If linearity assumptions are met

$$\mathbf{Y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon},\ \mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0},\ \mathrm{Cov}(\boldsymbol{\varepsilon}) = \mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^\intercal] = \sigma^2 I_{n\times n}.$$

(i) $\mathbb{E}[\widehat{\boldsymbol{\beta}}] = \boldsymbol{\beta}$: that is, $\widehat{\boldsymbol{\beta}}$ is **unbiased**

(ii) $\mathbb{E}[\hat{\mathbf{Y}}] = \mathbb{E}[\mathbf{Y}] = X\boldsymbol{\beta}$ which follows from (i). Moreover, $\mathbb{E}[\mathbf{r}] = \mathbf{0}$.

(iii) $\mathrm{Cov}(\widehat{\boldsymbol{\beta}}) = \sigma^2(X^\intercal X)^{-1}$

(iv) $\mathrm{Cov}(\hat{\mathbf{Y}}) = \sigma^2 P$, $\mathrm{Cov}(\mathbf{r}) = \sigma^2(I-P)$

If in addition $\varepsilon_1, .., \varepsilon_n$ are uncorrelated and ~ N(0, σ^2) or large n

(i) $\widehat{\boldsymbol{\beta}} \sim \mathcal{N}_p(\boldsymbol{\beta}, \sigma^2(X^\intercal X)^{-1})$

(ii) $\hat{\mathbf{Y}} \sim \mathcal{N}_n(X\boldsymbol{\beta}, \sigma^2 P),\quad \mathbf{r} \sim \mathcal{N}_n(\mathbf{0}, \sigma^2(I-P))$

(iii) $\hat{\sigma}^2 \sim \frac{\sigma^2}{n-p}\chi^2_{n-p}.$

$$\widehat{\boldsymbol{\beta}} = (X^\intercal X)^{-1}X^\intercal \mathbf{Y}.$$

Scatterplot

```
col.partn <- 1*(partners==0) + 2*(partners==1) + 3*(partners==8)
plot(thorax, longevity, pch=pch.type, col=col.partn,
     ylim=range(longevity),xlim=range(thorax))
```

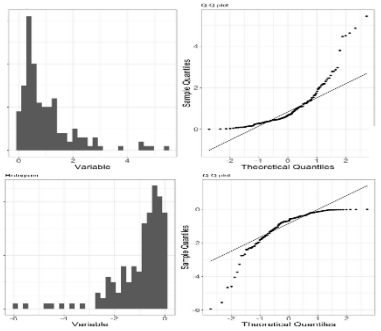
Does indicator variable affect thorax length?
Reject H_0 if Pr(>F) <= 0.05

```
fitfull<-lm(thorax~dummy.1.p+dummy.1.v+dummy.8.p+dummy.8.v)
fitintercept<-lm(thorax~1)
anova(fitintercept,fitfull)
```

Order data

```
data[order(data[,"People.per.Dr"],decreasing=T)[1:3],]
datanew <-data[complete.cases(data),]
predict(fit, newdata=newcountry, interval="predict")
predict(fit, newdata=newcountry, interval="confidence")
```

Q-Q plot



Mallows Cp statistic

The MSE can be estimated by:

$$n^{-1}SSE(\mathcal{M}) - \hat{\sigma}^2 + 2\hat{\sigma}^2|\mathcal{M}|/n,$$

which is prop. to Cp(M):

$$C_p(\mathcal{M}) = \frac{SSE(\mathcal{M})}{\hat{\sigma}^2} - n + 2|\mathcal{M}|$$

Cp in R

(note: object is of type fit, and sigma can be found using sigma(fit))

```
Cp <- function(object,sigma){
  res <- residuals(object)
  n <- length(res)
  p <- n - object$df.residual
  SSE <- sum(res^2)
  SSE / sigma^2 - n + 2 * p

# choose sigma
sigma <- summary(fit5)$sigma
```

Forward selection (in case p, the nr. of predictors vars. Is too large for exhaustive search):
1) start with smallest model
2) add predictor which reduces the MSE the most
3) repeat 2 until all predictors or large nr. of predictors selected (now a seq. models is produced)
4) choose the model in the seq. which has smallest Cp statistic.
Backward selection is similar.

Note that the linear regression model does not assume a normal distribution for the predictors, but a skewed distribution and outliers often result in regression solutions that are largely determined by very few points.

```
library(lattice)
splom(~mortality[,c("Mortality","Pop","HC","NOx","SO2")],pscales=0)
```

Forward and backward selection in R

```
mortal.bw <- step(mortal.full, dir="backward")
```

```
mortal.fw=step(mortal.empty,
  dir="forward", data=mortality,
  scope = list(upper=mortal.full,
               lower=mortal.empty))
```

Tukey Anscombe is residuals versus fitted values. Q-Q is empirical quantiles vs standard normal quantiles.

For lm, Tukey Anscombe is given by
`plot(fit,which =1)`
Whereas Q-Q plot is given by
`plot(fit,which =2)`

Shortcut for LOOCV (using hat matrix)

$$n^{-1}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{m}(X_i)}{1-\mathcal{S}_{ii}}\right)^2$$

Linearity assumptions (for LS regression)

The linear regression equation is correct. This means: $\mathbb{E}[\varepsilon_i] = 0$ for

All x_i's are exact. This means that we can observe them perfectly.

The variance of the errors is constant ("homoscedasticity"). This $\mathrm{Var}(\varepsilon_i) = \sigma^2$ for all i.

The errors are uncorrelated. This means: $\mathrm{Cov}(\varepsilon_i, \varepsilon_j) = 0$ for all $i \neq j$.

The errors $\{\varepsilon_i;\ i = 1,\ldots,n\}$ are jointly normally distributed. This that also $\{Y_i; i = 1,\ldots,n\}$ are jointly normally distributed.

LOOCV in R

```
loocv <- function(reg.data, reg.fcn)
{
  loo.reg.value <- function(i, reg.data, reg.fcn)
    return(reg.fcn(reg.data$x[-i], reg.data$y[-i],
                   reg.data$x[i]))
n <- nrow(reg.data)
loo.values <- sapply(1:n, loo.reg.value,
                     reg.data, reg.fcn)
mean((reg.data$y - loo.values)^2)
```

Non-parametric regression in R

Nadaraya-Watson (normal kernel)

```
reg.fcn.nw <- function(reg.x, reg.y, x)
  ksmooth(reg.x, reg.y, x.point = x,
          kernel = "normal", bandwidth = h)$y
```

Local polynomial

```
reg.fcn.lp <- function(reg.x, reg.y, x) {
  lp.reg <- loess(reg.y ~ reg.x,
                  enp.target = df.nw, surface = "direct")
  predict(lp.reg, x)
}
```

Smoothing splines

```
reg.fcn.ss <- function(reg.x, reg.y, x)
{
  ss.reg <- smooth.spline(reg.x, reg.y, spar = est.ss$spar)
  predict(ss.reg, x)$y
}
```

Bootstrap in R

```
library(boot)
bs <- function(formula, data, indices) {
  d <- data[indices,] # allows boot to select sample
  fit <- lm(formula, data=d)
  return(coef(fit))
}
# bootstrapping with 1000 replications
results <- boot(data=mtcars, statistic=bs,
                R=1000, formula=mpg~wt+disp)
plot(results, index=1) # intercept (histogram and quantile)
plot(results, index=2) # wt
plot(results, index=3) # disp
boot.ci(results, type="bca", index=1) # intercept
boot.ci(results, type="bca", index=2) # wt
boot.ci(results, type="bca", index=3) # disp
```

"basic" = "Reversed quantile"
"norm" = "Normal"
"perc" = "Quantile"

Expected size of out-of-bootstrap sample [roughly 1/3 of points will be out of sample]:

$$\mathbb{E}^*[|\mathcal{L}^*_{out}|] = \mathbb{E}^*[\sum_{i=1}^{n}\mathbf{1}_{[Z_i\in\mathcal{L}^*_{out}]}] = n\mathbb{P}^*[Z_i \in \mathcal{L}^*_{out}] \approx 0.368n.$$

Baves Risk

$$\mathbb{P}[\mathcal{C}_{Bayes}(X_{new}) \neq Y_{new}].$$

Baves Classifier

$$\mathcal{C}_{Bayes}(x) = \underset{0\le j\le J-1}{\arg\max}\ \pi_j(x).$$

LDA Classifier

$$\hat{\mu}_j = \frac{1}{n_j}\sum_{i;Y_i=j} X_i, \qquad \hat{\Sigma} = \frac{1}{n-J}\sum_{j=0}^{J-1}\sum_{i=1}^{n}(X_i-\hat{\mu}_j)(X_i-\hat{\mu}_j)^\intercal \mathbf{1}_{[Y_i=j]}$$

Prediction of LDA found by taking argmax:

$$\hat{\delta}_j(x) = x^\intercal\hat{\Sigma}^{-1}\hat{\mu}_j - \hat{\mu}_j^\intercal\hat{\Sigma}^{-1}\hat{\mu}_j/2 + \log(\hat{p}_j) = (x-\hat{\mu}_j/2)^\intercal\hat{\Sigma}^{-1}\hat{\mu}_j + \log(\hat{p}_j).$$

Decision boundary between class 0 and 1:

$$B = \{x\,|\,\hat{\delta}_0(x) = \hat{\delta}_1(x)\} = \{x|(x-z)^\intercal w = 0\}$$

if the covariance matrix ∑ is diagonal and const. then w is parallel to the line connecting μ_0 and μ_1

QDA Classifier (more easily overfits if p is big)

$$\hat{\Sigma}_j = \frac{1}{n_j-1}\sum_{i=1}^{n}(X_i-\hat{\mu}_j)(X_i-\hat{\mu}_j)^\intercal\mathbf{1}_{[Y_i=j]}$$

$$\hat{\delta}_j(x) = -\log(\det(\hat{\Sigma}_j))/2 - (x-\hat{\mu}_j)^\intercal\hat{\Sigma}_j^{-1}(x-\hat{\mu}_j)/2 + \log(\hat{p}_j)$$

Bootstrapping LDA/QDA (both part of MASS library)

```
index <- matrix(sample.int(n, n*B, replace = TRUE), nrow = n, ncol = B)
fit_lda <- vector("list", B)
for(i in 1:B) {
  ind <- index[, i]
  fit_lda[[i]] <- lda(x = Iris[ind , c("Petal.Length", "Petal.Width")],
                      grouping = Iris[ind , "Species"])
}
```

Decision boundary plot in R

```
xp <- seq(min(x[, 1]), max(x[, 1]), length = len)
yp <- seq(min(x[, 2]), max(x[, 2]), length = len)
grid <- expand.grid(xp, yp)
Z <- predict(object, grid, ...)
zp <- as.numeric(Z$class)
zp <- Z$post[, 3] - pmax(Z$post[, 2], Z$post[, 1])
contour(xp, yp, matrix(zp, len),
        add = TRUE, levels = 0, drawlabels = FALSE, col = colcont)
```

(Linear) logistic regression

$$\log\left(\frac{\pi(x)}{1-\pi(x)}\right) = \sum_{j=1}^{p}\beta_j x_j$$ LHS is logit transform of π

$$\log\left(\frac{\pi_j(x)}{1-\pi_j(x)}\right) = g_j(x) = \sum_{r=1}^{p}\beta_r^{(j)}x_r$$ For J>2

Likelihood (for Bernoulli x then for Binomial)

$$L(\beta;(x_1,Y_1),\ldots,(x_n,Y_n)) = \prod_{i=1}^{n}\pi(x_i)^{Y_i}(1-\pi(x_i))^{1-Y_i},$$

$$L(\beta;(x_1,m_1,N_1),\ldots,(x_n,m_n,N_n)) = \prod_{i=1}^{n}\binom{m_i}{N_i}\pi(x_i)^{N_i}(1-\pi(x_i))^{m_i-N_i}$$

```
fit <- glm(cbind(N, m - N) ~ age,
    family = binomial, data = heart)
```

```
predict(fit, new = data.frame(age =
      new.age), type = "response")
```

```
fit <- glm(Survival ~ ., data =
    d.baby, family = "binomial")
```

will give the probability of heart disease (N/m)

Multinomial logistic regression

```
m <- multinom(Species ~ . , data = Iris)
ZP <- predict(object, newdata = grid, type = "probs")
```

where the probabilities of each class are stored in the columns (columns same order as levels (...)) but we can also achieve logistic regression by using dummy encoding and glm:

```
levels(Iris1$Species) <- c("setosa",
                           "not", "not")
Iris1$Species <- relevel(Iris1$Species,
                         ref = "not")
fit.1 <- glm(Species ~ ., data = Iris1,
             family = "binomial")
```

ROC and Misclassification graph in R

```
require(ROCR)
fit <- glm(Survival ~ ., data = d.baby,
           family = "binomial")
pred <- prediction(fit$fitted.values,
                   d.baby$Survival)
perf <- performance( pred, "tpr", "fpr" )
plot(perf, main = "title")
```

which plots the TPR vs FPR. To plot the misclassification rate as a function of threshold prob. (i.e., after this it is considered positive)

```
perf.cost <- performance(pred, "cost")
plot(perf.cost, main = title)
```

K-Fold CV in R

```
folds <- sample(cut(seq(1, n), breaks = K,
        labels = FALSE), replace = FALSE)
for (i in 1:K){
  test.ind <- which(folds == i)
  df.train<- d.baby[-test.ind,]
  df.test <- d.baby[ test.ind,]
  ...}
```

To plot the average ROC (across all folds) initialize a vector of lists and assign them as follows:

```
all.y.true[[i]] <- y.true
all.y.pred[[i]] <- y.pred
```

And then pred.cv <- prediction (all.y.pred, all.y.true) and proceed as normal (but add avg = "threshold" for ROC and avg = "vertical" for misclassification) use add = T to add them to same plot. One can specify the cost for FN and FP using cost.fp = 0.8, cost.fn = 1.2 as args to performance().

Poly. regression using a polynomial of degree d (no interaction term):

$$y_i = \beta_0 + \beta_{1,1}x_{i1} + \beta_{1,2}x_{i1}^2 + \ldots\beta_{1,d}x_{i1}^d + \ldots + \beta_{p,1}x_{ip} + \beta_{p,2}x_{ip}^2 + \ldots\beta_{p,d}x_{ip}^d + \epsilon_i.$$

```
require(sfsmisc)
form1 <- as.formula("logupo3~.")
form3 <- wrapFormula(form1, data =
  d.ozone.e, wrapString="poly(*,degree=d)")
fit3 <- lm(form3, data = d.ozone.e)
```

Generalized additive model in R

```
require(mgcv)
form1 <- as.formula("logupo3~.")
gamForm <- wrapFormula(form1, data = d.ozone.e)
g1 <- gam(gamForm, data = d.ozone.e)
```

Alternatively,

```
fitA <- gam(O3 ~ s(vdht)+ s(wind)+
  s(humidity)+ s(temp)+ s(ibht)+
  s(dgpg)+ s(ibtp)+ s(vsty)+ s(day),
  data = d.ozone)
```

To plot the splines do:

```
par(mfrow=c(3,3))
plot(g1, shade = T)
```

To plot the same thing but for lm use:

```
par(mfrow=c(3,3))
termplot(fit5, partial.resid=TRUE,
 rug=FALSE, se=TRUE,
 col.res='#C0C0C050', pch=19)
```

R^2 describes the proportion of the total variation of the response Y around its mean Y bar which is explained by the regression Y hat.

$$R^2 = \frac{\|\hat{\mathbf{Y}} - \overline{\mathbf{Y}}\|^2}{\|\mathbf{Y} - \overline{\mathbf{Y}}\|^2}.$$

In regression we have: (which allows for writing R^2 differently)

$$\|\mathbf{Y} - \overline{\mathbf{Y}}\|^2 = \|\hat{\mathbf{Y}} - \overline{\mathbf{Y}}\|^2 + \|\mathbf{Y} - \hat{\mathbf{Y}}\|^2$$

The following removes col. upo3:

```
d.ozone <- subset(ozone, select=-upo3)
d.ozone <- subset(transform(d.ozone, "logO3" = log(O3)), select = -O3)
```

adds log(03) as a col. and removes 03

(In-sample) R^2 in R

```
r2 <- function(fitfn, formula = logO3 ~ . ,
               data = d.ozone.es, ...)
{
  modFrame <- model.frame(formula, data = data)
  Y <- model.response(modFrame)
  #calculate ssr and sst
  fit <- fitfn(formula=formula, data = data, ...)
  ssr <- sum((Y - predict(fit, modFrame))^2)
  sst <- sum((Y-mean(Y))^2)
  1-ssr/sst
}
```

(Cross-validated LOOCV) R^2 in R

```
cv_r2 <- function(fitfn, formula = logO3 ~ . , data =
                  d.ozone.es, ..., trace = TRUE)
{
  modFrame <- model.frame(formula, data = data)
  n <- nrow(data)
  ssr <- 0
  if(trace) cat(" j = ")
  for(j in 1:n) {
    ## Fit without 'j' :
    fit <- fitfn(formula=formula, data = data[-j ,],
                 ..., trace = FALSE)
    ## Evaluate at 'j':
    ssr <- ssr + (model.response(modFrame)[j] -
                  predict(fit, modFrame[j,]))^2
  }
  #calculate sst
  fit <- fitfn(formula=formula, data = data, ...)
  sst <- sum((model.response(modFrame) -
              mean(model.response(modFrame)))^2)
  if(trace) cat("\n")
  1-ssr/sst
}
```

MARS in R

```
require("earth")
Mfit <- earth(Volume ~ ., data = trees)
summary(Mfit)
```

```
predict(Mfit,
        data.frame(Girth= 5:15, Height= seq(60,80, length=11)))
```

We can then find their R^2 as follows:

```
r2(earth, formula = logO3 ~ . ,
   data = d.ozone.es, degree = 3)
cv_r2(earth, formula = logO3 ~ . ,
   data = d.ozone.es, degree = 3)
```

MARS fits a function of the form

$$g(x) = \mu + \sum_{m=1}^{M}\beta_m h_m(\mathbf{x}) = \sum_{m=0}^{M}\beta_m h_m(\mathbf{x}),$$

1. Start with the function $h_0(\mathbf{x}) = 1$. Initialize the model set $\mathcal{M} = \{h_0(\cdot) = 1\}$. Fit the function h_0 by least squares regression, yielding the estimate $\hat{\mu} = n^{-1}\sum_{i=1}^{n} Y_i$.
2. For $r = 1, 2, \ldots$ do the following:
   Search for the best pair of functions $((h_{2r-1}(\cdot), h_{2r}(\cdot))$ which are of the form
   $$h_{2r-1}(\cdot) = h_\ell(\cdot)\times(x_j - d)_+,$$
   $$h_{2r}(\cdot) = h_\ell(\cdot)\times(d - x_j)_+, \qquad (7.1)$$
   for some h_ℓ in the model set $\mathcal{M}$ which does not contain x_j[3] and some basis functions in $\mathcal{B}$. The best pair of functions is defined to be the one which reduces residual sum of squares most. The model fit is then
   $$\hat{y}(\mathbf{x}) = \hat{\mu} + \sum_{m=1}^{2r}\hat{\beta}_m h_m(\mathbf{x}),$$
   where the coefficients $\hat{\beta}_m$ are estimated by least squares.
   Enlarge the model set in every iteration (with index r) by
   $$\mathcal{M} = \mathcal{M}_{old}\cup\{h_{2r-1}(\cdot), h_{2r}(\cdot)\},$$
   with the functions h_{2r-1}, h_{2r} from (7.1).
3. Iterate step 2 until a large enough number of basis functions $h_m(\cdot)$ has been fitted.
4. Do backward deletion ("pruning"), allowing to remove single functions from a pair $h_{2r-1}(\cdot), h_{2r}(\cdot)$; i.e., of all single basis functions, delete the one which increases the residual sum of squares the least.
5. Stop the backward deletion by optimizing a GCV score.

summary (mars fit) gives the importance of each variable. If we fitted a model of degree 2, then we can get the graphs of the response variables as a function of each predictor variable as follows:

```
plotmo(fit, degree2=FALSE,
    caption="main effects")
```

We can also get the 3D plots (effect of the interaction of 2 terms on the response) using

```
plotmo(fit, degree1=FALSE,
    caption="interactions")
```

Neural Networks (1 Layer with q units)

$$g_k(\mathbf{x}) = f_0\left(\alpha_k + \sum_{h=1}^{q} w_{hk}\phi(\tilde{\alpha}_h + \sum_{j=1}^{p}\tilde{w}_{jh}x_j)\right)$$

where $\phi(t) = \frac{\exp(t)}{1+\exp(t)}$

and f_0 is typically identity for regression and sigmoid for classification. We can add a linear regression component (skip conxn):

$$g(\mathbf{x}) = f_0\left(\alpha + \sum_{j=1}^{p} w_{j,lin}x_j + \sum_{k=1}^{q} w_k\phi(\alpha_k + \sum_{j=1}^{p} w_{jk}x_j)\right)$$

Make sure to scale the data (except for response) to avoid getting stuck in the flat regions of the sigmoid function. Weight decay is good as it reduces the dependency on the starting values and reduces the importance of choosing the # of hidden units

```
Nfit <- nnet(log.O3 ~ . , data = sc.ozone,
             size = 3,       # chooses 3 h
             decay = 4e-4,   # "weight dec
             linout = TRUE,  # linear regr
             skip = FALSE,   # a N.N. with
             maxit = 500)
```

To get the weights use summary(Nfit). R prefixes input variables with *i* and hidden unit with *h*. To get the error use sum(residuals(Nfit)^2)

`optim(*, method="BFGS")`
is equivalent to using a neural network (note results may differ due to different seed).

Optimization in R

Assume we have a fct. like this for the neg. log likelihood which we want to minimize (case of binomial distribution of response and logist. regr.),

```
neg.ll <- function(beta, data){
  - sum(log(choose(data$m, data$N)) +
        data$N * g(beta,data$age) -
        data$m * log( 1 + exp(g(beta,data$age))))
}
```

Then the following will return the vector beta (same as that of logistic regression),

```
optim(c(0, 0), neg.ll, data = heart)$par
```

Trees (CART)

1. Start with $M = 1$ subset, $\mathcal{P} = \{\mathcal{R}\} = \{\mathbb{R}^p\}$.
2. Refine $\mathcal{R}$ into $\mathcal{R}_{left} \cup \mathcal{R}_{right}$ where:

$$\mathcal{R}_{left} = \mathbb{R} \times \mathbb{R} \times \ldots \times (-\infty, d] \times \mathbb{R} \ldots \times \mathbb{R},$$
$$\mathcal{R}_{right} = \mathbb{R} \times \mathbb{R} \times \ldots \times (d, \infty) \times \mathbb{R} \ldots \times \mathbb{R},$$

where one of the axes is split at the split point d, where d is from the finite set of mid-points between observed values. The search for the axes to split and the split point d are determined such that the negative log-likelihood is maximally reduced with the refinement (search over $j \in \{1, \ldots, p\}$ and $d \in \{\text{mid-points of observed values}\}$). Build the new partition $\mathcal{P} = \{\mathcal{R}_1, \mathcal{R}_2\}$ with $\mathcal{R}_1 = \mathcal{R}_{left}$, $\mathcal{R}_2 = \mathcal{R}_{right}$.

3. Refine the current partition $\mathcal{P}$ as in step 2 by refining **one** of the partition cells from the current partition $\mathcal{P}$. That is, we search for the best partition cell to refine which includes a search as in step 2 for the best axes to split and the best split point. Then, we up-date the partition:

$\mathcal{P} = \mathcal{P}_{old} \setminus$ partition cell selected to be refined $\cup$ {refinement cells $\mathcal{R}_{left}, \mathcal{R}_{right}$}.

4. Iterate step 3 for a large number, $M = M_{max}$, of partition cells.
5. Backward deletion: prune the tree (see below) until a reasonable model size, typically determined via cross-validation, is achieved.

The cost function is: (note that the size of the tree is the # of leaves, n, and that for a binary tree # of nodes = 2n - 1). For regression, R(T), can be -2*(log likelihood)=sum of squares as in the case for regression or the misclassification rate for classification

$$R_\alpha(T) := R(T) + \alpha \times \text{size}(T), \quad \alpha \geq 0$$

Pros of trees: Interpretable, can deal with missing values using "surrogate" split, does automatic variable selection.
Cons: piecewise constant (for probability estimate or regression). Unstable split: if a split is "wrong", then all of the splits below it will be "wrong".

CART in R

```
library(rpart)
tree <- rpart(Class ~ ., data = data,
control = rpart.control(cp = 0.0, minsplit = 30))
```

cp stands for cost complexity pruning (i..e penalizes large trees -> higher cp means shorter trees. cp = alpha/R(0)).

Plotting can be done as follows (note that each node will display J numbers for each category which falls into this region in the order given by levels(data$var)):

```
require(rpart.plot)
prp(tree, extra=1, type=1,
  box.col=c('pink', 'palegreen3',
'lightsteelblue 2','lightgoldenrod 1')[tree$frame$yval])
```

One-standard-error rule: Find the model with the lowest cross-validation error. Then choose the simplest model, which is at most one standard-deviation worse than that model.
To prune the tree according to this rule, we need cross validation error of the tree, use (ignore relative error column): `tree$cptable`

```
misclass.sample <- function(data, ind.training, ind.test)
{
 tree <- rpart(Class ~ ., data = data[ind.training, ],
               control = rpart.control(cp=0.0,minsplit = 30))

 ## choose optimal cp according to 1-std-error rule:
 cp <- tree$cptable
 min.ind <- which.min(cp[,"xerror"])
 min.lim <- cp[min.ind, "xerror"] + cp[min.ind, "xstd"]
 cp.opt <- cp[(cp[,"xerror"] < min.lim),"CP"][1]

 tree.sample <- prune.rpart(tree, cp=cp.opt)

 mean(data$Class[ind.test] != predict(tree.sample,
                                      newdata = data[ind.test, ], type = "class"))
}
```

Bootstrapping trees:

```
B <- 1000
n <- nrow(data)
boot.err <- function(data, ind)
  misclass.sample(data, ind, 1:nrow(data))
boot.samples <- replicate(B,
  boot.err(data, sample(1:n, replace = TRUE)))
errboot <- mean(boot.samples)
```

Ridge Regression

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} + \varepsilon_i, \quad i = 1, \ldots, n.$$

can be rewritten as

$$Y_i = \beta_0' + \beta_1(x_{i,1} - \overline{x_{\cdot,1}}) + \ldots + \beta_p(x_{i,p} - \overline{x_{\cdot,p}}) + \varepsilon_i.$$

which is beneficial for ridge since we do not want to penalize the intercept term (β_0' is simply Y bar, the avg of the Y values).
Ridge regression is formalized as:

$$\tilde{\beta}(s) = \underset{\|\beta\|^2 \leq s}{\arg\min} \|Y - X\beta\|^2$$

which is equivalent to solving [via Lagrange mult.]:

$$\hat{\beta}^*(\lambda) = \underset{\beta}{\arg\min} \{\|Y - X\beta\|^2 + \lambda \|\beta\|^2\}$$

$$\hat{\beta}^*(\lambda) = (X^T X + \lambda \mathbb{I})^{-1} X^T Y$$

Note that XtX + λI is always inv. for λ > 0

Note that the ridge estimate for the coefficient vector beta will be biased, but has less variance than the LS estimate (obtained when λ=0)

$$\hat{y}_{ridge}(\lambda) = \sum_{i=1}^{n} \frac{d_{ii}^2}{d_{ii}^2 + \lambda} \mathbf{u}_i \mathbf{u}_i^T \mathbf{y}$$

derived from SV Decomposition of X as UDV^T where U is n*n, V is p*p, and both are orthogonal. D is n*p diag.

Lasso Regression

Unlike ridge regression, solved by differentiating matrices, lasso regression requires quadratic programming. Has the goal of variable selection.

Several extensions are elastic net regression:

$$L(\lambda_1, \lambda_2, \beta) = \|Y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|$$

which is equivalent to solving the problem:

$$\hat{\beta} = \arg\min_{\beta} \|Y - X\beta\|^2, \quad \text{subject to } (1-\alpha)\|\beta\|_1 + \alpha\|\beta\|^2 \leq t \text{ for some } t.$$

where, $\alpha = \lambda_2 / (\lambda_1 + \lambda_2)$

and the constraint is called **elastic net penalty**

For all α in [0,1), the elastic net penalty is singular (no first derivative) at 0 and it is strictly convex for all α > 0, note that the lasso penalty (α=0) is convex, but not strictly.

Another extension is the **adaptive lasso** (shown to have oracle properties)

$$\arg\min_{\beta} \|Y - \sum_{j=1}^{p} \beta_j \mathbf{x}_j\|^2 + \lambda \cdot \sum_{j=1}^{p} w_j |\beta_j|$$

Suppose β hat is sqrt(n) consistent estimator of β (for example the LS estimate), pick a γ>0 and define w_j hat as $1/|\beta_j \text{hat}|$ then adaptive lasso is given by $\hat{\beta}^*_{(n)} = \arg\min_{\beta} \|Y - \sum_{j=1}^{p} \beta_j \mathbf{x}_j\|^2 + \lambda \cdot \sum_{j=1}^{p} \hat{w}_j |\beta_j|$

which is a strictly convex optimization problem (i.e. has no local minima) and can be solved efficiently

The final extension is the **relaxed lasso regression:**

Let $\hat{\beta}^{\mathcal{S}}(\lambda) = (\hat{\beta}^{\mathcal{S}}_1, \ldots, \hat{\beta}^{\mathcal{S}}_p)^T$ be the lasso estimated parameter vector of (7.10), and define

$$\mathcal{M}_\lambda = \{1 \leq k \leq p \mid \hat{\beta}^{\mathcal{S}}_k \neq 0\}, \tag{7.15}$$

the set of "significant" variables.
The *relaxed lasso estimator* is then defined for $\lambda \in [0, \infty)$ and $\phi \in [0, 1]$ as

$$\hat{\beta}^{\mathcal{R}, \lambda, \phi} = \underset{\beta}{\arg\min} \; n^{-1} \sum_{i=1}^{n} (Y_i - \sum_{k \in \mathcal{M}_\lambda} \beta_k x_{i,k})^2 + \phi\lambda \cdot \|\beta\|_1. \tag{7.16}$$

Lasso and ridge in R

Find Numerical Columns in R
`(isNum <- sapply(d.Wage, is.numeric))`

Combine two formulas and generate all deg <=3 interactions in R

```
require(sfsmisc) # for wrapFormula()
d.Wage <- subset(Wage, select = -region )

## logwage ~ poly(age, 3) + poly(year, 3)
(fpoly <- wrapFormula(logwage ~ .,
  data=d.Wage[, !isN.x], wrapString="poly(*,degree=3)"))

## logwage ~ year + age + maritl + race +
## + education + jobclass + health + health_ins
(ffac <- formula(terms(logwage ~ . -wage,
        data=d.Wage, simplify=TRUE)))

## combine last two formulas
(ff <- formula(paste(deparse1(ffac), "+", deparse1(fpoly[[3]]))))
## all interactions of degree 3 and less (eg. poly(age, deg = 3):year:race)
ff <- update(ff, wage ~ .^3)
```

Generate model frame [data frame containing all terms required by a formula] and design matrix:

```
mf <- model.frame(ff, d.Wage)
str( mm <- model.matrix  (mf, data=d.Wage) )## 3000 x 1375
```

Ridge and lasso fitting (and plotting coefficients vs log lambda)

```
require(glmnet)
f.ridge <- glmnet(mm, y, alpha=0)
f.lasso <- glmnet(mm, y, alpha=1)
op <- par(mfrow=c(1,2)) # op: save previous settings
plot(f.ridge, xvar="lambda", main="Ridge Regression")
plot(f.lasso, xvar="lambda", main="Lasso Regression")
par(op) # revert to previous
```

One split: high variance, high bias LOOCV: high variance, low bias LDOCV: low variance, higher bias KFCV: higher bias, not sure if bias < LOOCV

CV for Ridge/Lasso and Saving Objects

```
sFile <- "cv-elastn.rds"
if(!file.exists(sFile)) {
  set.seed(1)
  print(system.time(
    cv.eln <- cv.glmnet(mm, y, alpha=0.5, nfol
  ))
  ##    user  system elapsed
  ## 93.972   0.117  94.268  -- 1.5 min
  ## now save it in the "safe-file"
  saveRDS(cv.eln, sFile)
  ##
} else { ## read pre-computed result:
  cv.eln <- readRDS(sFile)
}
#

plot(cv.eln)
cv.eln$lambda.1se
```

Set precision in R

```
formatC(x, digits = 4, format = "f")
```

Formulas:

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N-1}} \qquad \text{SE} = \frac{\sigma}{\sqrt{n}}$$

$$Bias(\hat{\theta}) = E(\hat{\theta}) - \theta$$

Legend in R

```
legend("topleft",c("1 pregnant","8 pregnant","1 virgin",
       "8 virgin","0 partners"),
       pch=c(1,2,1,2,3),col=c(2,2,3,3,1))
```

Z value = (μ - μ_0)/SE

where μ_0 is the mean under the null hypothesis (0 for coefficient estimates) and μ is the measured mean

(Square of) residual standard error

$$\hat{\sigma}^2 = \frac{1}{n-p} \sum_{i=1}^{n} r_i^2, \quad \mathbb{E}[\hat{\sigma}^2] = \sigma^2$$

Linearity assumptions violations

In case of violations of item 3, we can use weighted least squares instead of least squares. Similarly, if item 4 is violated, we can use generalized least squares. If the normality assumption in 5 does not hold, we can use robust methods instead of least squares. If assumption 2 fails to be true, we need corrections known from "errors in variables" methods. If the crucial assumption in 1 fails, we need other models than the linear model.

If X has full rank p, then *X* is p-dimensional subspace:

$$X\hat{\beta} \text{ is the orthogonal projection of } \mathbf{Y} \text{ onto } \mathcal{X}.$$

$$\mathbf{r} = (I - P)\mathbf{Y}$$ which lies on an n-p dimensional subspace (complement space of *X*)

$$\text{Var}(r_i) = \sigma^2(1 - P_{ii}).$$

Multiple regression vs single predictor

Simple least squares regressions on single predictor variables yield the multiple regression least squares solution, *only* if the predictor variables are orthogonal.

ANOVA can be used to see if ANY variable is signif.

	sum of squares	degrees of freedom	mean square	E[mean square]
regression	$\|\hat{\mathbf{Y}} - \overline{\mathbf{Y}}\|^2$	$p-1$	$\|\hat{\mathbf{Y}} - \overline{\mathbf{Y}}\|^2/(p-1)$	$\sigma^2 + \frac{\|\mathbb{E}[\mathbf{Y}] - \mathbb{E}[\overline{\mathbf{Y}}]\|^2}{p-1}$
error	$\|\mathbf{Y} - \hat{\mathbf{Y}}\|^2$	$n-p$	$\|\mathbf{Y} - \hat{\mathbf{Y}}\|^2/(n-p)$	σ^2
total around global mean	$\|\mathbf{Y} - \overline{\mathbf{Y}}\|^2$	$n-1$	–	–

ANOVA relies on the F-statistic:

$$F = \frac{\|\hat{\mathbf{Y}} - \overline{\mathbf{Y}}\|^2/(p-1)}{\|\mathbf{Y} - \hat{\mathbf{Y}}\|^2/(n-p)} \sim F_{p-1, n-p}$$

$$\hat{\beta}_j \pm \sqrt{\hat{\sigma}^2 (X^T X)^{-1}_{jj}} \cdot t_{n-p; 1-\alpha/2}$$

Is a confidence interval which covers the true beta with probability 1-alpha

Generalized least squares (errors correlated with covariance matrix known up to scalar factor):

$$\hat{\beta} = (X^T \Sigma^{-1} X)^{-1} X^T \Sigma^{-1} \mathbf{Y}$$

NW Kernel Estimator

$$\hat{m}(x) = \sum_{i=1}^{n} w_i(x) Y_i, \; w_i(x) = \frac{K((x - x_i)/h)}{\sum_{j=1}^{n} K((x - x_j)/h)}$$

$$\hat{m}(x) = \underset{m_x \in \mathbb{R}}{\arg\min} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)(Y_i - m_x)^2$$

To get NW with optimal local bandwidth selection use library lokern, function lokerns.

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

The covariance of the estimator in terms of moother (hat) matrix: $\text{Cov}(\hat{\mathbf{m}}(x_\cdot)) = \sigma_\varepsilon^2 \mathcal{S}\mathcal{S}^T$,

$$\hat{\sigma}_\varepsilon^2 = \frac{\sum_{i=1}^{n}(Y_i - \hat{m}(x_i))^2}{n - df}$$

$$\widehat{s.e}(\hat{m}(x_i)) = \sqrt{\widehat{\text{Var}}(\hat{m}(x_i))} = \hat{\sigma}_\varepsilon \sqrt{(\mathcal{S}\mathcal{S}^T)_{ii}}$$

For non-parametric regression, degrees of freedom = trace of smoother S matrix

Bias variance decomposition

$$n^{-1} \sum_{i=1}^{n} \mathbb{E}[(m(\mathbf{x}_i) - \sum_{r=1}^{q} \hat{\beta}_{j_r} x_{ij_r})^2]$$
$$= n^{-1} \sum_{i=1}^{n} (\mathbb{E}[\sum_{r=1}^{q} \hat{\beta}_{j_r} x_{ij_r}] - m(\mathbf{x}_i))^2 + \underbrace{n^{-1} \sum_{i=1}^{n} \text{Var}(\sum_{r=1}^{q} \hat{\beta}_{j_r} x_{ij_r})}_{=\frac{q}{n}\sigma^2}.$$

Local polynomial non-parametric regr.

$$\hat{\beta}(x) = \underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)(Y_i - \beta_1 - \beta_2(x_i - x) - \ldots - \beta_p(x_i - x)^{p-1})^2$$

$$\hat{m}(x) = \hat{\beta}_1(x) \qquad \hat{m}^{(r)}(x) = r!\hat{\beta}_{r+1}(x) \; (r = 0, 1, \ldots p-1)$$

Smoothing spline (3(n-2)+2 constraints, 4*(n-1) coeff)

$$\sum_{i=1}^{n}(Y_i - m(x_i))^2 + \lambda \int m''(z)^2 dz, \qquad m_\lambda(x) = \sum_{j=1}^{n} \beta_j B_j(x),$$

$$\hat{\beta}_{n\times 1} = (B^T B + \lambda\Omega)^{-1} B^T \mathbf{Y}$$

All subsets regression in R

```
library(leaps)
mortal.alls  <- regsubsets(Mortality ~ .,data = mortality,nvmax=9)
p.regsubsets(mortal.alls,cex=0.8,cex.main=.8)
```

Generalized CV

$$\text{GCV} = \frac{n^{-1}\sum_{i=1}^{n}(Y_i - \hat{m}(X_i))^2}{(1 - n^{-1}\text{tr}(\mathcal{S}))^2}$$

Compute smoother (hat) matrices R

```
In <- diag(rep(1, n))
for(j in 1:n){
   Snw[,j] <- ksmooth(x, In[,j], kernel = "normal", bandwidth = 0.2,
                      x.points = x)$y
}
```

```
for(j in 1:n){
  Slp[,j] <- predict(loess(In[,j] ~ x,
                           span = span), newdata = x)
  Sss[,j] <- predict(smooth.spline(x, In[,j],
                     df = sum(diag(Snw))), x = x)$y
}
```

```
X <- 1:10
Y=X^2
Z=X^2-2*X
matplot(X,cbind(Y,Z),pch=c(16,1),xlab="x"
```

Bootstrap theory

$$\mathbb{E}^*[\hat{\theta}_n^*] \approx \frac{1}{B}\sum_{i=1}^{B} \hat{\theta}_n^{*i},$$
$$\text{Var}^*(\hat{\theta}_n^*) \approx \frac{1}{B-1}\sum_{i=1}^{B}\left(\hat{\theta}_n^{*i} - \frac{1}{B}\sum_{j=1}^{B}\hat{\theta}_n^{*j}\right)^2,$$
$$\alpha\text{-quantile of distribution of } \hat{\theta}_n^* \approx \text{empirical } \alpha\text{-quantile of } \hat{\theta}_n^{*1}, \ldots, \hat{\theta}_n^{*B}.$$

Consistency of the bootstrap typically holds if the limiting distribution of $\hat{\theta}_n$ is Normal, and if the data $Z_1, \ldots, Z_n$ are i.i.d.

Parametric bootstrap for autoregressive series

1. Generate $\varepsilon_1^*, \ldots, \varepsilon_{n+m}^*$ i.i.d. $\sim \mathcal{N}(0, \hat{\sigma}^2)$ with "burn-in time" $m \approx 1000$
2. Construct recursively, starting with $X_0^* = X_{-1}^* = \ldots = X_{-p+1}^* = 0$,
$$X_t^* = \sum_{j=1}^{p} \hat{\phi}_j X_{t-j}^* + \varepsilon_t^*, \; t = 1, \ldots, n+m.$$
3. Use the bootstrap sample
$$X_{m+1}^*, \ldots, X_{n+m}^*.$$

Param. Bootstrap linear model w/ fixed predictors

1. Simulate $\varepsilon_1^*, \ldots, \varepsilon_n^*$ i.i.d. $\sim \mathcal{N}(0, \hat{\sigma}^2)$.
2. Construct
$$Y_i^* = \hat{\beta}^T x_i + \varepsilon_i^*, \; i = 1, \ldots, n.$$

The parametric bootstrap regression sample is then
$$(x_1, Y_1^*), \ldots, (x_n, Y_n^*),$$
where the predictors x_i are as for the original data.

Double bootstrap in R

```
boot.outer.fnn <- function(data, ind, B){
  res.boot.inner <- boot(data = data[ind], statistic = tm, R = B,
                         sim = "ordinary")
  bci <- boot.ci(res.boot.inner, conf = levels,
                 type = c("basic"))
  ci <- bci[["basic"]][,4:5]
  # returns theta_hat*, all lower bound and all upper bounds
  c(res.boot.inner$t0, ci[,1], ci[,2])
}
```

```
get.level.prime <- function(res.boot.outer, conf){
  thh <- res.boot.outer$t0[1]
  lower <- res.boot.outer$t[, 2:(nl + 1)]
  upper <- res.boot.outer$t[, (nl + 2):(2*nl + 1)]
  included <- lower <= thh & thh <= upper
  colnames(included) <- levels
  cover <- apply(included, 2, mean)
  if(max(cover) >= conf){
    level.prime <- min(levels[cover >= conf])
  } else {
    # if we are unable to to receive the desired coverage
    level.prime <- NA
  }
  level.prime
}
```

```
get.ci <- function(res.boot.outer, conf, level.prime) {
  if(!is.na(level.prime)) {
    level.indices <- c(which(levels == conf), which(levels == level.prime))
  } else {
    level.indices <- c(which(levels == conf), nl)
  }

  lower0 <- res.boot.outer$t0[2:(nl + 1)]
  upper0 <- res.boot.outer$t0[(nl + 2):(2*nl + 1)]

  matrix(cbind(lower0[level.indices], upper0[level.indices]), nrow = 2,
         dimnames =  list(c("basic", "double"), c("lower", "upper")))
}
```

```
glm1 <- glmboost(DEXfat ~ hipcirc  +
  kneebreadth + anthro3a, data = bodyfat)
coef(glm1, off2int=TRUE)    ## off2int ad
```

```
res.boot.outer <-  boot(sample40, statistic = boot.outer.fnn, R = M, B = B)
(level.prime <- get.level.prime(res.boot.outer, conf))
get.ci(res.boot.outer, conf, level.prime)
```

Mboost

```
gam1 <- gamboost(DEXfat ~ bbs(hipcirc) +
bbs(kneebreadth) + bbs(anthro3a),
data = bodyfat)
```

```
cvm <- cvrisk(gam2)
mstop(cvm)    ## ex
gam2[ mstop(cvm) ]
```