Code

General Stuff

```
# limit number of digits:
options(digits=2)
# apply a function to vector
# compute corr of columns
cor.vec <- apply(x,2,cor,y=y)
# apply a function n times
replicate(n, cv.knn1(x.new,y,K=10))
# find the idx in order
order(c(3,6,2,8,4,9,1),decreasing=T)
#return bool vector not na rows
complete.cases(df)
#removes na rows from df
na.omit(df)
which.min # same as argmin
which # returns TRUE indices
rm(list=ls()) # clear everything
"X1" %in% c("X2","X3") #TRUE if exists in it
# for making grids
seq(from,to,length,by)
# reshaping into 10by10 matrix
matrix(1:100, nrow=10)

# string manipulation
# paste is like paste0 but sep=" "
paste0(c("x","+","y"),collapse="") # "x+y"
paste0("x","+","y") # "x+y"
paste("x","+","y") # "x + y"
paste("x","y", sep=" + ") # "x + y"
```

Linear Model

```
fit <- lm(y ~ x, data)
# 0 + x for no intercept
# functions
summary abline coef
residuals fitted confint
# get std of first feature
se1<-coef["x1","Std. Error"]
plot(fit) # diagnostic plots
# conducting a partial F-test:
anova(fit.partialmodel, fit.all)
# prediction
df <- data.frame(colname = x0)
predict(fit, df, level=0.95,
interval=c("p","c")
# manual calculation
quant <- qt(.975,n-p)
sigma.hat <- sqrt(sum((fit$resid)^2)/(n-p))
X <- as.matrix(cbind(1,x))
XtXi <- solve(t(X) %*% X)
x_00 <- as.matrix(c(1,x_0),nrow=2)
# for confidence interval use
se<-sigma.hat*sqrt(t(x_00)%*%XtXi%*%x_00)
# for prediction interval use
se <-sigma.hat*sqrt(1+t(x_00)%*%XtXi%*%x_00)
lower <- expected_y0 - quant*se
upper <- expected_y0 + quant*se
```

Matrix

```
t # transpose
%*% # matrix product
crossproduct
solve # inverse
as.matrix #to convert df to matrix
```

Plotting

```
par(mfrow=c(2,2)) # 2x2 grid
dev.off() # clear
pairs # pairs plot
plot(stuff, pch, xlim=c(-1,1),
xlab="X label",main="Main title")
legend("topleft", c(names), pch, col)
# pch/col is symbol/color for each entry
hist(breaks,freq=F,add=T)
abline(a, b, h, v , lwd, lty, col)
# plot distribution
lines(density(vector))
# Plot the columns of one matrix
# against the columns of another.
matlines(x, ys)
# for small data
stripchart(data,method="stack")
stem(data)
```

KNN

```
# Package for knn classification:
library(class)
# knn classifier
y.pred <- knn(x.train, x.test, y.train, k=1)
# Package for knn regression:
library(kknn)
fit <- kknn(Y ~ ., train_df, test_df, k=8)
# predictions on test data
pred <- predict(fit)
```

Cross Validation

```
#Automatic CV for linear models
library(boot)
lm.fit = glm(y ~ bs(x, df = i), data = df)
RSS[i] = cv.glm(df, lm.fit, K = 10)$delta[2]
# randomly shuffle the rows:
ind.x <- sample(1:n, replace=FALSE)
x <- x[ind.x,]
# create K folds:
folds <- cut(seq(1,n),breaks=K,
labels=FALSE)
ind.test <- which(folds==i)
```

Bootstrap

```
library(boot)
# non parametric bootstrap
mle.fn <- function(data, index, extra_par){
  return(estimator)
  #for example(regression)
  return(coef(lm(y~x,data=data,subset=index)))
}
boot.result <- boot(data, mle.fn, R,
                    extra_par=xx)$t
# empirical CDF
boot.scaled[,i] <-
sqrt(n)*(boot.result[,i] - mles[i])
ecdf(boot.scaled[,i])
# parametric bootstrap
# function to compute estimator
mle.fn <- function(data) {
  return(estimator) #like median
}
# function to generate data
data.gen <- function(data, mle) {
  out <- data
  out$hours <- rexp(nrow(out), mle)
  return(out)
}
# perform boot
air.boot <- boot(data, mle.fn, R=1000,
            sim = "parametric",
            ran.gen = data.gen,
            mle = 1/mean(aircondit$hours))
boot.ci(boot.out,
type = c("norm", "basic", "perc"),
index=i)
# note normal CI uses bias correction.
# its midpoint is \hat \theta_n - \hat bias
#"norm" = normal, "basic" = reversed quantile
#"perc" = quantile, "stud" = bootstrap T

# Fixed X approach: bootstrap residuals
# Resampling here. Can be model-based.
boot.resid <- function(data, indices) {
  resid.new <- fit$residuals[indices]
  y.new <- fit$fitted + resid.new
  data.new <- data.frame(x=x, y=y.new)
  fit.new <- lm(y ~ x, data=data.new)
  return(coef(fit.new))
}
fit <- lm(y ~ x, data=data)
boot(data=data, statistic=boot.resid, R=1000)
```

Power Assesment

```
# compute p-value
gen.samples.adj <- c(gen.samples, value)
p.val <- sum(gen.samples.adj <=value) /
    length(gen.samples.adj)
# compute rejection region
rej <- quantile(gen.samples, 0.05)-1
# compute power
power <- sum(gen.samples.alt <= rej) / nsimul
```

Permutations

```
combn(n,k) # choose k out of n
# e.g y1=control, y2=treatment
wilcox.test(y1, y2, alternative, paired)
# implicit paired signed rank test
wilcox.test(diff, alternative)
# signed rank test
V <- sum(rank(abs(data))[data>0])
```

Model Selection

```
library(leaps)
fit <- regsubsets(y~., df, nbest, nvmax,
force.in, force.out, intercept, method)
coef(fit, n) # best model with n predictors
fit.sm <- summary(fit)
# parameters used in each step:
fit.sm$which
# best model based on bic:
best <- which.min(fit.sm$bic)
params <- coef(fit, best)
# add intercept and handle categorical
test.mat <- model.matrix(y~., data=df[test,])
pred <- test.df[,names(params)] %*% params
plot(fit.sm, scale) # r2, adjr2, Cp, bic

# backward regression
# trace=1 for verbose
# k=log(n) gives BIC penalty
model <- step(lm(y ~ ., data=df), k = log(n),
trace = 0)
```

Ridge and Lasso

```
library(glmnet)
grid <- 10^seq(from = 10, to = -2, length=100)
model <- glmnet(x, y, alpha=0, lambda=grid)
# alpha=0 -> ridge, alpha=1 -> lasso
plot(model, xvar="lambda")
# get coefficient estimates for new lambda=35
predict(model, s=35, type="coefficients")
# predict on test data
predict(model, s=35, newx=x.test)
#automatic cross validation
cv.out <- cv.glmnet(x, y, alpha=0)
plot(cv.out)
cv.out$lambda.min # best lambda
# +1 standard error larger lambda
cv.out$lambda.1se
# set specific folds(inner cv)
inner.folds <- folds[folds != i]
inner.folds <- factor(inner.folds)
levels(inner.folds) <- 1:(k-1)
cv.glmnet(...,
foldid = as.numeric(inner.folds),...)
```

NonLinearity

```
# poly returns uncorrelated columns
# raw=T -> without orthogonalization
fit <- lm(y~poly(x,4),data=df)
predict(fit,newdata=data.frame(x=grid))

library(splines)
# bs is basis function
fit <- lm(y~bs(x,knots=c(k1,k2,k3)),data=df)
# For K knots, specify df=K+3
fit <- lm(y~bs(x,df=K+3),data=df)
# locations of knots
attr(bs(age,df=6),"knots")
# bs() default degree is 3 (cubic splines)
# use ns() to fit natural splines.
# for K knots, specify df=K+1
fit <- lm(y~ns(x,df=4),data=df)

# Smooth splines
fit <- smooth.spline(x,y,df=16)
fit.cv <- smooth.spline(x,y,cv=TRUE)
fit.cv$df
fit.cv$lambda

# Local Regression
# span is how much nearby data to consider
fit <- loess(y~x,span=.2,data=df)

# GAMs
library(gam)
```

```r
gam <- lm(y~ns(x1,4)+ns(x2,5)+x3,data=df)
plot.Gam(gam, se=TRUE, col="red")

# use backfitting
# s() for smoothing splines, lo() for loess
gam <- gam(y~s(x1,4)+s(x2,5)+x3,data=df)

# formula generation
# use all columns
list.var <- names(df)
f <- sapply(list.var,
function(x){paste0("s(",x,",4)")})
f <- paste0(f, collapse = "+")
f <- paste0("y ~ ", f)
f <- as.formula(f)
gam <- gam(f, data=df)
```

Trees

```r
library(tree)
#.-y means everything but y
fit <- tree(y~.-y, data=df, subset=train_idx)
plot(fit)
text(fit,pretty=0)
#prediction
tree.pred = predict(tree.fit, testdf,
type="class")
table(tree.pred, ytest)
cv.fit <- cv.tree(fit, FUN=prune.tree)
plot(cv.fit$size, cv.fit$dev,type="b")
# for classification:
# replace prune.tree to prune.misclass
# k is alpha
# dev means number of misclassifications

# best is number of terminals
pruned <- prune.tree(fit, best=2)

# Random Forests
library(randomForest)
fit <- randomForest(y~.,data=df, subset, mtry,
            norm.votes,importance=T, maxnodes)
# fit£predicted gives predicted values based on
#out of bag samples

# mtry number of vars checked on each split
# default is p/3 for regression
# default is  sqrt(p) for classification
# given MSE is out-of-bag MSE

# importance
varImpPlot(fit)
randomForest::importance(fit)

# fraction of trees voted for each class:
fit$vote
# aggregate class probabilities
# and not predictions:
library(ranger)
rf <- ranger( as.factor(y) ~. , data=data,
    mtry, probability=TRUE, min.node.size)
rf$predictions
```

Formulas

Linear Regression

$$y = X\beta + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2\mathbb{I})$$
$$\hat\beta = (X^TX)^{-1}X^Ty$$
$$P = X(X^TX)^{-1}X^T$$
$$\hat\beta \sim \mathcal{N}(\beta, \sigma^2(X^TX)^{-1})$$
$$\hat y \sim \mathcal{N}(X\beta, \sigma^2 P)$$
$$e = y - \hat y \sim \mathcal{N}(0, \sigma^2(\mathbb{I} - P))$$
$$\hat\sigma^2 = \frac{1}{n-p}\sum_{i=1}^{n}(y_i - x_i^T\hat\beta)^2$$
$$\hat\sigma^2 \sim \frac{\sigma^2}{n-p}\mathcal{X}^2_{n-p}$$

Adjusted R score

$$TSS = \sum_{i=1}^{n}(y_i - y_{mean})^2,\ RSS = \sum_{i=1}^{n}(y_i - \hat y_i)^2$$
$$\frac{1}{n-1}TSS = \text{sample variance of } y_1, ..., y_n$$
$$\frac{1}{n-1}RSS = \text{sample variance of } e_1, ..., e_n$$
$$R^2 = 1 - \frac{RSS}{TSS}$$
$$R^2 = 0 \text{ means model do nothing}$$
$$R^2 = 1 \text{ model completely fits}$$
$$R^2_{adj} = 1 - \frac{RSS/(n-p)}{TSS/(n-1)}$$
$$F = \frac{(RSS_1 - RSS_2)/(p_2 - p_1)}{RSS_2/(n-p_2)}$$

Std Errors and P values of beta

$$Se(\hat\beta_k) = \sqrt{\hat\sigma^2[(X^TX)^{-1}]_{kk}}$$
$$\frac{\hat\beta_j - 0}{\sqrt{\hat\sigma^2[(X^TX)^{-1}]_{kk}}} \sim t_{n-p} \text{ under null}$$

Type 1 and 2 errors

		True State of Nature	
		H_0 true	H_0 false
Decision	Accept H_0	Correct Probability $= (1-\alpha)$	Wrong (type II error) Probability $= \beta$
	Reject H_0	Wrong (type I error) Level of significance $= \alpha$	Correct Power $= 1 - \beta$

Confidence Intervals

$(1-\alpha)100\%$ confidence interval for $\mathbb{E}[y_0] = x_0^T\beta$:

$$x_0^T\hat\beta \pm \hat\sigma\sqrt{x_0^T(X^TX)^{-1}x_0}\ t_{1-\frac{\alpha}{2}, n-p}$$

$(1-\alpha)100\%$ confidence interval for y_0 :

$$x_0^T\hat\beta \pm \hat\sigma\sqrt{1 + x_0^T(X^TX)^{-1}x_0}\ t_{1-\frac{\alpha}{2}, n-p}$$

Bias Variance Decomposition

$$\mathbb{E}[(y_0 - \hat f(x_0))^2] = \overbrace{(f(x_0) - \mathbb{E}[\hat f(x_0)])^2}^{\text{Bias}^2} + \overbrace{\mathbb{E}[(\hat f(x_0) - \mathbb{E}[\hat f(x_0)])^2]}^{\text{Variance}} + \overbrace{\sigma^2}^{\text{Noise}}$$

Bootstrap

Consistency with convergence rate α_n for $\hat\theta_n$:

$$\overbrace{\mathbb{P}[\alpha_n(\hat\theta_n - \theta)]}^{\text{not random- want to know}} - \overbrace{\mathbb{P}^*[\alpha_n(\hat\theta_n^* - \hat\theta_n)]}^{\text{random- approx via bootstrap}} \xrightarrow{p} 0 \text{ as } n \to \infty$$

Consistency holds if $\alpha_n(\hat\theta_n - \theta) \sim \mathcal{N}(\theta, \sigma^2)$
Bias and Variance estimation for $\hat\theta_n$:

$$bias(\hat\theta_n) \approx \mathbb{E}^*[\hat\theta_n^*] - \hat\theta_n \approx \hat{\bar\theta}_n^* - \hat\theta_n$$
$$var(\hat\theta_n) \approx var(\hat\theta_n^*) \approx \frac{1}{B-1}\sum_{b=1}^{B}(\hat\theta_n^{*b} - \hat{\bar\theta}_n^*)^2$$

Bootstrap CIs:

- quantile(**percentile**): $[q_{\hat\theta_n^*}(\frac{\alpha}{2}), q_{\hat\theta_n^*}(1 - \frac{\alpha}{2})]$
- normal(**normal**): $\hat\theta_n \pm q_Z(1 - \frac{\alpha}{2})\hat{sd}(\hat\theta_n)$ where $Z \sim \mathcal{N}(0,1)$, $\hat{sd}(\hat\theta_n) = \sqrt{\hat{var}^*(\hat\theta_n^*)}$ **R uses $\hat\theta_n - \hat{bias}$ as midpoint**
- reversed quantile(**basic**): $[\hat\theta_n - q_{\hat\theta_n^* - \hat\theta_n}(1 - \frac{\alpha}{2}), \hat\theta_n - q_{\hat\theta_n^* - \hat\theta_n}(\frac{\alpha}{2})]$
- Bootstrap T(**stud**): $[\hat\theta_n - q_{\frac{\hat\theta_n^* - \hat\theta_n}{\hat{sd}(\hat\theta_n^*)}}(1 - \frac{\alpha}{2})\hat{sd}(\hat\theta_n), \hat\theta_n - q_{\frac{\hat\theta_n^* - \hat\theta_n}{\hat{sd}(\hat\theta_n^*)}}(\frac{\alpha}{2})\hat{sd}(\hat\theta_n)]$

Wilcoxon signed rank test

permute signes and use this statistic:
$V = \sum_i rank(|D_i|).\mathbb{1}_{D_i > 0}$

Multiple Testing

		True State of Nature		Total
		H_0 true	H_0 false	
Decision	H_0 is not rejected	U	T	m - R
	H_0 is rejected	V	S	R
	Total	m_0	$m - m_0$	m

- $FDP : Q = \frac{V}{R},\ FDR := \mathbb{E}[Q]$
- $FWER := \mathbb{P}(V \geq 1)$ global null : $m = m_0$
- $FWER = FDR$ under the global null
- $FWER \geq FDR$ in general
- if $m_0 \geq 1$ and the tests are exactly at level $\alpha \Rightarrow FWER \geq \alpha$
- $FWER \leq \alpha m$
- **FWER Control - Bonferroni**: conduct m tests at level $\frac{\alpha}{m}$
- under global null, with all tests at level α, $FWER = 1 - (1-\alpha)^m$ and with first order tylor: $FWER \approx m\alpha$
- in case of dependant tests(We want distribution of min(pvalues) under the **global null**): conduct **FWER Control-Westfall Young** and find $\sigma := \alpha$ quantile of $minp_j$ with permutation test under the global null and conduct all tests at level σ. For a dataset with n points and m variables: shuffle y column, compute p-value for each m variables (e.g Wilcoxon) store $min(p_1, ..., p_m)$ repeat.
- **FDR Control - Benjamini Hochberg**: want to have FDR rate q? order p-values and find the largest i such that $p_{(i)} \leq \frac{i}{m}q$ reject null hyp. for all tests j, $j \leq i$ (**Only works for independant p-values**)

Beyond Linearity

- **Orthogonal Basis:** when adding new orthogonal basis to a function: estimates for other coef will be the same, p-values will change. Anova will use the p-values of the largest model.
- Number of free parameters in **cubic spline** with k knots? $k+1$ intervals with 4 parameters $\Rightarrow 4(k+1)$ parameters. k knots with 3 constraints per knot: $3(k)$ to be reduced. So in total $4(k+1) - 3k = k+4$ parameters. (In R reduce one to omit intercept $k+3$)
- Number of free parameters in **natural cubic spline** with k knots(same as cubic spline but linear outside of knots)? $k + 4 - 2*2 = k$ (In R use $k+1$)
- Good Basis function for cubic spline? $\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \gamma_1(x - c_1)_+^3 + \gamma_2(x - c_2)_+^3 + ... + \gamma_k(x - c_k)_+^3$
- **Smoothing Spline** $\hat g = \underset{g \in G}{argmin}\{\sum(y_i - g(x_i))^2 + \lambda\int_a^b g''(x)^2dx\}$ if $\lambda = 0$ interpolate all points(**n degress of freedom**), if $\lambda = \infty$ LS solution(**2 degrees of freedom**). In general equivalent to **shrunken version of natural cubic spline**.
- **Backfitting Algorithm**

  repeat
      for each predictor j **do**
          $\hat g_j \leftarrow S_j(y - \hat\mu - \sum_{k \neq \hat g_k})$
          $\hat g_j \leftarrow \hat g_j - mean(\hat g_j)$ where
  $mean(\hat g_j) = \frac{1}{n}\sum_i \hat g_j(X_{ij})$
      end for
  until Convergence