R-Help

```
# All search objects matching the RegEx
apropos("topic")
# Get a summary of an object's structure.
str(iris)
# Find the class an object belongs to.
class(iris)
# Outputting variable (e.g. in loop)
print(x)
# Different ways to create vectors
c(1, 2) | 2:6 | seq(2, 3) | rep(1:2, times=3)
# Vector functions
sort, rev, table, unique
x <- append(x, 10) # Appending elements
data <- data[,c("x1","x2","x3")] # naming columns
# Creating dataframes
employ.data <- data.frame(em, sal, st) # from vectors
x <- data.frame("SN" = 1:2, "Age" = c(21,15), "Name"
↪  = c("John","Dora"))
# for loop:
for (i in 1:4) { }
# function:
square <- function(x) { }
# Converting types
as.logical | as.factor | as.numeric
# Math functions
quantile, rank, round, var, cor, sd
# Apply functions to array / matrix
apply(X, MARGIN, FUN, ...)
# Matrix operations
m[2, ] # Select row
t(m) # Transpose
diag(m) # Diagonal
A %*% B # Matrix multiplication
solve(m, n) # Find x in: m * x = n
cbind / rbind # Bind columns / rows
nrow / ncol # Number of rows / columns
# Select subset
d[d$x1 == 0 & d$x2 == 1,]
# Ordering data
d[order(d[,"x1"],decreasing=T)[1:3],]
# Prediction / Confidence Intervals
nc=data.frame(l2tv=log2(50),l2dr=log2(3000))
predict(fit, newdata=nc, interval="confidence")
predict(fit, newdata=nc, interval="predict")
# Excluding datapoints
data[-c(1,5),]
# Test if an element is in a list
if ("X1" %in% names(coef(m,mo)))
# Extracting fields from a named num
num[["1"]] # single field
unname(coefficients(out)[c("newx", "(Intercept)")])
# Fit distribution to data vector
library(MASS)
fit.gamma <- fitdistr(boogg, "gamma")
# Remove NA
data_comp <- data[complete.cases(data),]
# Creating categorical variables
High=ifelse(Carseats$Sales<=8,"No","Yes")
# Standardize data
scaled.dat <- scale(dat)
# ROC curve
require(ROCR)
fit <- glm(Survival ~ ., data = d.baby, family =
↪  "binomial")
pred <- prediction(fit$fitted.values,
↪  d.baby$Survival)
perf <- performance( pred, "tpr", "fpr" )
# Anova test to determine if there is a significant
# difference between models. Anova uses RSS and DoF
# of largest (last) model, so use ascending order!
anova(fit.0, fit.1, fit.2, fit.3)
# Given fixed x, error distribution and true param.:
# Power of test simulation. Know that y ~ poly(x, 3)
↪  + err (for typeI error: do same with y = err)
results.power <- numeric(n.sim)
for (i in 1:n.sim) {
    err <- rgamma(n, ...) - 2
    y <- beta.0 + beta.1 * I(x) + beta.2 * I(x^2) +
    ↪  beta.3 + I(x^3) + err
    fit.power <- lm(y ~ I(x) + I(x^2) + I(x^3))
    f1 <- summary(fit.power)$fstatistic
    p.val.power <- 1 - pf(f1[1], f1[2], f1[3])
    results.power[i] <- p.val.power < 0.05
}
power <- mean(results.power)
```

```
# Plotting
par(mfrow=c(1,1)) # Arrangement of plots
plot(..., xlim=c(-2,2), ylim=c(-5,8), xlab="x",
↪  ylab="y")
abline(a=..., b=..., col="red") # Add straight line
abline(h=..., col="red") # Add horizontal line
abline(v=..., col="red") # Add vertical line
lines(density(some.vector)) # Add density
# draw density and CDF
grid <- seq(from=0,to=5,length=200)
plot(grid, dlnorm(grid), type="l", main="density")
plot(grid, plnorm(grid), type="l", main="CDF")
# Add regression line to data plot
library(MASS)
attach(Boston)
fit <- lm(dis ~ poly(nox, degree = 3))
line.x <- seq(min(nox), max(nox), length.out = 1000)
line.y <- predict(fit, data.frame(nox = line.x))
plot(dis, nox)
lines(line.y, line.x, col = "red")
```

Cross Validation

Can be used for *model assessment* (estimate test MSE) and *model selection* (choose tuning parameters, variable selection). But not both at the same time (use double CV instead).
Validation set: split data into two halves, train on one, test on the other (high bias and variance as estimate of true generalization error).
k-Fold: same, but with many folds. Try all folds for test and average metrics over the folds (bias in between, variance compared to LOOCV unclear). $\mathrm{Var}(\hat{\theta}_k) = 1/K \cdot \hat{\mathrm{Var}}(MSEs)$
LOOCV: extreme version where each data point is a fold (least bias, typically high variance).
$\theta_k = \frac{1}{k}\sum_{i=1}^{k} \frac{1}{|I_k|}\sum_{i\in I_k}(y_i - \hat{f}^{-I_k}(x_i))^2$, $\theta_L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}^{-i}(x_i))^2$
Computational Shortcut: For linear fitting operator $\mathcal{S}$ and squared error loss function $\rho(y,x) = |y-x|^2$, can compute LOOCV score by fitting original estimator once on full dataset: $n^{-1}\sum_{i=1}^{n}\left(Y_i - \hat{m}_{n-1}^{(-i)}(X_i)\right)^2 = n^{-1}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{m}(X_i)}{1-\mathcal{S}_{ii}}\right)^2$. Historically, GCV was used because computing trace was easier: $\mathrm{GCV} = \frac{n^{-1}\sum_{i=1}^{n}(Y_i-\hat{m}(X_i))^2}{\left(1-n^{-1}\operatorname{tr}(\mathcal{S})\right)^2}$

Cross Validation

```
for(j in 1:n) { # LOOCV of sum of squared residuals
    fit <- fitfn(formula=formula, data = data[-j,])
    ssr <- ssr + (model.response(modFrame)[j] -
    ↪  predict(fit, modFrame[j,]))^2
}
d <- d[sample(nrow(data)),] # k-fold CV
folds <- cut(seq(1,nrow(d)),breaks=10,labels=FALSE)
for(i in 1:10){
    testIndexes <- which(folds==i,arr.ind=TRUE)
    testData <- d[testIndexes, ]
    trainData <- d[-testIndexes, ]
}
```

Linear Regression

$y_i = \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i$ (note: $x_{i1} \equiv 1$, so β_1 is intercept) where $\epsilon_1,\ldots,\epsilon_n$ independent, $E(\epsilon_1) = 0, \mathrm{Var}(\epsilon_i) = \sigma^2$ (homoscedasticity)
LSE: $\hat{\beta} = \mathrm{argmin}_\beta \|Y - Xb\|_2^2 = (X^\top X)^{-1}X^\top Y \sim \mathcal{N}_p(\beta, \sigma^2(X^\top X)^{-1})$
$\hat{\sigma}^2 \approx \frac{1}{n-p}RSS = \frac{1}{n-p}\sum_{i=1}^{n} r_i^2$ also if error Gaussian then: $\hat{Y} \sim \mathcal{N}_n(X\beta, \sigma^2 P)$, error $e \sim \mathcal{N}_n(0, \sigma^2(I-P))$, $\hat{\sigma}^2 \sim \sigma^2/(n-p)\cdot\chi^2_{n-p}$ where $P = X(X^\top X)^{-1}X^\top$ Assumptions: Correct linear equation ($\mathbb{E}(\epsilon_i) = 0$, violation: other models), exact x_i's (violation: corrections from ërrors in variables"), homoscedasticity ($Var(\epsilon_i) = \sigma^2$, violation: weighted least squares), uncorrelated and jointly normal distributed errors (violation: robust methods)
Geometric Interpretation: $X\hat{\beta}$ is orthogonal projection of Y onto p-dim. subspace described by $X\beta$. Residuals are orthogonal to this subspace, which means $r^t x^{(j)} = 0$.
Simple least squares regressions yield multiple regressions least squares solution only if predictor variables are orthogonal.
Properties: $\mathbb{E}(\hat{\beta}) = \beta$, $\mathbb{E}(\hat{Y}) = \mathbb{E}(Y) = X\beta$, $\mathrm{Cov}(\hat{\beta}) = \sigma^2(X^\top X)^{-1}$, $\mathrm{Cov}(\hat{Y}) = \sigma^2 P$, $\mathrm{Cov}(r) = \sigma^2(I-P)$, $\mathbb{E}(\hat{\sigma}) = \sigma$.

Measuring Goodness of Fit

$H_0: y = X\beta + \epsilon$ with $\beta_j = 0$ $H_A: y = X\beta + \epsilon$ with $\beta_j \neq 0$

Under H_0: $\frac{\hat{\beta}_j - (E[\hat{\beta}_j]=0)}{\sqrt{\sigma^2(X^\top X)^{-1}_{jj}}} \sim \mathcal{N}(0,1)$ t-statistic: $\frac{\hat{\beta}_j}{\sqrt{\hat{\sigma}^2(X^\top X)^{-1}_{jj}}} \sim t_{n-p}$

Total variation prop. of $\mathbf{Y}$ around $\overline{\mathbf{Y}}$ explained by regr.: $R^2 = \frac{\|\hat{\mathbf{Y}}-\overline{\mathbf{Y}}\|^2}{\|\mathbf{Y}-\overline{\mathbf{Y}}\|^2}$
F-test: Global null, test all $\beta_j = 0$ (can reject global H_0, even when all individual tests do not): $F = \frac{\|\hat{\mathbf{Y}}-\overline{\mathbf{Y}}\|^2/(p-1)}{\|\mathbf{Y}-\hat{\mathbf{Y}}\|^2/(n-p)} \sim F_{p-1,n-p}$
P-Value: P(obs. a value of the test stat. that is as extreme or more extreme than the one we saw if H_0 is true). If $< \alpha$ then reject H_0.
Polynomial Regression: With `raw = F`, `poly()` uses orthogonal polynomial. Does not change fitted values, but it is possible to see whether a certain order significantly improves regression over lower ones.
Generalized Least Squares: $X = X\beta + \epsilon$ with $\epsilon \sim \mathcal{N}_n(0,\Sigma)$. Solution: $\hat{\beta} = (X^\top\Sigma^{-1}X)^{-1}X^\top\Sigma^{-1}Y$ and $\mathrm{Cov}(\hat{\beta}) = (X^\top\Sigma^{-1}X)^{-1}$

Linear Regression

```
fit <- lm(y~x1+x2) # Fit only x1 and x2 (so p=3)
# alt.: lm(y~.) or lm(y~.-1) without intercept
predict(fit, pred.data.frame)
predict(fit, df)[["1"]] # Extracting value
# Getting coefficients
coef(fit)
# Fitting polynomial manually (only equiv. to poly if
↪  raw=TRUE)
fit <- lm(wage~1+a+I(a^2)+I(a^3)+I(a^4),data=Wage)
# Manual fit
X <- cbind(1, x1, x2) # p = 3
XtX.inv <- solve(t(X) %*% X)
beta.hat <- XtX.inv %*% t(X.int) %*% y
res <- y - X.int %*% beta.hat # Residuals
RSE <- sqrt(sum(res^2)/(n-p)) # Residual std. error.
↪  Est. of the sd of the noise in the linear model
se.x1 <- RSE * sqrt(XtX.inv[2, 2]) # Std. error of x1
t.val.x1 <- beta.hat[2] / se.x1 # T value of x1
p.val.x1 <- 2*pt(abs(t.val.x1), df=n-p, lower=F)
# R squared: proportion of Var(y) that is explained
↪  by the fitted linear model.
RSE <- sqrt(sum(residuals(fit)^2)/(n-p))
RSS <- sum(res^2) # Residual sum of squares
TSS <- sum((y - mean(y))^2) # Total sum of squares
R.sq <- 1 - RSS / TSS
AdjR2 <- 1 - (RSS/(n-p))/(TSS/(n-1))
# Alternative t-value
coef <- summary(fit1)$coefficients
t1 <- coef["x1","Estimate"]/coef["x1","Std. Error"]
# Finding p-values
fit.smaller <- lm(y ~ x1)
anova(fit.smaller, fit, fit.all)
# Overall F-Test
fit.empty <- lm(y ~ 1, data=...) # Empty model
anova(fit.empty, fit) # Compare models
```

Last line of `summary`: Residual standard error: $\sqrt{\hat{\sigma}^2}$ on $n-p$ df. `Std. Error` correspond to $\sqrt{\hat{\mathrm{Var}}(\hat{\beta}_j)} = \sqrt{\hat{\sigma}(X^\top X)^{-1}_{jj}}$
R Diagnostic plots: `plot(fit, which=...)` #1 Tukey-Anscombe Plot (residuals against fitted values) the points fluctuate randomly, else $E(\epsilon) = 0$ violated. For linear trend, take log, if trend sqrt, take square root #2 Q-Q Plot (empirical vs. theoretical quantiles) should follow line, else error not Gaussian (still all fine). #3 Scale-Location: should be flat, else $\mathrm{Var}(\epsilon_i) = \sigma^2$ violated (p-values wrong). #4/#5 Cook distance: shows if some data points have a larger impact on the fit than others (outliers). Note: cannot detect if the residuals are correlated with these plots!

Confidence Intervals

Want to calculate: $P\left(t_{\alpha/2,n-p} < \frac{\hat{\beta}_j - \beta_j}{\hat{se}(\hat{\beta}_j)} < t_{1-\alpha/2,n-p}\right) = 1-\alpha$.

$CI = \hat{\beta}_j \pm \hat{se}(\hat{\beta}_j)\cdot t_{1-\alpha/2,n-p} = \hat{\beta}_j \pm \hat{\sigma}\sqrt{x_0^\top(X^\top X)^{-1}x_0}\cdot t_{1-\alpha/2,n-p}$

Prediction Interval

For new point x_0: $\hat{y}_0 = x_0^\top\beta \pm \hat{\sigma}^2\sqrt{1 + x_0^\top(X^\top X)^{-1}x_0}\cdot t_{1-\alpha/2,n-p}$
Confidence interval gives range for $E[y|x]$, prediction interval for y (will always be wider).

Bias Variance Trade-Off

Expected Test MSE at x_0: $E[y_0 - \hat{f}(x_0)^2] = \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\hat{f}(x_0)) + \sigma^2$, where $\mathrm{Bias}^2(\hat{f}(x_0)) = (f(x_0) - E[\hat{f}(x_0)])^2$.

Confidence Intervals

```
confint(fit) # Automatic CI
# Manual CI (for intercept)
se.intercept <- summary(fit)$coef[1,2]
coef(fit)[1] - qt(.975, n-2)*se.intercept
coef(fit)[1] + qt(.975, n-2)*se.intercept
# Automatic Prediction CI
predict(fit,data.frame(name=5),level=.95,interval="p")
# Manual Prediction CI
fitted <- fit$coef[1] + fit$coef[2]*x0
quant <- qt(.975,n-2) # Quantile of t distribution
sigma.hat <- sqrt(sum((fit$resid)^2/(n-2)))
X <- as.matrix(cbind(1,thuesen[,1]))
XtXi <- solve(t(X) %*% X)
X00 <- as.matrix(c(1,x0), nrow=2)
se <- sigma.hat * sqrt(t(X00) %*% XtXi %*% x00)
lower <- fitted - quant * se
upper <- fitted + quant * se
# Bias Variance Trade-Off of a Method
Bias <- mean(EstimateUsingCV) - TrueValueSimulated
MSE <- Bias^2 + var(EstimateUsingCV)
```

Nonparametric Regression

$Y_i = m(x_i) + \epsilon_i$ with ϵ_i i.i.d., $E(\epsilon_i) = 0$, m arbitrary.
Kernel Regression: $m(x) = \mathbb{E}[Y \mid X = x]$. Using $\int_{\mathbb{R}} y f_{Y|X}(y \mid x)dy = \frac{\int_{\mathbb{R}} y f_{X,Y}(x,y)dy}{f_X(x)}$ and plugging in $\hat{f}_X(x) = \frac{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)}{nh}$, $\hat{f}_{X,Y}(x,y) = \frac{\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)K\left(\frac{y-Y_i}{h}\right)}{nh^2}$ leads to Nadaraya-Watson: $\hat{m}(x) = \frac{\sum_{i=1}^{n} K((x-x_i)/h)Y_i}{\sum_{i=1}^{n} K((x-x_i)/h)} = \frac{\sum_{i=1}^{n}\omega_i Y_i}{\sum_i \omega_i}$ (large h: high bias, small var.)
Alternative interpretation: $\hat{m}(x) = \underset{m_x\in\mathbb{R}}{\arg\min}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)(Y_i - m_x)^2$

Locally optimal bandwidth: $h_{\mathrm{opt}}(x) = n^{-1/5}\left(\frac{\sigma_\varepsilon^2\int K^2(z)dz}{\{m''(x)\int z^2 K(z)dz\}^2}\right)^{1/5}$

(estimated using plug-in principle, `lokern` in R)
Hat Matrix: $\hat{\mathbf{Y}} = \mathcal{S}\mathbf{Y}$, $[\mathcal{S}]_{r,s} = w_s(x_r)$, $w_i(x) = \frac{K((x-x_i)/h)}{\sum_{j=1}^{n} K\left((x-x_j)/h\right)}$, $df = \operatorname{tr}(\mathcal{S})$, $\mathrm{Cov}(\hat{\mathbf{m}}(x.)) = \sigma_\varepsilon^2\mathcal{S}\mathcal{S}^\top$, $\hat{\sigma}_\varepsilon^2 = \frac{\sum_{i=1}^{n}(Y_i - \hat{m}(x_i))^2}{n-df}$. Under regularity conditions: $\hat{m}(x_i) \approx \mathcal{N}(\mathbb{E}[\hat{m}(x_i)], \mathrm{Var}(\hat{m}(x_i)))$ s.t. $I = \hat{m}(x_i) \pm 1.96\cdot\widehat{\mathrm{s.e.}}(\hat{m}(x_i))$ Computation: For $\hat{\mathbf{Y}} = s(\mathbf{x},\mathbf{Y},h)$, $\mathcal{S}_{.,j} = s\left(\mathbf{x},\mathbf{e}_j,h\right)$ where $\mathbf{e}_j$ is unit vector.
Local Polynomial Regression: Solution to (for every x): $\widehat{\boldsymbol{\beta}}(x) = \underset{\beta\in\mathbb{R}^p}{\arg\min}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right)\left(Y_i - \beta_1 - \beta_2(x_i - x) - \ldots - \beta_p(x_i - x)^{p-1}\right)^2$. Often better at edges than locally constant Nadaraya-Watson.
Smoothing Splines: Minimizes $\sum_{i=1}^{n}(Y_i - m(x_i))^2 + \lambda\int m''(z)^2dz$ $\lambda \to 0$, $m_\lambda \to$ interpolating natural cubic spline and $\lambda = \infty$, least squares fit for linear regression (in general higher λ, smoother function) Sol. $\sum_{j=1}^{n}\beta_j B_j(x)$ (B_j: spline basis) is natural cubic spline with knots at predictors. We have $\Omega_{jk} = \int B_j''(z)B_k''(z)dz$, $\widehat{\boldsymbol{\beta}}_{n\times 1} = \left(B^\top B + \lambda\Omega\right)^{-1}B^\top\mathbf{Y}$, $\mathcal{S}_\lambda = B\left(B^\top B + \lambda\Omega\right)^{-1}B^\top$. There exists equiv. kernel with local bandwidth.

Nonparametric Regression

```
# N. Watson, Poly., Smoothing. Res in ks$y,ls$y,ss$y
ks <- ksmooth(x,y,kernel="normal",bandwidth=h)
ls <- loess(y ~ x)
dgf <- ls$trace.hat # retrieve dgf
ss <- smooth.spline(x,y,df = dgf)
# Locally optimal bandwidth
library(lokern)
lofit <- lokerns(cars$speed, cars$dist)
# Multinomial Regression
class_multinom <- multinom(Species ~ . , data = Iris)
```

```
# Calculating hat matrix
Snw <- matrix(0, nrow = 101, ncol = 101)
In <- diag(101) ## identity matrix
for(j in 1:101){ y <- In[,j]
    Snw[,j] <- ksmooth(x, y, kernel = "normal",
    ↪  bandwidth = 0.2, x.points = x)$y
}
```

Model Selection

Criteria for model selection (for linear models): Mallow's $C_p = \frac{1}{n}(RSS + 2d\cdot\hat{\sigma}^2)$, $AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\cdot\hat{\sigma}^2)$, $BIC = \frac{1}{n\hat{\sigma}^2}(RSS + \log(n)d\cdot\hat{\sigma}^2) = -2\cdot\log(\hat{L}) + d\cdot\log(n)$ where $\hat{L}$ is the maximized value of the likelihood of the model, $AdjR^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$.

Shrinkage Methods

Assume centered variable (no intercept). Also need so standardize data: **Ridge regression:** $\hat{\beta}_s^{ridge} = \text{argmin}_\beta RSS(\beta) + \lambda\|\beta\|_2^2 = (X^\top X + \lambda I)^{-1}X^\top y$ **Lasso:** $\hat{\beta}_s^{lasso}\text{argmin}_\beta RSS(\beta) + \lambda\|\beta\|_1$, **Elastic Net:** $\hat{\beta}_s^{elastic} = \text{argmin}_\beta RSS(\beta) + (1-\alpha)\lambda\|\beta\|_1 + \alpha\lambda\|\beta\|_2^2(\alpha = 1$: ridge, $\alpha = 0$: lasso),

Adaptive Lasso: Lasso with penalty weights:
$\hat{\beta}_s^{adapt.lasso}\text{argmin}_\beta RSS(\beta) + \lambda\sum_{j=1}^p w_j|\beta_j|$. Take a $\sqrt{n}$ consistent estimate $\hat{\beta}$ of β (e.g. least squares Choose $\gamma > 0$, then set $\hat{w}_j = 1/|\hat{\beta}|^\gamma$. This asymptotically selects the right covariates and has optimal estimation rate.

Relaxed Lasso: Estimate parameters by Lasso, define $\mathcal{M}_\lambda = \left\{1 \le k \le p \mid \hat{\beta}_k^\lambda \ne 0\right\}$ and let $\widehat{\boldsymbol{\beta}}^{\lambda,\phi} = \underset{\beta}{\arg\min} n^{-1}\sum_{i=1}^n\left(Y_i - \sum_{k\in\mathcal{M}_\lambda}\beta_k x_{i,k}\right)^2 + \phi\lambda\cdot\|\boldsymbol{\beta}\|_1$ (for $\phi = 1$, classical lasso, for $\phi = 0$, lasso only for variable selection) **Group Lasso:** Predictors are divided into L groups of size $p_1, ..., p_L$ s.t. $\sum p_i = p$. $\hat{\beta}_\lambda^{gr.lasso} = \text{argmin}_\beta RSS(\beta) + \lambda\sum_{l=1}^L\sqrt{p_l}\|\beta\|_2$. (if L=p, we get Lasso). Acts like Lasso on a group level. Useful if there are categorical variables with > 2 categories (put all corresponding dummy variables in a group).

Best subset: $\hat{\beta}_s^{subset} = \text{argmin}_{\beta,\|\beta\|_0\le s} RSS(\beta)$. 1) Fit M_0 (null model) = sample mean. 2) For $k = 1, ..., p$ fit $\binom{p}{k}$models that contain exactly k predictors and select best (smallest RSS): M_k. 3) Select best among $M_0, ..., M_p$ using CV or criteria.

Forward stepwise: 1) Fit M_0 2) For $k = 0, ..., p-1$ fit all $p-k$ models with 1 additional predictor and select best (smallest RSS): M_k. 3) Select best among $M_0, ..., M_p$ using CV or criteria.

Backward stepwise: 1) Fit M_p (full model). 2) For $k = p, p-1, ..., 1$: fit all k models that drop one perdictor in M_k. Choose best (smallest RSS): M_{k-1}. 3) Select best among $M_0, ..., M_p$ using CV or criteria.

Model Selection

```
# Lasso/Ridge
library(glmnet)
grid <- 10^seq(from=10,to=-2,length=100)
ridge <- glmnet(x[train,], y[train], alpha=0,
↪  lambda=gridm thres=1e-12)
lasso <- glmnet(x[train,], y[train], alpha=1,
↪  lambda=gridm thres=1e-12)
# Backward / Forward Selection
mortal.full <- lm(Mortality ~., data=mortality)
mortal.bw <- step(mortal.full, dir="backward")
mortal.empty <- lm(Mortality ~ 1, data = mortality)
mortal.fw <- step(mortal.empty, dir="forward",
↪  data=mortality,
scope = list(upper=mortal.full,lower=mortal.empty))
# All subsets regressions
library(leaps)
mortal.alls <- regsubsets(Mortality ~ .,data =
↪  mortality,nvmax=9)
```

Bootstrap

Sample uniform from data points with replacement, compute bootstrapped estimator: 1.) $Z_1^*, \ldots, Z_n^*$ i.i.d. $\sim \hat{P}_n$ 2.) Compute bootstrap estimator: $\hat{\theta}_n^{*1}, \ldots, \hat{\theta}_n^{*B}$ 3.) Repeat: $\hat{\theta}_n^{*1}, \ldots, \hat{\theta}_n^{*B}$ 4.) Calc. $\mathbb{E}^*\left[\hat{\theta}_n^*\right] \approx \frac{1}{B}\sum_{i=1}^B\hat{\theta}_n^{*i}$, $\text{Var}^*\left(\hat{\theta}_n^*\right) \approx \frac{1}{B-1}\sum_{i=1}^B\left(\hat{\theta}_n^{*i} - \frac{1}{B}\sum_{j=1}^B\hat{\theta}_n^{*j}\right)^2$ and empirical quantiles. For a large dataset $x_1, ..., x_n$ the probability that x_1 is contained in a random bootstrap dataset is:
$1 - (1 - 1/n)^n \approx 2/3$ (for large n, limit goes to $1 - 1/e$).

Bootstrap Consistency: For an increasing sequence a_n (often $\sqrt{n}$) where a_n^{-1} is the convergence rate of $\hat{\theta}_n$: $P(a_n(\hat{\theta}_n - \theta) \le x) - P^*(a_n(\hat{\theta}_n^* - \hat{\theta}_n) \le x) \to^P 0$ as $n \to \infty$. This holds when $\sqrt{n}(\hat{\theta}_n - \theta)$ is asympt. normal. Allows to estimate $\text{Bias}(\hat{\theta}_n) = E[\hat{\theta}_n] - \theta$ by $E^*[\hat{\theta}_n^*] - \hat{\theta}_n$. Can also estimate $\text{Var}(\hat{\theta}_n)$ by $\text{Var}^*(\hat{\theta}_n^*)$. **Confidence Interval:** Instead of taking quantiles of bootstrap distribution, following is often more appropriate (`type="basic"`): $\left[\hat{\theta}_n - q_{1-\alpha/2}, \hat{\theta}_n - q_{\alpha/2}\right] = \left[2\hat{\theta}_n - q_{1-\alpha/2}^*, 2\hat{\theta}_n - q_{\alpha/2}^*\right]$, where q_α^* is the α-bootstrap quantile of $\hat{\theta}_n^*$. **Normal Bootstrap CI:** Assumes $\hat{\theta}_n$ to be asympt. normal: $\hat{\theta}_n \pm Q_z(1-\alpha/2)\hat{sd}(\hat{\theta}_n)$ where $z \sim \mathcal{N}(0,1)$ and $\hat{sd}(\hat{\theta}_n) = \sqrt{(\text{Var}(\hat{\theta}_n^*))}$ (type="norm"), **Quantile Bootstrap CI:** not theoret. justified unless $\hat{\theta}_n$ is symm.: $[Q_{\theta_n^*}(\alpha/2), Q_{\theta_n^*}(1-\alpha/2)]$ (type="perc"). Same as *reversed quantile bootstrap CI* (basic) if $\hat{\theta}_n^* - \hat{\theta}_n$ is symm. around 0, **Bootstrap T:** Rely on $t = \frac{\hat{\theta}_n - \theta}{\hat{sd}(\hat{\theta}_n)}$ and $t* = \frac{\hat{\theta}_n^* - \hat{\theta}_n}{\hat{sd}(\hat{\theta}_n^*)}$ to be asympt. equal: $[\hat{\theta}_n - \hat{sd}(\hat{\theta}_n)\cdot Q_{t*}(1-\alpha/2), \hat{\theta}_n - \hat{sd}(\hat{\theta}_n)\cdot Q_{t*}(\alpha/2)]$ Note: $\hat{sd}(\hat{\theta}_n)$ is computed as above and $\hat{sd}(\hat{\theta}_n^*)$ is computed using a 2nd layer bootstrap. **Generalization Error:** 1.) Generate $\left(X_1^*, Y_1^*\right), \ldots, (X_n^*, Y_n^*)$ by resampling. 2.) Compute $\hat{m}^*(\cdot)$ based on resampled data. 3.) Evaluate $\text{err}^* = \frac{1}{n}\sum_{i=1}^n\rho(Y_i, \hat{m}^*(X_i))$ 4.) Estimate generalization error by $\frac{1}{B}\sum_{i=1}^B err^{*i}$. Can be over-optimistic, addressed by calculating out-of-bootstrap generalization error using only OOB samples ($\approx 36.8\%$) **Double Bootstrap:** Find α' s.t. $\mathbb{P}[\theta \in I^*(1-\alpha')] = 1 - \alpha$ with second level of bootstrap. E.g. CI: 1.) (M-times): Draw $Z_1^*, \ldots, Z_n^*$: a) Draw $Z_1^{**}, \ldots, Z_n^{**}$ B-times and compute bootstrap CI $I^{**}(1-\alpha)$. b) Evaluate whether $\hat{\theta}$ in $I^{**}(1-\alpha)$, i.e. $\text{cover}^*(1-\alpha) = \mathbf{1}_{\left[\hat{\theta}\in I^{**}(1-\alpha)\right]}$ 2.) Use $p^*(\alpha) := \frac{1}{M}\sum_{i=1}^M\text{cover}^{*i}(1-\alpha)$ as approx. for $\mathbb{P}^*\left[\hat{\theta} \in I^{**}(1-\alpha)\right]$ 3.) Vary α (in both steps) to find α'^* s.t. $p^*(\alpha'^*) = 1 - \alpha$ and use $\widehat{1-\alpha'} = 1 - \alpha'^*$. In total $M \times B$ comp. (M 1st, B 2nd level)

Parametric Bootstrap: Assume data is generated by some parametric distr. (e.g. $\mathcal{N}(\mu, \sigma^2)$), est. the param., then create new data sets from this distr. Works only well if distr. is approx. correct. E.g. linear model $Y_i = \beta^\top x_i + \varepsilon_i$ where ε_i i.i.d. $\mathcal{N}\left(0, \sigma^2\right)$: Simulate $\varepsilon_1^*, \ldots, \varepsilon_n^*$ from $\sim \mathcal{N}\left(0, \hat{\sigma}^2\right)$, calc. $Y_i^* = \hat{\beta}^\top x_i + \varepsilon_i^*$ to get bootstrap samples $\left(x_1, Y_1^*\right), \ldots, (x_n, Y_n^*)$

Model-based Bootstrap: Compute residuals $r_i = Y_i - \hat{m}(x_i)$ and center them, generate $\varepsilon_1^*, \ldots, \varepsilon_n^*$ i.i.d. $\sim \hat{P}_{\tilde{r}}$ from centered, construct $Y_i^* = \hat{m}(x_i) + \varepsilon_i^*$.

Bootstrap

```
library(boot)
sample(c(1:n), n, replace=T) # bootstrap sample
# f has args: (data, index)
# e.g. function(x, ind) {mean(x[ind]}
res.boot <- boot(Portfolio, f, R=1000)
res.boot$t0 # Estimates on original data
res.boot$t # Estimates on bootstrapped data
# Confidence intervals for variable i
boot.ci(res.boot, type="basic", index=i)
# Example to find all confidence intervals
tm <- function(x, ind) {mean(x[ind], trim = 0.1)}
tmv <- function(x, ind) {
  # bootstrap Var, required for the bootstrap T CI
  t2 <- var(boot(data=x[ind], statistic=tm, R=50)$t)
  return(c(tm(x, ind), t2)) }
res<-boot(data=..,statistic=tmv,R=10,sim="ordinary")
boot.ci(res, conf=0.95, type=c("basic","norm",
↪  "perc","stud"), var.t0=var(res.boot$t[,1]))
# Intervals by hand (t0: estimate, t: bootstrapped)
quantile.CI <- quantile(t,probs=c(0.025,0.975))
norm<-c(t0-qnorm(0.975)*sd(t),t0+qnorm(0.975)*sd(t))
reversed.CI <- t0-quantile(t-t0,probs=c(0.975,0.025))
# Parametric Bootstrap
# f1 is the bootstrap function: args (data)
# f2 returns a random dataset: args (data, mle)
res.boot <- boot(data, f1, R=1000, ran.gen=f2,
↪  sim="parametric",  mle=1/mean(x1))
```

Classification

Bayes Classifier: Optimal classifier: $\mathcal{C}_{\text{Bayes}}(x) = \underset{0\le j\le J-1}{\arg\max}\,\pi_j(x)$, where $\pi_j(x) = \mathbb{P}[Y = j \mid X = x]$

Linear Discriminant Analysis: Assume $(X \mid Y = j) \sim \mathcal{N}_p\left(\mu_j, \Sigma\right)$ and $\text{P}[Y = j] = p_j$, leads to: $\hat{\mu}_j = \frac{1}{n_j}\sum_{i;Y_i=j}X_i$, $\hat{\Sigma} = \frac{1}{n-J}\sum_{j=0}^{J-1}\sum_{i=1}^n\left(X_i - \hat{\mu}_j\right)\left(X_i - \hat{\mu}_j\right)^\top\mathbf{1}_{[Y_i=j]}$, decision function: $\hat{\delta}_j(x) = \left(x - \hat{\mu}_j/2\right)^\top\hat{\Sigma}^{-1}\hat{\mu}_j + \log\left(\hat{p}_j\right)$

Quadratic Discriminant Analysis: Non equal covariances, $\hat{\delta}_j(x) = -\log\left(\det\left(\hat{\Sigma}_j\right)\right)/2 - \left(x - \hat{\mu}_j\right)^\top\hat{\Sigma}_j^{-1}\left(x - \hat{\mu}_j\right)/2 + \log\left(\hat{p}_j\right)$ with $\hat{\Sigma}_j = \frac{1}{n_j - 1}\sum_{i=1}^n\left(X_i - \hat{\mu_j}\right)\left(X_i - \hat{\mu_j}\right)^\top\mathbf{1}_{[Y_i=j]}$. $Jp(p+1)/2$ parameters for matrices instead of $p(p+1)/2$

Logistic Regression: Model $\log\left(\frac{\pi(x)}{1-\pi(x)}\right) = g(x)$, where $\pi(x) = \mathbb{P}[Y = 1 \mid X = x]$. Linear: $g(x) = \sum_{j=1}^p\beta_j x_j$. MLE to fit, gradient descent minimizes $-\sum_{i=1}^n\left(Y_i\sum_{j=1}^p\beta_j x_{ij} - \log\left(\exp\left(\sum_{j=1}^p\beta_j x_{ij}\right) + 1\right)\right)$

Multiclass: 1.) J times class j vs. other classes, normalize $\hat{\pi}_j$ 2.) Multinomial for parametric linear logistic 3.) Model against reference class 0: $\log\left(\pi_j(x)/\pi_0(x)\right) = g_j(x)$
4.) Fit $\binom{J}{2}$ models for all pairs 5.) For ordered classes: $\text{logit}(P[Y \le k \mid x]) = \alpha_k + g(x)$ (`polr()`)

Classification

```
library(MASS)
c_l <- lda(x=d[,c("x1","x2")],grouping=d[,"y"])
c_q <- qda(x=d[,c("x1","x2")],grouping=d[,"y"])
predict(c_l, d)$class
fit <- glm(y~.,family=binomial) # Log. regression
fit <- glm(cbind(N, m - N) ~ age, family = binomial,
↪  data = heart)
# Pr(>|z|) contains P-val. for beta_j = 0
c_m <- multinom(y ~ . , data = Iris) # lib. nnet
# Multiclass approach 1.) with dummy enc.
levels(Iris1$Species) <- c("setosa", "not", "not")
Iris1$Species <- relevel(Iris1$Species, ref = "not")
fit.1 <- glm(Species ~ ., data = Iris1, family =
↪  "binomial")
levels(Iris2$Species) <- c("not", "versicolor",
↪  "not") # ...
predict(fit,new=d,type="response") # prob. new data
mean((predict(fit, type = "response") > 0.5) ==
↪  d.baby$Survival) # avg. in-sample accuracy
```

Flexible Regression and Classification

Additive Models: $g_{add}(x) = \mu + \sum_{j=1}^p g_j\left(x_j\right)$ with $\mathbb{E}\left[g_j\left(X_j\right)\right] = 0$. Backfitting: 1.) Use $\hat{\mu} = n^{-1}\sum_{i=1}^n Y_i$, start with $\hat{g}_j(\cdot) \equiv 0$. 2.) Cycle through j's and compute $\hat{\mathbf{g}}_j = \mathcal{S}_j\left(\mathbf{Y} - \hat{\mu}\mathbf{1} - \sum_{k\ne j}\hat{\mathbf{g}}_k\right)$ (where $\mathcal{S}_j$ is smoother) until $\hat{g}_j$ do not change much.
3.) Normalize: $\tilde{g}_j(\cdot) = \hat{g}_j(\cdot) - n^{-1}\sum_{i=1}^n\hat{g}_j\left(X_{ij}\right)$

MARS: Basis functions: $\left(x_j - d\right)_+ = \begin{cases} x_j - d & \text{if } x_j > d \\ 0 & \text{otherwise} \end{cases}$ and $\left(d - x_j\right)_+$. Model: $g(\mathbf{x}) = \mu + \sum_{m=1}^M\beta_m h_m(\mathbf{x})$ where h_m are combinations of basis. Algorithm: Greedily search for $h_{2r-1}(\cdot) = h_\ell(\cdot) \times \left(x_j - d\right)_+$, $h_{2r}(\cdot) = h_\ell(\cdot) \times \left(d - x_j\right)_+$ in every round (for some h_ℓ in model set) and estimate $\hat{\beta}$ by LS. Then backward deletion ("pruning").

Neural Networks: $g_k(\mathbf{x}) = f_0\left(\alpha_k + \sum_{h=1}^q w_{hk}\phi\left(\tilde{\alpha}_h + \sum_{j=1}^p\tilde{w}_{jh}x_j\right)\right)$, ϕ often sigmoid: $\phi(t) = \frac{\exp(t)}{1+\exp(t)}$. Centering/Scaling in practice important

Projection Pursuit Regression: $g_{\text{PPR}}(\mathbf{x}) = \mu + \sum_{k=1}^q f_k\left(\sum_{j=1}^p\alpha_{jk}x_j\right)$ with $\sum_{j=1}^p\alpha_{jk}^2 = 1$, $\quad\mathbb{E}\left[f_k\left(\sum_{j=1}^p\alpha_{jk}X_j\right)\right] = 0$.
Linear combinations $\sum_{j=1}^p\alpha_{jk}X_j$ are linear projections

Classification and Regression Trees: $g_{\text{tree}}(\mathbf{x}) = \sum_{r=1}^M\beta_r\mathbf{1}_{[\mathbf{x}\in\mathcal{R}_r]}$. Partitions chosen greedily (split evaluated at each data point), tree then pruned. Penalized goodness of fit used: $R_\alpha(\mathcal{T}) := R(\mathcal{T}) + \alpha \times \text{size}(\mathcal{T})$, α (or $\text{cp} = \alpha/R\left(\mathcal{T}_\emptyset\right)$) chosen via CV and 1 se rule (smallest tree s.t. error at most one se larger than minimal one)

Random Forest: Multiple bootstrap samples are drawn. For each, an unpruned tree is grown, but at each node, the best split is only chosen from randomly sampled features. Pred. by aggregation. Can get OOB estimate of error rate.

Flexible Regression & Classification

```
# Additive Models
library(mgcv)
fitA <- gam(y~s(x1)+s(x2)+...) # edf in summary ->
↪  estimated df, 1 corresponds to linear line
# MARS
library(earth)
Mfit <- earth(y ~ ., data = d)
# Neural Network
library(nnet)
Nfit <- nnet(y~.,data=d,size=3,decay=4e-
↪  4,linout=TRUE,skip=TRUE,maxit=500) # linout -> no
↪  prob.
# Projection Pursuit
fitP <- ppr(y~.,data=d,nterms=4) # nterms->number of
↪  ridge functions
# Classification / Regression Trees
library(rpart)
fitR <- rpart(y~.,data=d)
# Plotting the trees
require(rpart.plot)
prp(tree, extra=1, type=1)[tree$frame$yval])
# Random Forests
library(randomForest)
cs.bag <- randomForest(Sales ~ . , train.data,
↪  mtry=p-1, importance = TRUE) # Bagging
cs.forest <- randomForest(Sales ~ . , train.data,
↪  mtry=p/3, importance = TRUE) # Random Forest
importance(cs.forest) # Importance of predictors
# "Mean of squared residuals" (when calling
↪  cs.forest) contains OOB MSE
# Additive model with polynomials degree 2
form2 <- wrapFormula(form1, data = d.ozone.e,
↪  wrapString="poly(*,degree=2)")
fit2 <- lm(form2, data = d.ozone.e)
```

Bootstrap Aggregating (Bagging) & Boosting

Bagging for Regression: For data $(X_1, Y_1), ..., (X_n, Y_n)$ and base procedure $\hat{g}(\cdot) : \mathbb{R}^p \to \mathbb{R}$, take B bootstrap samples $\hat{g}_{bag}(x) = \frac{1}{B}\sum_{b=1}^B\hat{g}^{*b}(x)$ where $\hat{g}^{*b}$ is the estimate based on the b-th bootstrap sample. No pruning, since variance of single tree not a problem as we average. Linear predictions are the same under bagging, so only interesting for non-linear estimates. For regression can only improve or stay the same.

Bagging for Classification: $\hat{g}(\cdot) : \mathbb{R}^p \to \{1, ..., k\}$. $\hat{g}(x) = \text{argmax}_{k=1,...,K}\sum_{b=1}^B\mathbb{1}_{\hat{g}^{*b}(x)=k}$ (majority vote). Can also get class probability: $\hat{p}_k^{bag}(x) = \frac{1}{B}\sum_{b=1}^B\hat{p}_k^{*b}(x)$. Can also be formulated as $\hat{g}^{bag}(x) = \text{argmax}_{k=1,...,K}\hat{p}_k^{bag}(x)$ (better if interested in class probabilities, sometimes even helps accuracy). Bagging a good classifier can improve performance, but bagging a bad classifier can decrease performance.

Subagging: Instead of a bootstrap sample, only $m < n$ samples are drawn without replacement (i.e. no duplicates). $m = n/2$ is equivalent to bagging.

Out-of-Bag Error: Some bags have not trained on a particular sample. Can predict this only by the bags that have not been trained on it (should be $\sim 1/3$) for all samples and average to get a valid estimate for the test error.

	Tree	Bagging	Random Forest
Performance	-	+	++
Computation	+	-	+/-
Interpretation	+	-	-
Out-of-bag error	-	+	+

Boosting: Fit first function $\hat{g}_1(\cdot)$ and compute residuals. Obtain $\hat{g}_m$ by fitting function to residuals and set $\hat{f}_m(\cdot) = \hat{f}_{m-1}(\cdot) + \nu\hat{g}_m(\cdot)$, repeat for $m = 2, 3, \ldots, M$. Bias reduction technique (in contrast to bagging, which is variance reduction)