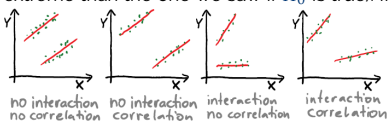
Linear Regression

$y_i = \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i$ (note: $x_{i1} \equiv 1$, so β_1 is intercept) where $\epsilon_1, ..., \epsilon_n$ independent, $E(\epsilon_1) = 0, \text{Var}(\epsilon_i) = \sigma^2$ (homoscedasticity)
LSE: $\hat{\beta} = \text{argmin}_\beta \|Y - Xb\|_2^2 = (X^\top X)^{-1} X^\top Y \sim \mathcal{N}_p(\beta, \sigma^2 (X^\top X)^{-1})$
$\hat{\sigma}^2 \approx \frac{1}{n-p} RSS$ also if error Gaussian then: $\hat{Y} \sim \mathcal{N}_n(X\beta, \sigma^2 P)$, error $e \sim \mathcal{N}_n(0, \sigma^2(I-P))$, $\hat{\sigma}^2 \sim \sigma^2/(n-p) \cdot X_{n-p}$ where $P = X(X^\top X)^{-1} X^\top$
Categorical Variables: For two levels: $y_i = \beta_1 x_{i1} + ... + \beta_p x_{ip} + \lambda d_{is} + \epsilon_i$ so if i is in category, then $d_{is} = 1$ else $d_{is} = 0$. This acts as a different intercept ($E(y_i) - E(y_j) = \lambda$). If more categories, add more dummy variables.
Interaction: dummy can also influence slope: add term $\delta d_i x_i$, can influence interaction between predictors: add term $\delta x_{i2} x_{i3}$, can influence other categorical variable: add term $\delta d_{i1} d_{i2}$.
Measuring Goodness of Fit
$H_0 : y = X\beta + \epsilon$ with $\beta_j = 0$ $H_A : y = X\beta + \epsilon$ with $\beta_j \neq 0$
Under H_0 : $\frac{\hat{\beta}_j - (E[\hat{\beta}_j]=0)}{\sqrt{\sigma^2 (X^\top X)^{-1}_{jj}}} \sim \mathcal{N}(0,1)$ t-statistic: $\frac{\hat{\beta}_j}{\sqrt{\hat{\sigma}^2 (X^\top X)^{-1}_{jj}}} \sim t_{n-p}$
P-Value: P(obs. a value of the test stat. that is as extreme or more extreme than the one we saw if H_0 is true). If $< \alpha$ then reject H_0.



Linear Regression

```
fit <- lm(y~x1+x2) # Fit only x1 and x2 (so p=3)
predict(fit, pred.data.frame)
# Manual fit
X <- cbind(1, x1, x2) # p = 3
XtX.inv <- solve(t(X) %*% X)
beta.hat <- XtX.inv %*% t(X.int) %*% y
res <- y - X.int %*% beta.hat # Residuals
RSE <- sqrt(sum(res^2)/(n-p)) # Residual std. error.
 ↪   Est. of the sd of the noise in the linear model
se.x1 <- RSE * sqrt(XtX.inv[2, 2]) # Std. error of x1
t.val.x1 <- beta.hat[2] / se.x1 # T value of x1
p.val.x1 <- 2*pt(abs(t.val.x1), df=n-p, lower=F)
# R squared: proportion of Var(y) that is explained
 ↪   by the fitted linear model.
RSE <- sqrt(sum(residuals(fit)^2)/(n-p))
RSS <- sum(res^2) # Residual sum of squares
TSS <- sum((y - mean(y))^2) # Total sum of squares
R.sq <- 1 - RSS / TSS
AdjR2 <- 1 - (RSS/(n-p))/(TSS/(n-1))
# Alternative t-value
coef <- summary(fit1)$coefficients
t1 <- coef["x1","Estimate"]/coef["x1","Std. Error"]
# Finding p-values
fit.smaller <- lm(y ~ x1)
anova(fit.smaller, fit, fit.all)
# Overall F-Test
fit.empty <- lm(y ~ 1, data=...) # Empty model
anova(fit.empty, fit) # Compare models
# Alternative F-test
Ftest <- summary(fit)$fstatistic
pval <- 1 - pf(Ftest[1], df1=Ftest[2], df2=Ftest[3])
# Categ. var. by hand & LOOCV
a1 <- (levels(shelveloc)[2]==shelveloc)*1
lcv<-mean((residuals(fit)/(1-lm.influence(fit)$h))^2)
```

R Diagnostic plots: #1 Tukey-Anscombe Plot the points follow the line, else $E(\epsilon) = 0$ violated. #2 Q-Q Plot should follow line, else error not Gaussian (still all fine). #3 Scale-Location: should be flat, else $\text{Var}(\epsilon_i) = \sigma^2$ violated (p-values wrong). #4/#5 Cook distance: shows if some data points have a larger impact on the fit than others (outliers). Note: cannot detect if the residuals are correlated with these plots!

Confidence Intervals

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

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

Prediction Interval
For new point x_0: $\hat{y}_0 = x_0^\top \beta \pm \hat{\sigma}^2 \sqrt{1 + x_0^\top (X^\top X)^{-1} x_0} \cdot t_{1-\alpha/2,n-p}$

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

Confidence Intervals

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

K Nearest Neighbors

Non-parametric method: $\hat{f}(x) = 1/k \sum_i yi$ where y_i are the k nearest neighbors of x. If k is larger has less variance, more bias. If k is smaller has more variance, less bias. Fails if there are lots of useless predictors.

KNN

```
library(kknn)
dfTrain=data.frame(y=Ytrain,x=Xtrain)
dfTest=data.frame(x=Xtest)
fit.kknn <- kknn(y ~ ., dfTrain,dfTest,k=8)
predTest=predict(fit.kknn) # predictions on dfTest
library(class) # Alternative library for knn
knn(train, test, k=5)
```

Cross Validation

Can be used for *model assessment* (estimate test MSE) and *model selection* (choose tuning parameters, variable selection). But not both at the same time (use double CV instead).
Validation set: split data into two halves, train on one, test on the other (most bias). **k-Fold:** same, but with many folds. Try all folds for test and average metrics over the folds (in between). $\text{Var}(\theta_k') = 1/K \cdot \hat{\text{Var}}(MSEs)$ **LOOCV:** extreme version where each data point is a fold (least bias).
$\theta_k = \frac{1}{k} \sum_{i=1}^k \frac{1}{|I_k|} \sum_{i \in I_k} (y_i - \hat{f}^{-I_k}(x_i))^2$, $\theta_L = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}^{-i}(x_i))^2$

Bootstrap

Sample uniform from data points with replacement, compute bootstrapped estimator. For a large dataset $x_1, ..., x_n$ the probability that x_1 is contained in a random bootstrap dataset is: $1 - (1 - 1/n)^n \approx 2/3$ (for large n, limit goes to $1 - 1/e$).
Bootstrap Consistency
For an increasing sequence a_n (often $\sqrt{n}$) where a_n^{-1} is the convergence rate of $\hat{\theta}_n$: $P(a_n(\hat{\theta}_n - \theta) \leq x) - P^*(a_n(\hat{\theta}_n - \hat{\theta}_n) \leq x) \to^P 0$ as $n \to \infty$. This holds when $\sqrt{n}(\hat{\theta}_n - \theta)$ is asympt. normal. Allows to estimate $\text{Bias}(\hat{\theta}_n) = E[\hat{\theta}_n] - \theta$ by $E^*[\hat{\theta}_n^*] - \hat{\theta}_n$. Can also estimate $\text{Var}^*(\hat{\theta}_n)$ by $\text{Var}^*(\hat{\theta}_n^*)$.
Resersed Quantile Bootstrap CI: $[\hat{\theta}_n - Q_{\hat{\theta}^* - \hat{\theta}}(1-\alpha/2), \hat{\theta}_n - Q_{\hat{\theta}^* - \hat{\theta}}(\alpha/2)]$ (type="basic"), **Normal Bootstrap CI:** Assums $\hat{\theta}_n$ to be asympt. normal: $\hat{\theta}_n \pm Q_z(1-\alpha/2) \hat{sd}(\hat{\theta}_n)$ where $z \sim \mathcal{N}(0,1)$ and $\hat{sd}(\hat{\theta}_n) = \sqrt{(\text{Var}(\hat{\theta}_n^*))}$ (type="norm"), **Quantile Bootstrap CI:** not theoret. justified unless $\hat{\theta}_n$ is symm.: $[Q_{\theta_n^*}(\alpha/2), Q_{\theta_n^*}(1-\alpha/2)]$ (type="perc"). Same as *reversed quantile bootstrap CI* if $\hat{\theta}_n^* - \hat{\theta}_n$ is symm. around 0, **Bootstrap T:** Rely on $t = \frac{\hat{\theta}_n - \theta}{\hat{sd}(\hat{\theta}_n)}$ and $t* = \frac{\hat{\theta}_n^* - \hat{\theta}_n}{\hat{sd}(\hat{\theta}_n^*)}$ to be asympt. equal: $[\hat{\theta}_n - \hat{sd}(\hat{\theta}_n) \cdot Q_{t*}(1-\alpha/2), \hat{\theta}_n - \hat{sd}(\hat{\theta}_n) \cdot Q_{t*}(\alpha/2)]$
Note: $\hat{sd}(\hat{\theta}_n)$ is computed as above and $\hat{sd}(\hat{\theta}_n^*)$ is computed using a 2nd layer bootstrap.
Parametric Bootstrap: Assume data is generated by some parametric distr. (e.g. $\mathcal{N}(\mu, \sigma^2)$), est. the param., then create new data sets from this distr. Works only well if distr. is approx. correct.
Smoothed Bootstrap: Given data $Z_1, ..., Z_n \sim_{i.i.d} P$, we estimate P by some smooth (non-parametric) estimate $\tilde{P}_n$, then generate bootstrap samples from $\tilde{P}_n$. In between non-param. and param. bootstrap. Works well if P is indeed smooth.

Bootstrap

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

Permutation Test

Non-parametric, simple model that works with any test statistic. P-values and type I error control exact/approximate (not asymptotic), but needs computational power and not everything can be modeled in this way (e.g. individual coefficients in LR)
1. Pick a test stat. that measures some difference between groups
2. Consider all possible permutations (or randomly permute) to obtain a permutation distribution.
3. Compare observed value to permutation distribution

Wilcoxon Test
Non-parametric, unpaired, robust test. $H_0 : F_1 = F_2$, $H_A : F_1$ shifted compared to F_2. Compute ranks of randomly switched sign (different group assignment), reject H_0 if observed rank over critical value of rank distribution.

Permutation Test & Wilcoxon Signed Rank Sum Test

```
# Permutation test
fit <- lm(y~X)
obsF <- summary(fit)$fstatistic[1]
res.f <- rep(NA, 10000)
for (i in 1:10000){
  y <- y[sample(1:nrow(X), nrow(X))]
  fit.tmp <- lm(y~X)
  res.f[i] <- summary(fit.tmp)$fstatistic[1]
}
pval<-(sum(obsF<=res.f,na.rm=T)+1)/(length(res.f)+1)
# Permutation Wilcoxon Signed Rank Sum Test
diff <- immer$Y1 - immer$Y2
V.obs <- sum(rank(abs(diff)) * (diff > 0))
V <- numeric(100000)
for(i in 1:100000){
  perm<-diff * sample(c(1,-1),nrow(immer),replace=T)
  V[i] <- sum(rank(abs(perm)) * (perm > 0))
}
p.value <- table(V >= V.obs)["TRUE"]/length(V)
# Automatic
wilcox.test(diff, alternative = "greater")
```

Multiple Testing

	H_0 true	H_A true	
H_0 not reject.	U true neg.	T false neg. Type II error	$m - R$
H_0 reject.	V false pos. Type I error	S true pos.	$R = V + S$
	m_0	$m - m_0$	m

Capital letters represent RV. Only R is observable. m is fixed and known, m_0 is fixed and unknown. If $m_0 = m$ then *global null*.
$P(\text{type I error}) = P(\text{rejecting } H_0 \text{ when } H_0 \text{ is true}) = \alpha$
$P(\text{type II error}) = P(\text{not rejecting } H_0 \text{ when } H_A \text{ is true}) = \beta$
Power of a test $1 - \beta$ **False discovery proportion (FDP):** $Q = V/R$ (note: $V/R = 0$ if $V = R = 0$) **False discovery rate (FDR):** $E(Q)$ Expected proportion of false discoveries among all disc. **Family wise error rate (FWER):** $P(V \geq 1)$ Prob. of making one or more false discoveries. Note: controlling the FWER is more strict than controlling the FDR: if $V \geq 1$, then $Q = V/R \leq 1$ and if $V = 0$, then $Q = V/R = 0$. So $\mathbb{1}_{\{V \geq 1\}} \geq Q$: $\text{FWER} = P(V \geq 1) = E[\mathbb{1}_{\{V \geq 1\}}] \geq E[Q] = \text{FDR}$. If $m = m_0$ then FWER = FDR.

Bonferroni Correction
Idea: $\text{FWER} = P(V \geq 1) = P(\text{at least one false rejection among tests } T_1, ..., T_m) = P(\cup_{i=1}^m \{\text{false rejection in test } T_i\}) \leq \sum_{i=1}^m P(\{\text{false rej. in test}\}) \leq \sum_{i=1}^m \alpha = m \cdot \alpha$ so we set the signif. level of individual tests to $\alpha' = \alpha/m$, then $\text{FWER} \leq m\alpha' = \alpha$. Power: if the tests are indep. and $m = m_0$ then $\text{FWER} = 1 - (1-\alpha)^m$ which is $\approx \alpha \cdot m$ for small α and moderate m. If the tests are correlated: too conservative.
Benjamini-Hochberg
Let $p(1) \leq p(2) \leq ... \leq p(m)$ be ordered p-values. Let i_0 be the largest i, s.t. $p(i) \leq q \cdot i/m$. Reject all $H_{(i)}$ with $i \leq i_0$. For independent test statistics (or p-values) this controls the FDR at level q, i.e. $\text{FDR} = q \cdot m_0/m \leq q$.
Westfall Young Permutation Procedure
Provides *weak control of the FWER* (i.e. under the global null).
Given a data $X \in \mathbb{R}^{n \times (m+1)}$ size and $y \in \{0,1\}^n$. If $m = m_0$ then one can permute the y-values:
1. Repeat many times: premute y-cols, do a two sample test (e.g. Wilcoxon) for each x_j col. (comparing $x_j[y == 1]$ and $x_j[y == 0]$). Let p_j for $j = 0, ..., m$ be the corresp. p-value. Store $\min(p_1, ..., p_m)$.
2. Set δ to the empirical α-quantile of the permutation distribution of $min(p_1, ..., p_m)$.
3. Reject any null hypothesis where the two-sample test on the original data has p-value $\leq \delta$.

Westfall Young Permutation Procedure

```
nr.sim <- 1000
min.p.values <- numeric(nr.sim)
for(sim in 1:nr.sim) {
  y_perm = sample(y, length(y), replace = FALSE)
  min.p.values[sim] <- min(apply(x, 2, function(j)
    ↪   chisq.test(x = j, y = y_perm)$p.value))
}
delta <- quantile(min.p.values, probs = 0.05)
table(p.values < delta)
```

Model Selection

Criteria for model selection (for linear models): Mallow's $C_p = \frac{1}{n}(RSS + 2d \cdot \hat{\sigma}^2)$, $AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d \cdot \hat{\sigma}^2)$, $BIC = \frac{1}{n\hat{\sigma}^2}(RSS + \log(n) d \cdot \hat{\sigma}^2) = -2 \cdot \log(\hat{L}) + d \cdot \log(n)$ where $\hat{L}$ is the maximized value of the likelihood of the model, $AdjR^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$.
Shrinkage Methods
Assume centered variable (no intercept). Also need so standardize data: **Ridge regression:** $\hat{\beta}_s^{ridge} = \text{argmin}_\beta RSS(\beta) + \lambda \|\beta\|_2^2 = (X^\top X + \lambda I)^{-1} X^\top y$ **Lasso:** $\hat{\beta}_s^{lasso} \text{argmin}_\beta RSS(\beta) + \lambda \|\beta\|_1$, **Elastic Net:** $\hat{\beta}_s^{elastic} = \text{argmin}_\beta RSS(\beta) + (1-\alpha)\lambda \|\beta\|_1 + \alpha\lambda \|\beta\|_2^2$ ($\alpha = 1$: ridge, $\alpha = 0$: lasso).
Adaptive Lasso: Lasso with penalty weights: $\hat{\beta}_s^{adapt.lasso} \text{argmin}_\beta RSS(\beta) + \lambda \sum_{j=1}^p w_j |\beta_j|$. Take a $\sqrt{n}$ consistent estimate $\hat{\beta}$ of β (e.g. least squares Choose $\gamma > 0$, then set $\hat{w}_j = 1/|\hat{\beta}|^\gamma$. This asymptotically selects the right covariates and has optimal estimation rate.
Group Lasso: Predictors are divided into L groups of size $p_1, ..., p_L$ s.t. $\sum p_i = p$. $\hat{\beta}_\lambda^{gr.lasso} = \text{argmin}_\beta RSS(\beta) + \lambda \sum_{l=1}^L \sqrt{p_l} \|\beta\|_2$. (if L=p, we get Lasso). Acts like Lasso on a group level. Useful if there are categorical variables with > 2 categories (put all corresponding dummy variables in a group).

Best subset: $\hat{\beta}_s^{subset} = \text{argmin}_{\beta,\|\beta\|_0 \le s} RSS(\beta)$. 1) Fit M_0 (null model = sample mean. 2) For $k = 1,...,p$ fit $\binom{p}{k}$models that contain exactly k predictors and select best (smallest RSS): M_k. 3) Select best among $M_0,...,M_p$ using CV or criteria.
Forward stepwise: 1) Fit M_0 2) For $k = 0,...,p-1$ fit all $p-k$ models with 1 additional predictor and select best (smallest RSS): M_k. 3) Select best among $M_0,....,M_p$ using CV or criteria.
Backward stepwise: 1) Fit M_p (full model). 2) For $k = p,p-1,...,1$: fit all k models that drop one perdictor in M_k. Choose best (smallest RSS): M_{k-1}. 3) Select best among $M_0,...,M_p$ using CV or criteria.

Model Selection

```
# Lasso/Ridge
library(glmnet)
grid <- 10^seq(from=10,to=-2,length=100)
ridge <- glmnet(x[train,], y[train], alpha=0,
 ↪   lambda=gridm thres=1e-12)
lasso <- glmnet(x[train,], y[train], alpha=1,
 ↪   lambda=gridm thres=1e-12)
# Best subset
regfit.full=regsubsets(Salary~., data=..., nvmax=19)
# Forward stepwise
regfit.full=regsubsets(Salary~., data=..., nvmax=19,
 ↪   method="forward")
# Backward stepwise
regfit.full=regsubsets(Salary~., data=..., nvmax=19,
 ↪   method="backward")
# Mallow Cp
library(leaps)
m <- leaps::regsubsets(y~., data=train, nvmax=10)
mo <- which.min(summary(m)$cp)
form <- as.formula(paste("y~",
    paste(names(coef(m,mo))[-1],collapse="+"), sep =
    ↪   ""))
fit <- lm(form, data=test)
# Workaround for categorical variables
predict.regsubsets <- function(reg, new.data, id) {
  form <- as.formula(reg$call[[2]])
  mat <- model.matrix(form, new.data)
  coefi <- coef(reg, id=id)
  return(mat[,names(coefi)]%*%coefi)
}
n <- nrow(data)
folds <- sample(cut(1:n, 10, labels=F), n, replace=F)
for (fold in 1:10) {
  test.fold <- which(folds == fold)
  data.train <- data[-test.fold,]
  data.test <- data[test.fold,]
  m <- nrow(data.train)
  cv.f <- sample(cut(1:m, 10, labels=F), m,
   ↪   replace=F)
  cv.errors <- matrix(nrow=10, ncol=19)
  for (k in 1:10) {
    cv.tf <- (cv.f == k)
    cv.m <- regsubsets(Salary~., data.train[-cv.tf,],
     ↪   nvmax=19)
    for (i in 1:19) {
      pred <- predict(cv.m, data.train[cv.tf,], id=i)
      cv.errors[k, i] <-
       ↪   mean((pred-data.train[cv.tf,]$Salary)^2)
    }
  }
  m <- regsubsets(Salary~., data=data.train,
   ↪   nvmax=19)
  best.cp <- which.min(summary(m)$cp)
  best.cv <- which.min(apply(cv.errors, 2, mean))
  pred.cp <- predict.regsubsets(m, data.test,
   ↪   best.cp)
  pred.cv <- predict.regsubsets(m, data.test,
   ↪   best.cv)
}
# For double CV with glmnet, can use cv.glmnet
inner.folds <- factors(folds[folds!=i])
levels(inner.folds) <- 1:(k-1)
inner.folds <- as.numeric(inner.folds)
```

Beyond Linearity

Basis Functions: $y_i = \beta_0 + \beta_1 b_1(x_i) + ... + \beta_k b_k(x_i) + \epsilon_i$
Polynomial Regression: $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + ... + \beta_d x_i^d + \epsilon_i$. Easy to fit, but unstable near boundaries. As basis function: $b_j(x_i) = (x_i)^j$. Can also use better (orthogonal) basis functions (R automatically uses these). If orthogonal, can check until which coefficient it is significant. Gives same result and accuracy as monomial basis.
Step Functions: Create k cut points $c_1,...,c_k$ in the range of x. Then basis functions: $b_j(x_i) = 1_{\{c_j < x_i \le c_{j+1}\}}$.
Regression Splines: Combines polynomial regression with step functions. A piecewise degree d polynomial with continuity in dimensions $0,...,d-1$. Generally has $(k+1)(d+1)$ parameters and $k \cdot d$ constraints, so $k+d+1$ degrees of freedom. (e.g. **piecewise cubic** has $4(k+1)$ parameters and $3k$ constraint, so $k+4$ degrees of freedom. Basis functions: $h(x,\xi) = (x-\xi)_+^3$ (0 for all values $\le \xi$). $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \gamma_1 h(x_i,\xi_1) + ... + \gamma_k h(x_i,\xi_k) + \epsilon_i$). **Natural splines:** Regression splines with additional boundary constraints: linear outside of outer knots (so k degrees of freedom).
Smoothing Splines: $G = \{g : [a,b] \to \mathbb{R} : g''$ exists and $\int_a^b g''(x)^2 dx < \infty\}$ is the class of functions to consider: $\hat{g} = \text{argmin}_{g \in G} \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int_a^b g''(x)^2 dx$. Note: if $\lambda = 0$, $\hat{g}$ is any function in G that passes through all data points. If $\lambda = \infty$, we get the least squares estimate. Shrunken version of natural spline with knots at $x_1,...,x_n$.
Generalized Additive Models: $y_i = \beta_0 + \sum_{j=1}^p f_j(x_{ij}) + \epsilon_i$. More general than linear model, does not allow for interactions automatically → no curse of dimensionality.
Local Regression: Algorithm for local regression at $X = x_0$: 1) Gather the fraction $s = k/n$ of trianing points there x_i are closest to x_0. 2) Assign weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero and the closest has the highest weight. All but these k nearest neighbors get weight zero. 3) Fit a weighted least squares regression of the y_i on the x_i using the weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize $\sum_{i=1}^n K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$. 4) The fitted value of x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.
Backfitting: Let $s_j : (u_1,...,u_n)^T \to (\hat{u}_1,...,\hat{u}_n)^T$ be a smoother (e.g. multiple linear regression). Order influences #iterations
Initialize $\hat{\mu} = \frac{1}{n}\sum_{i=1}^n y_i, \quad \hat{g}_j = 0 \quad \forall j$
Do until convergence:
  for each $j = 1,...p$:
    $\hat{g}_j \leftarrow s_j(y - \hat{\mu}\mathbb{1} - \sum_{k \ne j} \hat{g}_k)$
    $\hat{g}_j \leftarrow \hat{g}_j - \frac{1}{n}\sum_{i=1}^n \hat{g}_j(x_{ij})\vec{1}$

Using GAMs and KNN

```
# Polynomials
lm(wage~poly(age,4)) # Orthogonal polynomials
lm(wage~poly(age, 4, raw=T)) # Monomial basis
lm(wage~age+I(age^2)+I(age^3)+I(age^4)) # Alternative

# Polynomial regression
fit <- lm(wage ~ poly(age, 4), data=Wage)
fit2 <- lm(wage ~ poly(age, 3), data=Wage)
anova(fit, fit2) # Check if significantly better
# Automatically check significance (if orthognal)
round(coef(summary(fit2), 2), 4)

# Regular spline
fit <- lm(wage ~ bs(age, knots=c(25,40,60)),
 ↪   data=Wage)
# Natural spline
fit <- lm(wage ~ bs(age, df=4), data=Wage)
# Smoothing spline
fit <- smooth.spline(age, wage, df=16)
fit <- smooth.spline(age, wage, cv=T)

library(gam)
# s() for smoothing spline, lo() for loess
model.gam <- gam(y~s(X1, 4), data = dtrain)
mse.gam <- mean((ytest - predict(model.gam,
 ↪   dtest))^2)

library(FNN)
pred.knn <- knn.reg(train = matrix(dtrain[,1],ncol=1),
 ↪   test = matrix(dtest[,1], ncol=1), y = ytrain,
 ↪   k=i)
  results.knn[i] <- mean((ytest - pred.knn$pred)^2)
```

Backfitting Algorithm for MLR

```
backfit <- function(x, y, n, p, o, eps) {
  mu.hat <- mean(y) # Compute overall mean
  g <- matrix(0, nrow=n, ncol=p) # Initialize g
  converged <- FALSE
  while(!converged) {
    old.g <- g
    beta0.hat <- mu.hat
    beta.hat <- numeric(p+1)
    for(i in o) { # o: order e.g. 1:p
      r <- y - mu.hat - rowSums(g[,-i])
      fit <- lm(r~x[,i])
      g[,i] <- fit$fitted
      beta0.hat <- beta0.hat + fit$coeff[1]
      beta.hat[1+i] <- fit$coeff[2]
    }
    if(max(colSums((old.g - g)^2)/colSums(old.g^2)) <
     ↪   eps) {
      converged <- TRUE
    }
    beta.hat[1] <- beta0.hat
  }
  return(beta.hat)
}
```

Tree Based Methods

$y_i = \sum_{r=1}^M \beta_r \mathbb{1}_{x_i \in R_r} + \epsilon_i$. Where $R_1,...,R_M$ are regions (we limit ourselves to rectangular regions).
Recursive Binary Splitting: Greedy method to find the regions: For all predictors find the best cutting point, then take the cutting point that minimizes the error $\sum_{i:x \in R_1}(y_i - \bar{y}_{R_1})^2 + \sum_{i:x \in R_2}(y_i - \bar{y}_{R_2})^2$. Stop when region contains less than 5 elements.
Pruning: A deep tree T_0 can overfit. Pruning is a possible solution: For $\alpha > 0$: find $\text{argmin}_{T \subset T_0} err(T) + \alpha|T|$. ($\alpha$ is tuning parameter, find via CV).
Classification Trees: At each split, try to improve node purity measured by gini index: $I(D) = [\frac{n_L}{n} I(D_L) + \frac{n_R}{n} I(D_R)] > 0$ where the subtrees are $I(D_R) = \hat{p}(1-\hat{p})$ where $\hat{p} = \frac{\#yes}{\#yes+\#no}$. Can predict class probability $\hat{p}_k(x) =$ *proportion of observations with class k in leaf node that contains x.*

Using Trees

```
train.ind <- sample(n, round(n/2), replace = FALSE)
train.data <- Carseats[train.ind,]
test.data <- Carseats[-train.ind,]
# Create and plot tree
cs.tree <- tree(Sales ~ . , train.data)
plot(cs.tree)
text(cs.tree, pretty=1)
# Predict using tree
test.pred <- predict(cs.tree, test.data)
(MSE <- mean((test.data$Sales - test.pred)^2))
# Prune tree
cv.carseats = cv.tree(cs.tree, FUN=prune.tree)
best.size <- cv.carseats$size
 ↪   [which.min(cv.carseats$dev)]
pruned.tree <- prune.tree(cs.tree, best = best.size)
```

Bootstrap Aggregating (Bagging)

Bagging for Regression: For data $(X_1,Y_1),...,(X_n,Y_n)$ and base procedure $\hat{g}(\cdot) : \mathbb{R}^p \to \mathbb{R}$, take B bootstrap samples $\hat{g}_{bag}(x) = \frac{1}{B}\sum_{b=1}^B \hat{g}^{*b}(x)$ where $\hat{g}^{*b}$ is the estimate based on the b-th bootstrap sample. No pruning, since variance of single tree not a problem as we average. Linear predictions are the same under bagging, so only interesting for non-linear estimates. For regression can only improve or stay the same.
Bagging for Classification: $\hat{g}(\cdot) : \mathbb{R}^p \to \{1,...,k\}$. $\hat{g}(x) = \text{argmax}_{k=1,...,K} \sum_{b=1}^B \mathbb{1}_{\hat{g}^{*b}(x)=k}$ (majority vote). Can also get class probability: $\hat{p}_k^{bag}(x) = \frac{1}{B}\sum_{b=1}^B \hat{p}_k^{*b}(x)$. Can also be formulated as $\hat{g}^{bag}(x) = \text{argmax}_{k=1,...,K} \hat{p}_k^{bag}(x)$ (better if interested in class probabilities, sometimes even helps accuracy). Bagging a good classifier can improve performance, but bagging a bad classifier can decrease performance.
Out-of-Bag Error: Some bags have not trained on a particular sample. Can predict this only by the bags that have not been trained on it (should be $\sim 1/3$) for all samples and average to get a valid estimate for the test error.
Random Forests: Essentially bagged trees. Have B bootstrap samples → create trees. They reduce dependence between tree estimates by only allowing a random subset of predictors at each split. Default: regression $p/3$, classification $\sqrt{p}$. (in R option mtry).

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

Random Forests

```
library(randomForest)
cs.bag <- randomForest(Sales ~ . , train.data,
 ↪   mtry=p-1, importance = TRUE) # Bagging
cs.forest <- randomForest(Sales ~ . , train.data,
 ↪   mtry=p/3, importance = TRUE) # Random Forest
importance(cs.forest) # Importance of predictors
```

R-Help

```
# Workaround for automatic dimension collapse
matrix(X[test.fold,], nrow=n/k)

# Test if an element is in a list
if ("X1" %in% names(coef(m,mo)))

# Fit distribution to data vector
library(MASS)
fit.gamma <- fitdistr(boogg, "gamma")

# Remove NA
thuesen <- thuesen[!is.na(thuesen[,2]),]
data_comp <- data[complete.cases(data),]
# Creating categorical variables
High=ifelse(Carseats$Sales<=8,"No","Yes")
# Standardize data
scaled.dat <- scale(data)

# Anova test to determine if there is a significant
# difference between models. Anova uses RSS and DoF
# of largest (last) model, so use ascending order!
anova(fit.0, fit.1, fit.2, fit.3)

# Loess smoother
span <- c(0.1, 0.2, 0.3, 0.45, 0.7, 1)
lo <- loess(y ~ x, span=span[i])
prediction <- predict(object=lo, x)

# Given fixed x, error distribution and true param.:
# Power of test simulation. Know that y ~ poly(x, 3)
 ↪   + err (for typeI error: do same with y = err)
results.power <- numeric(n.sim)
for (i in 1:n.sim) {
    err <- rgamma(n, ...) - 2
    y <- beta.0 + beta.1 * I(x) + beta.2 * I(x^2) +
     ↪   beta.3 + I(x^3) + err
    fit.power <- lm(y ~ I(x) + I(x^2) + I(x^3))
    f1 <- summary(fit.power)$fstatistic
    p.val.power <- 1 - pf(f1[1], f1[2], f1[3])
    results.power[i] <- p.val.power < 0.05
}
power <- mean(results.power)

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