Linear Regression

1. The Linear Model

Assume $Y_i = x_i^\top \beta + \epsilon_i$ or $Y = X\beta + \epsilon$ with $X \in \mathbb{R}^{n\times p}, \beta \in \mathbb{R}^{p\times 1}; (n \geqslant p)$ and $\mathbb{E}[\epsilon_i] = 0, Var(\epsilon_i) = \sigma^2$. X is often augmented with $(1_{N\times 1})$ to use β_1 as bias. $df = n - p[-\text{intercept}]$

2. Least Squares Method

LS estimator is $\hat{\beta} = \arg\min_\beta ||Y - X\beta||_2^2 = (X^\top X)^{-1} X^\top Y$. Estimate $\hat{\sigma}^2 = \frac{1}{n-p}\sum_{i=1}^n (y_i - \hat{\beta}^\top x_i)^2$ with $\mathbb{E}[\hat{\sigma}^2] = \sigma^2$.

Gauss-Markov theorem

LS estimator $\hat{\beta}$ is the **Best Linear Unbiased Estimator**, i.e. for any linear unbiased estimator $\tilde{\beta}$ of β, $\text{var}(\tilde{\beta}) - \text{var}(\hat{\beta})$ is positive semidefinite.

Univormly Minimum Variance Unbiased

Assume x_i are fixed. $\hat{\beta}$ is **UMVU**, i.e. the best estimator among all unbiased estimators.

Assumptions for Linear Model

- i Linear regression equation is correct, i.e. $\mathbb{E}[\epsilon_i] = 0\ \forall i$.
- ii We measure x_i's exactly. Else, need correction (?).
- iii Error is homoscedastic, i.e. $Var(\epsilon_i) = \sigma^2\ \forall i$. Else, use "Weighted LS".
- iv Errors are uncorrelated, i.e. $Cov(\epsilon_i, \epsilon_j) = 0\ \forall i \neq j$. Else "Generalized LS".
- v Errors are jointly normally distributed. Else "Robust Methods".

Moments of least squares estimates

Assume $\mathbf{Y} = X\boldsymbol{\beta} + \epsilon, \mathbb{E}[\boldsymbol{\epsilon}] = \mathbf{0},\ Cov(\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top) = \sigma^2 I$ (all assumptions satisfied). Then

- i $\mathbb{E}[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}$ ($\hat{\boldsymbol{\beta}}$ is unbiased).
- ii $\mathbb{E}[\hat{\mathbf{Y}}] = \mathbb{E}[\mathbf{Y}] = X\boldsymbol{\beta}$ and $\mathbb{E}[\mathbf{r}] = \mathbf{0}$.
- iii $Cov(\hat{\boldsymbol{\beta}}) = \sigma^2(X^\top X)^{-1}$.
- iv $Cov(\hat{\mathbf{Y}}) = \sigma^2 P, Cov(\mathbf{r}) = \sigma^2(I - P), P = X(X^\top X)^{-1}X^\top$.

If additionally $\epsilon_i, \dots, \epsilon_n$ i.i.d. $\sim \mathcal{N}(0, \sigma^2)$, then

- i $\hat{\boldsymbol{\beta}} \sim \mathcal{N}_p(\boldsymbol{\beta}, \sigma^2(X^\top X)^{-1})$
- ii $\hat{\mathbf{Y}} \sim \mathcal{N}_n(X\boldsymbol{\beta}, \sigma^2)$, $\mathbf{r} \sim \mathbb{N}_n(\mathbf{0}, \sigma^2(I - P))$
- iii $\hat{\sigma}^2 \sim \frac{\sigma^2}{n-p}\chi^2_{n-p}$.

Even when normality assumption doesn't hold, central limit theorem is a justification.

3. Regression Theory

Weakly Universal Consistency

A sequence of regression function estimates $\{\hat{f_n}\}$ is called weakly universally consistent if it is weakly consistent, i.e. $\lim_{n\to\infty} \mathbb{E}\left[\int_{\mathbb{R}^p} (\hat{f}_n(x) - f^*(x)^2 P(dx)\right] = 0$. For any estimation method, we can make the rate of convergence arbitrarily slow on some distribution, hence assumptions on $Pr(X, Y)$, minmax approach: $\inf_{\hat{f_n}} \sup_{Pr(X,Y)\in\mathcal{P}}$ of the expectation.

Empirical Risk Minimization (ERM)

Impose restriction on f when minimizing training error:

- i **Roughness penalty** Add term $\lambda J(f)$
- ii **Dictionary methods** Use functions of the form $\sum_{k=1}^K \theta_k \phi_k(\mathbf{x}; \eta_k)$
- iii **Basis function methods** Same as dict but no η_k
- iv **Kernel methods and local regression**: f simple in local neighborhoods (e.g. kNN is constant).

4. Tests and Confidence Regions

T-test

Assume linear model with Gaussian errors (or "large enough" sample size), s.t. $\hat{\boldsymbol{\beta}} \sim \mathcal{N}_p\left(\boldsymbol{\beta}, \sigma^2(X^\top X)^{-1}\right)$ is normally distributed. Then we can test the null-hypothesis $H_{0,j} : \beta_j = 0$ against $H_{A,j} : \beta_j \neq 0$:

$$\frac{\hat{\beta}_j}{\sqrt{\sigma^2(X^\top X)^{-1}_{jj}}} \sim \mathcal{N}(0,1) \Rightarrow T_j = \frac{\hat{\beta}_j}{\sqrt{\hat{\sigma}^2(X^\top X)^{-1}_{jj}}} \sim t_{n-p}$$

under the null-hypothesis $H_{0,j}$. Unknown σ^2 is replaced by $\hat{\sigma}^2$. Note that $t_{n-p} \approx \mathcal{N}$. An individual t-test for $H_{0,j}$ gives the effect of β_j after subtracting the linear effect of all $\beta_{i\neq j}$. Note that in `summary.lm`, the term *Std. Error* is $\sqrt{\hat{\sigma}^2(X^\top X)^{-1}_{jj}} = \sqrt{V\hat{a}r(\hat{\beta}_j)}$. Finally, we can also build a confidence interval using $\hat{\beta}_j \pm \sqrt{\hat{\sigma}^2(X^\top X)^{-1}_{jj}} \cdot t_{n-p;1-\alpha/2}$.

```
confint(fit,level=0.95)
```

Global null hypothesis and ANOVA

We can also check the global null-hypothesis $H_0 : \beta_2 = \dots = \beta_p = 0$ using an *analysis of variance*<, which decomposes

$$||\mathbf{Y} - \bar{\mathbf{Y}}||_2^2 = ||\hat{\mathbf{Y}} - \bar{\mathbf{Y}}||_2^2 + ||\mathbf{Y} - \hat{\mathbf{Y}}||_2^2.$$

Under the global null-hypothesis $\mathbb{E}[\mathbf{Y}] = \mathbb{E}[\bar{\mathbf{Y}}] = const.$ (no effect of predictor variables). $\sigma^2/\hat{\sigma}^2$ yields F-statistic:

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

under the global null-hypothesis H_0. ANOVA also yields *goodness of fit* $R^2 = \frac{||\hat{\mathbf{Y}}-\bar{\mathbf{Y}}||^2}{||\mathbf{Y}-\bar{\mathbf{Y}}||^2}$, which should be close to 1.

```
anova(fit) # global F test
# partial F test - sig. of predictors in .full but not .part
anova(fit.part, fit.full)
```

ANalysis Of VAriance (ANOVA) Table

	sum sq.	df
Reg.	$\|\hat{Y} - \bar{Y}\|^2$	p-1
Error	$\|Y - \hat{Y}\|^2$	n-p
Total	$\|Y - \bar{Y}\|^2$	n-1

- i msq: $sum\ sq./df$
- ii RSE: $\sqrt{MSE}$
- iii Adjusted R^2: $R^2 \cdot (n-1)/df$

5. Checking Model Assumptions

Tukey-Anscombe Plot

Error should fluctuate randomly. Error increases linearly: log-transform $\mathbf{Y} \mapsto \log\mathbf{Y}$. Error increases with $\sqrt{Y}$: square-root-transform $\mathbf{Y} \mapsto \sqrt{\mathbf{Y}}$. Error has parabolic shape: add quadratic term. Error is groups: they have different *Variance*.

```
plot(fit, which=1) # Tukey-Anscombe plot
```

QQ-Plot/Normal-Plot

Plot empirical quantiles of residuals on y versus the theoretical quantiles of $\mathcal{N}(0,1)$ on x. If assumption holds, get straight line with intercept μ and slope σ. Z-shape: long-tailed distr.; Curved: skewed distr.

```
plot(fit, which=2) # QQ-plot (qqnorm, qqline)
```

- i Scale Location (`which=3`): should look random
- ii Cook dist (`which=4`): impact of sample on fit

6. Model Selection

Assume again $\mathbb{E}[\epsilon_i] = 0, Var(\epsilon_i = \sigma^2)$. Address *bias-variance trade-off*. Bias $:= \mathbb{E}[\hat{f}(x)] - f(x)$, variance $:= q/n \cdot \sigma^2$ $(q \leqslant p)$.

Mallows C_p statistic

Let $SSE(d)$ be the residual sum of squares. Then $n^{-1}\sum_{i=1}^n \mathbb{E}\left[(f(x) - \hat{f}(x))^2\right] \approx n^{-1}SSE(d) - \hat{\sigma}^2 + 2\hat{\sigma}^2 d/n$. Thus, we search for the model that minimizes $C_p(\mathcal{M}) = \frac{SSE(d)}{\hat{\sigma}^2} - n + 2d$. Alternatively, use AIC $= 2d - 2\log\hat{L}$, where $\hat{L}$ = maximal value of likelihood, or BIC. AIC is equivalent to C_p for linear Gaussian models.

```
# All subsets regression
require(leaps)
fit.all = regsubsets(y~., data=data)
p.regsubsets(fit.all)
```

Forwards and backwards selection

Forward selection: (i) Start with empty model. (ii) (Greedily) Keep adding variable that reduces the residual sum of squares the most. (iii) When done, pick submodel which minimizes C_p.
Backward selection: (i) Start with full model. (ii) (Greedily) Keep excluding predictor that increases the residual sum of squares the least. (iii) When done, pick submodel which minimizes C_p. Backwards selection typically better but more expensive. When $p \geqslant n$, use forward selection. Both methods prone to overfitting — p-values (and similar values) are *not* valid anymore and effects look too significant.

```
# Backward / forward selection
fit.empty = lm(y~1, data=data)
fit.full = lm(y~., data=data)
fit.bw = step(fit.full, direction="backward")
fit.fw = step(fit.empty, direction="forward",
↪  scope=list(upper=fit.full,lower=fit.empty)
```

```
predict.lm(fit, newdata, interval=..)
poly(formula, degree=d) # polynomial fits of the data in the
↪  predictive variables
up to degree d
```

Density Estimation

Kernel estimator

Estimate density $\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^n w((x - X_i)/h)$. Kernels include (i) rectangular $(w(x) = 0.5 \cdot \mathbb{1}_{|x|<1})$, (ii) triangular, or (iii) Gaussian. We require $\int_{\mathbb{R}} K(x)dx = 1$. The bandwidth parameter h is crucial and determines the "smoothness" of the density estimate.

Choosing a bandwidth h

A simple approach is using k-nearest neighbors, i.e. $h(x) = \max_{x_i\in KNN_k(x)} ||x - x_i||_2$ with tuning parameter k. Note that $\int_{\mathbb{R}} K(x)dx = 1$ might be violated. Naturally, the bandwidth also induces a *bias-variance trade-off*. Note that $MSE(x) = \mathbb{E}\left[\left(\hat{f}(x) - f(x)\right)^2\right] = \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + Var(\hat{f}(x))$, so we can try to minimize the integrated MSE over all points to find the best bandwidth.

Density estimation in higher dimensions

Basically use $\hat{f}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^n K((\mathbf{x} - \mathbf{X}_i)/h)$ with a Kernel that supports vectors. The Gaussian kernel is the only one that is radially symmetric. Note that in higher dimensions, density estimation becomes very hard, due to data points becoming very sparse.

Nonparametric Regression

Nonparametric regression with *one* predictor variable, i.e. $Y_i = m(x_i) + \epsilon_i$ with $\epsilon_{1:n}$ i.i.d and $\mathbb{E}[\epsilon_i] = 0$. We want $m(x) = \mathbb{E}[Y|x]$ and "some" smoothness.

Kernel regression estimator

A "locally weighted" approach yields the NW kernel estimator

$$\hat{m}(x) = \frac{\sum_{i=1}^n \omega_i Y_i}{\sum_i \omega_i} = \arg\min_{m_x\in\mathbb{R}} \sum_{i=1}^n \omega_i(Y_i - m_x)^2 \qquad (3.1)$$

with $\omega_i = K\left(\frac{x_i - x}{h}\right)$ a kernel centered at x_i and bandwidth h. As h small → large then (high variance) → (high bias). For x_i equidistant there exists $h_{opt} = f(\sigma^2_\epsilon, m''(x))$ which can be iteratively found.

```
ksmooth(x, y, kernel="normal", bandwidth=0.2, x.points=x)$y
```

```
# automatic bandwidth
fit.lo<-lokerns(X, Y, x.out=X, hetero=TRUE, is.rand=TRUE)
fit.gl<-glkerns(X, Y, x.out=X, hetero=TRUE, is.rand=TRUE)
```

- i Bias ($mean(\hat{y} - y_{true})$): high → under fitting
- ii Variance (measure of fluctuation): high → over fitting

Local polynomial regression estimator

Instead of finding a local constant m_x we can also find a *local polynomial*, i.e. we replace m_x with $\beta_1 + \sum_{i=2}^p \beta_i(x_i - x)^{i-1}$ (usually $p = 2$ or $p = 4$). Often better at edges and yields first derivative.

```
fit.loess <- loess(y ~ x, data=data.frame(x=x, y=y_pert),
↪  span=0.2971339, surface='direct')
fit.loess.pred <- predict(fit.loess, newdata=x)
```

The hat matrix $\mathcal{S}$

We want to construct $\mathcal{S}$ with $\hat{\mathbf{Y}} = \mathcal{S}\mathbf{Y}$, i.e. the linear operator mapping the labels to the predictions. Given the regression (smoothing) function s, we compute $\mathcal{S}_{.j} = s(\mathbf{x}, \mathbf{e}_j, h)$ with $\mathbf{e}_j$ the j-th unit vector. Then $Cov(\hat{m}(\mathbf{x})) = Cov(\mathcal{S}\mathbf{Y}) = \mathcal{S}Cov(\mathbf{Y})\mathcal{S}^\top = \sigma^2_\epsilon\mathcal{S}\mathcal{S}^\top$. Set $df = \mathbf{tr}\mathcal{S}$ and estimate $\hat{\sigma}^2_\epsilon = \sum_{i=1}^n (Y_i - \hat{m}(x_i))^2/(n - df)$. Then

- $\widehat{s.e.}(\hat{m}(x_i)) = \sqrt{\widehat{Var}(\hat{m}(x_i))} = \hat{\sigma}_\epsilon\sqrt{(\mathcal{S}\mathcal{S}^\top)_{ii}}$
- $\hat{m}(x_i) \approx \mathcal{N}\left(\mathbb{E}[\hat{m}(x_i)], Var(\hat{m}(x_i))\right)$
- $I = \hat{m}(x_i) \pm 1.96 \cdot \widehat{s.e.}(\hat{m}(x_i))$ → (pointwise) CI

```
# Construct S matrix (or hatMatrix() from sfsmics)
N <- length(x); Eye <- diag(N)
S.nw <- matrix(0, nrow=N, ncol=N)
for (j in 1:N) {
  y_ <- Eye[, j]
  S.nw[, j] <- ksmooth(x, y_, kernel="normal",
  ↪  bandwidth=0.2, x.points=x)$y
}
# Compute standard error
est.nw <- ksmooth(x=x, y=y, kernel="normal", bandwidth=0.2,
↪  x.points=x)$y
sig_sq.nw <- sum((y - est.nw)^2) / (N - sum(diag(S.nw)))
se.nw <- sqrt(sig_sq.nw * diag(S.nw \%*\% t(S.nw)))
```

Smoothing splines and penalized regression

High-order polynomials do not work, so splines are used. We discuss splines *without* having to specify the knots. Find $\arg\min_{m\in C^0(\mathbb{R})}\sum_{i=1}^n(Y_i - m(x_i))^2 + \lambda\int_{\mathbb{R}} m''(z)^2 dz$. Note that the minimizer is *finite dimensional* — it is a cubic spline that can be computed using a set of basis functions $m_\lambda(x) = \sum_{j=1}^n \beta_j B_j(x)$ or $||\mathbf{Y} - B\boldsymbol{\beta}||^2 + \lambda\boldsymbol{\beta}^\top\Omega\boldsymbol{\beta} \Rightarrow \hat{\boldsymbol{\beta}} = (B^\top B + \lambda\Omega)^{-1}B^\top\mathbf{Y}$. Choose λ on the scale of $df = \mathbf{tr}(\mathcal{S}_\lambda)$. Note that this is Ridge-type regression, which saves us from being overparametrized (n points, n parameters). In the exam, this is *not* considered "standard" least squares.

```
fit.ss = smooth.spline(x, y, df=df)
ss.preds = predict(fit.ss, newdata=x)$y
```

Cross Validation

Let $(X_1, Y_1), \dots, (X_n, Y_n)$ i.i.d $\sim P$. We would like to compute $\mathbb{E}_{(X_{new}, Y_{new})}[\rho(\mathbf{Y}_{new}, \hat{m}_{train}(X_{new}))]$.

Constructing cross-validation datasets

Approaches include

Validation set: —
Leave-one-out CV: $n^{-1}\sum_{i=1}^n \rho\left(Y_i, \hat{m}^{(-i)}_{n-1}(X_i)\right)$ ca. unbiased.
k-fold CV: $K^{-1}\sum_{i=1}^K |\mathcal{B}_k|^{-1}\sum_{i\in\mathcal{B}_k}\rho\left(Y_i, \hat{m}^{(-\mathcal{B}_k)}_{n-|\mathcal{B}_k|}(X_i)\right)$.
Smaller variance than LOOCV.
Random division: Like k-fold, but build $\mathcal{B}_k$ by sampling without replacement ($\approx$ 10%). Usually fastest.

Tricks using hat matrix

For linear fitting operators and the loss $\rho(y, x) = (y - x)^2$ we can exploit the hat matrix and get the full LOOCV result in a single step using

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

It can be cheaper to just compute $\mathbf{tr}(\mathcal{S})$ (instead of all $\mathcal{S}_{ii}$), which leads to the *generalized cross-validation*

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

The two equations coincide if $\mathcal{S}_{ii} = c \ \forall i$.

One-standard-error rule

Choose the most parsimonious model whose error is no more than one standard error above the error of the best model.

Bootstrap

Efron's *parametric* and *nonparametric bootstrap* can be described as "simulating from an estimated model" and can be used for *statistical inference (confidence intervals and testing)* and *estimating the predictive power of a model or algorithm.*

Nonparametric Bootstrap

Let $Z_{1:n}$ i.i.d $\sim P$ with $Z_i = (X_i, Y_i), X_i \in \mathbb{R}^p, Y_i \in \mathbb{R}$, and let $\hat{\theta}_n = g(Z_{1:n})$ be an estimator. We would like to know the *distribution of* $\hat{\theta}_n$. We approximate $\mathbf{P}$ by the *empirical distribution* $\hat{P}_n$. Then we can repeatedly sample $Z^*_{1:n} \sim \hat{P}_n$ independently and compute $\hat{\theta}^*_n = g(Z^*_{1:n})$. The histogram (or any density estimator) then describes the distribution of $\hat{\theta}^*_n$. The algorithm reads

a) Sample (with replacement) $Z^*_{1:n}$ i.i.d $\sim \hat{P}_n$.

b) Compute the bootstrapped estimator $\hat{\theta}^*_n = g(Z^*_{1:n})$.

c) Repeat B times to obtain $\hat{\theta}^{*1:B}_n$.

d) Approximate $\mathbb{E}^*[\hat{\theta}^*_n] \approx B^{-1}\sum_{i=1}^{B}\hat{\theta}^{*i}_n$ and $Var^*(\hat{\theta}^*_n) \approx (B-1)^{-1}\sum_{i=1}^{B}\left(\hat{\theta}^{*i}_n - B^{-1}\sum_{j=1}^{B}\hat{\theta}^{*j}_n\right)^2$. Then α-quantile of $\hat{\theta}^*_n \approx$ empirical α-quantile of $\hat{\theta}^{*1:B}_n$.

Central limit theorem

Let X_i be a random variable with $\mathbb{E}[X_i] = \mu$ and $Var(X_i) = \sigma^2$. Then $n^{-1}\sum_{i=1}^{n} X_i \overset{n\to\infty}{\to} \mathcal{N}(\mu, \sigma^2/n)$.

Bootstrap consistence

Consistency of the bootstrap typically holds if the limiting distribution of $\hat{\theta}_n$ is Normal and if $Z_{1:n}$ are i.i.d. Mathematically, for an increasing sequence a_n and $\forall x$, $\mathbb{P}[a_n(\hat{\theta}_n - \theta) \leqslant x] - \mathbb{P}^*[a_n(\hat{\theta}^*_n - \hat{\theta}_n) \leqslant x] \overset{P}{\to} 0$ as $n \to \infty$. Then $Op^*(\hat{\theta}^*_n)/Op(\hat{\theta}_n) \overset{P}{\to} 1$ with $Op \in \{Var, \mathbb{E}\}$.

Bootstrap confidence intervals

Given bootstrap consistence, we can compute confidence intervals:

i quantile / quantile: $[q_{\hat{\Theta}^*}(\alpha/2), q_{\hat{\Theta}^*}(1-\alpha/2)]$

ii rev. quantile: $[\hat{\Theta} - q_{\hat{\Theta}^*-\hat{\Theta}}(1-\alpha/2), \hat{\Theta} - q_{\hat{\Theta}^*-\hat{\Theta}}(\alpha/2)]$

iii normal: $2\hat{\Theta} - \overline{\hat{\Theta}^*} \pm q_X(1-\alpha/2) \cdot \hat{sd}(\hat{\Theta})$ ($X \sim \mathcal{N}(0,1)$; corrects for bias $\hat{\Theta} - \hat{\Theta}^*$)

Note $\hat{q}_\alpha = q^*_\alpha - \hat{\theta}_n$ with $q^*_\alpha = \alpha$-bootstrap quantile of $\hat{\theta}^*_n$. Thus $[\hat{\theta}_n - \hat{q}_{1-\alpha/2}, \hat{\theta}_n - \hat{q}_{\alpha/2}] = [2\hat{\theta}_n - q^*_{1-\alpha/2}, 2\hat{\theta}_n - q^*_{\alpha/2}]$.

```
require("boot")
tm = function(x, ind) {mean(x[ind], trim = 0.1)}
res.boot = boot(data=sample, statistic=tm, R=1000)
boot.ci(res.boot, conf=0.95, type=c("basic","norm","perc"))
# "basic"=reverse quantile, "norm"=normal, "perc"=quantile
```

```
# CIs by hand
require(MASS); mle = fitdistr(boogg, "gamma")$estimate
boot.est = matrix(NA, nrow=R, ncol=1)
for (i in 1:R) {
  boogg.s = rgamma(N, shape=mle[1], rate=mle[2])
  # boogg.s = sample(boogg, N, replace=T) # NP
  boot.est[i] = quantile(boogg.s, probs=0.75)
}; a = 0.05
# Quantile / Percentile
quantile(boot.est, probs=c(a/2, 1-a/2))
# Normal
mean.est = mean(boot.est)
sd.hat = sqrt(1/(R-1)*sum((boot.est - mean.est)^2))
2*est-mean.est + c(-1,1)*qnorm(1-a/2)*sd.hat
# Reverse quantile / Basic quantile
est - quantile(boot.est-est, probs=c(1-a/2, a/2))
```

Double bootstrap

Idea: Find α' s.t. actual coverage of bootstrap CI $I^*(1-\alpha')$ is equal to α.

i Draw BS sample Z^*. Sample from Z^* to obtain Z^{**}. Compute CI $I^{**}(1-\alpha)$ for $\hat{\Theta}^*$ based on B draws Z^{**}. Compute coverage of $\hat{\Theta}$ by I^{**} (1 or 0).

ii Repeat i) M times to obtain M coverage values. Compute mean to obtain actual coverage of I^{**}.

iii Vary α' and repeat previous steps to find α' with coverage$(I^{**}(1-\alpha')) = 1-\alpha$. Use CI $I^*(1-\alpha')$.

Parametric Bootstrap

Assume $Z = (Z_1, .., Z_n)$ *i.i.d.* $\sim P_\Theta$. Fit $\hat{\Theta}$ = MLE and generate samples Z^* *i.i.d.* $\sim P_{\hat{\Theta}}$. Usually better than nonparametric version when $P_{\hat{\Theta}}$ is a good fit (e.g. known model structure P) and few data points available.

```
require(MASS); fit.gamma = fitdistr(boogg, "gamma")
fun.theta = function(x) {quantile(x, probs=0.75)}
fun.gen = function(x, mle) {rgamma(length(x), shape=mle[1],
↪  rate=mle[2])}
res.boot = boot(data, fun.theta, R=1000, sim="parametric",
↪  ran.gen=fun.gen, mle=fit.gamma$estimate);
```

Bootstrap error estimate

Generalization error (loss $\rho(y, m(x))$) of model m (fitted to full data set) can be estimated by fitting models $m^{*,i}$ to bootstrap samples and computing

- errors on full data set $e^{*,i} = n^{-1}\sum_{i=1}^{n}\rho(y, m^{*,i}(x_i))$
- OOB errors $e^{*,i}_{ob} = n^{-1}_{ob,i}\sum_{i=1}^{n_{ob,i}}\rho(y_{ob,i}, m^{*,i}(x_{ob,i}))$

The error of m is then approximated by $R^{-1}\sum_{i=1}^{R} e^{*,i}$.

Classification

Given $(X_1, Y_1), \ldots, (X_n, Y_n)$ i.i.d. with $Y_i \in \{0, \ldots, J-1\}$, determine $\pi_j(x) = \mathbb{P}[Y = j \mid X = x] \ \forall j = 0, 1, \ldots, J-1$. The optimal classifier is $\mathcal{C}_{Bayes}(x) = \arg\max_{0 \leqslant j \leqslant J-1}\pi_j(x)$. Then Bayes risk for the 0-1-loss is $\mathbb{P}[\mathcal{C}_{Bayes}(X_{\text{new}}) \neq Y_{\text{new}}]$.

Discriminant analysis

LDA: Assume $X \mid Y = j \sim \mathcal{N}(\mu_j, \Sigma), \mathbb{P}[Y = j] = p_j,$. Then by Bayes formula

$$\pi_j(x) = \frac{f_{X|Y=j}(x) \cdot p_j}{\sum_{k=0}^{J-1} f_{X|Y=k}(x) \cdot p_k}$$

with each $f_{X|Y=j}$ a Gaussian $\mathcal{N}(\mu_j, \Sigma)$. We can estimate μ_j and Σ by the (closed-form) MLEs, and we also need a prior for Y, which often is fixed as $\hat{p}_j = n_j/n$. This results in $\hat{\delta}_j(x) = (x - \hat{\mu}_j/2)^\top\Sigma^{-1}\hat{\mu}_j + \log(\hat{p}_j)$ with linear (in x) decision boundaries $\hat{\delta}_j(x) = \hat{\delta}_{j'}(x)$ and $\mathcal{C}(x) = \arg\max_j \hat{\delta}_j(x)$.
QDA: Now we assume different Σ_j for each class and obtain quadratic decision boundaries $\hat{\delta}_j(x) = -\log(\det(\hat{\Sigma}_j))/2 - (x - \hat{\mu}_j)^\top\hat{\Sigma}_j^{-1}(x - \hat{\mu}_j)/2 + \log(\hat{p}_j)$. The price: $J \cdot p(p+1)/2$ parameters (for all Σ_j) vs. $p(p+1)/2$ (symmetry of Σ) for a single Σ (μ $(J \cdot p)$ and priors $(J-1)$ are the same),

```
require(MASS)
class_lda = lda(x=df[, c("x1", "x2")], grouping=df[, "y"])
```

Logistic regression for binary classification

Given some model $g : \mathbb{R}^p \to \mathbb{R}$ (e.g. a linear model) we can use the logistic transform $\pi \mapsto \log(\pi/(1-\pi))$ to get probabilities: $\log(\pi(x)/(1-\pi(x))) = g(x)$ and $\pi(x) = 1/(1+\exp(-g(x)))$. This implies $Y_i \sim \text{Bernoulli}(\pi(x_i))$ (e.g. weighted coin flip). The likelihood is thus $\prod_{i=1}^{n}\pi(x_i)^{Y_i}(1-\pi(x_i))^{1-Y_i}$. We typically estimate $\boldsymbol{\beta}$ using gradient descent. As $n \to \infty$ we can asymptotically compute the standard errors $\widehat{s.e.}(\hat{\beta}_j)$ and t-test statistics $\hat{\beta}_j/\widehat{s.e.}(\hat{\beta}_j) \sim \mathcal{N}(0,1)$ (under $H_{0,j} : \beta_j = 0$).

```
fit = glm(Y~., data=data, family="binomial")
mean((predict(fit, type="response") > 0.5) == data$Y)
```

Linear predictors

Note that both *LDA* and *Logistic regression* are *linear* in the prediction variables. For LDA that comes from the Gaussian assumption (i.e. "linearization" of the true distribution), for Logistic regression it comes from the linear log-odds function.

Multiclass case ($J > 2$)

a) J classes → J binary variables: $\tilde{\pi}_j(x) = \frac{\hat{p}i_j(x)}{\sum_{j=0}^{J-1}\hat{\pi}_j(x)}$

b) Using *multinomial distribution* (parametric linear logistic) (see `multinom`)

c) "Reference class" $\log(\pi_j(x)/\pi_0(x)) = g_j(x)$

d) Pairwise 1-vs-1, fitting $\binom{J}{2} \cdot p$ parameters

e) Exploiting "ordered" classes with proportional odds

Manually Calculating Predicted Class

i Predict class for x: $\arg\min_{k\in[1,J]}\sum_{j=1}^{J} L(j,k) \cdot \pi_j(x)$

ii E.g loss for predicting class c : $\sum_{j=1}^{J} L(j,c) \cdot \pi_j(x)$

Flexible regression and classification methods

We fight the *curse of dimensionality* by making some structural assumptions (although staying with methods $g(\cdot) : \mathbb{R}^p \to \mathbb{R}$ of nonparametric nature).

1. Additive models

Decompose multivariate function as $g_{add} : \mathbb{R}^p \to \mathbb{R}, x \mapsto \mu + \sum_{j=1}^{p} g_j(x_j)$ with $g_j(\cdot) : \mathbb{R} \to \mathbb{R}$, $\mathbb{E}[g_j(X_j)] = 0$. The zero-mean requirement for each $g_j(\cdot)$ makes the problem well posed. This approach is a generalization of linear models, and similarly can not model interaction terms $g_{j,k}(x_j, x_k)$. Due to the way they are constructed, additive linear models *avoid the curse of dimensionality.*

To fit a model, let $\mathcal{S}_j$ be a smoothing technique (e.g. *Nadaraya-Watson kernel estimators*). Then, the **backfitting** algorithm works as follows:

- Compute $\hat{\mu} = n^{-1}\sum_{i=1}^{n} Y_i$ and initialize $\hat{g}_j(\cdot) := 0$.
- Cycle through the indices $j = 1, 2, \ldots, p, 1, 2, \ldots$ and update $\hat{g}_j = \mathcal{S}_j(\mathbf{Y} - \hat{\mu}\mathbf{1} - \sum_{k\neq j}\hat{g}_k)$. Stop each function at convergence.
- Normalize the functions: $\tilde{g}_j(\cdot) = \hat{g}_j(\cdot) - n^{-1}\sum_{i=1}^{n}\hat{g}_j(X_{ij})$.

This basically makes the algorithm repeatedly solve the 1-dimensional fitting problem. The algorithm may be slow but often works and can use any 1-dimensional fitting technique.

When fitting Additive models in R with the function `gam`, the smoothers $\mathcal{S}_j$ are penalized regression splines, and the degrees of freedom for each spline (i.e. each variable) will be determined through cross-validation.

```
fit<-gam(Y~s(x1)+s(x2)+.., data=data) # wrapFormula()
plot(fit, pages=1, shade=TRUE) # show effect predictor
sfsmisc::TA.plot(fit, labels="o")
```

2. Trees

Classification and Regression Trees

Let $g_{tree}(\mathbf{x}) = \sum_{r=1}^{M}\beta_r 1_{[\mathbf{x}\in\mathcal{R}_r]}$, where $\{\mathcal{R}_1, \ldots, \mathcal{R}_M\}$ is a partition of $\mathbb{R}^p$. The function is piecewise constant. When a partition is given, we can estimate $\hat{\beta}_r = \sum_{i=1}^{n} Y_i 1_{[\mathbf{x}\in\mathcal{R}_r]}/\sum_{i=1}^{n} 1_{[\mathbf{x}\in\mathcal{R}_r]}$. For multiclass classification $\hat{\pi}_j(\mathbf{x}) = \sum_{i=1}^{n} 1_{[Y_i=j]}1_{[\mathbf{x}\in\mathcal{R}_r]}/\sum_{i=1}^{n} 1_{[\mathbf{x}\in\mathcal{R}_r]}$ for $\mathbf{x} \in \mathcal{R}_r$.
Greddy algorithm to find axes parallel partition:

i Initialize $M = 1$ subset $\mathcal{P} = \{\mathcal{R} = \mathbb{R}^p\}$

ii Split $\mathcal{R}$ at d in dimension j, where d is from the set of midpoints of observed values. Select j, d s.t. neg. log-likelihood decrease is maximized by refinement.

iii Apply ii) to one cell of the current partition (select like above). Add the resulting two cells and remove the refined one.

iv Iterate iii) until until specified max. partition size is achieved.

v Prune tree by removing leaves resulting in smalles increase in some (CV) metric.

The size of a tree $\mathcal{T}$ is the number of leaves (=1 + cuts).

For some goodness-of-fit measure $\mathcal{R}(\mathcal{T})$ (e.g. SSE, NLL), the cost-complexity measure is $\mathcal{R}_\alpha(\mathcal{T}) = \mathcal{R}(\mathcal{T}) + \alpha \cdot \text{size}(\mathcal{T})$. For some α, we thus choose $\mathcal{T}(\alpha) = \arg\min_{\mathcal{T}\subset\mathcal{T}_M}\mathcal{R}_\alpha(\mathcal{T})$. The parameter α is chosen by CV. The 1 s.e. rule says: Choose smallest tree such that its performance is at most one standard error larger than the minimal one.

```
num_leafs<-length(unique(tree$where))
depth <- trunc(log2(max(as.integer(row.names(tree$frame)))))
```

```
library(rpart); require(rpart.plot);
tree = rpart(y~., data=data, control=rpart.control(cp=0.0,
↪  minsplit=1)) # unregularized tree
plotcp(tree); cps = tree$cptable # cp = α/R_α(T_∅)
idx.min = which.min(cps[, "xerror"])
std.min = cps[idx.min, "xstd"]
cp.1se =
↪  cps[abs(cps[,"xerror"]-cps[idx.min,"xerror"])<std.min,]
pruned.tree = prune.rpart(tree, cp=cp.1se[1, "CP"])
```

```
rf <- randomForest(Boston.train, y.train, xtest=Boston.test,
↪  ytest=y.test, ntree=ntree, mtry=ncol(Boston.train))
rf$mse[ntree] # OOB error; rftestmse[ntree]
tail(rf$err.rate) # cumulative error rate
```

Bagging and Boosting

Bagging and Subbagging

Bootstrap **agg**regat**ing** (bagging) (mostly on trees), uses $\hat{g}(\cdot) : \mathbb{R}^p \to \mathbb{R}$ and ensembles them (which comes at the loss of interpretability).

i Generate bootstrap sample $(X^*_1, Y^*_1), \ldots, (X^*_n, Y^*_n)$ and compute $\hat{g}^*_{i=1}(\cdot)$. Repeat B times.

ii Aggregate bootstrap estimates with $\hat{g}_{\text{Bag}}(\cdot) = B^{-1}\sum_{i=1}^{B}\hat{g}^*_i(\cdot) \approx \mathbb{E}^*[\hat{g}^*(\cdot)]$.

Note that $\hat{g}_{\text{Bag}}(\cdot) = \hat{g}(\cdot) + \underbrace{(\mathbb{E}^*[\hat{g}^*(\cdot)] - \hat{g}(\cdot))}_{\text{bootstrap bias estimate}}$. We can reduce variance at price of higher bias (at least for trees). In fact, for many x, $Var(\hat{g}_{\text{Bag}}(x)) < Var(\hat{g}(x))$. We can use larger trees (higher variance) to balance the bias-variance trade-off.
For **Sub**sample **agg**regat**ing** (Subbagging), we draw $(X^*_1, Y^*_1), \ldots, (X^*_m, Y^*_m)$ without replacement (e.g. with $m = \lfloor n/2 \rfloor$), which can be cheaper overall and is equivalent to Bagging in some simple settings.

AdaBoost.M1

Misc

i RStudio: `Tools` → `Global Options` → `Code` → `Display`

ii ROC: FPR (x) vs TPR (y)

iii P[$> |t|$]: large (> 0.05) → $\beta_j \approx 0$ → β_j not relevent; small → $\beta_j! = 0$ → β_j relevant & reject $H_{j,0}$ (R: `pt(..)`)

```
print(.., digits=..); table(df$Class, y.pred); residuals(..)
factor(Class) # for classification; optim(..); termplot(..)
```