1 Linear Regression

Linear Model: $Y = X\beta + \epsilon$, $Y \in \mathbb{R}^n, X \in \mathbb{R}^{n\times p}, \beta \in \mathbb{R}^p$

Definitions and important results

$RSS(\beta) = \|Y - X\beta\|_2^2$, $\beta \in \mathbb{R}^p$; Residuals: $r := Y - \hat{Y}$

Estimated params.: $\hat\beta = (X^\top X)^{-1} X^\top Y$;
$\hat{Y} = X\hat\beta = X(X^\top X)^{-1}X^\top Y$; $\hat{Y}_{new} = X_{new}^\top \hat\beta$

Projection $P := X(X^\top X)^{-1}X^\top$ is projection onto column space of X $\implies \forall j \in 1,\dots,p : r^\top x^j = 0$

Inference Consider $Y = X\beta + \epsilon$, $\mathbb{E}[\epsilon] = 0, Cov(\epsilon) = \mathbb{E}[\epsilon\epsilon^\top] = \sigma^2 \mathbb{1}_{n\times n}, rank(X^\top X) = p$. Then,

- $\mathbb{E}[\hat\beta] = \mathbb{E}[(X^\top X)^{-1}X^\top(X\beta+\epsilon)] = \beta \rightarrow \hat\beta$ unbiased
- $\mathbb{E}[\hat Y] = \mathbb{E}[Y] = X\beta$ and $\mathbb{E}[r] = 0$
- $Cov(\hat\beta) = \sigma^2(X^\top X)^{-1}$; $Cov(\hat Y) = \sigma^2 P$
- $Cov(r) = \sigma^2(\mathbb{1} - P) \rightarrow$In general non-diagonal!
- $\mathbb{E}[\sum_{i=1}^n r_i^2] = \sum_{i=1}^n \mathbb{E}[r_i^2] = \sum_{i=1}^n \sigma^2(\mathbb{1} - P_{ii}) = \sigma^2(n - tr(P)) = \sigma^2(n-p) \rightarrow \hat\sigma^2 = (n-p)^{-1}\sum_{i=1}^n r_i^2$ unbiased for σ^2

Normality: Assume now $\epsilon_1,\dots,\epsilon_n \overset{iid}{\sim} \mathcal{N}(0,\sigma^2)$. Then,

- $\hat\beta \sim \mathcal{N}(\beta, \sigma^2(X^\top X)^{-1})$; $\hat Y \sim \mathcal{N}(X\beta, \sigma^2 P)$ and $r \sim \mathcal{N}(0, \sigma^2(\mathbb{1}-P))$
- $\hat\sigma^2 \sim \frac{\sigma^2}{n-p}\chi^2_{n-p}$

Tests and confidence intervals

Assume again **normality**. Null hyp.: $H_{0,j} : \beta_j = 0$

Under the null: $T_j = \dfrac{\hat\beta_j}{\sqrt{\hat\sigma^2[(X^\top X)^{-1}]_{jj}}} \sim t_{n-p}$

Note: An *individual* t-test for $H_{0,j}$ quantifies the effect of j-th predictor var. after having subtracted the linear effect of all other predictors on Y

Confidence Interval $\hat\beta_j \pm \sqrt{\hat\sigma^2[(X^\top X)^{-1}]_{jj}} \cdot t_{n-p;1-\alpha/2}$

CI and PI for $x_0^\top\beta$: CI $= \dfrac{x_0^\top\hat\beta - x_0^\top\beta}{\hat\sigma\sqrt{x_0^\top(X^\top X)^{-1}x_0}} \sim t_{n-p}$

and PI $= \dfrac{y_0 - x_0^\top\beta}{\hat\sigma\sqrt{1 + x_0^\top(X^\top X)^{-1}x_0}} \sim t_{n-p}$

Global: $H_0 : \beta_2,\dots,\beta_p = 0$ vs. $H_A : \exists j \in 2,\dots,p : \beta_j \neq 0$

Anova Decomp.: $\|Y - \overline{Y}\|^2 = \|Y - \hat Y\|^2 + \|\hat Y - \overline{Y}\|^2$, where under H_0, this is ≈ 0. Also, we have $R^2 := \frac{\|\hat Y - \overline{Y}\|^2}{\|Y - \overline{Y}\|^2}$. In SLR, this is : $R^2 = (\text{cor}(Y, \hat Y))^2$

Anova table:

Source	df	Sum of Squares	Mean Square
Regression	$p-1$	$\|\hat Y - \overline{Y}\|^2$	$\frac{\|\hat Y - \overline{Y}\|^2}{p-1}$
Error	$n-p$	$\|Y - \hat Y\|^2$	$\frac{\|Y - \hat Y\|^2}{n-p}$
Total	$n-1$	$\|Y - \overline{Y}\|^2$	

F-test: $\frac{\|\hat Y - \overline{Y}\|^2/(p-1)}{\|Y - \hat Y\|^2/(n-p)} \sim F_{p-1,n-p}$. Comp. alternative: Permutation test

Def. level Let $\alpha \in (0,1)$ and $\forall P \in H_0$ let $P^n := P \otimes \dots \otimes P$. A test $T_n : \mathcal{X}^n \rightarrow \{0,1\}$ (where $T_n = 1$ means that we reject H_0) is a test with...

1. level α if $\sup_{P\in H_0} \mathbb{P}_{P^n}(T_n = 1) \leq \alpha$
2. pointwise asymptotic level α if $\sup_{P\in H_0} \lim_{n\to\infty} \mathbb{P}_{P^n}(T_n = 1) \leq \alpha$
3. uniform asymptotic level α if $\lim_{n\to\infty}\sup_{P\in H_0} \mathbb{P}_{P^n}(T_n = 1) \leq \alpha$

Note: 2 weaker than 3 weaker than 1
p-value: Given a test, the p-value is the infimum over all significance levels s.t. the test rejects

Evaluating model assumptions

- TA-plot: r_i vs. $\hat Y_i$ (sample corr. btw. $\hat Y$ & r_i is = 0)
- QQ-plot: emp. quantile of (standardized) residuals $(Y_i - \hat Y_i)/\hat\sigma$ vs. th. quantile of $\mathcal{N}(0,1) \rightarrow$Too many (few) large values: right-(left-)skewed
- Cook's distance: $D_i := \frac{\sum_{j+1}(\hat Y_j - \widehat{Y_j^{-i}})^2}{p\|Y - \hat Y\|^2/(n-p)}$ ($D_i > 1$ infl.)

Model selection

If $p < n$ but $p \approx n$, we may be able to **decrease aMSE** (not RSS!):
Let $J_q := \{j_1,\dots,j_q\}, \#J_q = q \leq p$. With $\hat m_{J_q}(\cdot)$ the smaller model fitted with predictors J_q, we have $aMSE = \frac{1}{n}\sum_{i=1}^n \mathbb{E}[(\hat m_{J_q}(x_i) - m(x_i))^2]$
$= \frac{1}{n}\sum_{i=1}^n(\mathbb{E}[\hat m_{J_q}(x_i)] - m(x_i))^2 + \frac{1}{n}\sum_{i=1}^n \text{Var}(\hat m_{J_q}(x_i))$
$= \frac{1}{n}\sum_{i=1}^n \text{Bias}(\hat m_{J_q}(x_i))^2 + \frac{q}{n}\sigma^2 \rightarrow$var increases with q
Instead of incl. all variables, define $\forall\lambda \geq 0$:
$\hat\beta(\lambda) := argmin_\beta \|Y - X\beta\|^2 + \lambda\|\beta\|_0$, $\|\beta\|_0 := \{j : \beta_j \neq 0\}$ Choose λ with AIC ($\lambda = 2\hat\sigma^2$; equiv. to Mallow's CP with linear Gaussian models), BIC ($\lambda = log(n)\hat\sigma^2$), or CV. This problem is non-convex: can use forward & backward selection or combinations.

2 Nonparametric Density Estimation

$x_1,\dots,x_n \overset{i.i.d}{\sim} F$ diff. and $f := F' \rightarrow$Estimate f by $\hat f$
Surrogate Criterion: $MISE = \mathbb{E}[\int(f(x) - \hat f(x))^2dx] = \int MSE(x)dx = \int(\mathbb{E}[\hat f(x)] - f(x))^2 + Var(\hat f(x))dx = IMSE$
Est. 1: Histogram: Choose: $x_0 \in \mathbb{R}, h > 0$.
Then, $\forall x \in \mathbb{R} : \hat f_{x_0,h}(x) := \sum_{j\in\mathbb{Z}} \hat g_j \mathbb{1}_{[x\in I_j]}$, where $\forall j \in \mathbb{Z} : I_j := (x_0 + jh, x_0 + (j+1)h]$ and $\hat g_j := \frac{\#\{i\in\{1,\dots,n\}:x_i\in I_j\}}{nh}$. Note: $\hat f$ is not continuous.

Est. 2: KDE: Fix a kernel $k : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$ s.t. $\int_{-\infty}^{+\infty} k(x)dx = 1$, k is bounded and $\forall x \in \mathbb{R} : k(x) = k(-x)$ and $h > 0$. Define $\hat f_h(x) := \frac{1}{nh}\sum_{1=1}^n k\left(\frac{x - x_i}{h}\right)$

Bias-Variance Trade-off: the (absolute value of the) bias of $\hat f$ increases and the variance of $\hat f$ decreases as h increases.
$\mathbb{E}[\hat f(x)] = \int \frac{1}{h}K\left(\frac{x-y}{h}\right)f(y)dy$,
$Var(\hat f(x)) = n^{-1}\int\frac{1}{h^2}K\left(\frac{x-y}{h}\right)^2 f(y)dy - n^{-1}\left(\int\frac{1}{h}K\left(\frac{x-y}{h}\right)f(y)dy\right)^2$
Asymptotics: Assume $h = h_n \rightarrow 0$ with $nh_n \rightarrow \infty$. Then (with $z = (y-x)/h$),
$Bias(x) = h^2 f''(x)\int z^2K(z)dz/2 + o(h^2)$ as $(n\rightarrow\infty)$;
$Var(\hat f(x)) = (nh)^{-1}f(x)\int K(z)^2dz + o((nh)^{-1})$ as $(n \rightarrow \infty)$
$\rightarrow$bias and variance go to 0 asympt. as $h = h_n \rightarrow 0$ and $nh_n \rightarrow \infty$!
Optimal **local** bandwidth minimizes leading term in asymptotic $MSE(x)$:

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

Optimal **global** bandwidth minimizes the asymptotic $MISE$:
$h_{opt} = n^{-1/5}\left(R(K)/\sigma_K^4 * \frac{1}{R(f'')}\right)^{1/5}$, where $R(g) = \int g^2(x)dx$, and $\sigma_K^2 = \int x^2K(x)dx$. The **optimal rate** for the $MISE$ and $MSE(x)$ is thus of order $O(n^{-4/5})$.
$\rightarrow$asympt. best bandwidth depends on $R(f'')$, which is unknown. Can estimate f'' again by a kernel estimator with an "initial" bandwidth h_{init}, yielding $\hat f''_{init}$ (Sheather-Jones in R: density(*, bw='SJ'))

Curse of Dimensionality: In the multivariate case, we use $K : \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$ and have an asympt. optimal MSE of order $O\left(n^{-4/(4+d)}\right)$. Thus, KDE is often restricted to $d = 2$. To improve MSE by factor 0.1 for d=10, need to incr. sample size by factor 3160: $O(n^{-4/14}) = 0.1 \iff n = 0.1^{-14/4}$

3 Nonparametric Regression

Fixed design. $Y_i = m(x_i) + \epsilon_i$, where $m(x_i)$ is nonrandom, $\epsilon_1,\dots,\epsilon_n$ i.i.d., with $\mathbb{E}[\epsilon_1] = 0$, $Var(\epsilon_1) = \sigma_\epsilon^2$
Random design. $Y_i = m(X_i) + \epsilon_i$, where $\epsilon_1,\dots,\epsilon_n$ i.i.d. with $\mathbb{E}[\epsilon_i] = 0$ and $m : \mathbb{R} \rightarrow \mathbb{R}$ is an "arbitrary" function (*nonparametric regression function*), satisfying $m(x) = \mathbb{E}[Y|X = x]$.

Kernel regression estimator (ksmooth)

In the **random design**, we have
$\mathbb{E}[Y|X = x] = \int y f_{Y|X=x}(y)dy = \frac{\int y f_{X,Y}(x,y)dy}{f_X(x)}$. Plugging in the KDEs,
we get $\hat m_{kde}(x) = \frac{\sum_{i=1}^n K\left((x-x_i)/h\right)Y_i}{\sum_{j=1}^n K\left((x-x_j)/h\right)}$ (NW-kernel estimator)

In the **fixed design**, we solve $argmin_{m_x}\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)\left(Y_i - m_x\right)^2$, which is equal to $\hat m_{kde}(x)$ above! $\rightarrow$For every fixed x, search for best local constant m_x s.t. localized sum of squares is minimized.
Note: $h \rightarrow \infty$: straight line (high bias, small var.); $h \rightarrow 0$: interpolation.
Hat matrix: Kernel estimator evaluated at design points $\hat m(x_1),\dots,\hat m(x_n)$ is a linear operator:
$S : \mathbb{R}^n \rightarrow \mathbb{R}^n, (Y_1,\dots,Y_n)^\top \rightarrow (\hat m(x_1),\dots,\hat m(x_n))^\top =: \hat m(x.) = \hat{\mathbf{Y}}$, i.e.,
$\hat{\mathbf{Y}} = S\mathbf{Y}$, where S is the hat matrix representing the linear operator with $[S]_{r,s} = w_s(x_r), r,s \in \{1,\dots,n\}$, since $S\{(Y_1,\dots Y_n)^\top\} = (\hat m(x_1),\dots,\hat m(x_n))^\top$,
and $w_i(x) = \frac{K((x-x_i)/h)}{\sum_{j=1}^n K((x-x_j)/h)}$

Covariance: $Cov(\hat{\mathbf{m}}(x.)) = \sigma_\epsilon^2 SS^\top$, i.e., $Cov(\hat m(x_i), \hat m(x_j)) = \sigma_\epsilon^2(SS^\top)_{ij}$, and $Var(\hat m(x_i)) = \sigma_\epsilon^2(SS^\top)_{ii}$. We estimate the unknown σ_ϵ^2 by
$\hat\sigma_\epsilon^2 := \frac{\sum_{i=1}^n(\hat m(x_i) - Y_i)^2}{n - df}$, where $df := tr(S)$

Confidence Interval: We have $\widehat{s.e}(\hat m(x_i)) = \sqrt{\widehat{Var}(\hat m(x_i))} = \hat\sigma_\epsilon\sqrt{(SS^\top)_{ii}}$.
Because $\hat m(x_i) \approx \mathcal{N}(\mathbb{E}[\hat m(x_i)], Var(\hat m(x_i)))$, it follows that
$I = \hat m(x_i) \pm 1.96 * \widehat{s.e.}(\hat m(x_i))$ yields approx. pointw. conf. intervals for $\mathbb{E}[\hat m(x_i)]$ (**for the expected value**, not true underlying function $m(x_i)$!).
Correction of this interval: $I - \widehat{bias}$, where $\widehat{bias}$ is an esimate of the bias.
Note: We only have confidence and not prediction intervals!

Local polynomial nonparametric regression esimator (LOESS)

Extends the locally constant kernel regression estimator (ksmooth) to a locally polynomial regression estimator:
$\hat\beta(x) = \underset{\beta\in\mathbb{R}^p}{argmin}\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)\left(Y_i - \beta_1 - \beta_2(x_i - x) - \dots - \beta_p(x_i - x)^{p-1}\right)^2$
We usually use $p = 2$ or $p = 4$. The function estimator is given by evaluating $\sum_{j=1}^p \hat\beta_j(x)(u-x)^{j-1}$ at $u = x$. Due to the centering, only the intercept remains: $\hat m(x) = \hat\beta_1(x)$
$\rightarrow$often better at edges & yields derivatives $\hat m^{(r)}(x) = r!\hat\beta_{r+1}(x)$, for $r = 0,1,\dots,p-1$

Smoothing splines and penalized regression

Minimize $\sum_{i=1}^n(Y_i - m(x_i))^2 + \lambda\int m''(z)^2dz$ over twice diff. functions. $(*)$
$\rightarrow\lambda = 0$: perfect interpolation (not unique) $\rightarrow\lambda\rightarrow\infty$: linear regression
$\rightarrow 0 < \lambda < \infty$: Closed form solution available
Def. Natural cubic spline: Let $a \leq x_1 \leq \dots \leq x_n \leq b$. We call $g : [a,b] \rightarrow \mathbb{R}$ a *cubic spline* if *a)* on all intervals $[a, x_1),\dots,[x_n, b)$ g is a cubic polynomial, and *b)* g has two continuous derivatives on $[a,b]$. Furthermore, g is *natural* if $g''(a) = g''(b) = g'''(a) = g'''(b) = 0$
Thm.: The unique minimizer of $(*)$ is the unique natural cubic spline with $g = (\mathbb{1} + \lambda K)^{-1}Y$, where $K = QR^{-1}Q$, and Q, R are banded matrices (easy to compute)
Note: n free params. (but $df = tr(S_\lambda)$). Choose λ via CV.

4 Cross-Validation

$\rightarrow$Used for estimating generalization (out-of-sample) performance
When having an estimated target $\hat m$ and a loss function ρ, we would like to evaluate $\frac{1}{\ell}\sum_{i=1}^\ell \rho(Y_{new,i}, \hat m(X_{new,i}))$, where $\hat m(.)$ is constructed from training data only, and $(X_{new,1}, Y_{new,1}),\dots,(X_{new,\ell}, Y_{new,\ell}) \overset{iid}{\sim} P$ is new test data, independent from training data but with same distribution P.
If ℓ is large, this evaluation on the test set approximates the **theoretical test set error**: $\mathbb{E}_{X_{new},Y_{new}}[\rho(Y_{new}, \hat m(X_{new}))]$, which is still a fct. of training data, due to $\hat m$.
The expected value of this (w.r.t. training data) is the **generalization error**:
$\mathbb{E}_{training}\mathbb{E}_{X_{new},Y_{new}}[\rho(Y_{new}, \hat m(X_{new}))] = \mathbb{E}[\rho(Y_{new}, \hat m(X_{new}))]$.

CV Schemes

LOOCV. The cross-validated performance (of estimator $\hat m$, where $\hat m^{(-i)}$ is trained without the i-th data point) is: $\frac{1}{n}\sum_{i=1}^n \rho\left(Y_i, \hat m_{n-1}^{(-i)}(X_i)\right)$, which is an estimate of the test set error, or generalization error.
$\rightarrow$requires fitting the estimator n times

K-fold CV. Construct an estimator $\hat m_{n-|\mathcal{B}_k|}^{(-\mathcal{B}_k)}$, where for $k = 1,\dots,K$, $\mathcal{B}_k$ are K equally sized subsets *without* intersection. The cross-validated performance of $\hat m$ is then again: $\frac{1}{K}\sum_{k=1}^K \frac{1}{|\mathcal{B}_k|}\sum_{i\in\mathcal{B}_k}\rho\left(Y_i, \hat m_{n-|\mathcal{B}_k|}^{(-\mathcal{B}_k)}(X_i)\right)$.
$\rightarrow$requires fitting the estimator K times, and $K \ll n$

Leave-d-out CV. *Idea: if data is i.i.d., the indexing of the data should not matter!* Leave a set $\mathcal{C}$ comprising d obs. out and use the remaining $n - d$ data points for training. The cross-validated performance of $\hat m$ is then:
$\binom{n}{d}^{-1}\sum_{k=1}^{\binom{n}{d}} d^{-1}\sum_{i\in\mathcal{C}_k}\rho\left(Y_i, \hat m_{n-d}^{(-\mathcal{C}_k)}(X_i)\right)$. $\rightarrow$comp. infeasible if $d \geq 3$.
Computational shortcut is **randomization**: Draw B random test subsets $\mathcal{C}_1^*,\dots,\mathcal{C}_B^* \overset{i.i.d}{\sim} Unif(\{1,\dots,\binom{n}{d}\})$ ($\mathcal{C}^*$ is obtained by sampling d times without replacement from $\{1,\dots,n\}$). The random approx. then gives the CV performance $\frac{1}{B}\sum_{k=1}^B d^{-1}\sum_{i\in\mathcal{C}_k^*}\rho\left(Y_i, \hat m_{n-d}^{(-\mathcal{C}_k^*)}(X_i)\right)$. Often choose $d = [\gamma n], \gamma \approx 0.1$, and $B \approx 50 - 500$. Stochastic version may be even faster than LOOCV if $B < n$. Stochastic version equivalent to leave-d-out CV for $B = \infty$

Properties of CV-schemes

- One random split into test- and training-data: fastest; poor both in bias and variance
- LOOCV: approx. unbiased for true gen. error; Variance high, because the n training sets are so similar to each other
- Leave-d-out CV: Lower variance than LOOCV; Higher bias than LOOCV with $d > 1$
- K-fold CV & stochastic approx.: K-fold CV has larger bias than LOOCV. Effects on variance is not clear! Stochastic approx. expected to have higher variance than comp. infeasible leave-d-out CV.

Comp. shortcut for some linear fitting operators

Consider a linear fitting operator S, $(\hat m(x_1),\dots,\hat m(x_n))^\top = SY$, and the squared loss $\rho(y,x) = |y - x|^2$.
Then: $\frac{1}{n}\sum_{i=1}^n\left(Y_i - \hat m_{n-1}^{(-i)}(X_i)\right)^2 = \frac{1}{n}\sum_{i=1}^n\left(\frac{Y_i - \hat m(X_i)}{1 - S_{ii}}\right)^2$
$\rightarrow$Can compute CV score by fitting original estimator $\hat m(.)$ **once on full dataset**, without having to do it n times. S can be computed in $\mathcal{O}(n)$ time.

5 Bootstrap

Setting: $Z_1,\dots Z_n \overset{i.i.d}{\sim} P$; Target of inference: θ; Estimator of θ: $\hat\theta = \hat\theta_n = \hat\theta_n(Z_1,\dots,Z_n)$

Basic bootstrap algorithm

1) $Z_1^*,\dots,Z_n^* \overset{\text{iid}}{\sim} \hat P_n$ –> Do n uniform random drawings with replacement from the data set $Z_1,\dots,Z_n$
2) Compute bootstrapped estimator $\hat\theta_n^* = g(Z_1^*,\dots Z_n^*)$
3) Repeat steps 1 and 2 **B** times to obtain $\hat\theta_n^{*1},\dots,\hat\theta_n^{*B}$
4) Use these bootstrapped estimators as approximations for the bootstrap expectation, variance and quantiles: $\mathbb{E}^*[\hat\theta_n^*] \approx \frac{1}{B}\sum_{i=1}^B\hat\theta_n^{*i}$
$Var^*(\hat\theta_n^*) \approx \frac{1}{B-1}\sum_{i=1}^B(\hat\theta_n^{*i} - \frac{1}{B}\sum_{j=1}^B\hat\theta_n^{*j})^2$
α-quantile of distribution of $\hat\theta_n^* \approx$ empirical α-quantile of $\hat\theta_n^{*1},\dots,\hat\theta_n^{*B}$

Consistency

We call the bootstrap **consistent** for $\hat\theta$ if the following holds (for $n \rightarrow \infty$):
$\exists a_n\forall x : \mathbb{P}_p(a_n(\hat\theta_n - \theta) \leq x) - \mathbb{P}_{p^*}(a_n(\hat\theta_n^* - \hat\theta) \leq x) \overset{\mathbb{P}}{\rightarrow} 0$,
where P^* is the cond. distr. of $Z_1^*,\dots Z_n^*$ given $Z_1,\dots,Z_n$ and a_n incr. seq.
Note: Consistency of the bootst. (usually) implies consist. var. and bias est. Also, consistency typically holds if the limiting distr. of $\hat\theta_n$ is normal, and if $Z_1,\dots,Z_n$ are i.i.d. (Note: no bootstrap consist. for finite sample size)

Bootstrap confidence interval

Due to bootstrap consistency, we have:
$[\hat\theta - q_{1-\frac{\alpha}{2}}, \hat\theta - q_{\frac{\alpha}{2}}]$ (q is quantile of $\hat\theta - \theta$)
$\approx [\hat\theta - \hat q_{1-\frac{\alpha}{2}}, \hat\theta - \hat q_{\frac{\alpha}{2}}]$ ($\hat q$ is quantile of $\hat\theta^* - \hat\theta$)
$= [2\hat\theta - q^*_{1-\frac{\alpha}{2}}, 2\hat\theta - q^*_{\frac{\alpha}{2}}]$ (q^* is quantile of $\hat\theta^*$)
$= I^*(1-\alpha)$ (**Basic Bootstrap CI**) $\rightarrow$often better
$\neq [q^*_{\frac{\alpha}{2}}, q^*_{1-\frac{\alpha}{2}}]$ (**Percentile Bootstrap CI**; does not correct for bias in $\hat\theta_n$)
Another CI is based on the **normal approximation** with a bias correction:
$2\hat\theta - \overline{\hat\theta^*} \pm q_Z(1-\alpha/2)\cdot\hat{\text{sd}}(\hat\theta)$; $Z \sim \mathcal{N}(0,1)$; $\hat{\text{sd}}(\hat\theta) = \sqrt{\frac{1}{R-1}\sum_{i=1}^R(\hat\theta^{*i} - \overline{\hat\theta^*})^2}$

Generalization error

Goal: estimate $\mathbb{E}[\rho(Y_{new}, \hat m(X_{new}))]$ for loss fct. ρ, and $\hat m$ possibly a regr. estimator.
Bootstrap gen. error is: $\mathbb{E}^*[\rho(Y^*_{new}, \hat m^*(X^*_{new}))]$, where $\mathbb{E}^*$ is the bootstrap expectation w.r.t all the bootstrap variables $train^* = (X_1^*,\dots,X_n^*)$ and $test^* = (X^*_{new}, Y^*_{new})$
Shortcut: It holds that $\mathbb{E}^*[\rho(Y^*_{new}, \hat m^*(X^*_{new}))] = \frac{1}{n}\sum_{i=1}^n\mathbb{E}^*[\rho(Y_i, \hat m^*(X_i))]$ (avg. of boots. errs. over original data (X_i, Y_i)): no need to generate (X^*_{new}, Y^*_{new}))
Practical algorithm: 1) $(X_1^*, Y_1^*),\dots,(X_n^*, Y_n^*) \sim \hat P_n$; 2) compute $\hat m^*(.)$ based on $(X_1^*, Y_1^*),\dots,(X_n^*, Y_n^*)$; 3) evaluate $err^* = \frac{1}{n}\sum_{i=1}^n\rho(Y_i, \hat m^*(X_i))$;
4) Repeat steps 1-3 **B** times to obtain $err^{*1},\dots,err^{*B}$ and approx. bootst. error by $\frac{1}{B}\sum_{i=1}^B err^{*i}$ (=est. of true gen. err) $\rightarrow$*may be overoptimistic*
Out-of-bootstrap sample (OOB) gen. error:
$\frac{1}{B}\sum_{b=1}^B\frac{1}{|\mathcal{L}_{out}^{*(b)}|}\sum_{i\in\mathcal{L}_{out}^{*(b)}}\rho(Y_i, \hat m^{*(b)}(X_i))$, where for $\mathcal{L}^* = Z_1^*,\dots Z_n^*$,
$\mathcal{L}^*_{out} = \cup_{i=1}^n\{(X_i, Y_i) : (X_i, Y_i) \notin \mathcal{L}^*\}$. $\mathcal{L}^*_{out}$ has exp. size of $n * 0.368$

Double Bootstrap

Problem: $\mathbb{P}(\theta \in I^*(1-\alpha)) = 1 - \alpha + \triangle_n$. **Algorithm**:
1) Repeat M **times**:

- Draw $Z_1^*,\dots,Z_n^* \sim P_n$ and compute $\hat\theta^*$
- Repeat B times:
  - Generate $Z_1^{**},\dots,Z_n^{**} \sim Z_1^*,\dots,Z_n^*$ (with repl.) and compute $\hat\theta^{**}$
- $I^{**}(1-\alpha) := [2\hat\theta^* - q^{**}(1 - \frac{\alpha}{2}); 2\hat\theta^* - q^{**}(\frac{\alpha}{2})]$
- $cover^*(1-\alpha) := \mathbb{1}_{[\hat\theta\in I^{**}(1-\alpha)]}$

2) Take average over all M **covers** as approx. for $\mathbb{P}^*[\hat\theta \in I^{**}(1-\alpha)]$:
$p^*(\alpha) := \frac{1}{M}\sum_{i=1}^M cover^{*i}(1-\alpha)$
3) Vary α (in all of step 1 and 2) to find α'^* such that $p^*(\alpha'^*) = 1 - \alpha$ (desired nominal level) and use $\widehat{1 - \alpha'} = 1 - \alpha'^*$
Note: Requires $B * M$ bootstrap samples

Parametric Bootstrap

Assume data to be realizations from $Z_1,\dots,Z_n$ i.i.d. $\sim P_\theta$. Then, estimate θ by $\hat\theta$ and proceed by using $Z_1^*,\dots,Z_n^* \sim P_{\hat\theta}$

Parametric Bootstrap - Linear model

Let $Y_i = \beta^T x_i + \epsilon_i$, where $x_i \in \mathbb{R}^p$ are fixed and $\epsilon_1,\dots,\epsilon_n$ i.i.d$\sim\mathcal{N}(0,\sigma^2)$.
1) Estimate $\theta = (\beta, \sigma^2)$ by $\hat\theta = (\hat\beta, \hat\sigma^2)$. **2)** Construct $(x_1, Y_1^*),\dots,(x_n, Y_n^*)$, where $Y_i^* = \hat\beta^T x_i + \epsilon_i^*$ and $\epsilon_i^* \sim \mathcal{N}(0, \hat\sigma^2)$

Parametric Bootstrap - AR(p) Model

Let $X_t = \sum_{j=1}^p\phi_jX_{t-j} + \epsilon_t$, where $X_t \in \mathbb{R}$ and $\epsilon_1,\dots,\epsilon_n$ i.i.d$\sim\mathcal{N}(0,\sigma^2)$. **1)** Estimate $\theta = (\phi, \sigma^2)$ by $\hat\theta = (\hat\phi, \hat\sigma^2)$. **2)** Construct $X^*_{m+1},\dots,X^*_{m+n}$ recursively from $X_t^* = \sum_{j=1}^p\hat\phi_jX^*_{t-j} + \epsilon_t^*$, where $\epsilon_1^*,\dots,\epsilon^*_{n+m}$ i.i.d$\sim\mathcal{N}(0,\hat\sigma^2)$ and $m \approx 1000$ is the "burn-in time". *Note:* We throw away $X_1^*,\dots,X_m^*$ to obtain bootstrap sample that is approximately a stationary process.

Model-based bootstrap for regression

Model original data by $Y_i = m(x_i) + \epsilon_i$, where $\epsilon_1,\dots,\epsilon_n$ i.i.d. $\sim P_\epsilon$ with $\mathbb{E}_{P_\epsilon}[\epsilon_1] = 0$ and $m(.)$ being parametric or non-parametric. **1)** Estimate m by $\hat m$ and calculate centered residuals $\tilde r_i = r_i - \frac{1}{n}\sum_i^n r_i$. **2)** Construct $(x_1, Y_1^*),\dots,(x_n, Y_n^*)$, where $Y_i^* = \hat m(x_i) + \epsilon_i^*$ and $\epsilon_i^* \sim \hat P_{\tilde r}$

Independence Testing using Bootstrap

Let $B_n = \{\rho|\rho : \{1,\dots n\} \rightarrow \{1,\dots n\}\}$, $(X_1, Y_1),\dots,(X_n, Y_n) \in \mathcal{X}$ i.i.d. $\sim P^{(X,Y)}$, $H_0 = \{P^{(X,Y)} : X \perp Y\}$ and $T_n : \mathcal{X}^n \rightarrow \mathbb{R}$. Then, $\forall\psi \in B_n^2$, define resample $\psi((x_1,y_1),\dots,(x_n,y_n)) = \left((x_{\psi^1(1)}, y_{\psi^2(1)}),\dots,(x_{\psi^1(n)}, y_{\psi^2(n)})\right)$ and sample $\psi_1,\dots,\psi_B$ i.i.d. $Uniform(B_n^2)$. The p-value is then given by $\frac{1 + |\{i\in\{1,\dots B\} : T_n(\psi_i((x_1,y_1),\dots,(x_n,y_n))) \geq T_n((x_1,y_1),\dots,(x_n,y_n))\}|}{1+B}$.

6 Classification

Let $(X_1, Y_1),\dots,(X_n, Y_n),(X_{new}, Y_{new}) \overset{iid}{\sim} P$ with $\forall i \in \{1,\dots,n,new\} : X_i \in \mathcal{X}, Y_i \in \{0,\dots,J-1\} =: \mathcal{Y}$. Loss: $\ell : \mathcal{Y}^2 \rightarrow \mathbb{R}$ (e.g., 0-1-loss). **Wanted**: Classifier $C : \mathcal{X} \rightarrow \mathcal{Y}$, s.t., $\mathbb{E}[\ell(C(X_{new}), Y_{new})]$ is small.
The minimizer of this expect. with the 0-1-loss is the **Bayes classifier**:
$C_{Bayes}(x) := \text{argmax}_{j\in\{0,\dots,J-1\}}\mathbb{P}(Y = j|X = x) =: \text{argmax}_j\pi_j(x)$.

Discriminant Analysis; Multiclass

- **LDA**. Model: We assume $X|Y = j \sim \mathcal{N}(\mu_j, \Sigma)$; and $p_j := \mathbb{P}(Y = j)$.
  Bayes rule: $\rightarrow\mathbb{P}(Y = j|X = x) \propto f_{X|Y=j}(x)\cdot p_j$. Moment estimators:
  $\hat\mu_j = n_j^{-1}\sum_{i:Y_i=j}x_i$, where $n_j = \#\{i : Y_i = j\}$;
  $\hat\Sigma = (n-J)^{-1}\sum_{j=0}^{J-1}\sum_{i:Y_i=j}(x_i - \hat\mu_j)(x_i - \hat\mu_j)^\top$;
  $\hat\Sigma_j = (n_j - 1)^{-1}\sum_{i:Y_i=j}(x_i - \hat\mu_j)(x_i - \hat\mu_j)^\top$ *(for QDA)*;
  $\hat p_j = n_j/n$
  $\hat C_{LDA}(x) = \underset{0\leq j\leq J-1}{\text{argmax}}\,\hat\delta_j(x) = \underset{0\leq j\leq J-1}{\text{argmax}}\left(\left(x - \frac{\hat\mu_j}{2}\right)^\top\hat\Sigma^{-1}\hat\mu_j + log(\hat p_j)\right)$
  Note: Linear decision boundary: $B = \left\{x|\hat\delta_0(x) = \hat\delta_1(x)\right\}$
  $= \left\{x|x^T\hat\Sigma^{-1}(\hat\mu_0 - \hat\mu_1) + log\frac{\hat p_0}{\hat p_1} - \frac{1}{2}\hat\mu_0^T\hat\Sigma^{-1}\hat\mu_0 + \frac{1}{2}\hat\mu_1^T\hat\Sigma^{-1}\hat\mu_1 = 0\right\}$
- **QDA**. Model: Only diff. to LDA: different covariance matrices Σ_j, so $(X|Y = j) \sim \mathcal{N}(\mu_j, \Sigma_j)$. We obtain discr. functions which are quadratic in x:
  $\hat C_{QDA}(x) = \underset{0\leq j\leq J-1}{\text{argmax}}\left(\frac{-log(det(\hat\Sigma_j))}{2} - \frac{(x-\hat\mu_j)^\top\hat\Sigma_j^{-1}(x-\hat\mu_j)}{2} + log(\hat p_j)\right)$
  $\rightarrow$QDA needs $J\cdot p(p+1)/2$ parms. for cov matrices, LDA only $p\cdot(p+1)/2$. Both additionally $J\cdot p$ for means and $J - 1$ for priors

Logistic regression (where we model $\pi(x)$); Binary

Logistic Regression. Only for binary classification **J=2**.
Model: $\forall x \in \mathbb{R}^d : \pi(x) = \frac{1}{1+exp(-g(x))} = \text{logistic}(g(x)) \in (0,1)$, or
$log\left(\frac{\pi(x)}{1-\pi(x)}\right) = g(x)$, where $g : \mathbb{R}^p \rightarrow \mathbb{R}$ (*'log-odds'*)
Linear logistic regr.: $g(x) = \beta^\top x$ and $(Y|X = x) \sim Ber(\pi(x))$. Parameters are fitted using maximum likelihood.
The *likelihood* is $L(\beta, data) = \prod_{i=1}^n\pi_\beta(x_i)^{Y_i}(1 - \pi_\beta(x_i))^{1-Y_i}$,
and *nll*: $-\ell(\beta, data) = -\left(\sum_{i=1}^n Y_i log(\pi_\beta(x_i)) + (1 - Y_i)log(1 - \pi_\beta(x_i))\right) = -\sum_{i=1}^n\left(Y_i\beta^\top x_i - log(exp(\beta^\top x_i) + 1)\right) \rightarrow$nonlinear problem, but convex in β, with Hessian: $X^\top diag(\hat Y(1 - \hat Y))X \succcurlyeq 0$ (psd). $\rightarrow$Can use Newton's GD to find MLE $\hat\beta$.
Rem. 1: Fct. may not be strictly convex. E.g., consider collinearity or perfect separation. Then, there's no unique optimum; can yield convergence probs
Rem. 2: Heuristic: $n \geq max[10\cdot d/\mathbb{P}(Y = 1), 10\cdot d/\mathbb{P}(Y = 0)]$
Multiclass case, J>2: Logistic regression not directly applicable. But:
1) Build J classifiers 'one-vs-rest'
2) Build $J\cdot(J-1)/2$ classifiers 'one-vs-one' and at test time we take class that wins the most pairwise comparisons
3) Multinomial distribution
4) Neural networks (with softmax as last layer)
5) For ordered classes: proportional odds model, polr()

Evaluating classifiers

We have a classifier that models π by $\hat\pi$ which assigns x for some $\theta \in \mathbb{R}$ into a predicted label via $\hat Y(x) := \mathbb{1}_{[\hat\pi(x)>\theta]}$ and $\hat Y_i := \hat Y(x_i)$.
For a **given** θ, define the confusion matrix:

	$\hat Y = 0$	$\hat Y = 1$
$Y = 0$	TN	FP
$Y = 1$	FN	TP

N = TN + FP, and P = FN + TP
Sensitivity(θ) = TPR(θ) := TP/P
Specificity(θ) := TN/N
1 − Specificity(θ) = FPR := FP/N

The **ROC-curve** plots the TPR (y-axis) against the FPR (x-axis) for all values of θ simultaneously! The curve is monotonically increasing: if $\hat P$ increases, then $T\hat P$ increases or stays the same. To compare two classifiers, can look at **AUC** ($= 0.5$ for random guessing, and $= 1$ for perfect class.)

7 Flexible regr. & class. methods

$g(\cdot) : \mathbb{R}^p \rightarrow \mathbb{R}$ denotes either a regr. fct. $\mathbb{E}[Y|X = x]$ or the logit transform in a binary class. problem $log(\pi(x)/(1 - \pi(x)))$.

Additive models

Assume $\exists g_1,\dots,g_d\forall x \in \mathbb{R}^d : g(x) = \sum_{j=1}^d g_j(x^j)$, where x^j is the j-th component of x, the functions $g_j(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ are fully nonparametric and $\forall j : \mathbb{E}[g_j(x^j)] = 0$. The last requirement yields an identifiable model (o.w. we could add and subtract constants). The curse of dimensionality is **not** present in additive models (when all g_j have second continuous derivative, some $\hat g_{add}(.)$ have MSE rate $\mathcal{O}(n^{-4/5})$)
Backfitting: Input is smoother $S_j : \mathbb{R}^n \rightarrow \mathbb{R}^n$, $\mathbf{Y} \rightarrow \hat{\mathbf{Y}}$, where S_j uses $\mathbf{x^j}$.
1) Use $\hat\mu = n^{-1}\sum_{i=1}^n Y_i$. Start with $g_j(\cdot) \equiv 0$ for all $j = 1,\dots,d$
2) Let j cycle through $1,\dots,d,1,\dots,d,\dots$ & compute $\hat{\mathbf{g}}_\mathbf{j} = S_j(\mathbf{Y} - \hat\mu\mathbf{1} - \sum_{k\neq j}\hat{\mathbf{g}}_\mathbf{k})$, where $\mathbf{Y} = (Y_1,\dots,Y_n)^\top$, and $\hat{\mathbf{g}}_\mathbf{j} = \left(\hat g_j(x_1^j),\dots,\hat g_j(x_n^j)\right)^\top$. Stop if individual fcts. $\hat g_j(\cdot)$ do not change much anymore.
3) Normalize the functions $\tilde g_j(\cdot) = \hat g_j(\cdot) - n^{-1}\sum_{i=1}^n\hat g_j(x_i^j)$

Neural Networks

One-layer-feed-forward neural network: $g_k : \mathbb{R}^p \rightarrow \mathbb{R}$, with
$g_k(x) := f(\alpha_k + \sum_{h=1}^q w_{hk}\phi(\tilde\alpha_h + \sum_{j=1}^p\tilde w_{jh}x^j))$. For regr. use k=1, for class. use k=J. Note that f and ϕ are known activation functions. Common activation functions: *sigmoid/logistic, ReLU, GeLU, tanh, softmax....*
Softmax: With $\mathbf{g} := (g_1(x),\dots,g_k(x))^\top$, the softmax is defined as
$\text{softmax}(\mathbf{g})_k := \frac{exp(g_k)}{\sum_{j=1}^k exp(g_j)}$. Numerically: potentially over-/underflow!

Loss fct.:

- For *regression*, use squared loss.
- For *classification*: Use softmax with k=J (#classes) and then negative log-likelihood loss (which is equal to minimizing the cross-entropy loss if using softmax!). The negative gradient of this *nll loss* (=cross-entropy loss) then equals $-\frac{\partial}{\partial g_j} = -\text{softmax}(\mathbf{g})_j + \mathbf{1}_{Y_i=j}$

Regularization: Can add *'weight decay'*, i.e., a term $+\lambda\|\beta\|_2^2$ to loss fct. (where β is a vector of all (or a subset) of the coefs. in the model). Also, one can introduce *'skip layers'* to introduce linear dependencies.
Optimization: (i) Gradients can be computed efficiently (backpropagation); (ii) $+\lambda\|\beta\|_2^2$ yields a gradient step that subtracts a fraction of the weights from them; (iii) Newton-type methods require $\mathcal{O}(\#pars.^3)$ and $\mathcal{O}(\#pars.^2)$ for quasi-Newton $\rightarrow$too slow. Instead use SGD with $\mathcal{O}(knp)$
Tips: (1) center X and scale them (batchnorm); (2) momentum; (3) dropout (only during training); (4) random initialization of weights from $\text{Unif}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$, where m: #inputs, n: #outputs.

Classification and regression trees

Setting. $X \in \mathbb{R}^p, y \in \mathbb{R}$. Given partition $\mathcal{T} := \{R_1,\dots R_M\}, \mathbb{R}^p = \dot\bigcup_{j=1}^M R_j$, and $R_1,\dots,R_M$ are rectangular regions. We consider functions $g_{\text{tree},\beta} : \mathbb{R}^p \rightarrow \mathbb{R}, x \mapsto \sum_{r=1}^M\beta_r\mathbb{1}_{R_r}(x)$.
Given $\mathcal{T}$, what is a good value for β?

- **Regr. & J=2**: $\forall r \in \{1,\dots,M\}$: $\hat\beta_r := \frac{\sum_{i=1}^n Y_i\mathbb{1}_{R_r}(x_i)}{\sum_{j=1}^n\mathbb{1}_{R_r}(x_j)} \rightarrow$minimizes RSS!
- **Multiclass class. (J>2)**: We use $g_{\text{tree},\beta} : \mathbb{R}^p \rightarrow \mathbb{R}^J$, with $g^j_{\text{tree},\beta}(\cdot)$ modeling $\mathbb{P}(Y = j|\cdot)$. Then, $\hat\beta_r^j := \frac{\sum_{i=1}^n\mathbb{1}_{[Y_i=j]}\mathbb{1}_{R_r}(x_i)}{\sum_{k=1}^n\mathbb{1}_{R_r}(x_k)}$

$\rightarrow$minimizes cross-entropy loss: $\sum_{r=1}^{M} n_r \left(-\sum_{j=1}^{J} p_r^j log\left(\beta_r^j\right)\right)$,

where p_r^j is the fraction of times $Y=j$ in rectangle number r and $n_r := \sum_{i=1}^{n} \mathbb{1}_{R_r}(x_i)$ is number of datapoints in rectangle number r.

If $\beta_r^j = p_r^j$, this part is the entropy of p_r, i.e., $H(p_r)$. We want very small entropy in each of the leaves!

We cannot just minimize these losses directly to find partition $\mathcal{T}$, because this would interpolate the data. Instead, we use the **following algorithm**:
1) $M=1, R_1 := \mathbb{R}^p, \mathcal{T} = \{R_1\}$;
2) While $M < M_{max}$: (i) refine partition $\mathcal{T}$ by choosing (r,k,i) (r: rectangle, k: dimension, i: mid-point) and then splitting R_r into R_{left}, R_{right} at the midpoint between x_i^k and next largest $x_\cdot^k$. (ii) Set $M \leftarrow M+1$ and $\mathcal{T} := \mathcal{T} \setminus \{R_r\} \cup \{R_{left}, R_{right}\}$
3) Prune the tree.

- **Details to 2)**. Look for largest reduction in ...
  - Regr.: RSS.
  - Multiclass class.: Cross-entropy (equiv. to largest information gain); Alternatively, look at weighted Gini-impurity
- **Details to 3)**: Pruning = cut the tree at internal node and replace it by a new leaf node: $R_\alpha(\mathcal{T}) := R(\mathcal{T}) + \alpha \cdot \text{size}(\mathcal{T})$, where R is a cost fct. (e.g., RSS or cross-entropy). Changing $\alpha \in [0,\infty)$ yields smaller sub-trees (but not necessarily all of them). Instead of α, we use $Cp = \alpha / R(\mathcal{T}_\emptyset)$ (α divided by cost of empty tree). Use CV to find Cp (1-SE rule)

Random forests

1. Draw n_{tree} bootstrap samples of data
2. For each sample, grow a tree BUT
   - no pruning, maybe use node size to lower bound the # of data points in nodes
   - at each split, randomly choose m_{try} predictors (i.e., dimensions) to determine the split
3. Aggregate all trees (maj. vote for classification, avg. for regression)

Uncertainty quantification: For all bootstrap samples, compute error on out-of-bag sample and average.
Variable importance: E.g., permute data in OOB (may permute out of support!). Don't overinterpret variable importance! Usually: Var. importance = 0 $\rightarrow$var. not needed for maintaining pred. performance
Note: Individually grown trees usually have small bias but high var., while bagging (RF) can help reduce the variance. RF with $m_{try} = p \equiv$ bagging

Random forests as kernels

Assume $\forall$trees $\forall$leaves ℓ, $\exists! i \in \{i,...,n\}$: $x_i \in \ell$. Then,
$g_{\text{RF}}(x) = \frac{1}{n_{\text{tree}}} \sum_{b=1}^{n_{\text{tree}}} \sum_{r=1}^{M} \hat{\beta}_r^b \mathbb{1}_{R_r^b}(x)$
$= \frac{1}{n_{\text{tree}}} \sum_{b=1}^{n_{\text{tree}}} \sum_{r=1}^{M} \sum_{i=1}^{n} Y_i \mathbb{1}_{R_r^b}(x_i) \mathbb{1}_{R_r^b}(x) = \sum_{i=1}^{n} w_i Y_i$ (similar to kernel smoothing), where w_i: Fraction of tress s.t. x & x_i are in the same leaf. For general trees, also factor in how many others are in the same leaf! Sometimes called an ***adaptive kernel***.

8 Kernels & RKHS

Def. kernel: Let $\mathcal{X} \subseteq \mathbb{R}^d$. We call $k\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ a kernel if $\forall m \,\forall x_1,...,x_m \in \mathcal{X}$: $K \in \mathbb{R}^{m\times m}$ with $K_{ij} := k(x_i,x_j)$ is p.s.d, i.e., $\forall c \in \mathbb{R}^m$: $c^\intercal K c \geq 0$, and $\forall x,y$: $k(x,y) = k(y,x)$. Examples: *Gaussian kernel*: $k(x,y) := \exp\left(-\|x-y\|_2^2/2\sigma^2\right)$, *(inh.) polynomial kernel*: $k(x,y) := (\langle x,y\rangle + c)^d$, $c>0$.
Def. RKHS: Let $\mathcal{H}$ be a Hilbert space of functions $f\colon \mathcal{X} \to \mathbb{R}$, meaning
(i) $\forall \lambda \in \mathbb{R}\, \forall f \in \mathcal{H}$: $\forall x \in \mathcal{X}$ $(\lambda f)(x) := \lambda f(x)$, and
(ii) $\forall f,g \in \mathcal{H}$: $\forall x \in \mathcal{X}$ $(f+g)(x) := f(x)+g(x)$
Then, $\mathcal{H}$ is called a RKHS if there is a kernel $k\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ s.t.
$\forall x \in \mathcal{X}$: $k(x,\cdot) \in \mathcal{H}$ and $\forall f \in \mathcal{H}\, \forall x \in \mathcal{X}$: $\langle f(\cdot), k(x,\cdot)\rangle = f(x)$
Rem.: RKHS has inner product, and $\langle\cdot,\cdot\rangle \rightsquigarrow \|\cdot\| \rightsquigarrow d(\cdot) \rightsquigarrow \mathcal{T}$
Prop.: For all kernels, there is an RKHS with this kernel.
Def. feature map: We call $\Phi\colon \mathcal{X} \to \mathcal{H}$, $x \mapsto k(x,\cdot)$ the feature map.
Remark: Clearly, $\langle \Phi(x), \Phi(\tilde{x})\rangle = k(x,\tilde{x})$ ($\rightarrow$in this RKHS, evaluating the dot product is simple: you just evaluate the kernel). Thus, whenever a method uses data in form of dot products, we can 'kernelize' the method:
1) Map data into $\mathcal{H}$ using Φ
2) Apply the method/algorithm using the above remark

Median heuristic for Gaussian kernel

Given $x_1,...,x_n$: $\text{median}(\|x_i - x_j\|_2^2)_{i\neq j} = 2\sigma^2$

Support vector machines (SVM)

Given data $(x_1,Y_1),...,(x_n,Y_n)$, $\mathcal{X} \in \mathbb{R}^d, \mathcal{Y} \in \{-1,1\}$we are looking for a decision boundary $\{z \in \mathbb{R}^d : \langle w, x_i\rangle + b = 0\}$. Here, w and b are not unique! We can require $\min_{i\in\{1,...,n\}} |\langle w,x_i\rangle + b| = 1$ to get unique w,b that describe the hyperplane (in case of linearly seperable data). We have $\text{margin} = 1/\|w\|_2^2$.
$\mathcal{O}_1$: (Hard-SVM): $\min_{w,b} \frac{1}{2}\|w\|_2^2$ s.t. $\forall i \in \{1,...,n\}$: $Y_i(\langle x_i, w\rangle + b) \geq 1$
(Soft): $\min_{w,b} \frac{1}{2}\|w\|_2^2 + C\cdot\sum_{i=1}^{n}\xi_i$ s.t. $\forall i \in \{1,...,n\}$: $Y_i(\langle x_i,w\rangle + b) \geq 1 - \xi_i$.
$\mathcal{O}_1$ is equiv. to $\max_{\alpha\in\mathbb{R}^n} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j Y_i Y_j \langle x_i,x_j\rangle$ s.t. $\sum_{i=1}^{n}\alpha_i Y_i = 0$ and $\forall i \in \{1,...,n\}$: $C \geq \alpha_i \geq 0$, which is kernelizable!

Representer theorem

Let $C\colon \mathbb{R}^n \times \mathcal{X}^n \times \mathbb{R}^n \to \mathbb{R}$ be a loss function and $x_1,...,x_n \in \mathcal{X}$, $Y \in \mathbb{R}^n$, $\lambda \geq 0$. Let $\mathcal{H}$ be an RKHS with repr. kernel k and let $\mathbf{K}$ be the gram matrix, i.e., $\mathbf{K}_{i,j} := k(x_i,x_j)$. Then,
$\hat{f} \in \text{argmin}_{f\in\mathcal{H}} C(Y, x_1,...x_n, f(x_1),...,f(x_n)) + \lambda\cdot\|f\|_{\mathcal{H}}$ if and only if
$\exists \hat{\alpha}_1,...,\hat{\alpha}_n$: $\hat{f}(\cdot) = \sum_{i=1}^{n}\hat{\alpha}_i k(x_i,\cdot)$ and
$\hat{\alpha} \in \text{argmin}_{\alpha\in\mathbb{R}^n} C(Y, x_1,...,x_n, \mathbf{K}\alpha) + \lambda\alpha^\intercal\mathbf{K}\alpha$
Rem.: In particular, $\forall \tilde{x} \in \mathcal{X}$: $\hat{f}(\tilde{x}) = \tilde{\mathbf{K}}\alpha$, where $\tilde{\mathbf{K}}_{1,i} := k(x_i,\tilde{x})$

9 High-dimensional statistics

$X \in \mathbb{R}^{n\times p}, Y \in \mathbb{R}^n$. Consider **OLS**: $\hat{\beta} \in \text{argmin}_\beta \|Y - X\beta\|_2^2$
If $p \leq n$: $\hat{\beta} = (X^\intercal X)^{-1}X^\intercal Y$. If $p > n$: $\hat{\beta}$ not unique. Even if $p \approx n$, $\hat{\beta}_{\text{OLS}}$ potentially badly behaved.

Ridge

$\hat{\beta}^\lambda = \text{argmin}_{\beta\in\mathbb{R}^p} \|Y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$ (don't shrink intercept). Analytical solution: $\hat{\beta}^\lambda = (X^\intercal X + \lambda\mathbb{I})^{-1}X^\intercal Y$. Consider the two cases:

- **X only orthogonal predictors**: $\implies X^\intercal X$ diag.
  Define $(X^\intercal X)_{kk} =: d_k^2, D^2 := X^\intercal X$. Then, $(D^2 + \lambda\mathbb{I})^{-1}_{kk} = \frac{1}{d_k^2 + \lambda}$ and
  $\hat{\beta}_k^\lambda = \frac{1}{d_k^2+\lambda}(X^\intercal Y)_k$ vs. $\hat{\beta}_k^{\text{OLS}} = \frac{1}{d_k^2}(X^\intercal Y)_k$, so $\hat{\beta}_k^\lambda = \frac{d_k^2}{d_k^2+\lambda}\hat{\beta}_k^{\text{OLS}}$
  Thus, Ridge rescales OLS by this factor. If d_k^2 is small (which is the empirical variance of predictor k if we center the columns of X), the shrinkage is larger, and vice versa. Ridge keeps the pred. with large variance larger.
- **X non-orthogonal**: $\rightsquigarrow$ SVD $X = UDV^\intercal \rightsquigarrow$ rotate $\tilde{X} = XV$ (basis transform), $\tilde{X}$ orthogonal. Then, $\hat{\tilde{\beta}}^\lambda = (V^\intercal X^\intercal XV + \lambda\mathbb{I})^{-1}V^\intercal X^\intercal Y \implies$ shrink entries of $\hat{\tilde{\beta}}$ as before, i.e. shrink the coeff. in the rotated system.

Note: XV_j is j-th principal component of X: ridge works well if the signal lives in the space of the first principal components.
Choice of λ: If λ is chosen appropriately, ridge outperforms OLS w.r.t MSE. Note that $\mathbb{E}[\hat{\beta}^\lambda] \neq \beta$, $\text{Var}(\hat{\beta}^\lambda) \leq \text{Var}(\hat{\beta}^{\text{OLS}})$, so ridge is biased for $\lambda > 0$, but variance is lower. Use CV to choose λ.

Lasso

$\hat{\beta}^\lambda = \text{argmin}_{\beta\in\mathbb{R}^p} \|Y - X\beta\|_2^2 + \lambda\|\beta\|_1$ Assume $X^\intercal X = \mathbb{I}$. Then,
$\hat{\beta}_k^\lambda = \text{sign}(\hat{\beta}_k^{\text{OLS}})\cdot\max\{0, |\hat{\beta}_k^{\text{OLS}}| - \lambda/2\}$. Thus, small values of $\hat{\beta}^{\text{OLS}}$ are pushed to zero.
Note: As ridge, lasso is biased. It works well if solution can be approx. by a sparse one (solution needs only $s < min(n,p)$ variables). Lasso can be useful for model selection.
Extensions of Lasso:

- **Elastic net**: $\hat{\beta}^{\lambda,\alpha} = \text{argmin}_\beta \|Y - X\beta\|_2^2 + \lambda\left((1-\alpha)\|\beta\|_1 + \alpha\|\beta\|_2^2\right)$
- **Group Lasso**: Denote $\beta^\intercal = (\beta_1,...,\beta_L)^\intercal$, β_ℓ of length p_ℓ and $\sum_{\ell=1}^{L} p_\ell = p$. Then, $\hat{\beta}^g(\lambda) = \text{argmin}_{\beta\in\mathbb{R}^p} \|Y - \sum_{\ell=1}^{L} X_\ell\beta_\ell\|_2^2 + \lambda\sum_{\ell=1}^{L}\sqrt{p_\ell}\cdot\|\beta_\ell\|_2$. This will give $\hat{\beta}_\ell \equiv 0$ or $\hat{\beta}_{\ell,j} \neq 0 \forall j = 1,...,p_\ell$, so group lasso is sparse on the group-level but non-sparse within groups.

Kernel ridge regression

$\min_\beta \|Y - X\beta\|_2^2 + \lambda\|\beta\|_2^2 \iff \min_{f\in\mathcal{H}} \sum_{i=1}^{n}(Y_i - f(x_i))^2 + \lambda\|f\|_{\mathcal{H}}$, when $\mathcal{H}$ is an RKHS with **linear kernel** $k(x,\cdot) = \langle x,\cdot\rangle$, i.e., $\mathcal{H} = \{f : \mathbb{R}^p \to \mathbb{R}\colon f(x) = x^\intercal\beta$ for some $\beta \in \mathbb{R}^p\}$. Now, we solve this for general kernels/RKHS

- ... via representer theorem:
  The thm. says we can instead solve $\min_{\alpha\in\mathbb{R}^n} \|Y - K\alpha\|_2^2 + \lambda\alpha^\intercal K\alpha$, where $K \in \mathbb{R}^{n\times n}$. The solution satisfies $\hat{Y} = K\hat{\alpha} = K(K+\lambda\mathbb{I})^{-1}Y$
- ... via 'kernelization' of linear ridge regression:
  $\hat{Y} = X\hat{\beta}^\lambda = X(X^\intercal X + \lambda\mathbb{I})^{-1}X^\intercal Y$ which can be reformulated to $XX^\intercal(XX^\intercal + \lambda\mathbb{I})^{-1}Y$ and then kernelized as $K(K+\lambda\mathbb{I})^{-1}Y$

10 Bagging and Boosting

Bagging (Variance reducing technique)

Consider a base model, e.g., CART, which yields a function estimate $\hat{g}(\cdot)\colon \mathbb{R}^p \to \mathcal{T}$ (or $\hat{g}(\cdot) \in [0,1]$ for class. The bagging algorithm is as follows:

1. Generate a bootstrap sample $(X_1^*, Y_1^*),...,(X_n^*, Y_n^*)$ and compute the bootstrapped estimator $\hat{g}^*(\cdot)$
2. Repeat step 1 B times, yielding $\hat{g}^{*1}(\cdot),...,\hat{g}^{*B}(\cdot)$
3. Aggregate the bootstrap estimates to get $\hat{g}_{\text{Bag}}(\cdot) = B^{-1}\sum_{i=1}^{B}\hat{g}^{*i}(\cdot)$

The bagging algorithm is an approximation: $\hat{g}_{\text{Bag}}(\cdot) \approx \mathbb{E}^*[\hat{g}^*(\cdot)]$, which can be made arbitrarily good by increasing B.
Note: With $B = \infty$, we have $\hat{g}_{\text{Bag}}(\cdot) = \hat{g}(\cdot) + (\mathbb{E}^*[\hat{g}^*(\cdot)] - \hat{g}(\cdot))$
$= \hat{g}(\cdot) +$ *bootstrap bias estimate*. Thus, we add the bootstrap bias estimate, and hope for a reduction in variance.
Let $\mathcal{D} = \{(x_i,y_i)\colon i \in 1,...,n\}$, $(X_i,Y_i) \sim \mathcal{T}$, $Y = f_0(x) + \epsilon$, with $f\colon \mathbb{R}^p \to \mathbb{R}$, $f_0(x) = \mathbb{E}[Y|X]$, $\mathbb{E}[\epsilon|X] = 0$. We compare the normal model estimator $\hat{f}^*$ based on $\mathcal{D}$ with the ideal aggregated bootstrap estimator $\hat{f}_{\text{Bag}} = \mathbb{E}_{\mathcal{T}}[\hat{f}^*]$. For a new $(X,Y) \sim \mathcal{T}$, we have, $\mathbb{E}[(Y - \hat{f}^*(X))^2] = \mathbb{E}[(Y - \hat{f}_{\text{Bag}}(X))^2] + \text{Var}(\hat{f}^*(X)) \implies \mathbb{E}[(Y - \hat{f}^*(X))^2] \geq \mathbb{E}[(Y - \hat{f}_{\text{Bag}})^2]$, meaning that the ideal bagging estimator has lower MSE than the normal model without bagging. The additional error corresponds to the variance of the original estimation. Thus, it's useful for **large models** with **low bias** but **high variance**.
In **Subagging** we draw $(X_1^*, Y_1^*),...,(X_m^*, Y_m^*)$ without replacement in step 1 of the algorithm (for some $m < n$). In simple cases, $m = [n/2]$ makes subagging equiv. to bagging.

Boosting

Adaboost: $g\colon \mathbb{R}^p \to \{-1,1\}$, $Y = \{-1,1\}$. The algorithm is:

1. Init. observation weights $w_i = 1/N$, $i = 1,...,N$. For $m = 1,...,M$, do:
   a) Fit classifier $\hat{g}_m$ using w_i
   b) Compute weighted error $\text{err}_m = \sum_{i=1}^{N} w_i \mathbb{1}_{[Y_i \neq \hat{g}_m(x_i)]} / \sum_{i=1}^{N} w_i$
   c) Compute model weight $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$
   d) Update weights $w_i = w_i \exp\left(\alpha_m \mathbb{1}_{[Y_i \neq \hat{g}_m(x_i)]}\right)$
2. Final model: $\hat{G}(x) = \text{sign}\left(\sum_{m=1}^{M}\alpha_m\hat{g}_m(x)\right)$ (weighted maj. decision)

Gradient / L2-Boosting: The idea is to use functions to approximate the (negative) gradient of the loss w.r.t current prediction and then push this prediction step-wise into the direction of the true value. The algorithm is:

1. Initialize $G(x) = g_0(x)$ (e.g., mean or majority vote)
2. For $m = 1,...,M$, do:
   a) For $i = 1,...,N$ compute $r_{im} = -\frac{\partial L(Y_i, G(x_i))}{\partial G(x_i)}$
   b) Fit $g_m(x_i)$ to r_{im} (fit a fct. that predicts negative gradient)
   c) Set $G(x) = G(x) + \gamma g_m(x)$, for some step-size γ

Note: In many cases, r_{im} corresponds to the residuals (or some pseudo-residuals). For regression with the squared loss, $r_{im} = 2(Y_i - \hat{Y}_i)$, and for classification when using the negative likelihood and modelling the log-odds, $r_{im} = Y_i - \hat{p}_i$.

XGBoost: We consider $\hat{Y}_i = \phi(x_i) = \sum_{k=1}^{k} f_k(x_i)$, $f_k \in \mathcal{F}$, where $\mathcal{F} = \{f(x) = w_{q(x)}\}$ is the space of regression trees, so $q\colon \mathbb{R}^p \to T$, $w \in \mathbb{R}^T$, where T is the number of leafs, and w the leaf predictions. We can also write $f(x_i) = \sum_{j=1}^{T}\mathbb{1}_{[j=q(x_i)]}w_j$. We train the functions *forward additive*:
$L^{(t)} = \sum_{i=1}^{n} L\left(Y_i, \hat{Y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$, where $\Omega(f_t) := \gamma T + \frac{1}{2}\lambda\|w\|^2$, so we penalize the number of leafs per tree and add a ridge penalty for the leaf outputs (predictions). This can be simplified to
$\tilde{L}^t = \sum_{j=1}^{T}\left[\sum_{i\in I_j} g_i w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2\right] + \gamma T$, where
$g_i = \frac{\partial L(Y_i, \hat{Y}_i^{(t-1)})}{\partial \hat{Y}_i^{(t-1)}}$, h_i the second derivative and $I_j := \{i\colon q(x_i) = j\}$. Then, for a given structure $q(x)$ for the $t-th$ tree, we get the optimal prediction for that tree by $w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}$. *Note:* usually it is not possible to enumerate all possible tree structures, and in practice we use greedy algorithms (not covered in class). Final prediction is then, $\phi(x_i) = \sum_{k=1}^{K}\alpha * f_k(x_i)$, for learning rate α.

11 Coding Part

Exercise 1

```
x <- seq(from=1, to=40, by=1) # generate features
y <- 1 + 2*x + 5*rnorm(length(x)) # simulate data
X <- cbind(1, x) # model matrix X, model.matrix(fit)
XtXinv <- solve(crossprod(X)) # get (X^TX)^-1
tsd <- sqrt(5^2 * XtXinv[2, 2]) # get true sd of slope
```

Exercise 2

```
df[order(df[,"LifeExp"], decreasing=T), ] # Ordering
anova(small, big) # ANOVA, partial F-test
predict(fit, df_new, "confidence", 0.95) # CI
predict(fit, df_new, "prediction", 0.95) # PI

# Backward/Forward selection using step() --> AIC!
mortal.bw <- step(mortal.full, dir="backward")
mortal.fw <- step(mortal.empty, dir="forward",
                  scope=list(upper=mortal.full,
                             lower=mortal.empty))

# Regsubsets from leaps --> Cp or BIC!
library(leaps)
m <- regsubsets(Y~., data=mort, method=c("backward"))
ms <- summary(m); ncoef <- which.min(ms$cp)
coef(m, ncoef)
```

Exercise 3

```
kde <- function(X, K, h, x) {
    1/(length(X)*h) *rowSums(K(outer(x, X, "-")/h))}
# If dealing with non-equidistant: sort x values!
x <- seq(-1, 1, length = 101) # generate features
n <- length(x)

Snw <- Slp <- Sss <- matrix(0, nrow = n, ncol = n)
# Calculate the hat matrices; they only depends on x!
In <- diag(n)
for(j in 1:n) {
  Snw[,j] <- ksmooth(x, In[,j], kernel = "normal",
                     bandwidth = 0.2, x.points = x)$y
}
df.NW <- sum(diag(Snw)) # degrees of freedom NW est.

# Getting the span parameter for loess such that the
# degrees of freedom match with NW estimator
dflp <- function(span, val) {
  for(j in 1:n)
    Slp[,j] <- loess(In[,j] ~ x, span = span)$fitted
  sum(diag(Slp)) - val}

# What span leads to desired df-value
span <- uniroot(dflp, c(0.2, 0.5), val = df.NW)$root

# Smoothing matrix using loess and smooth.spline
for (j in 1:n) {
  Slp[,j]<-predict(loess(In[,j] ~ x, span), x)
  Sss[,j]<-predict(smooth.spline(x,In[,j],df.NW),x)$y
}

# Calculate degrees of freedom for LP and SS
df.LP <- sum(diag(Slp)); df.SS <- sum(diag(Sss))

# Get the spar value
spar <- smooth.spline(x, In[,1], df = df.NW)$spar

# Calculate predictions and standard errors
estnw <- estlp <- estss <- matrix(0, n, nrep)
senw <- selp <- sess <- matrix(0, n, nrep)

for (i in 1:nrep) {
  y <- m(x) + rnorm(length(x))
  estnw[, i] <- ksmooth(x, y, kernel = "normal",
                 bandwidth = 0.2, x.points = x)$y
  # repeat for estlp and estss (see above how to fit)
  sigma2nw <- sum((y-estnw[,i])^2)/(length(x)-df_nw)
  # repeat for sigma2lp and sigma2ss
  senw[, i] <- sqrt(sigma2nw * diag(Snw %*% t(Snw)))
  # repeat for selp and sess
}

# Coverage ratios
coverage <- function(x,est,se) {
  pos <- x == 0.5 # at position x=0.5 (for pointwise)
  pw <- sum(abs(est[pos,]-m(x)[pos])<=1.96*se[pos,])
  simult <- sum(apply(abs(est-m(x))<=1.96*se,2,all))
  return(c(pw, simult))
}
```

Exercise 4

```
# Helper function for LOOCV
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)
}

# Define regression function to be used on loocv()
reg.fcn.nw <- function(reg.x, reg.y, x) {
  ksmooth(reg.x, reg.y,"normal",h, x.points=x)$y
}

# Computational shortcut for LOOCV
y.fit.nw <- reg.fcn.nw(reg$x, reg$y, reg$x) # y^hat
(cv.nw.hat<-mean(((reg$y-y.fit.nw)/(1-diag(Snw)))^2))

# smooth.spline has built-in CV
est.ss <- smooth.spline(reg$x, reg$y, cv = T, df.NW)
est.ss$cv.crit

# Create K folds for k-fold CV
folds <- sample(cut(seq(1, n), breaks = K,
                labels = FALSE), replace = F)

# Alternative way to calc. hat matrix, dont specify y
library(sfsmisc)
Snw <-hatMat(reg$x,trace=F,pred.sm=reg.fcn.nw,x=reg$x)

# For pred. values outside of range of x-values used
# for estimation in loess:
reg.fcn.lp <- function(reg.x, reg.y, x) {
  predict(loess(reg.y ~ reg.x,enp.target=df.nw,
                surface="direct"), newdata = x)
}
```

Exercise 5

```
require("boot")
tIQR <- function(x, ind) IQR(x[ind])
res.boot <- boot(data = sample40, statistic = tIQR, R)
res.boot$t # stores R rows of bootstrap estimates
res.boot$t0 # stores theta_hat (original estimator)
bci <- boot.ci(res.boot, conf,
               type = c("basic","norm","perc")) # CI

# Helper function to check if ci covers true parameter
check_ci <- function(ci, ty, true.par) {
  # Get confidence interval of type ty from object ci
  type <- c("basic"= "basic", "norm" = "normal",
            "perc" = "percent")[ty]
  ci. <- ci[[type]]
  k <- length(ci.) # need last two entries
  lower <- ci.[k-1]; upper <- ci.[k  ]
  res <- if (true.par < lower) {c(1, 0)}
  else if (true.par > upper) {c(0, 1)}
  else {c(0, 0)}
  names(res) <- c("lower", "upper")
  # return result:
  res
}
```

Exercise 6

```
require(MASS)
fit.gamma<-fitdistr(boogg, "gamma") # MLE uni. distr.
mle <- c(fit.gamma$estimate[1], fit.gamma$estimate[2])

R <- 1000
set.seed(987) # 1) Parametric bootstrap by hand
boot_estimates <- numeric(length=R)
for (i in 1:R) {
  sample_ <- rgamma(length(boogg), shape = mle[1],
                    rate = mle[2])
  boot_estimates[i] <- quantile(sample_, 0.75)
}

set.seed(2020) # 2) Parametric bootstrap using boot
boogg.rg <- function(data, mle) {
  rgamma(length(data), shape = mle[1], rate = mle[2])
}
theta.fun <- function(x) quantile(x, probs = 0.75)
res_boot <- boot(boogg, statistic = theta.fun,
                 sim= "parametric",
                 ran.gen = boogg.rg,
                 mle = fit.gamma$estimate, R = R)
```

Exercise 7

```
require(MASS)
fit.lda <- lda(Species ~ ., data = Iris) # LDA
fit.qda <- lda(Species ~ ., data = Iris) # QDA

# If OOB bootstrap error asked, save indi. as matrix!
index<-matrix(sample.int(n,n*B,replace=T), n, B)

# Initialize the list for LDA and QDA fits
fit_lda<-vector("list", B);fit_qda<-vector("list", B)
for(i in 1:B) { # Use both meths on the boots samples
  ind <- index[, i]
  fit_lda[[i]] <- lda(Species ~ ., data = Iris[ind, ])
  fit_qda[[i]] <- qda(Species ~ ., data = Iris[ind, ])
}
# Determine the mu_hat bootstrap estiamtes
mu_hat_1<-mu_hat_2<-mu_hat_3<-matrix(0,ncol=B,nrow=2)
for(i in 1:B){ # Add means of preds (two) per Species
  mu_hat_all <- fit_lda[[i]]$means
  mu_hat_1[, i] <- mu_hat_all[1,]
  mu_hat_2[, i] <- mu_hat_all[2,]
  mu_hat_3[, i] <- mu_hat_all[3,]}

# Logistic regression N_i ~ Bin(m_i, pi_i)
# with m_i > 1. N_i is #successes
fit<-glm(cbind(N,m-N)~age,family=binomial,data=heart)
optim(c(0,0),neg.ll,data = heart) # Opt. a function
```

Exercise 8

```
# Multinomial regression for multiclass classification
require(nnet)
class_multinom <- multinom(Species ~ . , data = Iris)

# One-vs.-rest log regression for multi classification
Iris1<-Iris;levels(Iris1$Species)<-c("s","not","not")
Iris1$Species <- relevel(Iris1$Species, ref = "not")
fit.1<-glm(Species ~ ., Iris1, family = "binomial")

# Plot ROC and cost curves for classifiers
require(ROCR)
fit<-glm(Survival~.,d.baby, family = "binomial")
pred<-prediction(fit$fitted.values, d.baby$Survival)
perf<-performance(pred, "tpr", "fpr") # could "cost"
plot(perf, main = ...)

# Can also give list with K different CV predictions
# to prediction() and then, average over curves
plot(perf.cv, avg = "threshold", main = ...)

# Fitting GAMs and wraping formulas with sfsmisc
require(sfsmisc) # as.formula from sfsmisc, 2 deg poly
form2<-wrapFormula(f="logupo3␣~␣.", data = d.ozone.e,
                   wrapString="poly(*,␣degree␣=␣2)")
fit2 <- lm(form2, data = d.ozone.e)
require(mgcv) # GAMs
gamForm <- wrapFormula(as.formula("y~."), data = data)
g1 <- gam(formula = gamForm, data = d.ozone.e)
```

Exercise 9

```
# linout = TRUE --> regression, o.w. classification
require(nnet)
fit <- nnet(Fertility ~ ., data = train, size = 15,
            skip = TRUE, decay = 0, linout = TRUE,
            maxit = 100, trace = FALSE)

loss <- function(fitfn, formula = pclass ~ .,
                 data = etitanic, lossmatrix,
                 ..., trace=TRUE, type = "response") {
  modFrame <- model.frame(formula, data = data)
  class <- as.integer(model.response(modFrame))
  fit <- fitfn(formula, data = data, ..., trace=trace)
  pi_hat <- predict(fit, modFrame, type = type)
  L_pi_hat <- pi_hat %*% lossmatrix
  pred <- apply(L_pi_hat, MARGIN = 1, FUN = which.min)
  return(mean(lossmatrix[cbind(class, pred)]))
}
table(df$Class,ypreds)#misclass rate (off-diag/diag)
```

Exercise 10

```
require(rpart)
rp.veh <- rpart(Class ~ ., data = vehicle.dat,
                control = rpart.control(cp, minsplit))

require(rpart.plot)
prp(rp.veh, extra=1, type=1,
    box.col=c('pink', 'palegreen3',
              'lightsteelblue␣2',
              'lightgoldenrod␣1')[rp.veh$frame$yval])
plotcp(rp.veh) # cost-complexity plot

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

# If classification --> response has to be a factor!
require(randomForest)
rf_fit<-randomForest(factor(Class)~.,data=vehicle.dat)
plot(rf_fit) # show OOB error for the diff classes
rf_fit2 <- randomForest(factor(Class)~., # mtry to max
                        mtry=ncol(vehicle.dat)-1,
                        data = vehicle.dat)
```

Exercise 11

```
require(glmnet)
f.ridge <- glmnet(X, Y, alpha=0) # alpha = 0 --> Ridge
f.lasso <- glmnet(X, Y, alpha=1) # alpha = 1 --> Lasso
f.elasticnet <- glmnet(X,Y,alpha=0.5) # Elastic net

# Lasso-traces
plot(f.lasso, xvar="lambda", main="Lasso␣Regression")

# built-in CV
f.elasticnet.cv <- cv.glmnet(X,Y,alpha=0.5,nfolds = 10
plot(f.elasticnet.cv) # --> cv MSE in depend. of lambd
lambda.opt.1se <- f.elasticnet.cv$lambda.1se # 1-SE

# plot number of nonzero coeffs as a fct. of log(lambd
plot(log(f.lasso$lambda),
     apply(coef(f.lasso2), 2, function(x) sum(x != 0)),
     type = "l")

# largest lambda s.t. coeffs >= 15
first.lam.ind = min(which(f.lasso$df >= 16))
coef(f.lasso)[, first.lam.ind] # coeffs for this lambd

# names of nonzero coeffs
names(which(coef(f.lasso)[, first.lam.ind] != 0))
as.numeric(which(coef(f.lasso)[,first.lam.ind]!= 0))-1
```

Miscellaneous

```
# Create train-test split (90%, 10%)
sample <- sample(c(TRUE, FALSE), nrow(df),
                 replace=T, prob=c(0.9, 0.1))

# SVM
library(e1071); iris$Species <- factor(iris$Species)
#kernel SVM with median heuristic
med.heur <- 1/median(dist(iris[samp, 1:4])^2)
svm.med <- svm(Species ~ ., data = iris, subset = samp,
    cost = 1, gamma=med.heur)
Y.pred <- predict(svm.ir, iris[-samp,1:4])

# bagging using only one sample from P
f_Bag <- lapply(1:M, function(m){
  data <- dat_train[sample(1:N, N, replace = TRUE), ]
  f_m <- rpart(y ~ x, data = data, cp = cp)
  return(f_m)
})

# preds of f_Bag: mean of all predictions for every ob
pred_f_Bag <- rowMeans((sapply(f_Bag, function(f_m){
  predict(f_m, dat_test)})))
```