

$Y = X\beta + \varepsilon$, $E[\varepsilon] = 0$, $\text{Cov}(\varepsilon) = E[\varepsilon\varepsilon^T] = \sigma^2 I_{n \times n}$

This means that assumption 1-4. from section 1.3.2 are satisfied.

It can then be shown that:

- (i) $E[\hat{\beta}] = \beta$: that is, $\hat{\beta}$ is unbiased
- (ii) $E[\hat{Y}] = E[Y] = X\beta$ which follows from (i). Moreover, $E[r] = 0$.
- (iii) $\text{Cov}(\hat{\beta}) = \sigma^2(X^T X)^{-1}$
- (iv) $\text{Cov}(\hat{Y}) = \sigma^2 P$, $\text{Cov}(r) = \sigma^2(I - P)$
- (i) $\hat{\beta} \sim N_p(\beta, \sigma^2(X^T X)^{-1})$
- (ii) $\hat{Y} \sim N_n(X\beta, \sigma^2 P)$, $r \sim N_n(0, \sigma^2(I - P))$
- (iii) $\hat{\sigma}^2 \sim \frac{\sigma^2}{n-p} \chi^2_{n-p}$

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

Scatterplot

```
col.parta <- 1*(partners==0) + 2*(partners==1) + 3*(partners==2)
plot(thorax, longevity, pch=pch.type, col=col.parta,
     ylim=range(longevity), xlim=range(thorax))
legend("topleft", c("1 pregnant", "2 pregnant", "3 virgin", "0 partners"),
      pch=c(1,2,1,2,3), col=c(2,2,3,3,1))
```

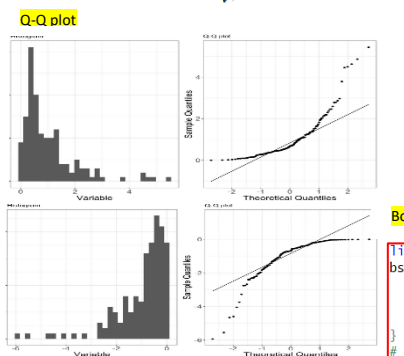
Does indicator variable affect thorax length?

Reject H_0 if $\text{Pr}(p) < 0.05$

```
fit.full <- lm(thorax ~ dummy.1 + dummy.1.v + dummy.8 + dummy.8.v)
fit.intercept <- lm(thorax ~ 1)
anova(fit.intercept, fit.full)
```

Order data

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



Mallows Cp statistic

The MSE can be estimated by:

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

which is prop. to $C_p(M)$:

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

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)
}
```

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 obvious.

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)

```
sploem("mortality", c("Mortality", "Pop", "HC", "NOx", "SO2"), pcales=0)
```

Tukey Anscombe is residuals versus fitted values.

Q-Q is empirical quantiles vs standard normal quantiles.

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))
```

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

Whereas Q-Q plot is given by `plot(fit, which=2)`

Maybe add all-subsets regression in R (from HW2)?

Add anova table and F-test and interpretation

Linearity assumptions (for LS regression)

The linear regression equation is correct. This means: $E[\varepsilon_i] = 0$ for all i

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

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

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

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

LOOCV in R

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

Non-parametric regression in R

Nadaraya Watson (normal kernel)

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

Local polynomial

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

Smoothing splines

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

TODO: Hat matrix (serie 5 methods and non-parametric regression)

TODO: Equivalent ways of calculating LOOCV using hat matrix and manually computation

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]:

$$E^*[\|L_{out}^*\|] = E^*\left[\sum_{i=1}^n 1_{[Z_i \in L_{out}^*]}\right] = nP^*[Z_i \in L_{out}^*] \approx 0.368n$$

Bayes Risk

$$P[C_{Bayes}(X_{new}) \neq Y_{new}]. \quad C_{Bayes}(x) = \arg \max_{0 \leq j \leq J-1} \pi_j(x).$$

LDA Classifier

$$\hat{\mu}_j = \frac{1}{n_j} \sum_{i: Y_i=j} X_i, \quad \hat{\Sigma}_j = \frac{1}{n-j} \sum_{i=0}^{J-1} \sum_{j=0}^{J-1} (X_i - \hat{\mu}_j)(X_i - \hat{\mu}_j)^T 1_{Y_i=j}$$

Prediction of LDA found by taking argmax:

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

Decision boundary between class 0 and 1:

$$B = \{x \mid \hat{\delta}_0(x) = \hat{\delta}_1(x)\} = \{x \mid (x - z)^T w = 0\}$$

if the covariance matrix $\hat{\Sigma}$ 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)

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

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

Bootstrapping LDA/QDA

```
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, ...)
z <- as.numeric(Z$class)
zp <- Z$post[, 3] - pmax(Z$post[, 2], Z$post[, 1])
plot(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)

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

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

```
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(Iris$Species) <- c("setosa", "not", "not")
Iris$Species <- relevel(Iris$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):

$$y_i = \beta_0 + \beta_{1,1} x_{i1} + \beta_{1,2} x_{i1}^2 + \dots + \beta_{1,d} x_{i1}^d + \dots + \beta_{p,1} x_{ip} + \beta_{p,2} x_{ip}^2 + \dots + \beta_{p,d} x_{ip}^d + \varepsilon_i$$

```
require(sfsmsic)
form1 <- as.formula("logop3~.")
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("logop3~.")
gamForm <- wrapFormula(form1, data = d.ozone.e)
gl <- gam(gamForm, data = d.ozone.e)
```

Alternatively,

```
fitA <- gam(log3 ~ s(vdht) + s(wind) + s(humidity) + s(temp) + s(ibht) + s(dppg) + s(ibtp) + s(vsty) + s(day), data = d.ozone)
```

To plot the splines do:

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

To plot the same thing but for lm use:

```
par(mfrow=c(3,3))
plotmplot(fit5, partial.resid=TRUE, rug=FALSE, se=TRUE, col.res="#C0C0C0", pch=19)
```

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

$R^2 = \frac{\| \hat{Y} - \bar{Y} \|^2}{\| Y - \bar{Y} \|^2}$

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

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

The following removes col. op3:

```
d.ozone <- subset(ozone, select=-op3)
d.ozone <- subset(transform(d.ozone, "logO3" = log(O3)), select = -O3)
```

adds `log(O3)` as a col. and removes `O3`

(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 ssp and sst
  fit <- fitfn(formula=formula, data = data, ...)
  ssp <- sum((Y - predict(fit, modFrame))^2)
  sst <- sum((Y - mean(Y))^2)
  1 - ssp / 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)
  for(j in 1:n) {
    if(trace) cat("j = ", j, "\n")
    fit <- fitfn(formula=formula, data = data[-j, ], trace = FALSE)
    # Evaluate ssp + (model.response(modFrame)[j] - predict(fit, modFrame[j,]))^2
    ssp <- ssp + (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 - ssp / 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

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

1. Start with the function $h_0(x) = 1$. Initialize the model set $M = \{h_0(x) = 1\}$. Fit the function h_0 by least squares regression, yielding the estimate $\hat{\beta} = n^{-1} \sum_{i=1}^n Y_i$.

2. For $r = 1, 2, \dots$, do the following: Search for the best pair of functions $(h_{2r-1}(x), h_{2r}(x))$ which are of the form

$$h_{2r-1}(x) = h_0(x) * (x_j - d_r)_+, \quad h_{2r}(x) = h_0(x) * (d_r - x_j)_+, \quad (7.1)$$

for some h_0 in the model set M which does not contain x_j^2 and some basis functions in B . The best pair of functions is defined to be the one which reduces residual sum of squares most. The model fit is then

$$\hat{g}(x) = \hat{\mu} + \sum_{m=1}^{2r} \hat{\beta}_m h_m(x),$$

where the coefficients $\hat{\beta}_m$ are estimated by least squares. Enlarge the model set in every iteration (with index r) by

$$M = M_{old} \cup \{h_{2r-1}(x), h_{2r}(x)\},$$

with the functions h_{2r-1}, h_{2r} from (7.1).

3. Iterate step 2 until a large enough number of basis functions $h_m(x)$ has been fitted.

4. Do backward deletion ("pruning"), allowing to remove single functions from a pair $h_{2r-1}(x), h_{2r}(x)$; i.e., of some 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:

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

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

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

Neural Networks (1 Layer with q units)

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

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 conn):

$$g(x) = f_0 \left(\alpha + \sum_{j=1}^p w_{j0} x_j + \sum_{k=1}^q w_k \phi(\alpha_k + \sum_{j=1}^p w_{kj} x_j) \right)$$

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 reg 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$y, data$N)) +
    data$y * g(beta,data$age) -
    data$N * log( 1 + exp(g(beta,data$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:
$$\begin{aligned} \mathcal{R}_{left} &= \mathbb{R} \times \mathbb{R} \times \dots \times (-\infty, d] \times \mathbb{R} \dots \times \mathbb{R}, \\ \mathcal{R}_{right} &= \mathbb{R} \times \mathbb{R} \times \dots \times (d, \infty) \times \mathbb{R} \dots \times \mathbb{R}, \end{aligned}$$

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 $j \in \{1, \dots, p\}$ 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:
$$\mathcal{P} = \mathcal{P}_{old} \setminus \text{partition cell selected to be refined} \cup \{\text{refinement cells } \mathcal{R}_{left}, \mathcal{R}_{right}\}.$$
- 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 $-2^*(\log \text{likelihood}) = \text{sum of squares as in the case for regression or the misclassification rate for classification}$

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

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(O)).

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$cpstable`

```
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$cpstable
  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

$Y_i = \beta_0 + \beta_1 x_{i,1} + \dots + \beta_p x_{i,p} + \varepsilon_i, \quad i = 1, \dots, n.$
can be rewritten as
 $Y_i = \beta_0' + \beta_1 (x_{i,1} - \bar{x}_{.,1}) + \dots + \beta_p (x_{i,p} - \bar{x}_{.,p}) + \varepsilon_i.$
which is beneficial for ridge since we do not want to penalize the intercept term (β_0' is simply \bar{Y} , the avg of the Y values).
Ridge regression is formalized as:

$$\hat{\beta}(s) = \arg \min_{\|\beta\|^2 \leq s} \|Y - X\beta\|^2$$

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

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

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

Note that $XTX + \lambda I$ is always inv. for $\lambda > 0$

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

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

derived from SV Decomposition of X as UDV^T where U is $n \times n$, V is $p \times p$, and both are orthogonal. D is $n \times p$ 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:

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

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 $\alpha > 0$, note that the lasso penalty ($\alpha=0$) is convex, but not strictly.

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

$$\arg \min_{\beta} \left\| Y - \sum_{j=1}^p \beta_j x_j \right\|^2 + \lambda \cdot \sum_{j=1}^p w_j |\beta_j|$$

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

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 $\hat{\beta}(\lambda) = (\hat{\beta}_1^{\lambda}, \dots, \hat{\beta}_p^{\lambda})^T$ be the lasso estimated parameter vector of (7.10), and define $\mathcal{M}_{\lambda} = \{1 \leq k \leq p \mid \hat{\beta}_k^{\lambda} \neq 0\}$, (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
$$\hat{\beta}^{\lambda, \phi} = \arg \min_{\beta} n^{-1} \sum_{i=1}^n (Y_i - \sum_{k \in \mathcal{M}_{\lambda}} \beta_k x_{i,k})^2 + \phi \lambda \cdot \|\beta\|_1. \quad (7.16)$$

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], wrapStrings="poly(*,degree=3)"))

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

## combine last two formulas
(ff <- formula(paste(deparse1(ffrac), "+", 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
```

CV for Ridge/Lasso and Saving Objects

note that the 1-std error rule was used to get λ (the λ that minimizes CV is given by `cv.e1n$lambda.min`)

```
sFile <- "cv-elastn.rds"
if(!file.exists(sFile)) {
  set.seed(1)
  print(system.time(
    cv.e1n <- cv.glmnet(mm, y, alpha=0.5, nfolds=10)
  ))
  ## user system elapsed
  ## 93.972 0.117 94.268 -- 1.5 min
  ## now save it in the "safe-file"
  saveRDS(cv.e1n, sFile)
} else { ## read pre-computed result:
  cv.e1n <- readRDS(sFile)
}

#

plot(cv.e1n)
cv.e1n$lambda.1se
```