Machine Learning in R

Alexandros Karatzoglou[1]

[1]Telefonica Research
Barcelona, Spain

December 15, 2010

Outline

R

- Environment for statistical data analysis, inference and visualization.
- Ports for Unix, Windows and MacOSX
- Highly extensible through user-defined functions
- Generic functions and conventions for standard operations like plot, predict etc.
- $\sim$ 1200 add-on packages contributed by developers from all over the world
- e.g. Multivariate Statistics, Machine Learning, Natural Language Processing, Bioinformatics (Bioconductor), SNA, .
- Interfaces to C, C++, Fortran, Java

Outline

Comprehensive R Archive Network (CRAN)

- CRAN includes packages which provide additional functionality to the one existing in R
- Currently over 1200 packages in areas like multivariate statistics, time series analysis, Machine Learning, Geo-statistics, environmental statistics etc.
- packages are written mainly by academics, PhD students, or company staff
- Some of the package have been ordered into Task Views

CRAN Task Views

- Some of the CRAN packages are ordered into categories called *Task Views*
- *Task Views* are maintained by people with experience in the corresponding field
- there are *Task Views* on Econometrics, Environmental Statistics, Finance, Genetics, Graphics Machine Learning, Multivariate Statistics, Natural Language Processing, Social Sciences, and others.

Online documentation

1. Go to `http://www.r-project.org`
2. On the left side under Documentation
3. Official Manuals, FAQ, Newsletter, Books
4. Other and click on *other publications* contains a collection of contributed guides and manuals

Mailing lists

- *R-announce* is for announcing new major enhancements in R
- *R-packages* announcing new version of packages
- *R-help* is for R related user questions!
- *R-devel* is for R developers

Command Line Interface

- R does not have a “real” gui
- All computations and statistics are performed with commands
- These commands are called “functions” in R

help

- click on help
- search help functionality
- FAQ
- link to reference web pages
- apropos

Outline

Objects

Everything in R is an object
Everything in R has a class.

- Data, intermediate results and even e.g. the result of a regression are stored in R objects
- The **Class** of the object both describes what the object contains and what many standard functions
- Objects are usually accessed by name. Syntactic names for objects are made up from letters, the digits 0 to 9 in any non-initial position and also the period “.”

Assignment and Expression Operations

- R commands are either assignments or expressions
- Commands are separated either by a semicolon ; or newline
- An expression command is evaluated and (normally) printed
- An assignment command evaluates an expression and passes the value to a variable but the result is not printed

Expression Operations

```
> 1 + 1
[1] 2
```

Assigment Operations

```
> res <- 1 + 1
```

- "<-" is the assignment operator in R
- a series of commands in a file (script) can be executed with the command : source("myfile.R")

Sample session

```
> 1:5
[1] 1 2 3 4 5
> powers.of.2 <- 2^(1:5)
> powers.of.2
[1]  2  4  8 16 32
> class(powers.of.2)
[1] "numeric"
> ls()
[1] "powers.of.2" "res"
> rm(powers.of.2)
```

Workspace

- R stores objects in workspace that is kept in memory
- When quiting R ask you if you want to save that workspace
- The workspace containing all objects you work on can then be restored next time you work with R along with a history of the used commands.

Outline

Basic Data Structures

- Vectors
- Factors
- Data Frame
- Matrices
- Lists

Vectors

In R the building blocks for storing data are vectors of various types. The most common classes are:

- "character", a vector of character strings of varying length. These are normally entered and printed surrounded by double quotes.
- "numeric", a vector of real numbers.
- "integer", a vector of (signed) integers.
- "logical", a vector of logical (true or false) values. The values are printed and as TRUE and FALSE.

Numeric Vectors

```
> vect <- c(1, 2, 99, 6, 8, 9)
> is(vect)
[1] "numeric" "vector"
> vect[2]
[1] 2
> vect[2:3]
[1]  2 99
> length(vect)
[1] 6
> sum(vect)
[1] 125
```

Character Vectors

```
> vect3 <- c("austria", "spain",
+      "france", "uk", "belgium",
+      "poland")
> is(vect3)
[1] "character"
[2] "vector"
[3] "data.frameRowLabels"
[4] "SuperClassMethod"
> vect3[2]
[1] "spain"
> vect3[2:3]
[1] "spain"  "france"
> length(vect3)
[1] 6
```

Logical Vectors

```
> vect4 <- c(TRUE, TRUE, FALSE, TRUE,
+      FALSE, TRUE)
> is(vect4)
```

```
[1] "logical" "vector"
```

Factors

```
> citizen <- factor(c("uk", "us",
+     "no", "au", "uk", "us", "us"))
> citizen

[1] uk us no au uk us us
Levels: au no uk us

> unclass(citizen)

[1] 3 4 2 1 3 4 4
attr(,"levels")
[1] "au" "no" "uk" "us"

> citizen[5:7]

[1] uk us us
Levels: au no uk us

> citizen[5:7, drop = TRUE]

[1] uk us us
Levels: uk us
```

Factors Ordered

```
> income <- ordered(c("Mid", "Hi",
+       "Lo", "Mid", "Lo", "Hi"), levels = c("Lo",
+       "Mid", "Hi"))
> income

[1] Mid Hi  Lo  Mid Lo  Hi
Levels: Lo < Mid < Hi

> as.numeric(income)

[1] 2 3 1 2 1 3

> class(income)

[1] "ordered" "factor"

> income[1:3]

[1] Mid Hi  Lo
Levels: Lo < Mid < Hi
```

Data Frames

- A **data frame** is the type of object normally used in R to store a data matrix.
- It should be thought of as a list of variables of the same length, but possibly of different types (numeric, factor, character, logical, etc.).

Data Frames

```
> library(MASS)
> data(painters)
> painters[1:3, ]
```

```
           Composition Drawing Colour
Da Udine            10       8     16
Da Vinci            15      16      4
Del Piombo           8      13     16
           Expression School
Da Udine            3      A
Da Vinci           14      A
Del Piombo          7      A
```

```
> names(painters)
```

```
[1] "Composition" "Drawing"
[3] "Colour"      "Expression"
[5] "School"
```

```
> row.names(painters)
```

```
 [1] "Da Udine"          "Da Vinci"
```

Data Frames

- Data frames are by far the commonest way to store data in `R`
- They are normally imported by reading a file or from a spreadsheet or database.
- However, vectors of the same length can be collected into a data frame by the function `data.frame`

Data Frame

```
> mydf <- data.frame(vect, vect3,
+     income)
> summary(mydf)
```

```
      vect           vect3    income
 Min.   : 1.00   austria:1   Lo :2
 1st Qu.: 3.00   belgium:1   Mid:2
 Median : 7.00   france :1   Hi :2
 Mean   :20.83   poland :1
 3rd Qu.: 8.75   spain  :1
 Max.   :99.00   uk     :1
```

```
> mydf <- data.frame(vect, I(vect3),
+     income)
```

Matrices

- A data frame may be printed like a matrix, but it is not a matrix.
- Matrices like vectors have all their elements of the same type

```
> mymat <- matrix(1:10, 2, 5)
> class(mymat)

[1] "matrix"

> dim(mymat)

[1] 2 5
```

Lists

- A list is a vector of other R objects.
- Lists are used to collect together items of different classes.
- For example, an employee record might be created by :

Lists

```
> Empl <- list(employee = "Anna",
+      spouse = "Fred", children = 3,
+      child.ages = c(4, 7, 9))
> Empl$employee
[1] "Anna"
> Empl$child.ages[2]
[1] 7
```

Basic Mathematical Operations

```
> 5 - 3
[1] 2
> a <- 2:4
> b <- rep(1, 3)
> a - b
[1] 1 2 3
> a * b
[1] 2 3 4
```

Matrix Multiplication

- "

```
> a <- matrix(1:9, 3, 3)
> b <- matrix(2, 3, 3)
> c <- a %*% b
```

Outline

Missing Values

- Missing Values in `R` are labeled with the logical value NA

Missing Values

```
> mydf[mydf == 99] <- NA
> mydf
  vect   vect3 income
1    1 austria    Mid
2    2   spain     Hi
3   NA  france     Lo
4    6      uk    Mid
5    8 belgium     Lo
6    9  poland     Hi
```

Missing Values

```
> a <- c(3, 5, 6)
> a[2] <- NA
> a
[1]  3 NA  6
> is.na(a)
[1] FALSE  TRUE FALSE
```

Special Values

- R has the also special values NaN , Inf and -Inf

```
> d <- c(-1, 0, 1)
> d/0
```

```
[1] -Inf  NaN  Inf
```

Outline

Entering Data

- For all but the smallest datasets the easiest way to get data into R is to import it from a connection such as a file

Entering Data

```
> a <- c(5, 8, 5, 4, 9, 7)
> b <- scan()
```

Entering Data

```
> mydf2 <- edit(mydf)
```

Outline

File Input and Output

- read.table can be used to read data from text file like csv
- read.table creates a data frame from the values and tries to guess the type of each variable

```
> mydata <- read.table("file.csv",
+       sep = ",")
```

Packages

- Packages contain additional functionality in the form of extra functions and data
- Installing a package can be done with the function `install.packages()`
- The default `R` installation contains a number of contributed packages like MASS, foreign, utils
- Before we can access the functionality in a package we have to load the package with the function `library()`

Outline

Packages

- Help on the contents of the packages is available

```
> library(help = foreign)
```

- Help on installed packages is also available by `help.start()`
- Vignettes are also available for many packages

SPSS files

- Package foreign contains functions that read data from SPSS (sav), STATA and other formats

```
> library(foreign)
> spssdata <- read.spss("ees04.sav",
+     to.data.frame = TRUE)
```

Excel files

- To load Excel file into `R` an external package `xlsReadWrite` is needed
- We will install the package directly from CRAN using `install.packages`

```
> install.packages("xlsReadWrite",
+      lib = "C:\temp")
```

Excel files

```
> library(xlsReadWrite)
> data <- read.xls("sampledata.xls")
```

Data Export

- You can save your data by using the functions `write.table()` or `save()`
- `write.table` is more data specific, while `save` can save any `R` object
- `save` saves in a binary format while `write.table()` in text format.

```
> write.table(mydata, file = "mydata.csv",
+     quote = FALSE)
> save(mydata, file = "mydata.rda")
> load(file = "mydata.rda")
```

Outline

Indexing

- R has a powerful set of Indexing capabilities
- This facilitates Data manipulation and can speed up data pre-processing
- Indexing Vectors, Data Frames and matrices is done in a similar manner

Indexing by numeric vectors

```
> letters[1:3]
[1] "a" "b" "c"
> letters[c(7, 9)]
[1] "g" "i"
> letters[-(1:15)]
 [1] "p" "q" "r" "s" "t" "u" "v" "w" "x"
[10] "y" "z"
```

Indexing by logical vectors

```
> a <- 1:10
> a[c(3, 5, 7)] <- NA
> is.na(a)

 [1] FALSE FALSE  TRUE FALSE  TRUE FALSE
 [7]  TRUE FALSE FALSE FALSE

> a[!is.na(a)]

[1]  1  2  4  6  8  9 10
```

Indexing Matrices and Data Frames

```
> mymat[1, ]

[1] 1 3 5 7 9

> mymat[, c(2, 3)]

     [,1] [,2]
[1,]    3    5
[2,]    4    6

> mymat[1, -(1:3)]

[1] 7 9
```

Selecting Subsets

```
> attach(painters)
> painters[Colour >= 17, ]
```

	Composition	Drawing	Colour	Expression	School
Bassano	6	8	17	0	D
Giorgione	8	9	18	4	D
Pordenone	8	14	17	5	D
Titian	12	15	18	6	D
Rembrandt	15	6	17	12	G
Rubens	18	13	17	17	G
Van Dyck	15	10	17		

Selecting Subsets

```
> painters[Colour >= 15 & Composition >
+     10, ]
```

	Composition	Drawing	Colour	Expression	School
Tintoretto	15	14	16	4	D
Titian	12	15	18	6	D
Veronese	15	10	16	3	D
Corregio	13	13	15	12	E
Rembrandt	15	6	17	12	G
Rubens	18	13	17	17	G
Van Dyck	15	10	17		

Graphics

- R has a rich set of visualization capabilities
- scatter-plots, histograms, box-plots and more.

histograms

```
> data(hills)
> hist(hills$time)
```

Scatter Plot

```
> plot(climb ~ time, hills)
```

Paired Scatterplot

```
> pairs(hills)
```

Box Plots

```
> boxplot(count ~ spray, data = InsectSprays,
+         col = "lightgray")
```

Formula Interface

- A formula is of the general form *response* $\sim$ *expression* where the left-hand side, *response*, may in some uses be absent and the right-hand side, *expression*, is a collection of terms joined by operators usually resembling an arithmetical expression.
- The meaning of the right-hand side is context dependent
- e.g. in linear and generalized linear modelling it specifies the form of the model matrix

Scatterplot and line

```
> library(MASS)
> data(hills)
> attach(hills)
> plot(climb ~ time, hills, xlab = "Time",
+      ylab = "Feet")
> abline(0, 40)
> abline(lm(climb ~ time))
```

Multiple plots

```
> par(mfrow = c(1, 2))
> plot(climb ~ time, hills, xlab = "Time",
+      ylab = "Feet")
> plot(dist ~ time, hills, xlab = "Time",
+      ylab = "Dist")
> par(mfrow = c(1, 1))
```

Plot to file

```
> pdf(file = "H:/My Documents/plot.pdf")
> plot(dist ~ time, hills, xlab = "Time",
+      ylab = "Dist", main = "Hills")
> dev.off()
```

Paired Scatterplot

```
> splom(~hills)
```

Box-whisker plots

```
> data(michelson)
> bwplot(Expt ~ Speed, data = michelson,
+        ylab = "Experiment No.")
> title("Speed of Light Data")
```

Box-whisker plots

```
> data(quine)
> bwplot(Age ~ Days | Sex * Lrn *
+     Eth, data = quine, layout = c(4,
+     2))
```

Discriptive Statistics

```
> mean(hills$time)
[1] 57.87571
> colMeans(hills)
      dist       climb        time
  7.528571 1815.314286   57.875714
> median(hills$time)
[1] 39.75
> quantile(hills$time)
     0%     25%     50%     75%    100%
 15.950  28.000  39.750  68.625 204.617
> var(hills$time)
[1] 2504.073
> sd(hills$time)
[1] 50.04072
```

Discriptive Statistics

```
> cor(hills)

             dist      climb       time
dist  1.0000000 0.6523461 0.9195892
climb 0.6523461 1.0000000 0.8052392
time  0.9195892 0.8052392 1.0000000

> cov(hills)

              dist        climb        time
dist     30.51387    5834.638   254.1944
climb 5834.63782 2621648.457 65243.2567
time   254.19442   65243.257  2504.0733

> cor(hills$time, hills$climb)

[1] 0.8052392
```

Distributions

- R contains a number of functions for drawing from a number of distribution like e.g. normal, uniform etc.

Sampling from a Normal Distribution

```
> nvec <- rnorm(100, 3)
```

Outline

Testing for the mean T-test

```
> t.test(nvec, mu = 1)

One Sample t-test

data:  nvec
t = 21.187, df = 99, p-value <
2.2e-16
alternative hypothesis: true mean is not equal to 1
95 percent confidence interval:
 2.993364 3.405311
sample estimates:
mean of x
 3.199337
```

Testing for the mean T-test

```
> t.test(nvec, mu = 2)

One Sample t-test

data:  nvec
t = 11.5536, df = 99, p-value <
2.2e-16
alternative hypothesis: true mean is not equal to 2
95 percent confidence interval:
 2.993364 3.405311
sample estimates:
mean of x
 3.199337
```

Testing for the mean Wilcox-test

```
> wilcox.test(nvec, mu = 3)

Wilcoxon signed rank test with
continuity correction

data:  nvec
V = 3056, p-value = 0.06815
alternative hypothesis: true location is not equal to
```

Two Sample Tests

```
> nvec2 <- rnorm(100, 2)
> t.test(nvec, nvec2)

Welch Two Sample t-test

data:  nvec and nvec2
t = 7.4455, df = 195.173, p-value =
3.025e-12
alternative hypothesis: true difference in means is nc
95 percent confidence interval:
 0.8567072 1.4741003
sample estimates:
mean of x mean of y
 3.199337  2.033933
```

Two Sample Tests

```
> wilcox.test(nvec, nvec2)

Wilcoxon rank sum test with
continuity correction

data:  nvec and nvec2
W = 7729, p-value = 2.615e-11
alternative hypothesis: true location shift is not equ
```

Paired Tests

```
> t.test(nvec, nvec2, paired = TRUE)

Paired t-test

data:  nvec and nvec2
t = 7.6648, df = 99, p-value =
1.245e-11
alternative hypothesis: true difference in means is nc
95 percent confidence interval:
 0.863712 1.467096
sample estimates:
mean of the differences
                1.165404
```

Paired Tests

```
> wilcox.test(nvec, nvec2, paired = TRUE)

Wilcoxon signed rank test with
continuity correction

data:  nvec and nvec2
V = 4314, p-value = 7.776e-10
alternative hypothesis: true location shift is not equ
```

Density Estimation

```
> library(MASS)
> truehist(nvec, nbins = 20, xlim = c(0,
+     6), ymax = 0.7)
> lines(density(nvec, width = "nrd"))
```

Density Estimation

```
> data(geyser)
> geyser2 <- data.frame(as.data.frame(geyser)[-1,
+     ], pduration = geyser$duration[-299])
> attach(geyser2)
> par(mfrow = c(2, 2))
> plot(pduration, waiting, xlim = c(0.5,
+     6), ylim = c(40, 110), xlab = "previous duration
+     ylab = "waiting")
> f1 <- kde2d(pduration, waiting,
+     n = 50, lims = c(0.5, 6, 40,
+         110))
> image(f1, zlim = c(0, 0.075), xlab = "previous durat
+     ylab = "waiting")
> f2 <- kde2d(pduration, waiting,
+     n = 50, lims = c(0.5, 6, 40,
+         110), h = c(width.SJ(duration),
+         width.SJ(waiting)))
> image(f2, zlim = c(0, 0.075), xlab = "previous durat
```

Simple Linear Model

Fitting a simple linear model in R is done using the lm function

```
> data(hills)
> mhill <- lm(time ~ dist, data = hills)
> class(mhill)

[1] "lm"

> mhill

Call:
lm(formula = time ~ dist, data = hills)

Coefficients:
(Intercept)         dist
     -4.841        8.330
```

Simple Linear Model

```
> summary(mhill)
> names(mhill)
> mhill$residuals
```

Multivariate Linear Model

```
> mhill2 <- lm(time ~ dist + climb,
+      data = hills)
> mhill2

Call:
lm(formula = time ~ dist + climb, data = hills)

Coefficients:
(Intercept)         dist        climb
   -8.99204      6.21796      0.01105
```

Multifactor Linear Model

```
> summary(mhill2)
> update(update(mhill2, weights = 1/hills$dist^2))
> predict(mhill2, hills[2:3, ])
```

Bayes Rule

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \tag{1}$$

Consider the problem of email filtering: x is the email e.g. in the form of a word vector, y the label *spam*, *ham*

Prior $p(y)$

$$p(ham) \approx \frac{m_{ham}}{m_{total}}, \quad p(spam) \approx \frac{m_{spam}}{m_{total}} \tag{2}$$

Likelihood Ratio

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \tag{3}$$

key problem: we do not know $p(x|y)$ or $p(x)$. We can get rid of $p(x)$ by settling for a likelihood ratio:

$$L(x) := \frac{p(spam|x)}{p(ham|x)} = \frac{p(x|spam)p(spam)}{p(x|ham)p(ham)} \tag{4}$$

Whenever $L(x)$ exceeds a given threshold c we decide that x is spam and consequently reject the e-mail

Computing $p(x|y)$

Key assumption in Naive Bayes is that the variables (or elements of x) are conditionally independent i.e. words in emails are independent of each other. We can then compute $p(x|y)$ in a *naive* fashion by assuming that:

$$p(x|y) = \prod_{j=1}^{Nwords \in x} p(w^j|y) \tag{5}$$

w_j denotes the j-th word in document x. Estimates for $p(w|y)$ can be obtained, for instance, by simply counting the frequency occurrence of the word within documents of a given class

Computing p(x|y)

$$p(w|spam) \approx \frac{\sum_{i=1}^{m} \sum_{j=1}^{Nwords \in x_i} \{y_i = spam \& w_i^j = w\}}{\sum_{i=1}^{m} \sum_{j=1}^{Nwords \in x_i} \{y_i = spam\}} \quad (6)$$

$\{y_i = spam \& w_i^j = w\}$ equals 1 if x_i is labeled as spam and w occurs as the j-th word in x_i. The denominator counts the number of words in spam documents.

Laplacian smoothing

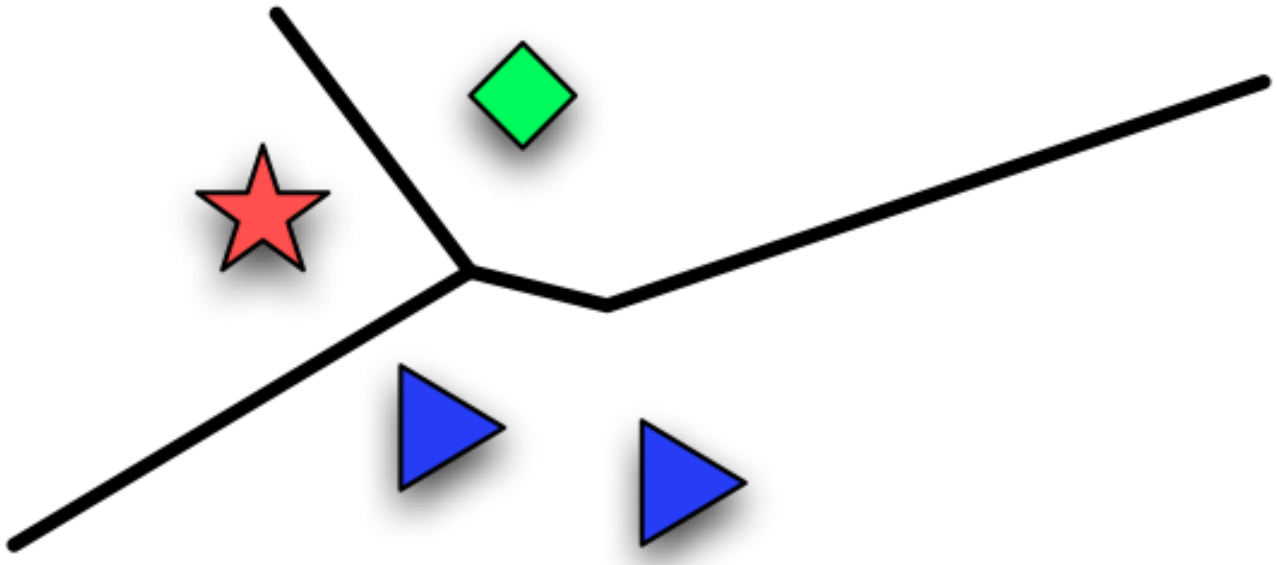
Issue: estimating $p(w|y)$ for words w which we might not have seen before.
Solution:
increment all counts by 1. This method is commonly referred to as Laplace smoothing.

Naive Bayes in R

```
> library(kernlab)
> library(e1071)
> data(spam)
> idx <- sample(1:dim(spam)[1], 300)
> spamtrain <- spam[-idx, ]
> spamtest <- spam[idx, ]
> model <- naiveBayes(type ~ ., data = spamtrain)
> predict(model, spamtest)
> table(predict(model, spamtest),
+       spamtest$type)
> predict(model, spamtest, type = "raw")
```

k-Nearest Neighbors

An even simpler estimator than Naive Bayes is nearest neighbors. In its most basic form it assigns the label of its nearest neighbor to an



observation x

Identify k nearest neigbhboors given distance metric $d(x, x')$ (e.g. euclidian distance) and determine label by a vote.

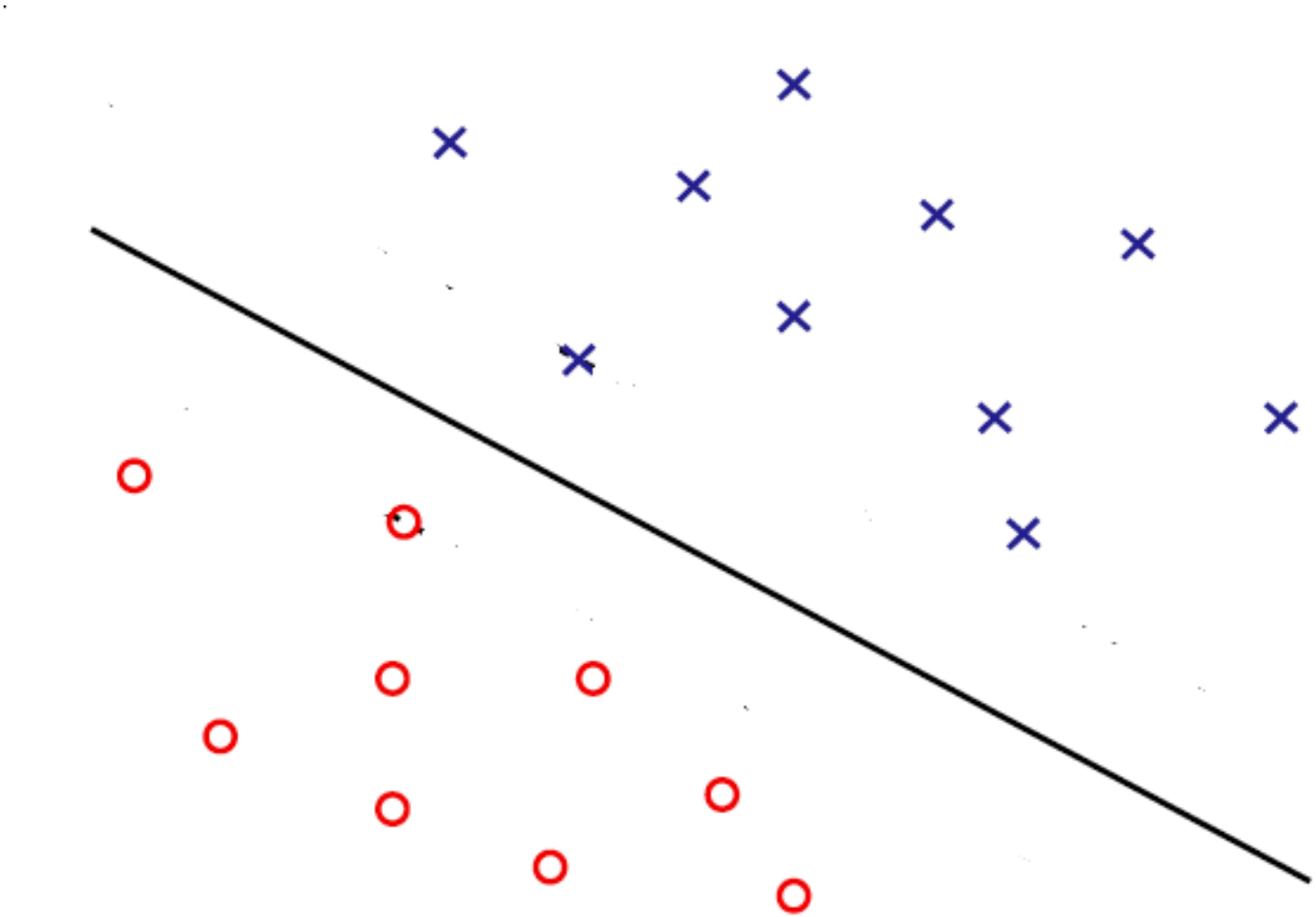
k-Nearest Neighbors

```
> library(animation)
> knn.ani(k = 4)
```

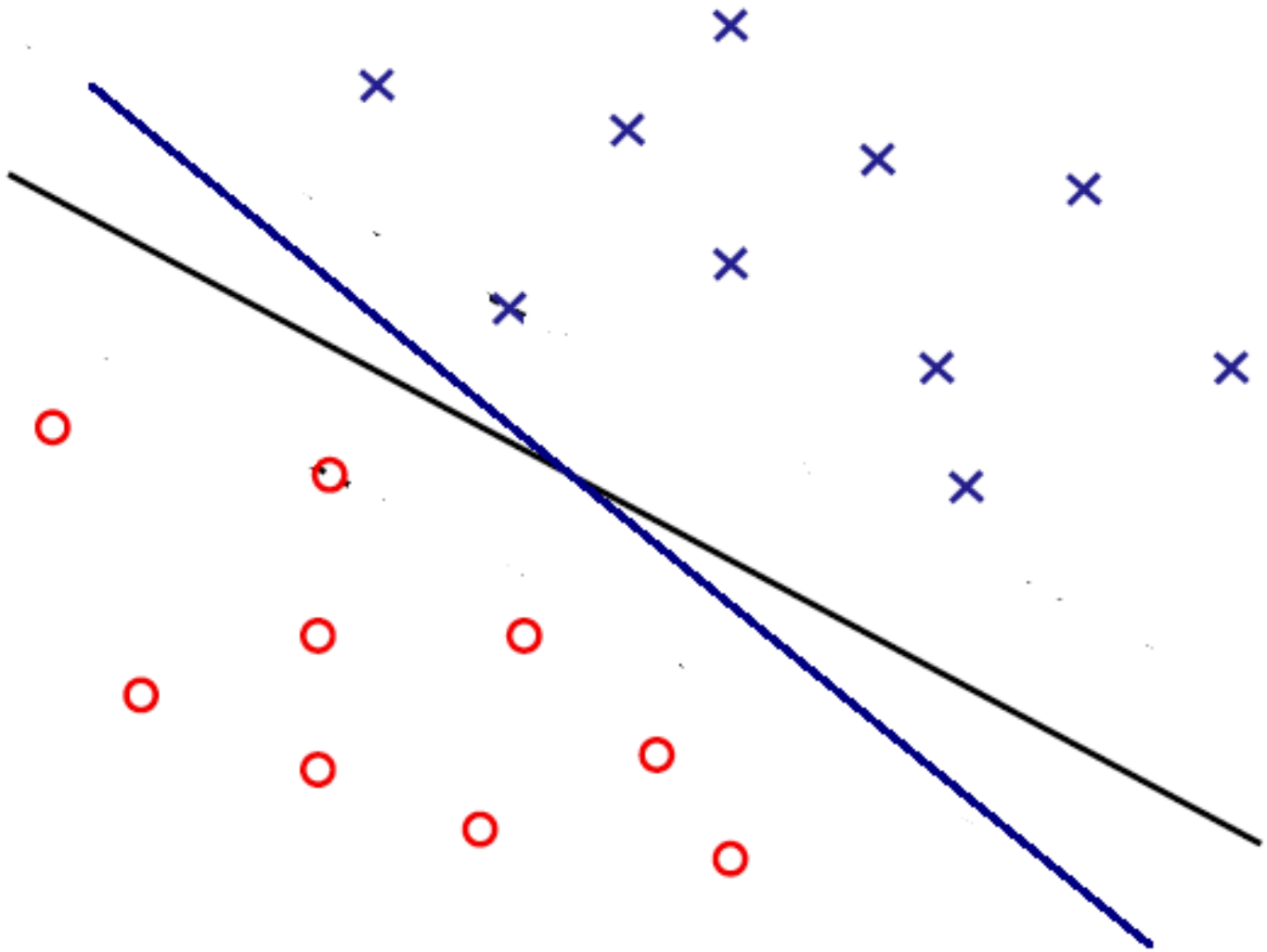
KNN in R

```
> model <- knn(spamtrain[, -58],
+     spamtest[, -58], spamtrain[,
+         58])
> predict(model, spamtest)
> table(predict(model, spamtest),
+     spamtest$type)
```

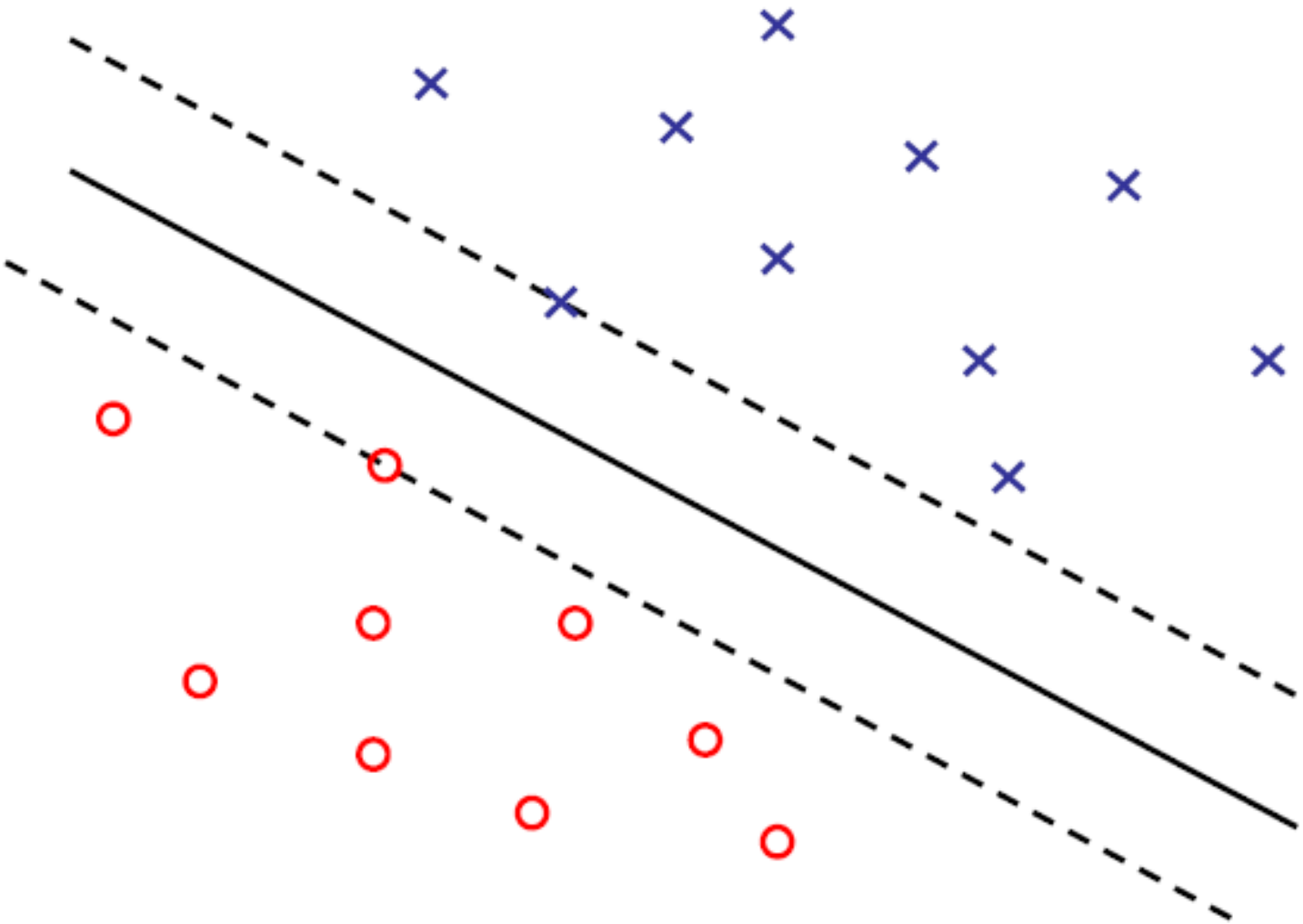
Support Vector Machines



Support Vector Machines



Support Vector Machines



Support Vector Machines

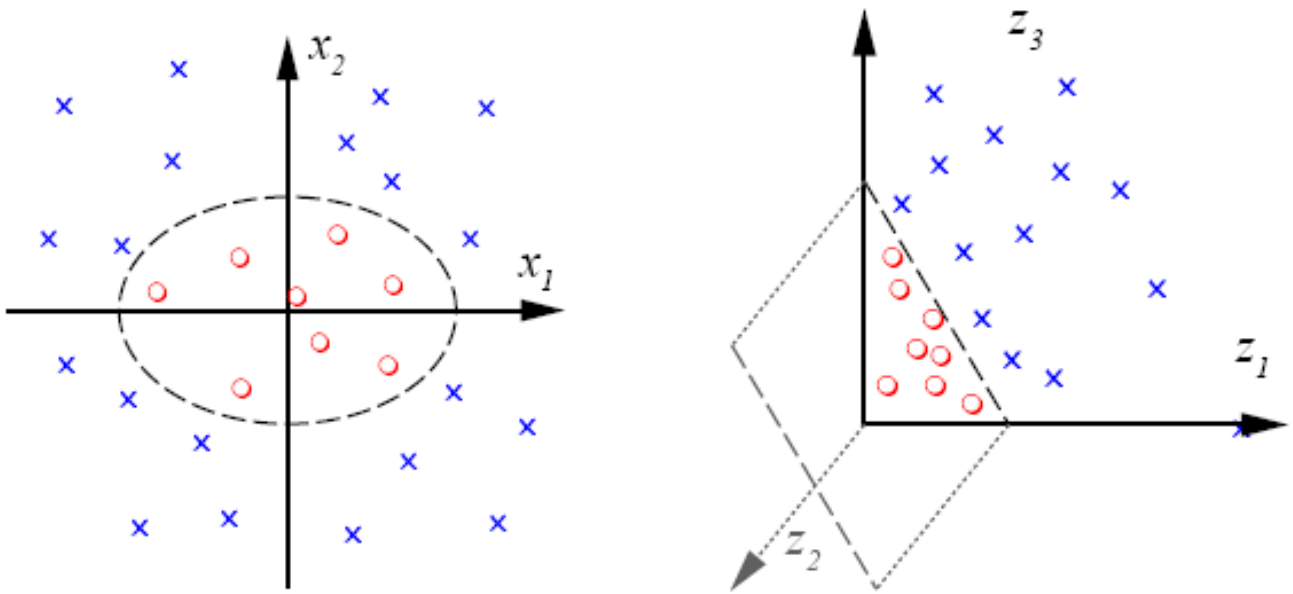
- Support Vector Machines work by maximizing a margin between a hyperplane and the data.
- SVM is a simple well understood linear method
- The optimization problem is convex thus only one optimal solution exists
- Excellent performance

Kernels: Data Mapping

Data are implicitly mapped into a high dimensional feature space
$\Phi : X \rightarrow H$

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x_1, x_2) \rightarrow (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



Kernel Methods: Kernel Function

- In Kernel Methods learning takes place in the feature space, data only appear inside inner products $\langle \Phi(x), \Phi(x') \rangle$
- Inner product is given by a kernel function ("kernel trick")

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

- Inner product :

$$\langle \Phi(x), \Phi(x') \rangle = (x_1^2, \sqrt{2}x_1x_2, x_2^2)({x'_1}^2, \sqrt{2}x'_1{x'}_2, {x'_2}^2)^T$$

$$= \langle x, x' \rangle^2$$

$$= k(x, x')$$

Kernel Functions

- Gaussian kernel $k(x, x') = \exp(-\sigma \|x - x'\|^2)$
- Polynomial kernel $k(x, x') = (\langle x, x' \rangle + c)^p$
- String Kernels (for text):

$$k(x, x') = \sum_{s \sqsubseteq x, s' \sqsubseteq x'} \lambda_s \delta_{s,s'} = \sum_{s \in A} num_s(x) num_s(x') \lambda_s$$

- Kernels on Graphs, Mismatch kernels etc.

SVM Primal Optimization Problem

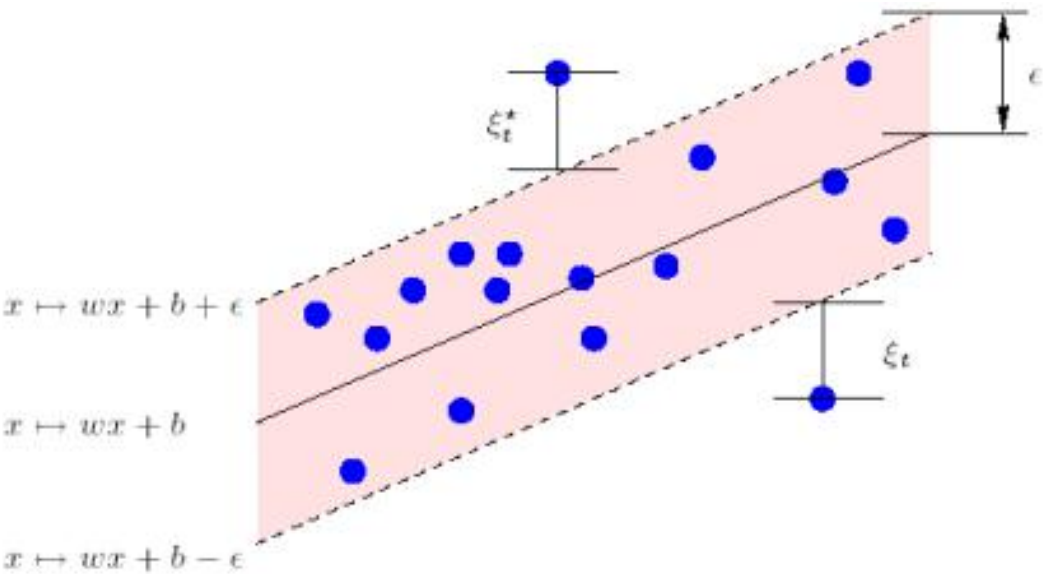
$$\begin{aligned} \text{minimize} \quad & t(\mathbf{w}, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\langle \Phi(x_i), \mathbf{w}\rangle + b) \geq 1 - \xi_i \qquad (i = 1, \ldots, m) \\ & \xi_i \geq 0 \qquad (i = 1, \ldots, m) \end{aligned}$$

where m is the number of training patterns, and $y_i = \pm 1$.
SVM Decision Function

$$f(x_i) = \langle \mathbf{w}, \Phi(x_i)\rangle + b$$

$$f_i = \sum_{j=1}^{m} k(x_i, x_j)\alpha_j$$

SVM for Regression



SVM in R

```
> filter <- ksvm(type ~ ., data = spamtrain,
+     kernel = "rbfdot", kpar = list(sigma = 0.05),
+     C = 5, cross = 3)
> filter
> mailtype <- predict(filter, spamtest[,
+     -58])
> table(mailtype, spamtest[, 58])
```

SVM in R

```
> filter <- ksvm(type ~ ., data = spamtrain,
+     kernel = "rbfdot", kpar = list(sigma = 0.05),
+     C = 5, cross = 3, prob.model = TRUE)
> filter
> mailpro <- predict(filter, spamtest[,
+     -58], type = "prob")
> mailpro
```

SVM in R

```
> x <- rbind(matrix(rnorm(120), ,
+     2), matrix(rnorm(120, mean = 3),
+     , 2))
> y <- matrix(c(rep(1, 60), rep(-1,
+     60)))
> svp <- ksvm(x, y, type = "C-svc",
+     kernel = "vanilladot", C = 200)
> svp
> plot(svp, data = x)
```

SVM in R

```
> svp <- ksvm(x, y, type = "C-svc",
+       kernel = "rbfdot", kpar = list(sigma = 1),
+       C = 10)
> plot(svp, data = x)
```

SVM in R

```
> svp <- ksvm(x, y, type = "C-svc",
+       kernel = "rbfdot", kpar = list(sigma = 1),
+       C = 10)
> plot(svp, data = x)
```

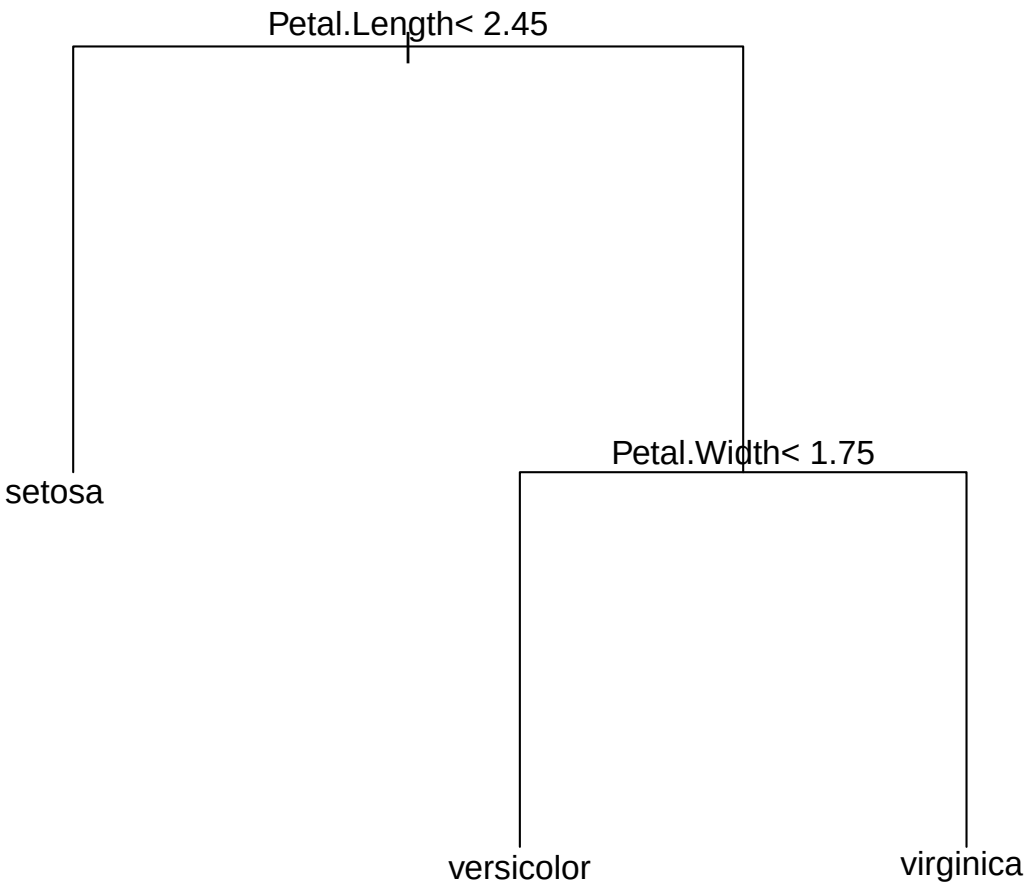
SVM in R

```
> data(reuters)
> is(reuters)
> tsv <- ksvm(reuters, rlabels, kernel = "stringdot",
+      kpar = list(length = 5), cross = 3,
+      C = 10)
> tsv
```

SVM in R

```
> x <- seq(-20, 20, 0.1)
> y <- sin(x)/x + rnorm(401, sd = 0.03)
> plot(x, y, type = "l")
> regm <- ksvm(x, y, epsilon = 0.02,
+      kpar = list(sigma = 16), cross = 3)
> regm
> lines(x, predict(regm, x), col = "red")
```

Decision Trees



Decision Trees

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model in each one. The partitioning is done taking one variable so that the partitioning of that variable increases some impurity measure $Q(x)$. In a node m of a classification tree let

$$p_{mk} = \frac{1}{N} \sum_{x_i \in R_m} I(y_i = k) \tag{7}$$

the portion of class k observation in node m. Trees classify the observations in each node m to class

$$k(m) = argmax_k p_{mk} \tag{8}$$

Impurity Measures

Misclassification error:

$$\frac{1}{N_m} \sum_{i \in R_m} I(y \neq k(m)) \tag{9}$$

Gini Index:

$$\sum_{k \neq k'} p_{mk} p_{mk'} = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{10}$$

Cross-entropy or deviance:

$$-\sum_{k=1}^{K} p_{mk} log(p_{mk}) \tag{11}$$

Classification Trees in R

```
> data(iris)
> tree <- rpart(Species ~ ., data = iris)
> plot(tree)
> text(tree, digits = 3)
```

Classification Trees in R

```
> fit <- rpart(Kyphosis ~ Age + Number +
+     Start, data = kyphosis)
> fit2 <- rpart(Kyphosis ~ Age +
+     Number + Start, data = kyphosis,
+     parms = list(prior = c(0.65,
+         0.35), split = "information"))
> fit3 <- rpart(Kyphosis ~ Age +
+     Number + Start, data = kyphosis,
+     control = rpart.control(cp = 0.05))
> par(mfrow = c(1, 3), xpd = NA)
> plot(fit)
> text(fit, use.n = TRUE)
> plot(fit2)
> text(fit2, use.n = TRUE)
> plot(fit3)
> text(fit3, use.n = TRUE)
```

Regression Trees in R

```
> data(cpus, package = "MASS")
> cpus.ltr <- tree(log10(perf) ~
+      syct + mmin + mmax + cach +
+           chmin + chmax, cpus)
> cpus.ltr
```

Random Forests

RF is an ensemble classifier that consist of many decision trees. Decisions are taken using a majority vote from the trees.

- Number of training cases be N, and the number of variables in the classifier be M.
- m is the number of input variables to be used to determine the decision at a node of the tree; m should be much less than M.
- Sample a training set for this tree by choosing N times with replacement from all N available training cases (i.e. take a bootstrap sample).
- For each node of the tree, randomly choose m variables on which to base the decision at that node.
- Calculate the best split based on these m variables in the training set.

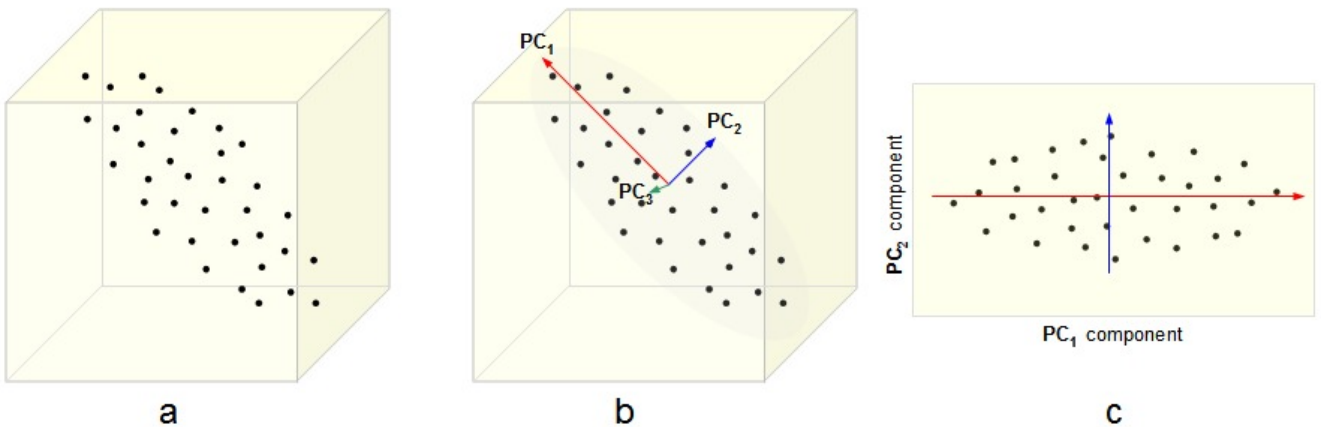
Random Forests in R

```
> library(randomForest)
> filter <- randomForest(type ~ .,
+     data = spam)
> filter
```

Principal Component Analysis (PCA)

- A projection method finding projections of maximal variability
- i.e. it seeks linear combinations of the columns of X with maximal (or minimal) variance
- The first *k* principal components span a subspace containing the "best" kdimensional view of the data
- Projecting the data on the first few principal components is often useful to reveal structure in the data
- PCA depends on the scaling of the original variables,
- thus it is conventional to do PCA using the correlation matrix, implicitly rescaling all the variables to unit sample variance.

Principal Component Analysis (PCA)



Principal Components Analysis (PCA)

```
> ir.pca <- princomp(log(iris[, -5]),
+     cor = T)
> summary(ir.pca)
> loadings(ir.pca)
> ir.pc <- predict(ir.pca)
```

PCA Plot

```
> plot(ir.pc[, 1:2], xlab = "first principal component",
+     ylab = "second principal component")
> text(ir.pc[, 1:2], labels = as.character(iris[,
+     5]), col = as.numeric(iris[,
+     5]))
```

kernel PCA

```
> test <- sample(1:150, 20)
> kpc <- kpca(~., data = iris[-test,
+     -5], kernel = "rbfdot", kpar = list(sigma = 0.2),
+     features = 2)
> plot(rotated(kpc), col = as.integer(iris[-test,
+     5]), xlab = "1st Principal Component",
+     ylab = "2nd Principal Component")
> text(rotated(kpc), labels = as.character(iris[-test,
+     5]), col = as.numeric(iris[-test,
+     5]))
```

kernel PCA

```
> kpc <- kpca(reuters, kernel = "stringdot",
+     kpar = list(length = 5), features = 2)
> plot(rotated(kpc), col = as.integer(rlabels),
+     xlab = "1st Principal Component",
+     ylab = "2nd Principal Component")
```

Distance methods

- Distance Methods work by representing the cases in a low dimensional Euclidian space so that their proximity reflects the similarity of their vairables
- A distance measure needs to be defined when using Distance Methods
- The function `dist()` implements four distance measures between the points in the p-dimensional space of variables; the default is Euclidean distance
- can be used with categorical variables!

Multidimensional scaling

```
> dm <- dist(iris[, -5])
> mds <- cmdscale(dm, k = 2)
> plot(mds, xlab = "1st coordinate",
+      ylab = "2nd coordinate", col = as.numeric(iris[,
+          5]))
```

Miltidimensional scaling for categorical v.

```
> library(kernlab)
> data(income)
> inc <- income[1:300, ]
> daisy(inc)
> csc <- cmdscale(as.dist(daisy(inc)),
+     k = 2)
> plot(csc, xlab = "1st coordinate",
+     ylab = "2nd coordinate")
```

Shannon's non-linear mapping

```
> snm <- sammon(dist(ir[-143, ]))
> plot(snm$points, xlab = "1st coordinate",
+     ylab = "2nd coordinate", col = as.numeric(iris[,
+         5]))
```

Biplot

```
> data(state)
> state <- state.x77[, 2:7]
> row.names(state) <- state.abb
> biplot(princomp(state, cor = T),
+     pc.biplot = T, cex = 0.7, expand = 0.8)
```

Stars plot

stars(state.x77[, c(7, 4, 6, 2, 5, 3)], full = FALSE,key.loc = c(10, 2))

Factor Analysis

- Principal component analysis looks for linear combinations of the data matrix that are uncorrelated and of high variance
- Factor analysis seeks linear combinations of the variables, called factors, that represent underlying fundamental quantities of which the observed variables are expressions
- the idea being that a small number of factors might explain a large number of measurements in an observational study

Factor Analysis

```
> data(swiss)
> swiss.x <- as.matrix(swiss[, -1])
> swiss.FA1 <- factanal(swiss.x,
+     method = "mle")
> swiss.FA1
> summary(swiss.FA1)
```

k-means Clustering

Partition data into k sets $S = \{S_1, S_2, \ldots, \S_k\}$ so as to minimize the within-cluster sum of squares (WCSS):

$$argmin_S \sum_{i=1}^{k} \sum_{x_j \in S_i} |x_j - \mu_i|^2 \tag{12}$$

k-means Clustering

```
> data(iris)
> clust <- kmeans(iris[, -5], centers = 3)
> clust
```

k-means Clustering I

```
> data(swiss)
> swiss.x <- as.matrix(swiss[, -1])
> km <- kmeans(swiss.x, 3)
> swiss.pca <- princomp(swiss.x)
> swiss.px <- predict(swiss.pca)
```

k-means Clustering II

```
> dimnames(km$centers)[[2]] <- dimnames(swiss.x)[[2]]
> swiss.centers <- predict(swiss.pca,
+     km$centers)
> plot(swiss.px[, 1:2], xlab = "first principal compon
+     ylab = "second principal component",
+     col = km$cluster)
```

k-means Clustering III

```
> points(swiss.centers[, 1:2], pch = 3,
+      cex = 3)
> identify(swiss.px[, 1:2], cex = 0.5)
```

kernel k-means

Partition data into k sets $S = \{S_1, S_2, \ldots, \S k\}$ so as to minimize the within-cluster sum of squares (WCSS) in kernel feature space:

$$argmin_S \sum_{i=1}^{k} \sum_{x_j \in S_i} |\Phi(x_j) - \Phi(\mu_i)|^2 \tag{13}$$

kernel k-means

```
> sc <- kkmeans(as.matrix(iris[,
+     -5]), kernel = "rbfdot", centers = 3)
> sc
> matchClasses(table(sc, iris[, 5]))
```

kernel k-means

```
> str <- stringdot(lenght = 4)
> K <- kernelMatrix(str, reuters)
> sc <- kkmeans(K, centers = 2)
> sc
> matchClasses(table(sc, rlabels))
```

Spectral Clustering in `kernlab`

- Embedding data points into the subspace of eigenvectors of a kernel matrix
- Embedded points clustered using k-means
- Better performance (embedded points form tighter clusters)
- Can deal with clusters that do not form convex regions

Spectral Clustering in R

```
> data(spirals)
> plot(spirals)
> sc <- specc(spirals, centers = 2)
> sc
> plot(spirals, col = sc)
```

Spectral Clustering

```
> sc <- specc(reuters, kernel = "stringdot",
+     kpar = list(length = 5), centers = 2)
> matchClasses(table(sc, rlabels))
> par(mfrow = c(1, 2))
> kpc <- kpca(reuters, kernel = "stringdot",
+     kpar = list(length = 5), features = 2)
> plot(rotated(kpc), col = as.integer(rlabels),
+     xlab = "1st Principal Component",
+     ylab = "2nd Principal Component")
> plot(rotated(kpc), col = as.integer(sc),
+     xlab = "1st Principal Component",
+     ylab = "2nd Principal Component")
```

Hierarchical Clustering

```
> data(state)
> h <- hclust(dist(state.x77), method = "single")
> plot(h)
```

Hierarchical Clustering

```
> pltree(diana(state.x77))
```