1 Linear Regression

$y_i = \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i$ ($x_{i1} \equiv 1$, so β_1 is intercept) $Y = X\beta + \epsilon$

$\epsilon_1, ..., \epsilon_n$ are indep., $\mathbb{E}(\epsilon_1) = 0, \text{Var}(\epsilon_i) = \sigma^2$ (homoscedastic)

Least square solution $\hat{\beta} = \text{argmin}_\beta \|Y - Xb\|_2^2 = (X^\top X)^{-1} X^\top Y \sim \mathcal{N}_p(\beta, \sigma^2 (X^\top X)^{-1})$

$RSS = \|Y - X\hat{\beta}\|_2^2 \qquad \hat{\sigma}^2 \approx \frac{1}{n-p} RSS$. If $\epsilon \sim \mathcal{N}$ then: $\hat{Y} \sim \mathcal{N}_n(X\beta, \sigma^2 P)$, $\mathbf{r} \sim \mathcal{N}_n(0, \sigma^2(I-P))$, $\hat{\sigma}^2 \sim \sigma^2/(n-p) \cdot \chi_{n-p}$ where $P = X(X^\top X)^{-1} X^\top$

Interpreting R-output: Residuals: `y-model$fitted.values`;
Residual Standard Error: $\hat{\sigma}$; Coeff. Estimate = $\hat{\beta}$;
Coeff. Std Error = $\hat{\sigma}(X^\top X)^{-1}$; Coeff t value: Estimate / Std. Error;

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)
# Alternative t-value
coef <- summary(fit1)$coefficients
t1 <- coef["x1","Estimate"]/coef["x1","Std. Error"]
# Poly regression
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
```

1.1 Tests and model selection

Entry-wise test

$H_0 : y = X\beta + \epsilon$ 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.

ANOVA (Analysis of variance)

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

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

Under the null hypothesis $H_0 : \beta = 0$ We have that:

$F = \frac{\|\hat{Y} - \overline{Y}\|^2/(p-1)}{\|Y - \hat{Y}\|^2/(n-p)} \sim F_{p-1, n-p}$; Bigger F = better fit = lower P-value

P-Values & ANOVA

```
fit.smaller <- lm(y ~ x1)
# Partial F Test. Anova uses RSS and DoF
# of largest (last) model, so use ascending order!
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])
```

R^2 (Coefficient of determination): the proportion of the total variation of the response Y around its mean $\overline{Y}$ that is explained by the regression $\hat{Y}$ (via the ANalysis Of VAriance decomposition)

$R^2 = \frac{\|\hat{Y} - \overline{Y}\|^2}{\|Y - \overline{Y}\|^2} = 1 - \frac{\|Y - \hat{Y}\|^2}{\|Y - \overline{Y}\|^2} = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$; Bigger R^2 = better fit

R squared

```
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
# WHERE IS THIS FORMULA FROM?
AdjR2 <- 1 - (RSS/(n-p))/(TSS/(n-1))
```

1.2 Model selection

Penalized RSS: $\|Y - X\beta\|_2^2 + \lambda\|\beta\|_0$

Forward Selection: Greedily add feature that brings best improvement (CV or penalized RSS). Iterate.

Backward Selection: Same but start from full feature set and eliminate features.

Stepwise methods

```
library(leaps)
# Try all the submodels
regfit.full=regsubsets(Salary~., data=..., nvmax=19)
# Forward stepwise (method="backward" for backward)
regfit.full=regsubsets(Salary~., data=..., nvmax=19,
↪  method="forward")
```

1.3 R Diagnostic plots

#1 **Tukey-Anscombe Plot** the points follow the line, else $E(\epsilon) = 0$ violated (if so, try data transforms, log if variance grows linearly, sqrt if variance grows as sqrt, or weighted regression) #2 **Q-Q Plot** should follow a straight line, else error not Gaussian (still all fine). #3 **Scale-Location**: Checks for heteroscedasticity (similar to Tukey, but here we have standardized residuals on the y axis). #4 **Cook distance**: shows if some data points have a larger impact on the fit than others (outliers) #5 **Residuals vs Leverage** Leverage = measure of points being outliers in terms of the feature distribution. Points with high leverage normally also have high residuals, but not always

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. This can be done for any feature too, not just for intercept

Categorical variables

```
# overwrite club as a factor, otherwise will be
↪  considered a continuous variable
# if done correctly, for club in [1,2,3], the summary
↪  will show club2, club3 (first is baseline)
clubs_dset$club = factor(clubs_dset$club)
reg2 = lm(clubs_dset$distance ~ ., data=clubs_dset)
```

Interaction Definition If there is no interaction, then the difference in the predicted values between samples with type=0 and type=1 should be the same if the other predictors coincide, no matter whether partners=1 or partners=8. Mathematically, this means that $\gamma 1,0 - \gamma 1,1 = \gamma 8,0 - \gamma 8,1$.

2 Confidence Intervals

We find some CI of level α, $\mathbb{P}(\in \text{CI}) = 1 - \alpha$

For Coefficients: $\beta_j \in [\hat{\beta}_j \pm \hat{se}(\hat{\beta}_j) \cdot t_{n-p, 1-\alpha/2}]$ (meaning the $1 - \alpha/2$ quantile of a t_{n-p} distribution)

For predictions: there are 2 types: confidence (where the true model should lie) and prediction (where individual predictions might end up), the latter is always wider.

Confidence Intervals

```
confint(fit, level=0.95) # CI for coeffs
# CI for new samples, see ?predict.lm
predict(reg, sample, interval="confidence", level)
predict(reg, sample, interval="prediction", level)
```

Bias Variance Trade-Off

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

Bias Variance Trade-Off of a Method

```
# IS EstimateUsingCV AN ARBITRARY FITTED MODEL?
Bias <- mean(EstimateUsingCV) - TrueValueSimulated
MSE <- Bias^2 + var(EstimateUsingCV)
```

3 Non-parametric Density Estimation

Estimate true density $f(x)$ with $\hat{f}(x)$.
Ie. minimize $MSE(x) = E[(\hat{f}(x) - f(x))^2]$

Kernel density est.: $\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)$.

Conditions for K: $K(z) \geq 0 \forall z$, $\int K(z)dz = 1$, $K(z) = K(-z) \forall z$. Common choices for K:

Naive/histogram-like approach.: $K(z) = \frac{1}{2}\mathbf{1}\{|z| \leq 1\}$

Gaussian: $K(z) = (2\pi)^{-\frac{1}{2}} \exp(-\frac{1}{2}z^2)$

Non Parametric Density Estimation

```
denshat <- density(data, bw=0.1, kernel="gaussian")
# construct approximate pdf from density estimate
# rule says how to pred outside [min(x),max(x)]
pdf <- approxfun(denshat$x, denshat$y, rule=2)
# taking the log likelihood of some data
log_likelihood = mean(log(pdf(eval_data)))
```

4 Non-parametric Regression

We want to estimate $m(x) = \mathbb{E}[Y \mid X = x]$

4.1 Nadaraya-Watson kernel method

$\hat{m}(x) = \frac{\sum_i K((x - x_i)/h) Y_i}{\sum_i K((x - x_i)/h)}$. h is the bandwidth. K is the kernel function

Local bandwidth To improve one idea is to have $h(x)$ depend on x and not be the same everywhere.

Local polynomial: Extension of Watson method that fits a local polynomial instead of a local constant.

Smoothing Splines:

$G = \{g : [a,b] \to \mathbb{R} : g'' \text{ exists and } \int_a^b g''(x)^2 dx < \infty\}$ 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 $\lambda = \infty$, then least squares estimate. Shrunken version of natural spline with knots at $x_1, ..., x_n$.

Closed form solution: $\hat{\beta} = (B^\top B + \lambda\Omega)^{-1} B^\top Y$, where $\Omega_{jk} := \int B_j''(z) B_k''(z) dz$, with spline basis functions B_j, $j \in \{1, ..., n\}$. Smoother matrix $S = B(B^\top B + \lambda\Omega)^{-1} B^\top Y$.

Code for non-parametric regression methods

```
# Nadaraya-Watson, x.points are predicted on!
# CAREFUL! this reorders things internally!
watson = ksmooth(x, y, bandwidth=0.2, x.points=x)
plot(watson$x, watson$y)
# Local polynomials, span or enp.target for smoothing
poly = loess(y ~ x)
plot(x, predict(poly, x))
# Splines -> df, spar, lambda for deg free
spline <- smooth.spline(x, y)
y_hat <- predict(spline, x = x)$y # or spline$y
# Global optimum bandwidth
global_opt = glkerns(x,y,x.out=x) # pred on x.out
plot(global_opt$x.out, global_opt$est)
# Local optimum bandwidth
local_opt = lokerns(x,y,x.out=x)
plot(local_opt$x.out, local_opt$est)
```

4.2 Degrees of freedom

Hat Matrix $\hat{Y} = \mathcal{S}Y$. Every linear method in Y has such a matrix

Degrees of freedom $df = \text{tr}(\mathcal{S})$.

Compute hat matrix and df

```
Id <- diag(nrow(data))
S <- matrix(0, nrow(data), nrow(data))
for (j in 1:nrow(data))
  S[, j] <- ksmooth(x, Id[,j], x.point=x)$y
df <- sum(diag(S)))
```

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). $\text{Var}(\hat{\theta_k}) = 1/K \cdot \hat{\text{Var}}(MSEs)$ **LOOCV:** extreme version where each data point is a fold (least bias, high variance since the datasets are correlated).

LOOCV effiecent computation: Assuming a linear model ($\hat{Y} = \mathcal{S}Y$) we have a fast way to find LOOCV.

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

A historical approximation of this is GCV:

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

Cross validation

```
indices = sample(n_elements, replace=F)
# or, if ordering is needed
indices = 1:n_elements

fold_indicator = cut(indices, breaks=n_folds,
↪  labels=F)

for(i in 1:nfolds){
  train <- data[i != fold_indicator,]
  test <- data[i == fold_indicator,]
  # ...
}
```

6 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 CI

Reversed Quantile CI: (intelligent) $[\hat{\theta}_n - q_{\hat{\theta}^* - \hat{\theta}}(1 - \alpha/2), \hat{\theta}_n - q_{\hat{\theta}^* - \hat{\theta}}(\alpha/2)]$ (type="basic")

Quantile Bootstrap CI: (naive) 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.

Normal Bootstrap CI: Assume $\hat{\theta}_n$ 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")

Double bootstrap CI: For a regular bootstrap CI, we have some error: $\mathbb{P}[\theta \in I(1 - \alpha)] = 1 - \alpha + \Delta_n$. The goal is to adjust the interval such that we actually get the coverage probability that we originally aimed for, ie: $\mathbb{P}[\theta \in I(1 - \alpha')] = 1 - \alpha$

Parametric Bootstrap: We assume $X_1 ... X_n \sim P_\theta$. We estimate $\hat{\theta}$ then we draw bootstraps $X_1^* ... X_n^* \sim P_{\hat{\theta}}$

Regression bootstrap: We do $Y_i^* = \hat{m}(x_i) + \epsilon_i$ **parametric:** $\epsilon_i \sim \mathcal{N}(0, \hat{\sigma}^2)$ **model-based:** $\epsilon_i \sim \hat{P}_r$, the empirical dist. of residuals

Bootstrap

```
# Nonparametric
statistic <- function(orig_data, ind)
↪  {mean(orig_data[ind,]}
non_para_boot = boot(data, statistic, R=100)
non_para_boot$t0 # final bootstrap estimate
non_para_boot$t # each individual estimate
boot.ci(non_para_boot, conf, type=c("basic", "norm",
↪  "perc"))

# Parametric
statistic <- function(data) {mean(data)}
fitted_distr = fitdistr(data, "gamma")
ran.gen <- function(data, mle){
  rgamma(length(data), shape = mle[1], rate = mle[2])
}
para_boot = boot(data, statistic, R, sim="parametric",
↪  ran.gen=ran.gen, mle=fitted_distr$estimate)

# Intervals by hand
ci_quantile =
↪  quantile(boot_estimates,probs=c(0.025,0.975))
center = 2*theta_hat - mean(boot_estimates)
spread = qnorm(1 - alpha/2) * sd(boot_estimates)
ci_normal = c(center - spread, center + spread)
ci_reversed <- theta_hat-quantile(boot_estimates-
↪  theta_hat,probs=c(0.975,0.025))
```

7 Classification

For any class we estimate the probability of being in such class: $\pi_j(x) = \mathbb{P}(Y = j \mid X = x)$

The **Bayes estimator** is $\mathcal{C}(x) = \arg\max_j \pi_j(x)$.

Linear & quadratic discriminant analysis: Assumption: $X \mid Y \sim \mathcal{N}(\mu_j, \Sigma)$. To approximate

$$\hat{\mu}_j = \sum_{i=1}^{n} X_i \mathbf{1}_{[Y_i=j]} / \sum_{i=1}^{n} \mathbf{1}_{[Y_i=j]} = \frac{1}{n_j} \sum_{i;Y_i=j} X_i,$$

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

Quadratic discriminant analysis: Compute Σ group-wise by only looking at elements in the group and by correcting with $\frac{1}{n_j - 1}$ instead of $\frac{1}{n-j}$

Logistic regression: Let $\pi(x) = \mathbb{P}(Y = 1 \mid X = x)$. Estimate with a linear model $g(x) = \text{logit}(\pi(x)) = \log \frac{\pi(x)}{1-\pi(x)}$. Assuming $g(x) = X\beta$, for the log-likelyhood: $\ell(\beta; Y, X) = \sum_{i=1}^{n} \left(Y_i \beta^T x_i - \log\left(e^{\beta^T x_i} + 1\right)\right)$

Comment: LDA is a logistic regression model, since log-odds ratio is linear. But not every logistic regression model is LDA due to normal feature assumption in LDA.

Techniques for multi-class:

1) One versus rest: Train a model per class against all other classes to receive $\pi_j(x)$. Normalization needed.

2) Everyone versus reference: Train all other $J-1$ classes against reference class 0: $g(x) = \log(\pi_j(x)/\pi_0(x))$

3) Multinomial Distribution: Use multinomial distribution instead of binomial

4) One vs. one Train a model to distinguish each class from each other. In total $\binom{J}{2}$ models.

5) Ordered classes: $\text{logit}(\mathbb{P}(Y \leq k \mid x)) = \alpha_k + g(x)$ with $\alpha_0 \leq \alpha_1 \leq \cdots \leq \alpha_{J-1}$

Classification in R

```
# train and test contain numeric features, while
↪ _labels are a factor
knn_preds = knn(train = train, test = test,
                cl = train_labels, k=5, prob=F)
acc = mean(knn_preds == test_labels)
# LDA/QDA
c_lda<-lda(x=data[,c("x1","x2")],grouping=data[,"y"])
c_qda<-qda(x=data[,c("x1","x2")],grouping=data[,"y"])
acc=mean(labels == predict(c_lda, newdata)$class)
# Logistic Regression
# family can be other distributions, gaussian=binary
↪ log reg
mod <- glm(y ~ ., data = data, family = gaussian)
predict(mod, test_data, type = "response") # return
↪ the probability of being 1
# for multinomial classification
class_multinom <- multinom(y ~ . , data = data)
```

ROC (Receiver Operating Characteristic):
Predict $\hat{Y}(x) = \mathbb{I}(\hat{\pi}(x) \geq \theta)$. Basics

$TP + FN = P$; $TN + FP = N$

sensitivity $= TPR = \frac{TP}{P}$; estimates $\mathbb{P}(\hat{Y} = 1 | Y = 1)$

specitivity $= TNR = \frac{TN}{N}$; estimates $\mathbb{P}(\hat{Y} = 0 | Y = 0)$

Changing all values of $\theta \in [0, 1]$, look at specificity vs. sensitivity. Integrating the resulting function yields the *area under the curve*. ROC AUC $= \mathbb{P}(\hat{\pi}_{I_0} < \hat{\pi}_{I_1})$, where I_0 and I_1 are sampled uniformly at random from indexes with classes $Y = 0$ and $Y = 1$, respectively.

default cost = missclass. rate $= \frac{FP+FN}{n}$

Evaluation

```
#1 is a vector of 0/1 (the truth)
#p is a vector of predicted probs to be 1
pred <- prediction(predictions = p, labels = 1)
perf <- performance(pred, "tpr", "tnr")
plot(perf)# plots the ROC curve
cost <- performance(pred, "cost", cost.fp = 1,
↪ cost.fn = 2)
plot(cost)
```

8 Flexible regression

Making some structural assumption, we estimate

$$g(x) = \begin{cases} \mathbb{E}[Y \mid x] & \text{in regression} \\ \text{logit}(\pi(x)) & \text{in classification} \end{cases}$$

8.1 Additive model

In additive model we assume that

$$g(x) = \mu + \sum_j g_j(x_j) \qquad \mathbb{E}[g_j(x_j)] = 0 \quad \forall j$$

To find $\hat{g}_j$, use **backfitting**: Let $s_j : (u_1, ..., u_n)^T \to (\hat{u}_1, ..., \hat{u}_n)^T$ be a smoother (any prev model that we used, eg. lin reg, kernel estimation, splines)

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 $j = 1, ...p$:

$\hat{g}_j \leftarrow s_j(y - \hat{\mu}\mathbb{1} - \sum_{k \neq j} \hat{g}_k)$

Normalize the functions $\hat{g}_j \leftarrow \hat{g}_j - \frac{1}{n} \sum_{i=1}^{n} \hat{g}_j(x_{ij})$

8.2 MARS (Multivariate Adaptive Regression Splines)

We make a regression with functions in $\mathcal{M}$. We start with $\mathcal{M} = \{h_0(\cdot) = 1\}$. We then iteratively add functions in $\mathcal{M}$ in the form:

$$h_{\text{new},1-2}(x) = h_l(x)(\pm x_j \mp d)_+$$

Where h_l is in $\mathcal{M}$. We then prune back useless stuff.

Flexible regression in R

```
# Additive model
gamForm <- wrapFormula(logupo3 ~ ., data=d.ozone.e)
gam_fit <- gam(gamForm, data = mydata)
predicted_y <- predict(gam_fit, testdata)
# MARS model
mars_fit <- earth(y ~ ., data = mydata, degree = 3) #
↪ no. of interactions
predict(mars_fit, newdata = test_data, type =
↪ "response")
```

8.3 CART (Classification and regression trees)

The tree devides the domain in rectangles, in the end you have a piece-wise constant function.

$g_{\text{tree}}(x) = \sum_{r=1}^{M} \beta_r \mathbb{I}_{\mathcal{R}_m}(x)$

We can penalize larger tree to reduce variance. We choose the simplest model which is within standard error to the best one in cross-validation. To fit we are greedy. **Pros** and **cons**:

- Highly flexible and interpretable
- Too simplistic, only piece-wise constant

CART in R

```
library(rpart)
tree = rpart(y ~ ., data = data, control =
↪ rpart.control(cp = 0.0, minsplit = 30))
↪ #minsplit: min no. of datapoints in a node to
↪ attempt a split
#cp = complexity factor, the R^2 must increase at
↪ least of a factor cp after split
#for no regular. at all: cp=0, minsplit=minbucket=1
plotcp(tree) or printcp(tree) #estimate the best cp
prune.rpart(tree, cp=cp.opt) #to prune the tree
```

9 Bagging-Boosting

9.1 Bootstrap Aggregating (Bagging)

Bagging is a variance reduction technique. We need a simple estimator $\hat{g}$ (for instance a tree).

1) We take B bootstrap samples

2) For each bootstrap sample we train an estimator $\hat{g}_1, \ldots, \hat{g}_B$.

3) Aggregation: $\hat{g}_{bag}(\cdot) = \frac{1}{B} \sum_{i=1}^{B} \hat{g}_i(\cdot)$

classification: in this case we have majority voting (or we can average base estimator probs of being in each class)

Subsample aggregating (Subagging): Like Bagging but we just take subset of the dataset (without repetition) instead of bootstrapping. $m < n$ size of the subsets

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 $\to$ 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).

Random forest

```
library(randomForest)
rf <- randomForest(y ~ . , data = mydata, mtry=p-1,
↪ importance = TRUE, ntree = 100)
# mtry: no. of features from which we chose a split.
# if mtry = n_predictors then we have bagging trees
oob <- mean((rf$predicted-y.train)^2) #OOB Reg.
oob <- rf$err.rate #Classification
importance(rf) #Get feature importance
```

9.2 Feature Importance Measures for RF

TODO

9.3 Boosting

Boosting is a bias reduction technique. We have $\hat{g}$ to be very simple estimator (stamp or small tree). We iteratively train a $\hat{g}$ predictor on the current model and then update the model: $f \leftarrow f + v g$. (v usually is .1) The other parameter is M.

10 Ridge/Lasso:

We are in the situation where $p >> n$.We center and scale all the x_i so that we can penalize all β_i equally. In Ridge we penalize with $\|\beta\|_2^2$, in Lasso $\|\beta\|_1$.

Elastic net Mix of Ridge and Lasso, the regularization is $\lambda_2 \|\beta\|_2^2 + \lambda_1 \|\beta\|_1$. We call $\alpha = \frac{\lambda_2}{\lambda_1 + \lambda_2}$.

Ridge & Lasso

```
library(glmnet) # data must be a MATRIX!
grid <- 10^seq(from=10,to=-2,length=100)
ridge <- glmnet(x[train,], y[train], alpha=0,
↪ lambda=grid)
lasso <- glmnet(x[train,], y[train], alpha=1,
↪ lambda=grid)
# Coefficients for specific lambda_val
coef(lasso, s=lambda_val)
cv.glmnet(mm, y, alpha=0.5, nfolds=10)#For inbuilt CV

\textbf{Adaptive lasso:} First you get an estimate of
↪ $\hat \beta$. Then we penalize more $\beta_i$ we
↪ think they are $0$, we set
↪ $w_j=\vert\beta\vert^{-\gamma}$. The
↪ regularization term will be $\lambda\sum_j
↪ w_j\vert\beta_i\vert$. \\
\text{Relaxed Lasso:} Like lasso but we use
↪ $\phi\lambda$ instead of $\lambda$ ($\phi \in
↪ [0,1]$). We then take out variables we
↪ don't need.\\
\textbf{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).
```

R-Help

```
#argmax_per row
probs <- predict(tree, newdata = data)
pred<-colnames(probs)[max.col(probs)]
# 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")
thuesen <- thuesen[!is.na(thuesen[,2]),] # Remove NA
data_comp <- data[complete.cases(data),]
scaled.dat <- scale(dat) # Standardize data
apply(X, d, func) #apply func to all rows(d = 1) or
↪ cols(d = 2) of matrix X
sapply(X, func) #apply func to elems in the list X
rep(x=0, times=100) # Create empty vector
dataset[,"logX"] <- log(dataset[,"X"])# Log transform
# Order dataframe accordind to column x1

data.matrix(df) # Dataframe to Matrix
myList <- numeric(n) # Initialize empty list size n
i <- a %% c #modulo operator
# Predict help to get model specific params
?predict.loess # Replace 'loess' with wanted function
ls("package:lokern") # list of methods in package

# Linear regression
coefficients(fit)
# access estimate/se/t-value/p-value of a coeff
# is ore precise than what is printed!!!
summary(fit)$coefficients["row_name", "col_name"]

# MATRIX OPS ---------------------------------
%*% # matrix multiplication
solve(X) # matrix inversion

# DATAFRAME ----------------------------------
# indexing always is df[row(s), col(s)]
dfOrdered <- df[order(df$x1, decreasing = F),]
col_indices = which(names(df) %in% c("col1","col2"))
df <- df[,-col_indices] # remove columns named above

# DATA PROCESSING  ---------------------------
complete.cases # remove nans
which(array==target_value) # get idx of target val

# DISTRIBUTION OPS ---------------------------
runif(10, min=0, max=1), rnorm(10, mean=0, sd=1)
# constructing approximate pdf from density estimates
pdf = approxfun(denshat$x, denshat$y, rule=2)
log(pdf(eval_data)) # get log likelihood of some data
```