


WERKZEUGE DER MUSTERERKENNUNG UND DES MASCHINELLEN LERNENS

Vorlesung im Sommersemester 2017

Prof. E.G. Schukat-Talamazzini

Stand: 30. März 2017

Teil V

Einige Highlights aus der reichhaltigen
Werkzeugkiste der Sprache 

Implementationsschema für (etliche) -Modelle

Klasse: `mod` · Konstruktor: `mod()` · Methoden: `predict()`, `coef()`, ...

Standardschnittstelle

`mod[.fit] (x, y, ...)`

`x` Quellvariablen: matrix/data.frame
`y` Zielvariablen: matrix/vector
`grouping` statt 'y': factor

Formelschnittstelle

`mod (formula, data, ...)`

`formula` Modell `tgt ~ src`
`data` Datensatz: data.frame

... weitere Aufrufargumente:

`weights` Fehlergewicht: numeric
`subset` Datenauswahl: logical
`na.action` Defekte (NA): function

Vorhersagemethode

`predict (x, newdata, ...)`

`x, object` gefittetes Modell: `mod`

`newdata` Datensatz: data.frame

`type` $\left\{ \begin{array}{l} n, q\text{-Matrix: 'response' } \\ n, k\text{-Matrix: 'probs' } \\ \text{Faktor: 'class' } \\ n, p^*\text{-Matrix: 'terms' } \\ \text{Vektor: 'link' } \end{array} \right.$

Diverse Modellkomponenten

`coef(x)` gelernte Modellparameter
`fitted(x)` Vorhersage für Lerndaten
`residuals(x)` Abweichung dabei
`AIC(x)` Akaiikes Modellgütwert
`summary(x)` Modellüberblick
... einige weitere generische Methoden:
`print(x)`, `plot(x)`, `str(x)`, ...

Digitale Signalverarbeitung

Merkmalttransformation und Dimensionsreduktion

Numerische und ordinale Vorhersage

Klassifikation und überwachtes Lernen

Clusteranalyse und unüberwachtes Lernen

Digitale Signalverarbeitung

Theorie: $\tilde{f} \triangleq [\dots, f_{n-1}, f_n, f_{n+1}, \dots] \in \mathbb{R}^{\mathbb{Z}}$ · Praxis: $f \triangleq (f_1, \dots, f_N)^\top \in \mathbb{R}^N$

Repräsentation (Zeit)

Vektor **numeric**

Zeitreihe(n) **ts, mts**

LSI-Systeme $\tilde{f} \mapsto \tilde{h}$

Faltung $\tilde{h} = \tilde{f} \star \tilde{g}$

mit **Impulsantwort** $\tilde{g} \in \mathbb{R}^{\mathbb{Z}}$

ARMA-Systeme (p, q)

$$h_n = \underbrace{\sum_{j=0}^q b_j \cdot f_{n-j}}_{\text{moving average}} - \underbrace{\sum_{j=1}^p a_j \cdot h_{n-j}}_{\text{autoregressive}}$$

Repräsentation (Frequenz)

Vektor **complex**

Diskrete Fouriertransform.

$$F_\nu = \frac{1}{N} \cdot \sum_{n=0}^{N-1} f_n \cdot e^{-2\pi \frac{\nu n}{N}}$$

für $\nu = 0, 1, 2, \dots, N-1$ bzw. $N/2$

Faltungssätze (LSI, ARMA, ACF)

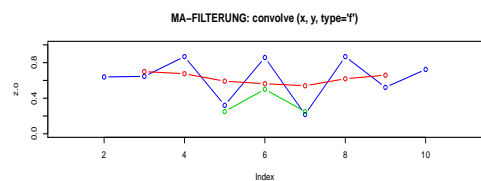
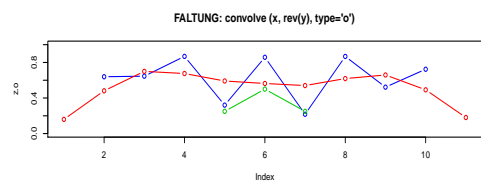
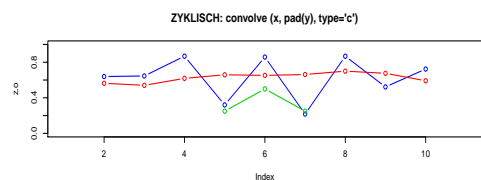
$$H_\nu = F_\nu \cdot G_\nu$$

$$H_\nu = F_\nu \cdot B_\nu / A_\nu$$

$$R_\nu^f = |F_\nu|^2 \quad \text{wegen } r^f = f \star \overleftarrow{f}$$

Faltungsoperation mit Hilfe der FFT

`stats::convolve(x, y, conj=TRUE, type=c('circular','open','filter'))`



Funktionsargumente

x Vektor Nr.1
y Vektor Nr.2
conj komplex konjugiert?
type Periode/Nullen
return Vektor $x \star \overleftarrow{y}$

Filtern versus Falten

Implement. Faltung mit `rev(y)`:

$$z_i \stackrel{\text{def}}{=} \sum_{j \in \mathbb{Z}} x_j \cdot y_{j-i+|y|}$$

'c' periodisch fortsetzen

$|x|$ Koeffizienten, $|x| = |y|$

'o' mit 0 auffüllen: **Faltmodus**

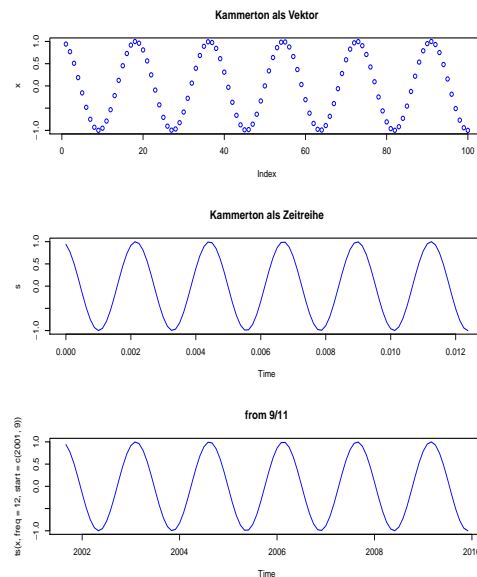
$|x| + |y| - 1$ Koeffizienten

'f' mit 0 auffüllen: **Filtermodus**

$|x| - |y| + 1$ Koeffizienten

'R'-Klassen für Zeitreihen: ts & mts

`stats::ts (data=NA, start=1, end=, frequency=1, deltat=1, class=, names=)`



Funktionsargumente

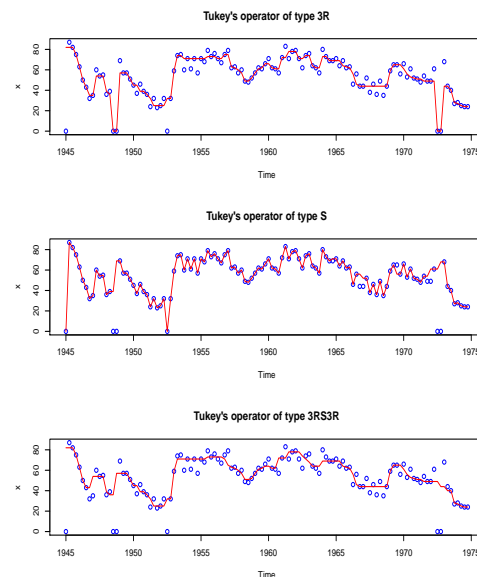
data Vektor/Matrix
start Index(-paar)
end Index(-paar)
frequency #Observ./Einheit
deltat (reziprok dazu)
names multiple Zeitreihen

R-Programmcode

```
layout(1:3)
sp <- 1:100/8000
x <- cos(2*pi*sp*440)
s <- ts(x, start=0, freq=8000)
plot(x, col="blue", main="Kammerton als Vektor")
plot(s, col="blue", main="Kammerton als Zeitreihe")
plot(ts(
  x, freq=12, start=c(2001,9)
), col="blue", main="from 9/11")
```

Tukeys nichtlineare Glättungsoperatoren

`stats::smooth(x, kind = c('3RS3R','3RSS','3RSR','3R','3','S'),
twiceit=FALSE, endrule=c('Tukey','copy'), do.ends=FALSE)`



Funktionsargumente

x Vektor oder Zeitreihe
kind Operatortypcode
twiceit doppelt gemoppelt
endrule Verhalten am Rand
return Vektor oder Zeitreihe

Typcode-Konventionen

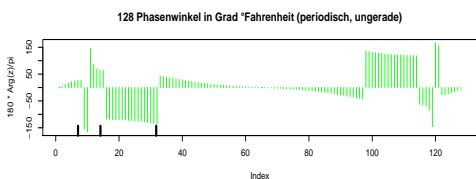
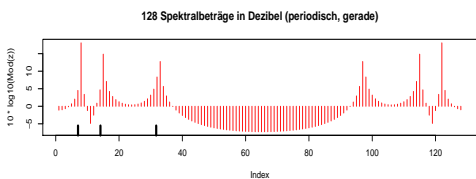
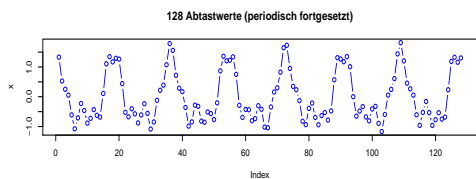
3: Median der Ordnung 3
S: Spalte alle 2/3-Läufe
R: Wiederhole bis Konvergenz

R-Programmcode

```
x <- presidents
x[is.na(x)] <- 0
for(w in c("3R", "S", "3RS3R")) {
  plot(x, col="blue", type="p")
  lines(smooth(x, w), col="red")
}
```

(Schnelle) Fouriertransformation DFT : $\mathbb{C}^N \rightarrow \mathbb{C}^N$

`stats::mvfft(z, inverse=FALSE)`



Funktionsargumente

z Vektor/Array über \mathbb{R}/\mathbb{C}
inverse rechne $N \cdot \text{DFT}^{-1}\{z\}$
return Vektor/Array über \mathbb{R}/\mathbb{C}

Schnelle diskrete FT

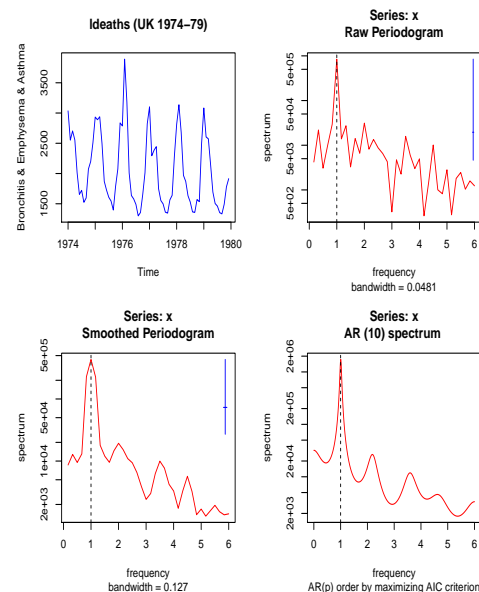
- FFT schnell für $N = 2^n$
- **fft**: D -dimensionale FT, $D \in \mathbb{N}$
- **mvfft**: $M \times$ eindimensionale FT

R-Programmcode

```
sp <- seq_len(N<-128) / (SF<-8000)
hertz <- c(440,883,1984)
x <- sapply(hertz, function(h)
  cos(2*pi*sp*h)) %*% (1/i:3)
z <- fft(x)
plot(x, col="blue", type="b")
plot(10*log10(Mod(z)), col="red", type="h")
rug(hertz/SF*N, tick=.1, lwd=3)
plot(180*Arg(z)/pi, col="green", type="h")
```

Berechnung und Darstellung der Spektraldichte

`stats::spectrum(x, ..., method=c('pgram','ar'))` ➔ `spec.pgram` & `spec.ar`



Funktionsargumente

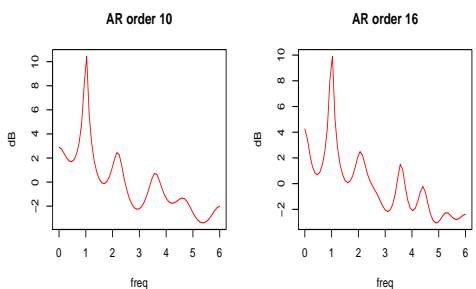
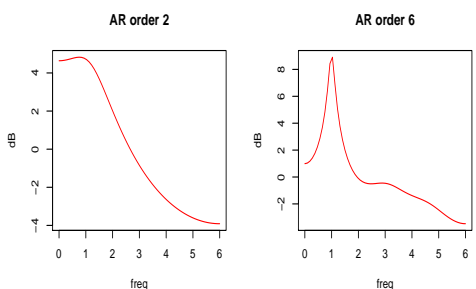
x Vektor oder Zeitreihe(n)
method Periodogramm/AR-Modell
taper=0.1 Ausblendung (cos)
pad=0 Nullanreicherung (prop.)
demean=FALSE $\hat{\mu}$ subtrahieren?
detrend=TRUE $\hat{\mu} + \hat{\alpha}t$ subtr.?
spans=NULL Daniell-Glättung
order=NULL AR-Ordnung/AIC
freq Frequenzstützstellen
spec Spektraldichten dazu
coh,phase Kohärenz/Phase mts

R-Programmcode

```
plot(ideaths, col="blue", ylab=NULL)
spectrum(ideaths, col="red")
spectrum(ideaths, spans=3, col="red")
spectrum(ideaths, method="ar", col="red")
```

Anpassung eines autoregressiven Modells $\text{AR}(p)$

`stats::ar(x, aic=TRUE, order.max=NULL, method='yw', ...)`



Funktionsargumente

x Vektor oder Zeitreihe(n)
aic Ordnung via AIC wählen?
order $\min(N-1, 10 \cdot \log_{10}(N))$
method $\begin{cases} \text{yule-walker} \\ \text{burg, ols, mle} \end{cases}$
return Objekt der Klasse `ar`

$\text{AR}(p)$ -Modell

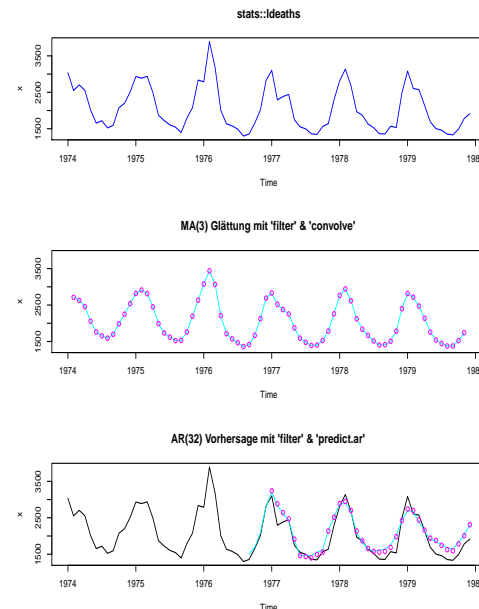
$$x_t - \mu = \tilde{x}_t = e_t + \sum_{j=1}^p a_j \cdot \tilde{x}_{t-j}$$

R-Programmcode

```
M <- 128
freq <- seq(0, frequency(ideaths)/2, len=1+M/2)
sapply(c(2,6,10,16), function(p) {
  a <- c(1, rep(0,M-1))
  a[1:p] <- ar(ideaths, aic=FALSE, order=p)$ar
  dB <- 10*log10(1/Mod(fft(a)))[1+0:(M/2)]
  plot(freq, dB, type="l", col="red")
})
```

Filterung für Systeme $\text{MA}(q)$ und $\text{AR}(p)$

`filter(x, filter, method='convolution', sides=2, circular=FALSE, init)`



Funktionsargumente

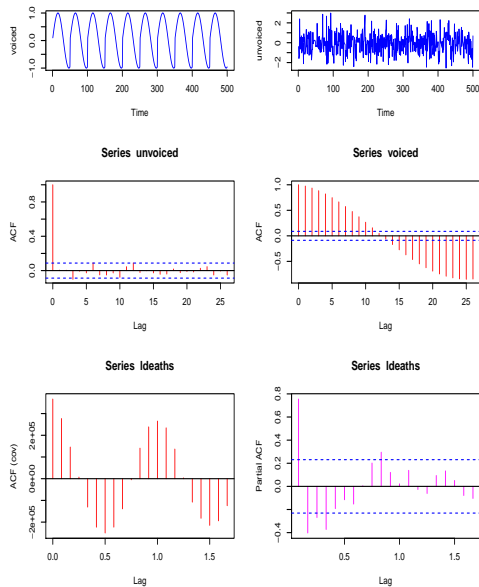
x Vektor oder Zeitreihe(n)
filter Koeffizienten (rückwärts)
method convolution/recurse
sides 1=causal, 2=center; MA
circular periodisch forts.; MA
init Burn-in (rückwärts); AR
return Zeitreihe(n)

R-Programmcode

```
plot(x, col="blue")
g <- c(1,2,1)/4
lines(filter(x, g, "c"), col="cyan")
points(time(x), c(NA, convolve(
  x, g, type="filter"), NA), col="magenta")
p <- 32
a <- ar(x, aic=FALSE, order=p)
x.1 <- window(x, end=c(1976,12))
x.2 <- predict(a, x.1, 3*12)$pred
e <- window(x, start=c(1974,p+1))
e[] <- rnorm(length(x)-p)
y <- filter(e, a$ar, "r", init=x[p:1]-mean(x))
lines(y+mean(x), col="cyan")
points(x.2, col="magenta")
```

Autokorrelationsfunktion

```
stats::acf (x, lag.max=NULL, type='corr', plot=TRUE, demean=TRUE, ...)
```



Funktionsargumente

x Vektor/Matrix oder Zeitreihe(n)
lag.max $\min(N-1, 10 \cdot \log_{10}(N/M))$
type correlation/covariance/partial
... für den plot-Aufruf
return Objekt der Klasse acf

Autokorrelation (unnormiert)

$$r_\ell \stackrel{\text{def}}{=} \frac{1}{n-m} \cdot \sum_{j=m}^{n-1} x_j \cdot x_{j-\ell}$$

R-Programmcode

```
voiced <- rep (sin (1:50/10), 10)
unvoiced <- rnorm (500)
plot.ts (voiced, col="blue")
plot.ts (unvoiced, col="red")
acf (unvoiced, col="red")
acf (voiced, col="red")
acf (Ideaths, lag=20, "covar", col="red")
pacf (Ideaths, lag=20, col="magenta")
```

Digitale Signalverarbeitung

MerkmalsTransformation und Dimensionsreduktion

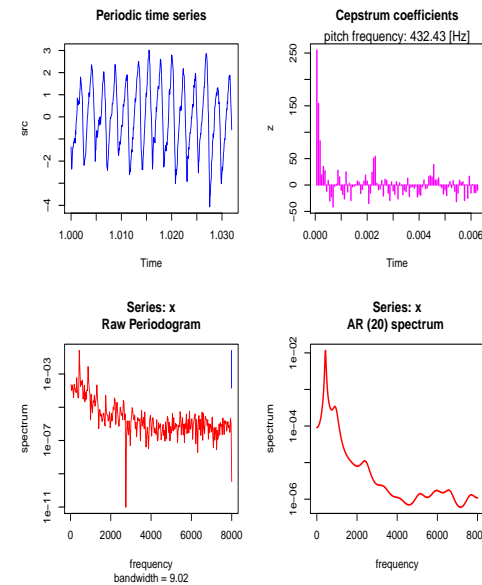
Numerische und ordinale Vorhersage

Klassifikation und überwachtes Lernen

Clusteranalyse und unbeaufsichtigtes Lernen

Beispiel „Grundfrequenzdetektion“

```
myutils::cepstrum (x, size=length(x)/2, k.att=0, do.pitch=0.1)
```



Funktionsargumente

x Zeitreihenobjekt (ts)
size Anzahl Cepstrumkoeffizienten
k.att harmonische Verstärkung
do.pitch Strahlkonstante für F_0
return Objekt der Klasse ts

R-Programmcode

```
SF <- 16000; N <- 512; f0 <- 440
src <- (1:N)%%(SF/f0)/(SF/f0)
src <- ts (src - rnorm (
  src, .5, sd=.2), freq=SF)
src <- filter (src,
  sort(runif(11)), circ=TRUE)
z <- cepstrum (src, size=100)
pfr <- attr (z, "pitch")$Hertz
pfr <- pfr[80<pfr&pfr<600]
mtext (paste (
  "pitch frequency:",
  round (pfr[1], 2), "[Hz]"))
```

Transformation numerischer Merkmalsräume

$$\mathbf{x} = (\xi_1, \dots, \xi_N)^\top \mapsto \mathbf{y} = (\eta_1, \dots, \eta_M)^\top$$

GEGEBEN:

Ein N -dimensionaler Datensatz
 $\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathbb{R}^N$ von
 Merkmalsvektoren für die Objekte
 $\phi_1, \dots, \phi_T \in \Omega$

GESUCHT:

Ein Datensatz M -dimensionaler
 Merkmalsvektoren $\mathbf{y}_1, \dots, \mathbf{y}_T \in \mathbb{R}^M$

GÜTEKRITERIUM:

Kleinstmöglicher Informationsverlust
 der \mathbf{y}_t gegenüber \mathbf{x}_t (trotz
 Dimensionsreduktion $M \ll N$)

Data Mining / EDA

Visualisierung H.D. Daten
 $(M \in \{1, 2, 3\})$

Musternalyse

Unterdrückung/Hervorhebung
 störender/nützlicher Anteile

Explizite Abbildung

Finde Berechnungsvorschrift
 $f: \mathbb{R}^N \rightarrow \mathbb{R}^M$

Implizite Abbildung

Finde Repräsentantenfolge
 $\mathbf{y}_1, \dots, \mathbf{y}_T \in \mathbb{R}^M$

Transformationsverfahren

kardinal/relational · explizit/implizit

Numerische Daten

$\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathbb{R}^N$

linear

PCA, FA, ICA

konvex

NMF

nichtlinear

SOM, AANN

nichtparametrisch

PSA (Hauptflächen)

diskriminativ

LDA, PLS

Relationale Daten

- gemischte Skalen
- Abstand/Ähnlichkeit
- Adjazenz/Nachbarschaft

metrisch

MDS (Sammon)

Skalarprodukt

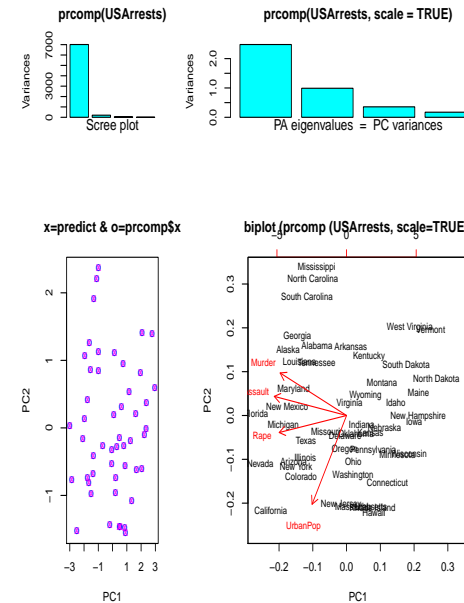
Kernel-PCA

topologisch

geodätische PCA/MDS

Principal Component Analysis (via 'svd')

`stats::prcomp(x, retx=TRUE, center=TRUE, scale.=FALSE, tol=NULL, ...)`



Funktionsargumente

x Datensatz: matrix, data.frame
retx liefere die PC von x
center Daten zentrieren ($\mu_n = 0$)
scale. Daten skalieren ($\sigma_n = 1$)
tol Abrissparameter für $\hat{\sigma}_n / \hat{\sigma}_{\max}$
return Objekt der Klasse prcomp

Hauptachsen/komponenten

$$\mathbf{X} = \mathbf{V} \cdot \mathbf{D} \cdot \mathbf{U}^T \Rightarrow \begin{cases} \text{PA: } \mathbf{U} \\ \text{PC: } \mathbf{V} \cdot \mathbf{D} \end{cases}$$

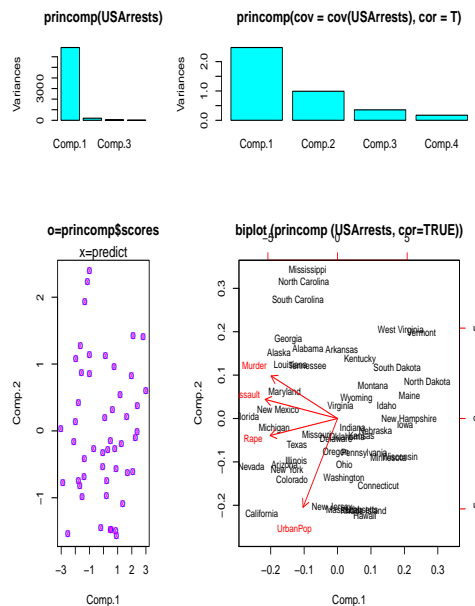
R-Programmcode

```
plot(prcomp(USArrests), col="cyan")
plot(prcomp(USArrests, scale=TRUE), col="cyan")
biplot(prcomp(USArrests, scale=T), color="blue")

o <- prcomp(formula=".", data=USArrests, scale=T)
plot(o$x[,1:2], pch='o', col="blue")
pc <- predict(o, USArrests)
points(pc[,1:2], pch='x', col="magenta")
```

Principal Component Analysis (via 'eigen')

`stats::princomp(x, cor=FALSE, scores=TRUE, covmat=NULL, ...)`



Funktionsargumente

x Datensatz: matrix, data.frame
cor correlation/covariance?
scores liefere die PC von x
covmat Matrix, z.B. cov.*(x)
return Objekt d. Klasse princomp

Hauptachsen/komponenten

$$\mathbf{S} = \mathbf{U} \cdot \mathbf{D}^2 \cdot \mathbf{U}^T \Rightarrow \begin{cases} \text{PA: } \mathbf{U} \\ \text{PC: } \mathbf{X} \cdot \mathbf{U} \end{cases}$$

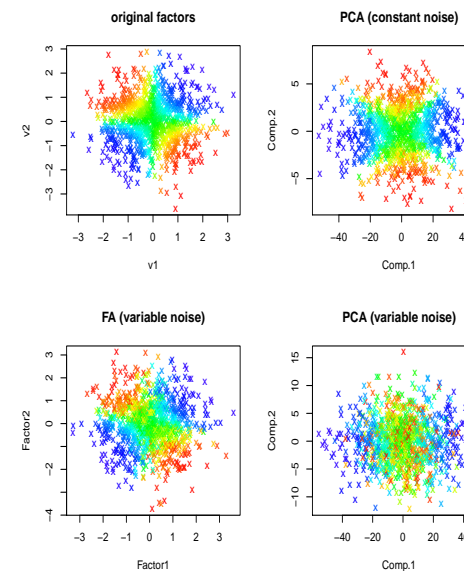
R-Programmcode

```
plot(princomp(USArrests), col="cyan")
plot(princomp(cov=cov(USArrests), cor=T), col="cyan")
biplot(princomp(USArrests, cor=T), color="blue")

o <- princomp(form=".", USArrests, cor=T)
plot(o$scores[,1:2], pch='o', col="blue")
pc <- predict(o, USArrests)
points(pc[,1:2], pch='x', col="magenta")
```

Maximum-Likelihood Faktoranalyse

`stats::factanal(x, factors, covmat=NULL, n.obs=NA, start=NULL, scores=c('none','regression','Bartlett'), rotation='varimax')`



Funktionsargumente

x Matrix, Datensatz, Formel
factors Anzahl Ladungsvektoren
covmat Kovarianz (zzgl. n.obs)
start initiale Kommunalitäten
rotation Fkt. Faktordrehung
return Klasse factanal

Faktoranalysemodell

$$\mathbf{S} = \mathbf{A} \cdot \mathbf{A}^T + \mathbf{R}$$

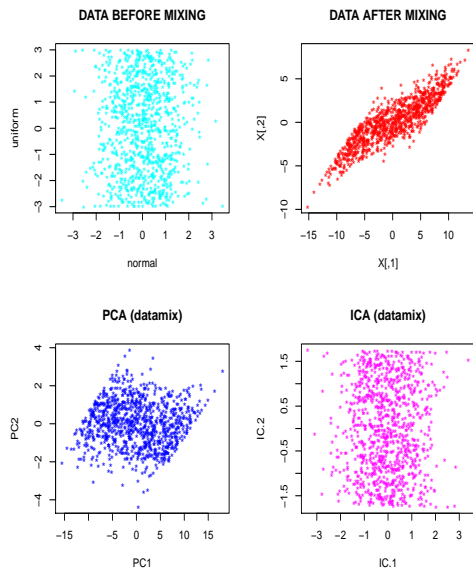
$\mathbf{A} \in \mathbb{R}^{N \times M}$ und $\mathbf{R} = \text{diag}(\mathbf{r}_1, \dots, \mathbf{r}_N)$

R-Programmcode

```
plot(v1, v2, col=cop, main="original factors")
# X <- DATA + HOMOSCEDASTIC NOISE
plot(princomp(X)$scores[,1:2], col=cop)
# X <- DATA + HETEROSCEDASTIC NOISE
plot(factanal(X, 2, scores="B")$scores, col=cop)
plot(princomp(X)$scores[,1:2], col=cop)
```

Independent Component Analysis

```
ICS::ics (X, S1=cov, S2=cov4, S1args=list(), S2args=list(),
          stdB=c('Z','B'), stdKurt=TRUE)
```



Funktionsargumente

X Matrix oder Dataframe
S1 Scattermatrix für 'X'
S2 (matrix oder function)
S[12]args Argumente zu S1, S2
stdB Kriterium zur A^{-1} -Auswahl
stdKurt Scatter/Formmatrizen?
return Objekt der Klasse ics

Quellentrennungsmodell

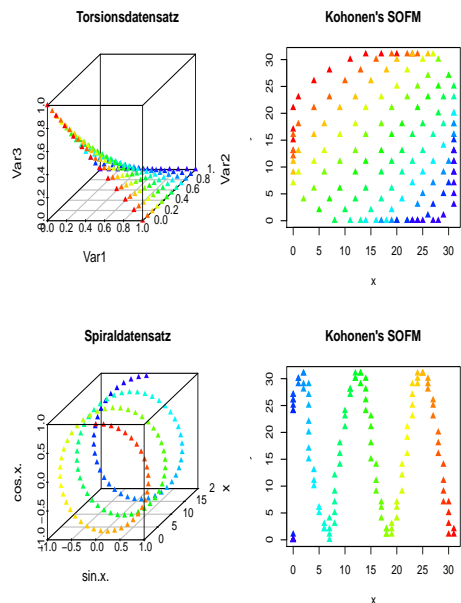
$X = V \cdot A^T$ bzw. $V = X \cdot A^{-T}$

R-Programmcode

```
X <- cbind (normal=rnorm(n),
            uniform=runif(n,-3,+3))
A <- rbind (c(2,3), c(2,1))
plot (X, col="cyan")
X <- X %*% t(A)
plot (X, col="red")
plot (prcomp(X)$x, col="blue")
plot (ics(X)$Scores, col="magenta")
```

Kohonen's Self-Organizing Map

```
som::som (data, xdim, ydim, init='l', neigh='g', topol='r', ...)
```



Funktionsargumente

x Matrix oder Dataframe
xdim, ydim Gitterdimensionen
init linear/sample/random
neigh gaussian/bubble
topol rect/hexa
return Objekt der Klasse som
som\$visual Dataframe (x, y, ε)

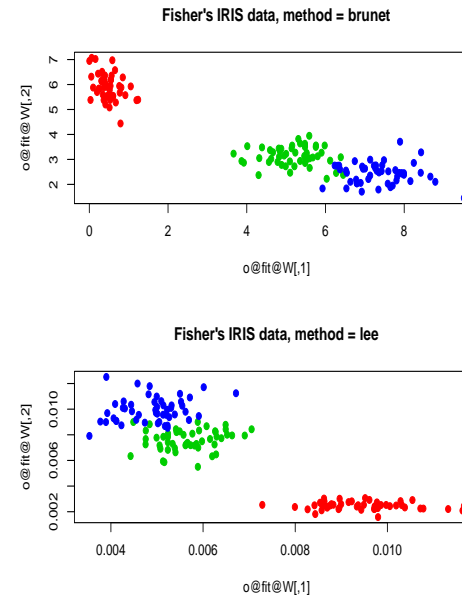
R-Programmcode

```
xy <- as.matrix (expand.grid(x,x))
X <- data.frame (xy,
                 Var3=(1-xy[,1])*(1-xy[,2]))
scatterplot3d (X, color=co)
plot (som(X,k,k)$vis[1:2], col=co)
```

```
x <- seq (0, xmax, length=n)
X <- data.frame (sin(x), x, cos(x))
scatterplot3d (X, color=co)
plot (som(X,k,k)$vis[1:2], col=co)
```

Nichtnegative Matrixfaktorisierung

```
NMF::nmf (x, rank, method="brunet", seed, ...)
```



Funktionsargumente

x Matrix oder Dataframe
rank Anzahl Ladungsvektoren
method Iterationsschema/Zielfkt.
seed Startkonfigurationsauswahl
return S4-Klasse NMFfit

Konvexe Faktorisierung

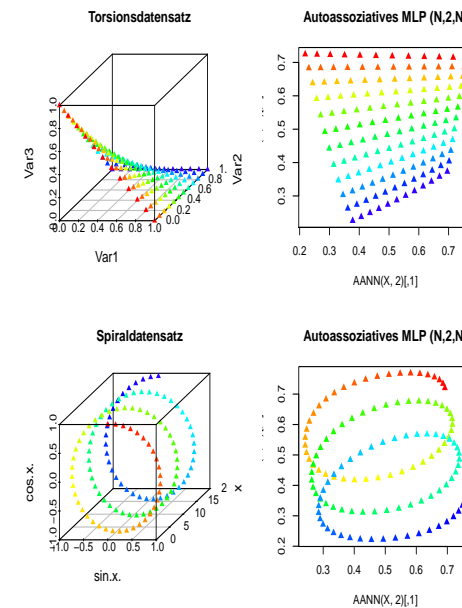
$$X \approx V \cdot A^T \quad \text{minim.} \quad \left\{ \begin{array}{l} \mathcal{D}(X \| Y) \\ \|X - Y\|_F^2 \end{array} \right\}$$

R-Programmcode

```
require (NMF)
for (m in c("brunet","lee")) {
  o <- nmf (iris[1:4],
            rank=2, method=m)
  plot (o@fit@W, col=1+
        unclass(iris$Species))
}
```

Autoassoziatives Mehrschichtenperzeptron

```
neuralnet::neuralnet (formula, data, hidden=1, algorithm='rprop+', ...)
```



Funktionsargumente

formula Output/Input-Variablen
data Datensatz
hidden Zwischenschichtengrößen
algorithm $\left\{ \begin{array}{l} \text{backprop, sag, slr} \\ \text{rprop+, rprop-} \end{array} \right\}$
return Objekt der Klasse nn

R-Programmcode

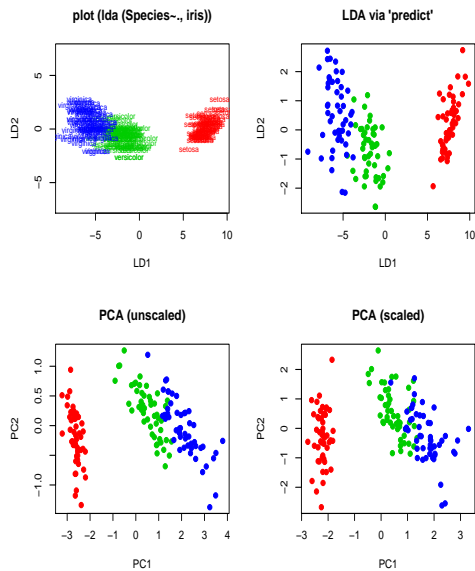
```
AANN <-function (X, k=2, sd=1/3, ...)
{
  require (neuralnet)
  X <- scale(X) * sd

  vn <- colnames(X)
  iv <- paste (vn, collapse="+")
  ov <- paste (paste (
    vn, 1, sep="."), collapse="+")
  fo <- formula (paste (iv, "", ov))

  o <- neuralnet (formula=fo,
                  data.frame(X,X), hid=k, ...)
  compute(o,X)$neurons[[2]][,-1]
}
```


Lineare Diskriminanzanalyse

`MASS::lda (x, grouping, prior=, tol=1.0e-4, method, ...)`



Funktionsargumente

x Matrix oder Dataframe
grouping Klasseninfo (factor)
prior Klassenwahrscheinlichkeiten
method $\begin{cases} \text{moment} \\ \text{mle, mve, t} \end{cases}$
return Objekt der Klasse lda

Fisherdiskriminanten

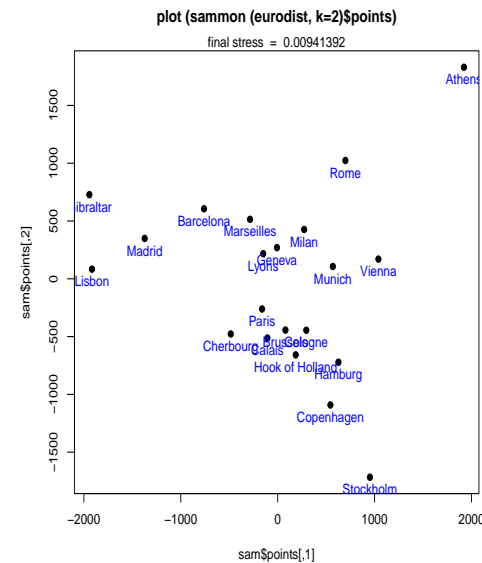
$(K-1)$ Hauptträgheitsachsen von $Q_{\text{Kitano}} := S_W^{-1} S_B$

R-Programmcode

```
require (MASS)
ld <- lda (iris[-5], iris[[5]])
pcu <- prcomp (iris[-5], scale=F)
pcs <- prcomp (iris[-5], scale=T)
plot (ld, col=co)
plot (predict(ld)$x[,1:2], col=co)
plot (pcu$x[,1:2], col=co)
```

Mehrdimensionale Skalierung (Sammon)

`MASS::sammon (d, y=cmdscale(d,k), k=2, niter=100, magic=0.2, tol=1e-4)`



Funktionsargumente

d Distanzmatrix (dist, matrix)
y Startkonfiguration (k Spalten)
k Zieldimension
niter maximale Schrittzahl
points Zielkonfig. (k Spalten)
stress erzielter Schlusswert

Gütekriterium Stress

$$S_q(D, D^*) \stackrel{\text{def}}{=} \sum_{i < j} \frac{(d_{ij}^* - d_{ij})^2}{d_{ij}^q}$$

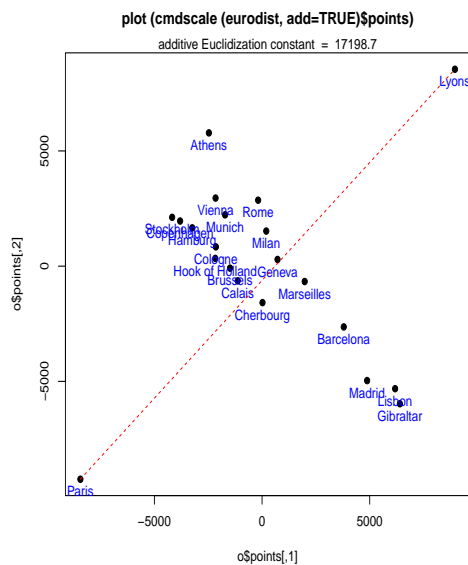
$q = 1, 2, 0$ \star Sammon, relativ, absolut

R-Programmcode

```
require (MASS)
sam <- sammon (eurodist)
plot (sam$points, pch=19)
text (sam$points,
      labels=labels (eurodist),
      pos=1, col="blue")
```

Mehrdimensionale Skalierung (klassisch: duale PCA)

`stats::cmdscale (d, k=2, eig=FALSE, add=FALSE, x.ret=FALSE)`



Funktionsargumente

d Distanzmatrix (dist, matrix)
k Zieldimension
eig Eigenwerte abliefern?
add Euklid via c^* forcieren?
x.ret $\tilde{D} = HDH$ abliefern?
points Zielkonfig. (k Spalten)
eig, x, ac λ, \tilde{D}, c^*

Distanzen \rightsquigarrow Produkt

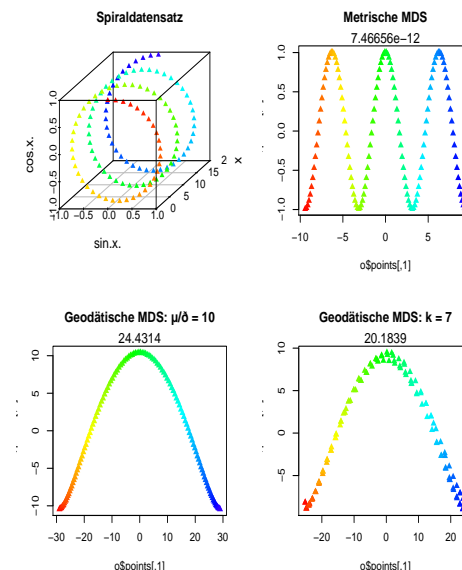
$$\|x - y\|^2 = \|x\|^2 - 2 \cdot x^T y + \|y\|^2$$

R-Programmcode

```
eurodist[179] <- eurodist[179]+8000 # Par
o <- cmdscale (eurodist, add=TRUE)
plot (o$points, pch=19)
text (o$points,
      labels=labels (eurodist),
      pos=1, col="blue")
lines (o$points[c("Paris", "Lyons"),], col
```

Geodätische Skalierung

`floyd_warshall (dist, makesym=c('none', 'first', 'second'), way.off=Inf)`



Funktionsargumente

dist Distanzen (dist, matrix)
makesym Symmetrisierung?
way.off D_{ij} -Wert für $i \not\sim j$
return min. Pfadkosten (matrix)

Floyd-Warshall-Algorithm.

D_{ij} = minimale Distanzsumme eines Weges von i nach j

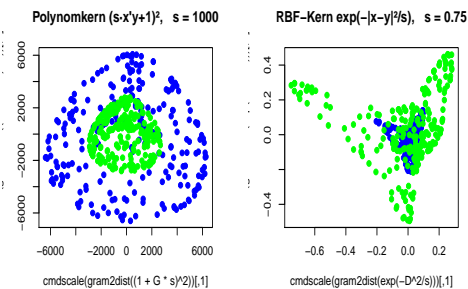
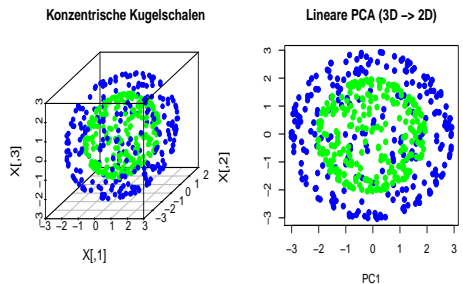
R-Programmcode

```
D <- as.matrix (dist(X))
D <- ifelse (D < mean(D)/s, 1, NA)
diag(D) <- 0
D <- floyd_warshall (D)
```

```
D <- as.matrix (dist(X))
D <- apply (D, 1, rank)
D <- ifelse (D <= 1+k, 1, +Inf)
D <- floyd_warshall (D, "first")
```

Nichtlineare PCA

Originaldaten \Rightarrow [Termexpansion] \Rightarrow Distanzmatrix \Rightarrow cmdscale()



Der Kernel-Trick

Originalpunkte $\mathbf{x}_1, \dots, \mathbf{x}_T$

nichtlin. Expansion $\phi \mathbf{x}_1, \dots, \phi \mathbf{x}_T$

$$G_{st} = \langle \phi \mathbf{x}_s, \phi \mathbf{x}_t \rangle = K(\mathbf{x}_s, \mathbf{x}_t)$$

$$D_{st} = \sqrt{G_{ss} - 2 \cdot G_{st} + G_{tt}}$$

\leadsto keine explizite Expansion!

R-Programmcode

```
gram2dist <- function (G) {
  D <- -2*G
  D <- sweep (D, 1, diag(G), "+")
  D <- sweep (D, 2, diag(G), "+")
  sqrt (D)
}
scatterplot3d (X, color=co)
plot (prcomp (X, scale=FALSE)$x, col=co)
G <- tcrossprod (X)
D <- as.matrix (dist (X))
plot (cmdscale (
  gram2dist ((1+G*s)^2)), col=co)
plot (cmdscale (
  gram2dist (exp(-D^2/s))), col=co)
```

Regression ist überwachtes Lernen

Numerische Regression: $\mathbf{x} = (\xi_1, \dots, \xi_N)^\top \mapsto \mathbf{y} = (\eta_1, \dots, \eta_M)^\top$

GEGEBEN:

Ein N -dimensionaler
Quelldatensatz $\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathbb{R}^N$
und ein M -dimensionaler
Zielatensatz $\mathbf{y}_1, \dots, \mathbf{y}_T \in \mathbb{R}^M$

GESUCHT:

Eine Abbildungsvorschrift
 $f: \mathbb{R}^N \rightarrow \mathbb{R}^M$ mit

$$\mathbf{y}_t \approx \hat{\mathbf{y}}_t \stackrel{\text{def}}{=} f(\mathbf{x}_t)$$

Gütekriterium

- Methode kleinster Quadrate
- allgemeiner Kostenansatz
- max/mean posteriori Verteilung

Dimensionalität

- univariat $N = M = 1$
- multivariat $N > 1$
- multipel $N, M > 1$

Abbildungsfamilie

- linear/affin $\hat{\mathbf{y}} = \mathbf{a}_0 + \mathbf{a}^\top \mathbf{x}$
- Kerneltrick $\hat{\mathbf{y}} = \mathbf{w}^\top \phi(\mathbf{x})$
- nichtlinear $\hat{\mathbf{y}} = f(\mathbf{x}|\boldsymbol{\theta})$
- nichtparam. $\hat{\mathbf{y}} = f(\mathbf{x} | \mathbf{X}, \mathbf{Y})$

Zielskalentyp

- numerisch *Ausgleichsrechnung*
- nominal *Klassifikation*
- ordinal *Präferenz*

Digitale Signalverarbeitung

MerkmalsTransformation und Dimensionsreduktion

Numerische und ordinale Vorhersage

Klassifikation und überwachtes Lernen

Clusteranalyse und unüberwachtes Lernen

Werkzeuge zur Ausgleichsrechnung

Modellieren in 'R': N/L Modelle · Klassifikatoren

Lineare Modelle

(linear in den Koeffizienten)

`stats::lm (formula, data, subset, weights, ..., contrasts)`

Verallgemeinert lineare Modelle

(link function: spezielle Fehlerverteilung)

`stats::glm (formula, family, data, weights, subset, ...)`

Nichtlineare Modelle

(LSE-Parameter, nichtlinearer Fkt.Prototyp)

`stats::nls (formula, data, start, control, algorithm, ...)`

Nichtparametrische Modelle

(lokale Polynomapproximation, $N \leq 4$)

`stats::loess (formula, data, ..., span=.75, degree=2, ...)`

Konnektionistische Modelle

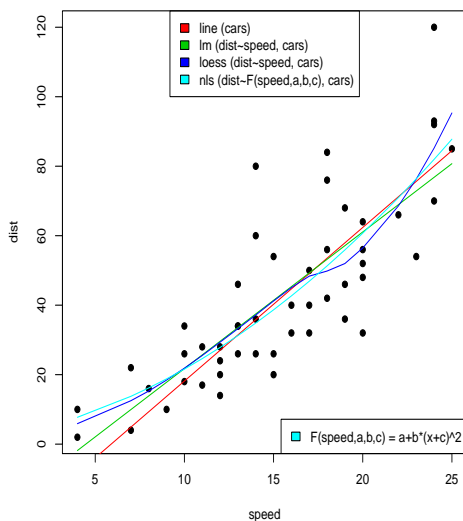
(MLP mit einer verborgenen Neuronenschicht)

`nnet::nnet (formula, data, weights, ...,`
`size, Wts, mask, $\begin{cases} \text{linout} \\ \text{entropy} \\ \text{softmax} \\ \text{censored} \end{cases} = \text{F}, \text{skip}=\text{F}, \text{MaxNWts}=1000, \dots)$`

Robuste Ausgleichsgerade (Tukey, EDA 1977)

`stats::line (x, y)` (alternativ zu `lm`, `loess`, `nls`)

Prädiktion 'cars\$dist' aus 'cars\$speed'



Funktionsargumente

`x, y` Quell- und Zielvektor
`return` Klasse `tukeyline`

R-Programmcode

```
plot (cars, pch=19)
oln <- line (cars)
olm <- lm (dist~speed, cars)
olo <- loess (dist~speed, cars)
F <- function (x,a,b,c) a+b*(x+c)^2
ols <- nls (dist~F(speed,a,b,c),
  cars, start=list(a=1,b=1,c=1))
lines (cars$speed,
  fitted (oln), col=2)
lines (cars$speed,
  fitted (olm), col=3)
lines (cars$speed,
  fitted (olo), col=4)
lines (cars$speed,
  fitted (ols), col=5)
```

Naiver Bayesprädiktor

`myutils::nbp (x, y, adjust=TRUE)`

Funktionsargumente

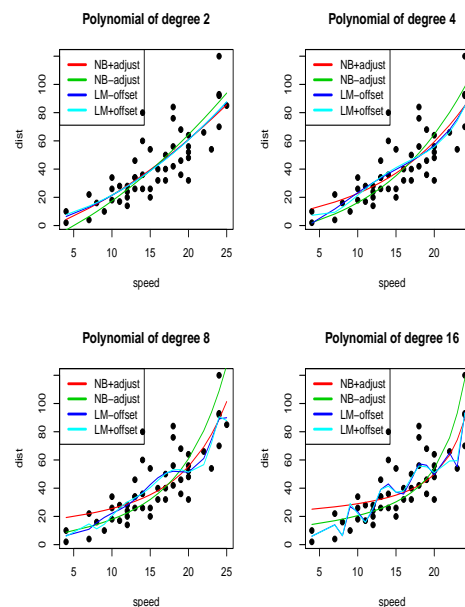
`x` Matrix der Quellvariablen
`y` Vektor der Zielvariable
`adjust` (lin.) Residualkompens.
`return` Objekt der Klasse `nbp`

Sternennetz-Regression

$$f(\mathbf{x}, y) = \mathcal{N}(y; \theta_0) \cdot \prod_{n=1}^N \mathcal{N}(x_n | y; \theta_n)$$

R-Programmcode

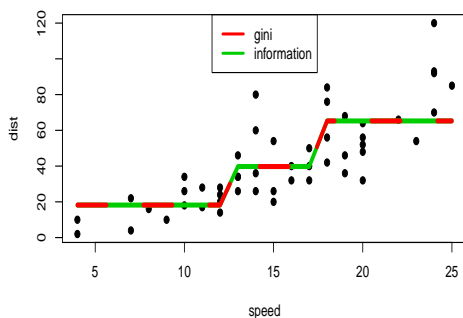
```
require (myutils)
for(p in 2^(1:4)) {
  plot (cars, pch=19)
  X <- outer (cars$speed, 1:p, "^")
  y <- predict (nbp(X,cars$dist), X)
  y <- lm.fit (X, cars$dist)$fitted
  # dto. mit 'cbind(1,X)' statt 'X'
}
```



Statistische Regressionsbäume

`rpart::rpart (formula, data, method, parms, ...)`

`cars$speed ==> cars$dist`



Funktionsargumente

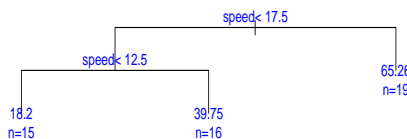
`formula, data` (Formelinterface)
`method` `anova`, `poisson`, `class`, `exp`
`parms` Splittingparameter
`return` Klasse `rpart`

Entscheidungsbäume

`numeric~.` `method='anova'`

R-Programmcode

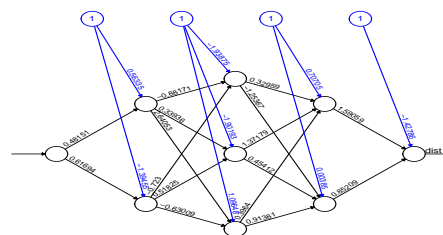
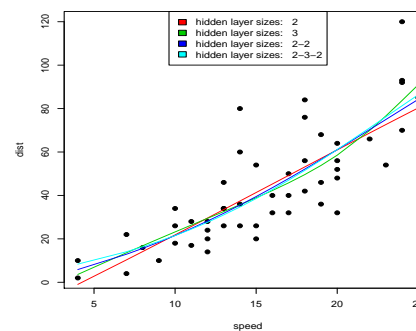
```
require (rpart)
plot (cars, pch=19)
for (s in c("gini", "information")){
  o <- rpart (dist~., cars,
    parms=list(split=s))
  lines (cars$speed, predict(o), # etc.
  )
}
plot (o, uni=T, bran=1, comp=T)
text (o, use.n=TRUE, col="blue")
```



Mehrschichtenperzeptron (MLP)

`neuralnet::neuralnet (formula, data, hidden=1, algorithm='rprop+', ...)`

MLP-Prädiktion 'cars\$dist' aus 'cars\$speed'



Funktionsargumente

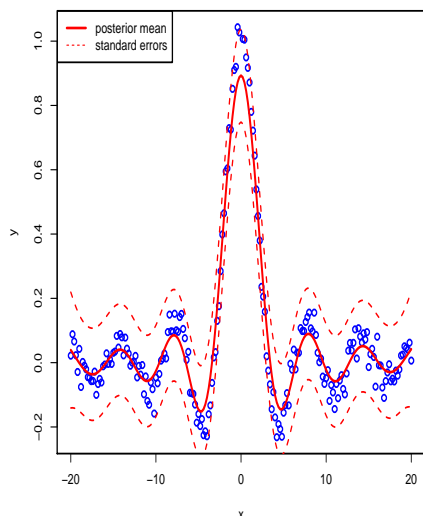
`formula` Output/Input-Variablen
`data` Datensatz
`hidden` Zwischenschichtengrößen
`algorithm` { `backprop`, `sag`, `slr`, `rprop+`, `rprop-` }
`return` Objekt der Klasse `nn`

R-Programmcode

```
require (neuralnet)
s <- 100
H <- list (2, 3, c(2,2), c(2,3,2))
plot (cars, pch=19)
for (j in seq_along(H)) {
  o <- neuralnet (
    dist~speed, cars/s, H[[j]])
  y <- compute(o, cars$speed/s)$net
  lines (cars$speed, y*s, col=1+j)
}
plot (o, rep="best")
```

Gaußprozesse (GP)

`kernlab::gausspr (formula, data, scaled=TRUE, kernel='rbfdot', ...)`



Funktionsargumente

formula Output/Input-Variablen
data Datensatz
type Regression/Klassifikation
kpar Welche Kernparameter?
variance.model Error Bars?
return Objekt der Klasse `gausspr`

R-Programmcode

```
require(kernlab)
x <- seq(-20,20,len=n)
y <- sin(x)/x + rnorm(n,sd=0.03)
o <- gausspr(x, y,
  variance.model=TRUE)
y.m <- predict(o, x, "response")
y.s <- predict(o, x, "sdeviation")
plot(x, y, pch=21, col="blue")
lines(x, y.m, lwd=3, col="red")
lines(x, y.m+y.s, lty=2, col="red")
lines(x, y.m-y.s, lty=2, col="red")
```

Digitale Signalverarbeitung

MerkmalsTransformation und Dimensionsreduktion

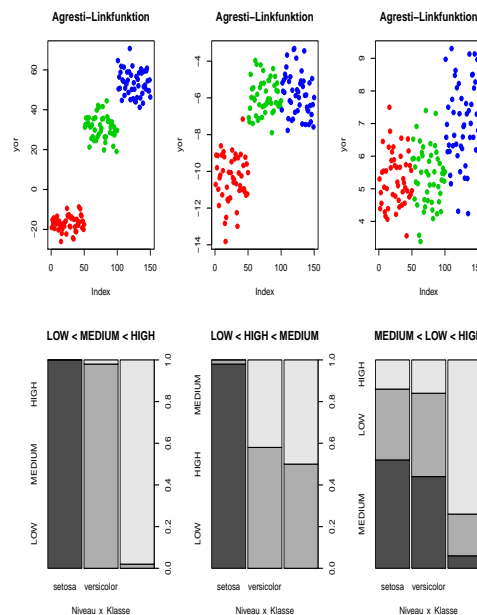
Numerische und ordinale Vorhersage

Klassifikation und überwachtes Lernen

Clusteranalyse und unbeaufsichtigtes Lernen

Ordinale Regression (Agresti 2002)

`MASS::polr (formula, data, weights, start, ..., method='logistic')`



Funktionsargumente

start Parameter `c(coeff,zeta)`
method `{logistic, probit, cloglog, cauchit}`
return Objekt der Klasse `polr`

Proportional-Odds LR

Gelenkfunktion für CPF: $\zeta_0, \dots, \zeta_\ell$

R-Programmcode

```
require(MASS)
for (P in list(c(1,2,3),c(1,3,2),c(2,1,3))) {
  osiris <- data.frame(iris[1:4],
    AR=ordered(P[iris$Species],
      labels=c("LOW","MEDIUM","HIGH")[P]))
  oor <- polr(AR ~ ., osiris,
    start=c(c(+1,+1,-1,-1),0,1))
  yor <- as.matrix(osiris[-5]) %*% coef(oor)
  plot(yor, col=co)
  plot(iris$Species, predict(oor), ylab="")
}
```

Flächenverhältnis $\frac{\text{Sepal.Length} \cdot \text{Sepal.Width}}{\text{Petal.Length} \cdot \text{Petal.Width}}$

Klassifikation ist Regression auf nominales Ziel

$$\mathbf{x} = (\xi_1, \dots, \xi_N)^T \in \Omega_1 \times \dots \times \Omega_N \mapsto y \in \{1, 2, \dots, K\}$$

GEGEBEN:

Ein Quelldatensatz
 $\mathbf{x}_1, \dots, \mathbf{x}_T \in \Omega$ und eine
 Klassenetikettierung
 $y_1, \dots, y_T \in \{1, \dots, K\}$

GESUCHT:

Eine Entscheidungsfunktion
 $f: \Omega \rightarrow \{1, \dots, K\}$ zur
 Klassenvorhersage
 $\hat{y}_t \stackrel{\text{def}}{=} f(\mathbf{x}_t) \stackrel{?}{=} y_t$

Gütekriterium

- Fehlerrate (Lerndaten)
- Fehlerwahrsch. (Testdaten)
- Risiko (Fehlklassif.kosten)

Bayesregel

Theoretisch optimale

Entscheidungsvorschrift:

$$y^*(\mathbf{x}) \stackrel{\text{def}}{=} \underset{\kappa=1..K}{\operatorname{argmax}} P(\mathbb{Y} = \kappa \mid \mathbb{X} = \mathbf{x})$$

Klassifikatortyp

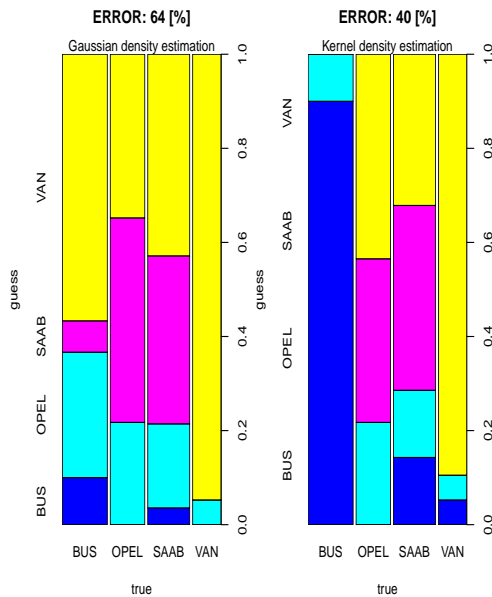
- statistisch $W.\text{modelle } f(\mathbf{x}|\kappa)$
- diskriminativ $W.\text{modell } P(\kappa|\mathbf{x})$
- parametrisch $NVK, \text{PolyK}, \text{MLP}$
- nichtparametrisch $KNN\text{-Regel}$
- semiparametrisch $SCT, \text{SVM}, \text{GMK}$

Quellskalentyp

- numerisch
- nominal
- gemischt, metrisch, Kernel ...

Naiver Bayesklassifikator

`klaR::NaiveBayes (x, grouping, prior, usekernel=FALSE, fL=0, ...)`



Funktionsargumente

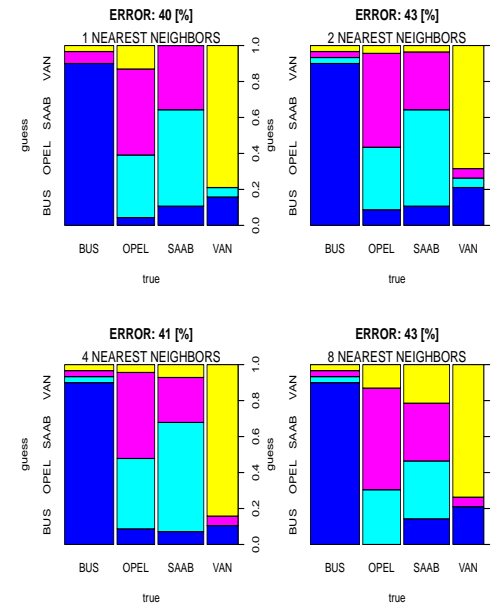
x Matrix oder Dataframe
grouping Klasseninfo (factor)
formula, data (Formelinterface)
usekernel density-Aufruf ?
fL Faktor Laplacekorrektur
return Klasse NaiveBayes

R-Programmcode

```
require (klaR)
df <- lapply (load (
  "~/data/set/statlog/vehicle/rda"),
  get)
true <- df[[2]]$CLASS
o <- NaiveBayes (CLASS~., df[[1]])
guess <- predict (o, df[[2]])$class
plot (data.frame (true, guess),
  col=3+1:4)
o <- NaiveBayes (CLASS~., df[[1]],
  usekernel=TRUE)
```

k-Nächste-Nachbarn Klassifikator

`klaR::sknn (x, grouping, kn=3, gamma=0)`



Funktionsargumente

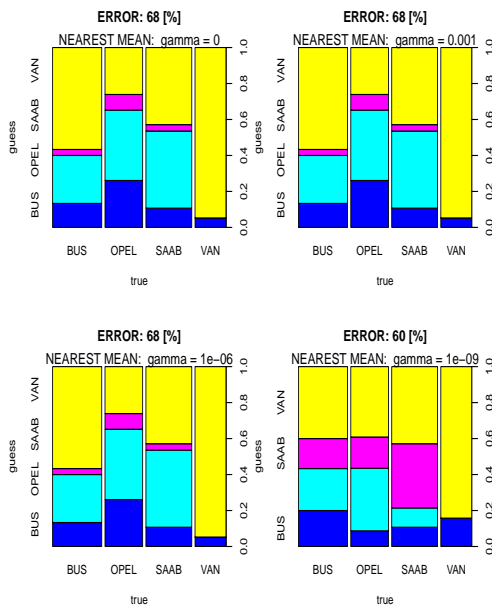
x, grouping Datensatz & Klasse
formula, data (Formelinterface)
kn Anzahl zu nutzender Nachbarn
gamma ggf. RBF-Parameter
return Klasse sknn

R-Programmcode

```
require (klaR)
df <- lapply (load (
  "~/data/set/statlog/vehicle/rda"),
  get)
for (k in c(1,2,4,8)) {
  o <- sknn (CLASS~.,
    df[[1]], kn=k)
  guess <- predict (o,
    df[[2]])$class
  plot (data.frame(true,guess),
    col=3+1:4)
}
```

Minimum-Abstand Klassifikator

`klaR::nm (x, grouping, kn=3, gamma=0)`



Funktionsargumente

x, grouping Datensatz & Klasse
formula, data (Formelinterface)
gamma ggf. RBF-Parameter
return Klasse sknn

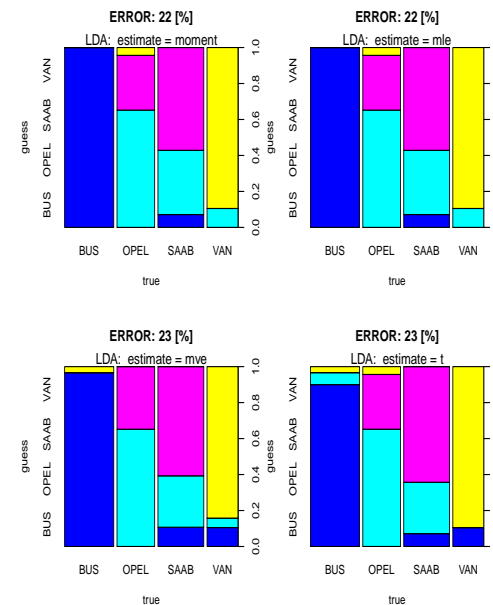
Diskriminante ($\gamma = 2\sigma^2$)
 $u_k(x) = \mathcal{N}(x | \mu_k, \sigma^2 \cdot E)$

R-Programmcode

```
require (klaR)
for (g in c(0,10^-c(3,6,9))) {
  o <- nm (CLASS~.,
    df[[1]], gamma=g)
  guess <- predict (o,
    df[[2]])$class
  plot (data.frame(true,guess),
    col=3+1:4)
}
```

Mahalanobis Klassifikator

`MASS::lda (x, grouping, prior, tol=1e-4, method='moment', CV=FALSE, nu)`



Funktionsargumente

x, grouping Datensatz, Klasse
formula, data (Formelinterface)
method moment, mle, mve, t
return Klasse lda

Diskriminante

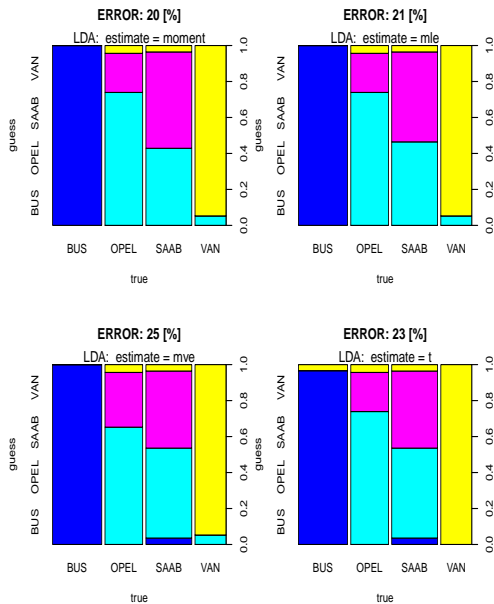
$$u_k(x) = p_k \cdot \mathcal{N}(x | \mu_k, S_0)$$

R-Programmcode

```
require (MASS)
for (mth in c(
  "moment", "mle", "mve", "t")) {
  o <- lda (CLASS~.,
    df[[1]], method=mth)
  guess <- predict (o,
    df[[2]])$class
  plot (data.frame(true,guess),
    col=3+1:4)
}
```

Normalverteilungsdichte-Klassifikator

MASS::qda (x, grouping, prior, method='moment', CV=FALSE, nu)



Funktionsargumente

x, grouping Datensatz, Klasse
formula, data (Formelinterface)
method moment, mle, mve, t
return Klasse lda

Diskriminante

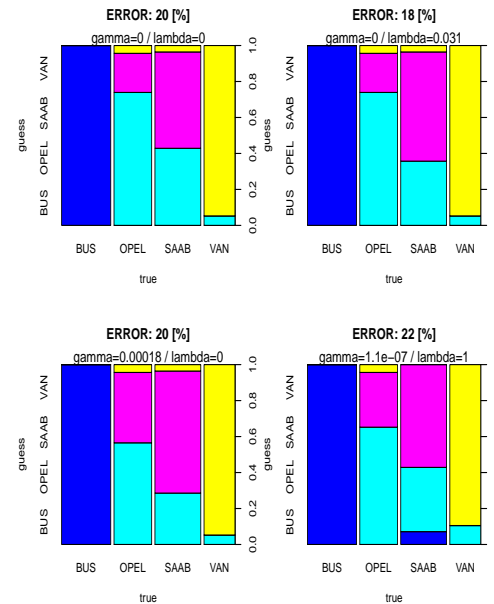
$$u_k(x) = p_k \cdot \mathcal{N}(x | \mu_k, S_k)$$

R-Programmcode

```
require(MASS)
for (mth in c(
  "moment", "mle", "mve", "t")) {
  o <- qda (CLASS~.,
    df[[1]], method=mth)
  guess <- predict (o,
    df[[2]])$class
  plot (data.frame(true,guess),
    col=3+1:4)
}
```

Regularisierte Normalverteilungs-Klassifikatoren

klaR::rda (x, grouping, gamma=NA, lambda=NA, crossval=TRUE, fold=10)



Funktionsargumente

x, grouping Datensatz, Klasse
formula, data (Formelinterface)
gamma, lambda Regularisierung
crossval, fold Parametertest
return Klasse rda

Diskriminante

$$S_k^\lambda := (1 - \lambda) \cdot S_k + \lambda \cdot S_0$$

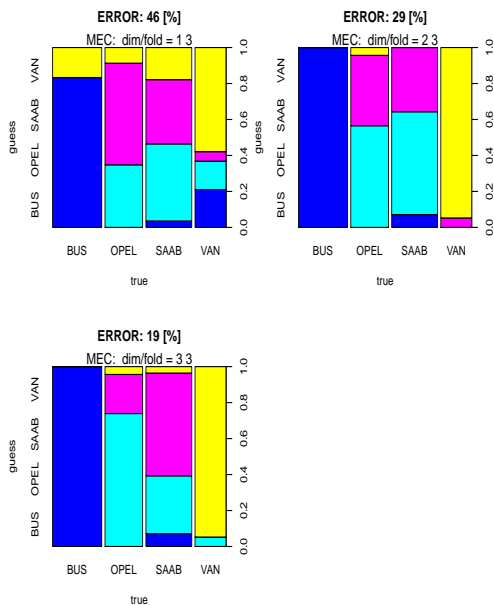
$$S_k^{\lambda\gamma} := (1 - \gamma) \cdot S_k^\lambda + \gamma \cdot \overline{\sigma^2(S_k^\lambda)} \cdot E$$

R-Programmcode

```
require(klaR)
for (g in c(0,NA)) {
  for (l in c(0,NA)) {
    o <- rda (CLASS~., df[[1]],
      gamma=g, lambda=l)
    guess <- predict (o,
      df[[2]])$class
  }
}
```

Linearer Minimalfehlerklassifikator

klaR::meclight (x, grouping, r=1, fold=10, ...)



Funktionsargumente

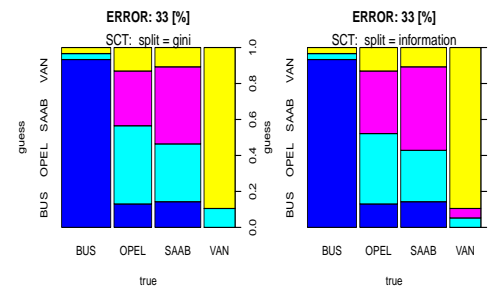
x Matrix/Dataframe
grouping Klasse (factor)
formula, data (Formelinterface)
r # Diskriminanten
fold # CV-Scheiben
... Argumente für lda()
return Objekt der Klasse lda

R-Programmcode

```
require(klaR)
for (f in 3)
  for (r in 1:3) {
    o <- meclight (CLASS~.,
      df[[1]], r=r, fold=f)
    guess <- predict (o,
      df[[2]])$class
    plot (data.frame(true,guess),
      col=3+1:4)
  }
```

Statistische Klassifikationsbäume

rpart::rpart (formula, data, method, parms, ...)



Funktionsargumente

formula, data (Formelinterface)
method anova,poisson,class,exp
parms Splittingparameter
return Klasse rpart

Entscheidungsbäume

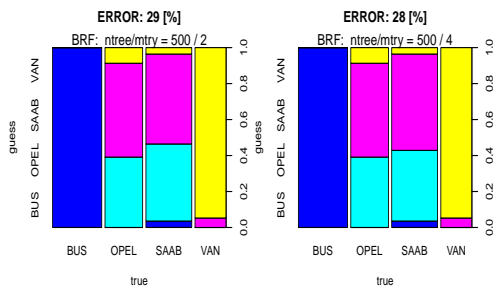
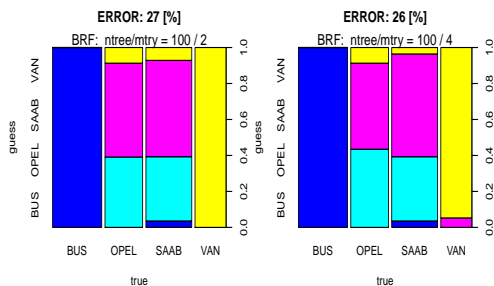
factor~. → method='class'

R-Programmcode

```
require(rpart)
for (s in c("gini","information")){
  o <- rpart (CLASS~., df[[1]],
    parms=list(split=s))
  guess <- predict (o,
    df[[2]], type="class")
  plot (o, uni=T, bran=0, comp=T)
  text (o, use.n=TRUE, col="blue")
}
```

Breimans Zufallsensemble von Klassifikationsbäumen

`randomForest::randomForest (formula, data, ntree=500, mtry=, ...)`



Funktionsargumente

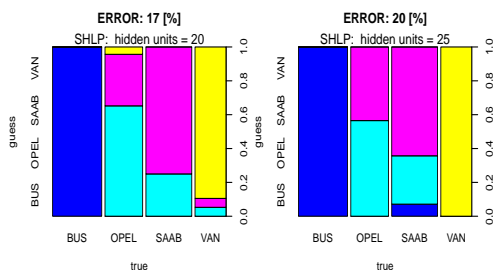
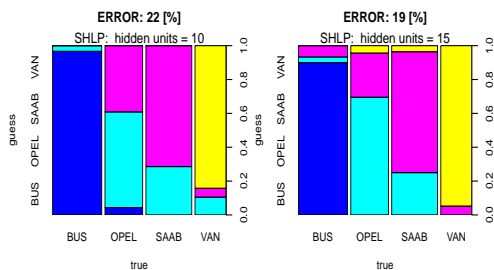
`formula, data` (*Formelinterface*)
`ntree` Anzahl Bootstrap-Bäume
`mtry` Auswahl Variablen/Frage
`replace=TRUE` Bootstrapped Technik
`nodesize` Stoppkriterium (lokal)
`maxnodes` Stoppkriterium (global)
`return` Klasse `randomForest`

R-Programmcode

```
require (randomForest)
for (n in c(100,500))
  for (m in c(2,4)) {
    o <- randomForest (
      CLASS~., X[[1]],
      ntree=n, mtry=m)
    guess <- predict (o, X[[2]])
    plot (data.frame(true,guess),
          col=3+1:4)
  }
```

Dreischichtenperzeptron (SHLP)

`nnet::nnet (x, y, size, Wts, mask, linout, entropy, softmax, censored, skip=FALSE, MaxNWts=1000, ...)`



Funktionsargumente

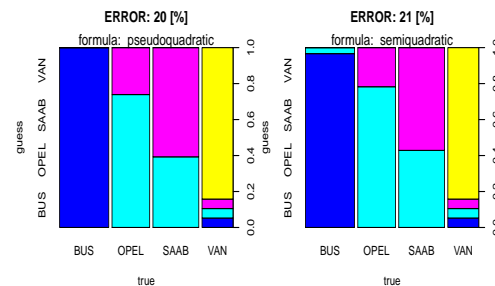
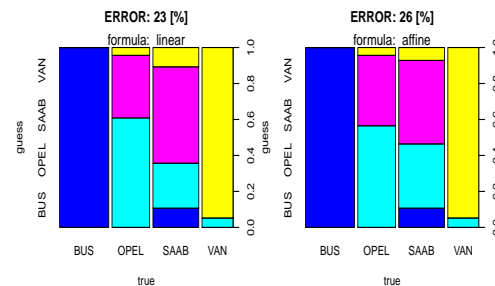
`x, y` Matrix/Dataframe für E/A
`formula, data` (*Formelinterface*)
`size` verborgene Neuronen
`skip` Kurzschlusskanten?
`Wts, MaxNWts` Startwerte, $\#_{\max}$
`return` Objekt der Klasse `nnet`

R-Programmcode

```
require (nnet)
for (h in 5*2:5) {
  o <- nnet (CLASS~., df[[1]],
    Wts=sin(1:nwts),
    MaxNWts=nwts,
    size=h)
  guess <- predict (o, df[[2]],
    type="class")
  plot (data.frame(true,guess),
        col=3+1:4)
}
```

Linearer Quadratmittelklassifikator

`stats::lm (formula, data, singular.ok=TRUE, contrasts=NULL, offset, ...)`



Funktionsargumente

`formula, data` (*Formelinterface*)
`return` Objekt der Klasse `lm`

Lineare Diskriminante

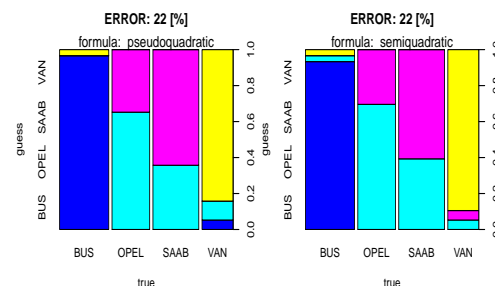
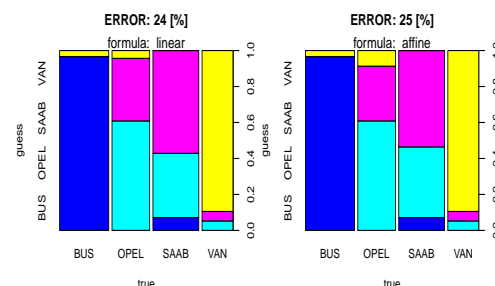
$$u_k(\mathbf{x}) = \mathbf{a}_k^\top \mathbf{x}$$

R-Programmcode

```
y <- df[[1]]$CLASS
Y <- diag(nlevels(y))[y,]
forms <- list (
  linear=Y~.-CLASS-1,
  affine=Y~.-CLASS+1, # etc.
)
for (j in seq_along(forms)) {
  o <- lm (forms[[j]], df[[1]])
  Y. <- predict (o, df[[2]])
  k. <-apply (Y., 1, which.max)
  guess <- levels(y)[k.]
}
```

Log-linearer Klassifikator ($K \geq 2$ Klassen)

`nnet::multinom (formula, data, contrasts=NULL, ...)`



Funktionsargumente

`formula, data` (*Formelinterface*)
`contrasts` für nominale Attribute
`...` Argumente für `nnet()`
`return` Objekt der Klasse `nnet`

Diskriminante

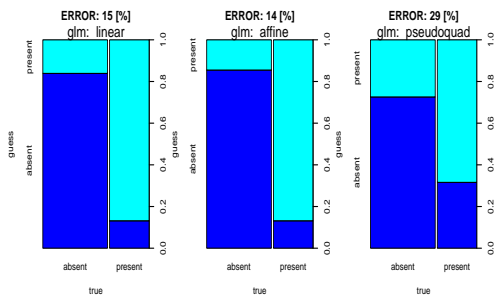
$$u_k(\mathbf{x}) = e^{\mathbf{a}_k^\top \mathbf{x}} / \sum_{\ell} e^{\mathbf{a}_{\ell}^\top \mathbf{x}}$$

R-Programmcode

```
require (nnet)
fo <- list (
  linear=CLASS~.-1,
  affine=CLASS~.+1, # etc.
)
for (j in seq_along(fo)) {
  o <- multinom (fo[[j]], df[[1]])
  guess <- predict (o, df[[2]])
}
```

Log-linearer Klassifikator (IRLS, $K = 2$ Klassen)

`stats::glm (formula, family=gaussian, data, start=NULL, ...)`



Funktionsargumente

formula, data (*Formelinterface*)
family { gaussian, poisson, Gamma
 binomial(link); quasi[...] }
link { logit, probit, cauchit, [clog]log }
start für IRLS (def=OLS)
return Objekt der Klasse glm

Diskriminante

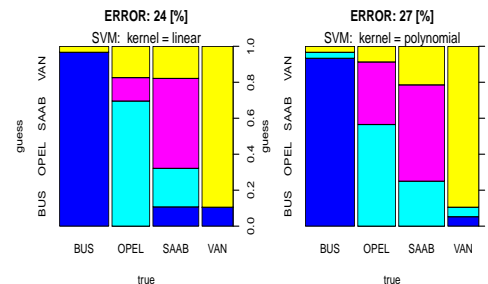
$$u(x) = e^{a^T x} / (1 + e^{a^T x})$$

R-Programmcode

```
# Use 'heart' dataset:
for(j in seq.along(fo)) {
  o <- glm (fo[[j]], df[[1]],
    family=binomial)
  lo <- predict (o, df[[2]])
  guess <- levels(true)[1+(lo>0)]
}
```

Supportvektormaschine (via LIBSVM)

`e1017::svm (x, y, scale=TRUE, type=NULL, kernel='radial', degree=3, gamma, coef0=0, cost=1, nu=0.5, epsilon=0.1, ...)`



Funktionsargumente

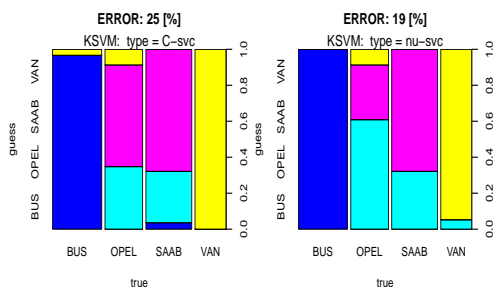
x Datenmatrix, div. Formate
y Vektor (factor, numeric)
type { C-class, nu-class, one-class
 eps-regress, nu-regress }
kernel { linear, poly, radial, sigmoid }
degree, gamma, coef0 (Kernel)
cost, nu, epsilon Regularis.
return Objekt der Klasse svm

R-Programmcode

```
require (e1071)
for (k in c("linear", # etc.
  o <- svm (CLASS~., df[[1]],
    type="nu-class", kernel=k)
  guess <- predict (o, df[[2]])
  plot (data.frame(true,guess),
    col=3+1:4)
}
```

Supportvektormaschine (via Platts SMO)

`kernlab::ksvm (x, y, scaled=TRUE, type=NULL, kernel='rbfdot', kpar='automatic', C=1, nu=0.2, epsilon=0.1, fit=TRUE, ...)`

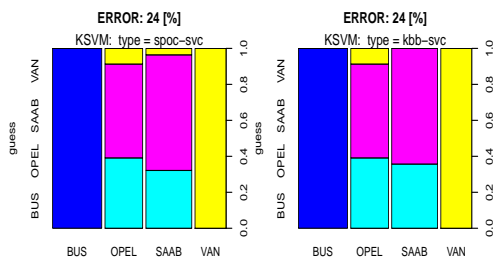


Funktionsargumente

x, data (*Formelinterface*)
type { {C,nu,spoc,kbb,one}-svc
 {eps,nu}-svr, eps-bsvr }
kernel { {rbf,poly,vanilla,tanh}-dot
 {laplace,bessel}-dot
 {anova,spline,string}-dot }
kpar Kernkonfiguration (list)
C, nu, epsilon Regularisierung
return Objekt der Klasse svm

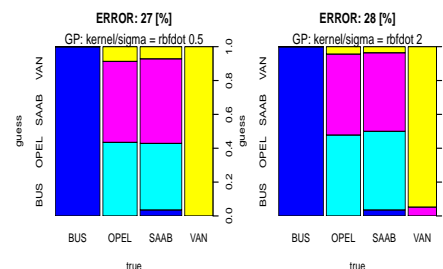
R-Programmcode

```
require (kernlab)
for (t in c("C-svc","nu-svc", # etc.
  o <- ksvm (CLASS~., df[[1]],
    type=t, kernel="rbfdot")
  guess <- predict (o, df[[2]])
  plot (data.frame(true,guess),
    col=3+1:4)
}
```



Gaußprozesse (GP)

`kernlab::gausspr (formula, data, scaled=TRUE, kernel='rbfdot', ...)`

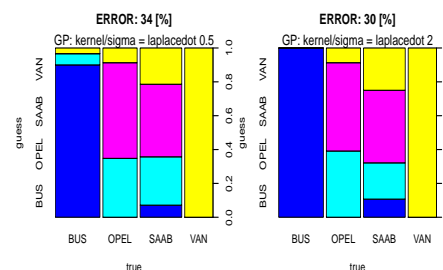


Prädiktorargumente

object Objekt der Klasse gausspr
newdata Input-Variablensatz
type response, probability
coupler minpair, pkpd
return Faktor oder W'keitsmatrix

R-Programmcode

```
require (kernlab)
for (k in c("rbfdot","laplacedot"))
  for (s in c(1/2,2)) {
    o <- gausspr (CLASS~., df[[1]],
      kernel=k,
      kpa=list(sigma=s))
    guess <- predict (o, df[[2]])
    plot (data.frame(true,guess),
      col=3+1:4)
  }
```



Digitale Signalverarbeitung

Merkmals- und Dimensionsreduktion

Numerische und ordinale Vorhersage

Klassifikation und überwachtes Lernen

Clusteranalyse und unüberwachtes Lernen

Die vielen Gesichter der Clusteranalyse

$$\omega = \{x_1, \dots, x_T\} \subset \Omega \Rightarrow \omega_1 \uplus \omega_2 \uplus \dots \uplus \omega_K = \omega$$

GEGEBEN:

Ein Quelldatensatz

 $x_1, \dots, x_T \in \Omega$ ohne

Klassenetikettierung

GESUCHT¹: **EX**tensional GruppierenEine **Partitionierung** $\pi : \{1, \dots, T\} \rightarrow \{1, \dots, K\}$ des Datensatzes ω GESUCHT²: **IN**tensional GruppierenEine **Entscheidungsfunktion** $f : \Omega \rightarrow \{1, \dots, K\}$ zurGruppierung des Bereichs Ω

Gütekriterien

global:· Likelihood (*Mischungsverteilung*)· Verzerrung (*Gruppenprototypen*)**lokal:**· Homogenität (*divisiv*)· Ähnlichkeit (*agglomerativ*)

Gruppenrepräsentation

$$\left\{ \begin{array}{l} \text{scharf} \\ \text{unscharf} \\ \text{gestaffelt} \end{array} \right\} \times \left\{ \begin{array}{l} \text{stetig} \\ \text{diskret} \\ \text{Metrik} \end{array} \right\} \times \left\{ \begin{array}{l} \text{sphärisch} \\ \text{geformt} \\ \text{amorph} \end{array} \right\}$$

Regularisierung

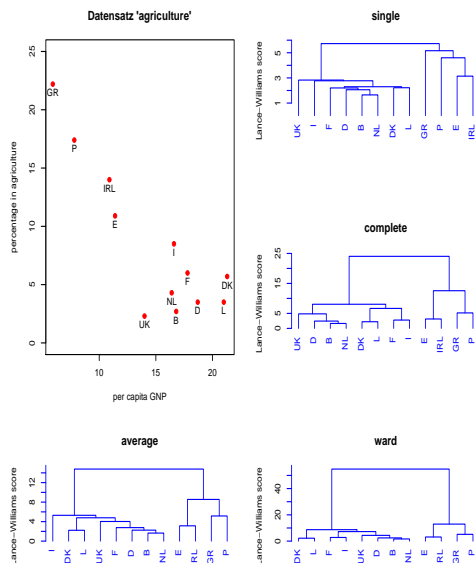
Clustering $\hat{=}$ „ill-formed problem“:

· Anzahl der Gruppen

· Komplexität ihrer Gestalt

Agglomerative Gruppierung

```
stats::hclust(d, method='complete', members=NULL)
```



Funktionsargumente

d Distanzmatrix (dist)
method { complete, single, average, ward, mcquitty, median, centroid }
members (zum Neuaufsetzen)
return Klasse hclust

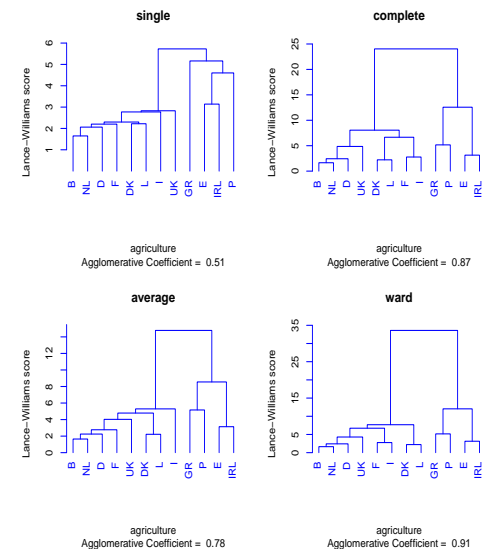
R-Programmcode

```
require(cluster) # agriculture
for(m in hcm)
  plot(hclust(
    dist(agriculture),
    method=m),
    hang=-1,
    main=m, sub="", xlab="",
    ylab="Lance-Williams score",
    col="blue")
```

```
cutree(tree, k=NULL, h=NULL)
```

Agglomerative Gruppierung (Lance-Williams)

```
cluster::agnes(x, diss, metric='euclidean', stand=FALSE,
  method='average', par.method)
```



Funktionsargumente

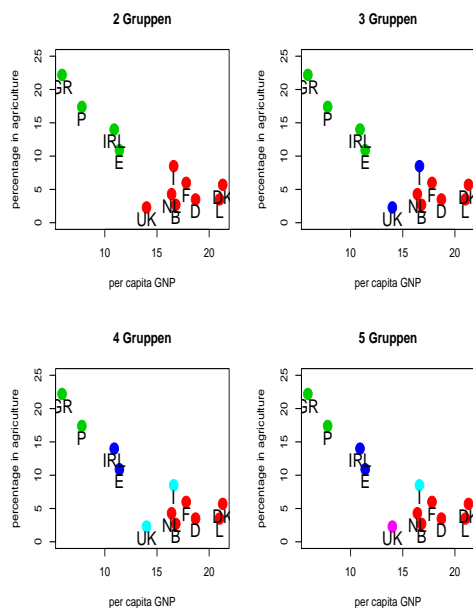
x Distanz- oder Datenmatrix
diss ist x Typ dist?
metric falls diss=FALSE
stand skalare Normierung?
method { complete, single, average, weighted, ward, flexible }
par.method LW($\alpha_1, \alpha_2, \beta, \gamma$)
return Klasse agnes

R-Programmcode

```
require(cluster)
for(m in hcm) plot(
  agnes(agriculture, method=m),
  which.plots=2,
  main=m, hang=-1,
  ylab="Lance-Williams score",
  col="blue")
```

Agglomerative Gruppierung (Gaußmischung)

`mclust::hc (modelName, data, ...) & hclass (hcPairs, G)`



Funktionsargumente

`modelName` $\left\{ \begin{array}{l} E = \text{univariat, } \sigma_k^2 \text{ fest} \\ V = \text{univariat, } \sigma_k^2 \text{ var.} \\ EII = \text{sphär., } V_k \text{ fest} \\ VII = \text{sphär., } V_k \text{ var.} \\ EEE = \text{ellipt., } S_k \text{ fest} \\ VVV = \text{ellipt., } S_k \text{ var.} \end{array} \right.$

`data` Datensatz, numerisch(!)

`return` $(n, 2)$ -Indexmatrix (Aggl.)

`hcPairs` eine hc-Rückgabe

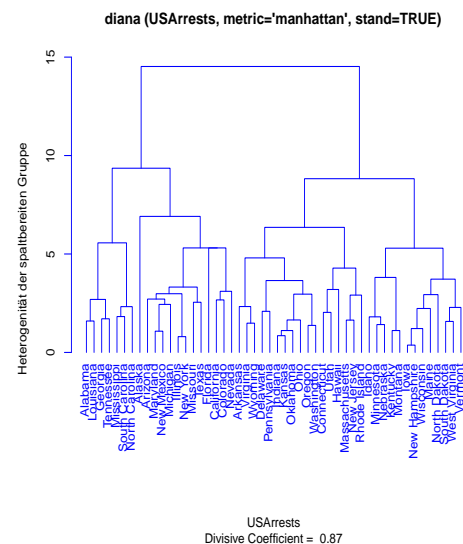
`G` welche HC-Ebenen?

R-Programmcode

```
require (mclust)
tree <- hc ("VVV", agriculture)
A <- hclass (tree, 1:5)
for (k in 2:5) {
  plot (agriculture,
        col=1+A[,k], cex=2,
        main=paste (k, "Gruppen"))
}
```

Divisive Gruppierung (Splintertechnik)

`cluster::diana (x, diss, metric='euclidean', stand=FALSE, keep.diss=n<100, keep.data=!diss)`



Funktionsargumente

`x` Distanz- oder Datenmatrix

`diss` ist x Typ dist?

`metric` {euclidean, manhattan}

`stand` skalare Normierung?

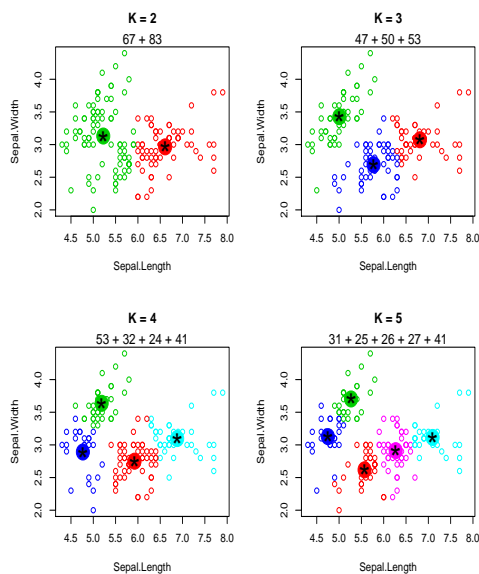
`return` Klasse diana

R-Programmcode

```
require (cluster)
plot (diana (
  USArrests,
  metric="manhattan",
  stand=TRUE),
  hang=-1,
  col="blue",
  which.plots=2)
```

Austauschverfahren K-Means (numerisch)

`stats::kmeans (x, centers, iter.max=10, nstart=1, algorithm='Hartigan')`



Funktionsargumente

`x` Datenmatrix (numeric)

`centers` Anzahl K, Startzentren

`nstart` Anzahl Zufallsstarts

`algorithm` $\left\{ \begin{array}{l} \text{Hartigan-Wong, Lloyd} \\ \text{Forgy, MacQueen} \end{array} \right.$

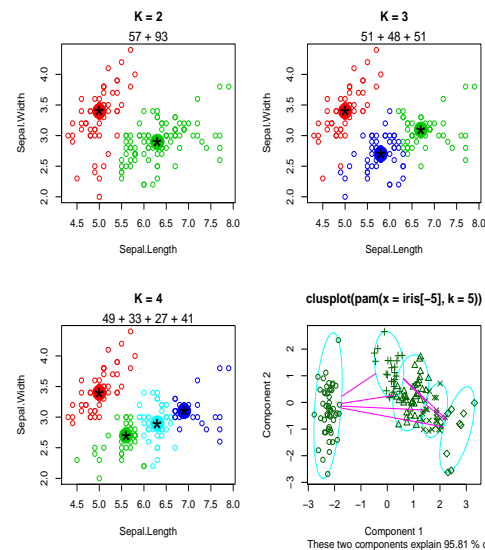
`return` Klasse kmeans: $\left\{ \begin{array}{l} \text{cluster} \\ \text{centers} \\ \text{size} \end{array} \right.$

R-Programmcode

```
xy <- iris[,1:2]
for(k in 2:5) {
  o <- kmeans (xy, c=k, ns=10)
  plot (xy, col=1+o$cluster)
  points (o$centers,
          pch=19, cex=3, col=1+1:k)
  title (main=paste ("K =", k))
  mtext (paste (
    o$size, collapse=" + "))
}
```

Austauschverfahren K-Medoids (metrisch)

`cluster::pam (x, k, diss, metric='euclidean', medoids=NULL, stand=FALSE, do.swap=TRUE, trace.lev=0)`



Funktionsargumente

`x` Distanz- oder Datenmatrix

`k` Anzahl Gruppen

`diss` ist x Typ dist?

`metric` falls diss=FALSE

`medoids` ggf. die Startmedoide

`stand` skalare Normierung?

`do.swap` mit Schüttelphase?

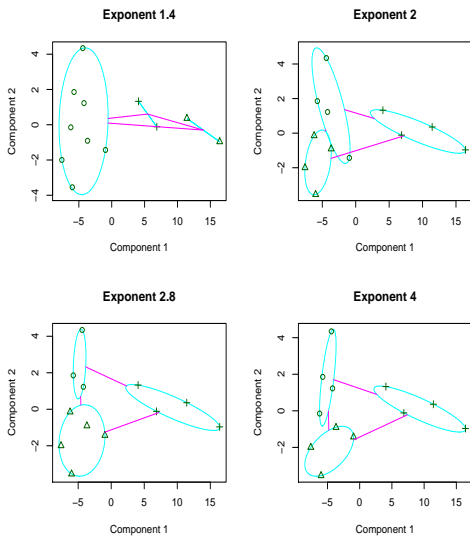
`return` Klasse pam

R-Programmcode

```
require (cluster)
xy <- iris[,1:2]
for(k in 2:4) {
  o <- pam (xy, k)
  plot (xy, col=1+o$clustering)
  points (o$medoids, pch="*", cex=3)
}
clusplot (pam (iris[-5], k=5))
```

Fuzzy K-Means (metrisch)

```
cluster::fanny (x, k, diss, memb.exp=2, metric='euclidean', stand=FALSE,
  iniMem.p=NULL, trace.lev=0)
```



Funktionsargumente

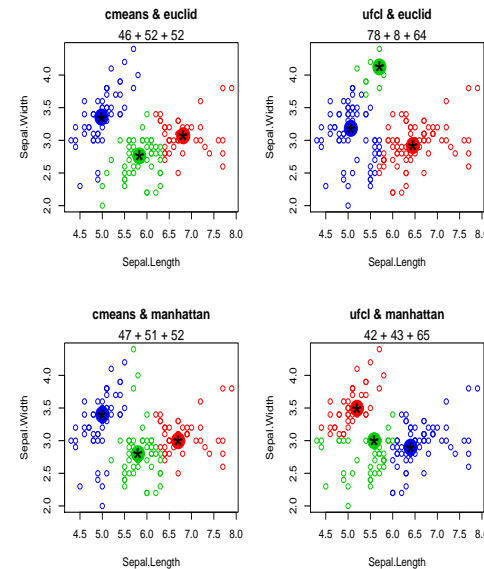
x Distanz- oder Datenmatrix
k Anzahl Gruppen
diss ist x Typ dist?
metric falls diss=FALSE
iniMem.p Startmemberships
stand skalare Normierung?
return Klasse fanny

R-Programmcode

```
require (cluster)
for (ex in c(1.4,2.0,2.8,4.0))
  clusplot (fanny (agriculture,
    k=3, memb.exp=ex),
    main=paste ("Exponent", ex),
    sub="")
```

Fuzzy K-Means (batch/online)

```
e1071::cmeans (x, centers, iter.max=100, dist='euclidean',
  method='cmeans', m=2, rate.par=0.3)
```



Funktionsargumente

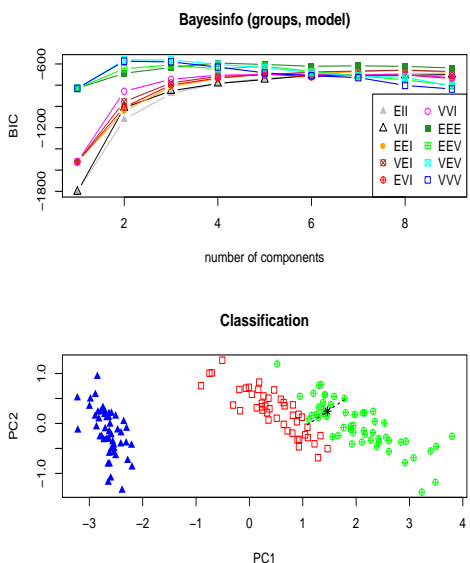
x Datenmatrix
centers #Gruppen, Startprotot.
dist {euclidean, manhattan}
method {cmeans, ufcl}
m Fuzzyexponent
rate.par Lernrate (*online*)
return Klasse fclust

R-Programmcode

```
require (e1071)
xy <- iris[,1:2]
for (d in c("euclid","manhattan"))
  for (mth in c("cmeans","ufcl")) {
    o <- cmeans (xy, centers=3,
      dist=d, method=mth)
    plot (xy, col=1+o$cluster)
    points (o$cent, pch="*", cex=3)
  }
```

EM-Algorithmus (Gaußmischung)

```
mclust::Mclust (data, G=1:9, modelNames=NULL, prior=NULL,
  control=emControl(), initialization=NULL, warn=FALSE, ...)
```



Funktionsargumente

data Datensatz, numerisch(!)
G welche K-Werte? (integer)
modelName $\left\{ \begin{array}{l} \text{EII} = \text{sphär.}, V_k \text{ fest} \\ \text{VII} = \text{sphär.}, V_k \text{ var.} \\ \text{EVI} = \text{diag.}, D_k \text{ var.} \\ \dots \\ \text{EEE} = \text{ellipt.}, S_k \text{ fest} \\ \text{VVV} = \text{ellipt.}, S_k \text{ var.} \end{array} \right.$
prior konjugierte Fam.par.
control für EM-Aufrufe
return BIC-optimales Modell

R-Programmcode

```
require (mclust)
o <- Mclust (iris[1:4])
plot.mclustBIC (o$BIC)
o <- Mclust (iris[1:4], G=3:5)
xy <- prcomp (iris[1:4])$x[,1:2]
plot (o, data=xy, what="classification")
```

Zusammenfassung (5)

1. Für die meisten Werkzeuge reserviert das 'R'-Implementierungsschema eine **Klasse** (S3/S4), einen gleichnamigen **Konstruktor** für die Lernphase und eine Methode **predict()** für die Abrufphase.
2. Zur Digitalen Signalverarbeitung gibt es Methoden für das **Falten** und **Filtern**, die **Spektralanalyse**, die **Autokorrelationsanalyse** und die Anpassung **autoregressiver-** und **ARMA-Modelle**.
3. Zur Koordinatentransformation gibt es lineare Methoden (**PCA**, **Faktoranalyse** und **ICA**), konvexe Abbildungen (**NMF**), konnektionistische Modelle (**SOFM** und **AANN**) sowie metrik- und skalarproduktorientierte Ansätze (**MDS**, **Kernel-PCA** und **geodätische Projektionen**).
4. Zur Vorhersage numerischer Attribute gibt es **LSE-** und **ML-Ansätze** mit **linearen**, **baumförmigen** oder **neuronalen** Funktionsprototypen sowie **Gaußprozesse**.
5. Zur Klassifikation kennen wir distanzbezogene (**K-NN**), statistische (**NVK**), baumförmige (**SCT**) und zahlreiche (lineare und nichtlineare) diskriminative (**QMK**, **MLP**, **Logit**, **SVM**) Verfahren.
6. Zur Clusteranalyse unterscheiden wir hierarchische (**agglomerativ** und **divisiv**), austauschende (**fuzzy**) **K-means** und **EM-Entmischung** und globale Partitionierer (**spektral**).