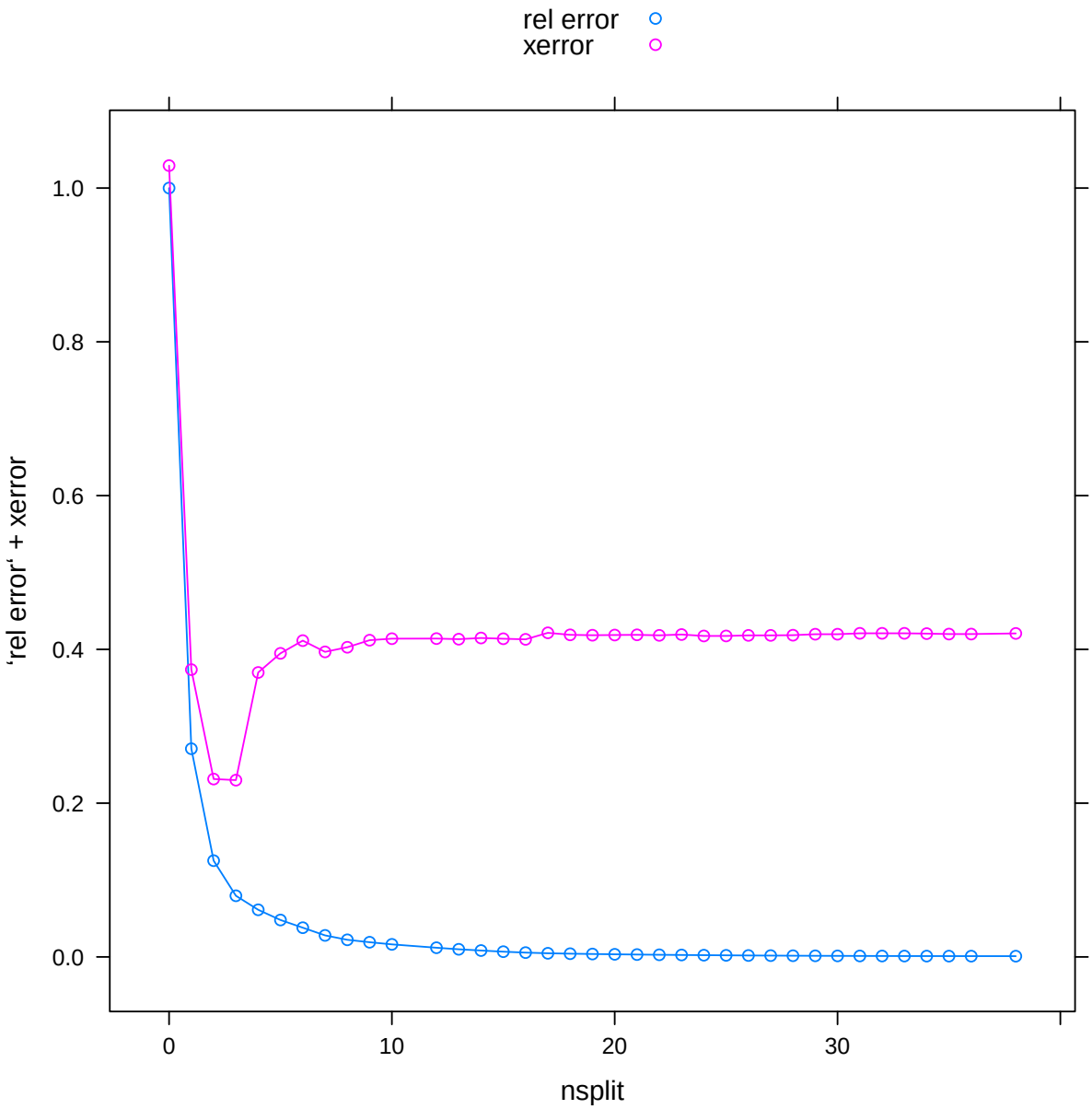
```
# gasoline

data(gasoline)
names(gasoline)

## [1] "octane" "NIR"

dim(gasoline)

## [1] 60  2

class(gasoline$NIR) <- NULL
colnames(gasoline$NIR) <- paste("S", seq(900, 1700, 2), sep = "")
gasoline <- cbind(subset(gasoline, select = octane), as.data.frame(gasoline$NIR))
dim(gasoline)

## [1]  60 402

rp <- rpart(octane ~ ., data = gasoline, model = TRUE,
            method = "anova", # Just Least-squares
            control = rpart.control(cp = 0, xval = 10, minsplit = 3))

rpart.plot(rp)
```



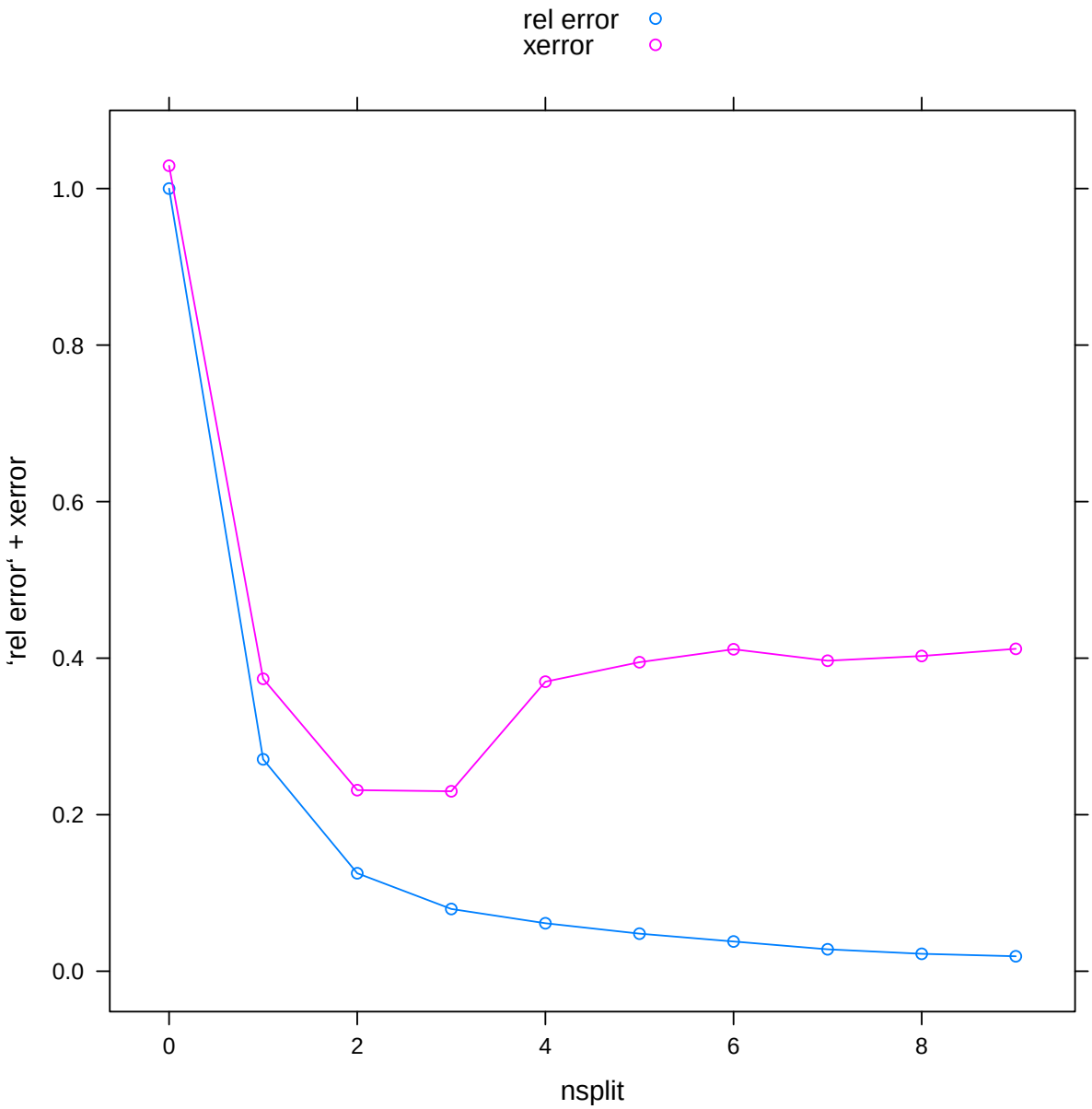

```
tn.rp <- tune.rpart(octane ~ ., data = gasoline, model = TRUE,
                    cp = 0, xval = 0, minsplit = 3,
                    tunecontrol = tune.control(cross = nrow(gasoline)))
tn.rp

##
## Error estimation of 'rpart.wrapper' using leave-one-out: 0.9349271

printcp(rp) -> prp

##
## Regression tree:
## rpart(formula = octane ~ ., data = gasoline, method = "anova",
##     model = TRUE, control = rpart.control(cp = 0, xval = 10,
##         minsplit = 3))
```

```
##
## Variables actually used in tree construction:
##  [1] S1108 S1118 S1120 S1124 S1126 S1130 S1134 S1138 S1174
## [10] S1182 S1204 S1208 S1218 S1226 S1232 S1574 S1644 S1682
## [19] S1688 S1690 S1692 S1694 S1700 S900  S902  S926  S948
## [28] S956  S988
##
## Root node error: 138.13/60 = 2.3021
##
## n= 60
##
##            CP nsplit rel error  xerror     xstd
## 1  7.2922e-01      0 1.0000000 1.02918 0.138669
## 2  1.4558e-01      1 0.2707755 0.37366 0.097719
## 3  4.5653e-02      2 0.1251940 0.23137 0.091674
## 4  1.8237e-02      3 0.0795411 0.22993 0.105107
## 5  1.3284e-02      4 0.0613039 0.36997 0.202777
## 6  1.0011e-02      5 0.0480197 0.39484 0.203846
## 7  9.9909e-03      6 0.0380082 0.41139 0.204395
## 8  5.7519e-03      7 0.0280173 0.39674 0.204757
## 9  3.1464e-03      8 0.0222654 0.40277 0.204673
## 10 2.7966e-03      9 0.0191190 0.41194 0.204491
## 11 2.1802e-03     10 0.0163224 0.41391 0.204439
## 12 2.0342e-03     12 0.0119620 0.41404 0.204441
## 13 1.5958e-03     13 0.0099278 0.41326 0.204461
## 14 1.5083e-03     14 0.0083321 0.41479 0.204423
## 15 1.2120e-03     15 0.0068238 0.41383 0.204454
## 16 8.7178e-04     16 0.0056118 0.41304 0.204477
## 17 5.4449e-04     17 0.0047400 0.42152 0.205117
## 18 4.0784e-04     18 0.0041955 0.41891 0.205175
## 19 3.8008e-04     19 0.0037877 0.41841 0.205183
## 20 3.2579e-04     20 0.0034076 0.41866 0.205171
## 21 3.0165e-04     21 0.0030818 0.41892 0.205135
## 22 2.5490e-04     22 0.0027802 0.41824 0.205146
## 23 2.4434e-04     23 0.0025253 0.41931 0.205261
## 24 2.2064e-04     24 0.0022809 0.41745 0.205115
## 25 1.9306e-04     25 0.0020603 0.41746 0.205114
## 26 1.6289e-04     26 0.0018672 0.41820 0.205092
## 27 1.4781e-04     27 0.0017043 0.41820 0.205092
## 28 1.0860e-04     28 0.0015565 0.41846 0.205089
## 29 9.6529e-05     29 0.0014479 0.41972 0.204955
## 30 7.5414e-05     30 0.0013514 0.41972 0.204955
## 31 7.5414e-05     31 0.0012760 0.42090 0.204942
## 32 7.2397e-05     32 0.0012006 0.42090 0.204942
## 33 4.8265e-05     33 0.0011282 0.42090 0.204942
## 34 2.7149e-05     34 0.0010799 0.42050 0.204947
## 35 2.4132e-05     35 0.0010528 0.41992 0.204961
## 36 1.2066e-05     36 0.0010286 0.41992 0.204961
```

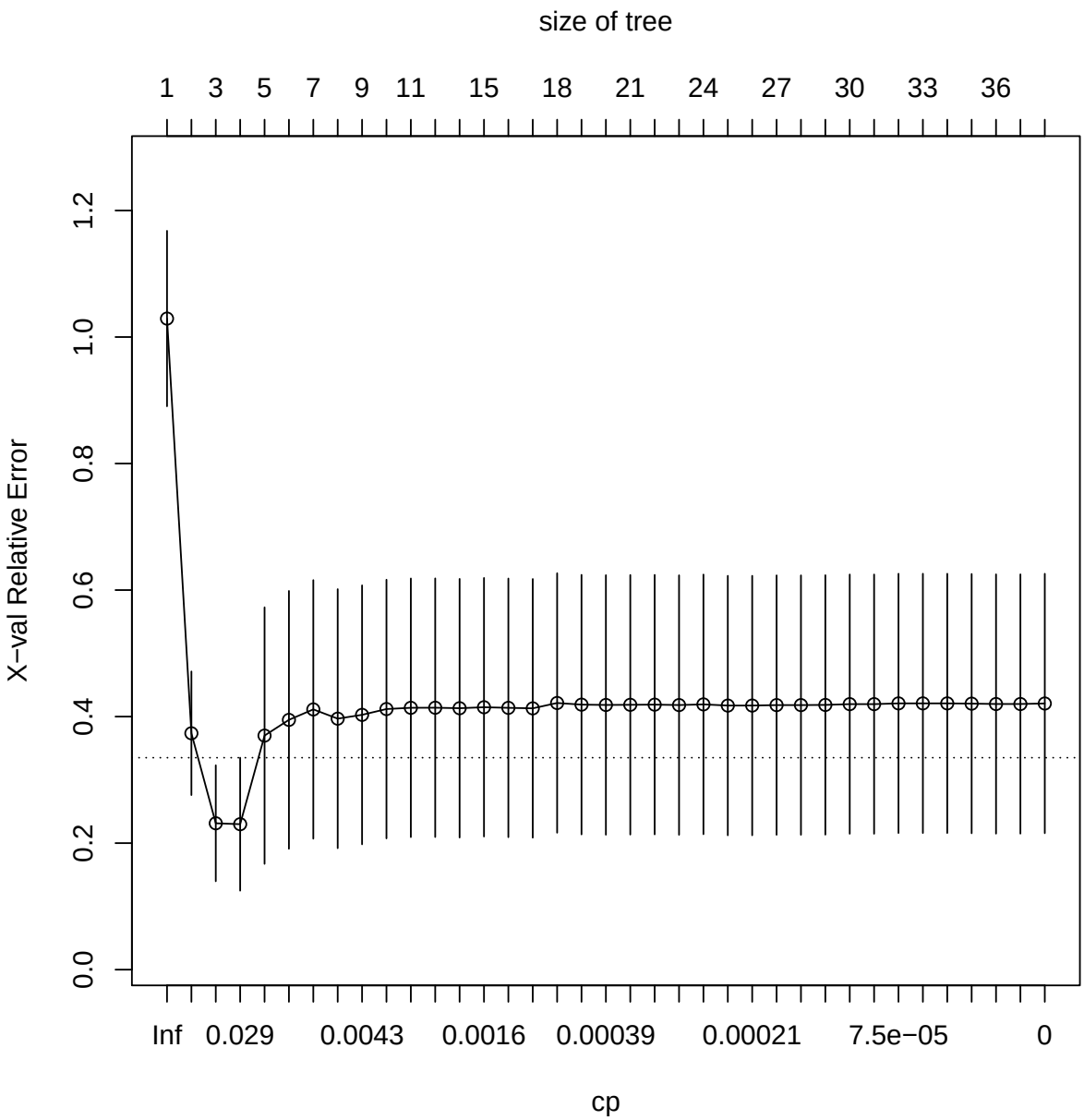
```
## 37 0.0000e+00     38 0.0010045 0.42073 0.205047

prp <- as.data.frame(prp)
xyplot(`rel error` + xerror ~ nsplit, data = prp, type = "b",
       auto.key = TRUE)
```



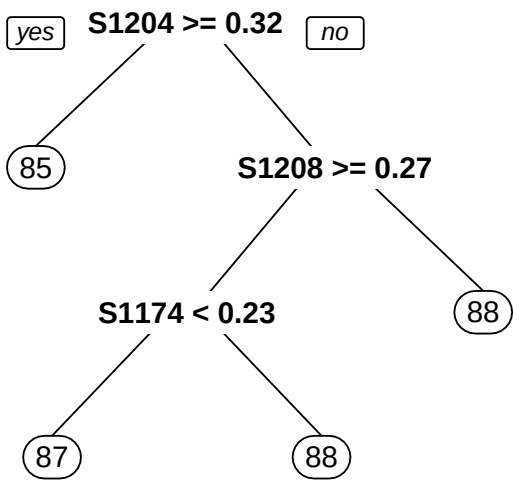

```
xyplot(`rel error` + xerror ~ nsplit, data = prp[1:10, ], type = "b",
       auto.key = TRUE)
```


rel error
xerror
1.0
0.8
0.6
0.4
0.2
0.0
‘rel error’ + xerror
0
2
4
6
8
nsplit

```
plotcp(rp)
```

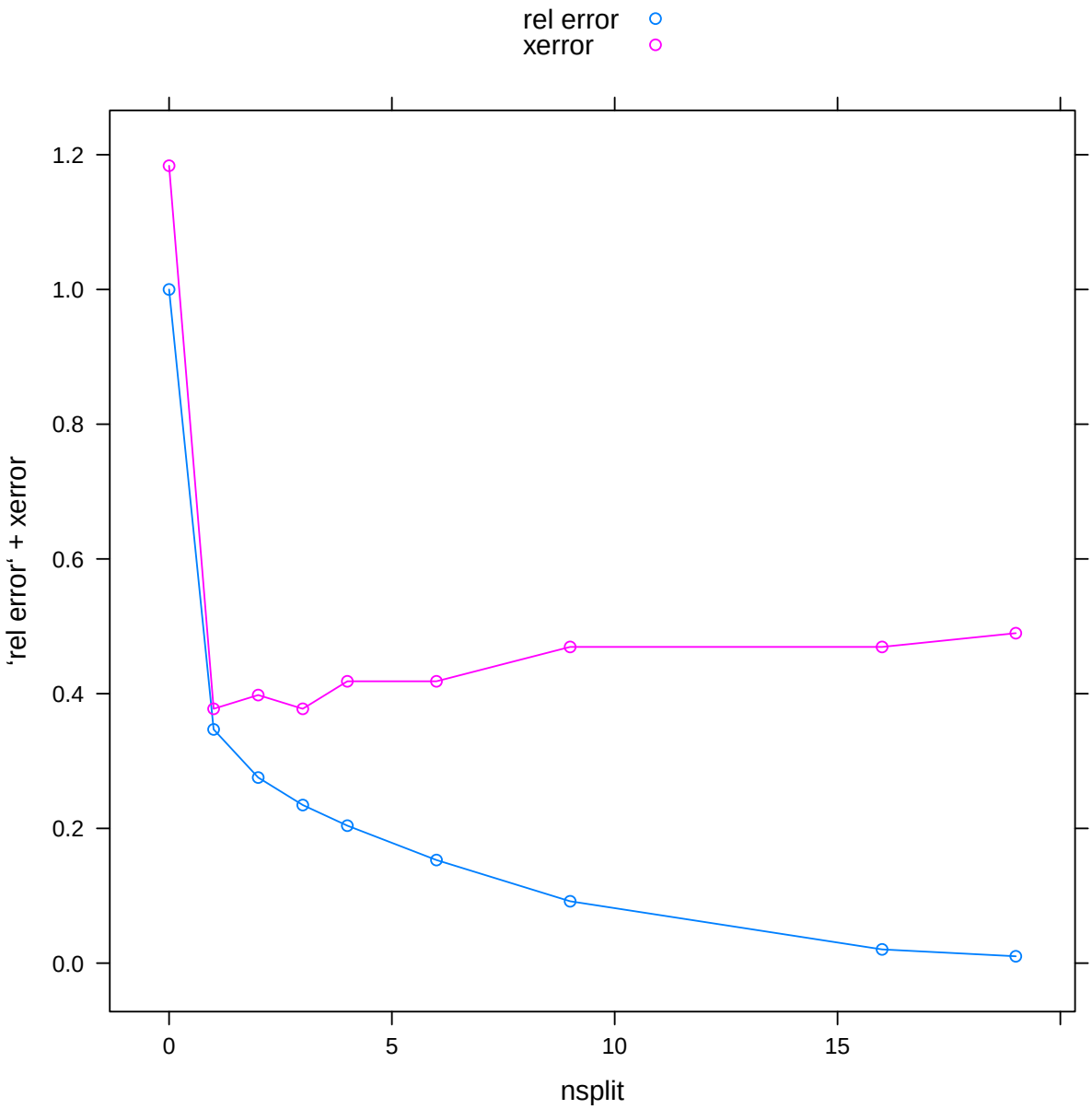



```
cp <- 0.02

tn.rp <- tune.rpart(octane ~ ., data = gasoline, model = TRUE,
                    cp = cp, xval = 0, minsplit = 3,
                    tunecontrol = tune.control(cross = nrow(gasoline)))
tn.rp

##
## Error estimation of 'rpart.wrapper' using leave-one-out: 0.5400096

rpart.plot(tn.rp$best.model)
```

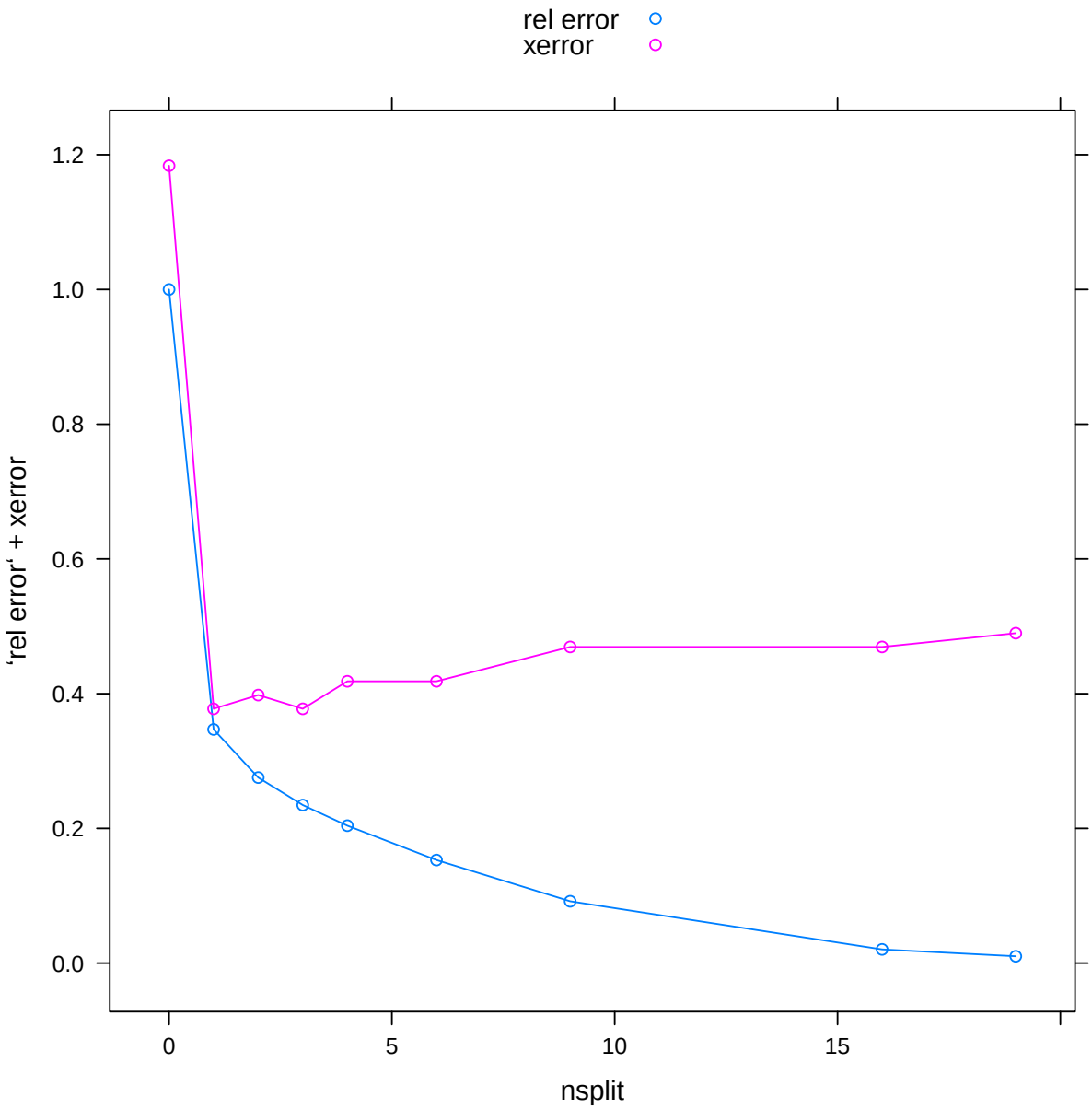



```
# GlaucomaM
glaucomaM <- read.table("svm/GlaucomaM.txt",
    stringsAsFactors = TRUE, header = TRUE,
    sep = " ", strip.white = TRUE, blank.lines.skip = TRUE)
table(glaucomaM$Class)

##
## glaucoma   normal
##       98       98

rp <- rpart(Class ~ ., data = glaucomaM,
    model = TRUE, method = "class", control = rpart.control(cp = 0,
        xval = 10, minbucket = 1))
prp <- printcp(rp)
```

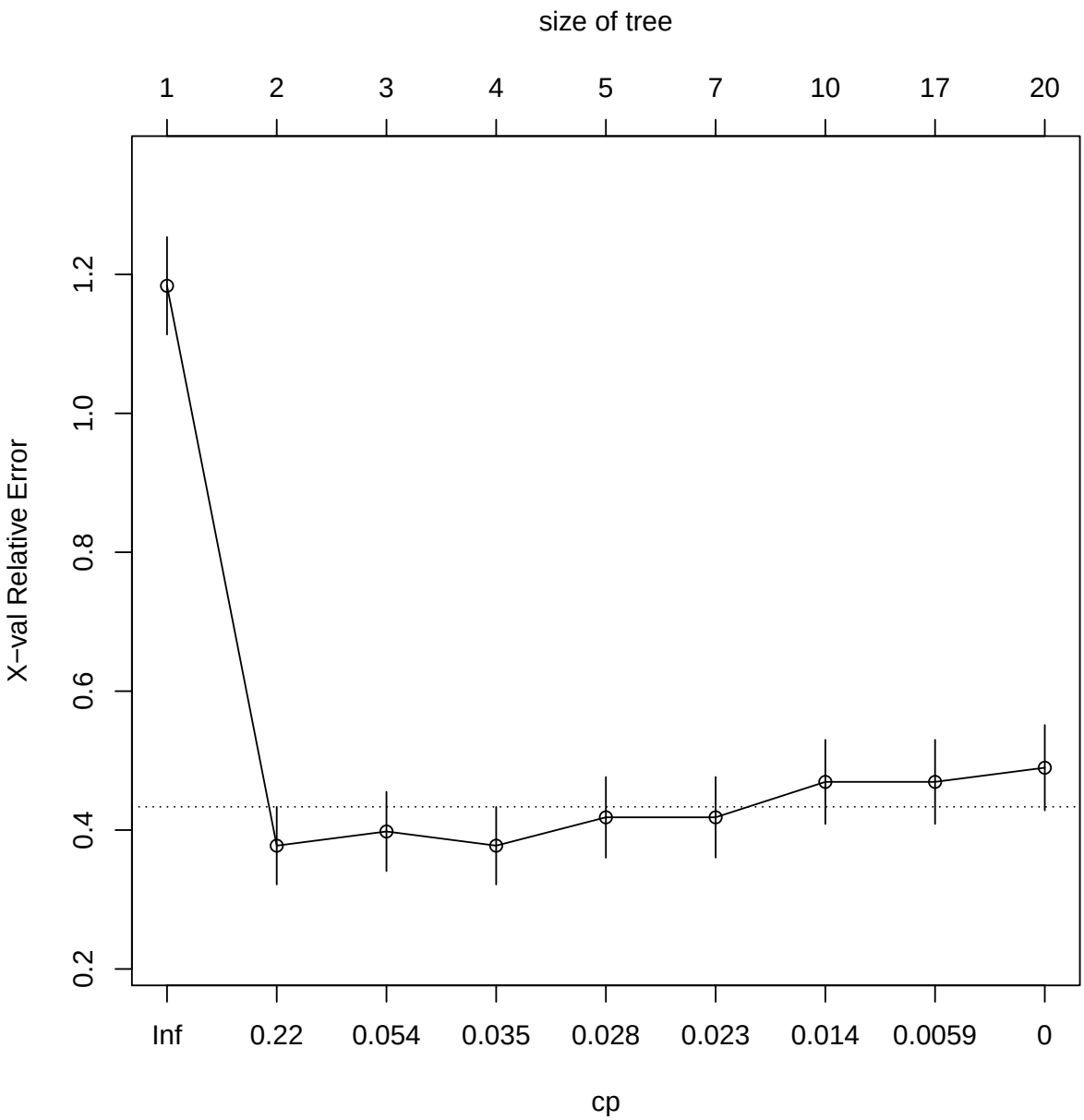
```
##
## Classification tree:
## rpart(formula = Class ~ ., data = glaucomaM, method = "class",
##     model = TRUE, control = rpart.control(cp = 0, xval = 10,
##         minbucket = 1))
##
## Variables actually used in tree construction:
##  [1] abri ag   as   at   eai  eas  hvc  mdi  mdn  mhcg mhci
## [12] mv   tmn  tms  varg vars vass
##
## Root node error: 98/196 = 0.5
##
## n= 196
##
##          CP nsplit rel error  xerror     xstd
## 1 0.6530612      0  1.000000 1.18367 0.070213
## 2 0.0714286      1  0.346939 0.37755 0.055904
## 3 0.0408163      2  0.275510 0.39796 0.057033
## 4 0.0306122      3  0.234694 0.37755 0.055904
## 5 0.0255102      4  0.204082 0.41837 0.058104
## 6 0.0204082      6  0.153061 0.41837 0.058104
## 7 0.0102041      9  0.091837 0.46939 0.060544
## 8 0.0034014     16  0.020408 0.46939 0.060544
## 9 0.0000000     19  0.010204 0.48980 0.061432

prp <- as.data.frame(prp)
xyplot(`rel error` + xerror ~ nsplit, data = prp,
    type = "b", auto.key = TRUE)
```



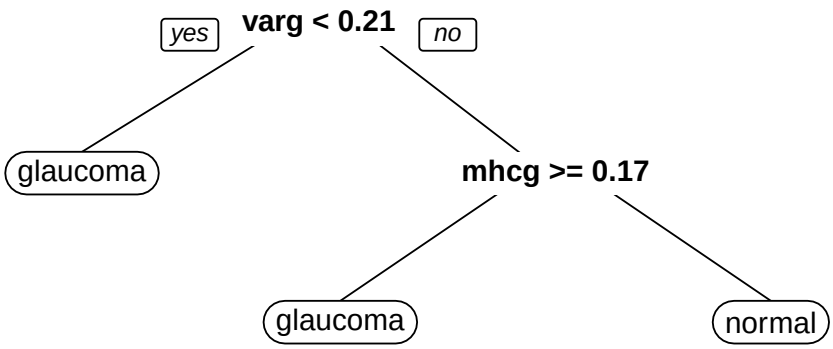

```
xyplot(`rel error` + xerror ~ nsplit, data = prp[1:10,
    ], type = "b", auto.key = TRUE)
```


rel error
xerror
1.2
1.0
0.8
0.6
0.4
0.2
0.0
'rel error' + xerror
0
5
10
15
nsplit

```
plotcp(rp)
```




```
cp <- 0.05
tn.rp <- tune.rpart(Class ~ ., data = glaucomaM,
    model = TRUE, cp = cp, xval = 0, minsplit = 3,
    tunecontrol = tune.control(cross = nrow(glaucomaM)))
tn.rp

##
## Error estimation of 'rpart.wrapper' using leave-one-out: 0.1734694

rpart.plot(tn.rp$best.model)
```

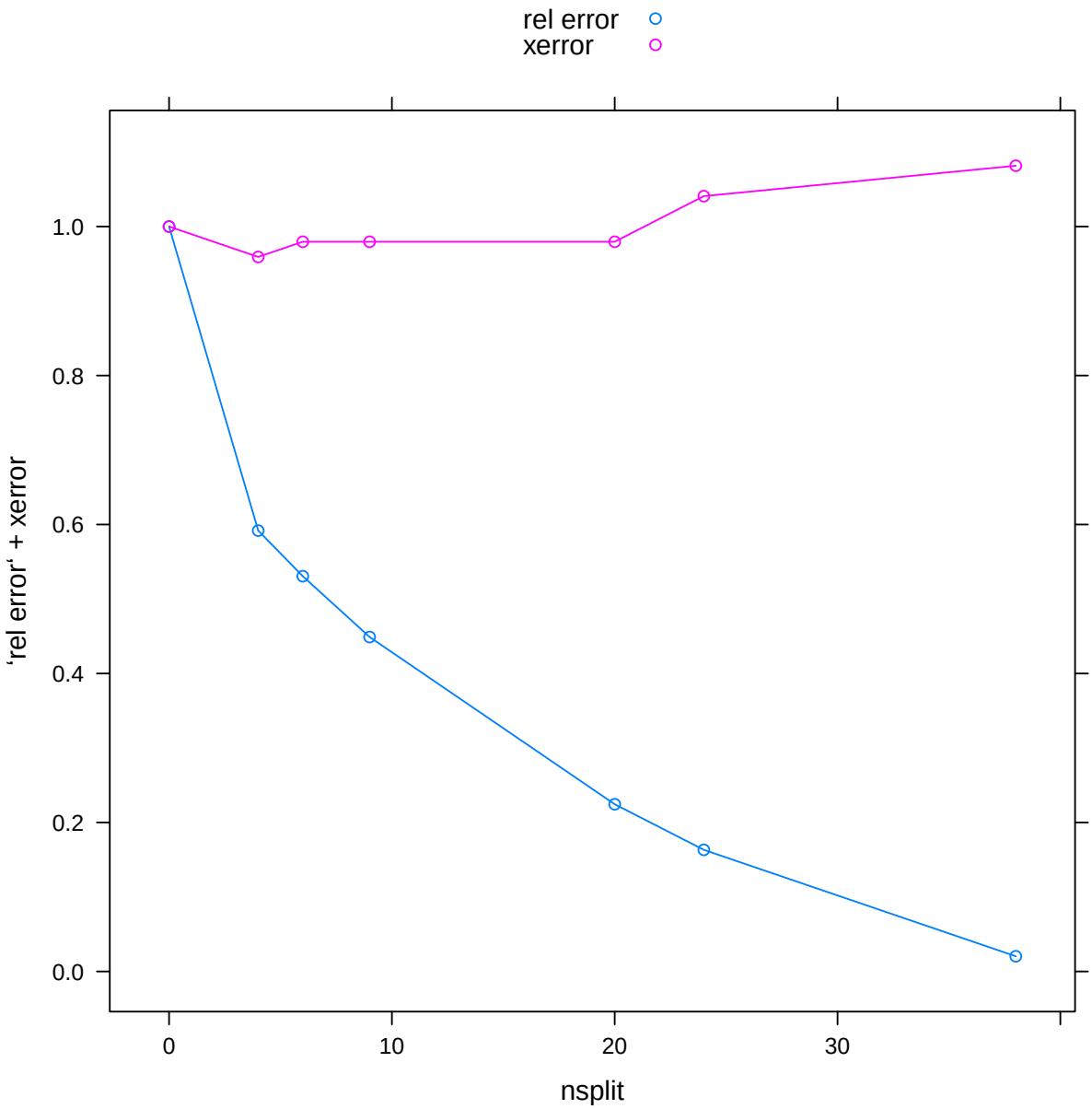



```
summary(tn.rp$best.model)

## Call:
## best.rpart(formula = Class ~ ., data = glaucomaM, minsplit = 3,
##     cp = cp, xval = 0, model = TRUE, tunecontrol = tune.control(cross = nrow(gla
##   n= 196
##
##           CP nsplit rel error
## 1 0.65306122      0 1.0000000
## 2 0.07142857      1 0.3469388
## 3 0.05000000      2 0.2755102
##
## Variable importance
## varg varn vari vars  rnf vbri mhcg mhcn mhcs phcn mhct phcg
##   19   16   15   14   13   11    4    2    2    2    1    1
```

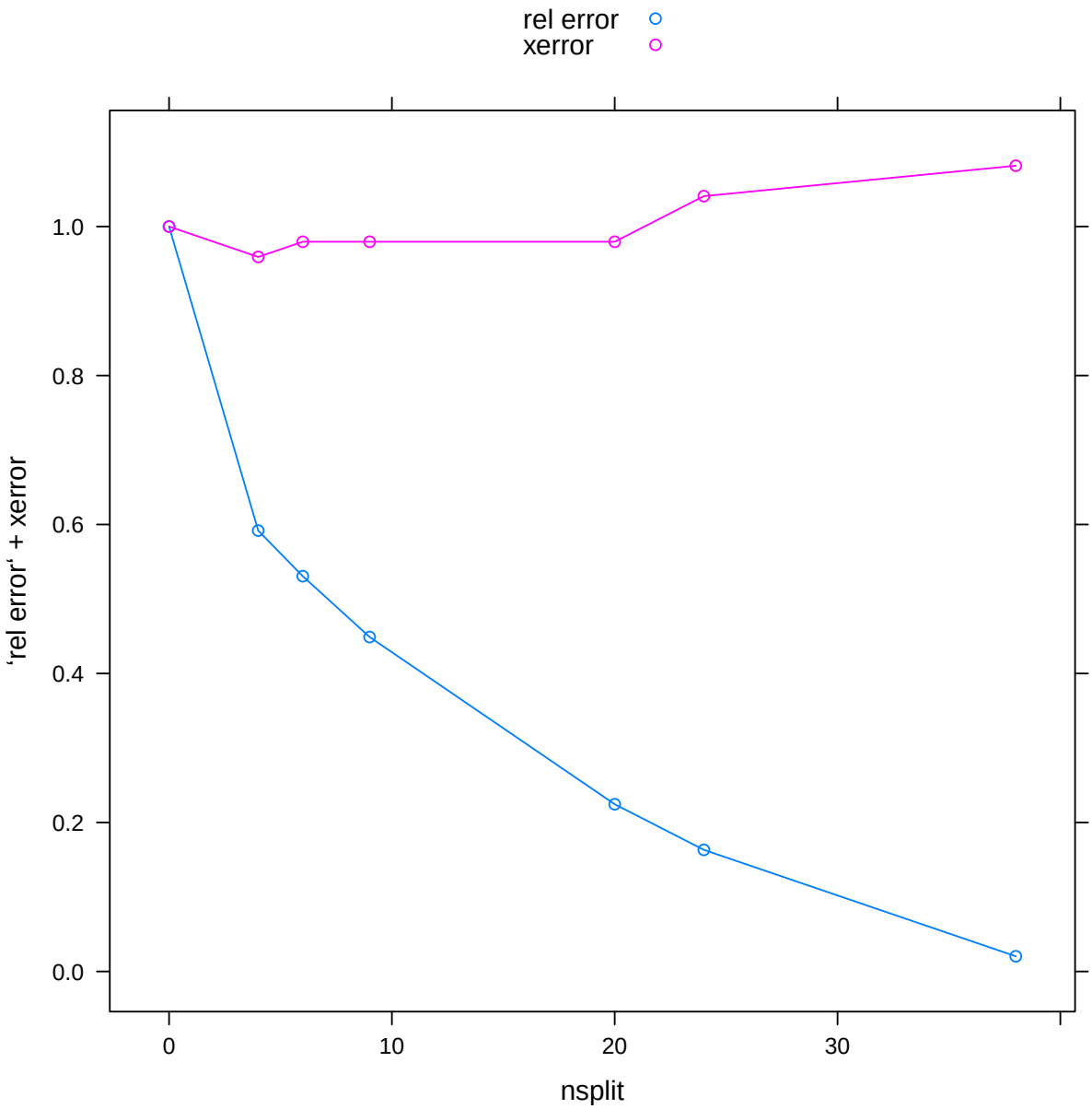
```
##
## Node number 1: 196 observations,    complexity param=0.6530612
##   predicted class=glaucoma  expected loss=0.5  P(node) =1
##     class counts:    98    98
##    probabilities: 0.500 0.500
##   left son=2 (76 obs) right son=3 (120 obs)
##   Primary splits:
##       varg < 0.209   to the left,  improve=44.01404, (0 missing)
##       vari < 0.0615  to the left,  improve=41.01677, (0 missing)
##       vars < 0.059   to the left,  improve=37.50000, (0 missing)
##       varn < 0.0895  to the left,  improve=33.82974, (0 missing)
##       tmi  < -0.0115 to the right, improve=31.82501, (0 missing)
##   Surrogate splits:
##       varn < 0.0895  to the left,  agree=0.934, adj=0.829, (0 split)
##       vari < 0.049   to the left,  agree=0.918, adj=0.789, (0 split)
##       vars < 0.059   to the left,  agree=0.888, adj=0.711, (0 split)
##       rnf  < 0.1525  to the left,  agree=0.878, adj=0.684, (0 split)
##       vbri < 0.109   to the right, agree=0.842, adj=0.592, (0 split)
##
## Node number 2: 76 observations
##   predicted class=glaucoma  expected loss=0.07894737  P(node) =0.3877551
##     class counts:    70     6
##    probabilities: 0.921 0.079
##
## Node number 3: 120 observations,    complexity param=0.07142857
##   predicted class=normal    expected loss=0.2333333  P(node) =0.6122449
##     class counts:    28    92
##    probabilities: 0.233 0.767
##   left son=6 (7 obs) right son=7 (113 obs)
##   Primary splits:
##       mhcg < 0.1695  to the right, improve=8.738643, (0 missing)
##       mhci < 0.099   to the right, improve=7.836770, (0 missing)
##       phci < 0.0375  to the right, improve=7.006061, (0 missing)
##       phcg < -0.1055 to the right, improve=6.607440, (0 missing)
##       hic  < 0.412   to the right, improve=6.142535, (0 missing)
##   Surrogate splits:
##       mhcn < 0.176   to the right, agree=0.975, adj=0.571, (0 split)
##       mhcs < 0.207   to the right, agree=0.967, adj=0.429, (0 split)
##       phcn < 0.0415  to the right, agree=0.967, adj=0.429, (0 split)
##       mhct < 0.433   to the right, agree=0.958, adj=0.286, (0 split)
##       phcg < -0.001  to the right, agree=0.958, adj=0.286, (0 split)
##
## Node number 6: 7 observations
##   predicted class=glaucoma  expected loss=0  P(node) =0.03571429
##     class counts:     7     0
##    probabilities: 1.000 0.000
##
## Node number 7: 113 observations
```

```
##   predicted class=normal    expected loss=0.1858407  P(node) =0.5765306
##     class counts:    21    92
##    probabilities: 0.186 0.814
```

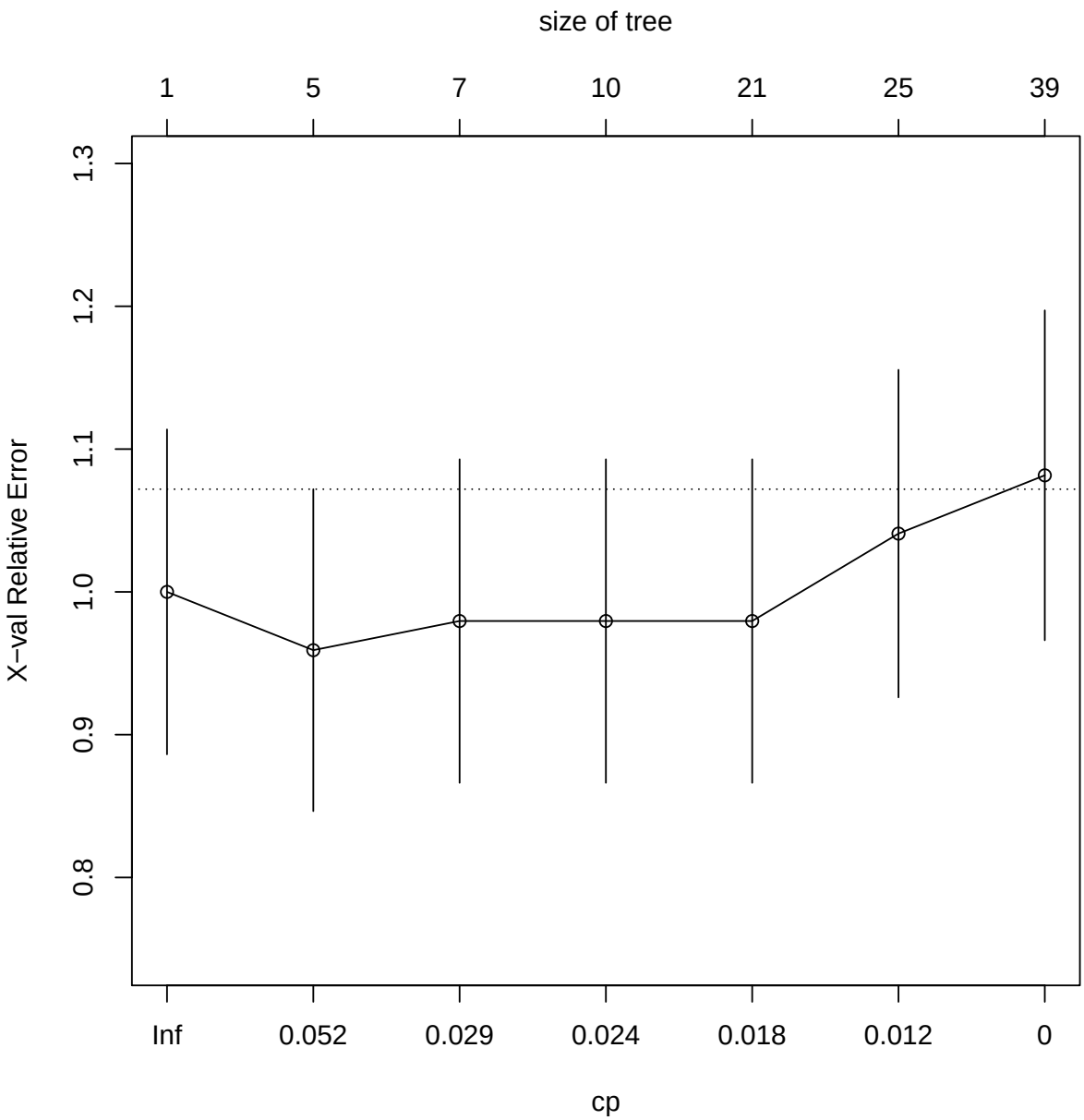
```
data(stagec)
stagec$pgstat <- factor(stagec$pgstat, levels = 0:1,
    labels = c("No", "Prog"))
stagec <- na.omit(stagec)
rp <- rpart(pgstat ~ age + eet + g2 + grade +
    gleason + ploidy, data = stagec, method = "class",
    control = rpart.control(cp = 0, xval = 10,
        minbucket = 1))
prp <- printcp(rp)

##
## Classification tree:
## rpart(formula = pgstat ~ age + eet + g2 + grade + gleason + ploidy,
##     data = stagec, method = "class", control = rpart.control(cp = 0,
##         xval = 10, minbucket = 1))
##
## Variables actually used in tree construction:
## [1] age     eet     g2      gleason grade   ploidy
##
## Root node error: 49/134 = 0.36567
##
## n= 134
##
##         CP nsplit rel error  xerror    xstd
## 1 0.088435      0  1.000000 1.00000 0.11378
## 2 0.030612      4  0.591837 0.95918 0.11274
## 3 0.027211      6  0.530612 0.97959 0.11327
## 4 0.020408      9  0.448980 0.97959 0.11327
## 5 0.015306     20  0.224490 0.97959 0.11327
## 6 0.010204     24  0.163265 1.04082 0.11470
## 7 0.000000     38  0.020408 1.08163 0.11551

prp <- as.data.frame(prp)
xyplot(`rel error` + xerror ~ nsplit, data = prp,
    type = "b", auto.key = TRUE)
```



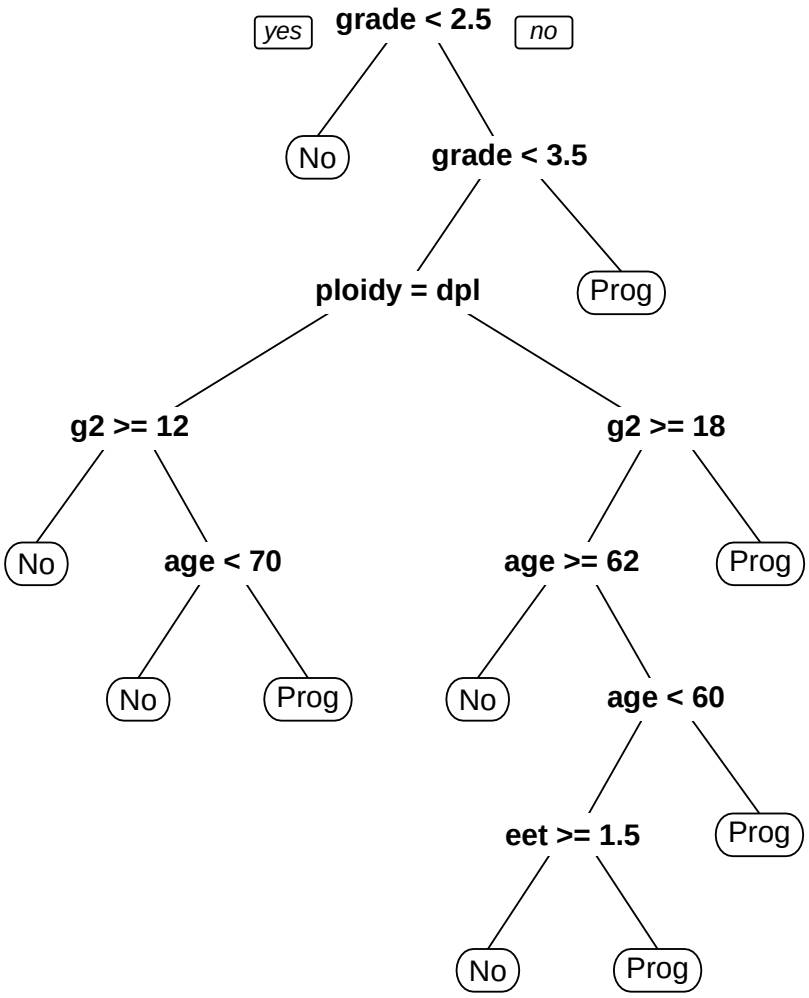

```
xyplot(`rel error` + xerror ~ nsplit, data = prp[1:10,
    ], type = "b", auto.key = TRUE)
```


rel error
xerror
1.0
0.8
0.6
0.4
0.2
0.0
‘rel error’ + xerror
0
10
20
30
nsplit

```
plotcp(rp)
```



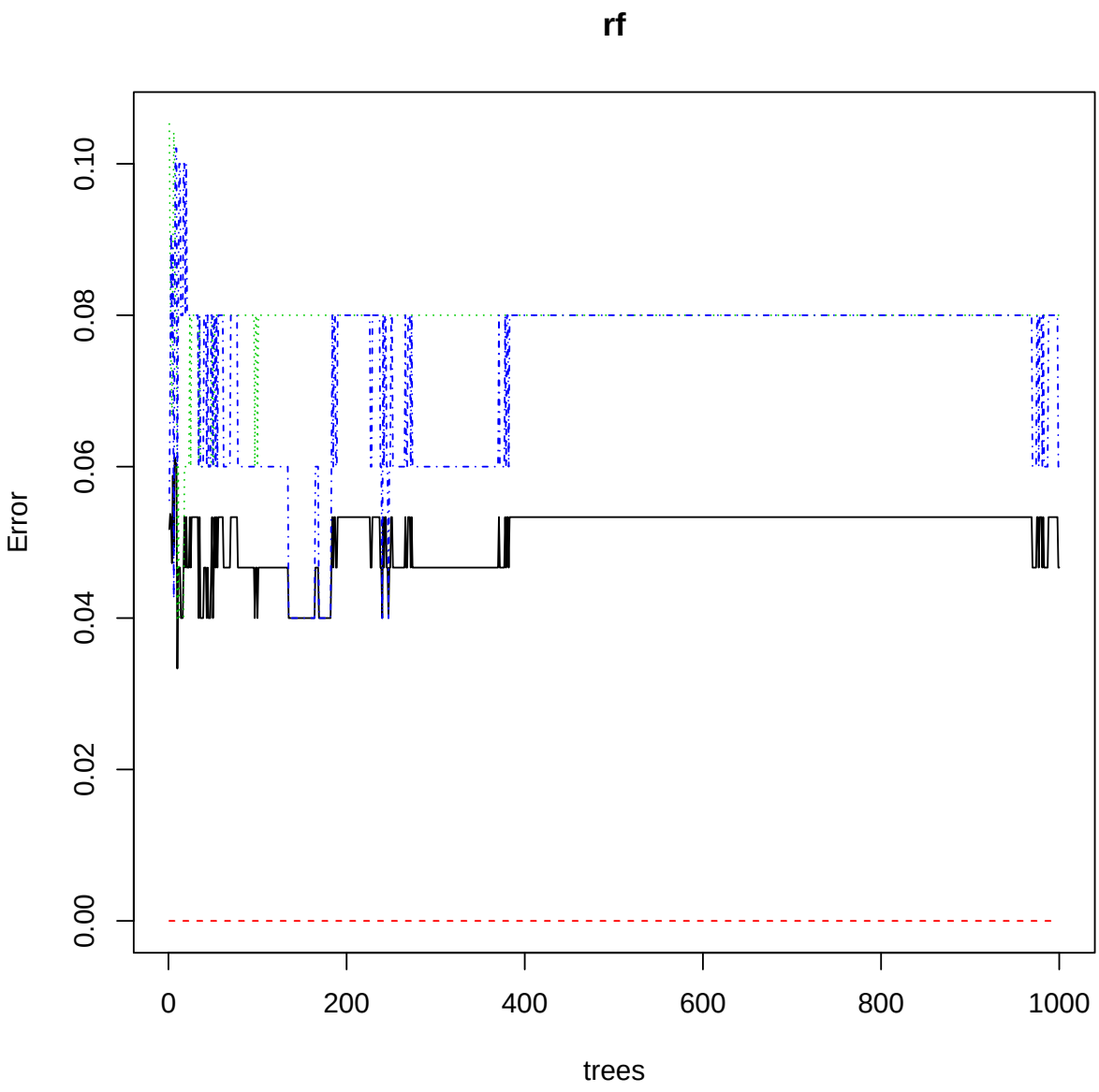

```
cp <- 0.025
tn.rp <- tune.rpart(pgstat ~ age + eet +
    g2 + grade + gleason + ploidy, data = stagec,
    model = TRUE, cp = cp, xval = 0, minsplit = 3,
    tunecontrol = tune.control(cross = nrow(stagec)))
tn.rp

##
## Error estimation of 'rpart.wrapper' using leave-one-out: 0.3283582

rpart.plot(tn.rp$best.model)
```




```
# Mmm... unbalanced design
table(stagec$pgstat)

##
##   No Prog
##   85   49

source("class.R")
tn.rp <- tune.rpart(pgstat ~ age + eet +
    g2 + grade + gleason + ploidy, data = na.omit(stagec),
    model = TRUE, cp = cp, xval = 0, minsplit = 3,
    tunecontrol = tune.control(cross = 10,
        error.fun = error.fun.max))
tn.rp

##
## Error estimation of 'rpart.wrapper' using 10-fold cross validation: 0.6024747
```

11.2 Random Forest

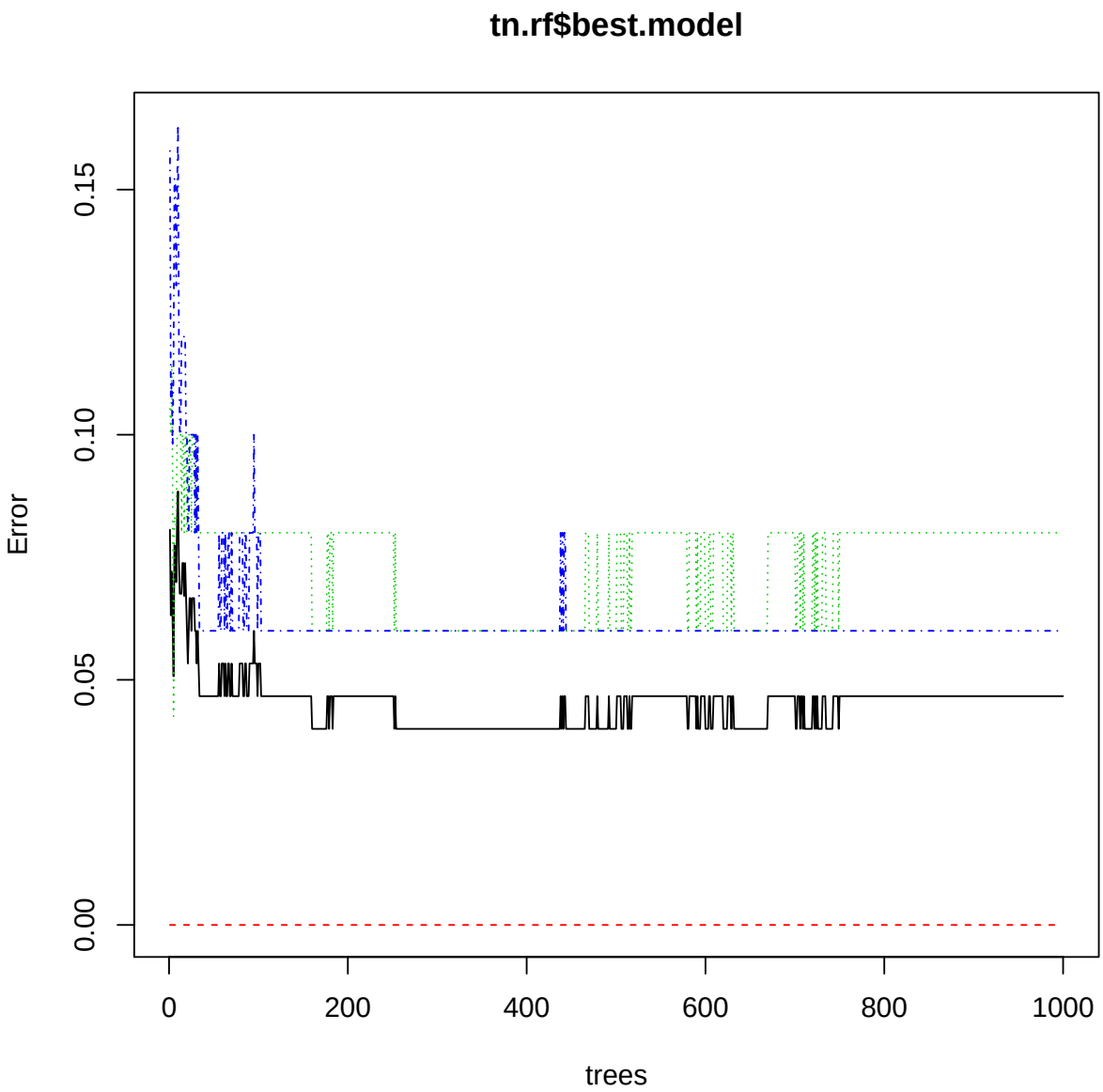
```
library(rpart)
library(rpart.plot)
library(randomForest)

## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.

library(party)  # http://cran.r-project.org/web/packages/party/vignettes/party.pdf
library(e1071)
library(lattice)
library(latticeExtra)
library(pls)
library(MASS)
# iris
rf <- randomForest(Species ~ ., data = iris,
    ntree = 1000, mtry = 2, nodesize = 1,
    maxnodes = 10)
plot(rf)
```



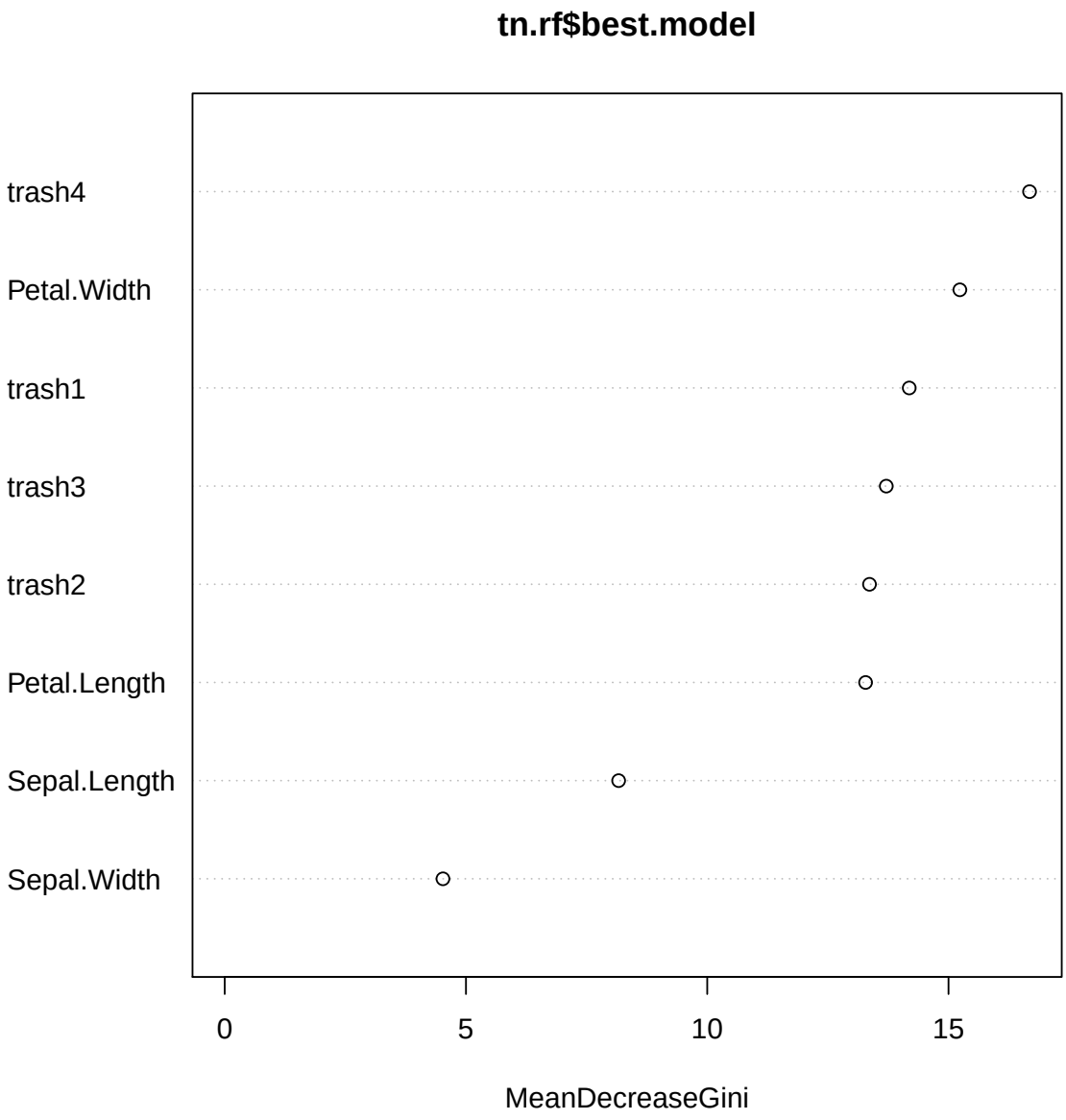

```
tn.rf <- tune.randomForest(Species ~ ., data = iris,
    mtry = 1:4, nodesize = 1, ntree = 1000)
tn.rf

##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  nodesize mtry ntree
##         1    1  1000
##
## - best performance: 0.04666667

tn.rf$best.model
```

```
##
## Call:
##  best.randomForest(x = Species ~ ., data = iris, nodesize = 1,      mtry = 1:4,
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 4.67%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         46         4        0.08
## virginica       0          3        47        0.06
```

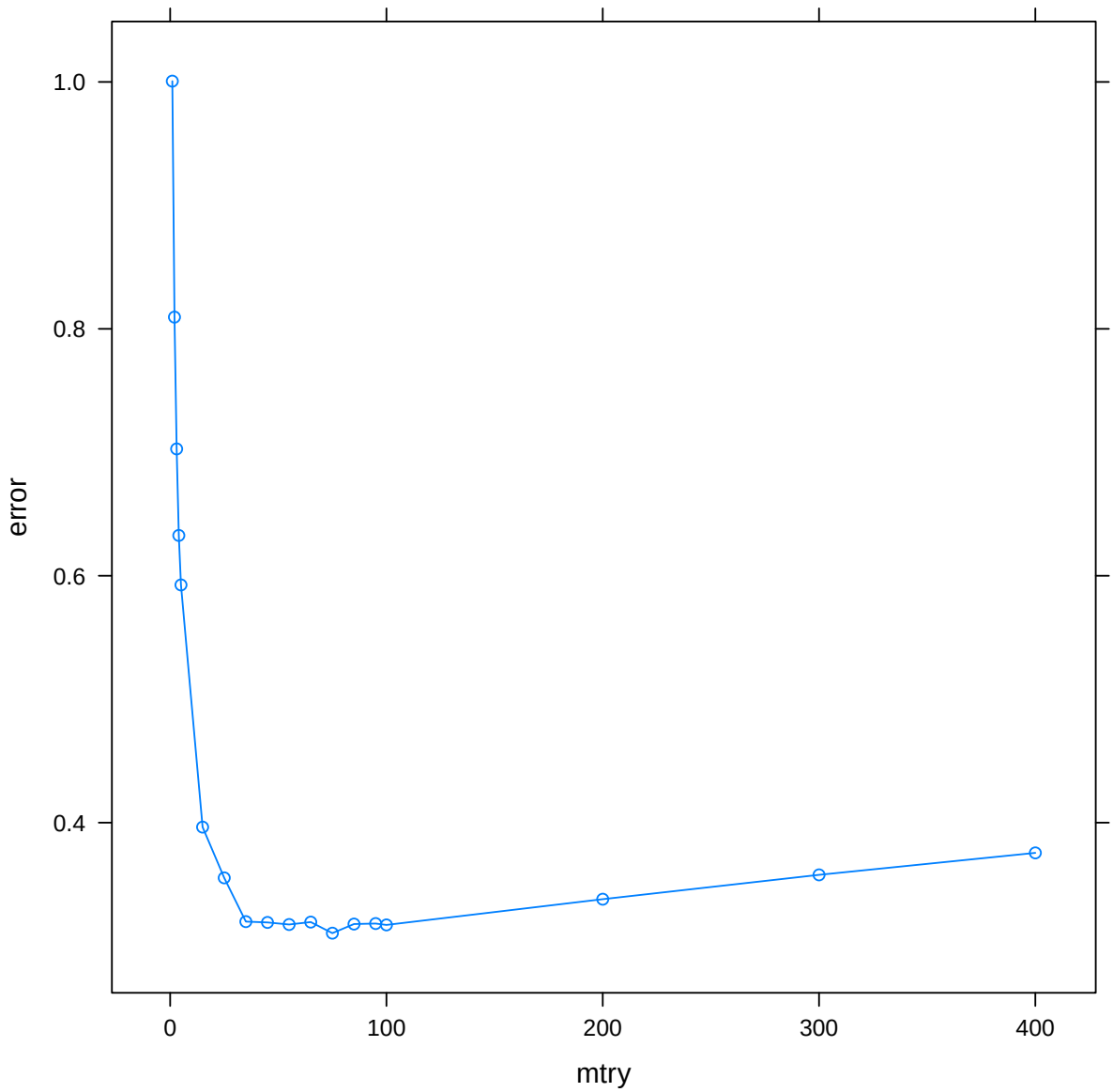
```
plot(tn.rf$best.model)
```



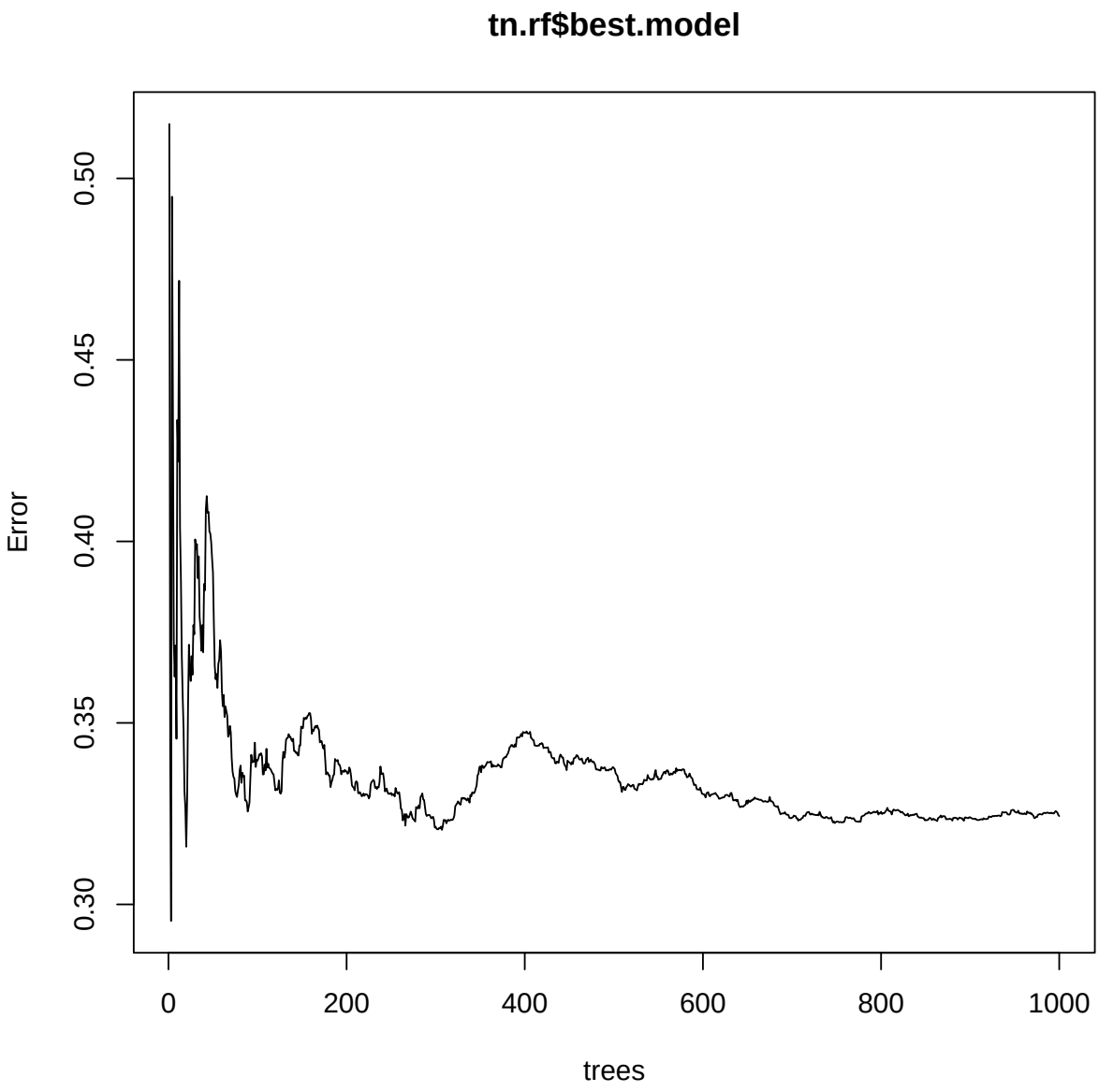
```r
importance(tn.rf$best.model)

##               MeanDecreaseGini
## Sepal.Length         8.163575
## Sepal.Width          4.522660
## Petal.Length        13.281274
## Petal.Width         15.234752
## trash1              14.186083
## trash2              13.362199
## trash3              13.710047
## trash4              16.678938

varImpPlot(tn.rf$best.model)
```



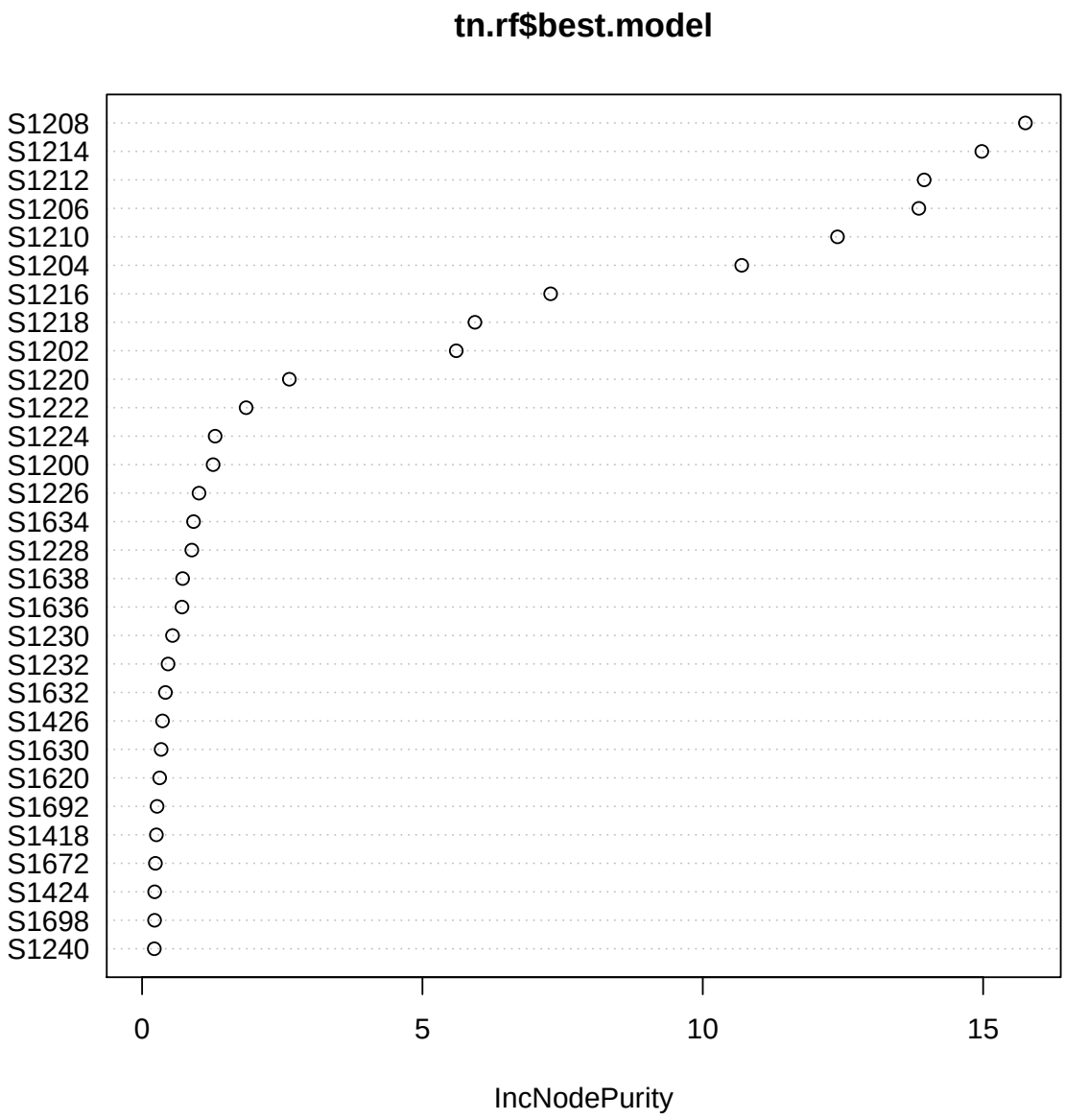
```r
# gasoline
data(gasoline)
names(gasoline)

## [1] "octane" "NIR"

dim(gasoline)

## [1] 60  2

class(gasoline$NIR) <- NULL
colnames(gasoline$NIR) <- paste("S", seq(900,
    1700, 2), sep = "")
gasoline <- cbind(subset(gasoline, select = octane),
    as.data.frame(gasoline$NIR))
dim(gasoline)

## [1]  60 402

tn.rf <- tune.randomForest(octane ~ ., data = gasoline,
    mtry = c(1:4, seq(5, 100, 10), 100, 200,
        300, 400), nodesize = 5, ntree = 1000)
tn.rf

##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  nodesize mtry ntree
##         5   75  1000
##
## - best performance: 0.3104851

xyplot(error ~ mtry, data = as.data.frame(tn.rf$performances),
    type = "b")
```



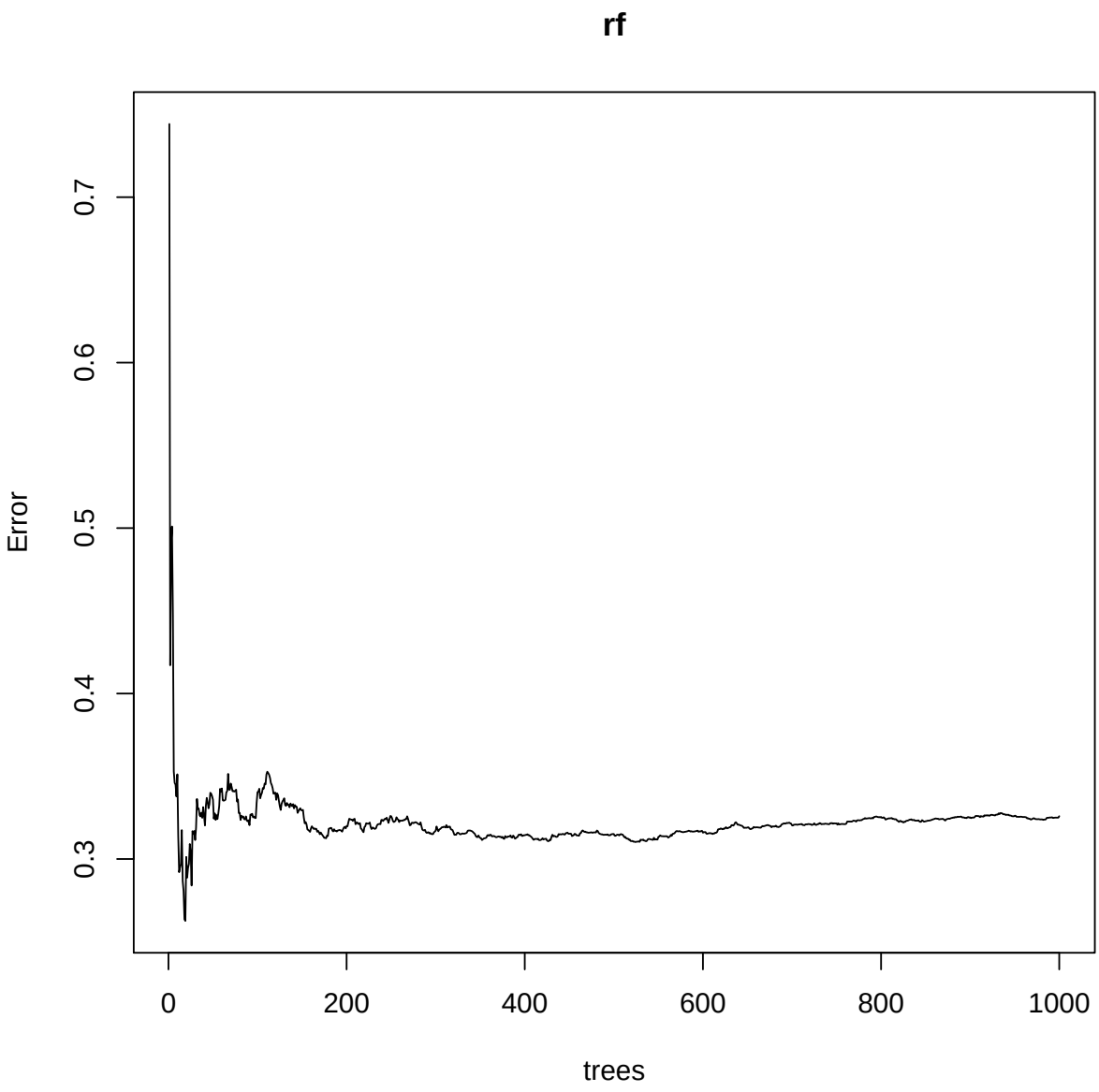

```
tn.rf$best.model

##
## Call:
##  best.randomForest(x = octane ~ ., data = gasoline, nodesize = 5,      mtry = c(
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 75
##
##           Mean of squared residuals: 0.3243393
##                     % Var explained: 85.91

plot(tn.rf$best.model)
```




```
varImpPlot(tn.rf$best.model)
```

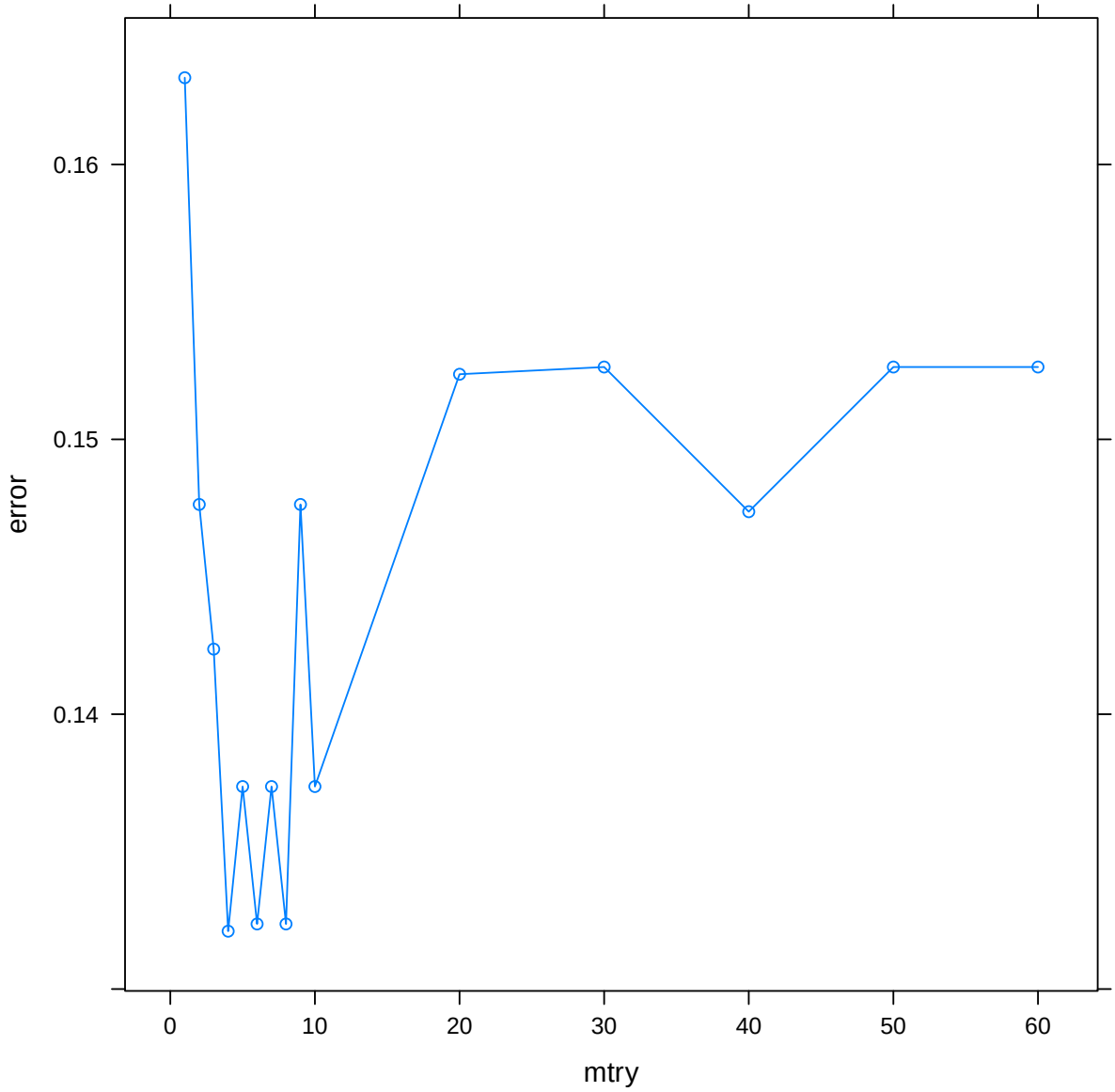



```
rf <- randomForest(octane ~ ., data = gasoline,
    mtry = 75, nodesize = 5, ntree = 1000)
plot(rf)
```



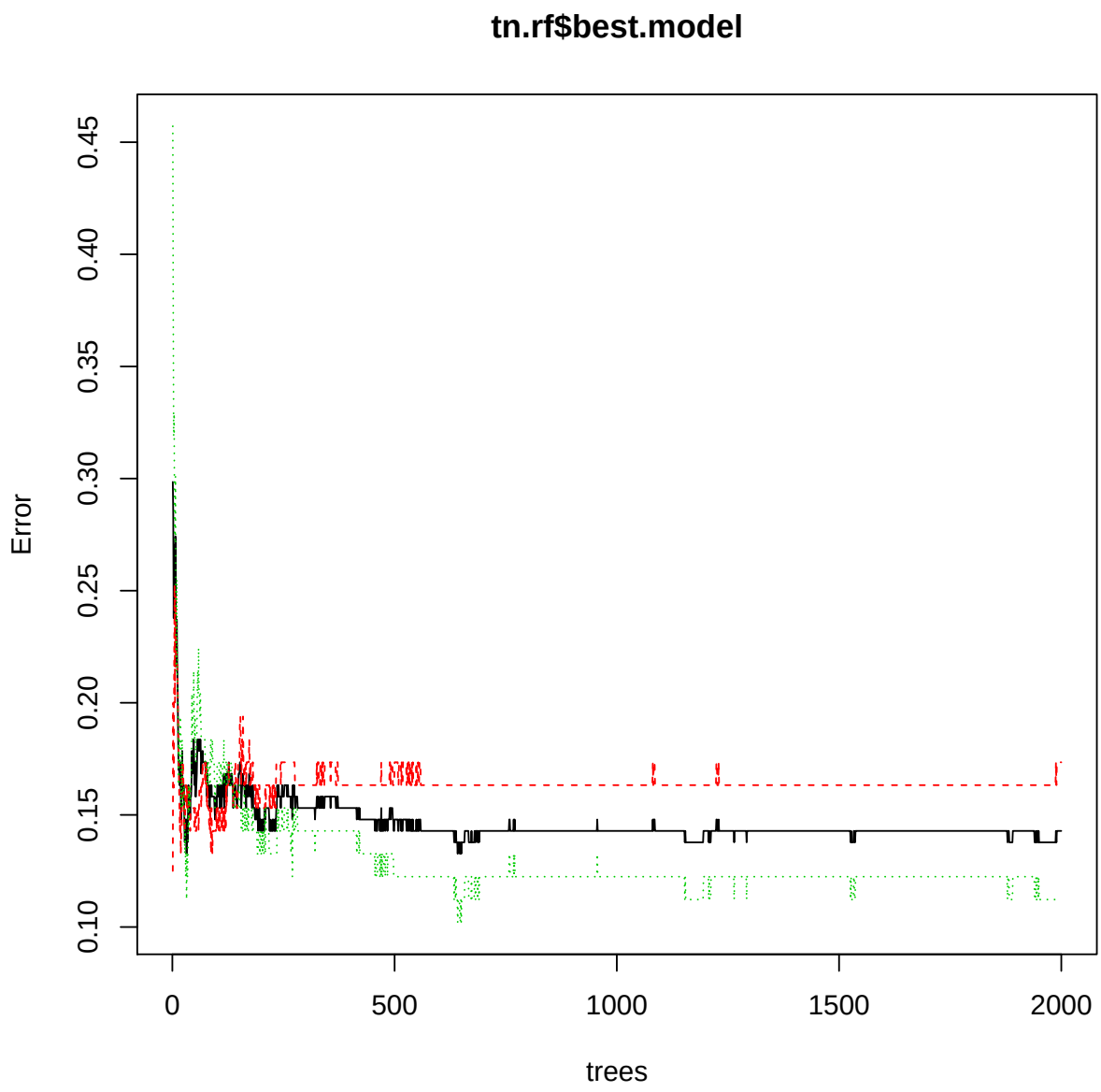

```
# GlaucomaM
glaucomaM <- read.table("svm/GlaucomaM.txt",
    stringsAsFactors = TRUE, header = TRUE,
    sep = " ", strip.white = TRUE, blank.lines.skip = TRUE)
tn.rf <- tune.randomForest(Class ~ ., data = glaucomaM,
    mtry = c(1:9, seq(10, 63, 10)), ntree = 2000)
tn.rf

##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  mtry ntree
```

```
##     4  2000
##
## - best performance: 0.1321053

xyplot(error ~ mtry, data = as.data.frame(tn.rf$performances),
    type = "b")
```

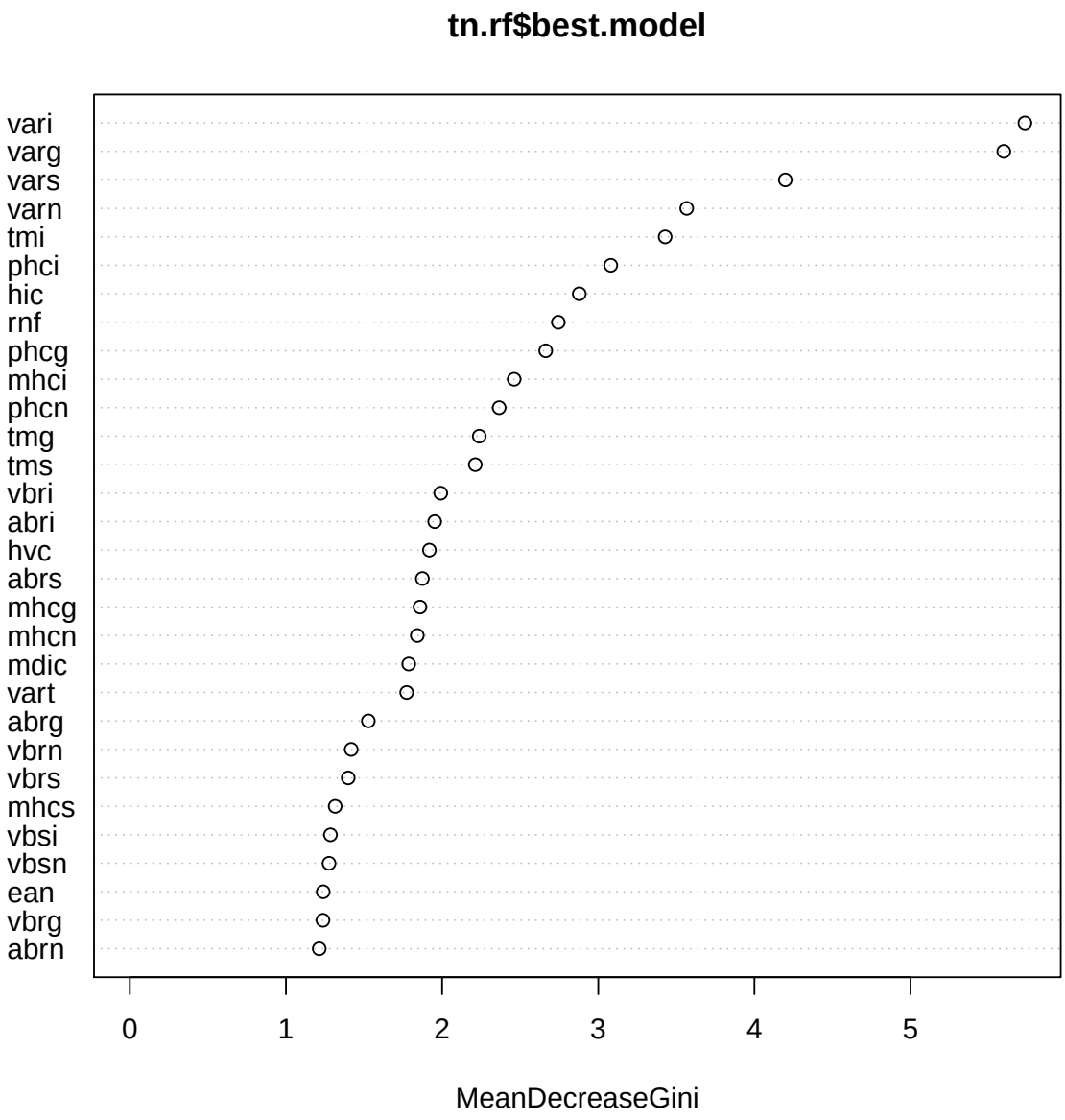



```
tn.rf$best.model

##
## Call:
##  best.randomForest(x = Class ~ ., data = glaucomaM, mtry = c(1:9,      seq(10, 6
##                Type of random forest: classification
##                      Number of trees: 2000
## No. of variables tried at each split: 4
```

```
##
##          OOB estimate of  error rate: 14.29%
## Confusion matrix:
##           glaucoma normal class.error
## glaucoma        81     17   0.1734694
## normal          11     87   0.1122449

plot(tn.rf$best.model)
```

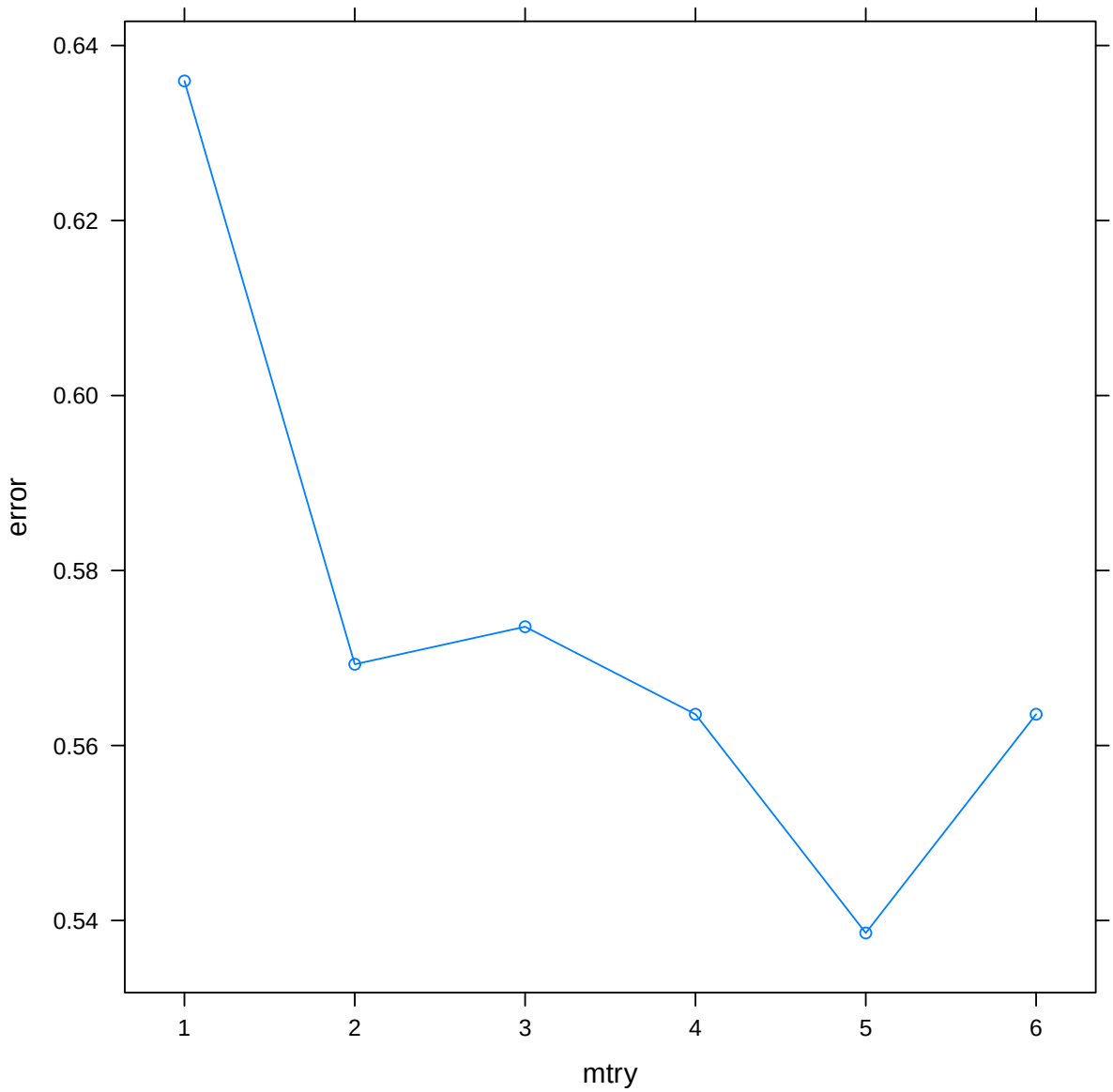



```
varImpPlot(tn.rf$best.model)
```



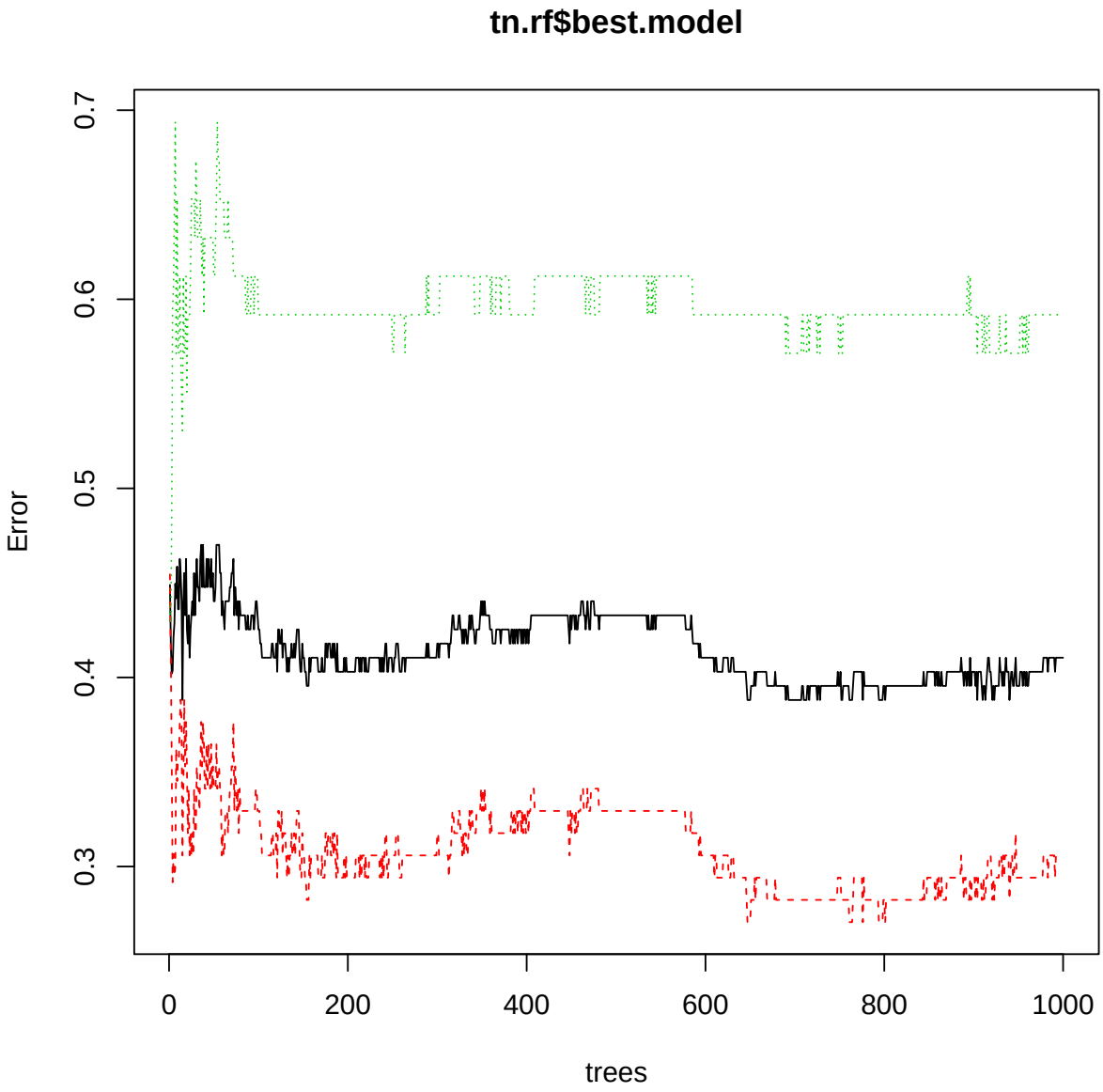

```
data(stagec)
stagec$pgstat <- factor(stagec$pgstat, levels = 0:1,
    labels = c("No", "Prog"))
stagec <- na.omit(stagec)
source("class.R")
tn.rf <- tune.randomForest(pgstat ~ age +
    eet + g2 + grade + gleason + ploidy,
    data = stagec, mtry = c(1:6), ntree = 1000,
    tunecontrol = tune.control(error.fun = error.fun.max))
tn.rf

##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  mtry ntree
##     5  1000
##
## - best performance: 0.5385714

xyplot(error ~ mtry, data = as.data.frame(tn.rf$performances),
    type = "b")
```

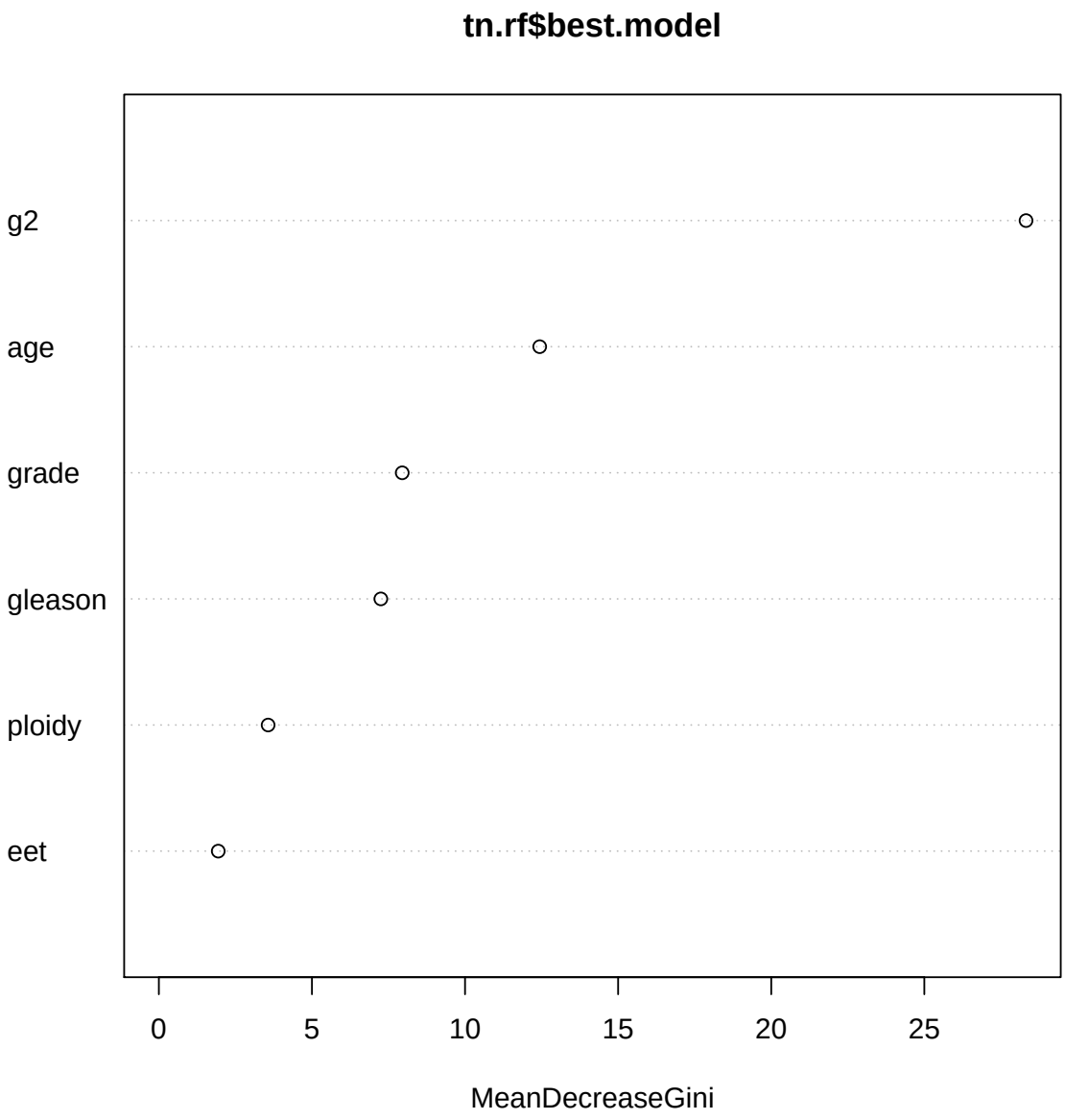



```
tn.rf$best.model

##
## Call:
##  best.randomForest(x = pgstat ~ age + eet + g2 + grade + gleason +      ploidy,
```

```
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 41.04%
## Confusion matrix:
##      No Prog class.error
## No   59   26   0.3058824
## Prog 29   20   0.5918367

plot(tn.rf$best.model)
```

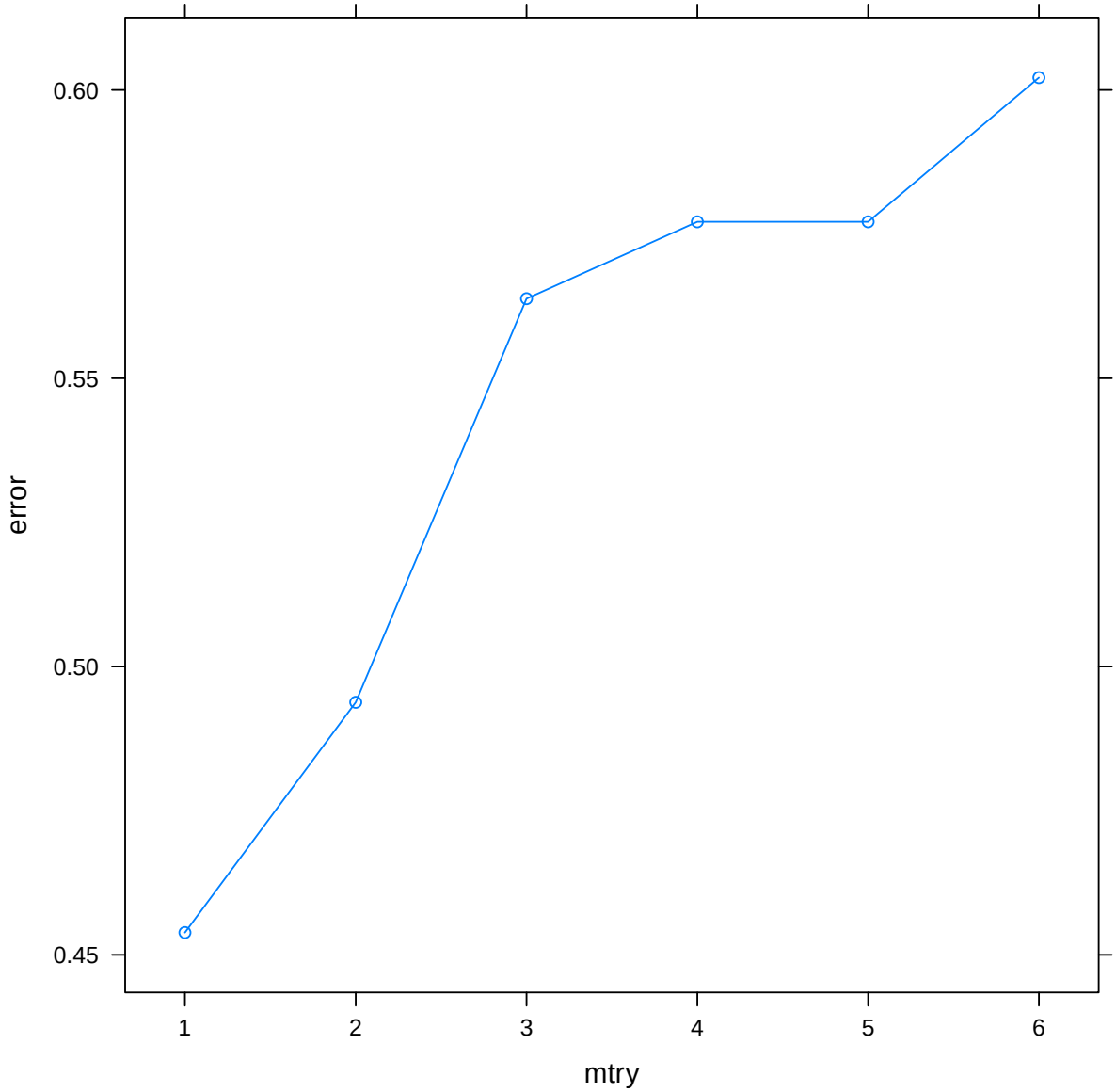



```
varImpPlot(tn.rf$best.model)
```

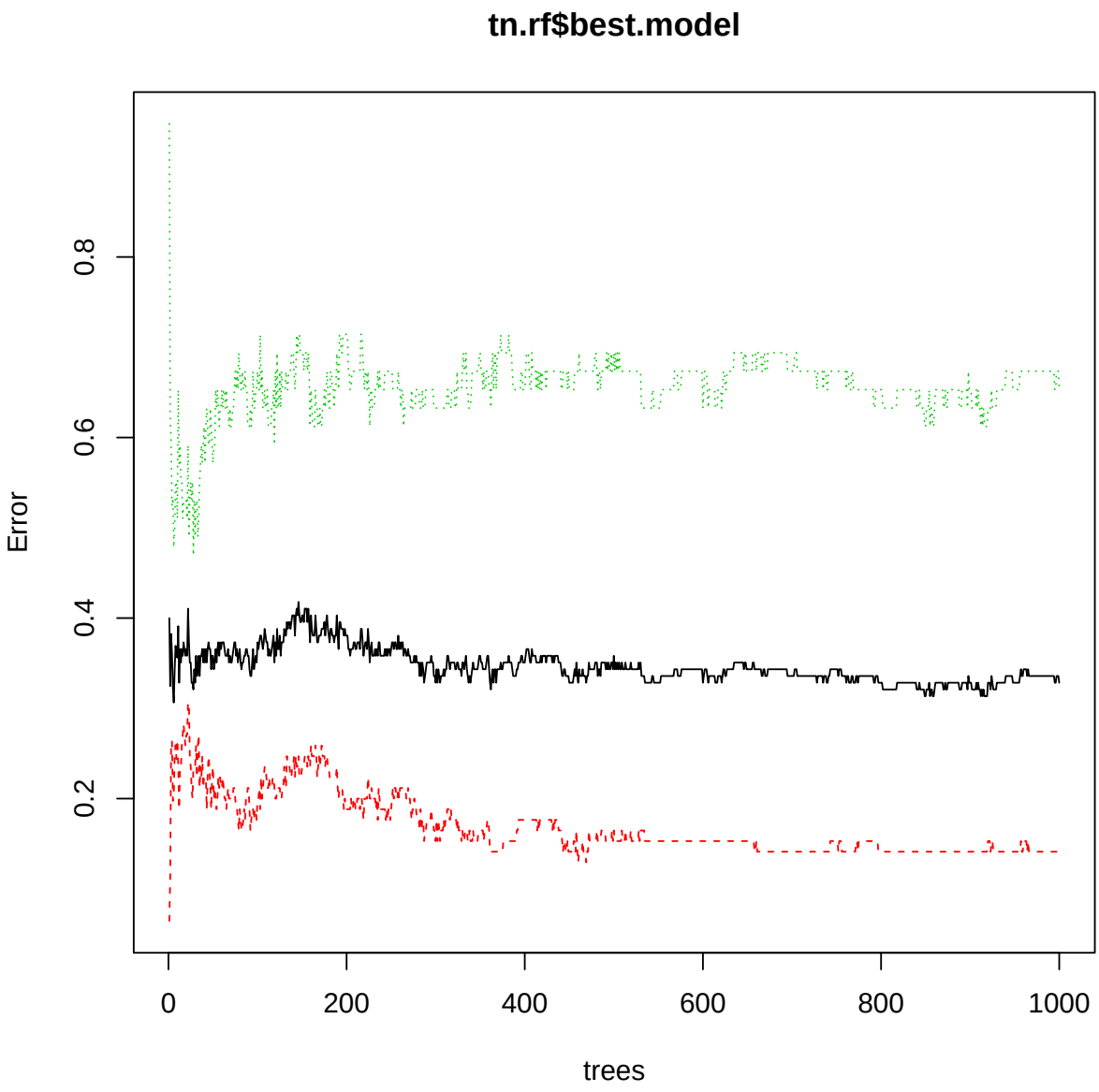



```
my.rf <- function(..., data) {
    rf <- randomForest(..., data = data)
    rf$data <- data
    rf
}
my.predict.rf <- function(x, newdata, cutoff.data = x$data,
    ..., measure = "max") {
    response <- model.frame(x$terms, cutoff.data)[,
        1]
    opt <- roc.opt(predict(x, cutoff.data,
        type = "prob")[, 2], as.numeric(response),
        measure = measure)
    cutoff <- opt$cutoff
    factor(as.integer(predict(x, newdata = newdata,
        type = "prob")[, 2] > cutoff), labels = levels(response))
```

```
}
tn.rf <- tune(my.rf, pgstat ~ age + eet +
    g2 + grade + gleason + ploidy, data = stagec,
    ranges = list(mtry = c(1:6)), ntree = 1000,
    predict.func = my.predict.rf, tunecontrol = tune.control(error.fun = error.fun.
tn.rf

##
## Parameter tuning of 'my.rf':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  mtry
##     1
##
## - best performance: 0.45386

xyplot(error ~ mtry, data = as.data.frame(tn.rf$performances),
    type = "b")
```




```
tn.rf$best.model

##
## Call:
##  best.tune(method = my.rf, train.x = pgstat ~ age + eet + g2 +      grade + glea
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 32.84%
## Confusion matrix:
##      No Prog class.error
## No   73   12   0.1411765
## Prog 32   17   0.6530612

plot(tn.rf$best.model)
```

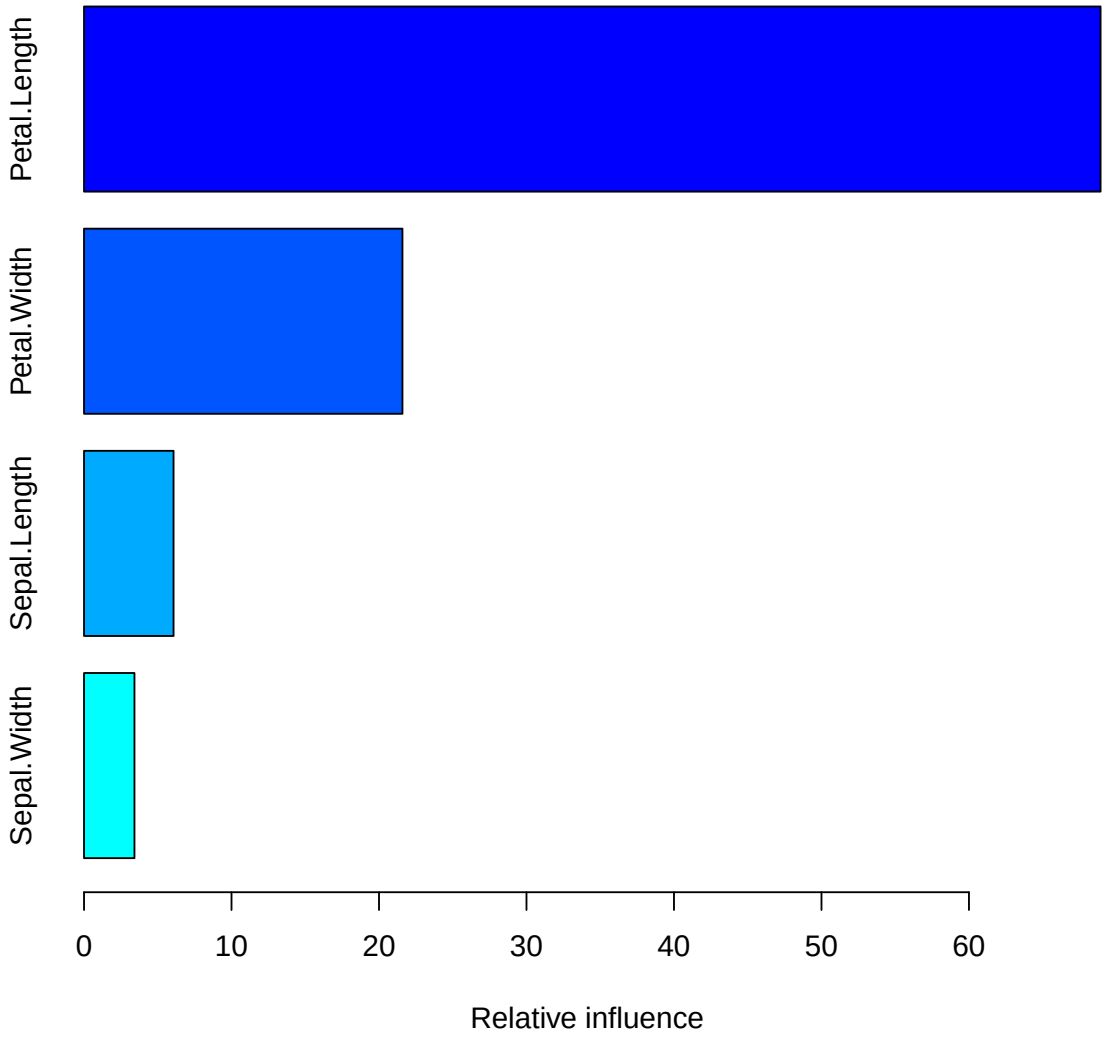


12 Материалы с занятия 12 декабря

```
read_chunk("rpart/boost.R")
read_chunk("rpart/mboost.R")
```

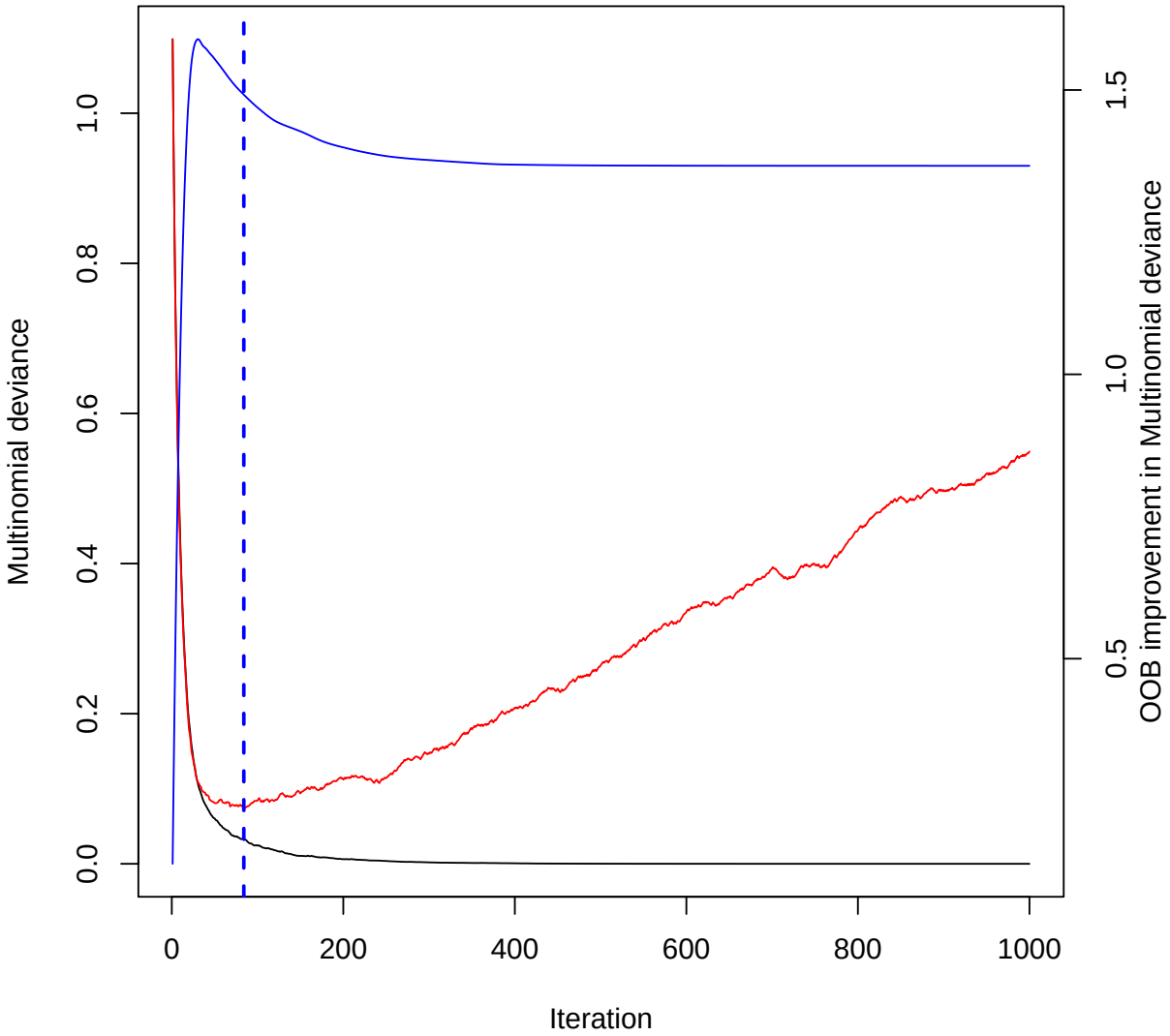
```
library(e1071)
library(lattice)
library(latticeExtra)
library(pls)
library(MASS)
library(gbm)  # http://www.saedsayad.com/docs/gbm2.pdf http://www.ncbi.nlm.nih.gov/
library(ada)  # http://dept.stat.lsa.umich.edu/~gmichail/ada_final.pdf
```

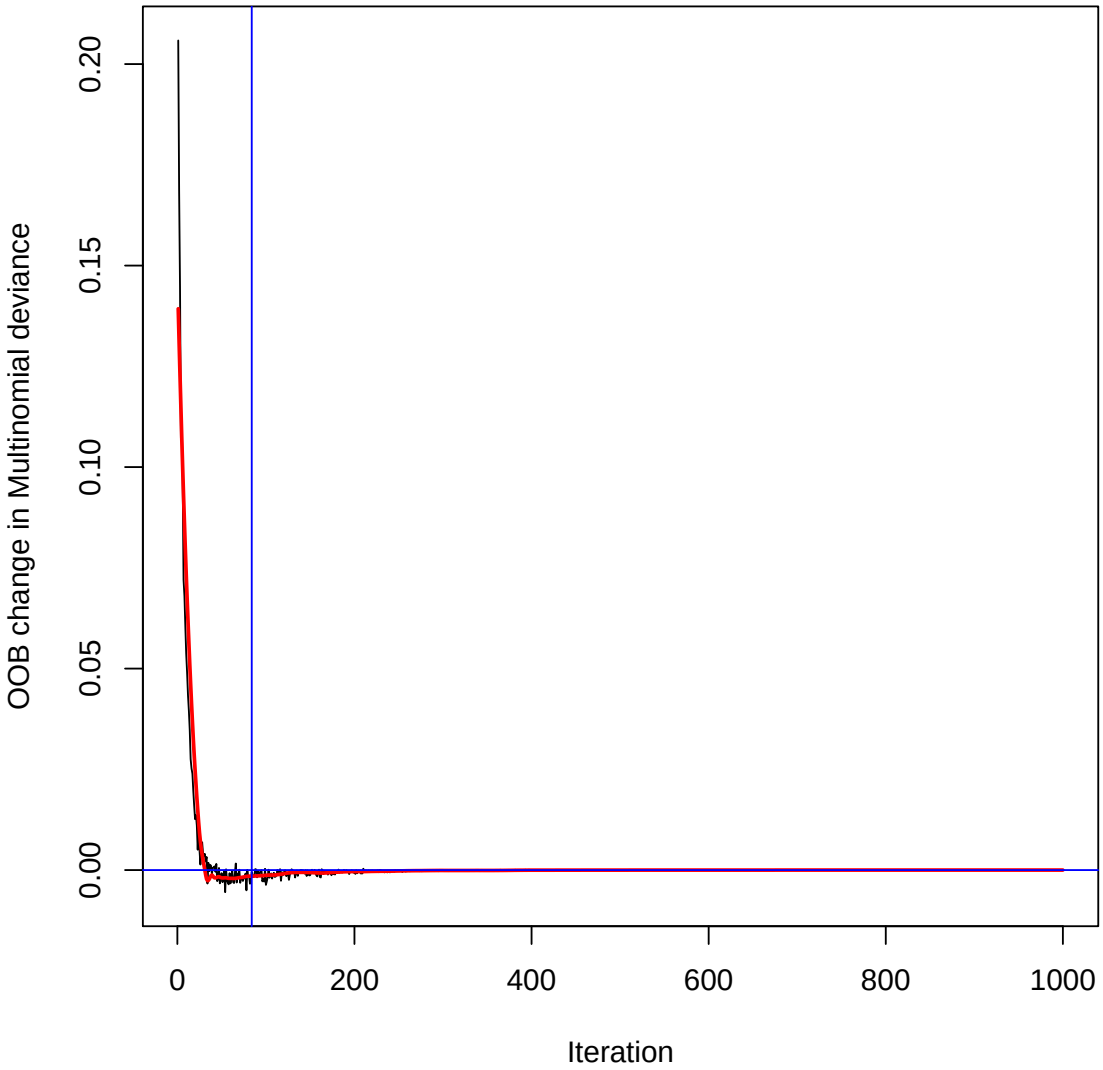
```r
library(adabag)
library(mboost)
rm(iris)
data(iris)
iris <- iris[sample(nrow(iris)), ]
gb <- gbm(Species ~ ., data = iris, distribution = "multinomial",
    interaction.depth = 2, n.trees = 1000,
    n.minobsinnode = 1, shrinkage = 0.05,
    bag.fraction = 0.5, train.fraction = 0.66,
    n.cores = 4)
gb

## gbm(formula = Species ~ ., distribution = "multinomial", data = iris,
##     n.trees = 1000, interaction.depth = 2, n.minobsinnode = 1,
##     shrinkage = 0.05, bag.fraction = 0.5, train.fraction = 0.66,
##     n.cores = 4)
## A gradient boosted model with multinomial loss function.
## 1000 iterations were performed.
## The best test-set iteration was 84.
## There were 4 predictors of which 4 had non-zero influence.

summary(gb, plotit = TRUE, method = relative.influence)
```



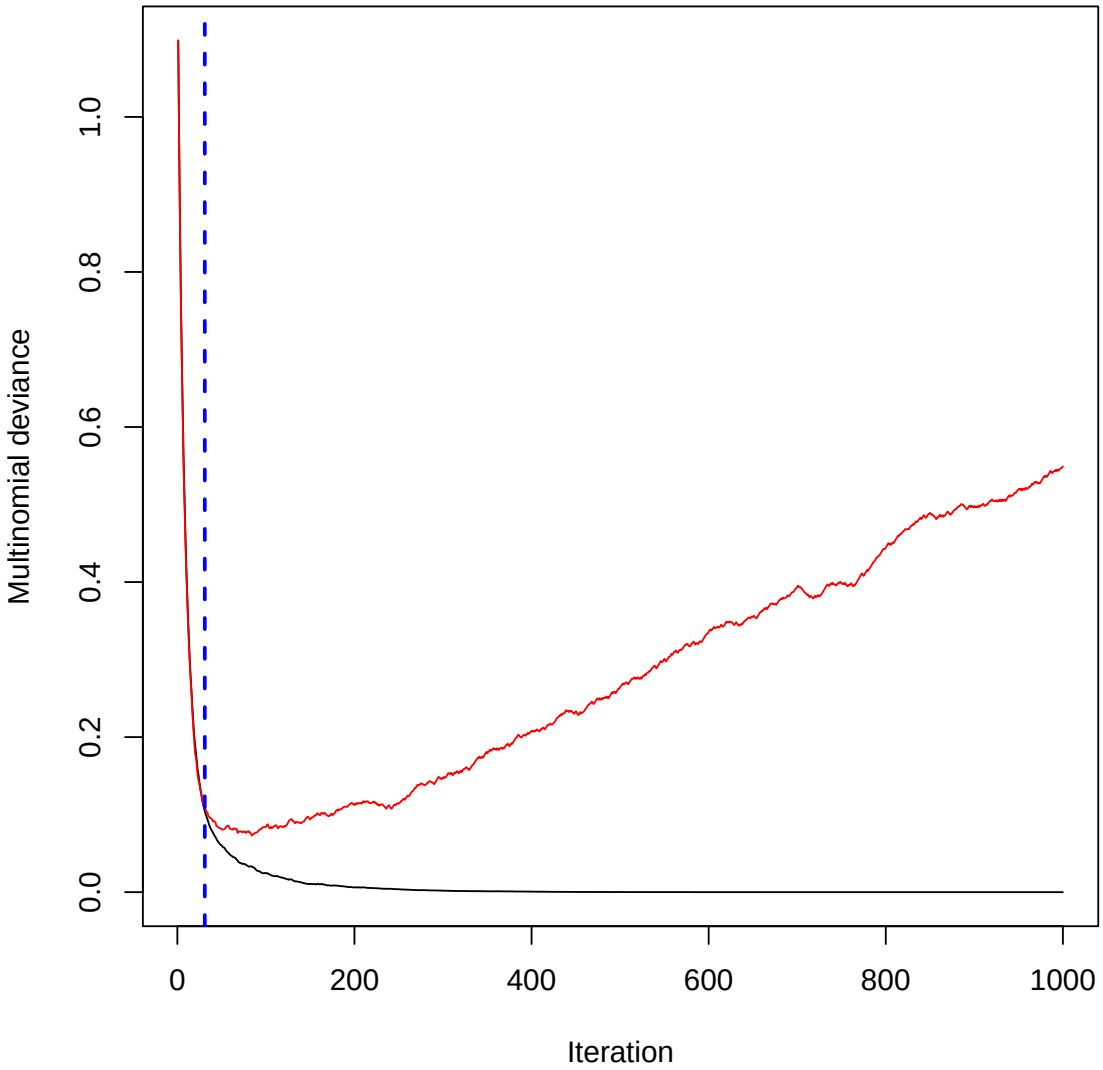

```
##                        var   rel.inf
## Petal.Length Petal.Length 68.922808
## Petal.Width   Petal.Width 21.588642
## Sepal.Length Sepal.Length  6.069377
## Sepal.Width   Sepal.Width  3.419172

# No supported yet for multinomial
# classification summary(gb, plotit =
# TRUE, method = permutation.test.gbm)
pred <- predict(gb)[, , 1]

## Using 84 trees...

pred <- colnames(pred)[max.col(pred)]
table(actual = iris$Species, predicted = pred)
```

```
##             predicted
## actual       setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         48         2
##   virginica       0          0        50

pred <- predict(gb, n.trees = 1000)[, , 1]
pred <- colnames(pred)[max.col(pred)]
table(actual = iris$Species, predicted = pred)

##             predicted
## actual       setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         49         1
##   virginica       0          0        50

gbm.perf(gb, plot.it = TRUE, oobag.curve = TRUE,
    overlay = TRUE, method = "test")
```


Multinomial deviance
0.0
0.2
0.4
0.6
0.8
1.0
OOB improvement in Multinomial deviance
0.5
1.0
1.5
0
200
400
600
800
1000
Iteration




```
## [1] 84

gbm.perf(gb, plot.it = TRUE, method = "OOB")

## Warning in gbm.perf(gb, plot.it = TRUE, method = "OOB"): OOB generally underesti
the optimal number of iterations although predictive performance is reasonably
competitive. Using cv.folds>0 when calling gbm usually results in improved predicti
performance.
```




```
## [1] 31

# pretty.gbm.tree(gb, 1)
# interact.gbm(gb, n.trees = 70)
gb <- gbm(Species ~ ., data = iris, distribution = "multinomial",
    interaction.depth = 2, n.trees = 1000,
    n.minobsinnode = 1, shrinkage = 0.05,
    bag.fraction = 0.5, train.fraction = 1,
    cv.folds = 10, n.cores = 4)
gb

## gbm(formula = Species ~ ., distribution = "multinomial", data = iris,
##     n.trees = 1000, interaction.depth = 2, n.minobsinnode = 1,
##     shrinkage = 0.05, bag.fraction = 0.5, train.fraction = 1,
##     cv.folds = 10, n.cores = 4)
```

```
## A gradient boosted model with multinomial loss function.
## 1000 iterations were performed.
## The best cross-validation iteration was 53.
## There were 4 predictors of which 4 had non-zero influence.
```

```
my.gbm <- function(..., data, subset) {
    data <- data[subset, ]
    gbm(..., data = data)
}
my.predict.gbm <- function(x, newdata, ...) {
    pred <- predict(x, newdata, ...)
    labels <- dimnames(pred)[[2]]
    dim(pred) <- dim(pred)[1:2]
    pred <- factor(max.col(pred), levels = seq_along(labels),
        labels = labels)
    pred
}
tn.gb <- tune(my.gbm, Species ~ ., data = iris,
    distribution = "multinomial", n.trees = 1000,
    shrinkage = 0.05, bag.fraction = 0.5,
    train.fraction = 0.66, n.cores = 4, ranges = list(interaction.depth = 1:5,
        n.minobsinnode = c(1, 5)), tunecontrol = tune.control(cross = 2,
        best.model = TRUE), predict.func = my.predict.gbm)
```

```
## Using 111 trees...
## Using 70 trees...
## Using 121 trees...
## Using 47 trees...
## Using 181 trees...
## Using 46 trees...
## Using 908 trees...
## Using 39 trees...
## Using 44 trees...
## Using 55 trees...
## Using 106 trees...
## Using 70 trees...
## Using 99 trees...
## Using 823 trees...
## Using 105 trees...
## Using 664 trees...
## Using 384 trees...
## Using 968 trees...
## Using 74 trees...
## Using 961 trees...
```

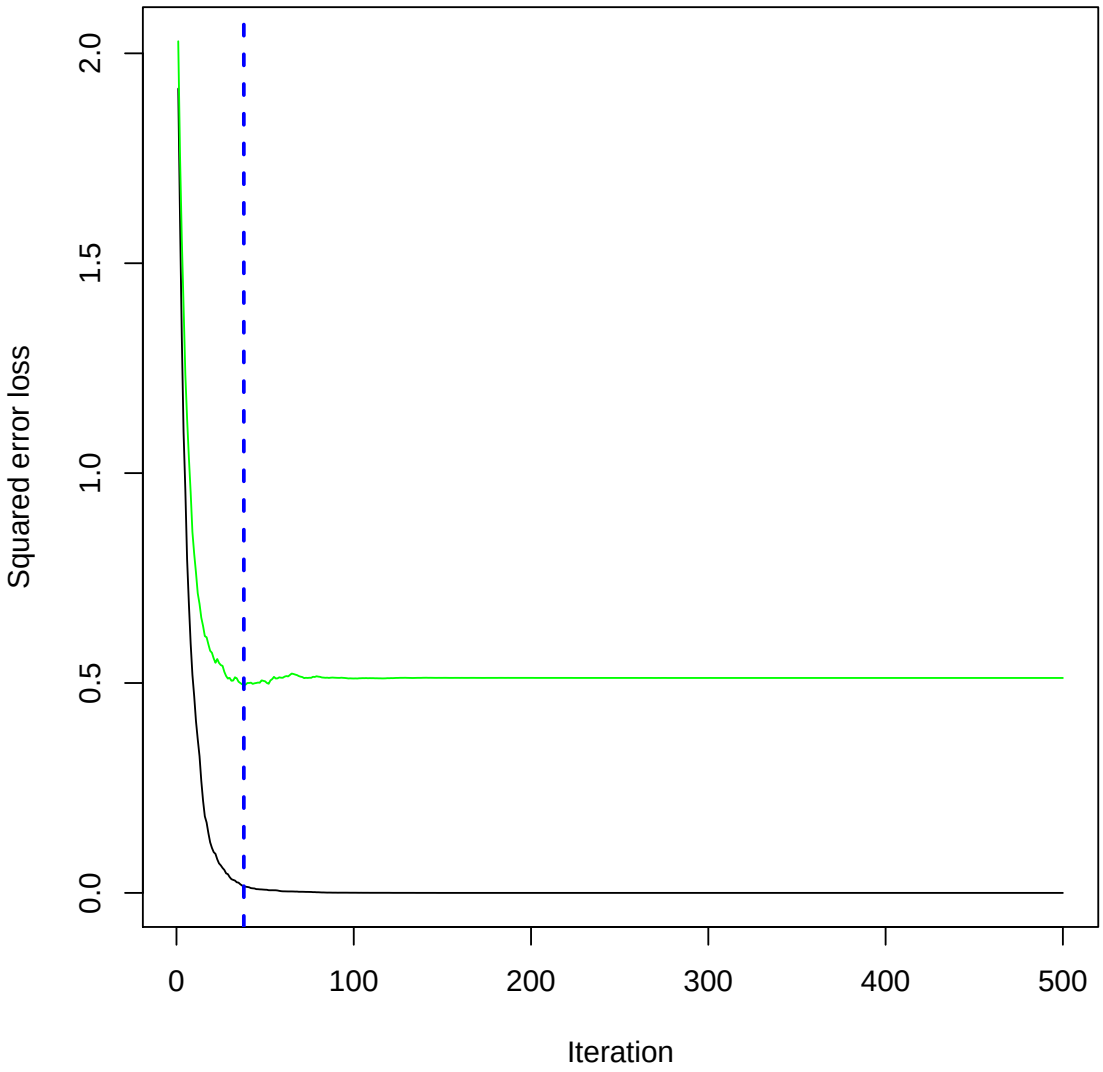
```
tn.gb
```

```
##
## Parameter tuning of 'my.gbm':
```

```
##
## - sampling method: 2-fold cross validation
##
## - best parameters:
##  interaction.depth n.minobsinnode
##                  3              1
##
## - best performance: 0.04

trash <- capture.output({
    tn.gb <- tune(my.gbm, Species ~ ., data = iris,
        distribution = "multinomial", n.trees = 1000,
        shrinkage = 0.05, bag.fraction = 0.5,
        train.fraction = 0.66, n.cores = 4,
        ranges = list(interaction.depth = 1:5,
            n.minobsinnode = c(1, 3, 5, 10)),
        tunecontrol = tune.control(cross = 10,
            best.model = TRUE), predict.func = my.predict.gbm)
})
tn.gb

##
## Parameter tuning of 'my.gbm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  interaction.depth n.minobsinnode
##                  5              3
##
## - best performance: 0.02666667
```

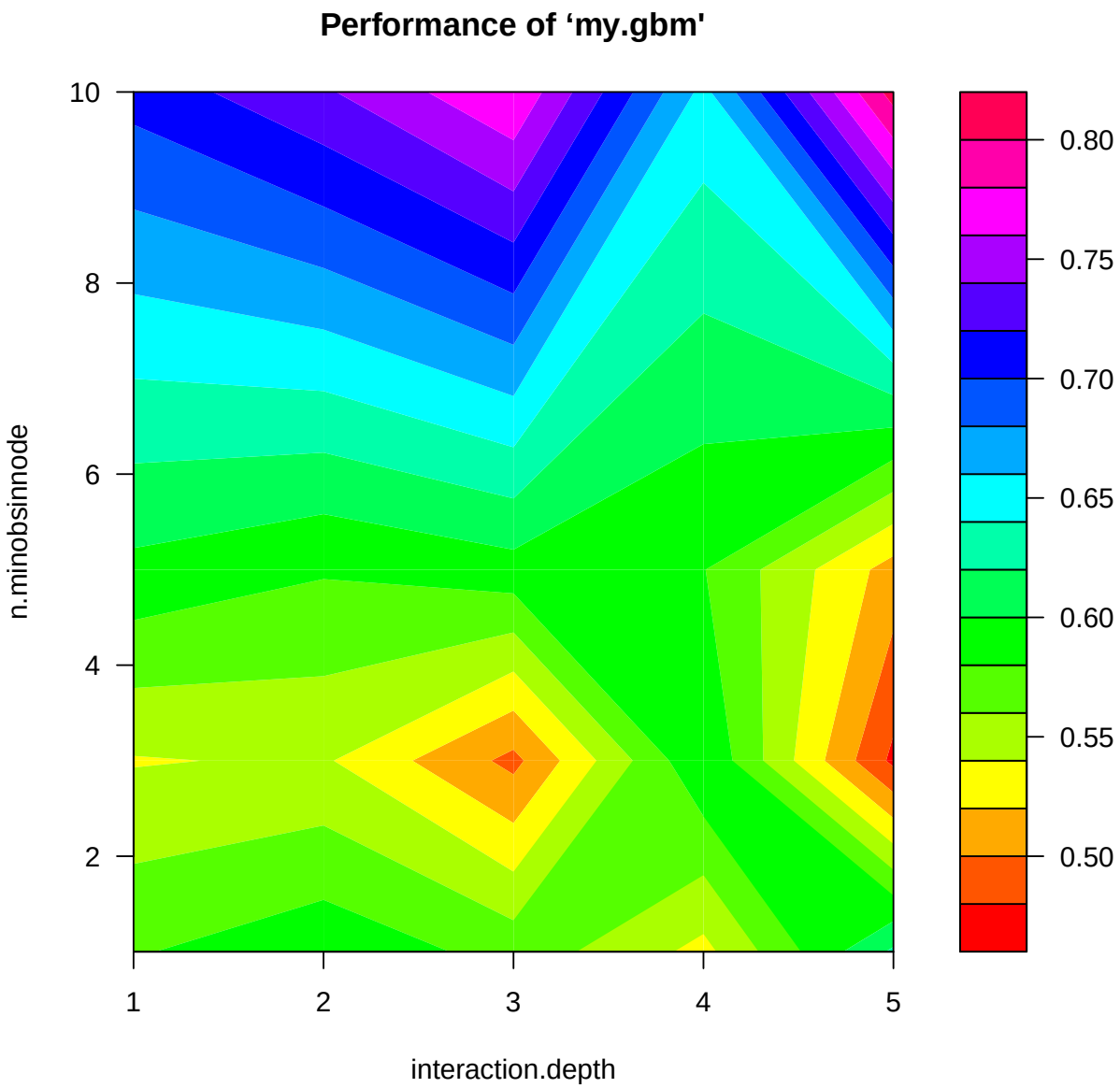
```
# gasoline
data(gasoline)
names(gasoline)

## [1] "octane" "NIR"

dim(gasoline)

## [1] 60  2

class(gasoline$NIR) <- NULL
colnames(gasoline$NIR) <- paste("S", seq(900,
    1700, 2), sep = "")
gasoline <- cbind(subset(gasoline, select = octane),
    as.data.frame(gasoline$NIR))
dim(gasoline)
```

```
## [1]  60 402

gasoline <- gasoline[sample(nrow(gasoline)),
    ]
gb <- gbm(octane ~ ., data = gasoline, distribution = "gaussian",
    n.trees = 500, shrinkage = 0.1, bag.fraction = 0.5,
    cv.folds = 10, n.cores = 4, interaction.depth = 10,
    n.minobsinnode = 1)
best.n.trees <- gbm.perf(gb)

## Using cv method...
```

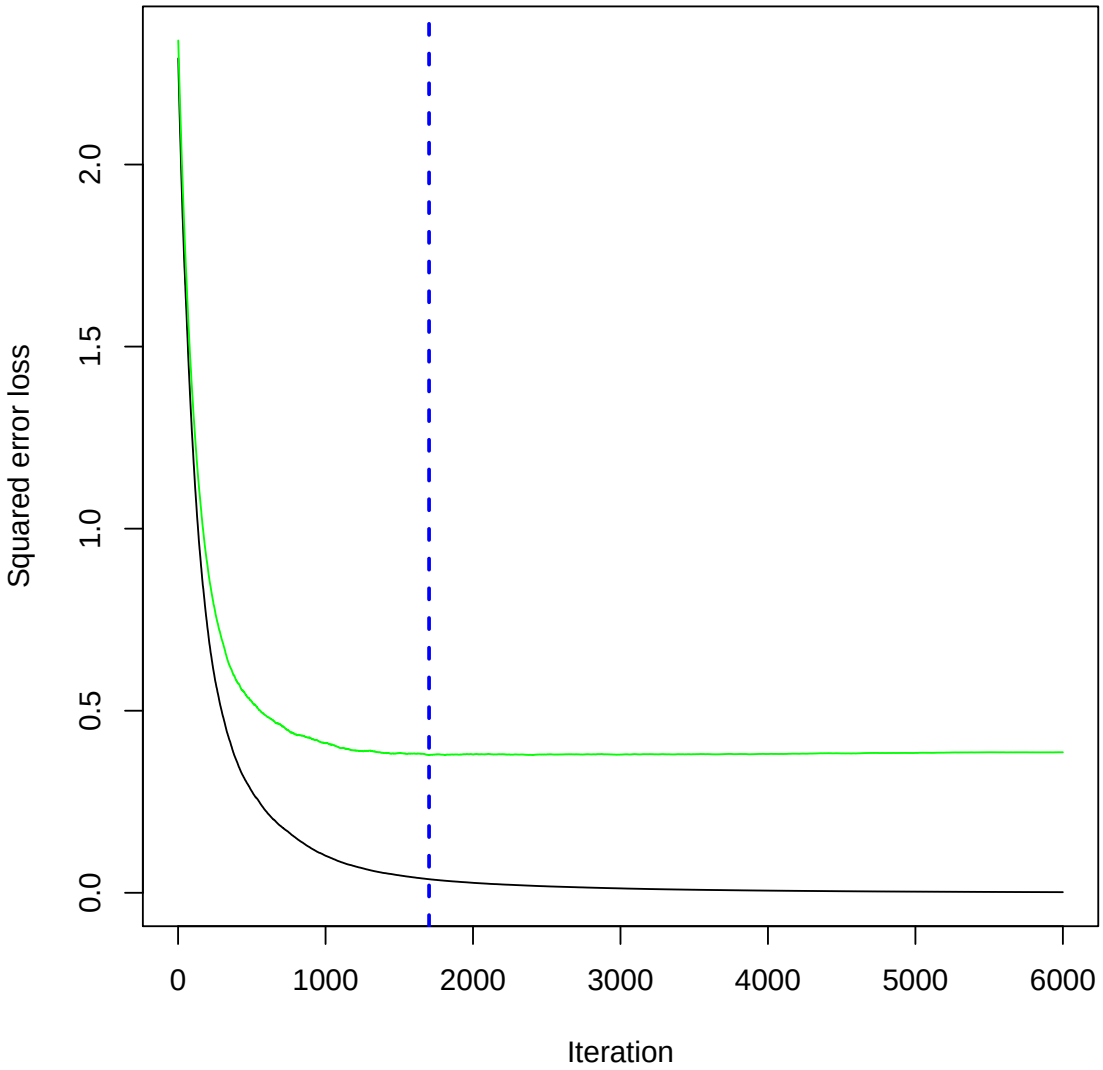



```
best.n.trees

## [1] 38
```

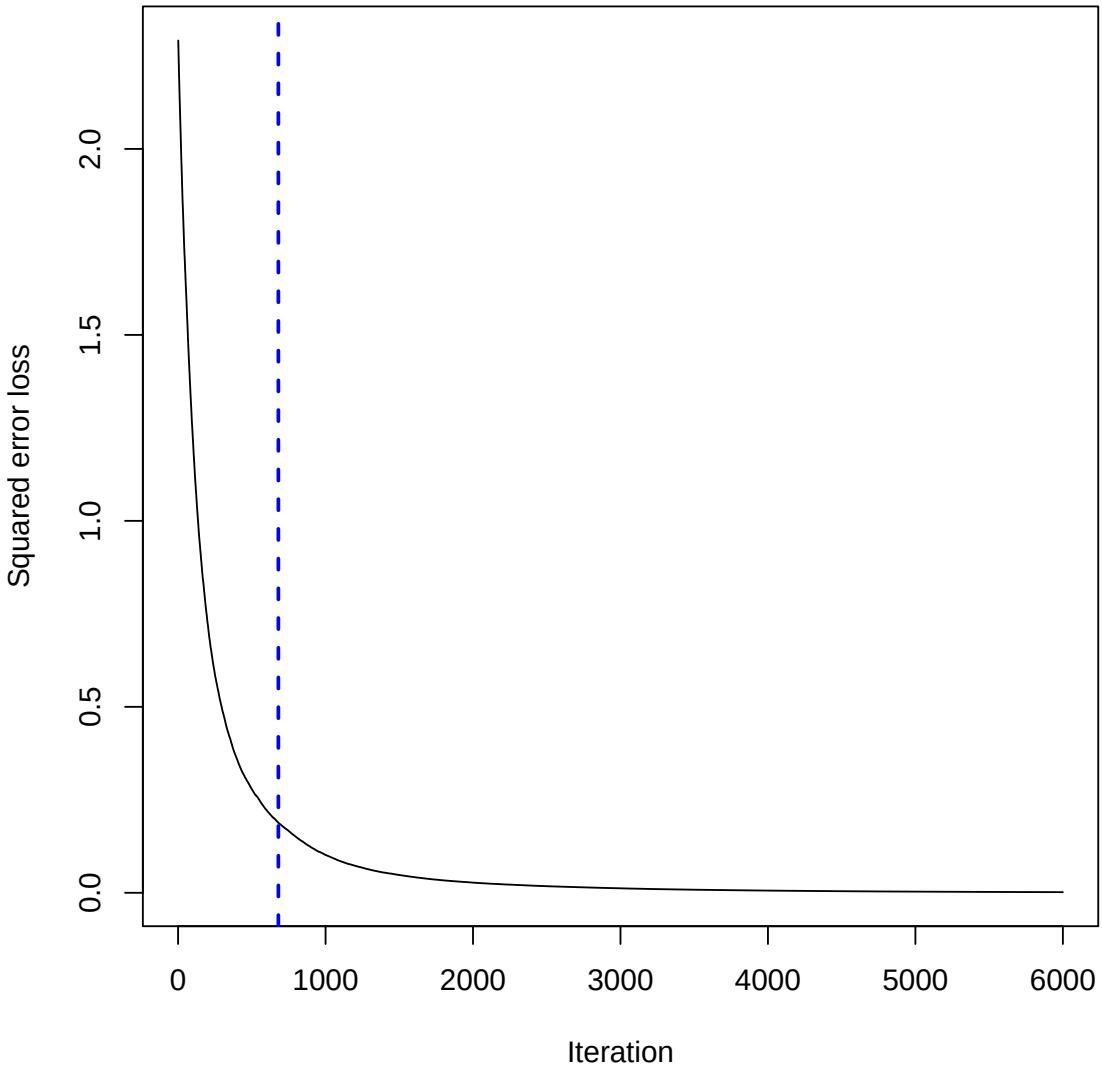
```r
trash <- capture.output({
    tn.gb <- tune(my.gbm, octane ~ ., data = gasoline,
        distribution = "gaussian", n.trees = 1000,
        shrinkage = 0.05, bag.fraction = 0.5,
        train.fraction = 0.8, n.cores = 4,
        ranges = list(interaction.depth = 1:5,
            n.minobsinnode = c(1, 3, 5, 10)),
        tunecontrol = tune.control(cross = 10,
            best.model = TRUE), predict.func = predict)
})
tn.gb

##
## Parameter tuning of 'my.gbm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  interaction.depth n.minobsinnode
##                  5              3
##
## - best performance: 0.4752277

plot(tn.gb, color.palette = rainbow)
```




```
gb <- gbm(octane ~ ., data = gasoline, distribution = "gaussian",
    n.trees = 6000, shrinkage = 0.005, bag.fraction = 0.5,
    cv.folds = 10, n.cores = 4, interaction.depth = 1,
    n.minobsinnode = 1)
best.n.trees <- gbm.perf(gb)

## Using cv method...
```

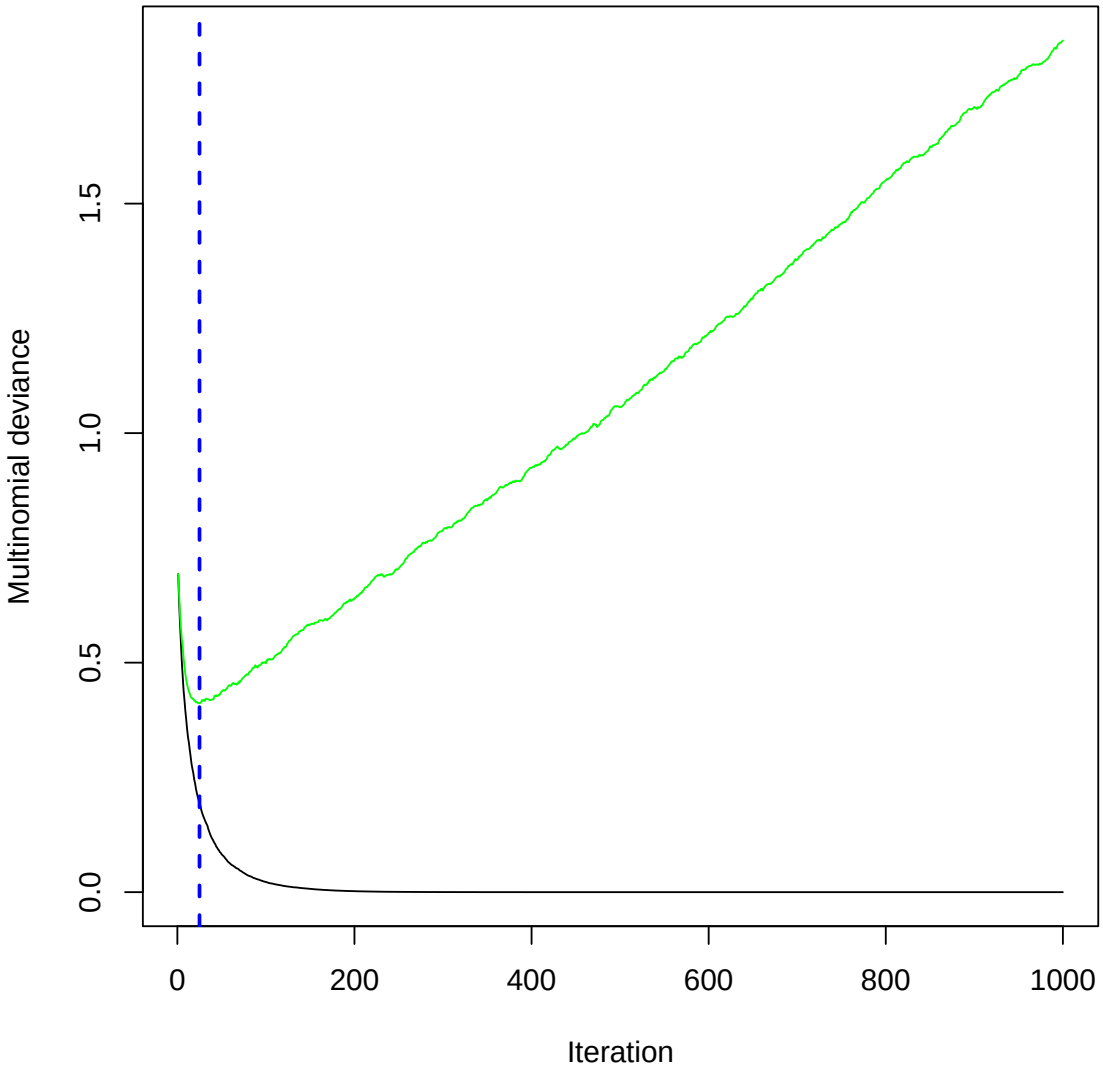



```
best.n.trees

## [1] 1702

gbm.perf(gb, method = "OOB")

## Warning in gbm.perf(gb, method = "OOB"): OOB generally underestimates the optima
number of iterations although predictive performance is reasonably competitive.
Using cv.folds>0 when calling gbm usually results in improved predictive performanc
```



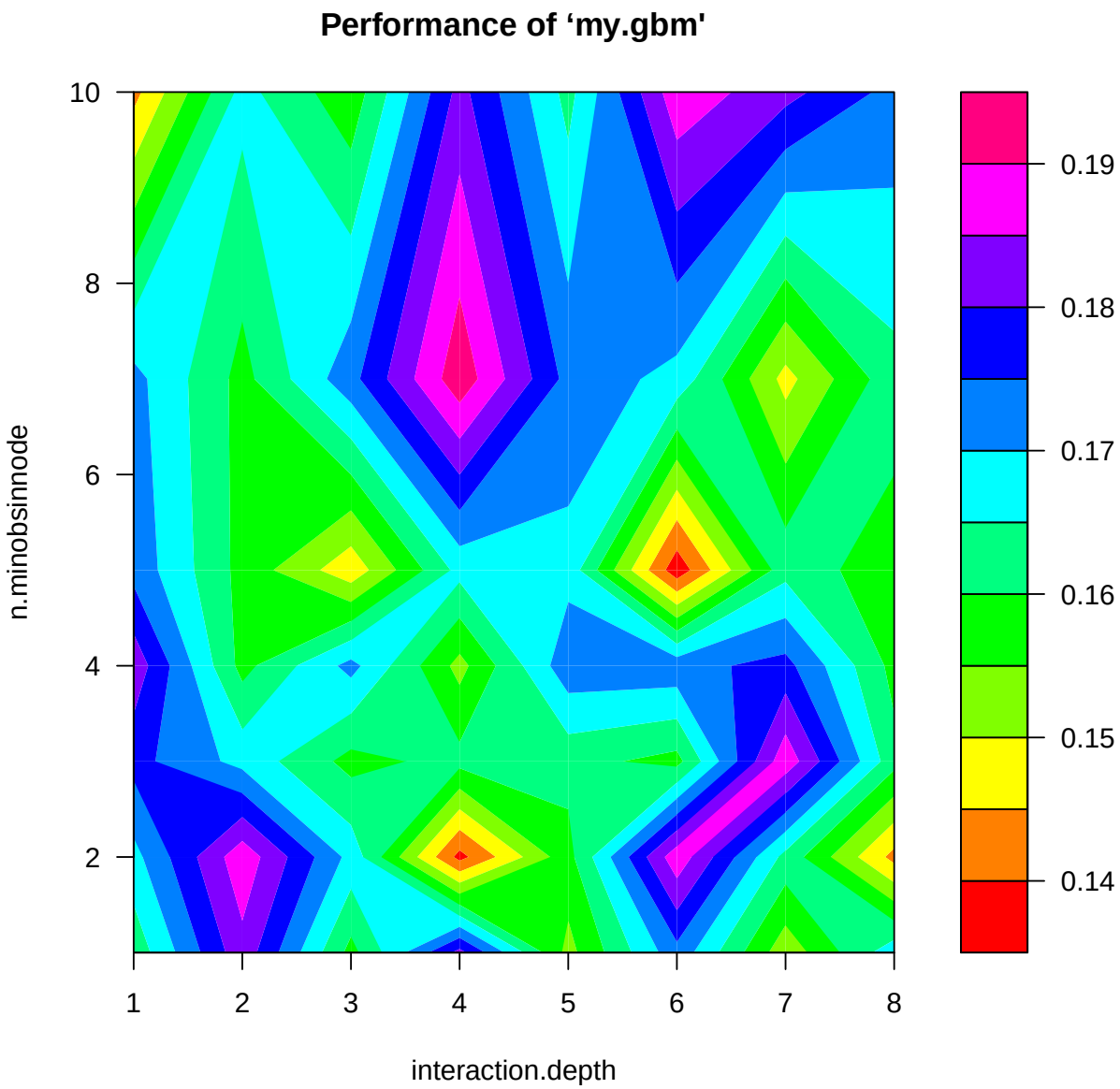

```
## [1] 680

trash <- capture.output({
    tn.gb <- tune(my.gbm, octane ~ ., data = gasoline,
        distribution = "gaussian", n.trees = best.n.trees,
        shrinkage = 0.005, bag.fraction = 0.5,
        train.fraction = 1, n.cores = 4,
        interaction.depth = 1, n.minobsinnode = 1,
        tunecontrol = tune.control(cross = 10,
            best.model = TRUE), predict.func = function(x,
            ...) predict(x, ..., n.trees = x$n.trees))
})
tn.gb

##
## Error estimation of 'my.gbm' using 10-fold cross validation: 0.3310305
```

```r
# GlaucomaM
glaucomaM <- read.table("svm/GlaucomaM.txt",
    stringsAsFactors = TRUE, header = TRUE,
    sep = " ", strip.white = TRUE, blank.lines.skip = TRUE)
table(glaucomaM$Class)

##
## glaucoma   normal
##       98       98

gb <- gbm(Class ~ ., data = glaucomaM, distribution = "multinomial",
    n.trees = 1000, shrinkage = 0.05, bag.fraction = 0.5,
    cv.folds = 10, n.cores = 4, interaction.depth = 5,
    n.minobsinnode = 5)
best.n.trees <- gbm.perf(gb)

## Using cv method...
```



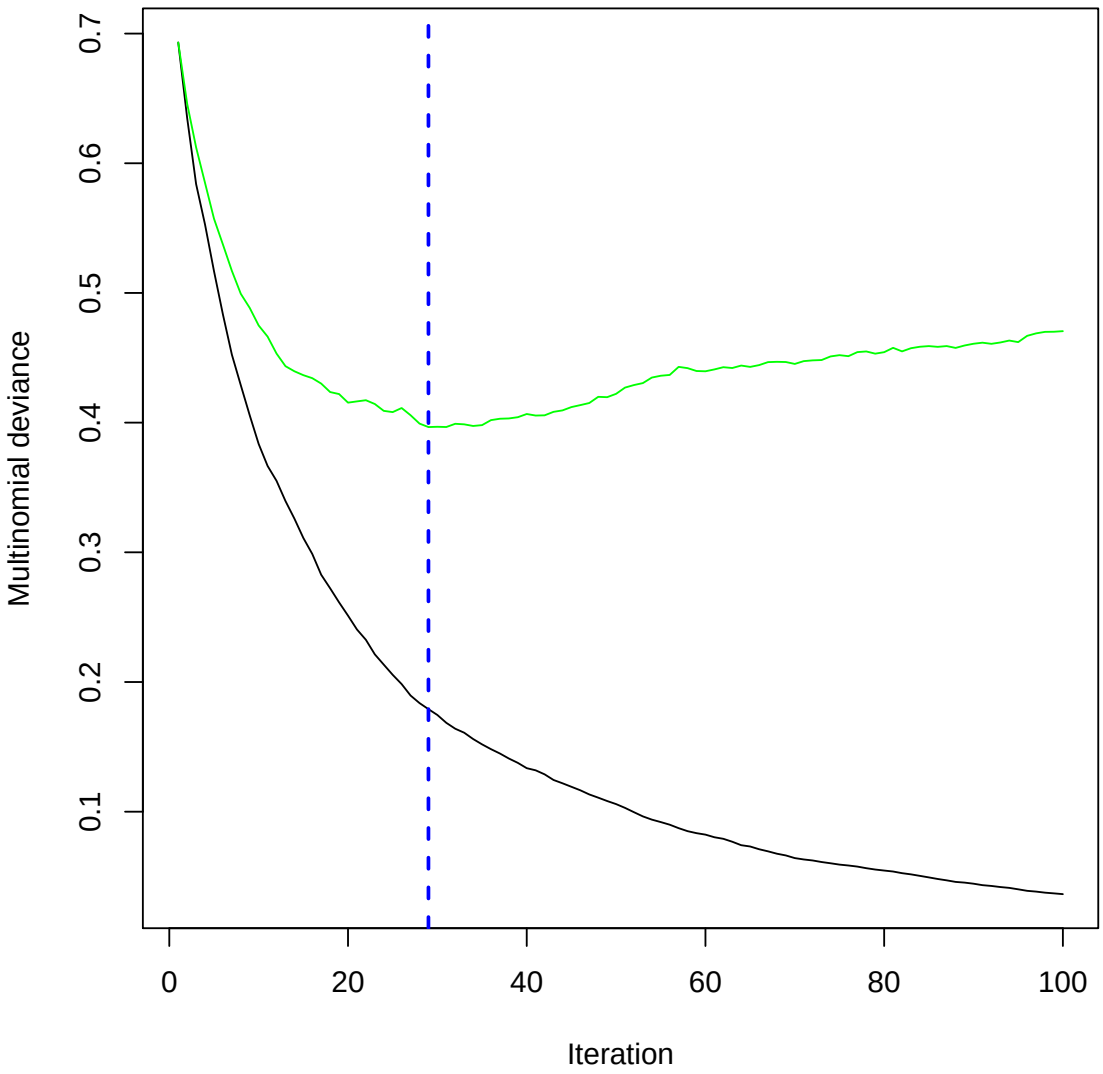

```
best.n.trees

## [1] 25

# OR gb <- gbm(as.numeric(Class) ~ .,
# data = glaucomaM, distribution =
# 'bernoulli', n.trees = 1000, shrinkage
# = 0.05, bag.fraction = 0.5, cv.folds =
# 10, n.cores = 4, interaction.depth = 5,
# n.minobsinnode = 5)
table(actual = glaucomaM$Class, predicted = my.predict.gbm(gb))

## Using 25 trees...
##           predicted
## actual     glaucoma normal
```

```
##   glaucoma       93      5
##   normal          4     94

# Not bad!
# Althought....
tn.gb <- tune(my.gbm, Class ~ ., data = glaucomaM,
    distribution = "multinomial", n.trees = best.n.trees,
    shrinkage = 0.05, bag.fraction = 0.5,
    train.fraction = 1, n.cores = 4, interaction.depth = 5,
    n.minobsinnode = 5, tunecontrol = tune.control(cross = 50,
        best.model = TRUE), predict.func = function(x,
        ...) my.predict.gbm(x, ..., n.trees = x$n.trees))
tn.gb

##
## Error estimation of 'my.gbm' using 50-fold cross validation: 0.1766667

trash <- capture.output({
    tn.gb <- tune(my.gbm, Class ~ ., data = glaucomaM,
        distribution = "multinomial", n.trees = best.n.trees,
        shrinkage = 0.05, bag.fraction = 0.5,
        train.fraction = 1, n.cores = 4,
        ranges = list(interaction.depth = 1:8,
            n.minobsinnode = c(1:5, 7, 10)),
        tunecontrol = tune.control(cross = 50,
            best.model = TRUE), predict.func = function(x,
            ...) my.predict.gbm(x, ..., n.trees = x$n.trees))
})
tn.gb

##
## Parameter tuning of 'my.gbm':
##
## - sampling method: 50-fold cross validation
##
## - best parameters:
##  interaction.depth n.minobsinnode
##                  6              5
##
## - best performance: 0.1366667

plot(tn.gb, color.palette = rainbow)
```

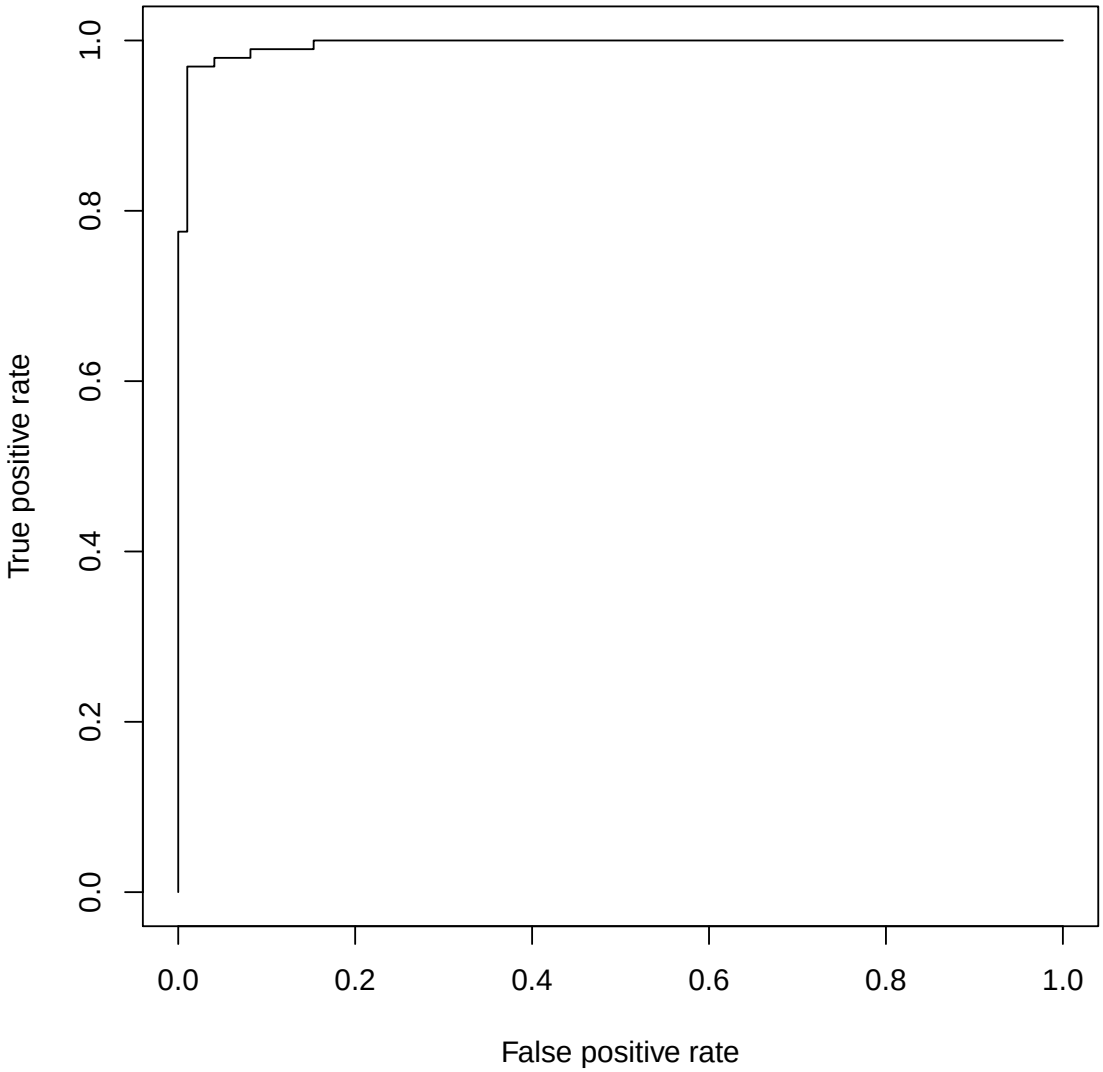



```
tn.gb <- tune(my.gbm, Class ~ ., data = glaucomaM,
    distribution = "multinomial", n.trees = best.n.trees,
    shrinkage = 0.05, bag.fraction = 0.5,
    train.fraction = 1, n.cores = 4, interaction.depth = 4,
    n.minobsinnode = 1, tunecontrol = tune.control(cross = 50,
        best.model = TRUE), predict.func = function(x,
        ...) my.predict.gbm(x, ..., n.trees = x$n.trees))
tn.gb

##
## Error estimation of 'my.gbm' using 50-fold cross validation: 0.135

gb <- gbm(Class ~ ., data = glaucomaM, distribution = "multinomial",
    n.trees = 100, shrinkage = 0.05, bag.fraction = 0.5,
    cv.folds = 10, n.cores = 4, interaction.depth = 4,
```

```
    n.minobsinnode = 2)
gbm.perf(gb)

## Using cv method...
```

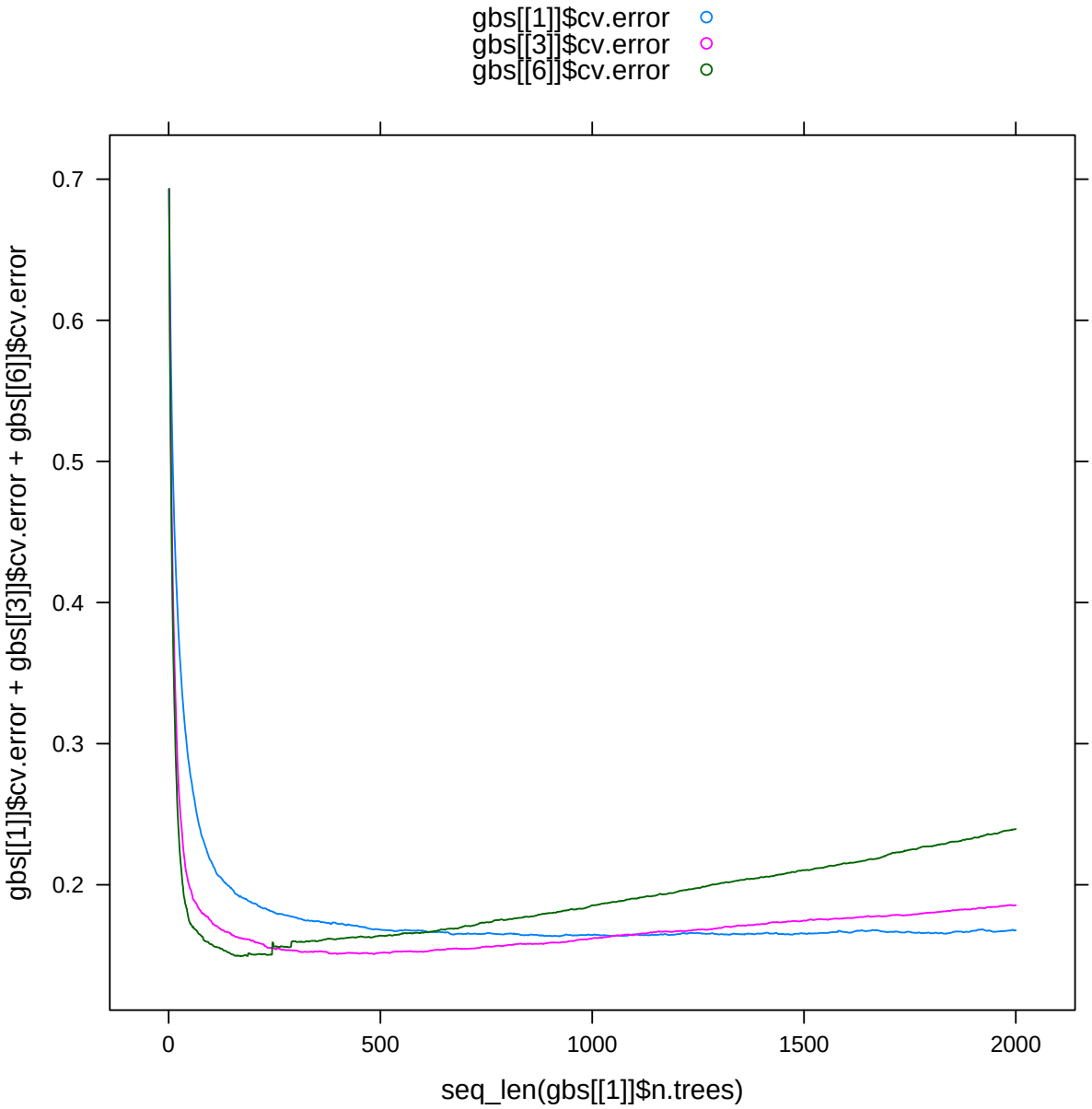



```
## [1] 29

prob <- predict(gb)[, 2, 1]

## Using 29 trees...

source("class.R")
plot(ROC(prob, glaucomaM$Class))
```



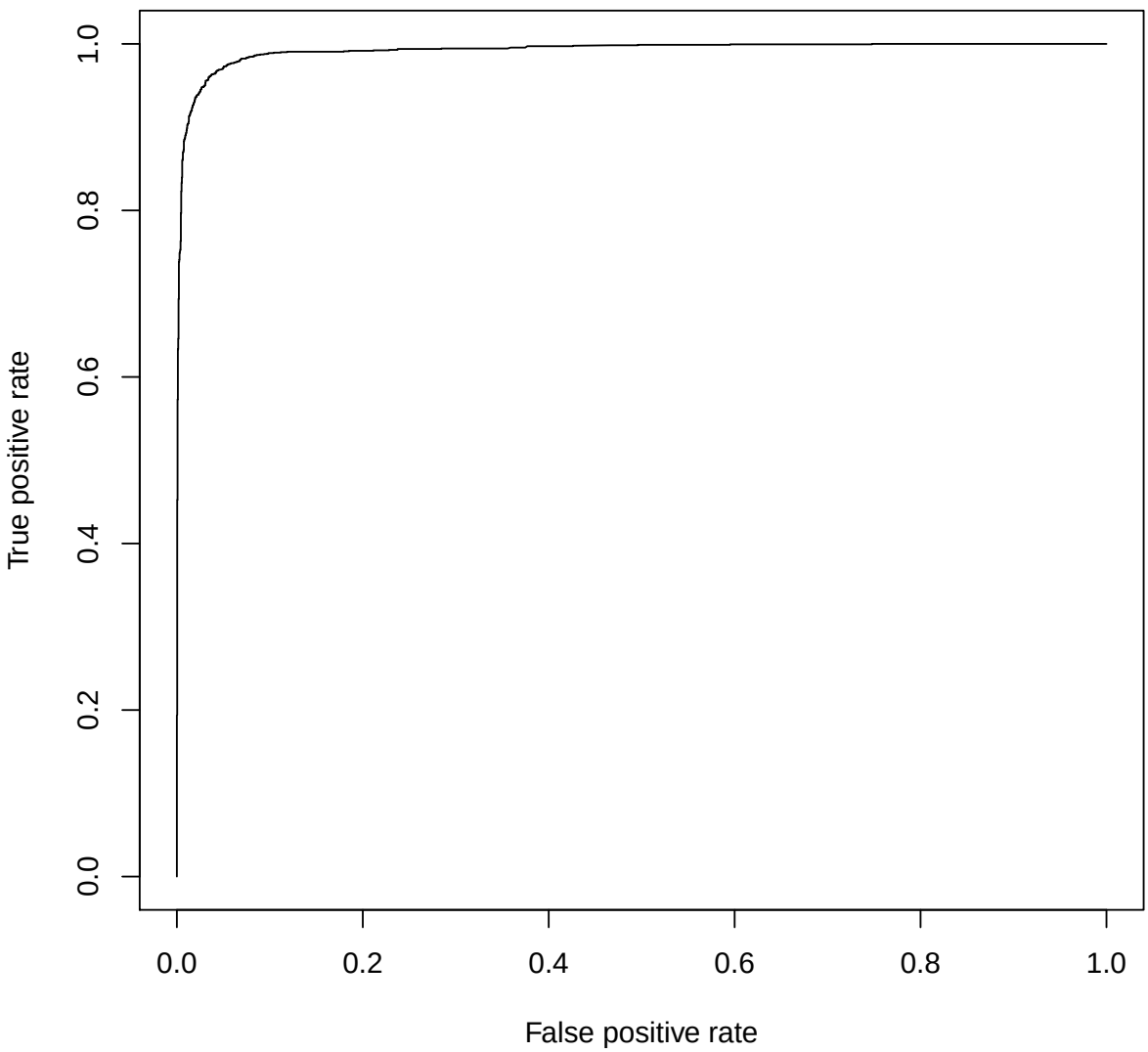

```
library(kernlab)
data(spam)
gbs <- lapply(1:6, function(id) gbm(type ~
    ., data = spam, distribution = "multinomial",
    n.trees = 2000, shrinkage = 0.05, bag.fraction = 0.5,
    cv.folds = 2, n.cores = 4, interaction.depth = id,
    n.minobsinnode = 1))
gbm.perf(gbs[[1]], plot.it = FALSE)

## Using cv method...
## [1] 923

gbm.perf(gbs[[3]], plot.it = FALSE)

## Using cv method...
## [1] 484
```

```r
gbm.perf(gbs[[6]], plot.it = FALSE)

## Using cv method...
## [1] 169

xyplot(gbs[[1]]$cv.error + gbs[[3]]$cv.error +
    gbs[[6]]$cv.error ~ seq_len(gbs[[1]]$n.trees),
    type = "l", auto.key = list(type = "l"))
```



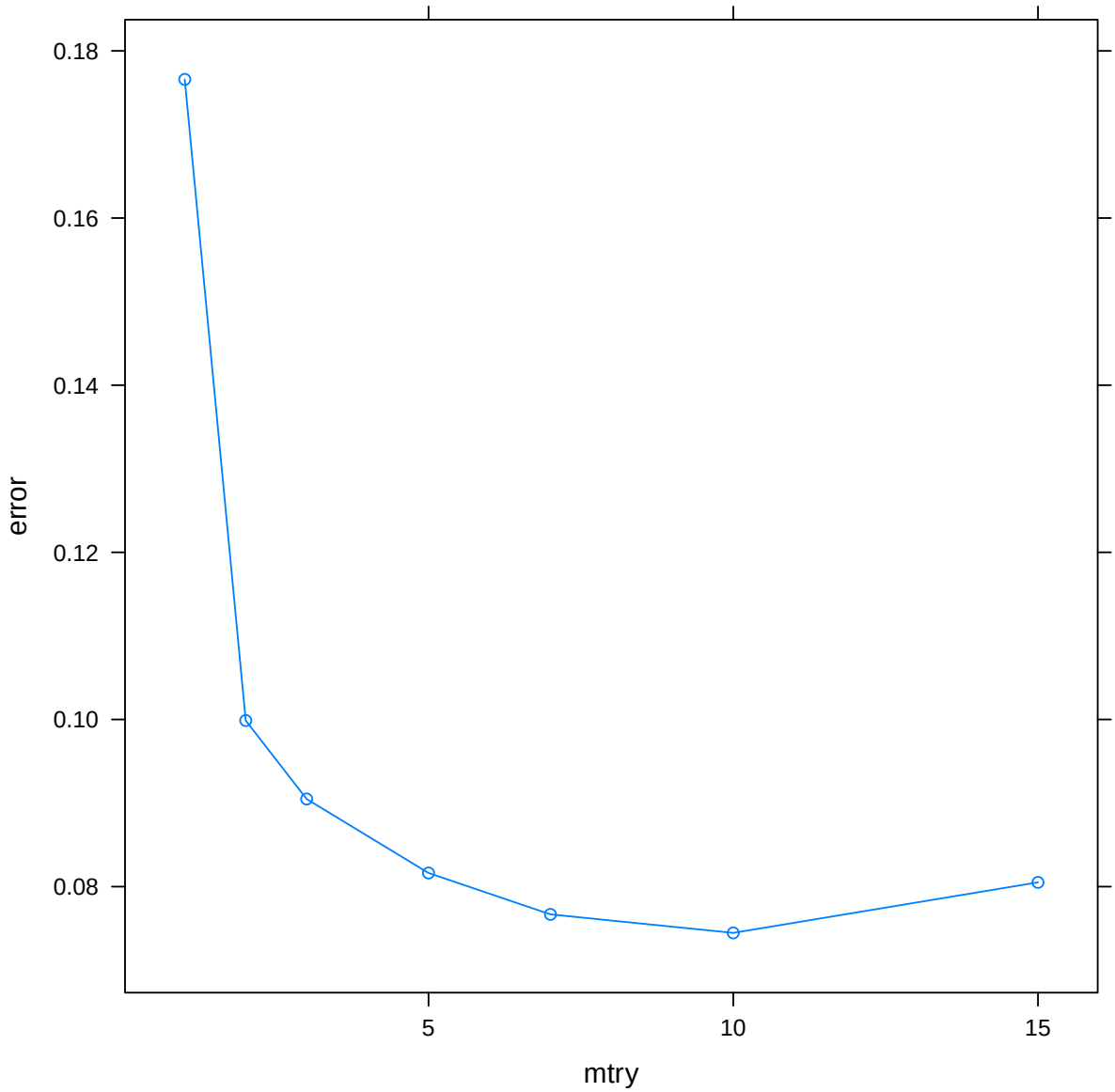

```r
best.n.trees <- gbm.perf(gbs[[6]], plot.it = FALSE)

## Using cv method...

best.n.trees

## [1] 169
```

```r
table(actual = spam$type, predicted = my.predict.gbm(gbs[[6]]))

## Using 99 trees...
##          predicted
## actual    nonspam spam
##   nonspam    2718   70
##   spam        104 1709

prob <- predict(gbs[[6]])[, 2, 1]

## Using 99 trees...

source("class.R")
plot(ROC(prob, spam$type))
```

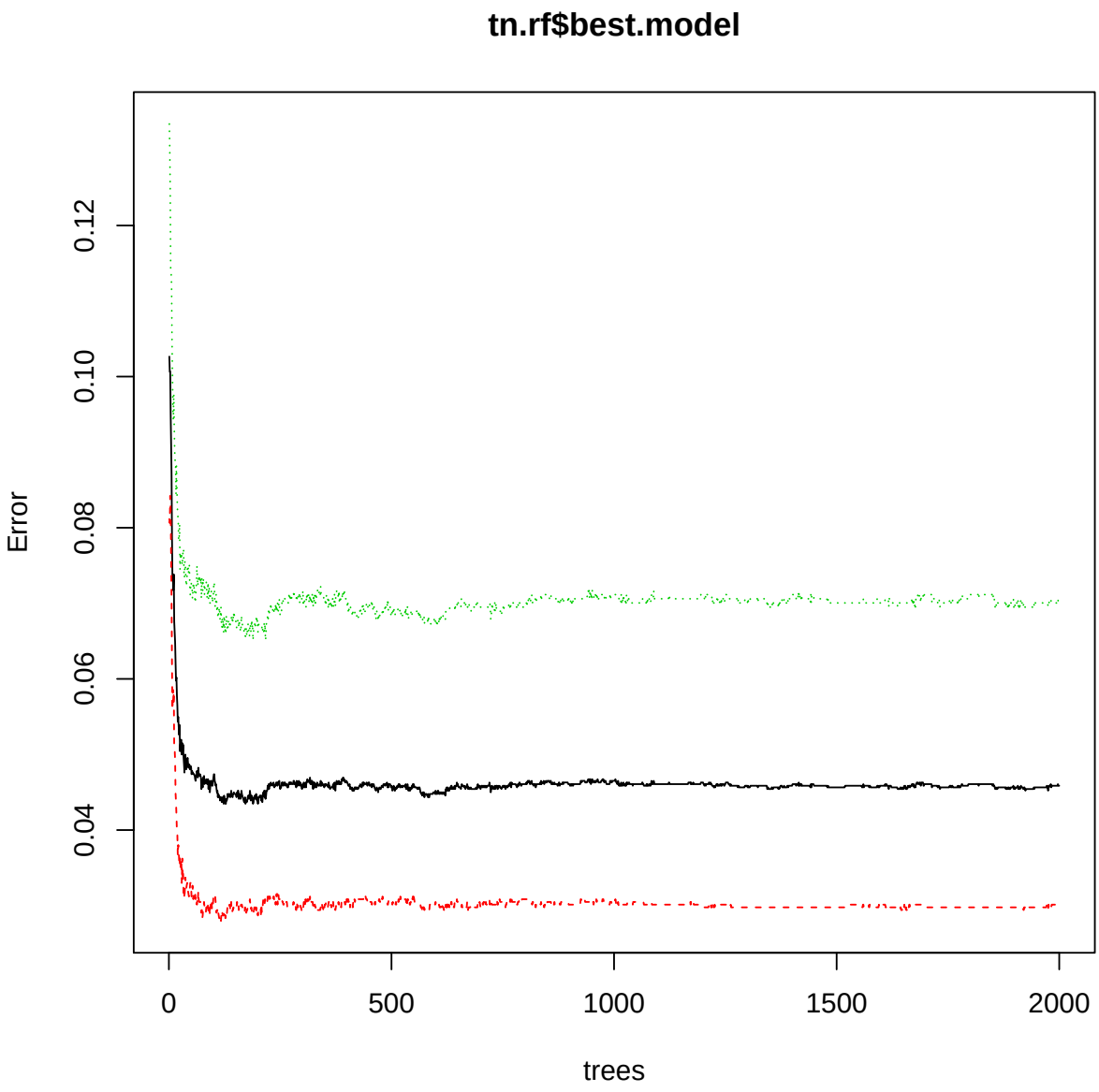


```r
tn.gb <- tune(my.gbm, type ~ ., data = spam,
    distribution = "multinomial", n.trees = best.n.trees,
    shrinkage = 0.05, bag.fraction = 0.5,
    cv.folds = 0, n.cores = 4, interaction.depth = 6,
    n.minobsinnode = 5, predict.func = function(x,
        ...) my.predict.gbm(x, ..., n.trees = x$n.trees))
tn.gb

##
## Error estimation of 'my.gbm' using 10-fold cross validation: 0.04868433
```

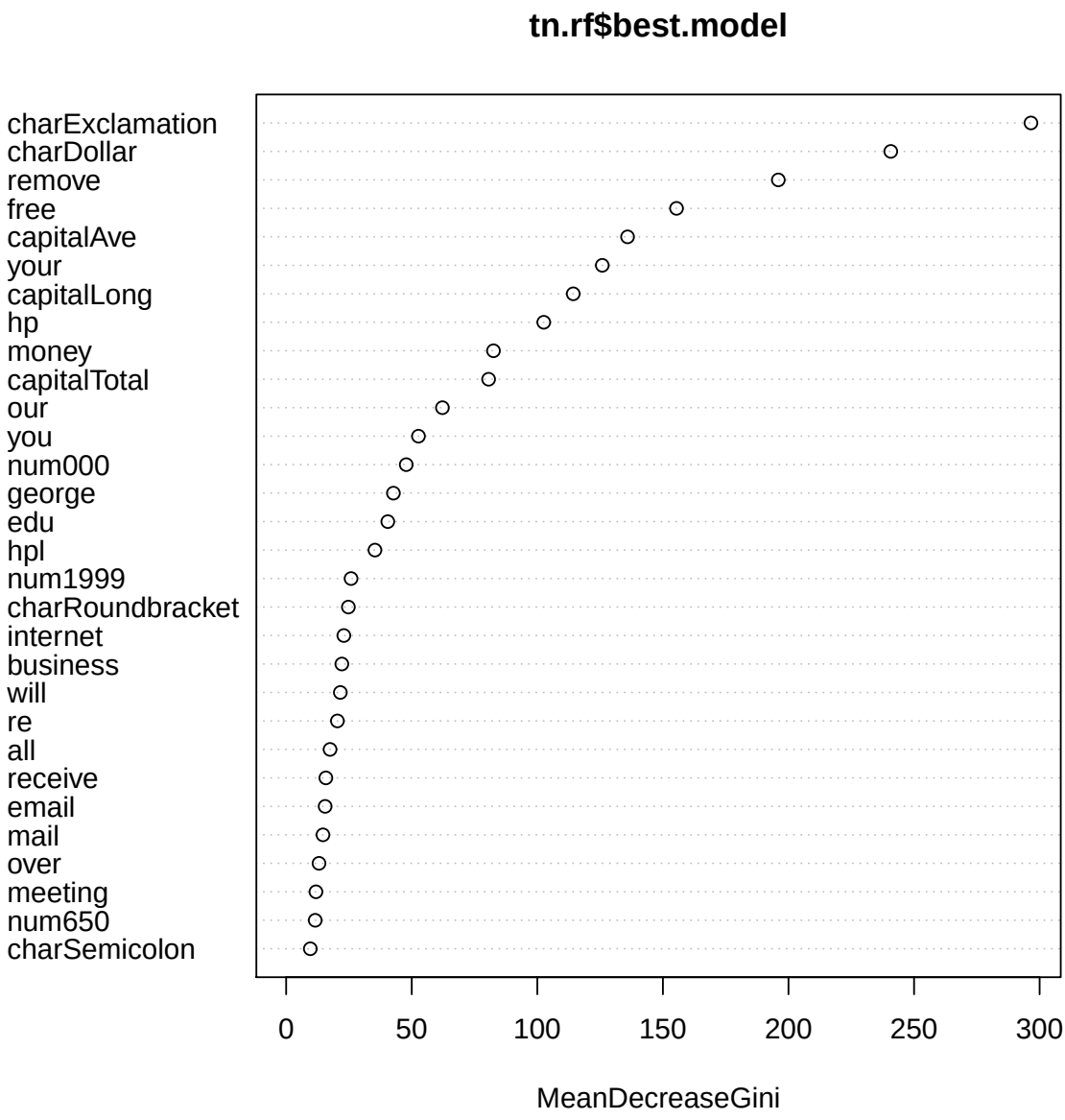
```r
library(kernlab)
data(spam)
tn.rf <- tune.randomForest(type ~ ., data = spam,
    mtry = c(1:3, 5, 7, 10, 15), ntree = 2000,
    tunecontrol = tune.control(cross = 2,
        error.fun = error.fun.max))
tn.rf

##
## Parameter tuning of 'randomForest':
##
## - sampling method: 2-fold cross validation
##
## - best parameters:
##  mtry ntree
##    10  2000
##
## - best performance: 0.07446275

xyplot(error ~ mtry, data = as.data.frame(tn.rf$performances),
    type = "b")
```



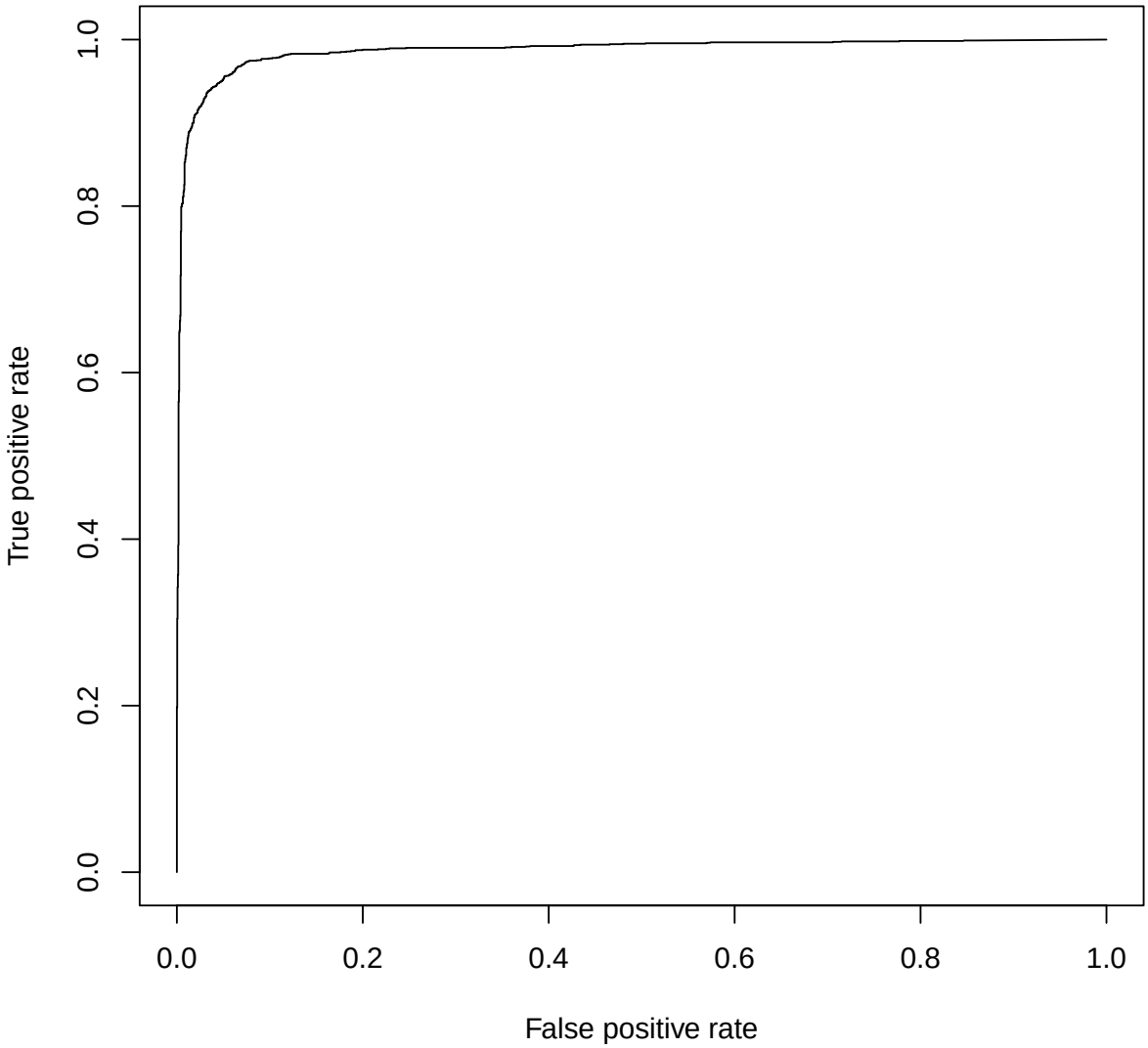

```
tn.rf$best.model

##
## Call:
##  best.randomForest(x = type ~ ., data = spam, mtry = c(1:3, 5,      7, 10, 15),
##                Type of random forest: classification
##                      Number of trees: 2000
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 4.59%
## Confusion matrix:
##         nonspam spam class.error
## nonspam    2705   83  0.02977044
## spam        128 1685  0.07060121

plot(tn.rf$best.model)
```




```
varImpPlot(tn.rf$best.model)
```

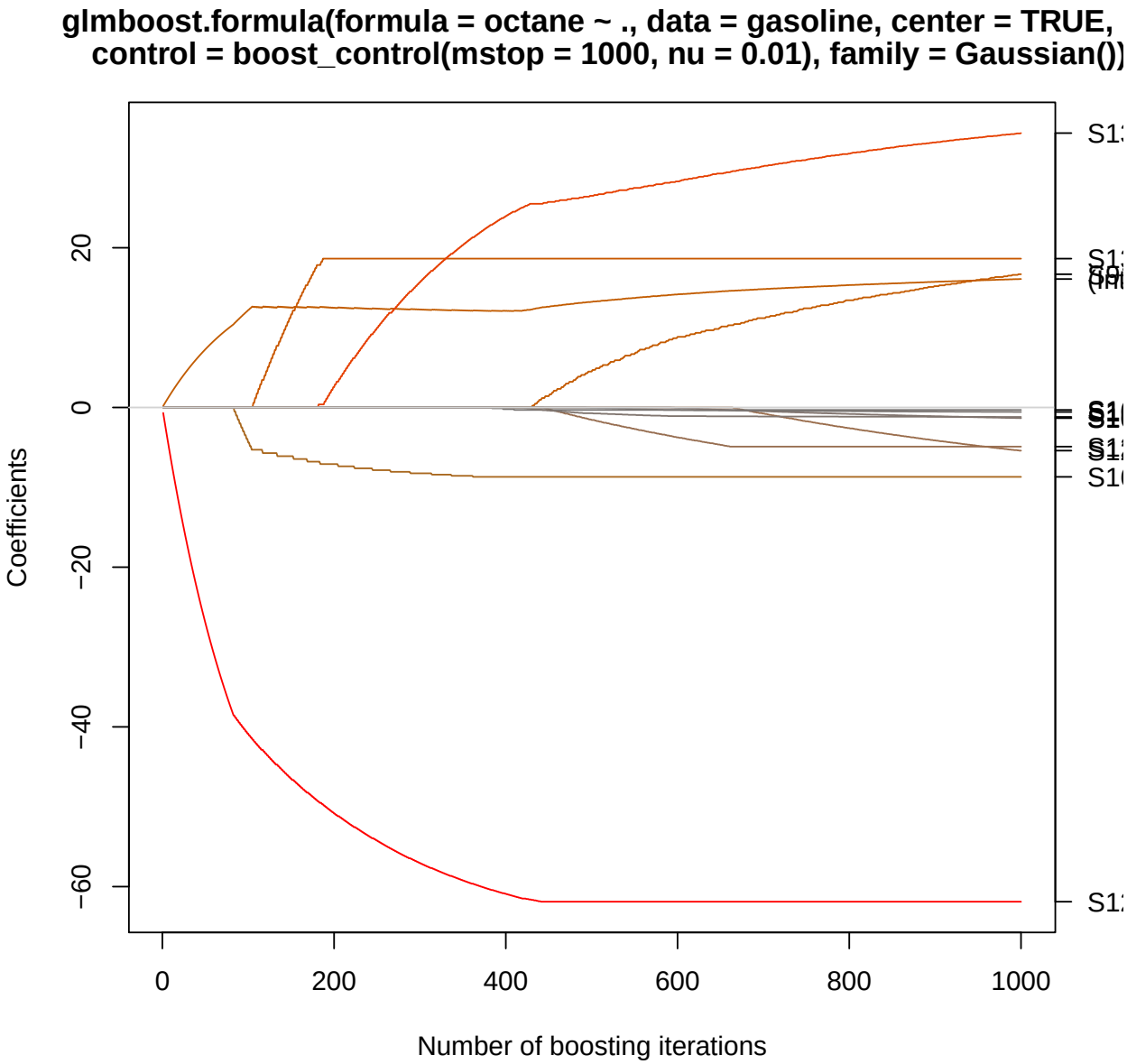



```
table(actual = spam$type, predicted = my.predict.gbm(gb))

## Using 394 trees...
##          predicted
## actual    nonspam spam
##   nonspam    2775   13
##   spam         45 1768

prob <- predict(tn.rf$best.model, type = "prob")[,
    2]
source("class.R")
plot(ROC(prob, spam$type))
```



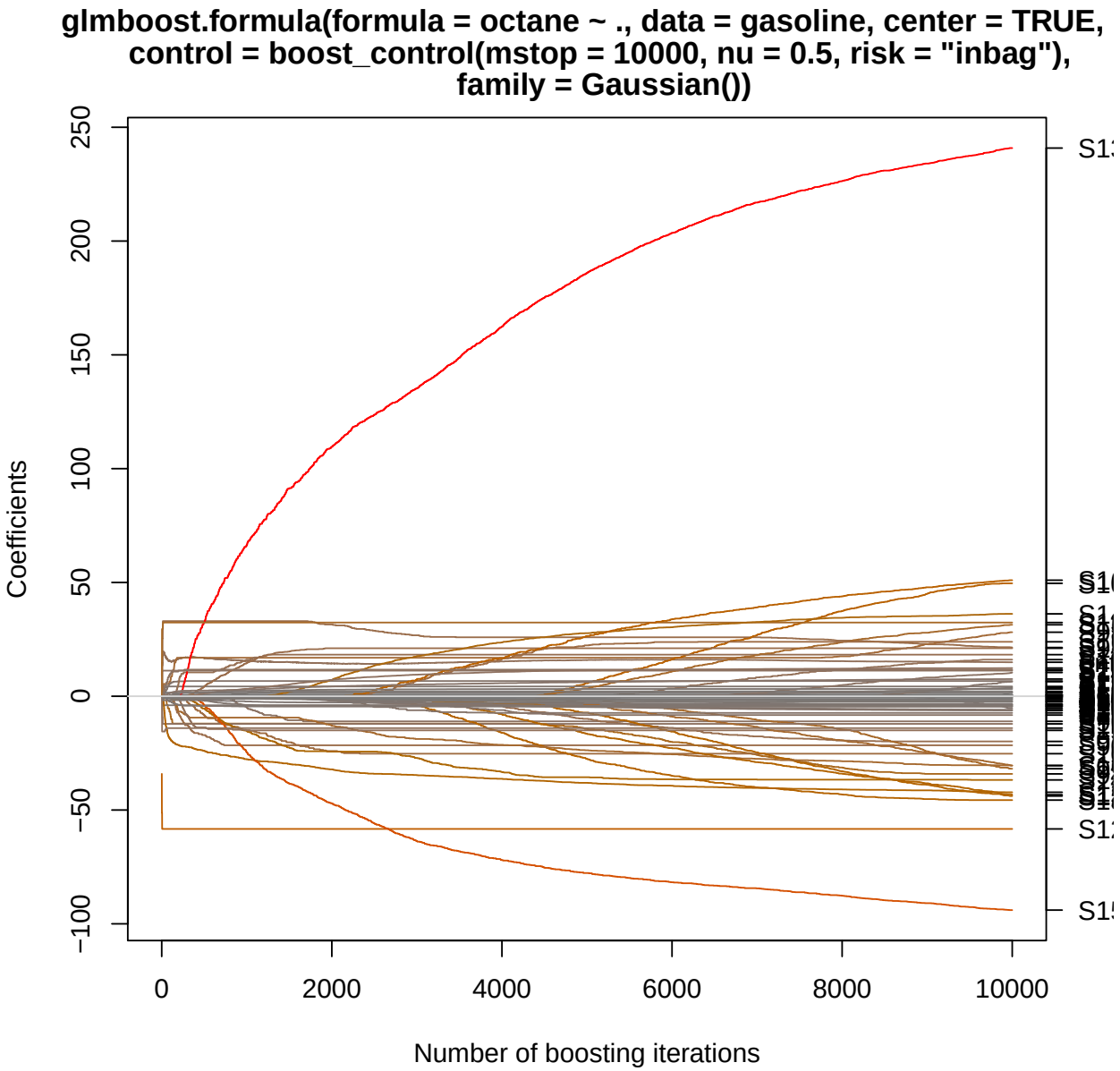

```
library(e1071)
library(lattice)
library(latticeExtra)
library(pls)
library(MASS)
library(mboost)  # http://cran.r-project.org/web/packages/mboost/vignettes/mboost_t
# http://cran.r-project.org/web/packages/mboost/vignettes/mboost_illustrations.pdf
# gasoline
data(gasoline)
names(gasoline)

## [1] "octane" "NIR"

dim(gasoline)

## [1] 60  2
```

```r
class(gasoline$NIR) <- NULL
colnames(gasoline$NIR) <- paste("S", seq(900,
    1700, 2), sep = "")
gasoline <- cbind(subset(gasoline, select = octane),
    as.data.frame(gasoline$NIR))
dim(gasoline)

## [1]  60 402

mb <- glmboost(octane ~ ., data = gasoline,
    center = TRUE, family = Gaussian(), control = boost_control(mstop = 1000,
        nu = 0.01))
summary(mb)

##
##   Generalized Linear Models Fitted via Gradient Boosting
##
## Call:
## glmboost.formula(formula = octane ~ ., data = gasoline, center = TRUE,     contr
##
##
##   Squared Error (Regression)
##
## Loss function: (y - f)^2
##
##
## Number of boosting iterations: mstop = 1000
## Step size:  0.01
## Offset:  87.1775
##
## Coefficients:
## (Intercept)        S912       S1208       S1224       S1228
##  16.0822831  16.6853056 -61.8895373  -4.9017664  -5.4066460
##       S1360       S1362       S1634       S1636       S1690
##  18.6401746  34.3585005  -8.6901908  -0.3105508  -1.3376806
##       S1692       S1698
##  -0.5632201  -1.1897509
## attr(,"offset")
## [1] 87.1775
##
## Selection frequencies:
## S1208 S1362 S1228  S912 S1690 S1224 S1360 S1698 S1692 S1634
## 0.251 0.240 0.118 0.107 0.080 0.076 0.038 0.034 0.029 0.024
## S1636
## 0.003

plot(mb)
```




```
my.glmboost <- function(..., data, subset,
    mstop, nu = 0.01, risk = "inbag") {
    data <- data[subset, ]
    glmboost(..., data = data, control = boost_control(mstop = mstop,
        risk = risk, nu = nu))
}
tn.mb <- tune(my.glmboost, octane ~ ., data = gasoline,
    center = TRUE, family = Gaussian(), ranges = list(mstop = seq(1,
        10000, length.out = 5)), tunecontrol = tune.control(cross = 2))
tn.mb

##
## Parameter tuning of 'my.glmboost':
##
## - sampling method: 2-fold cross validation
```

```
##
## - best parameters:
##    mstop
##  2500.75
##
## - best performance: 0.07067316

mb <- glmboost(octane ~ ., data = gasoline,
    center = TRUE, family = Gaussian(), control = boost_control(mstop = 10000,
        nu = 0.01))
best.mstop <- mstop(AIC(mb))
best.mstop

## [1] 9955

tn.mb <- tune(my.glmboost, octane ~ ., data = gasoline,
    center = TRUE, family = Gaussian(), mstop = best.mstop)
tn.mb

##
## Error estimation of 'my.glmboost' using 10-fold cross validation: 0.05579574

mb <- glmboost(octane ~ ., data = gasoline,
    center = TRUE, family = Gaussian(), control = boost_control(mstop = 10000,
        nu = 0.5, risk = "inbag"))
plot(mb)
```



13 Рисование

Вообще в R существует по крайней мере три “школы” рисования. Во-первых, это классический пакет `graphics`. Пользоваться я им не советую, во-первых, потому что в результате обычно получаются картинки посредственного качества, а во-вторых из-за его крайне примитивной идеологии: график рассматривается как холст, на котором можно что-то рисовать и подрисовывать, но нельзя ничего исправить.

Современный объектный подход к рисованию реализуют пакеты `lattice` и `ggplot2`. В них функции рисования возвращают некий объект, который можно модифицировать, хранить, передавать и, конечно же, отрисовывать на конкретном устройстве.

Ниже я буду все графики делать в `lattice` из личных предпочтений. Некоторые находят `ggplot2` более эффективным и современным, но я привык к `lattice`.

Собственно, нам понадобится пакет `lattice` (он уже установлен, надо только подключить), а также будет полезным пакет `latticeExtra` (как видно из названия, он

расширяет возможности `lattice`; его нужно поставить с CRAN).

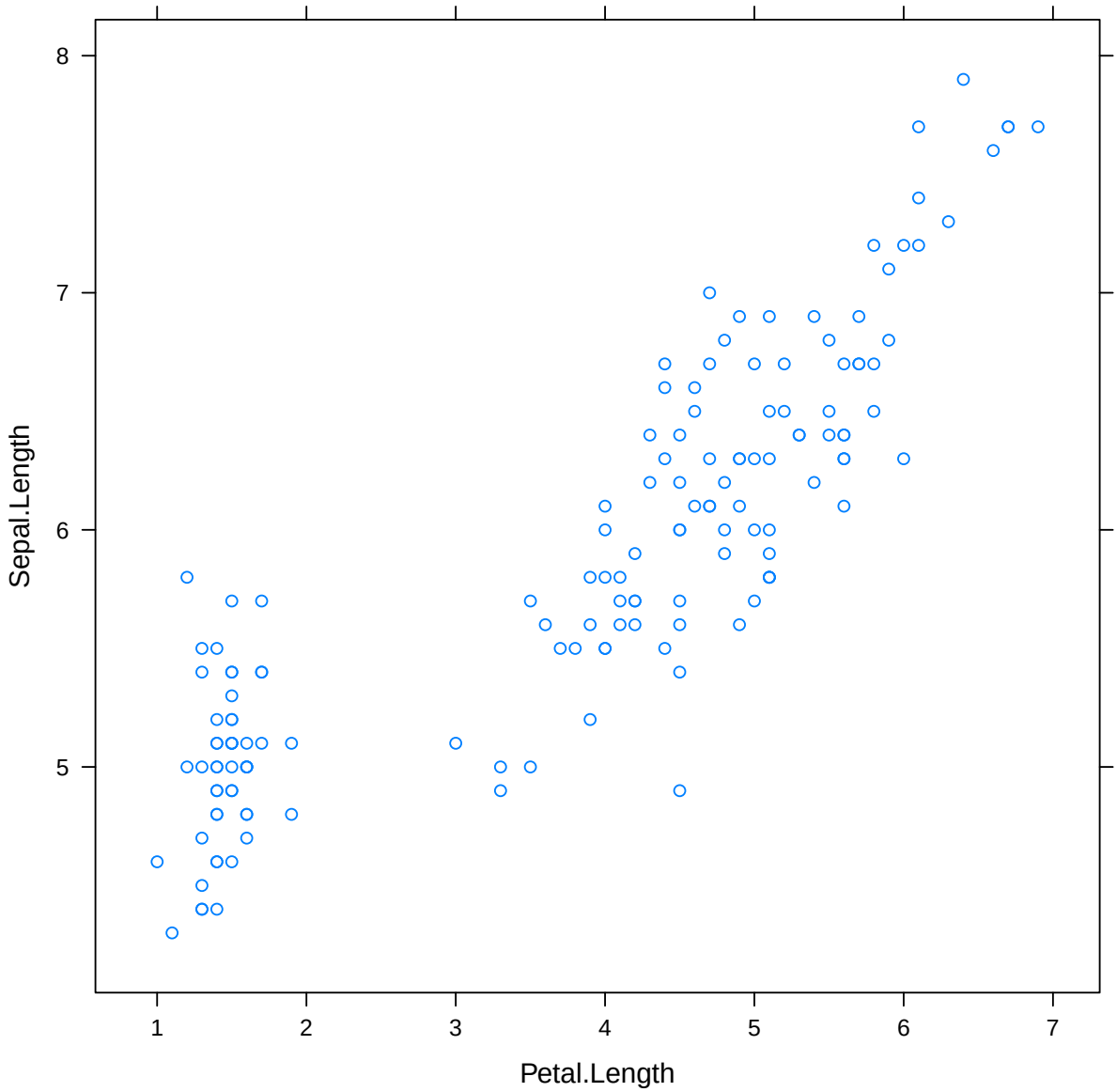
Все функции рисования в `lattice` принимают следующие параметры:

1. `x` — формула; собственно, зависимость, которую мы хотим изобразить
2. `data` — данные, относительно которых будет вычисляться формула (просто датафрейм или именованный список, содержащий использованные в формуле переменные)
3. `groups` — параметр, позволяющий нарисовать несколько наложенных линий на одном графике
4. `panel` — панельная функция, как именно рисовать каждый отдельный график (панель)
5. `...` — параметры, передаваемые в панельную функцию, а также всякие дополнительные параметры, вроде расположения и общего количества панелей.

На примере данных `iris`, посмотрим, какие бывают графики:

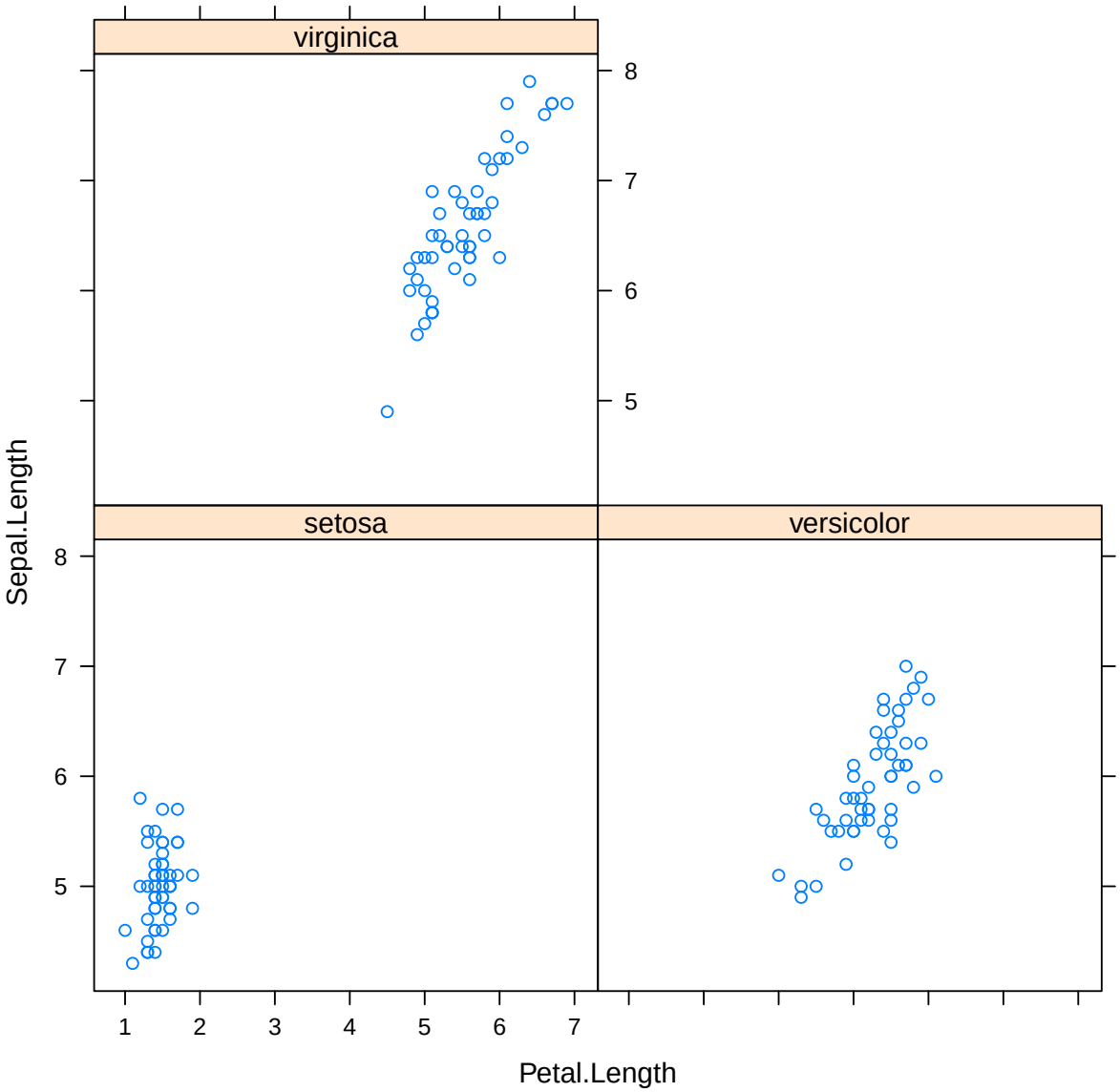
xyplot() — скаттерплот. Каждая строка датафрейма изображается отдельной точкой в координатах двух выбранных столбцов.

```
library(lattice)
xyplot(Sepal.Length ~ Petal.Length, data = iris)
```



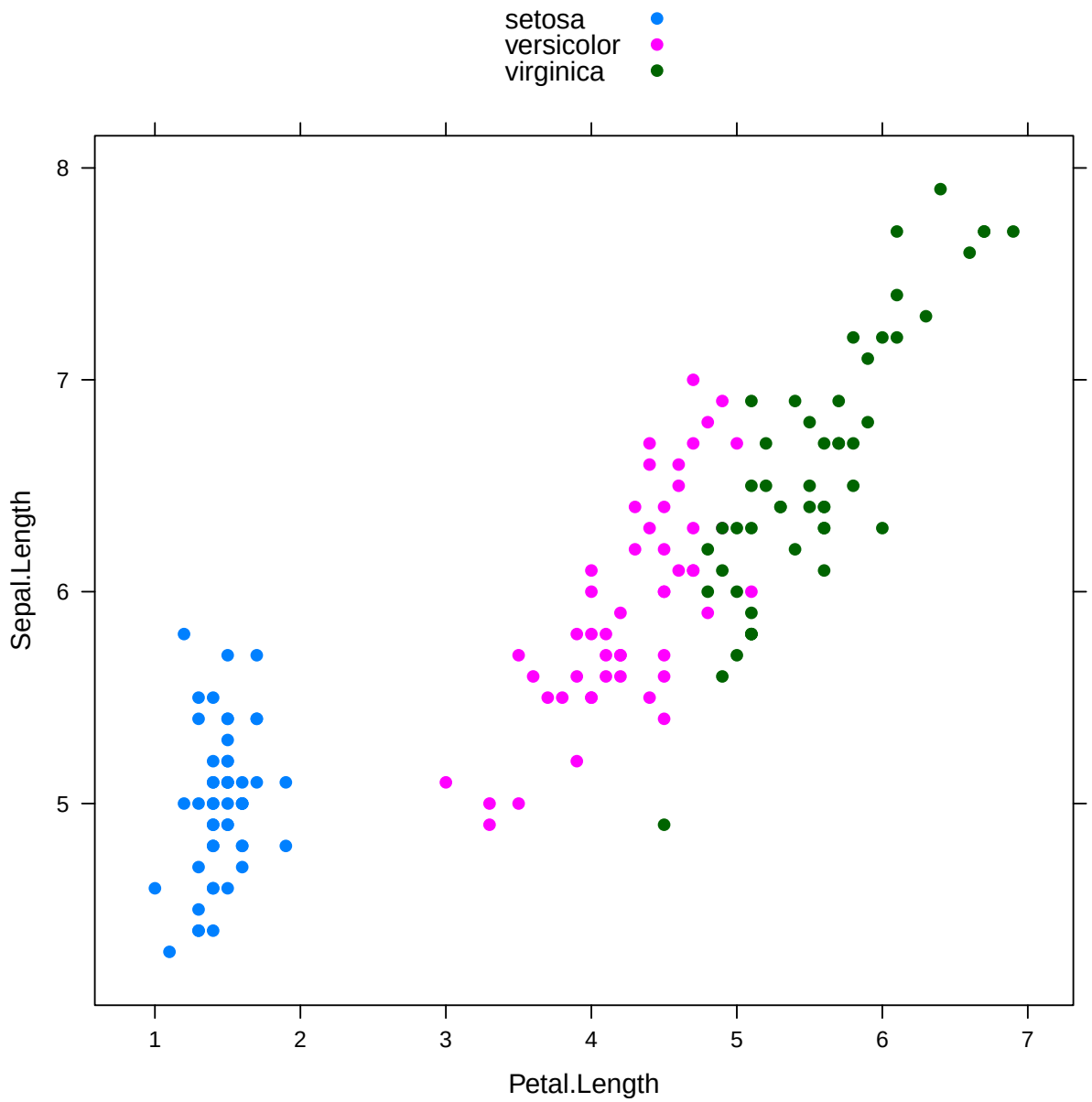
Не очень красиво и малоинформативно. Давайте нарисуем сорта на отдельных графиках:

```
xyplot(Sepal.Length ~ Petal.Length | Species,
    data = iris)
```



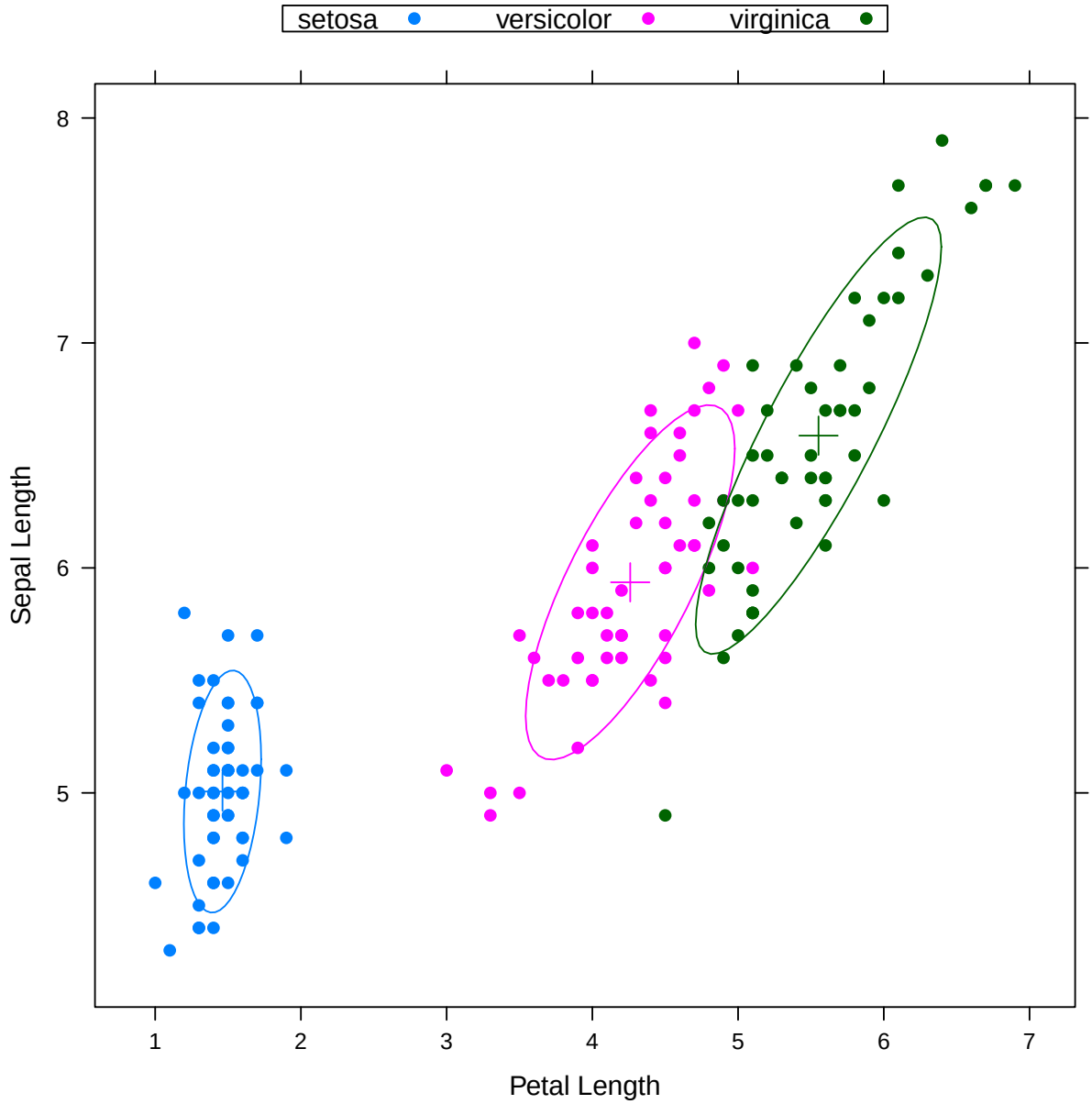
А может лучше все-таки на одном, но разными цветами?.. А еще я хочу сделать точки сплошным и добавить легенду:

```
xyplot(Sepal.Length ~ Petal.Length, groups = Species,
    data = iris, par.settings = simpleTheme(pch = 19),
    auto.key = TRUE)
```




```
library(latticeExtra)
```

```
xyplot(Sepal.Length ~ Petal.Length, groups = Species,
    data = iris, par.settings = simpleTheme(pch = 19),
    auto.key = list(columns = 3, border = TRUE),
    panel = function(...) {
        panel.xyplot(...)
        panel.ellipse(...)
    }, xlab = "Petal Length", ylab = "Sepal Length")
```

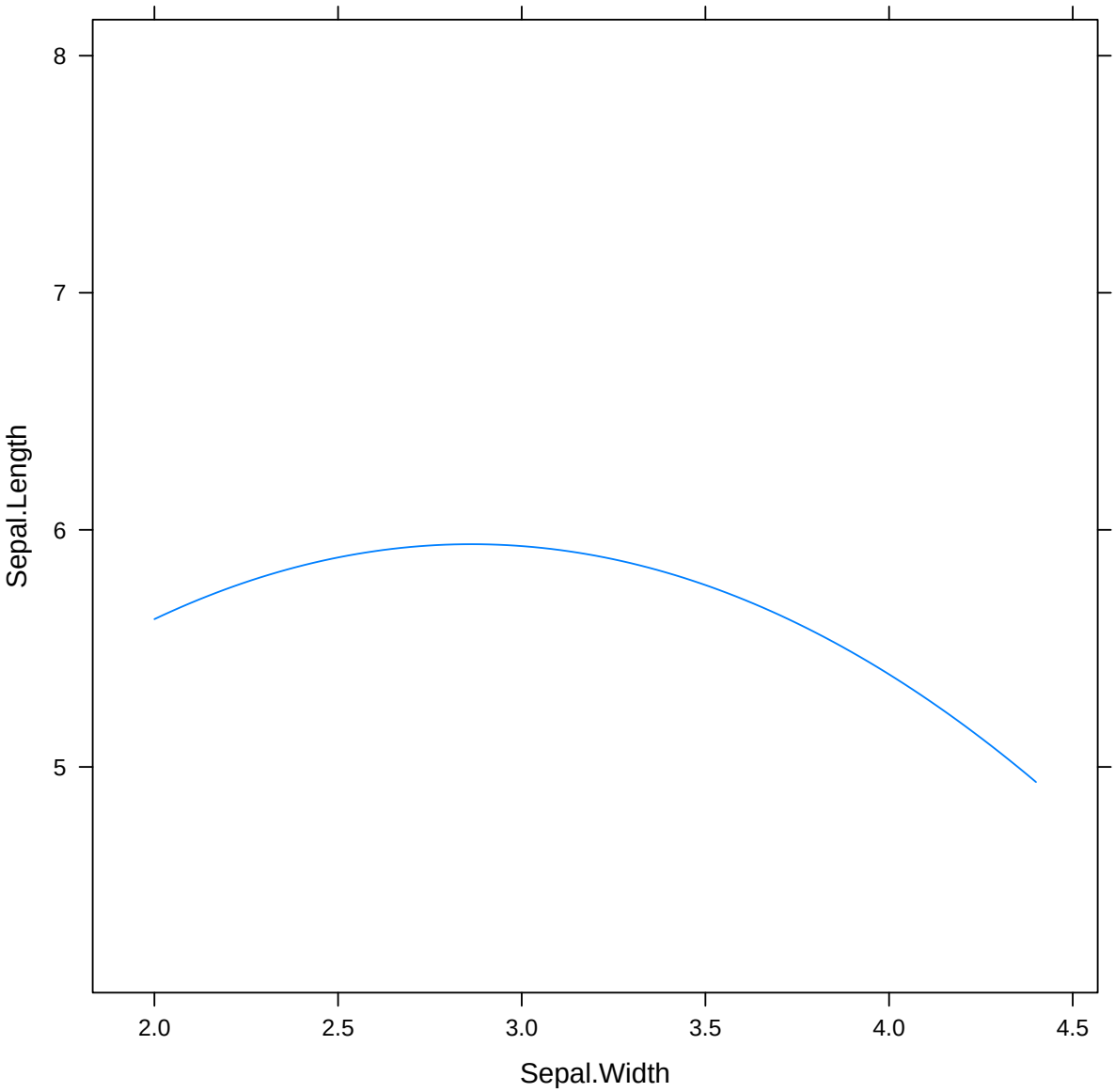


Напоследок приведу пример написания своей панельной функции для `xyplot()`:

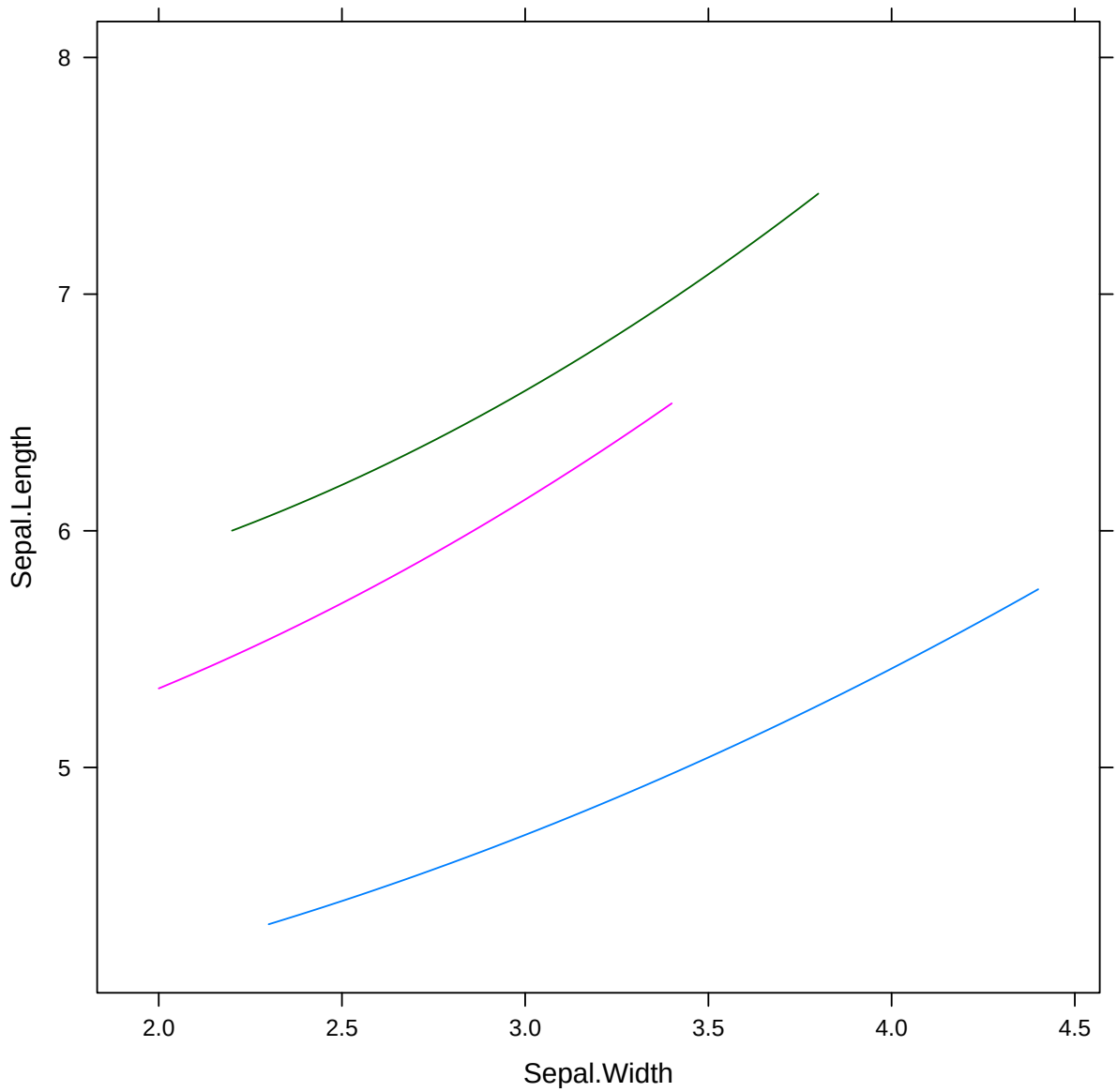
```
read_chunk("panel.lmpolyline.R")
```

```
panel.lmpolyline <- function(x, y, groups = NULL,
    degree = 1, col.line = par.line$col,
    lty = par.line$lty, lwd = par.line$lwd,
    alpha = par.line$alpha, ..., identifier = "lmpolyline") {
    x <- as.numeric(x)
    y <- as.numeric(y)
    if (!is.null(groups)) {
        par.line <- trellis.par.get("superpose.line")
        panel.superpose(x = x, y = y, groups = groups,
            degree = degree, col.line = col.line,
            lty = lty, lwd = lwd, alpha = alpha,
```

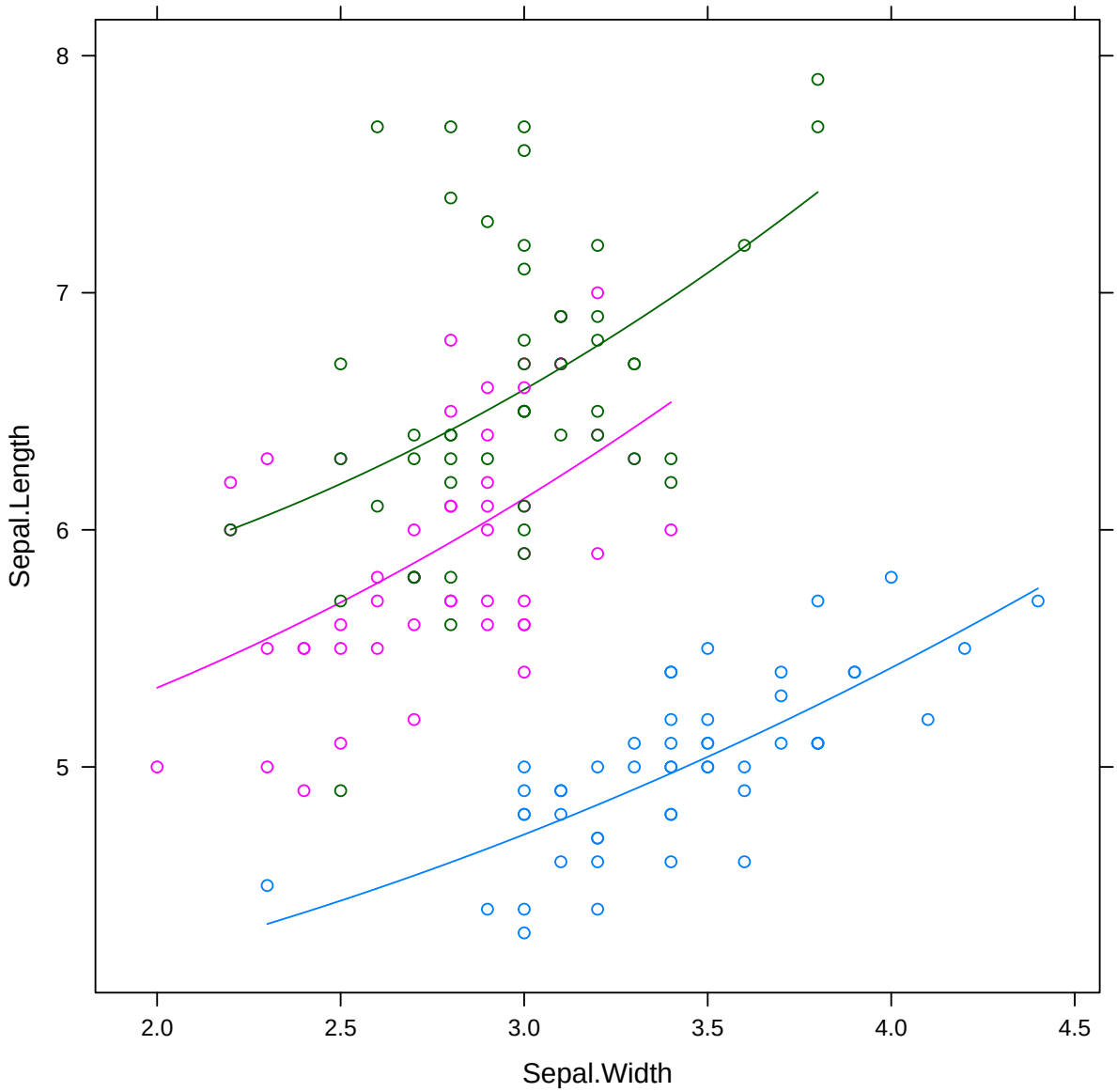
```
            panel.groups = sys.function(),
            ...)
    } else {
        if (length(x) > degree) {
            l <- lm(y ~ poly(x, degree = degree))
            par.line <- trellis.par.get("plot.line")
            panel.curve(predict(l, list(x = x)),
                from = min(x), to = max(x),
                col.line = col.line, lty = lty,
                lwd = lwd, alpha = alpha,
                ..., identifier = identifier)
        }
    }
}
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
    panel = panel.lmpolyline, degree = 2)
```




```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species,
    data = iris, panel = panel.lmpolyline,
    degree = 2)
```



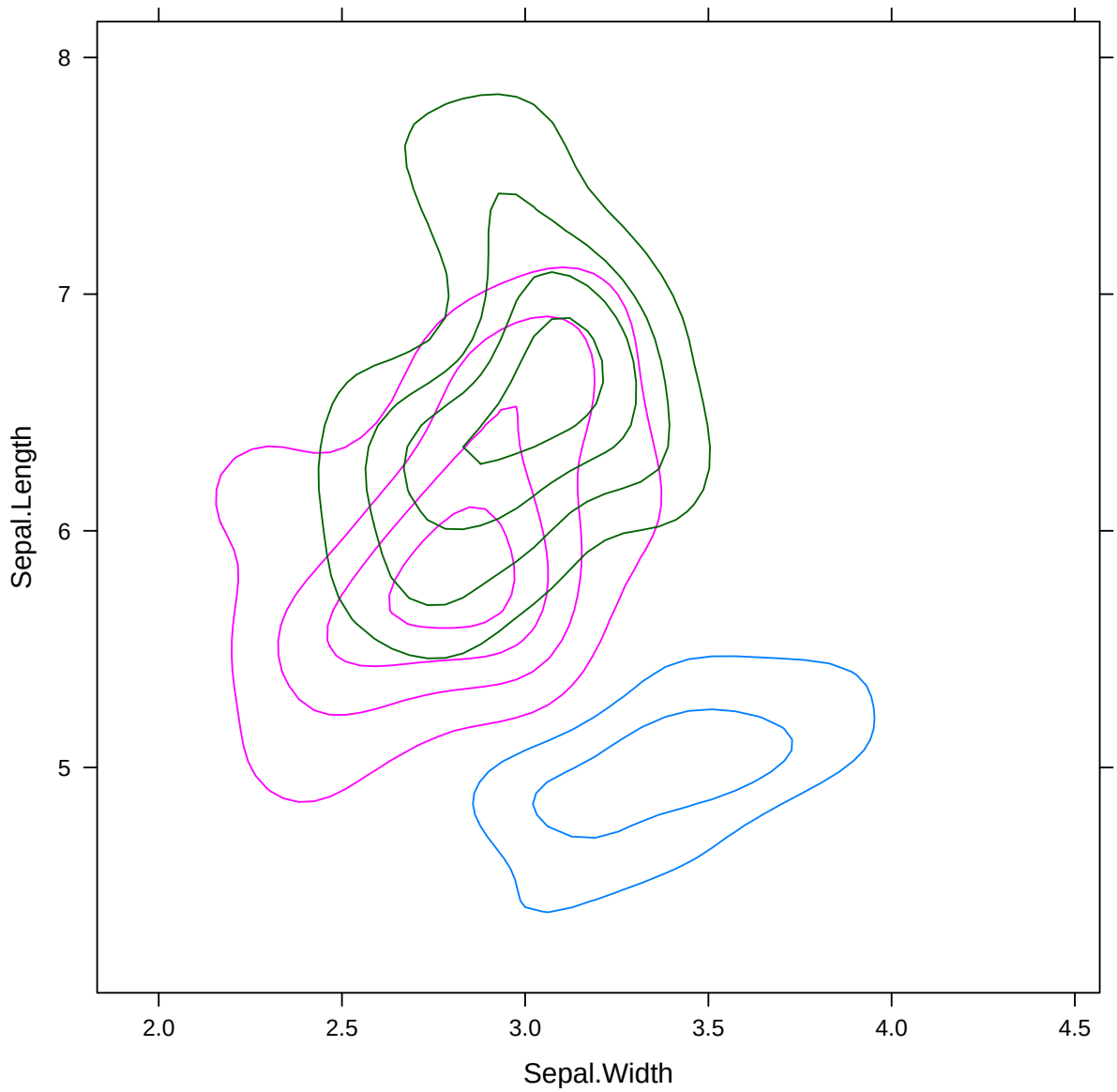

```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species,
    data = iris) + layer_(panel.lmpolyline(...,
    degree = 2))
```



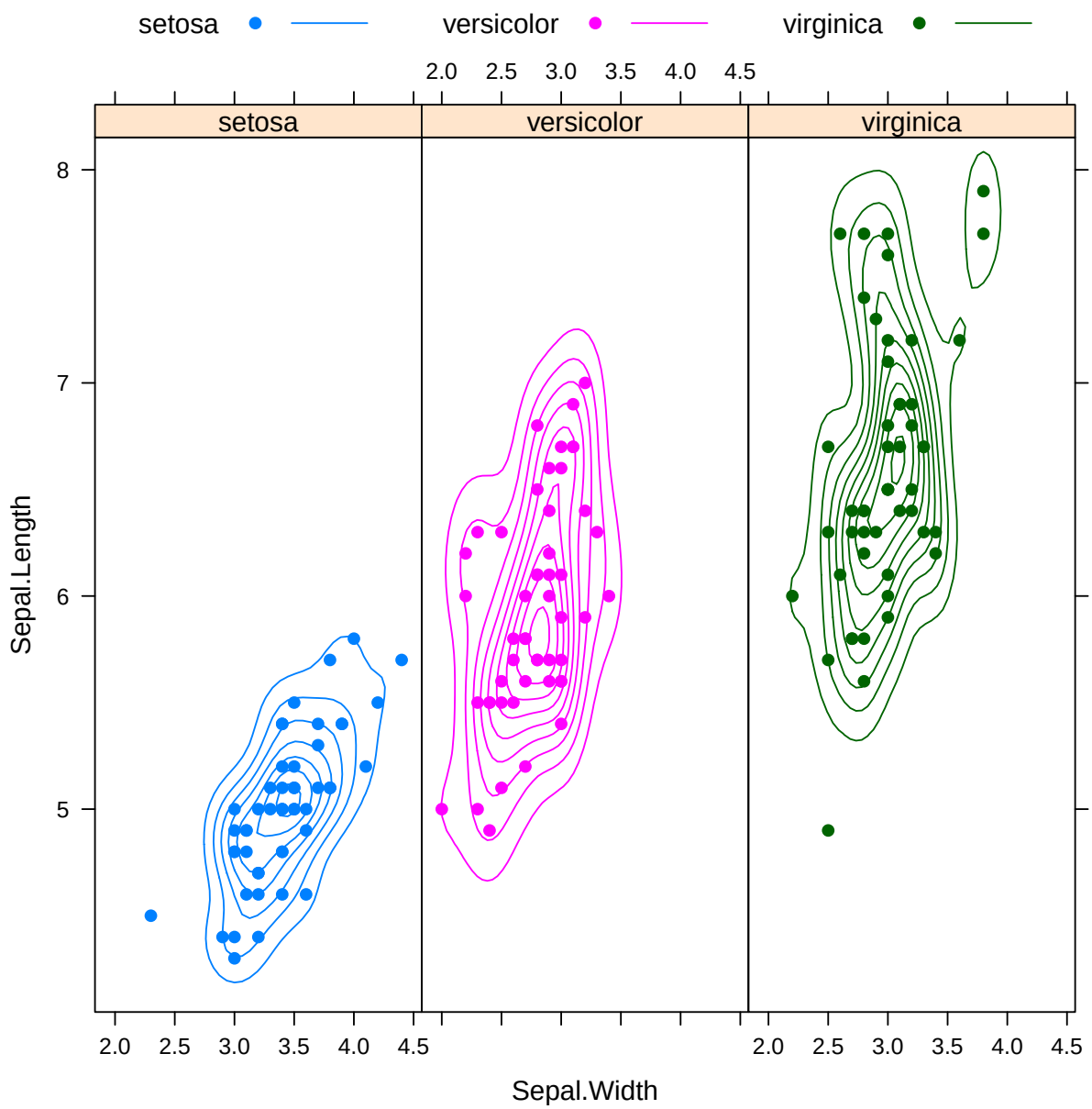
13.1 Оценка двумерной плотности (kde2)

```
panel.kde2d <- function(x, y, groups = NULL,
    subscripts, n = 100, cuts = 5, col.line = par.line$col,
    lty = par.line$lty, lwd = par.line$lwd,
    alpha = par.line$alpha, ..., identifier = "kde2d",
    col) {
    require("MASS")
    x <- as.numeric(x)
    y <- as.numeric(y)
    if (!is.null(groups)) {
        par.line <- trellis.par.get("superpose.line")
        panel.superpose(x = x, y = y, groups = groups,
            subscripts = subscripts, n = n,
```

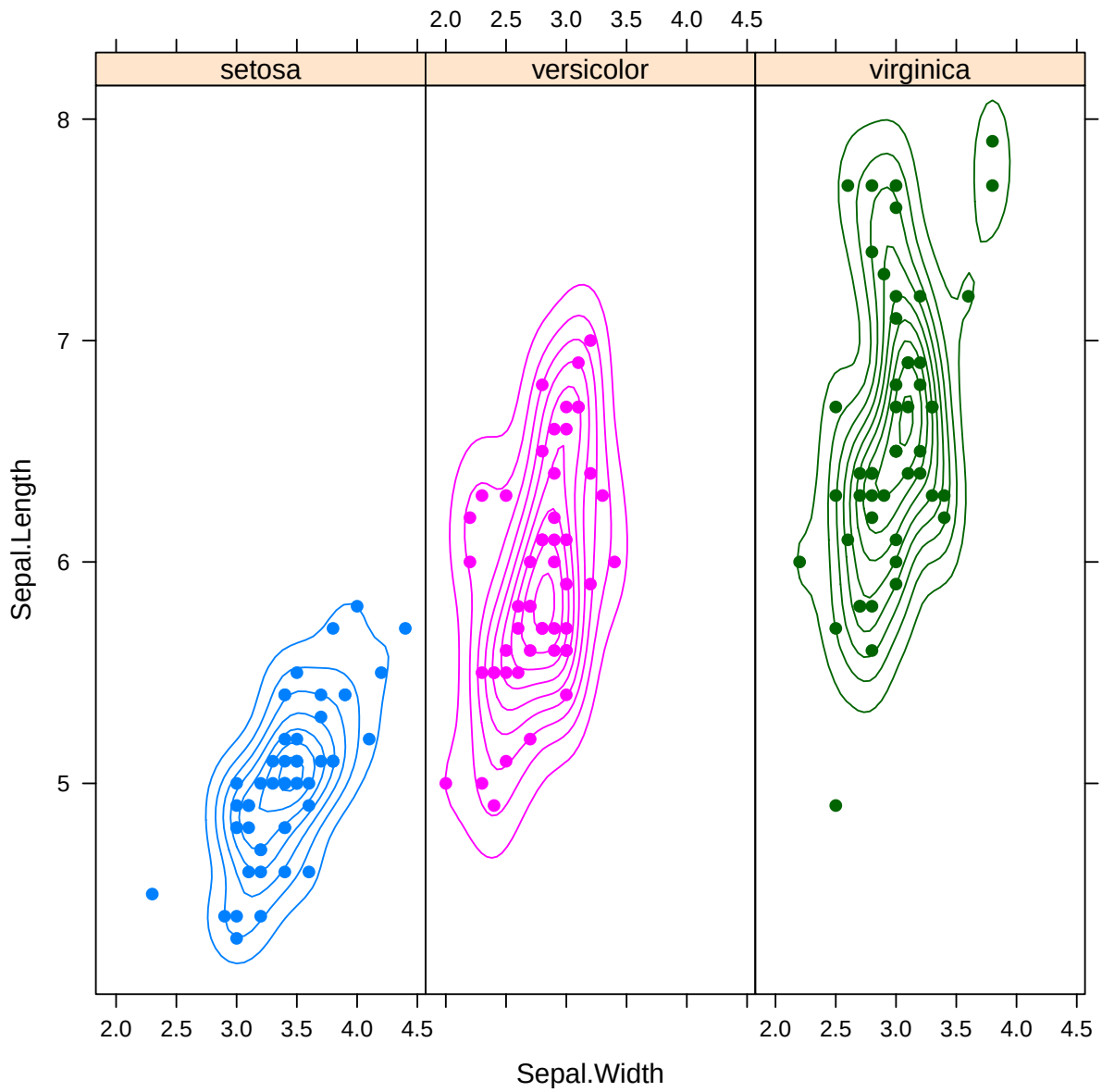
```
            cuts = cuts, panel.groups = sys.function(),
            col.line = col.line, lty = lty,
            lwd = lwd, alpha = alpha, ...)
    } else {
        drange <- function(x) {
            r <- range(x)
            d <- diff(r)
            r + c(-d, d)
        }
        kde <- kde2d(x, y, n = n, lims = c(drange(x),
            drange(y)))
        data <- expand.grid(x = kde$x, y = kde$y)
        data$z <- as.vector(kde$z)
        plot.line <- trellis.par.get("plot.line")
        panel.contourplot(data$x, data$y,
            data$z, at = pretty(data$z, n = cuts),
            subscripts = seq_along(data$x),
            contour = TRUE, region = FALSE,
            col = col.line, lty = lty, lwd = lwd,
            alpha = alpha, ..., identifier = identifier)
    }
}
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
    groups = Species, panel = panel.kde2d)
```



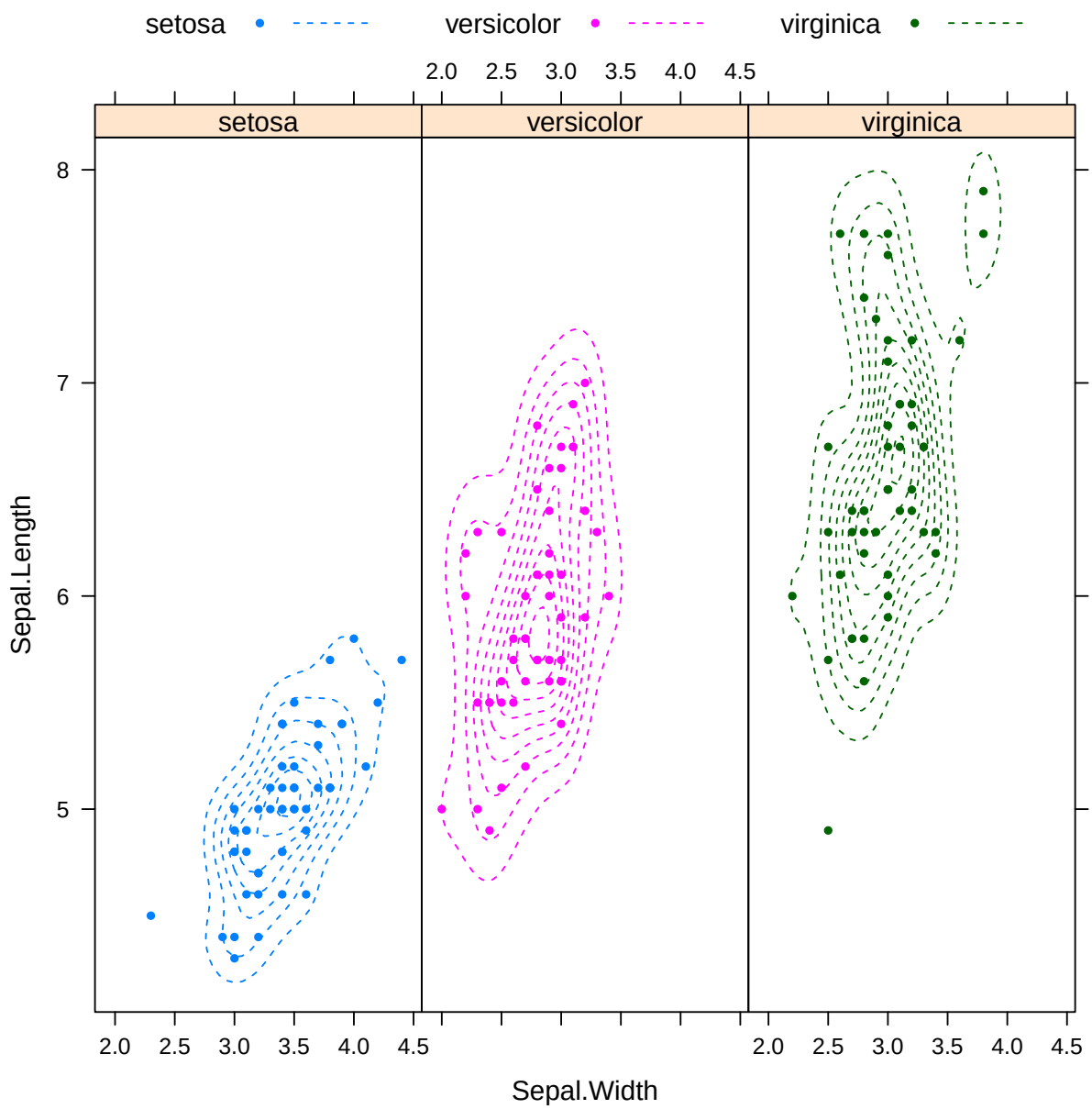

```
xyplot(Sepal.Length ~ Sepal.Width | Species,
    data = iris, groups = Species, par.settings = simpleTheme(pch = 19),
    auto.key = list(columns = 3, lines = TRUE),
    layout = c(3, 1)) + layer_(panel.kde2d(...,
    cuts = 10))
```




```
xyplot(Sepal.Length ~ Sepal.Width | Species,
    data = iris, groups = Species, par.settings = simpleTheme(pch = 19),
    layout = c(3, 1)) + layer_(panel.kde2d(...,
    cuts = 10))
```

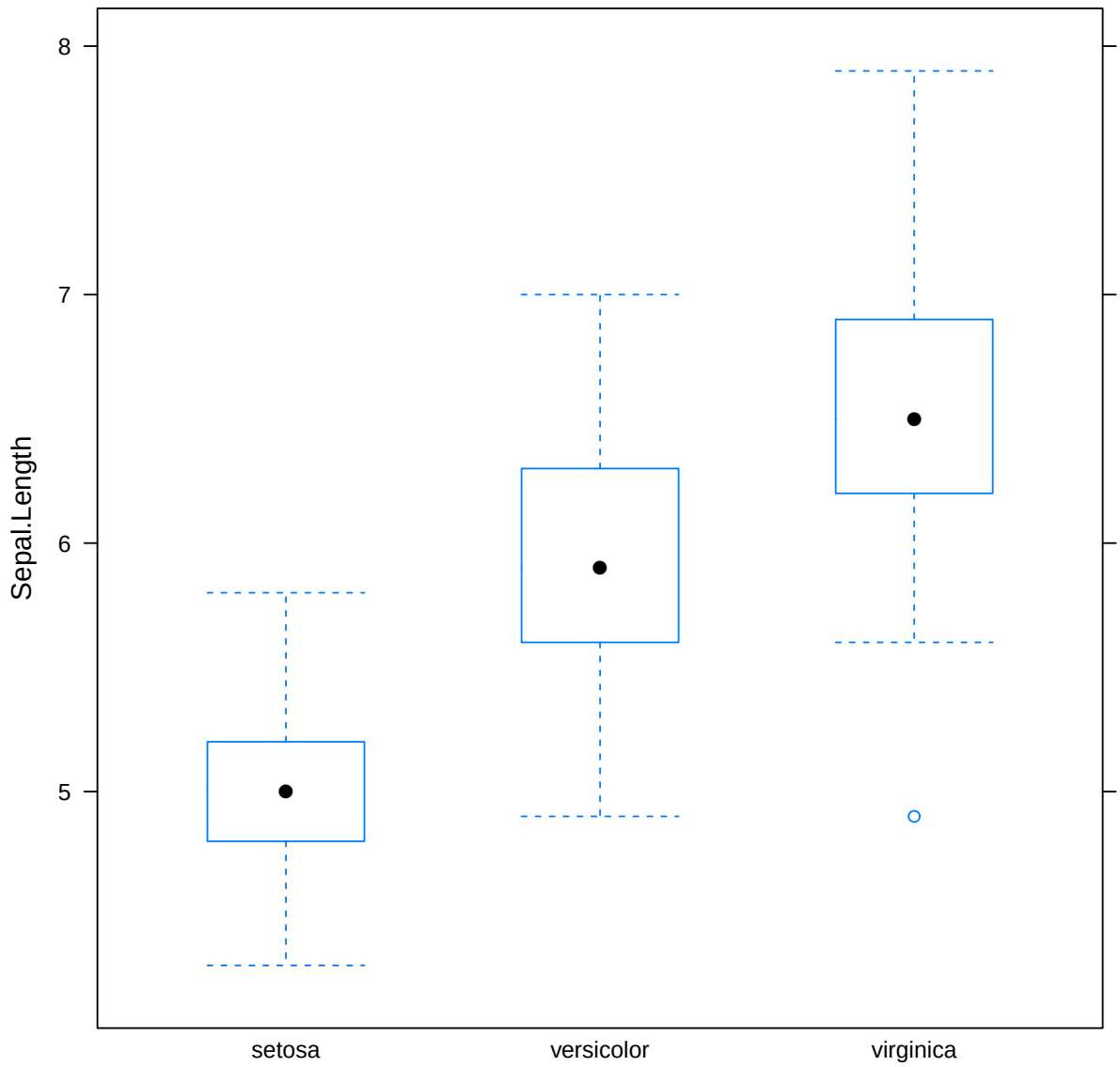



```
xyplot(Sepal.Length ~ Sepal.Width | Species,
    data = iris, groups = Species, par.settings = simpleTheme(pch = 19,
        cex = 0.5, lwd = 1, lty = "dashed"),
    auto.key = list(columns = 3, lines = TRUE),
    layout = c(3, 1)) + layer_(panel.kde2d(...,
    cuts = 10))
```

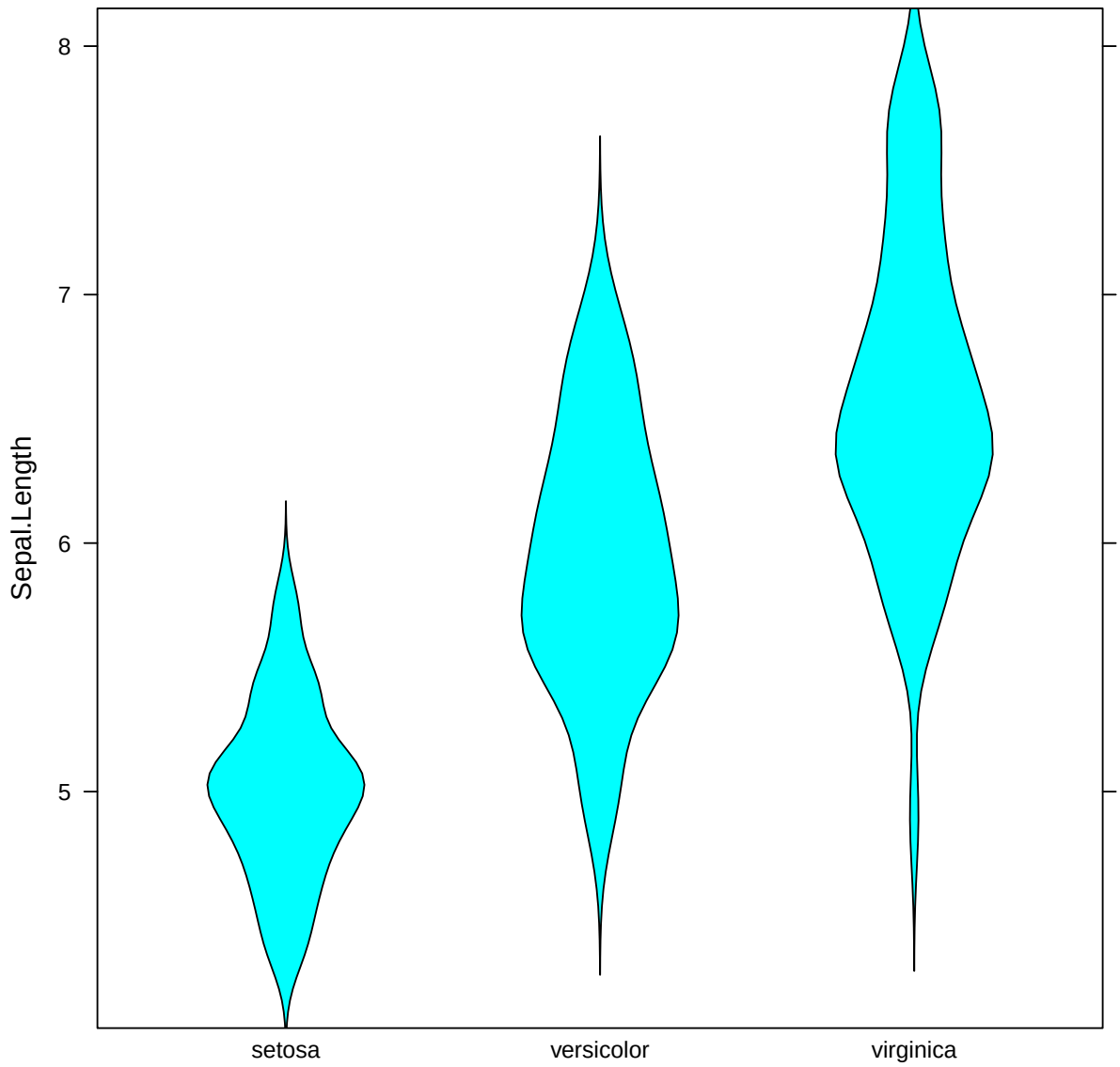


13.2 Boxplots, stripplots and dotplots (draft)

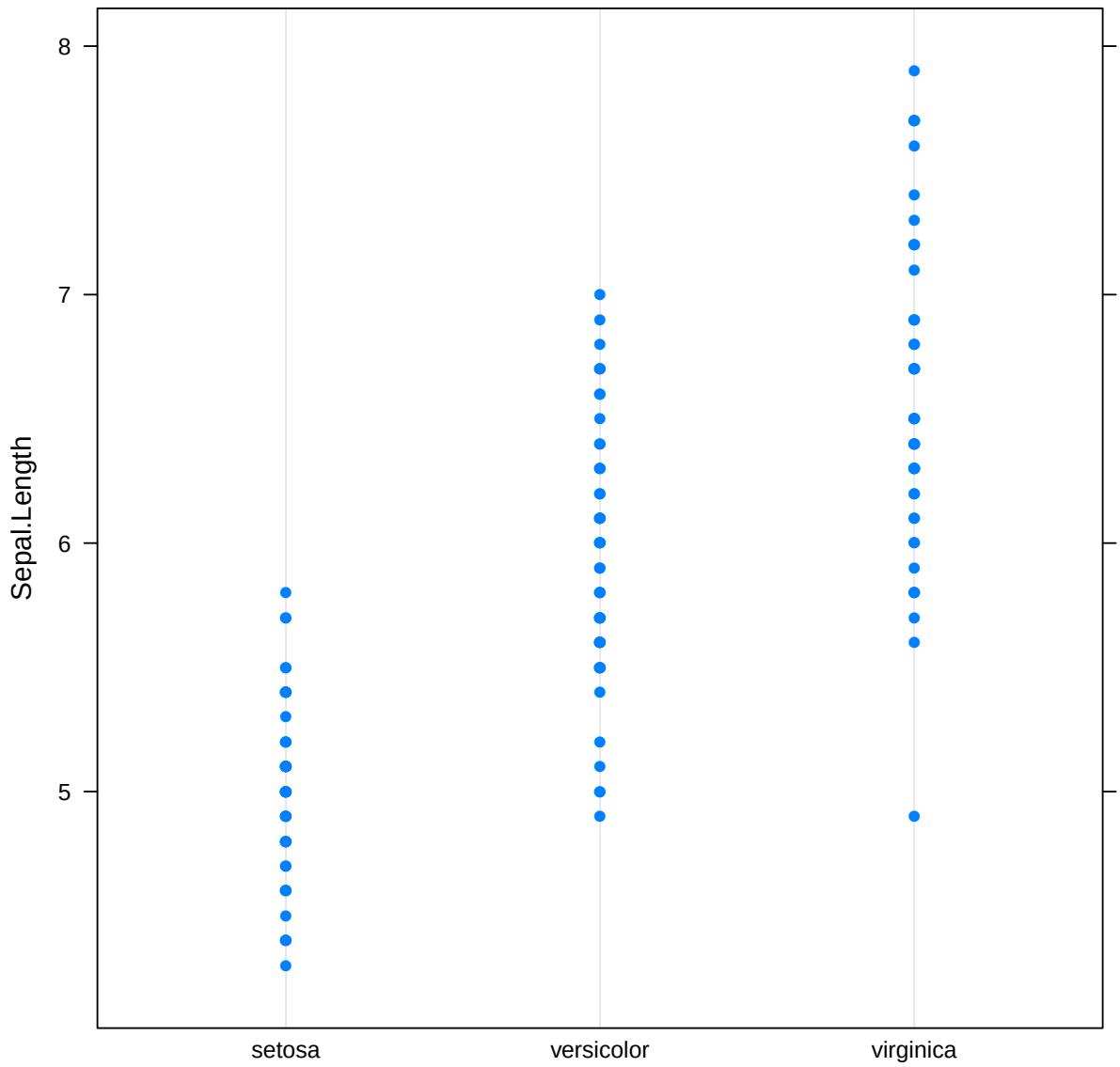
```
bwplot(Sepal.Length ~ Species, data = iris)
```



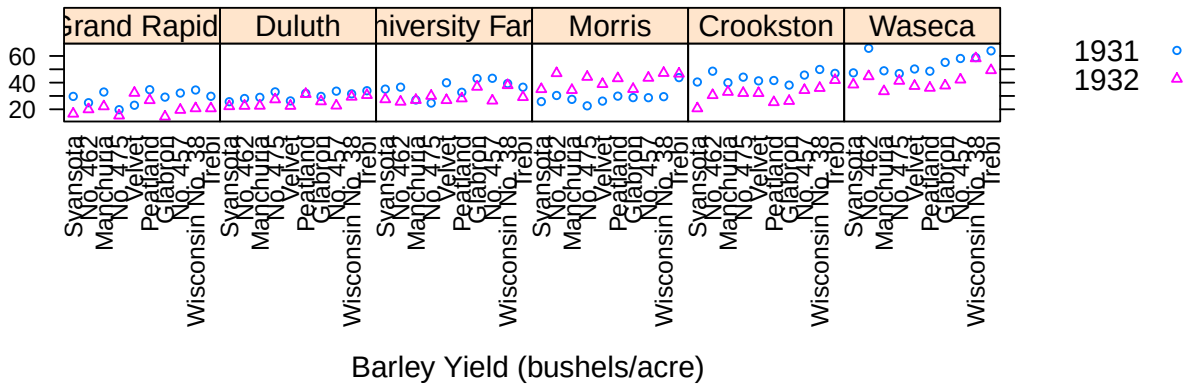

```
bwplot(Sepal.Length ~ Species, data = iris,
    panel = panel.violin)
```



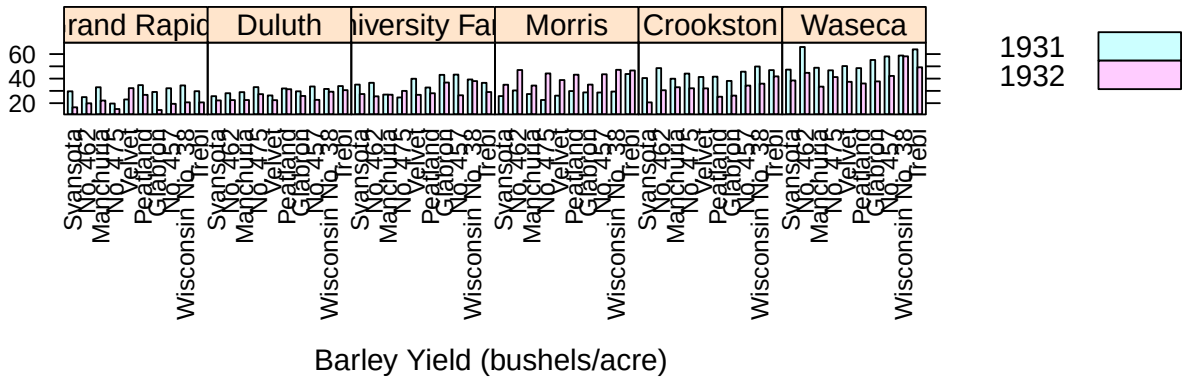

```
bwplot(Sepal.Length ~ Species, data = iris,
    panel = panel.dotplot)
```



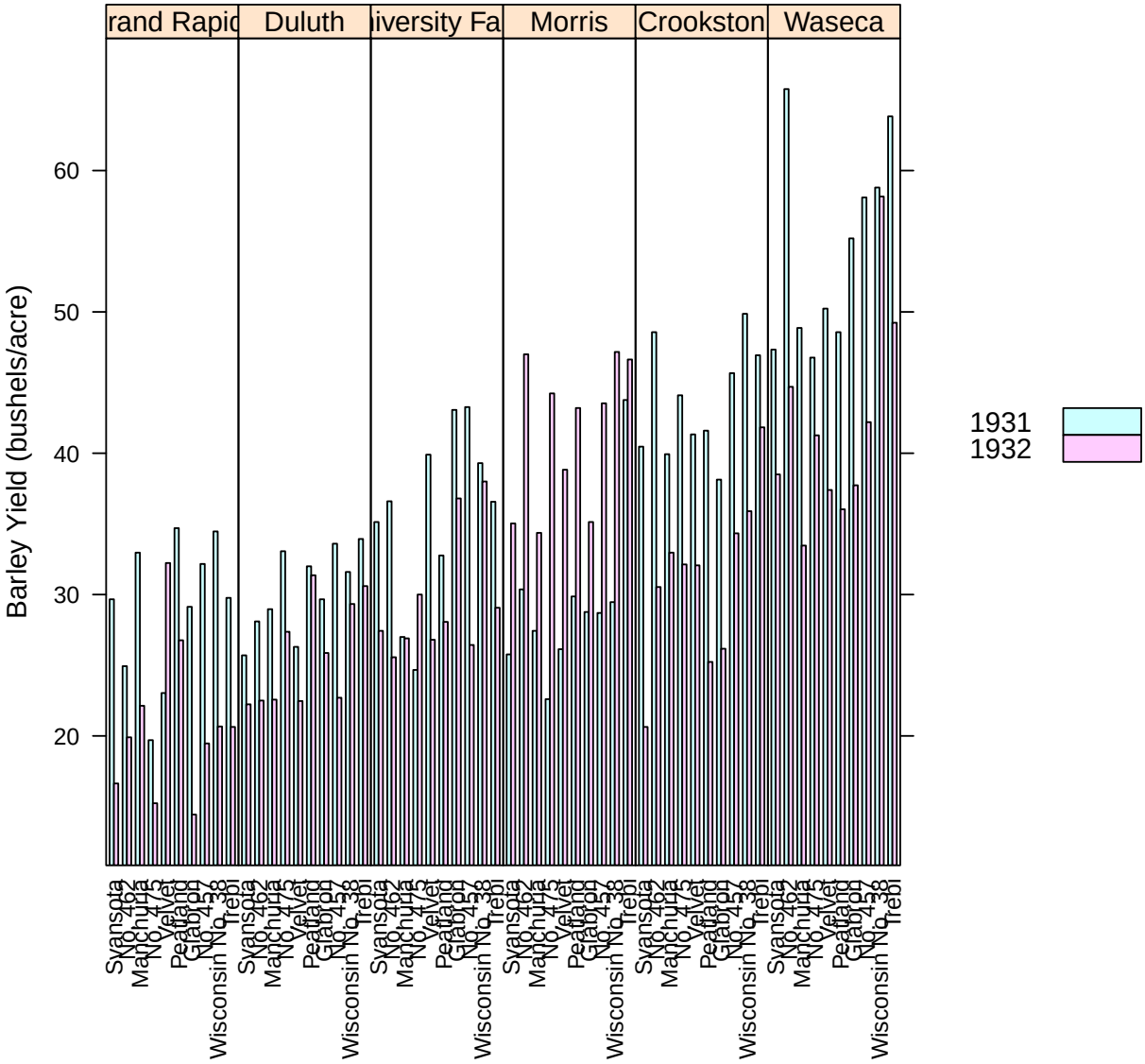

```
barley$year <- factor(barley$year, levels = c(1931,
    1932))
stripplot(yield ~ variety | site, data = barley,
    groups = year, par.settings = simpleTheme(pch = c(1,
        2), cex = 0.5), auto.key = list(space = "right"),
    aspect = 0.5, layout = c(6, 1), xlab = "Barley Yield (bushels/acre) ",
    ylab = NULL, scales = list(x = list(rot = 90)))
```



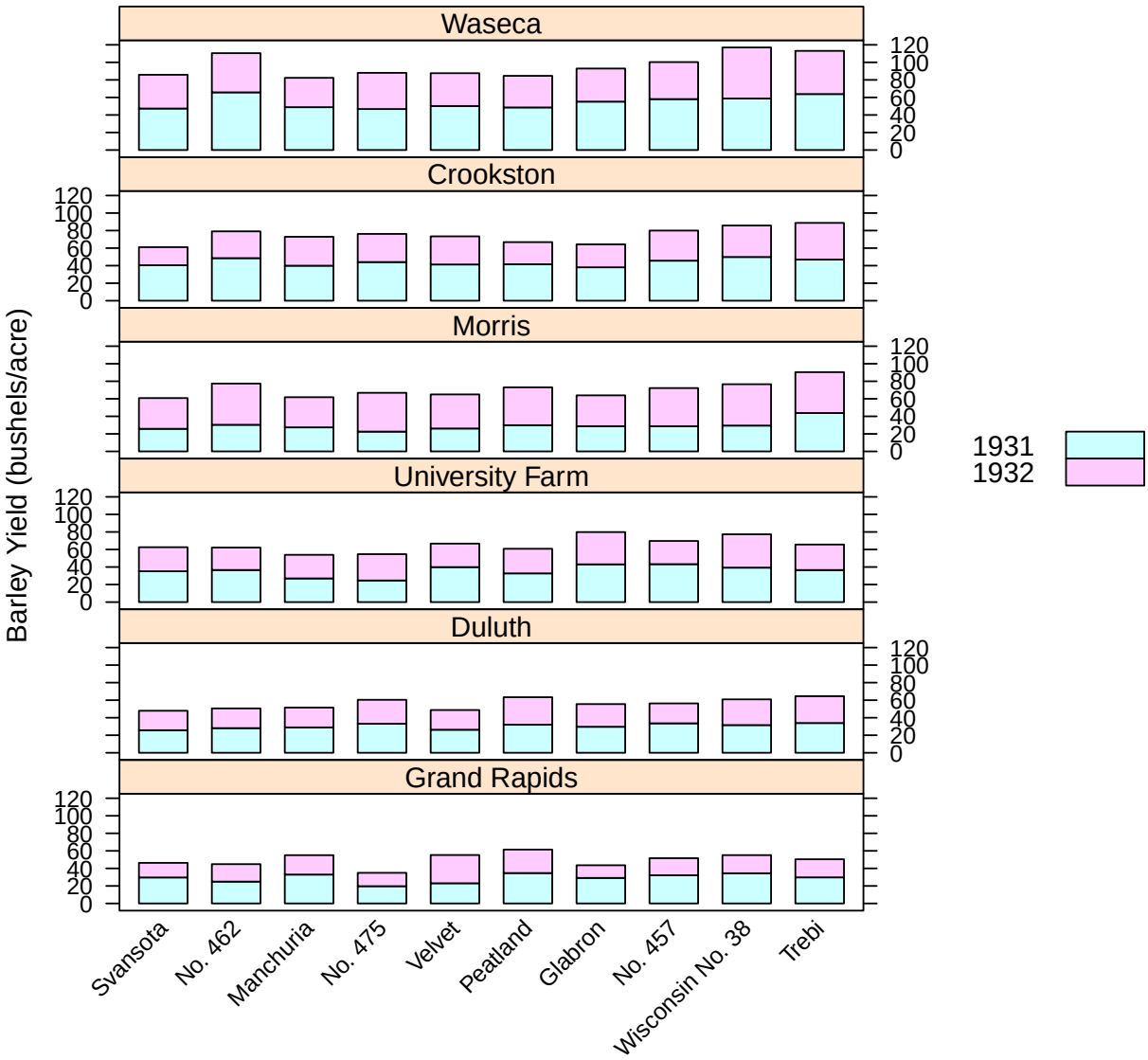

```
barchart(yield ~ variety | site, data = barley,
    groups = year, par.settings = simpleTheme(pch = c(1,
        2), cex = 0.5), auto.key = list(space = "right"),
    aspect = 0.5, layout = c(6, 1), xlab = "Barley Yield (bushels/acre) ",
    ylab = NULL, scales = list(x = list(rot = 90)))
```



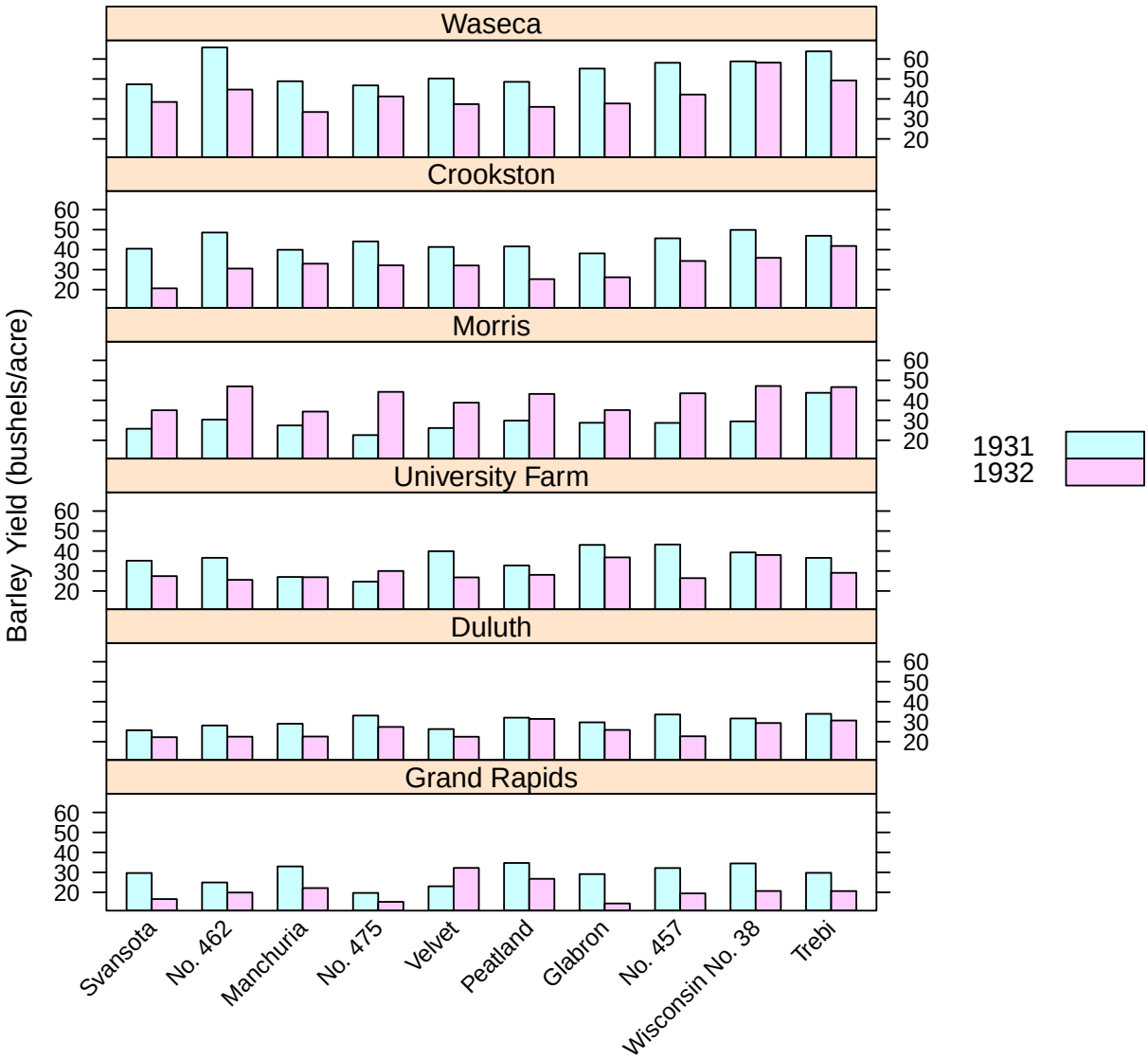

```
barchart(yield ~ variety | site, data = barley,
    groups = year, layout = c(6, 1), stack = FALSE,
    auto.key = list(space = "right"), ylab = "Barley Yield (bushels/acre)",
    scales = list(x = list(rot = 90)))
```



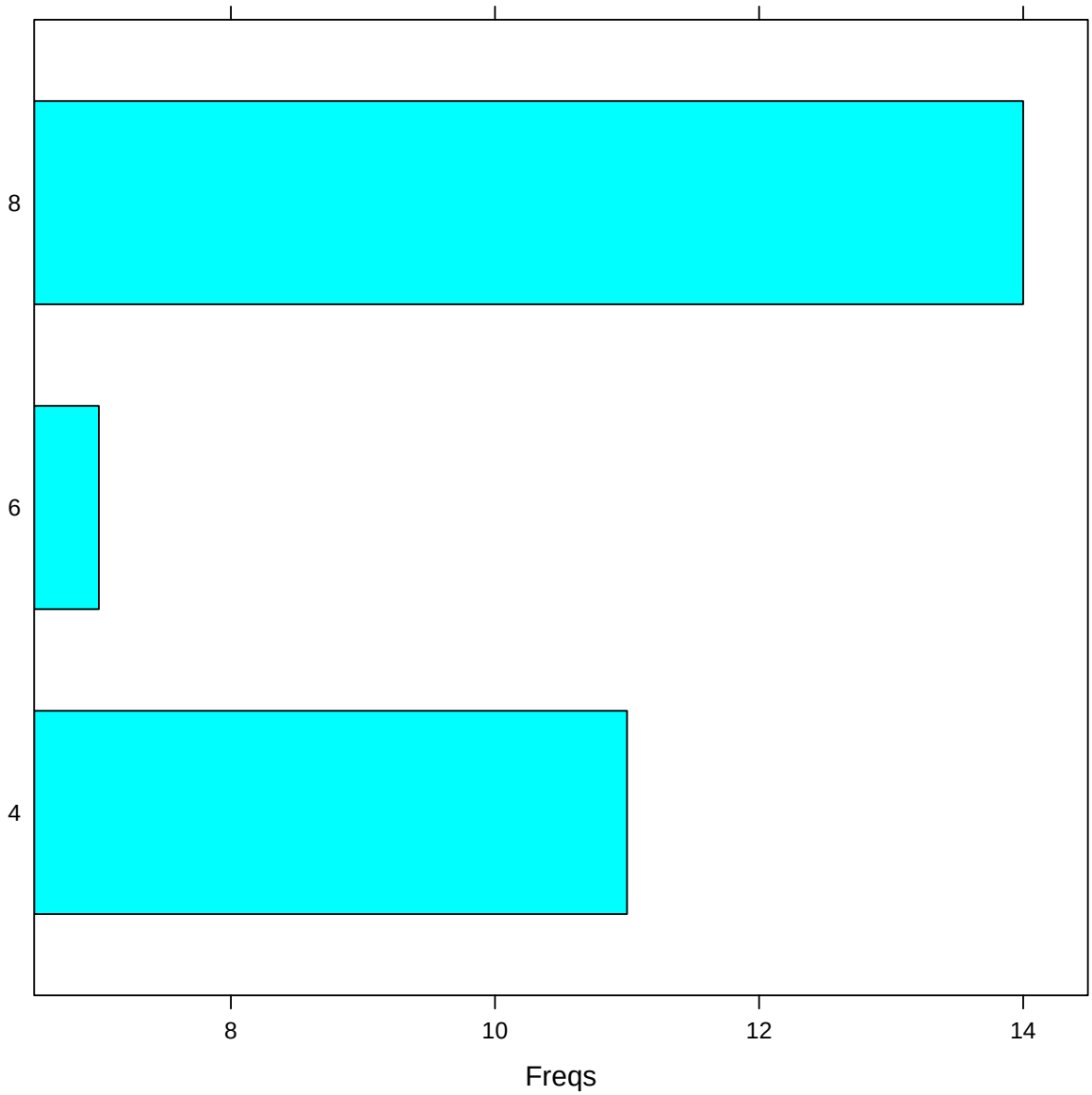

```
barchart(yield ~ variety | site, data = barley,
    groups = year, layout = c(1, 6), stack = TRUE,
    auto.key = list(space = "right"), ylab = "Barley Yield (bushels/acre)",
    scales = list(x = list(rot = 45)))
```



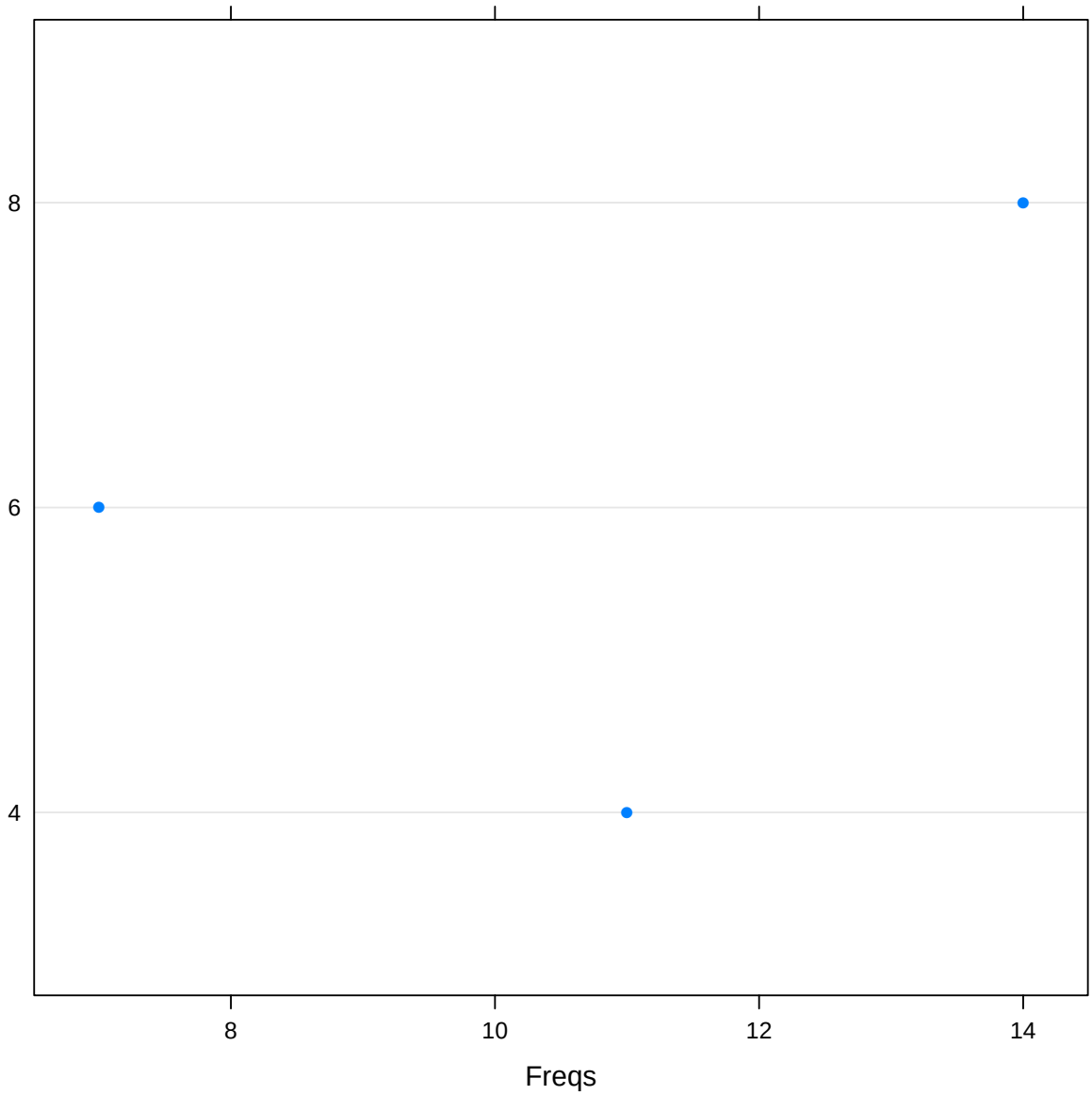

```
barchart(yield ~ variety | site, data = barley,
    groups = year, layout = c(1, 6), stack = FALSE,
    auto.key = list(space = "right"), ylab = "Barley Yield (bushels/acre)",
    scales = list(x = list(rot = 45)))
```



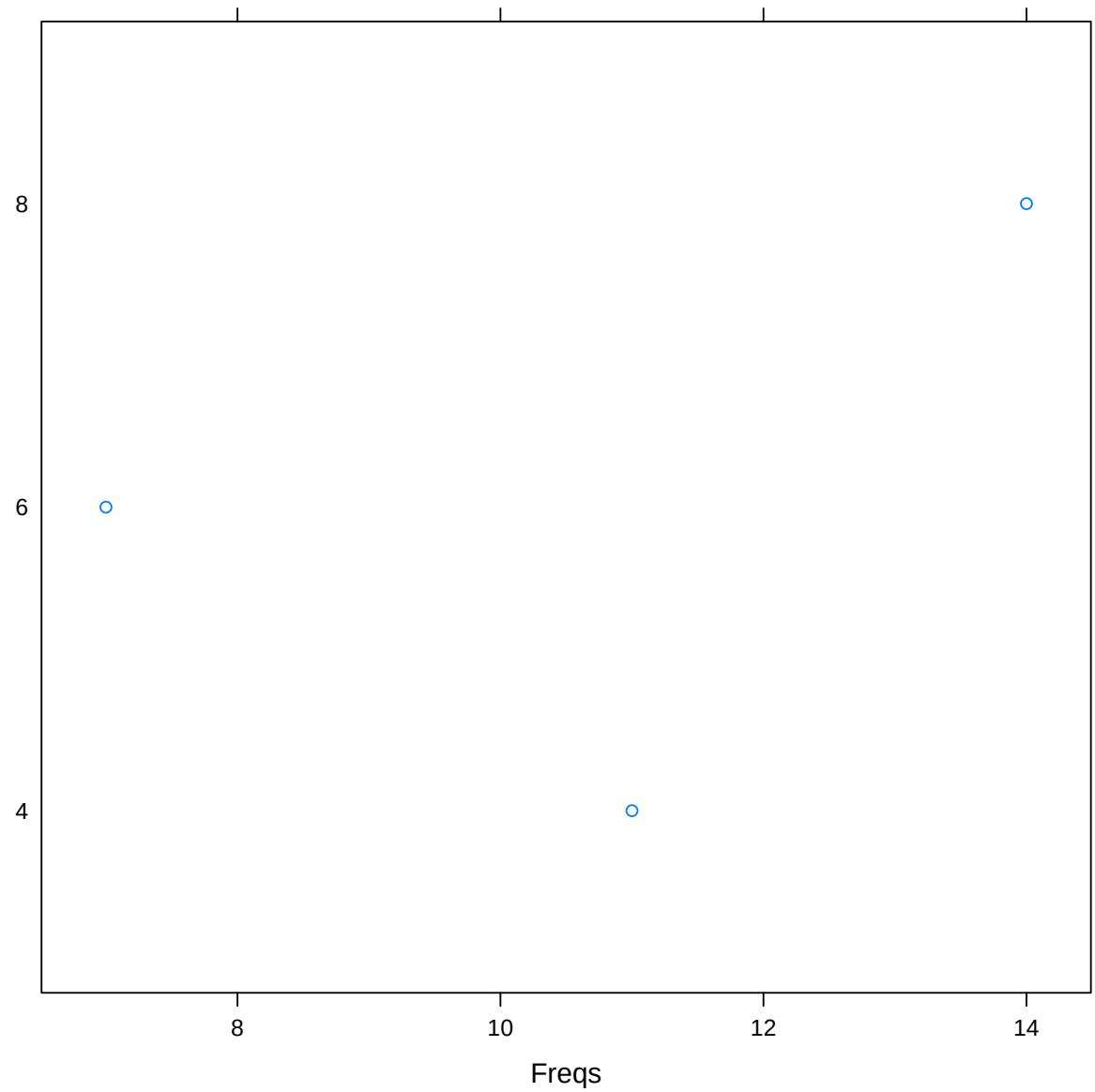

```
barchart(~table(cyl), data = mtcars, xlab = "Freqs")
```



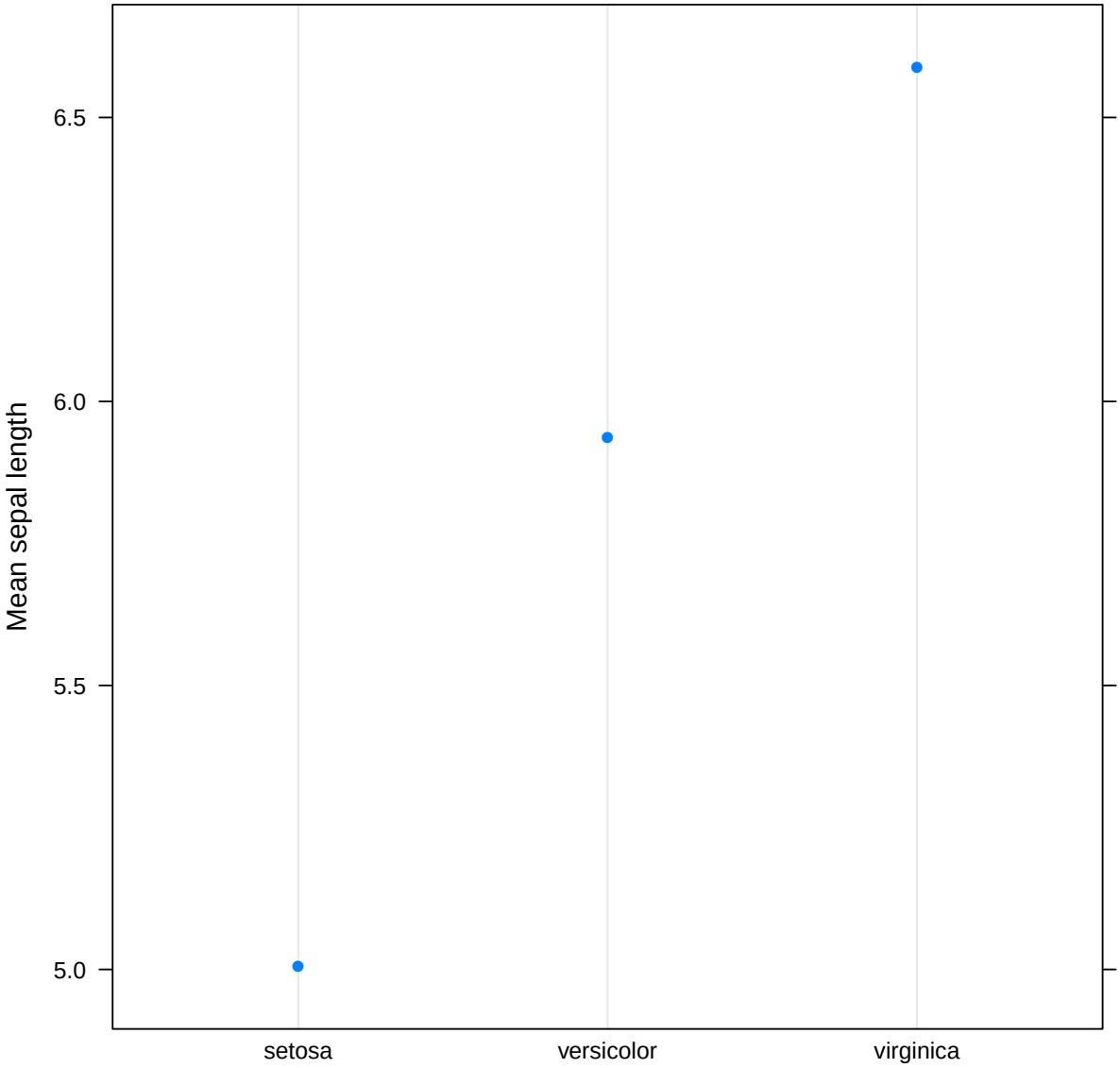

```
dotplot(~table(cyl), data = mtcars, xlab = "Freqs")
```



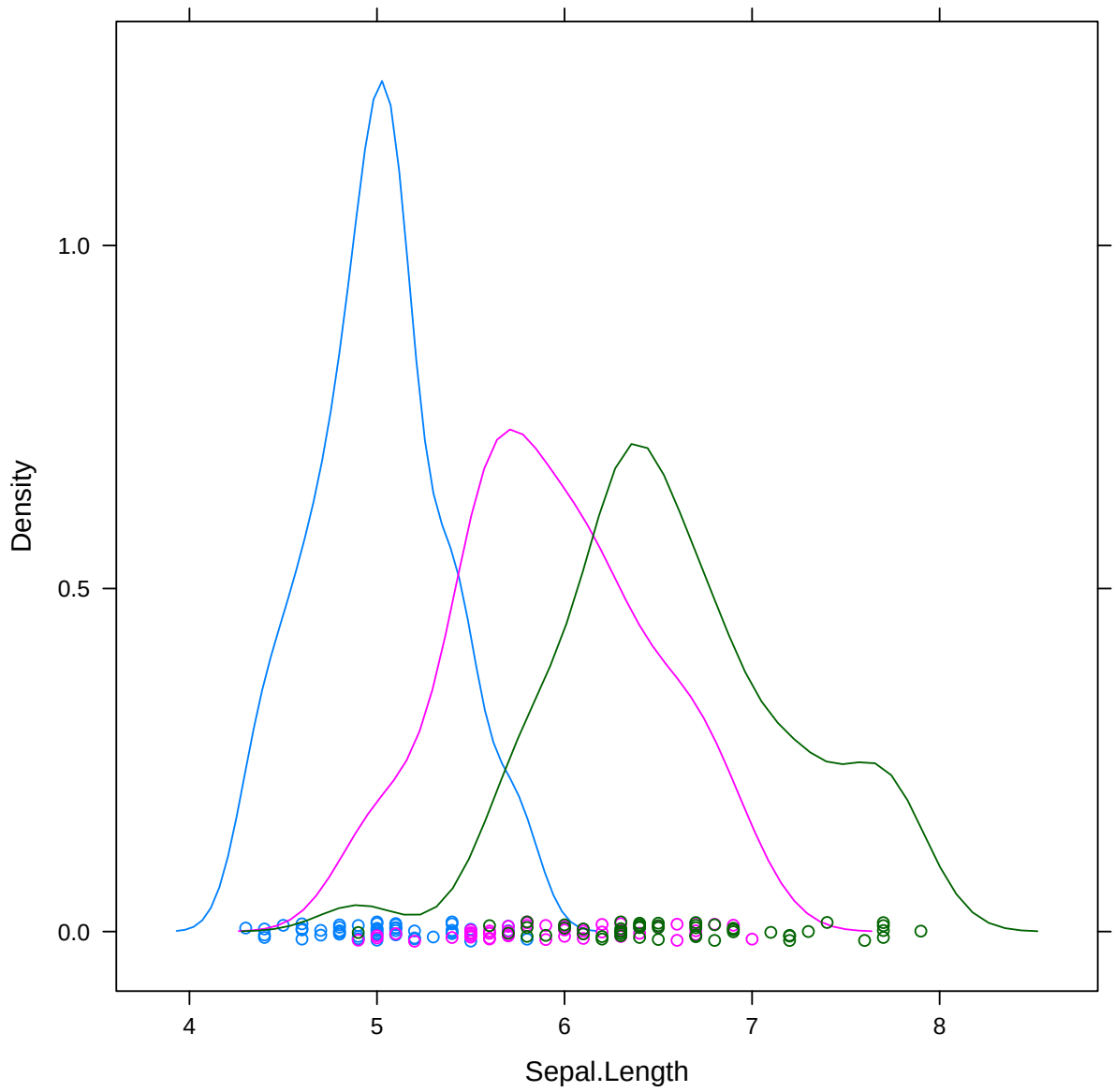

```
stripplot(~table(cyl), data = mtcars, xlab = "Freqs")
```



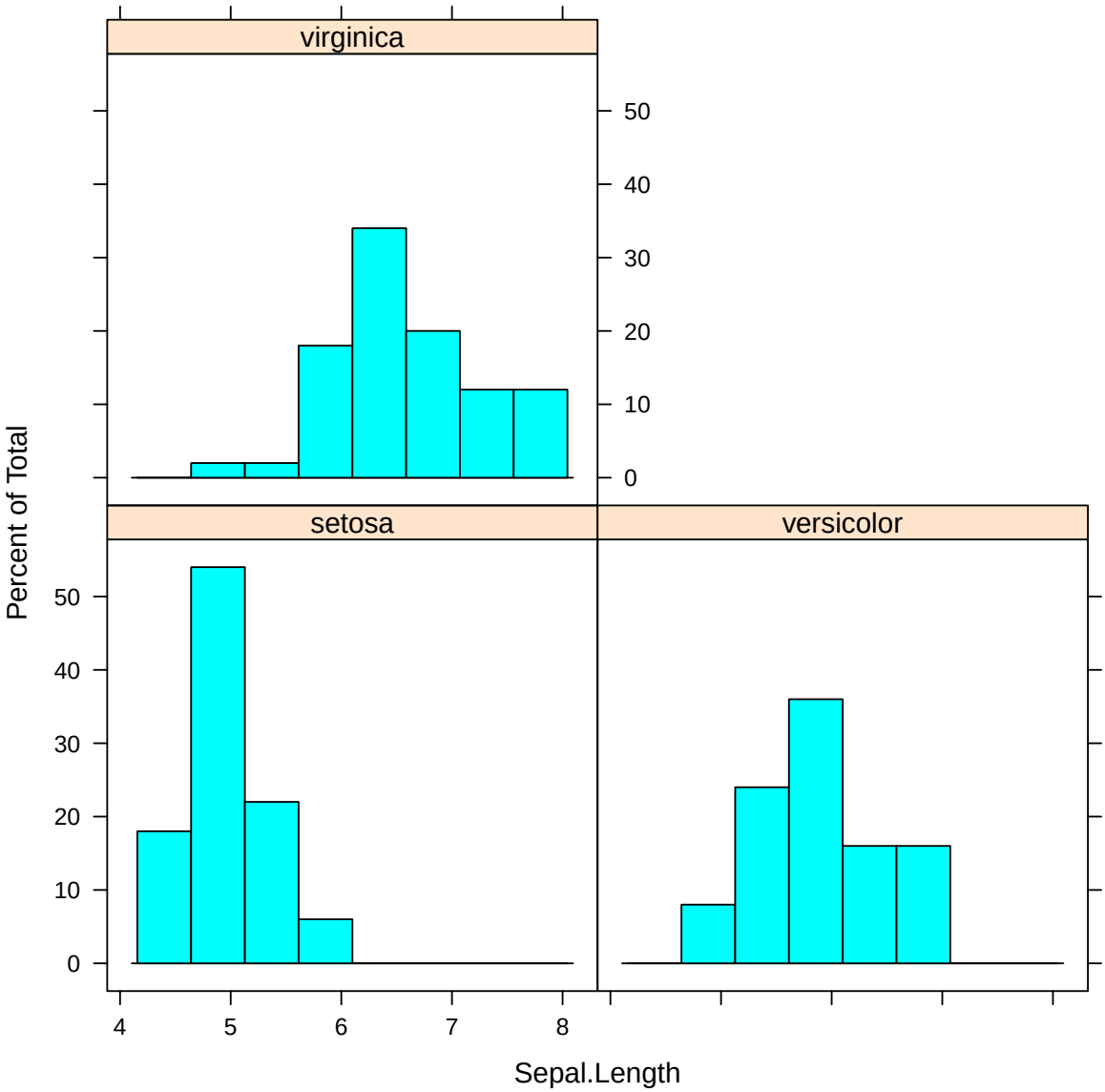

```
dotplot(Sepal.Length ~ Species, data = aggregate(subset(iris,
    select = -Species), list(Species = iris$Species),
    mean), xlab = NULL, ylab = "Mean sepal length")
```



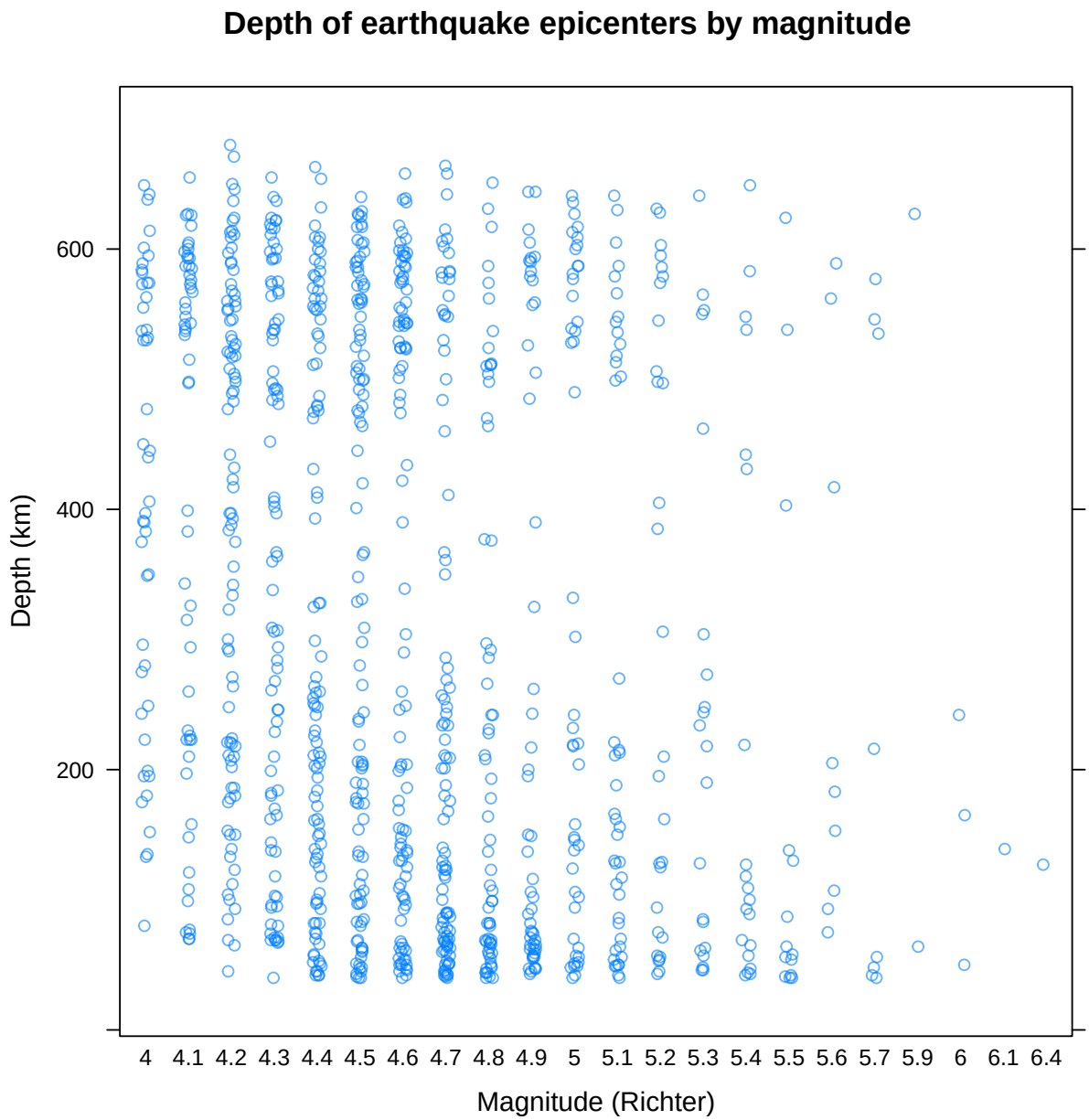

```
densityplot(~Sepal.Length, groups = Species,
    data = iris)
```




```
histogram(~Sepal.Length | Species, data = iris)
```

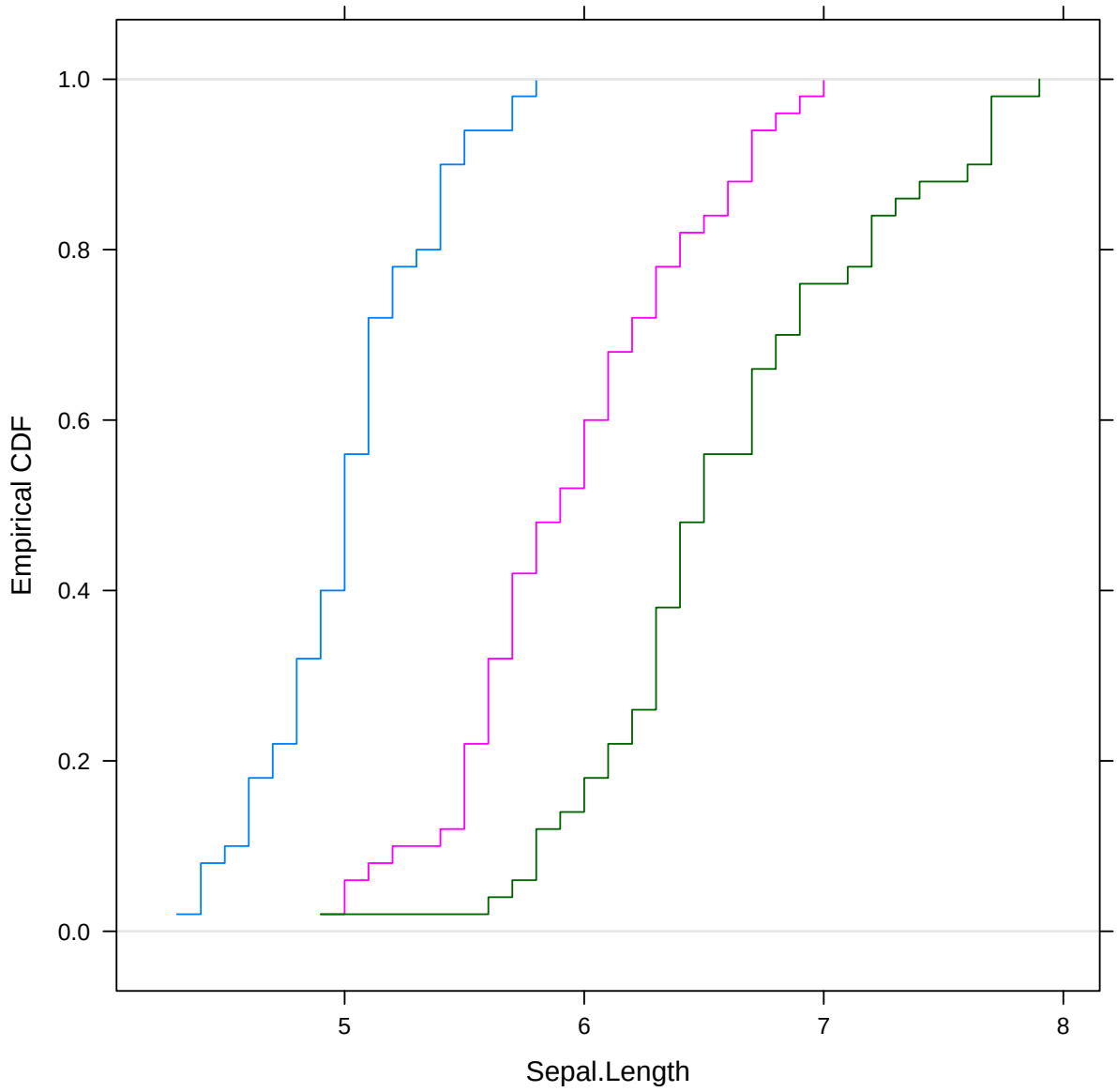



```
# from
# http://lattice.r-forge.r-project.org/Vignettes/src/lattice-intro/lattice-intro.pd
stripplot(depth ~ factor(mag), data = quakes,
    jitter.data = TRUE, alpha = 0.6, main = "Depth of earthquake epicenters by magn
    xlab = "Magnitude (Richter)", ylab = "Depth (km)")
```

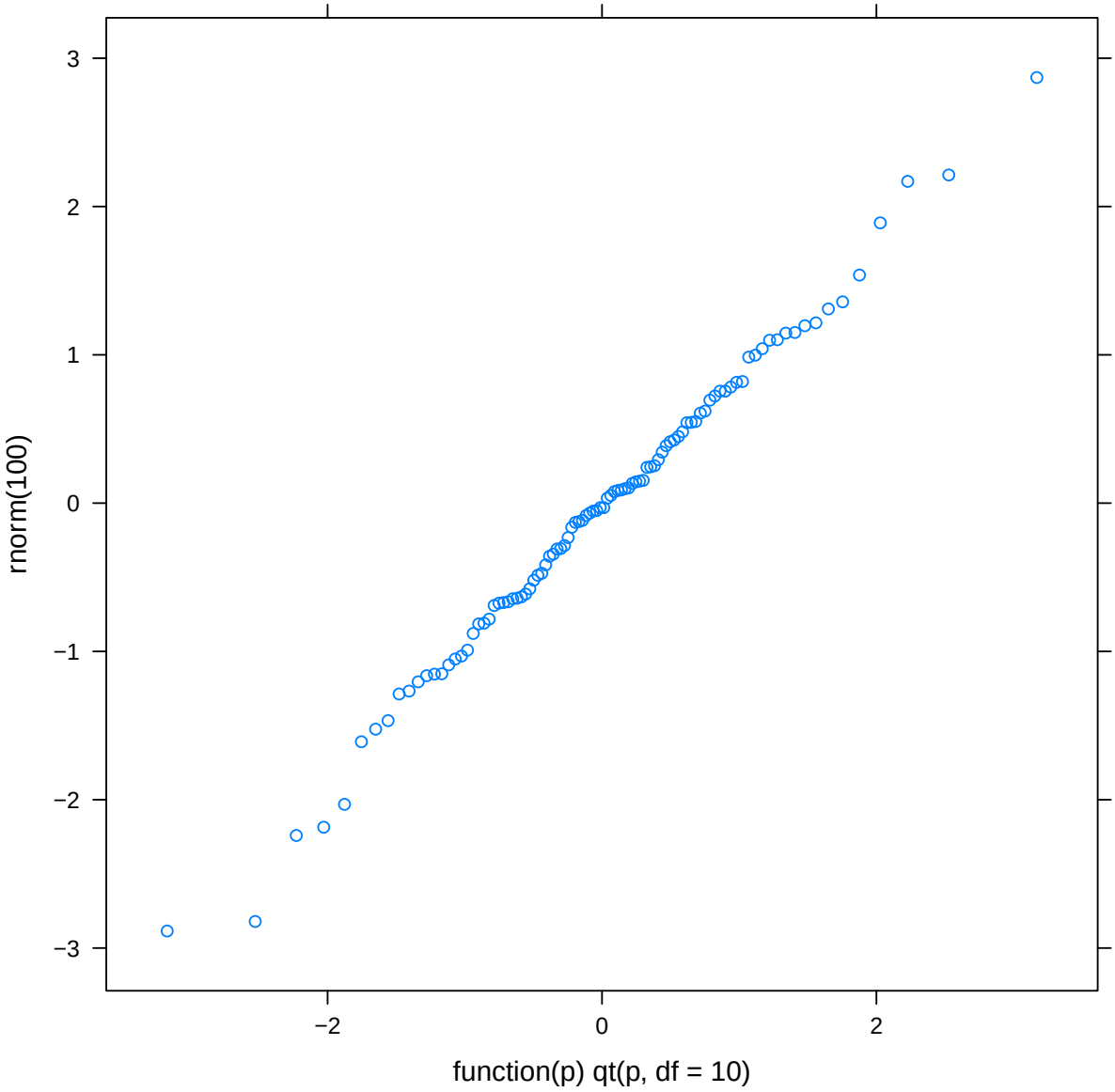


13.3 Densityplots, ecdfplots, qqmath (draft)

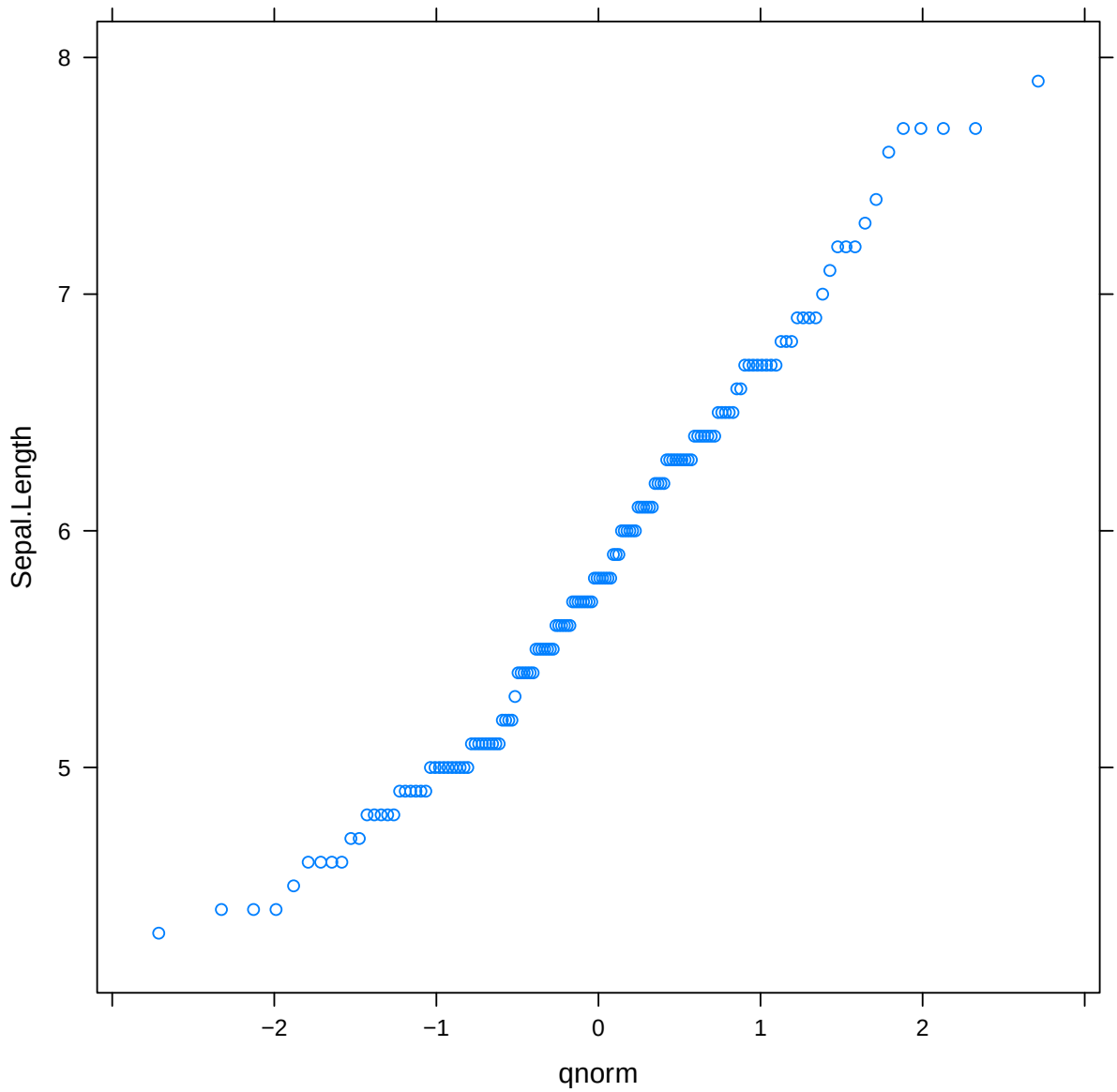
```
ecdfplot(~Sepal.Length, groups = Species,
    data = iris)
```




```
qqmath(~rnorm(100), distribution = function(p) qt(p,
    df = 10))
```




```
qqmath(~Sepal.Length, distribution = qnorm,
    data = iris)
```




```
qqmath(~Sepal.Length | Species, distribution = qnorm,
    data = iris) + layer_(panel.qqmathline(...))

## Error in get(x, envir = this, inherits = inh)(this, ...): '...' used in an
incorrect context
```

TODO: marginal.plot, countourplot