Data Mining

Sunghoon Kwon
[1]*Department of Applied Statistics, Konkuk University*

1 Simple linear regression

1.1 Simple Regression

- The simple regression is a method of finding a relationship f between two variables y and x under the presence of a random error ε.

$$y = f(x) + \varepsilon \tag{1}$$

  - $y \in \mathbb{R}$: output/dependent/explanatory variable (known)
  - $x \in \mathbb{R}$: input/independent/response variable or covariate (known)
  - $f : \mathbb{R} \to \mathbb{R}$: model/target function of interest (unknown)
  - $\varepsilon \in \mathbb{R}$: random error that cannot be controlled - usually assumed to have a statistical (normal) distribution (unknown)

* Real life examples
  - Does your weight increase as your height does? If so, how?
  - Can you say your sons or daughters will be tall because you are?

- The simple regression consists of two processes: modeling (deciding a form of f) and estimating (specifying f) the model.

1.2 Simple linear regression

- The simple linear regression assumes that the function f is linear which implies the modeling f is done.

$$y = f(x) + \varepsilon = \beta_0 + \beta_1 x + \varepsilon \tag{2}$$

  - The model f is often called population regression line.
  - Two uknown constants, $\beta_0 \in \mathbb{R}$ (intercept) and $\beta_1 \in \mathbb{R}$ (slope) are the parameters/coefficients that specify f.
  - The model f can be understood as the conditional expectation of y given x by assuming $E(\varepsilon|x) = 0$.

$$E(y|x) = \beta_0 + \beta_1 x = f(x)$$

  - The final work to do is estimating the parameters β_0 and β_1.

1.2.1 Observational expression

- Given n samples, $(x_1, y_1)^T, \cdots, (x_n, y_n)^T$, we can write an observational expression.

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i \leq n \tag{3}$$

  - We can write the equations by using matrix-vector notations.

$$y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i, \quad i \leq n \tag{4}$$

  - $\mathbf{x}_i = (1, x_{i1})^T \in \mathbb{R}^2$
  - $\boldsymbol{\beta} = (\beta_0, \beta_1)^T \in \mathbb{R}^2$
- We can use more compact form.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} = \beta_0 + \mathbf{X}_1\beta_1 + \boldsymbol{\varepsilon} \tag{5}$$

  - $\mathbf{y} = (y_1, \ldots, y_n)^T \in \mathbb{R}^n$
  - $\mathbf{X} = (\mathbf{x}_1^T, \cdots, \mathbf{x}_n^T)^T = (\mathbf{1}, \mathbf{X}_1) \in \mathbb{R}^{n\times 2}$
  - $\mathbf{1} = (1, \ldots, 1)^T \in \mathbb{R}^n$
  - $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)^T \in \mathbb{R}^n$

- R example

```
##### car accident samples

### reading samples
rm(list=ls());
car.df <- read.csv(file="D://car.sample.csv",sep=",")
str(car.df); car.df[1:5,]

### vectors and matrix
y.vec <- car.df$spd[1:100]   # output vector
x.vec <- car.df$x1[1:100]    # input vector
x.mat <- cbind(1,x.vec)      # design matrix
y.vec[1:10]; x.mat[1:10,]

### some algebra
b.vec <- c(-5,5)                    # coefficients
xb.vec <- as.vector(x.mat%*%b.vec) # conditional expectations
e.vec <- y.vec-xb.vec               # errors
cbind(y.vec,xb.vec,e.vec)[1:10,]

### some plots
plot(x.vec,xb.vec,type="n")
points(x.vec, y.vec,pch=1,col=1)  # original samples
points(x.vec,xb.vec,pch=3,col=3)  # regression lines
```

* (Homework) Search the origin of the word "regression" by googling and then find the actual meaning of the word at that time.

1.3 Interpretation of the model

- From the model, we directly can check the followings:
  - model $f(x)$: conditional expectation of y given x

$$f(x) = \beta_0 + \beta_1 x = E(y|x)$$

  - intercept β_0: conditional mean of y given $x = 0$

$$\beta_0 = E(y|x = 0)$$

  - slope β_1: difference between two conditional expectations

$$\beta_1 = E(y|x = k + 1) - E(y|x = k),\ \forall k \in \mathbb{R}$$

    * In this sense, we often say β_1 is the effect size of x on y.

1.3.1 Centering/scaling/standardization

- Centering x often helps the interpretation, here centering implies substituting

$$z = x - \mu_x$$

for x, where $\mu_x = E(x)$ so that the model becomes

$$y = f(x) + \varepsilon = \beta_0 + \beta_1 x + \varepsilon \Leftrightarrow y = g(z) = \alpha_0 + \alpha_1 z + \varepsilon.$$

  - model $g(z)$: conditional expectation of y given z

$$g(z) = \alpha_0 + \alpha_1 z = E(y|z)$$

  - intercept α_0: conditional expectation of y given $x = \mu_x$

$$\alpha_0 = E(y|z = 0) = E(y|x = \mu_x) = \beta_0 + \beta_1 \mu_x$$

  - slope α_1: difference between two conditional expectations

$$\begin{aligned} \alpha_1 &= E(y|z = k+1) - E(y|z = k) \\ &= E(y|x = \mu_x + k + 1) - E(y|x = \mu_x + k) = \beta_1 \end{aligned}$$

- Scaling x implies substituting

$$z = x/\sigma_x$$

for x, where $\sigma_x^2 = Var(x)$.

* (Homework) What happens in interpreting the coefficients?

- Centering and scaling x (standardization of x) implies substituting

$$z = (x - \mu_x)/\sigma_x$$

* (Homework) What happens in interpreting the coefficients?

- R example

```
### centering
cx.vec <- x.vec-mean(x.vec); mean(cx.vec); sd(cx.vec)

### scaling
sx.vec <- x.vec/sd(x.vec);   mean(sx.vec); sd(sx.vec)

### standardization
csx.vec <- (x.vec-mean(x.vec))/sd(x.vec);  mean(csx.vec); sd(csx.vec)

### results
cbind(x.vec,sx.vec,csx.vec)[1:5, ]
```

1.4 Loss and Risk functions

- Loss function - from 'google'
  - In mathematical optimization, statistics, decision theory and machine learning, a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An objective function is either a loss function or its negative (sometimes called a reward function or a utility function), in which case it is to be maximized.
  - The loss function should be defined as a random function of parameters.
  - In this class, we denote the loss function by L, and for example, $L(\beta_0, \beta_1)$ for the simple linear regression model.
- Risk function
  - Risk function is the expected value of a loss function denoted by $R = E(L)$

* This page will be revised further.

1.4.1 Various loss functions

- Popular choices for $L(\beta_0, \beta_1)$ for the simple linear regression model
  - square loss (least squares estimation)

$$L(\beta_0, \beta_1) = (y - \beta_0 - \beta_1 x)^2$$

  - absolute loss (least absolute deviation estimation)

$$L(\beta_0, \beta_1) = |y - \beta_0 - \beta_1 x|$$

  - Huber loss (Huberiezed estimation)

$$L(\beta_0, \beta_1) = \begin{cases} t^2, & |t| \le \delta \\ \{2\delta|t| - \delta^2\}, & \text{otherwise} \end{cases}$$

    * $t = y - \beta_0 - \beta_1 x$
    * $\delta > 0$: Huber's tuning parameter (Huber's recommendation $\delta = 1.345$)
  - penalized (regularized) loss (penalized estimation)

$$Q(\beta_0, \beta_1) = L(\beta_0, \beta_1) + J_\lambda(|\beta_1|)$$

    * L: any kind of loss
    * J_λ: penalty function
      · least absolute shrinkage and selection operator (LASSO)

$$J_\lambda(t) = \lambda t$$

      · ridge

$$J_\lambda(t) = \lambda t^2$$

      · smoothly clipped absolute deviation (SCAD)

$$\nabla J_{a,\lambda}(t) = \lambda I(t \le \lambda) + \{(a\lambda - t)_+/(a-1)\} I(t > \lambda)$$

· elastic net (ENET)

$$J_{\lambda,\gamma}(t) = \lambda t + \gamma t^2$$

* $\lambda > 0, \gamma > 0, a > 2$: tuning parameter

- In fact, different choice of losses result in different properties of estimators.
  - prediction, model identification, robustness, ...
- Hence, we need to carefully consider what kind of loss we shall use.

1.5 Estimation based on a loss function

- Steps for estimating two parameters (coefficients) β_0 and β_1
  - Determine an appropriate loss function $L(\beta_0, \beta_1)$.
  - Find the population minimizer $\hat{\beta}_0^{pop}$ and $\hat{\beta}_1^{pop}$ by minimizing the risk function

$$R(\beta_0, \beta_1) = E\{L(\beta_0, \beta_1)\}$$

    with respect to β_0 and β_1.
  * Here, note that we often cannot determine the minimizer for some possible reasons.
- For example, we can see that

$$\begin{aligned}(\hat{\beta}_0^{pop}, \hat{\beta}_1^{pop}) &= \arg\min_{\beta_0,\beta_1} E(y - \beta_0 - \beta_1 x)^2 \\ &= (\mu_y - \hat{\beta}_1 \mu_x, \sigma_{xy}^2/\sigma_x^2), \end{aligned} \tag{6}$$

  where $\mu_y = E(y)$, $\sigma_{xy}^2 = Cov(x, y)$ and $\sigma_x^2 = Var(x)$ for the square loss.
  - It is impossible to specify the minimizer unless the joint distribution of x and y is known (we do not know μ_x, μ_y, σ_x and σ_{xy}).

* (Homework) Prove (6).

1.5.1 Empirical risk function

- For the reason above, we use an empirical risk function R_n instead of a risk function R based on the samples $(x_i, y_i), i \leq n$.
  - sum of squared residuals (square)

$$R_n(\beta_0, \beta_1) = \textstyle\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2/2n$$

```
### square loss function
sq.loss.fun <- function(y.vec,x.mat,b.vec){
    r.vec <- y.vec-x.mat%*%b.vec
    n.num <- length(y.vec)
    return(sum((r.vec)^2)/2/n.num)
}

### function value comparison
b.vec.01 <- c(1,1)
sq.loss.fun(y.vec,x.mat,b.vec.01)
b.vec.02 <- c(1,0)
```

```
sq.loss.fun(y.vec,x.mat,b.vec.02)

### plot when b.vec[0]==0
g.num <- 1e+3                           # number of grids
b1.vec <- seq(-10,10,length.out=g.num)  # beta_1 vector
f.vec <- rep(0,g.num)
for(pos in 1:g.num){
    b.vec <- c(0,b1.vec[pos])           # set beta_0 to be 0
    f.vec[pos] <- sq.loss.fun(y.vec,x.mat,b.vec)
}
plot(b1.vec,f.vec,type="l",col=1,lwd=2)
```

– sum of absolute-valued residuals (absolute deviation)

$$R_n(\beta_0,\beta_1) = \sum_{i=1}^n |y_i - \beta_0 - \beta_1 x_i|/2n$$

```
### absolute deviation loss function
abs.loss.fun <- function(y.vec,x.mat,b.vec){
    r.vec <- y.vec-x.mat%*%b.vec
    n.num <- length(y.vec)
    return(sum(abs(r.vec))/2/n.num)
}

### function value comparison
b.vec.01 <- c(1,1)
abs.loss.fun(y.vec,x.mat,b.vec.01)
b.vec.02 <- c(1,0)
abs.loss.fun(y.vec,x.mat,b.vec.02)

### plot when b.vec[0]==0
g.num <- 1e+3                           # number of grid
b1.vec <- seq(1,3,length.out=g.num)     # beta_1 vector
f.vec <- rep(0,g.num)
for(pos in 1:g.num){
    b.vec <- c(0,b1.vec[pos])           # set beta_0 to be 0
    f.vec[pos] <- abs.loss.fun(y.vec,x.mat,b.vec)
}
plot(b1.vec,f.vec,type="l",col=1,lwd=2)
```

– sum of Huberized residuals (Huber)

$$R_n(\beta_0,\beta_1) = \begin{cases} \sum_{i=1}^n r_i^2/2n, & |r_i| \le \delta, \\ \sum_{i=1}^n (2\delta|r_i| - \delta^2)/2n, & |r_i| > \delta, \end{cases}$$

for some constant $\delta > 0$, where $r_i = y_i - \beta_0 - \beta_1 x_i$.

```
### Huber loss function
hub.loss.fun <- function(y.vec,x.mat,b.vec,del){
    r.vec <- y.vec-x.mat%*%b.vec
    i.vec <- r.vec<del
    return(sum(r.vec^2*i.vec + (2*del*abs(r.vec)-del^2)*(1-i.vec)))
}

### function value comparison when delta=1.345
```

```
del <- 1.345
b.vec.01 <- c(1,1)
hub.loss.fun(y.vec,x.mat,b.vec.01,del)
b.vec.02 <- c(1,0)
hub.loss.fun(y.vec,x.mat,b.vec.02,del)

### plot when b.vec[0]==0
g.num <- 1e+3                          # number of grid
b1.vec <- seq(1,3,length.out=g.num)    # beta_1 vector
f.vec <- rep(0,g.num)
for(pos in 1:g.num){
    b.vec <- c(0,b1.vec[pos])          # set beta_0 to be 0
    f.vec[pos] <- hub.loss.fun(y.vec,x.mat,b.vec,del)
}
plot(b1.vec,f.vec,type="l",col=1,lwd=2)
```

  - penalized sum of squared residuals (LASSO)

$$R_n(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2/2n + \lambda|\beta_1|,$$

    for some constant $\lambda > 0$.

  * (Homework) Write an R code as above. You need to put lambda into the function argument.

- Estimating the parameters β_0 and β_1 from the samples is minimizing empirical risk function, that is,

$$(\hat{\beta}_0, \hat{\beta}_1)^T = \arg\min_{\beta_0,\beta_1} R_n(\beta_0, \beta_1)$$

* Homework

  - Find and report some properties and algorithms for each loses.

1.6 Least squares estimation

1.6.1 Least square estimator

- Assume that the loss function is the square loss.

$$R_n(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2/2n$$

  - The resulting estimator (minimizer) is called the least square estimator (LSE).

$$(\hat{\beta}_0, \hat{\beta}_1)^T = \arg\min_{\beta_0,\beta_1} R_n(\beta_0, \beta_1).$$

  - The gradient vector of $R_n(\beta_0, \beta_1)$ becomes

$$\nabla R_n(\beta_0, \beta_1) = \begin{pmatrix} -\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)/n \\ -\sum_{i=1}^n x_i(y_i - \beta_0 - \beta_1 x_i)/n \end{pmatrix}$$

  - By putting $R_n(\beta_0, \beta_1) = \mathbf{0}$, we have

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \quad \text{and} \quad \hat{\beta}_1 = S_{xy}/S_{xx}, \tag{7}$$

    where $\bar{x} = \sum_{i=1}^n x_i/n$ and $S_{xy} = \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})/n$.

- We have corresponding matrix forms

$$\begin{aligned} R_n(\boldsymbol{\beta}) &= \|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|^2/2n \\ \nabla R_n(\boldsymbol{\beta}) &= -\mathbf{X}^T(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})/n \\ \hat{\boldsymbol{\beta}} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \Leftrightarrow \nabla R_n(\boldsymbol{\beta}) = \mathbf{0}. \end{aligned}$$

```
### gradient vector of the square loss
sq.grad.fun <- function(y.vec,x.mat,b.vec){
    r.vec <- y.vec-x.mat%*%b.vec
    return(as.vector(-t(x.mat)%*%r.vec)/length(y.vec))
}

### function value comparison
b.vec <- c(0,2)                                          # not minimizer
sq.grad.fun(y.vec,x.mat,b.vec)

lse.b.vec <- solve(t(x.mat)%*%x.mat)%*%t(x.mat)%*%y.vec  # minimizer
sq.grad.fun(y.vec,x.mat,lse.b.vec)
```

* Homework
  - Prove (7).
  - Check some statistical properties of the LSE such as unbiasedness in other classes.

1.6.2 Estimated conditional expectation and variance

- estimated conditional expectation (sample regression or fitted line)

$$\hat{f}(x) = \hat{E}(y|x) = \hat{\beta}_0 + \hat{\beta}_1 x$$

  - prediction for (new and unknown) y from an (new but known) observation x

$$\hat{y}_x = \hat{E}(y|x) = \hat{\beta}_0 + \hat{\beta}_1 x$$

    * If x increases by 1, then $\hat{y}_x$ does by $\hat{\beta}_1$.
    * If $\hat{\beta}_1 > 0$ then $\hat{y}_x$ increases as x does.
    * ...

- estimated error variance

$$\hat{\sigma}^2 = \hat{Var}(\varepsilon) = \textstyle\sum_{i=1}^n (y_i - \hat{f}(x_i))^2/(n-2) \tag{8}$$

  - $\hat{\sigma}^2$ is often referred to mean squared error (MSE) and used for statistical tests and model diagnostics.

1.6.3 A test for the model significance

- How to measure the significance or performance of the estimated line? Is this line enough to reflect the data information?
  - The slope β_1
    * $\beta_1 = 0$ implies there is no significant linear relationship between x and y since $y = \beta_0 + \varepsilon$.

  - All predictions are $\hat{y}_x = \bar{y}$, hence no predictive power of x for y.
- Statistical test for $H_0 : \beta_1 = 0$ when $\varepsilon_i \sim^{i.i.d} N(0, \sigma^2)$
  - equivalent to testing model significance
  - test statistic: $t = \hat{\beta}_1 / s.e.(\hat{\beta}_1) \sim t(n-2)$
  - reject H_0 if $|t_0| > t_{a/2}(n-2)$, which is equivalent to
  - reject H_0 if $\mathbf{P}(|t| > |t_0|) < \alpha$ (p-value), where
    - $s.e.(\hat{\beta}_1)$ is an estimate of the standard error of $\hat{\beta}_1$
    - t_0 is the observation of t under $\beta_1 = 0$
    - α is a significance level, usually $\alpha = 0.05$
- Other tests will be introduced with further complicates in the multiple linear regression model.
  - $H_0 : \beta_0 = c_0$, $H_0 : \beta_0 > 0$, $H_0 : \beta_1 = c_1$, ...

1.7 Model performance/assessment measures

Once a model is constructed, we need to assess the model performance. There may be infinitely many measures but some of them are useful and have been used. Here we see several popular examples.

1.7.1 Coefficient of determination (R^2)

- The R-square is one of the most popular measures.

$$R^2 = SSR/SST$$

  - $SST = \sum_{i=1}^n (y_i - \bar{y})^2$ (total sum of squares)
  - $SSR = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$ (regression sum of squares)
  - $SSE = SST - SSR = \sum_{i=1}^n (y_i - \hat{y}_i)^2$ (error sum of squares)
- $0 \leq R^2 \leq 1$ and better when close to 1.
- Note, R^2 cannot be used for model comparison when candidate models are nested.

* Homework

  - Draw the table for the analysis of variance.

1.7.2 Akaike's information criterion (AIC)

Assume that $\varepsilon_i \sim^{i.i.d.} N(0, \sigma^2)$ and σ^2 is known.

- Akaike's information criterion (AIC)

$$\text{AIC} = \text{AIC}(k) = -2 \log \ell(\hat{\beta}_0, \hat{\beta}_1) + 2k/n, \tag{9}$$

  where k is the number of parameters in the model (here $k = 2$) and

$$\log \ell(\beta_0, \beta) = -\log(2\pi)/2 - \log \sigma^2/2 - \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 / 2n\sigma^2$$

  is the log-likelihood divided by the sample size n.

  - The lower the AIC is the better the model is.

- When σ^2 is unknown, we may use

$$\sigma^2 \approx (n-2)\hat{\sigma}^2/n = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2/n, \tag{10}$$

  where $\hat{\sigma}^2$ is defined in (8).

  * There are many similar equations that differ from (10) but up to constants.

- If we use (10) then the AIC in (9) becomes

$$\begin{aligned} \text{AIC} &= \log(2\pi) + \log\left(\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2/n\right) + 1 + 2k/n \\ &\propto \log\left(\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2/n\right) + 2k/n \end{aligned} \tag{11}$$

- The followings are exactly equivalent to the AIC when $\varepsilon \sim N(0,1)$.
  * χ^2 test statistic
  * Mallows's C_p

* Homework
  - Prove (11).
  - Find other model assessment measures.

1.8 Statistical assumptions

- Assumptions (there can be variation) for the analysis
  - linearity: exclusion of non-linear relationships, e.g., $y_i = \beta_0 + sin(\beta_1 x_i) + \varepsilon_i$
  * c.f., $y_i = \beta_0 + \beta_1 sin(x_i) + \varepsilon_i$
  - independence : $\varepsilon_1, \cdots, \varepsilon_n$ are independent and $E(\varepsilon_i) = 0, i \leq n$
  - homoscedasticity: common fixed variance, $var(\varepsilon_i) = \sigma^2, i \leq n$
  * normality: $\varepsilon_i \sim^{i.i.d} N(0, \sigma^2), i \leq n$

1.8.1 Assumption violations and remedies

- remedies for assumption violations
  - non-linearity
    - There will be a non-linear pattern in the scatter plot between y_i and x_i
    - Consider non-linear or non-parametric models
    * e.g., polynomial regression, regression tree, neural network
  - heteroscedasticity:
    - There will be a certain pattern in the residual plot
    * residual plot: a scatter plot between $y_i - \hat{y}_i$ (residual) and x_i
    - Consider weighted linear regression
  - auto collinearity
  - non-normality

1.9 R practice

1.9.1 Diabetes samples in R

- Compare the AIC values from the simple linear regression models whose parameters are estimated by minimizing the square, absolute deviation and Huber losses.

```
rm(list=ls())

##### diabetes samples
library(lars)
data(diabetes); names(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x))
colnames(diab.df) <- c("diab",colnames(diabetes$x))
colnames(diab.df)

##### square loss
ls <- lm(diab~bmi,data=diab.df)
summary(ls)
ls$coef

##### quantle loss (tau=0.5 -> least absolute deviation loss)
library(quantreg)
lad <- rq(diab~bmi,data=diab.df,tau=0.5)
summary(lad)
lad$coef

##### Huber loss (tune=1.345 -> Huber's recommendation of delta)
library(robustreg)
hub <- robustRegH(diab~bmi,data=diab.df,tune=1.345)
summary(hub)
hub$coef

##### AIC calculation for the three methods
aic.fun <- function(beta.vec,x.mat,y.vec){ #x.mat must include the intercpt
sse <- sum((y.vec-x.mat%*%beta.vec)^2)
return(log(sse/length(y.vec))+2*length(beta.vec)/length(y.vec))
}
x.mat <- as.matrix(cbind(1,diab.df$bmi))
y.vec <- as.numeric(diab.df$diab)
aic.ls <- aic.fun(ls$coef,x.mat,y.vec)
aic.lad <- aic.fun(lad$coef,x.mat,y.vec)
aic.hub <- aic.fun(hub$coef,x.mat,y.vec)
```

1.9.2 Simulation I

- For the simple linear regression model

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,\ x_i \sim^{iid} U(0,1),\ \varepsilon_i \sim^{iid} N(0,\sigma^2),\ i \le n, \tag{12}$$

design simulation studies to check the estimation consistency of the LSE

$$n \to \infty \Rightarrow (\hat{\beta}_0, \hat{\beta}_1)^T \to (\beta_0, \beta_1)^T.$$

```
rm(list=ls())

##### true regression coefficients
true.beta.vec <- c(0,1)

##### simulations
case.num <- 100; sam.num.vec <- 10^seq(1,5,length.out=case.num)
beta.mat <- matrix(0,nrow=case.num,ncol=2)
for(case.pos in 1:case.num){
  x.mat <- cbind(1,runif(sam.num.vec[case.pos]))
  e.vec <- rnorm(sam.num.vec[case.pos])
  y.vec <- as.vector(x.mat%*%true.beta.vec+e.vec)
  beta.mat[case.pos,] <- solve(t(x.mat)%*%x.mat)%*%t(x.mat)%*%y.vec
}

##### plots
par(mfrow=c(1,2))
plot(1:case.num,beta.mat[,1]); plot(1:case.num,beta.mat[,2])
```

1.9.3 Simulation II

- For the simple linear regression model

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,\ x_i \sim^{iid} U(0,1),\ i \leq n, \tag{13}$$

we consider two distributions of the random errors: (a) $\varepsilon_i \sim^{iid} N(0,\sigma^2)$ and (b) $\varepsilon_i \sim^{iid} t(2)$. Design simulation studies to compare the model performance when the losses are square, least absolute deviation and Huber.

```
rm(list=ls())

library(quantreg)
library(robustreg)

##### functions
aic.fun <- function(beta.vec,x.mat,y.vec){ #x.mat must include the intercpt
  sse <- sum((y.vec-x.mat%*%beta.vec)^2)
  return(log(sse/length(y.vec))+2*length(beta.vec)/length(y.vec))
}
rsq.fun <- function(beta.vec,x.mat,y.vec){
    sse <- sum((y.vec-x.mat%*%beta.vec)^2); sst <- sum((y.vec-mean(y.vec))^2)
    return(1-sse/sst)
}

##### training and test sample generation
b.vec <- c(0,5) # true coefficients
tr.n.num <- 500 # tr -> training
ts.n.num <- 500 # ts -> test
#dist <- "n"
dist <- "t"; t.df <- 1

tr.x.vec <- runif(tr.n.num); ts.x.vec <- runif(ts.n.num)
```

```
if(dist=="n"){ tr.e.vec <- rnorm(tr.n.num); ts.e.vec <- rnorm(ts.n.num) }
if(dist=="t"){ tr.e.vec <- rt(tr.n.num,df=t.df); ts.e.vec <- rt(ts.n.num,df=t.df) }
tr.y.vec <- b.vec[1]+b.vec[2]*tr.x.vec+tr.e.vec
ts.y.vec <- b.vec[1]+b.vec[2]*ts.x.vec+ts.e.vec

par(mfrow=c(2,1)); plot(tr.x.vec,tr.y.vec); plot(ts.x.vec,ts.y.vec)

tr.data.df <- as.data.frame(cbind(tr.y.vec,tr.x.vec)) # training samples (dataframe)
ts.data.df <- as.data.frame(cbind(ts.y.vec,ts.x.vec)) # training samples (dataframe)
colnames(tr.data.df) <- c("y","x"); colnames(ts.data.df) <- c("y","x");

tr.x.mat <- cbind(1,tr.x.vec) # training design (matrix)
ts.x.mat <- cbind(1,ts.x.vec)
colnames(tr.x.mat) <- c("int","x"); colnames(ts.x.mat) <- c("int","x")

##### estimation
ls <- lm(y~x,data=tr.data.df) # data.frame
ls$coef
#solve(t(tr.x.mat)%*%tr.x.mat)%*%t(tr.x.mat)%*%tr.y.vec
lad <- rq(y~x,data=tr.data.df,tau=0.5)
hub <- robustRegH(y~x,data=tr.data.df,tune=1.345)

##### R square
rsq.fun(ls$coef,tr.x.mat,tr.y.vec)
rsq.fun(lad$coef,tr.x.mat,tr.y.vec) # it has no meaning
rsq.fun(hub$coef,tr.x.mat,tr.y.vec) # it has no meaning

##### AIC
aic.fun(ls$coef,tr.x.mat,tr.y.vec)
aic.fun(lad$coef,tr.x.mat,tr.y.vec)
aic.fun(hub$coef,tr.x.mat,tr.y.vec)

##### training error
sum((ls$residual)^2)/tr.n.num
#sum((tr.y.vec-predict(ls,newdata=tr.data.df))^2)/tr.n.num
#sum((tr.y.vec-tr.x.mat%*%ls$coef)^2)/tr.n.num
sum((lad$residual)^2)/tr.n.num
#sum((tr.y.vec-predict(lad,newdata=tr.data.df))^2)/tr.n.num
#sum((tr.y.vec-tr.x.mat%*%lad$coef)^2)/tr.n.num
#predict(hub,newdata=data.df)/tr.n.num  #an error occurs !!!
sum((tr.y.vec-tr.x.mat%*%hub$coef)^2)/tr.n.num

##### test error
sum((ts.y.vec-predict(ls,newdata=ts.data.df))^2)/ts.n.num
#sum((ts.y.vec-ts.x.mat%*%ls$coef)^2)/ts.n.num
sum((ts.y.vec-predict(lad,newdata=ts.data.df))^2)/ts.n.num
#sum((tr.y.vec-tr.x.mat%*%lad$coef)^2)
sum((ts.y.vec-ts.x.mat%*%hub$coef)^2)/ts.n.num
```

2 Multiple linear regression

2.1 Model and definitions

- The multiple regression is an analysis of finding a relationship f between a variable y and a vector $\mathbf{x} = (x_1, \ldots, x_p)^T$ under the presence of a random error ε.

$$y = f(\mathbf{x}) + \varepsilon \tag{14}$$

  - $y \in \mathbb{R}$: output/dependent/target variable
  - $\mathbf{x} \in \mathbb{R}^p$: input/independent/covariate vector
  - $f : \mathbb{R} \to \mathbb{R}$: model/target function of interest (unknown)
  - $\varepsilon \in \mathbb{R}$: a random error that cannot control - usually assumed to have normal distribution

- The multiple linear regression model assumes a linear relationship (modeling) between $\mathbf{x}$ and y.

$$y = f(\mathbf{x}) + \varepsilon = \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon = \mathbf{x}^T\boldsymbol{\beta} + \varepsilon \tag{15}$$

  - $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T \in \mathbb{R}^p$ is parameter/coefficient vector that specifies the model.
  - Hence the final work to do is estimating $\boldsymbol{\beta}$, that is, we are supposed to estimate p unknown coefficients $\beta_j, j \leq p$.
  - The model implies the conditional expectation of y given $\mathbf{x}$ is equal to $f(\mathbf{x})$ by assuming $E(\varepsilon|\mathbf{x}) = 0$, that is,

$$E(y|\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} = f(\mathbf{x})$$

  * When the model includes an intercept, we may think that $x_1 = 1$.

- Given n samples, $(\mathbf{x}_1, y_1)^T, \cdots, (\mathbf{x}_n, y_n)^T$, we can write an observational expression of the model as follows.

$$y_i = \mathbf{x}_i^T\boldsymbol{\beta} + \varepsilon_i, \quad i \leq n \tag{16}$$

- R Example

```
##### diabetes samples
library(lars); data(diabetes); names(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x))
colnames(diab.df) <- c("diab",colnames(diabetes$x))

##### least square estimation
diab.mod <- lm(diab~.,data=diab.df)

##### analysis of variance and other statistical results
summary(diab.mod)

##### final fitted model
diab.mod$coef
```

2.2 Interpretation of the model

- From the model, we directly can check the followings.
  - model f: conditional expectation of y given $\mathbf{x}$ (population regression line)

$$f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta} = E(y|\mathbf{x})$$

  - coefficient β_j: difference between the conditional expectations of y given $x_j = k$ and $x_j = k+1$ for any k, when the other inputs are fixed.

$$\beta_j = E(y|x_j = k+1, \mathbf{x}_{-j}) - E(y|x_j = k, \mathbf{x}_{-j}), \ \forall k$$

    * $\mathbf{x}_{-j}$: the vector obtained by deleting x_j from $\mathbf{x}$
    * In this sense, we often say β_j the effect (size) of x_j on y.
  - Note that we will see later another useful interpretation of $\hat{\beta}_j$ when we apply the least square estimation, often called the partial regression coefficient.

- The meaning of a coefficient depends on which input variables are included. For example, the following two coefficients are different.
  - β_1 in the model $E(y|x_1) = \beta_1 x_1$
  - β_1 in the model $E(y|x_1, x_2) = \beta_1 x_1 + \beta_2 x_2$

2.2.1 Special input variables - categorical variables

- When there is an indicator (often called dummy variable) among the input variables, we have an extra interpretation.
  - Let $x_1 \in \{0, 1\}$ (dummy, level 0 and 1) then
    * we may think there are two population regression lines w.r.t the levels of x_1

$$E(y|x_1 = 0, \mathbf{x}_{-1}) = \mathbf{x}_{-1}^T \boldsymbol{\beta}_{-1}$$
$$E(y|x_1 = 1, \mathbf{x}_{-1}) = \beta_1 + \mathbf{x}_{-1}^T \boldsymbol{\beta}_{-1},$$

      where $\boldsymbol{\beta}_{-1}$ is the vector obtained by deleting β_j from $\boldsymbol{\beta}$.
    * β_1 implies the difference of two conditional expectations between levels $x_1 = 0$ and $x_1 = 1$ when $\mathbf{x}_{-1}$ is fixed.

$$\beta_1 = E(y|x_1 = 1, \mathbf{x}_{-1}) - E(y|x_1 = 0, \mathbf{x}_{-1})$$

- When there is a categorical variable with $K + 1$ levels, we need to convert them into K dummy variables in the model.
  - Let $x_1 \in \{C_0, \ldots, C_K\}$ then
    * we need to consider $\mathbf{z} = (z_1, z_2, \ldots, z_K) \in \mathbb{R}^K$ with $z_k \in \{0, 1\}, k \leq K$ that satisfies $\sum_{k=1}^K z_k \in \{0, 1\}$, so that
      · $z_k = 1$ for some k implies $x_1 = C_k$
      · $z_k = 0$ for all k implies $x_1 = C_0$, which we call the base line/class/category/level/....
  - In this case, the model becomes

$$E(y|\mathbf{z}, \mathbf{x}_{-1}) = \mathbf{z}^T \boldsymbol{\alpha} + \mathbf{x}_{-1}^T \boldsymbol{\beta}_{-1},$$

    where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)^T$.
  - Hence we need to estimate $p - 1 + K$ unknown coefficients.
  - There are K population regression lines w.r.t the levels of x_1

$$E(y|x_1 = C_0, \mathbf{x}_{-1}) = \mathbf{x}_{-1}^T \boldsymbol{\beta}_{-1}$$
$$E(y|x_1 = C_1, \mathbf{x}_{-1}) = \alpha_1 + \mathbf{x}_{-1}^T \boldsymbol{\beta}_{-1}$$
$$\ldots$$
$$E(y|x_1 = C_K, \mathbf{x}_{-1}) = \alpha_K + \mathbf{x}_{-1}^T \boldsymbol{\beta}_{-1}$$

- α_k implies the difference of conditional expectations between levels C_k and C_0 when $\mathbf{x}_{-1}$ is fixed.

$$\alpha_k = E(y|x_1 = C_k, \mathbf{x}_{-1}) - E(y|x_1 = C_0, \mathbf{x}_{-1})$$

- R Example

```
##### how to make a dummy variable matrix from a vector
x.vec <- sample(x=c("A","B","O","AB"),size=15,replace=TRUE)
ux.vec <- unique(x.vec)
ux.vec <- ux.vec[-1] # the base line
z.mat <- t(t(matrix(rep(x.vec,length(ux.vec)),ncol=length(ux.vec)))==ux.vec)*1
colnames(z.mat) <- ux.vec
#cbind(z.mat,x.vec) #checking the result
```

2.2.2 Special input variables - product terms

- Consider the model

$$E(y|x_1, x_2, x_3) = \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \beta_3 x_3.$$

  - It is easy to see that

$$E(y|x_1, x_2, x_3 = k + 1) - E(y|x_1, x_2, x_3 = k) = \beta_3, \forall k \in \mathbb{R}$$

    as we have seen above, but

$$E(y|x_1 = k + 1, x_2, x_3) - E(y|x_1 = k, x_2, x_3) = \beta_1 + \beta_{12} x_2, \forall k \in \mathbb{R}$$

    which shows that the effect size of x_1 on y depends on x_2.
  - We often call this phenomenon interaction between x_1 and x_2

- If x_1 is dummy then we are investigating two models

$$E(y|x_1 = 0, x_2, x_3) = \beta_2 x_2 + \beta_3 x_3$$
$$E(y|x_1 = 1, x_2, x_3) = \beta_1 + (\beta_2 + \beta_{12}) x_2 + \beta_3 x_3$$

  that have different intercept and slope of x_2.

- If both x_1 and x_2 are dummies then we are investigating four models

$$E(y|x_1 = 0, x_2 = 0, x_3) = \beta_3 x_3$$
$$E(y|x_1 = 0, x_2 = 1, x_3) = \beta_2 + \beta_3 x_3$$
$$E(y|x_1 = 1, x_2 = 0, x_3) = \beta_1 + \beta_3 x_3$$
$$E(y|x_1 = 1, x_2 = 1, x_3) = \beta_1 + \beta_2 + \beta_{12} + \beta_3 x_3$$

  that have different intercepts.

- R Example

```
library(lars)
data(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x2))
colnames(diab.df) <- c("diab",colnames(diabetes$x2))
colnames(diab.df)
diab.df[1:5,]
lm(diab~.,data=diab.df)$coef
```

2.3 Estimation

2.3.1 Matrix-vector notation

- Given n samples, $(\mathbf{x}_1^T, y_1)^T, \cdots, (\mathbf{x}_n^T, y_n)^T$, we can write observational expression from the model as follows:

$$y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i, \quad i \leq n, \tag{17}$$

  where $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})^T \in \mathbb{R}^p$.

- Let $\mathbf{y} = (y_1, \ldots, y_n)^T \in \mathbb{R}^n$, $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_p) \in \mathbb{R}^{n \times p}$, where $\mathbf{X}_j = (x_{1j}, \ldots, x_{nj})^T \in \mathbb{R}^n$ and $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_n)^T \in \mathbb{R}^n$, then we can write the equations in (17) as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

2.3.2 Design matrix

- We call the matrix $\mathbf{X}$ the design matrix of the model,

$$\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_p) = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix}$$

  - Usually, $\mathbf{X}^T\mathbf{X}/n$ (Hessian matrix) is assumed to be invertible so that $(\mathbf{X}^T\mathbf{X}/n)^{-1}$ exists.
  - Hence, $\mathbf{X}$ must satisfy
    - the number of variables is less than the number of samples ($p \leq n$),
    - all the columns are linearly independent.

- R example

```
##### invertibility of the Hessian
library(lars); data(diabetes)
x.mat <- cbind(1,as.matrix(diabetes$x))
eigen(t(x.mat)%*%x.mat)
```

2.3.3 Least square estimation

- The least square estimation is minimizing the sum of squared residuals.

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \textstyle\sum_{i=1}^n (y_i - \mathbf{x}_i^T\boldsymbol{\beta})^2/2n = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2/2n \tag{18}$$

  - The solution must satisfy

$$-2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{0} \Leftrightarrow \hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \tag{19}$$

    which is called the least square estimator (LSE).

- R example

```
##### least square estimator
library(lars); data(diabetes)
x.mat <- cbind(1,as.matrix(diabetes$x)); y.vec <- as.vector(diabetes$y)
drop(solve(t(x.mat)%*%x.mat)%*%t(x.mat)%*%y.vec)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x)); colnames(diab.df)[1] <- "diab"
lm(diab~.,data=diab.df)$coef
```

2.4 Estimated conditional expectation and variance

- estimated conditional expectation

$$\hat{f}(\mathbf{x}) = \hat{E}(y|\mathbf{x}) = \mathbf{x}^T\hat{\boldsymbol{\beta}}$$

  - prediction for new and unobserved y from a new but observed $\mathbf{x}$

$$\hat{y}_{\mathbf{x}} = \hat{E}(y|\mathbf{x}) = \mathbf{x}^T\hat{\boldsymbol{\beta}} = \mathbf{x}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

    * The predictive value of y increases by $\hat{\beta}_j$ as x_j increases by 1 when $\mathbf{x}_{-j}$ is fixed.

$$\hat{\beta}_j = \hat{E}(y|x_j = k+1, \mathbf{x}_{-j}) - \hat{E}(y|x_j = k, \mathbf{x}_{-j})$$

- estimated error variance

$$\hat{\sigma}^2 = \hat{V}ar(\varepsilon) = \textstyle\sum_{i=1}^{n}(y_i - \hat{f}(\mathbf{x}_i))^2/(n-p)$$

2.4.1 Interpretation of estimated coefficients $\hat{\beta}_j$

- The estimated coefficient $\hat{\beta}_j$ often called partial regression coefficients of x_j since it corresponds to the estimated coefficient of the simple linear regression of y on x_j after taking out the effect of $\mathbf{x}_{-j}$.
  - First, fit the model $x_j = \mathbf{x}_{-j}^T\alpha_{-j} + \varepsilon_1$,
  - and then construct the residuals $r_{x_j} = x_j - \hat{x}_j = x_j - \mathbf{x}_{-1}\hat{\boldsymbol{\alpha}}_{-1}$
  - Second, fit the model $y = \mathbf{x}_{-j}^T\boldsymbol{\gamma}_{-j} + \varepsilon_2$,
  - and then construct the residuals $r_y = y - \hat{y} = y - \mathbf{x}_{-1}\hat{\boldsymbol{\gamma}}_{-1}$
  - Third, fit the model $r_y = \theta_j r_{x_j} + \varepsilon_3$
  - and then eventually we have $\hat{\beta}_j = \hat{\theta}_j$.

- R example

```
##### least square estimator: dataframe mode with intercept
library(lars); data(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x)); colnames(diab.df)[1] <- "diab"

### the first step
diab.x.df <- subset(diab.df,select=-diab)
diab.x.mod <- lm(bmi~.,data=diab.x.df)
x.res <- diab.x.mod$resid

### the second step
diab.mod <- lm(diab~.,data=subset(diab.df,select=-bmi))
y.res <- diab.mod$resid

### the third step
lm(y.res~x.res)$coef[2]

### cross check
lm(diab~.,data=diab.df)$coef

##### least square estimator: matrix mode
```

```
x.mat <- cbind(1,diabetes$x)
y.vec <- diabetes$y

### the first step
x0.mat <- x.mat[,colnames(x.mat)!="bmi"]
y0.vec <- x.mat[,colnames(x.mat)=="bmi"]
x.res <- y0.vec-as.vector(x0.mat%*%solve(t(x0.mat)%*%x0.mat)%*%t(x0.mat)%*%y0.vec)

### the second step
y.res <- y.vec-as.vector(x0.mat%*%solve(t(x0.mat)%*%x0.mat)%*%t(x0.mat)%*%y.vec)

### the third step
sum(x.res*y.res)/sum(x.res^2) # sum of residuals (x.res) is zero

### cross check
solve(t(x.mat)%*%x.mat)%*%t(x.mat)%*%y.vec
```

2.5 Tests for the model/coefficients significance

- model significance
  - Let $\mathbf{X} = (\mathbf{1}, \mathbf{X}_1, \ldots, \mathbf{X}_p)$ and $\mathbf{\Pi_X} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}$, where $X_j = (x_{j1}, \ldots, x_{jn})^T, 1 \le j \le p$ and $\mathbf{1} = (1, \cdots, 1)^T$ for the linear regression model:

$$y_i = \beta_0 + \mathbf{x}_i^T\boldsymbol{\beta} + \varepsilon_i = \beta_0 + \beta_1 x_{1i} + \cdots \beta_p x_{pi} + \varepsilon_i,\ i \le n$$

  - $\boldsymbol{\beta} = \mathbf{0}$ ($\beta_j = 0, 1 \le j \le p$) implies there is no significant (statistical) evidence on constructing a linear relationship between $\mathbf{x}$ and y.
  - statistical test for $H_0 : \boldsymbol{\beta} = \mathbf{0}$ when $\varepsilon_i \sim^{i.i.d} N(0, \sigma^2)$
    - test statistic:

$$F_0 = (SSR/p)/\{SSE/(n-p-1)\} = \{\mathbf{y}^T(\mathbf{\Pi_X} - \mathbf{\Pi_1})\mathbf{y}/p)\}/\{\mathbf{y}^T(I - \mathbf{\Pi_X})\mathbf{y}/(n-p-1)\}$$

    - reject H_0 if $|F_0| > F_\alpha(p, n-p-1)$ or
    - reject H_0 if $\mathbf{P}(F > F_0) < \alpha$ (p-value)

- coefficients significance after model significance
  - $\beta_j = 0$ implies there is no significant (statistical) evidence on the inclusion of x_j in the model.
  - statistical test for $H_0 : \beta_j = 0$ when $\varepsilon_i \sim^{i.i.d} N(0, \sigma^2)$
    - test statistic: $t_{j0} = \hat{\beta}_j / s.e.(\hat{\beta}_j)$
    - reject H_0 if $|t_{j0}| > t_{\alpha/2}(n-p-1)$, which is equivalent to
    - reject H_0 if $\mathbf{P}(|t| > t_{j0}) < \alpha$ (p-value)

- other tests on parameters
  - Various types of tests such as $\mathbf{C}\boldsymbol{\beta} = 0$ for some $\mathbf{C}$ can be done.
    - If $\mathbf{C} = \mathbf{C}_1 = (1, 0, \ldots, 0)$ then $\mathbf{C}\boldsymbol{\beta} = \mathbf{0} \Leftrightarrow \beta_1 = 0$.
    - If $\mathbf{C} = \mathbf{C}_2 = (0, 1, -1, 0, \ldots, 0)$ then $\mathbf{C}\boldsymbol{\beta} = \mathbf{0} \Leftrightarrow \beta_2 = \beta_3$.
    - If $\mathbf{C} = (\mathbf{C}_1^T, \mathbf{C}_2^T)^T$ then $\mathbf{C}\boldsymbol{\beta} = \mathbf{0} \Leftrightarrow \beta_1 = 0$ and $\beta_2 = \beta_3$.

- The full model $y = \mathbf{x}^T\boldsymbol{\beta}$ (FM) always includes the constrained or reduced model $y = \mathbf{x}^T\boldsymbol{\beta}, \mathbf{C}\boldsymbol{\beta} = 0$ (RM). Hence we can use the F test.

$$F = \{(SSE_R - SSE_F)/\Delta_{df}\}/\{SSE_F/(n-p-1)\} \sim F(\Delta_{df}, n-p-1)$$

  * SSE_F and SSE_R are SSEs of the FM and RM, respectively.
  * $\Delta_{df} = p + 1 -$ # of parameters in the RM, for example, $\Delta_{df} = 1, 1, 2$ in the above constraints.

2.6 Model performance measrues

2.6.1 R square (R^2)

- Coefficient of determination (R^2, R-square)

$$R^2 = SSR/SST$$

  - $SST = \sum_{i=1}^n (y_i - \bar{y})^2 = \mathbf{y}^T(I - \mathbf{\Pi_1})\mathbf{y}$ (total sum of squares)
  - $SSR = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 = \mathbf{y}^T(\mathbf{\Pi_X} - \mathbf{\Pi_1})\mathbf{y}$ (regression sum of squares)
  - $SSE = SST - SSR = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \mathbf{y}^T(I - \mathbf{\Pi_X})\mathbf{y}$ (error sum of squares)

- R^2 means the proportion of the variation in the data explained by the model.
  * $0 \le R^2 \le 1$ and better when close to 1.
  * Note adding more input variables into current model (and hence the model size increases) implies that R^2 always increases, no matter what those input variables are.

- R example

```
rm(list=ls())
##### R square of the linear regression model from the least square estimation
library(lars); data(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x2)); colnames(diab.df)[1] <- "diab"

##### example 1 variable vs 2 variables
r.sq.1 <- summary(lm(diab~.,data=diab.df[,1:2]))$r.sq
r.sq.2 <- summary(lm(diab~.,data=diab.df[,1:3]))$r.sq

##### pattern plot
x.num <- ncol(diab.df)-1; y.num <- nrow(diab.df)
r.sq.vec <- rep(0,x.num)
for(x.pos in 1:x.num){
  r.sq.vec[x.pos] <- summary(lm(diab~.,data=diab.df[,1:(x.pos+1)]))$r.sq
}
plot(r.sq.vec); max(r.sq.vec)

##### adding some noisy variables
noise.df <- as.data.frame(matrix(rnorm(x.num*y.num),ncol=x.num,nrow=y.num))
diab.df <- cbind(diab.df,noise.df)
x.num <- ncol(diab.df)-1; y.num <- nrow(diab.df)
r.sq.vec <- rep(0,x.num)
for(x.pos in 1:x.num){
  r.sq.vec[x.pos] <- summary(lm(diab~.,data=diab.df[,1:(x.pos+1)]))$r.sq
}
plot(r.sq.vec); max(r.sq.vec)
```

2.6.2 Adjusted R square (R_a^2)

- Adjusted coefficient of determination (R_a^2, adjusted-R-square)

$$\begin{aligned} R_a^2 &= 1 - \{SSE/(n-p-1)\}/\{SST/(n-1)\} \\ &= 1 - \{(n-1)/(n-p-1)\}(1-R^2) \end{aligned}$$

  - By adjusting the model size R_a^2 can be used for comparing models of different sizes.

- R example

```
rm(list=ls())
##### Adjusted R square of the linear regression model from the least square estimation
library(lars); data(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x2)); colnames(diab.df)[1] <- "diab"

##### example 1 variable vs 2 variables
adj.adj.r.sq.1 <- summary(lm(diab~.,data=diab.df[,1:2]))$adj.r.sq
adj.r.sq.2 <- summary(lm(diab~.,data=diab.df[,1:3]))$adj.r.sq

##### pattern plot
x.num <- ncol(diab.df)-1; y.num <- nrow(diab.df)
adj.r.sq.vec <- rep(0,x.num)
for(x.pos in 1:x.num){
  adj.r.sq.vec[x.pos] <- summary(lm(diab~.,data=diab.df[,1:(x.pos+1)]))$adj.r.sq
}
plot(adj.r.sq.vec); max(adj.r.sq.vec)

##### adding some noisy variables
noise.df <- as.data.frame(matrix(rnorm(x.num*y.num),ncol=x.num,nrow=y.num))
diab.df <- cbind(diab.df,noise.df)
x.num <- ncol(diab.df)-1; y.num <- nrow(diab.df)
adj.r.sq.vec <- rep(0,x.num)
for(x.pos in 1:x.num){
  adj.r.sq.vec[x.pos] <- summary(lm(diab~.,data=diab.df[,1:(x.pos+1)]))$adj.r.sq
}
plot(adj.r.sq.vec); max(adj.r.sq.vec)
```

2.6.3 Other performance measures

We will see how to compare given candidate models later. Here I list up some of them.

- Generalized information criteria (AIC, BIC, FIC, ...)
- Cross validation errors
- Independent test errors

2.7 Variations of the linear regression model

2.7.1 Polynomial regression model

- The polynomial regression model includes higher order terms of input variables:

$$\begin{aligned} y &= \beta_0 + \sum_{j=1}^{p}\sum_{k=1}^{q_j}\beta_{jk}x_j^k + \varepsilon \\ &= \beta_0 + (\beta_{11}x_1 + \cdots + \beta_{1p_1}x_1^{p_1}) + \cdots + (\beta_{p1}x_1 + \cdots + \beta_{pq_p}x_1^{q_p}) + \varepsilon \end{aligned} \tag{20}$$

- R example

```
##### formula example
#symbol example action
+      +X      include X
-      -X      delete X
:      X:Y     include XY
*      X*Y     include X,Y and XY
1      -1      exclude the intercept
I     I(X*Y)   include the term in the indicator

##### second order polynomial regression with one input variable
n.num <- 1e+2
x.vec <- runif(n.num,0,2)
y.vec <- x.vec^2 + rnorm(n.num)
data.df <- as.data.frame(cbind(y.vec,x.vec))
quad.mod <- lm(y.vec~x.vec+I(x.vec^2),data=data.df)
lin.mod <- lm(y.vec~x.vec,data=data.df)
plot(x.vec,y.vec)
points(x.vec,quad.mod$fit,col="blue")
points(x.vec,lin.mod$fit,col="red")

##### second order polynomial regression with two input variables
x.mat <- matrix(runif(n.num*2,0,4),ncol=2)
y.vec <- as.vector(x.mat%*%c(2,2)+(x.mat^2)%*%c(2,2)+rnorm(n.num))
data.df <- as.data.frame(cbind(y.vec,x.mat))
colnames(data.df) <- c("y","x1","x2")
quad.mod <- lm(y~x1+x2+I(x1^2)+I(x2^2),data=data.df)

##### example of formula construction with an error
library(lars); data(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x) )
colnames(diab.df)[1] <- "diab"
diab.x.df <- subset(diab.df,select=-diab)
x.name <- colnames(diab.x.df)
x2.name <- paste("I(",x.name,"^2)",sep="")
lm.form <- paste("diab~",paste(c(x.name,x2.name),collapse="+"),sep="")
lm.form <- as.formula(lm.form)
lm(lm.form,data=diab.df)#  option: singular=F

##### error correction
x2.name <- x2.name[x2.name!="I(sex^2)"]
lm.form <- paste("diab~",paste(c(x.name,x2.name),collapse="+"),sep="")
lm.form <- as.formula(lm.form)
lm(lm.form,data=diab.df)
```

2.7.2 Generalized least squares

See later

2.8 R practice

2.8.1 Simulation I

- For the model, design simulation studies for checking the followings:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \varepsilon_i,\ i \le n,$$

where $x_{i1}, x_{i2} \sim^{i.i.d.} U(0,1)$ and $\varepsilon_i \sim^{i.i.d.} N(0,\sigma^2)$, respectively.

  - estimation consistency of $\hat{\boldsymbol{\alpha}} = (\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2)^T$ for $\boldsymbol{\alpha} = (\beta_0, \beta_1, \beta_2)^T$
    - c.f., $\|\hat{\boldsymbol{\alpha}} - \boldsymbol{\alpha}\| = O_p(1/\sqrt{n})$

2.8.2 Simulation II

- Consider the model with $p = 10$ input variables,

$$y_i = \beta_0 + \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i,\ i \le n,$$

  - $\mathbf{x}_i \sim^{i.i.d} N(\boldsymbol{\mu}, \boldsymbol{\Sigma}),\ i \le n$ with $\mu_j = 0$ and $\Sigma_{jk} = \rho^{|j-k|},\ j,k \le p$,
  - $\beta_0 = 0$ and $\beta_j = 3/j,\ j \le p$,
  - $\varepsilon_i \sim^{i.i.d.} N(0,\sigma^2),\ i \le n$

Let '#Sig' be the number of significant input variables based on the t-test with significance level $\alpha = 0.05$. For $p \in \{5, 10, 20\}$, $n \in \{100, 500\}$, $\rho \in \{0.2, 0.8\}$ and $\sigma \in \{2, 4\}$, construct a table that includes the average number of '#Sig' in 300 simulations.

```
##### variation with respect to the sample size n others being fixed
rm(list=ls()); library(MASS)

### other constants
s.num <- 300  # number of simulations
p.num <- 10   # number of input variables
r.val <- 0.2  # correlation factor
a.val <- 0.05 # significance level

### parameters for sample generation
sig.mat <- r.val^(abs(outer(1:p.num,1:p.num,FUN="-"))) # variance of input vector
mu.vec <- rep(0,p.num)                                 # mean of input vector
b.vec <- 1/(1:p.num)                                   # true regression coefficient vector
s.val <- 2                                             # error variance

### simulations w.r.t sample size
n.vec <- (5:25)*100; n.num <- length(n.vec)
rst.mat <- matrix(0,n.num,p.num+1)
for(n.pos in 1:n.num){
    in.rst.mat <- matrix(0,s.num,p.num+1)
    for(s.pos in 1:s.num){
        x.mat <- mvrnorm(n.vec[n.pos],mu.vec,sig.mat)
        y.vec <- as.vector(x.mat%*%b.vec)+rnorm(n.vec[n.pos],0,s.val)
        xy.df <- as.data.frame(cbind(y.vec,x.mat))
        mod <- as.formula(paste(colnames(xy.df)[1],"~.",sep=""))
        in.rst.mat[s.pos,] <- coef(summary(lm(mod,data=xy.df)))[,4] < a.val
```

```
    }
    rst.mat[n.pos,] <- colMeans(in.rst.mat)
    print(n.pos)
}

### plotting
plot(1:n.num,rst.mat[,1],xlim=c(0,n.num+7),ylim=c(0,1.1),type="n")
for(p.pos in 1:(p.num+1)){
    lines(1:n.num,rst.mat[,p.pos],col=p.pos,lty=2)
    points(1:n.num,rst.mat[,p.pos],col=p.pos,pch=p.pos)
}
leg.vec <- c("int",colnames(xy.df)[-1])
legend("right",legend=leg.vec,pch=1:(p.num+1),col=1:(p.num+1),lty=2)
```

3 Variable selection

3.1 Preliminaries

- Analysis strategy and model uncertainty
  - parameter estimation and test: (generalized) least square/maximum likelihood/robust/penalized/Bayesian/... together with $t/F/\mathrm{LR}/R^2/...$
  - model diagnostics: checking assumptions (constant variance, linearity, normality, ...) and detecting outliers/influential points/serial correlations/collinearity ...
  - variable transformation: transforming the input/output variables by using Box-Cox/polynomial/spline/...
  - variable selection: selection procedures based on test statistic/information criterion/prediction eror/...
- We will focus on the multiple linear regression only. The fundamental idea is almost common to other methods.

3.1.1 Interpretation vs Prediction

Consider the linear regression model,

$$y = \mathbf{x}^T\boldsymbol{\beta} + \varepsilon, \tag{21}$$

where $\mathbf{x} = (x_1, \ldots, x_p)^p$.

- Two primary purposes of regression analysis:
  - easy model interpretation
  - high prediction accuracy
- We prefer to the simplest (smallest) model among possible linear models while do not want to lose prediction power.
- Unfortunately, it is known that we cannot achieve both goals simultaneously even asymptotically!
- R example

```
##### prediction vs model identification
rm(list=ls()); library(MASS)

### constants
s.num <- 300  # number of simulations
p.num <- 10   # number of input variables
r.val <- 0.5  # correlation factor
n.num <- 500  # sample size (try n.num <- 50)

### parameters for sample generation
sig.mat <- r.val^(abs(outer(1:p.num,1:p.num,FUN="-"))) # variance of input vector
mu.vec  <- rep(0,p.num)                                # mean of input vector
b.vec   <- 1/(1:p.num)                                 # true regression coefficient vector
s.val   <- 2                                           # error standard deviation

### training sample generation
x.mat <- mvrnorm(n.num,mu.vec,sig.mat)
y.vec <- as.vector(x.mat%*%b.vec)+rnorm(n.num,0,s.val)
```

```
xy.df <- as.data.frame(cbind(y.vec,x.mat))
colnames(xy.df) <- c("y",paste("x",1:p.num,sep=""))

### test sample generation
nx.mat <- mvrnorm(n.num,mu.vec,sig.mat)
ny.vec <- as.vector(nx.mat%*%b.vec)+rnorm(n.num,0,s.val)
nxy.df <- as.data.frame(cbind(ny.vec,nx.mat))
colnames(nxy.df) <- c("y",paste("x",1:p.num,sep=""))

### training/test error
tr.vec <- te.vec <- rep(0,p.num)
for(p.pos in 1:p.num){
    fit <- lm(y~.,data=xy.df[,c(1,2:(p.pos+1))])
    tr.vec[p.pos] <- mean(( y.vec-predict(fit,newdata= xy.df[,c(1,2:(p.pos+1))]))^2)
    te.vec[p.pos] <- mean((ny.vec-predict(fit,newdata=nxy.df[,c(1,2:(p.pos+1))]))^2)
}
par(mfrow=c(1,2))
plot(tr.vec); plot(te.vec) # check the pattern!
```

3.2 Variable selection

- We first should investigate at least two issues:
  - How to check whether $\mathbf{x}$ includes all the relevant variables?
  - How to include all the relevant variables among $\mathbf{x}$ if they are included in?
- In fact, the first issue is the hardest to resolve, and to my private opinion, it may be far away from statisticians but close to data specialists.
  - Hence, we assume that all the relevant variables are really included in $\mathbf{x}$ (thinking $\mathbf{x}$ is the best we can consider), so that we focus on the second issue in this class.
- We usually say we are doing variable/model selection or dimension reduction when our task belongs to the issues above.
  - To have an ideal model, we need to include all the relevant variables.
  - To have a simple model, we may exclude many of them.

3.2.1 Variable selection process

- Almost all the selection procedures consist of two steps, constructing and comparing candidate models, or a hybrid of these two.
  - constructing candidate models
    - Shall we consider all the possible candidate models? If possible yes, but no in practice.
    - The model includes 2^p candidate models!
    - Then it is the key how to reduce the number of candidate models.
      - subset selection/model constraining/dimension reduction/...
  - comparing candidate models
    - We need to define a rule or measure of comparing two different candidate models.
      - statistical test/theoretical comparison/data adaptive comparison/...

3.3 Comparing models - Statistical test

- When two models are nested, we can use the F-test to compare them.
- Here the term nested implies there exists a matrix $\mathbf{C}$ such that two models can be represented as

$$\begin{aligned} \text{full model (FM)} \quad & y = \mathbf{x}^T\boldsymbol{\beta} + \varepsilon \\ \text{reduced model (RM)} \quad & y = \mathbf{x}^T\boldsymbol{\beta} + \varepsilon, \quad \mathbf{C}\boldsymbol{\beta} = 0. \end{aligned}$$

- In this case the F-statistic, $F \sim F(D_R - D_F, D_F)$ decides which one is better by testing $H_0 : \mathbf{C}\boldsymbol{\beta} = 0$, where

$$F = \{(SSE_R - SSE_F)/(D_R - D_F)\}/\{SSE_F/D_F\}$$

  - SSE_F and SSE_R: SSEs of the FM and RM.
  - D_F and D_R: $n-$(# of parameters) in the FM and RM.

- R example

```
##### F-test for quadratic and interaction effect
rm(list=ls())

### diabetes samples with quadratic and interaction terms
library(lars)
data(diabetes); names(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x2))
n.num <- dim(diab.df)[1]; p.num <- dim(diab.df)[2]-1
colnames(diab.df) <- c("diab",colnames(diabetes$x2))
colnames(diab.df)

### full model vs reduced model via F test
f.fit <- lm(diab~.,data=diab.df)
r.mod <- paste("diab~",paste(colnames(diab.df)[2:11],collapse="+"),sep="")
r.fit <- lm(r.mod,data=diab.df)
anova(r.fit,f.fit)

### direct calculation
up <- ((sum(r.fit$res^2)-sum(f.fit$res^2)))/(p.num-10)
down <- sum(f.fit$res^2)/(n.num-p.num-1)
f.stat <- up/down
pf(f.stat,p.num-10,n.num-p.num-1)
```

3.4 Comparing models - Information criteria

- The generalized information criterion (GIC) indexed by a λ is defined as follows (Kim and Kwon, 2012).

$$\text{GIC}_\lambda(\mathcal{S}) = \|\mathbf{y} - \mathbf{X}_\mathcal{S}\hat{\boldsymbol{\beta}}_\mathcal{S}\|^2 + \lambda|\mathcal{S}|\sigma^2,$$

  - $\mathbf{X}_\mathcal{S}$: sub-matrix consists of columns $\mathbf{X}_j, j \in \mathcal{S} \subset \{1, \ldots, p\}$,
  - $\hat{\boldsymbol{\beta}}_\mathcal{S} = \arg\min_{\beta_j=0: j\in\mathcal{S}^c} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$: least square estimator based on $\mathbf{X}_\mathcal{S}$
  - $|\mathcal{S}|$: the cardinality of $\mathcal{S}$,
  - λ: a predefined positive sequence of the sample size n and model size p.

  - $\sigma^2 = Var(\varepsilon)$ which is known
    * When σ^2 is unknown, we may use the MSE of the full or any overfitted model.

- The final estimator from the GIC is defined as

$$\hat{\mathcal{S}} = \arg\min_{\mathcal{S}\in\{1,\ldots,p\}} \mathrm{GIC}_\lambda(\mathcal{S}).$$

  - When p is large, we may consider $|\mathcal{S}| < s < p$ for a small $s < 15$.

- Examples of the GIC
  - AIC if $\lambda = 2$: Akaike IC by Akaike (1973),
  - BIC or SIC if $\lambda = \log n$: Bayesian or Schwarz IC by Schwarz (1978),
  - RIC if $\lambda = 2\log p$: Risk IC by Foster and George (1994),
  - MRIC if $\lambda =$???: Modified RIC by Foster and George (1994),
  - CRIC if $\lambda = 2(\log p + \log\log p)$: Corrected RIC by Zhang and Shen (2010),
  - MBIC if $\lambda = \log\log p \log n$: Modified BIC by Wang et al. (2009).
  - EBIC (asymptotically) if $\lambda = \log n + 2\kappa \log p$ for some $0 < \kappa \leq 1$: Extended BIC by Chen and Chen (2008), where

$$\mathrm{EBIC}_\lambda(\mathcal{S}) = \|\mathbf{y} - \mathbf{X}_\mathcal{S}\hat{\boldsymbol{\beta}}_\mathcal{S}\|^2 + (\lambda|\mathcal{S}| + 2\kappa \log \tbinom{p}{|\mathcal{S}|})\sigma^2$$

  * $\lambda|\mathcal{S}|\sigma^2 + 2\kappa \log \binom{p}{|\mathcal{S}|}\sigma^2 \asymp (\log n + 2\kappa \log p)|\mathcal{S}|\sigma^2$

- R example

```
##### AIC vs BIC over some candidate models
rm(list=ls()); library(lars); data(diabetes)
diab.df <- as.data.frame(cbind(diabetes$y,diabetes$x2)); colnames(diab.df)[1] <- "diab"
n.num <- dim(diab.df)[1]; p.num <- dim(diab.df)[2]-1
a.lam <- 2                          # AIC
b.lam <- log(n.num)    # BIC
sig.hat <- sum((lm(diab~.,data=diab.df[1:9])$res)^2)/(n.num-11) # MSE from full model
aic.vec <- bic.vec <- rep(0,p.num)
for(pos in 1:p.num){
    mod <- paste("diab~",paste(colnames(diab.df)[-1][1:pos],collapse="+"),sep="")
    aic.vec[pos] <- sum((lm(mod,data=diab.df)$res)^2)+a.lam*(pos+1)*sig.hat
    bic.vec[pos] <- sum((lm(mod,data=diab.df)$res)^2)+b.lam*(pos+1)*sig.hat
}
plot(aic.vec/n.num,pch="+",col=1); points(bic.vec/n.num,pch="-",col=2)
points(which.min(aic.vec),min(aic.vec)/n.num,pch="A",col=1)
points(which.min(bic.vec),min(bic.vec)/n.num,pch="B",col=2)
```

3.4.1 Theories on the GIC

- Let $\mathcal{S}^*$ be the index set of truly relevant predictors.
- We say that the GIC is consistent in model selection if $\hat{\mathcal{S}}$ satisfies

$$\mathbf{P}(\hat{\mathcal{S}} = \mathcal{S}^*) \to 1.$$

  - The GIC is consistent for any constants $0 \leq 2c_1 < c_2 \leq 1$ that depend on regularity conditions,

  - if $\lambda = o(n^{c_1 - c_1})$ and $p/(\lambda\rho_n)^k \to 0$ for some positive integer k when $E(\varepsilon^{2k}) < \infty$.
  - if $\lambda = o(n^{c_1 - c_1})$, $s \log p = o(n^{c_1 - c_1})$ and $\lambda - 2\log p - \log\log p \to \infty$ when $\varepsilon \sim N(0, \sigma^2)$ for some constant $\sigma^2 < \infty$.

- We can see that the GIC is consistent for any constant $c > 0$ along to several scenarios
  - (fixed dimension) $p = cn^0 = c$ is fixed, $\lambda/n \to 0$ and $\lambda \to \infty$.
    - BIC (but not AIC), any GIC with $\lambda = n^\delta, 0 < \delta < 1$
  - (polynomial dimension) $p = cn^\gamma, \gamma > 0$, $\lambda = n^\xi, , 0 < \xi < 1$ and $\gamma < k\xi$.
    - BIC with $\gamma < 1/2$, MBIC with $0 < \gamma < 1$
    - EBIC with $\kappa > 1 - 1/2\gamma$
  - (exponential dimension) $p = \exp(cn^\gamma)$, $\lambda = n^\xi, 0 < \xi < 1$, $0 < \gamma < \xi$ and $\varepsilon \sim N(0, \sigma^2)$.
    - EBIC with $\gamma = 1$ and $0 < \gamma < 1/2$, CRIC
  - * This suggests that the class of consistent model selection criteria is quite large, and we need to choose a IC very carefully.

- R example

```
Sorry!!!!!
```

3.5 Comparing models - Prediction based approaches

- There are many data predictive measures that try to estimate the expected model error or true prediction error.
  - training error, prediction error with independent samples, cross validation error, leave-one-out error, ...
  - These criteria are
    - quite popular for the simplicity!
    - known to overfit the true subset $\mathcal{S}^*$ (as the AIC) for some cases, that is

$$\mathbf{P}(\hat{\mathcal{S}} \supset \mathcal{S}^*) \to 1.$$

    - suspicious since the asymptotic properties are largely unknown.
  - You may use them in selecting variables but keep in you mind they do not give you a consistent model but a predictive model.

3.5.1 Independent training-test error

- Assume that we have enough samples so that we can divide the samples into two homogeneous sets of samples: one is for training and the other is for testing.
  - We often say the two sets of samples training and test samples.
  - The number of training samples often set to be larger than the number of test samples but it depends on situations.
  - I often use one or two times more training samples than test samples.

- Once we have training and test samples we do as follows.
  - Fit a model by using the training samples.

  - Calculate prediction error by using test samples.
  * If you have another model assessment measure then you may use it.

- The independent test error depends on how to divide the samples so that there must be unexpected error we cannot control.

- R example

```
##### independent test error calculation, diabetes samples
rm(list=ls()); library(lars); data(diabetes)

### whole samples
y.vec <- diabetes$y; x.mat <- diabetes$x; n.num <- length(y.vec)

### randomly select training sample positions
tr.n.num <- round(n.num/2) # number of training samples
pos.vec <- sample(1:n.num) # re-order the samples randomly
pos.vec <- pos.vec[1:tr.n.num]

### training samples
tr.y.vec <- y.vec[pos.vec]
tr.x.mat <- x.mat[pos.vec,]
tr.df <- as.data.frame(cbind(tr.y.vec,tr.x.mat))
colnames(tr.df) <- c("diab",colnames(tr.x.mat))
tr.df[1:10,]

### test samples
ts.y.vec <- y.vec[-pos.vec]
ts.x.mat <- x.mat[-pos.vec,]
ts.df <- as.data.frame(cbind(ts.y.vec,ts.x.mat))
colnames(ts.df) <- c("diab",colnames(ts.x.mat))
ts.df[1:10,]

### fit a model by using the training samples
tr.fit <- lm(diab~.,data=tr.df)
tr.fit$coef

### calculate test error (prediction error) by using the test samples
ts.y.hat.vec <- predict(tr.fit,newdata=ts.df)  # prediction of y.vec
ts.err <- mean((ts.y.vec-ts.y.hat.vec)^2)
ts.err
```

3.5.2 Cross validation (CV) error

- The cross-validation (CV) method is a predictive measure that is useful when the sample size is relatively small so that we cannot use independent test samples.

- Here are steps for the K-fold cross-validation method:

  - Divide the whole sample into K distinct sets of samples randomly.
  - Say the kth set $\mathcal{S}_k$ and the others $\mathcal{S}_{-k}$ for any $k \leq K$.
  - Fit a model by using $\mathcal{S}_{-k}$ and then calculate the test error by using $\mathcal{S}_k$ for all $k \leq K$.

  - Calculate the cross-validation error (CV error) defined as the mean of test errors we calculate above.

- The cross-validation is simple and intuitive but
  - often suffers from computational burden.
  - depends on how to split the samples.
- When K is the sample size we say leave-one-out or one-against-all error.
- R example

```
##### cross-validation error calculation, diabetes samples
rm(list=ls()); library(lars); data(diabetes)

### whole samples
y.vec <- diabetes$y; x.mat <- diabetes$x; n.num <- length(y.vec)

### randomly select fold positions
f.num <- 10                # number of folds
pos.vec <- sample(1:n.num) # re-order the samples randomly
f.list <- split(pos.vec,1:f.num)

### do cross validation process
cv.err.vec <- rep(0,f.num)
for(f.pos in 1:f.num){
    ts.pos <- f.list[[f.pos]]

    tr.df <- as.data.frame(cbind(y.vec[-ts.pos],x.mat[-ts.pos,]))
    colnames(tr.df) <- c("diab",colnames(x.mat))

    tr.fit <- lm(diab~.,data=tr.df)

    ts.df <- as.data.frame(cbind(y.vec[ts.pos],x.mat[ts.pos,]))
    colnames(ts.df) <- c("diab",colnames(x.mat))

    ts.y.hat.vec <- predict(tr.fit,newdata=ts.df)  # prediction of y.vec
    ts.err <- sum((y.vec[ts.pos]-ts.y.hat.vec)^2)
    cv.err.vec[f.pos] <- ts.err
}
cv.err.vec
cv.err <- sum(cv.err.vec)/n.num
cv.err
```

3.5.3 Training-test randomization

- The independent test and cross validation error depend on how to split the samples.
- To avoid this kind of uncertainty, we often do randomization experiments as follows.
  - Randomly split the samples into training and test samples.
  - Calculate the training-test error.
  - Repeat above steps sufficiently many times.

  - Calculate the averages of the training-test errors.

- R examples

```
##### independent training-test randomization error calculation, diabetes samples
rm(list=ls()); library(lars); data(diabetes)

### whole samples
y.vec <- diabetes$y; x.mat <- diabetes$x; n.num <- length(y.vec)

### randomization

s.num <- 500  # number of simulations
r.err.vec <- rep(0,s.num)

for(s.pos in 1:s.num){
    print(s.pos)

    ### randomly select training sample positions
    tr.n.num <- round(n.num/2) # number of training samples
    pos.vec <- sample(1:n.num) # re-order the samples randomly
    pos.vec <- pos.vec[1:tr.n.num]

    ### training samples
    tr.y.vec <- y.vec[pos.vec]
    tr.x.mat <- x.mat[pos.vec,]
    tr.df <- as.data.frame(cbind(tr.y.vec,tr.x.mat))
    colnames(tr.df) <- c("diab",colnames(tr.x.mat))
    tr.df[1:10,]

    ### test samples
    ts.y.vec <- y.vec[-pos.vec]
    ts.x.mat <- x.mat[-pos.vec,]
    ts.df <- as.data.frame(cbind(ts.y.vec,ts.x.mat))
    colnames(ts.df) <- c("diab",colnames(ts.x.mat))
    ts.df[1:10,]

    ### fit a model by using the training samples
    tr.fit <- lm(diab~.,data=tr.df)
    tr.fit$coef

    ### calculate test error (prediction error) by using the test samples
    ts.y.hat.vec <- predict(tr.fit,newdata=ts.df)  # prediction of y.vec
    ts.err <- mean((ts.y.vec-ts.y.hat.vec)^2)
    r.err.vec[s.pos] <- ts.err
}
r.err.vec[1:10]
r.err <- mean(r.err.vec)
```

3.5.4 Others

- Assume that the full model has k input variables then, for the sub-model that has p input variables,

- $R^2(p) = SSE(k)/SSR(p)$
- $R^2_a(p) = 1 - \{(n-1)/(n-p-1)\}(1-R(p)^2)$
- $C_p = SSE(p)/\sigma^2 - n + 2p$, where $\sigma^2 = Var(\varepsilon)$ is known, by Mallows, C. L. (1973)
  - If σ^2 is unknown, $C_p = SSE(p)/MSE(k) - n + 2p$.
- $\bar{C}_p = C_p - 2(k-p+1)/(n-k-3)$, for unknown σ^2 by Gilmour, S. G. (1996)

3.6 Constructing candidate models

- Let $\mathcal{M}$ be a set of candidate models.
  - Once a model comparison measure is given we can use it to compare models in $\mathcal{M}$!
  - For example, we can construct $\mathcal{M}$ by using the best subset selection (BSS) methods together with the BIC.
  - The key issue is whether we can construct an $\mathcal{M}$ that includes the true subset $\mathcal{S}^*$ with a high probability, that is,
    $$\mathbf{P}(\mathcal{M} \supset \mathcal{S}^*) \to 1.$$
  - It is easy to see that constructing such a nice $\mathcal{M}$ becomes harder as p gets large or n does small.

3.6.1 Best subset selection

- The best subset selection (BSS) methods are the most popular in practice but there is no theoretical back-up on the use of them.
  - all possible search (APS)
    - the set $\mathcal{M}$ always includes the true model $\mathcal{S}^*$!
    - the most ideal but impossible to implement when p is large!
  - forward selection (FS)
  - backward elimination (BE)
  - stepwise selection (SS)
- The last three methods may be said to be heuristic since there is no known theory on the set $\mathcal{M}$ constructed by them.
- R example

```
##### forward/backward selection equipped with an IC
rm(list=ls())

### constants and parameters
n.num <- 100 # sample size
p.num <- 10  # total number of variables
q.num <- 5   # number of relevant variables
r.val <- 0.1 # correlation factor
s.val <- 3   # error standard deviation
b.vec <- seq(1,0.5,length.out=q.num)   # true coefficients
b.vec <- c(b.vec, rep(0,p.num-q.num))  # true coefficients

### sample generation
```

```
co.mat <- r.val^(abs(outer(c(1:p.num),c(1:p.num),"-")))       # covariance matrix
cco.mat <- chol(co.mat)                                       # cholesky decomposition
x.mat <- matrix(rnorm(n.num*p.num),n.num,p.num)%*%t(cco.mat)  # multivariate from univariate
y.vec <- drop(x.mat%*%b.vec+rnorm(n.num,0,s.val))
xy.df <- as.data.frame(cbind(y.vec,x.mat))
colnames(xy.df) <- c("y",paste("x",1:p.num,sep=""))

### formula/fitted line for the full and reduced model
f.mod <- paste(names(xy.df)[1],"~",paste(names(xy.df)[-1],collapse="+"))
f.fit <- lm(f.mod,data=xy.df)
r.mod <- "y~1"
r.fit <- lm(r.mod,data=xy.df)

### forward selection + AIC (k=2)
fit.aic <- step(r.fit,f.mod,direction="forward",k=2)
summary(fit.aic)

### backward selection + BIC (k=log(n))
fit.bic <- step(f.fit,r.mod,direction="backward",k=log(n.num))
summary(fit.bic)

### stepwise selection + RIC (k=2log(p))
fit.ric <- step(r.fit,f.mod,direction="both",k=2*log(p.num+1))
summary(fit.ric)
```

- Read Guyon and Elisseeff (2013)!

3.6.2 Model constraining

- Recently, the penalized estimation (PE) becomes nice alternatives to the BBS.

$$\hat{\boldsymbol{\beta}}^{\lambda} = \arg\min\{||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||^2/2n + \|\boldsymbol{\beta}\|_{J_\lambda}\}.$$

  - J_λ: a function (penalty), e.g., LASSO, SCAD, MCP, ...
  - λ: a constant that indexes a model (tuning parameter)
  - $\|\boldsymbol{\beta}\|_{J_\lambda} = \sum_{j=1}^{p} J_\lambda(|\beta_j|)$

- The PE is developed for simultaneous parameter estimation and model selection but you may think of the PE as a way of constructing an $\mathcal{M}$.
  - For example, some non-convex penalties such as SCAD and MCP produces a $\mathcal{M}$ that satisfy

$$\mathbf{P}(\mathcal{M} \subset \hat{\mathcal{S}}) \to 1.$$

* We will study on the PE later!

3.6.3 Overfit vs Underfit

3.7 R practice

3.7.1 Simulations 1 (BSS)

- Consider the model,

$$y_i = \beta_0 + \mathbf{x}_i^T\boldsymbol{\beta} + \varepsilon_i, \ i \le n, \tag{22}$$

- $\mathbf{x}_i \sim^{i.i.d} N(\boldsymbol{\mu}, \boldsymbol{\Sigma}),\ i \leq n$ with $\mu_j = 0$ and $\Sigma_{jk} = \rho^{|j-k|},\ j, k \leq p$,
- $\beta_0 = 0$ and $\beta_j = 3/j,\ j \leq p$,
- $\varepsilon_i \sim^{i.i.d.} N(0, \sigma^2),\ i \leq n$,
- $p = 10$, $n \in \{100, 500\}$, $\rho \in \{0.2, 0.8\}$ and $\sigma \in \{2, 4\}$.

- For all the combinations of between APS/FS/BE/SS and F-test/GIC/Validation-error/CV-error, construct a table that includes the average number of the followings in 300 simulations.
  - $I(\hat{\beta}_j ! = 0),\ j \leq p$
  - test errors from 3000 independent samples

3.7.2 Simulations 2 (PE)

- For the model (23), do the same things between LASSO/SCAD and F-test/GIC/Validation-error/CV-error.
- Compare the results to those of Simulation 1.

3.8 Guyon and Elisseeff (2013)

3.8.1 Questions raised

1 Do you have domain knowledge? If yes, construct a better set of ad hoc features.

2 Are your features commensurate? If no, consider normalizing them.

3 Do you suspect interdependence of features? If yes, expand your feature set by constructing conjunctive features or products of features, as much as your computer resources allow you

4 Do you need to prune the input variables (e.g. for cost, speed or data understanding reasons)? If no, construct disjunctive features or weighted sums of features (e.g. by clustering or matrix factorization).

5 Do you need to assess features individually? If yes, use a variable ranking method; else, do it anyway to get baseline results.

6 Do you need a predictor? If no, stop.

7 Do you suspect your data is dirty? If yes, detect the outlier examples using the top ranking variables obtained in step 5 as representation; check and/or discard them.

8 Do you know what to try first? If no, use a linear predictor. Use a forward selection method with the probe method as a stopping criterion or use the ‘0-norm embedded method. For comparison, following the ranking of step 5, construct a sequence of predictors of same nature using increasing subsets of features. Can you match or improve performance with a smaller subset? If yes, try a non-linear predictor with that subset.

9 Do you have new ideas, time, computational resources, and enough examples? If yes, compare several feature selection methods, including your new idea, correlation coefficients, backward selection and embedded methods. Use linear and non-linear predictors. Select the best approach with model selection.

10 Do you want a stable solution (to improve performance and/or understanding)? If yes, subsample your data and redo your analysis for several bootstraps.

3.8.2 Small but Revealing Examples

- Can Presumably Redundant Variables Help Each Other?
  - Noise reduction and consequently better class separation may be obtained by adding variables that are presumably redundant.
- How Does Correlation Impact Variable Redundancy?
  - Perfectly correlated variables are truly redundant in the sense that no additional information is gained by adding them.
  - Very high variable correlation (or anti-correlation) does not mean absence of variable complementarity.
- Can a Variable that is Useless by Itself be Useful with Others?
  - A variable that is completely useless by itself can provide a significant performance improvement when taken with others
  - Two variables that are useless by themselves can be useful together.

3.8.3 Advanced topics and open problems

1 Variance of Variable Subset Selection

2 Variable Ranking in the Context of Others

3 Unsupervised Variable Selection

4 Forward vs. Backward Selection

4 The Multi-class Problem

5 Selection of Examples

6 Inverse Problems

4 Logistic regression

4.1 Classification

- Analysis of constructing a relationship f (regression function) between a variable $y \in \{0, 1\}$ and a vector $\mathbf{x} = (x_1, \ldots, x_p)^T$ that can be used to obtain a rule ϕ for classifying y based on $\mathbf{x}$.
  - $y \in \{0, 1\}$: binary output/dependent/response variable
  - $\mathbf{x} \in \mathbb{R}^p$: input/independent/explonatory vector or covariates
  - $f : \mathbb{R}^p \to \mathbb{R}$: model/target function of interest (unknown)
  - $\phi : \mathbb{R}^p \to \{0, 1\}$: classifier or rule for determining a value of y

```
##### car accident samples

### reading samples
rm(list=ls());
car.df <- read.csv(file="D://car.sample.csv",sep=",")
str(car.df); car.df[1:5,]

### dummy variables
car.a <- car.df$car=="A"
car.b <- car.df$car=="B"
sex.w <- car.df$sex=="W"

### training and test samples
x.mat <- as.matrix(cbind(car.a,car.b,sex.w,car.df[,-(1:5)]))[1:100,]
y.vec <- car.df$acd[1:100]
nx.mat <- as.matrix(cbind(car.a,car.b,sex.w,car.df[,-(1:5)]))[-(1:100),]
ny.vec <- car.df$acd[-(1:100)]
```

- Two steps for classification
  - deciding a function family $\mathcal{F}$ (modeling)
  - specifying a function $f \in \mathcal{F}$ (estimating)
- Methods for classification
  - logistic regression, linear discriminant analysis, classification tree, support vector machine, ...

4.1.1 Classification from the linear regression

- Recall $y \in \{0, 1\}$.
- One simple way of classifying y can be constructed by the linear regression, considering y as a real number.
  - Fit the model $y = \mathbf{x}^T\boldsymbol{\beta} + \varepsilon = E(y|\mathbf{x}) + \varepsilon$ to get $\hat{\boldsymbol{\beta}}$
  - Classify new y_0 for new $\mathbf{x}_0$ by the rule

$$y_0 = \phi(\mathbf{x}_0) = I(\mathbf{x}_0^T\hat{\boldsymbol{\beta}} \geq 1/2).$$

  - Is this the best we can do?

  - Can you answer to a simple question: What is the probability $\mathbf{P}(y_0 = 1|\mathbf{x}_0)$?
  - I hope you feel uneasy since $\mathbf{x}_0^T\hat{\boldsymbol{\beta}} \in \mathbb{R}$ but $y_0 \in \{0, 1\}$.

- This can be resolved by modeling the conditional (posterior) probability instead of the conditional expectation.

$$\mathbf{P}(y = 1|\mathbf{x}) \text{ vs } E(y|\mathbf{x})$$

- R example

```
##### classification from linear regression

### linear regression
lin.fit <- lm(y.vec~.,data=as.data.frame(cbind(y.vec,x.mat)))

### estimated coefficients vector (beta)
lin.fit$coef

### expectation of y given x over training samples
e.vec <- cbind(1,x.mat)%*%lin.fit$coef
cbind(y.vec,e.vec)[1:5,]

### prediction over training samples
yh.vec <- as.numeric(e.vec>0.5)

### error table and rate over training samples
chisq.test(y.vec,yh.vec)$obs; mean(abs(y.vec-yh.vec))

### expectation of y given x over test samples
ne.vec <- cbind(1,nx.mat)%*%lin.fit$coef
cbind(ny.vec,ne.vec)[1:5,]

### prediction over test samples
nyh.vec <- as.numeric(ne.vec>0.5)

### error table and rate over test ssamples
chisq.test(ny.vec,nyh.vec)$obs; mean(abs(ny.vec-nyh.vec))
```

4.2 Logistic regression

4.2.1 Modeling and estimation

- The most popular modeling scheme is the logistic regression that uses the Bernoulli distribution, considering $y \in \{0, 1\}$ is binary.

$$y|\mathbf{x} \sim B(1, p(\mathbf{x}))$$

  - We may use other distributions. Google Probit regression and Gumbel regression.

- A usual choice of loss function for estimating $\boldsymbol{\beta}$ is the negative log probability density function.

$$L(\boldsymbol{\beta}) = -\log\left\{p(\mathbf{x})^y(1 - p(\mathbf{x}))^{1-y}\right\}$$

- Given samples $(y_i, \mathbf{x}_i), i \leq n$, the empirical risk becomes the average of the negative log-likelihood function (homework)

$$R_n(\boldsymbol{\beta}) = \sum_{i=1}^{n} \{-y_i \mathbf{x}_i^T \boldsymbol{\beta} + \log(1 + \exp(\mathbf{x}_i^T \boldsymbol{\beta}))\}/n,$$

  if we take $p(\mathbf{x}) = \exp(f(\mathbf{x}))/(1 + \exp(f(\mathbf{x})))$ (logistic function), where $f(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}$ is the regression function.

- We say the minimizer of R_n the maximum likelihood estimator (MLE)

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} R_n(\boldsymbol{\beta})$$

```
##### negative log-likelihood

log0.fun <- function(x.mat,y.vec,b.vec){
    xb.vec <- as.vector(x.mat%*%b.vec); xb.vec[xb.vec>700] <- 700
    sum(-y.vec*xb.vec + log(1+(exp(xb.vec))))/length(y.vec)
}
log0.fun(cbind(1,x.mat),y.vec,lin.fit$coef)
```

4.2.2 Newton-Rhpason algorithm

- One of efficient algorithms for minimizing $R_n(\boldsymbol{\beta})$ to obtain the minimizer $\hat{\boldsymbol{\beta}}$.

  - Initialize $\hat{\boldsymbol{\beta}}$ and then update it as follows until convergence:

$$\hat{\boldsymbol{\beta}} \leftarrow \hat{\boldsymbol{\beta}} - \{\partial^2 R_n(\boldsymbol{\beta})/\partial \boldsymbol{\beta}^2\}^{-1} \{\partial R_n(\boldsymbol{\beta})/\partial \boldsymbol{\beta}\}.$$

```
##### gradient and Hessian of negative log-likelihood

### gradient function
log1.fun <- function(x.mat,y.vec,b.vec){
    xb.vec <- as.vector(x.mat%*%b.vec); xb.vec[xb.vec>700] <- 700
    exb.vec <- exp(xb.vec); p.vec <- exb.vec/(1+exb.vec)
    as.vector(-t(x.mat)%*%(y.vec-p.vec)/length(y.vec))
}

### Hessian function
log2.fun <- function(x.mat,y.vec,b.vec){
    xb.vec <- as.vector(x.mat%*%b.vec); xb.vec[xb.vec>700] <- 700
    exb.vec <- exp(xb.vec); p.vec <- exb.vec/(1+exb.vec)
    w.vec <- p.vec*(1-p.vec)
    t(x.mat)%*%diag(w.vec)%*%x.mat/length(y.vec)
}
```

  - The solution satisfies the estimating equation

$$dR_n(\boldsymbol{\beta})/d\boldsymbol{\beta} = -\sum_{i=1}^{n} (y_i - p(\mathbf{x}_i))\mathbf{x}_i = \mathbf{0}$$

```
##### estimating equation

### logistic regression
log.fit <- glm(y.vec~.,data=as.data.frame(cbind(y.vec,x.mat)),family="binomial")
log.fit$coef
```

```
### equation check
log1.fun(cbind(1,x.mat),y.vec,log.fit$coef)
#log1.fun(cbind(1,x.mat),y.vec,lin.fit$coef)

### positive definiteness check
eigen(log2.fun(cbind(1,x.mat),y.vec,log.fit$coef))$values
```

  - This algorithm is often presented via iterative re-weighted least squares since the update step becomes

$$\hat{\boldsymbol{\beta}} \leftarrow \arg\min_{\boldsymbol{\beta}} (\hat{\mathbf{z}} - \mathbf{X}\boldsymbol{\beta}) \hat{\mathbf{W}} (\hat{\mathbf{z}} - \mathbf{X}\boldsymbol{\beta}),$$

    where $\hat{\mathbf{z}} =?$ and $\hat{\mathbf{W}} =?$ (homework) are adjusted response and weight matrix, and $\mathbf{X}$ is the design matrix.

* Google the term 'scoring method' for a stabilized version of the Newton-Rhapson algorithm.

- R example

```
##### NR algorithm for logistic regression

### NR function: x.mat must include intercept
nr.log.fun <- function(x.mat,y.vec,b.vec,iter.max=1e+4,eps=1e-7){
    for(iter in 1:iter.max){
        log.val  <- log0.fun(x.mat,y.vec,b.vec)
        print(log.val)
        grad.vec <- log1.fun(x.mat,y.vec,b.vec)
        hess.mat <- log2.fun(x.mat,y.vec,b.vec)
        nb.vec <- b.vec-as.vector(solve(hess.mat)%*%grad.vec)
        if(sum(abs(b.vec-nb.vec))<eps) break
        b.vec <- nb.vec
    }
    return(list(coef=b.vec,grad=grad.vec,hess=hess.mat))
}

### initial vector for nr.log.fun
b.vec <- lin.fit$coef*0

### coefficient vector form nr.log.fun
my.fit <- nr.log.fun(cbind(1,x.mat),y.vec,b.vec)

### coefficient vector from glm
log.fit <- glm(y.vec~.,data=as.data.frame(cbind(y.vec,x.mat)),family="binomial")

### cross check
cbind(my.fit$coef,log.fit$coef)
```

4.2.3 Classifier and classification (prediction)

- The estimated conditional (posterior) probability becomes

$$\hat{\mathbf{P}}(y = 1|\mathbf{x}) = \hat{p}(\mathbf{x}) = \exp(\mathbf{x}^T\hat{\boldsymbol{\beta}})/(1 + \exp(\mathbf{x}^T\hat{\boldsymbol{\beta}})).$$

  - And the corresponding classifier becomes

$$\hat{\phi}(\mathbf{x}) = I(\hat{p}(\mathbf{x}) > 0.5) = I[\hat{f}(\mathbf{x}) > 0] = I[\mathbf{x}^T\hat{\boldsymbol{\beta}} > 0].$$

  - Finally, the prediction or classification of y_0 based on $\mathbf{x}_0$ becomes

$$\hat{y}_0 = \hat{\phi}(\mathbf{x}_0).$$

- R example

```
##### classification from logistic regression

### logistic regression
log.fit <- glm(y.vec~.,data=as.data.frame(cbind(y.vec,x.mat)),family="binomial")

### beta
log.fit$coef

### probability of y=1 given x over training samples
f.vec <- as.numeric(cbind(1,x.mat)%*%log.fit$coef)
p.vec <- exp(f.vec)/(1+exp(f.vec))
cbind(y.vec,p.vec)[1:5,]

### prediction over training samples
yh.vec <- as.numeric(p.vec>0.5)

### error table and rate over training samples
chisq.test(y.vec,yh.vec)$obs; mean(abs(y.vec-yh.vec))

### probability of y=1 given x over test samples
nf.vec <- as.numeric(cbind(1,nx.mat)%*%log.fit$coef)
np.vec <- exp(nf.vec)/(1+exp(nf.vec))
cbind(ny.vec,np.vec)[1:5,]

### prediction over test samples
nyh.vec <- as.numeric(np.vec>0.5)

### error table and rate over test samples
chisq.test(ny.vec,nyh.vec)$obs; mean(abs(ny.vec-nyh.vec))
```

4.3 Interpretation of parameters: Odds ratio

- Odds: the ratio between the probabilities of success and failure at $\mathbf{x}$

$$O(\mathbf{x}) = \mathbf{P}(y = 1|\mathbf{x})/\mathbf{P}(y = 0|\mathbf{x}) = \exp(\mathbf{x}^T\boldsymbol{\beta})$$

```
##### odds

### odds of the first observation
x.vec <- x.mat[1,]
exp(sum(c(1,x.vec)*log.fit$coef))
```

```
### odds of the second observation
x.vec <- x.mat[2,]
exp(sum(c(1,x.vec)*log.fit$coef))

### odds of the whole observations
as.vector(cbind(1,x.mat)%*%log.fit$coef)

### odds of a new observation
x.vec <- sample(c(1,0),dim(x.mat)[2],rep=T)
exp(sum(c(1,x.vec)*log.fit$coef))
```

- Odds ratio: the ratio between the odds at $\mathbf{x}$ and $\mathbf{x}'$

$$OR(\mathbf{x}, \mathbf{x}') = O(\mathbf{x})/O(\mathbf{x}') = \exp((\mathbf{x} - \mathbf{x}')^T \boldsymbol{\beta})$$

  - For example, if we set $x'_j = x_j + 1, j = k$ and $x'_j = x_j, j \neq k$ then

$$OR(\mathbf{x}, \mathbf{x}') = \exp(\beta_k),$$

    which often translated as "the odds ratio of the kth input from the logistic regression becomes $\exp(\beta_k)$"

* Google (homework) odds ratio!

```
##### odds ratio

### odds ratio between the first and second observations
x1.vec <- x.mat[1,]
x2.vec <- x.mat[2,]
exp(sum((x1.vec-x2.vec)*log.fit$coef[-1]))

### what is the meaning of the coefficient of car.a?
log.fit$coef[2]

### what is the meaning of the coefficient of sex.m?
log.fit$coef[4]

### what is the meaning of the coefficient of x1?
log.fit$coef[5]
```

4.4 Loss functions for classification

- Let $\eta_k(\mathbf{x}) = \mathbf{P}(y = k|\mathbf{x})$ for any $k \in \mathbb{Z}$ and $\mathbf{x} \in \mathbb{R}^p$.

4.4.1 0-1 loss: $y \in \{0, 1\}$ case

- For a classifier $\phi(\mathbf{x}) \in \{0, 1\}$, the 0-1 loss is defined by

$$L(y, \phi(\mathbf{x})) = I[y \neq \phi(\mathbf{x})].$$

  - The 0-1 loss is the most intuitive choice for the classification problem.

  - (Homework) The risk becomes $R(\phi) = E(h_\phi(\mathbf{x}))$, and satisfies
$$h_\phi(\mathbf{x}) = \eta_1(\mathbf{x})I[\phi(\mathbf{x}) = 0] + \eta_0(\mathbf{x})I[\phi(\mathbf{x}) = 1] \geq h_{\hat{\phi}}(\mathbf{x})$$
for any $\mathbf{x} \in \mathbb{R}^p$ if
$$\hat{\phi}(\mathbf{x}) = \arg\max_{k\in\{0,1\}} \eta_k(\mathbf{x}).$$

- We say the classification rule $\hat{\phi}$ as Bayes rule or Bayes classifier.
  - The Bayes rule is the most ideal or intuitive rule for the classification problem.
  - We classify y into the class for which the conditional (posterior) probability is the largest.
$$\mathbf{P}(y = 1|\mathbf{x}) \geq \mathbf{P}(y = 0|\mathbf{x}) \Leftrightarrow \hat{y} = 1$$
  - We classify y into 1 if the conditional (posterior) probability is larger than 0.5.
$$\mathbf{P}(y = 1|\mathbf{x}) \geq 0.5 \Leftrightarrow \hat{y} = 1$$
  * We call the risk $R(\phi) = \mathbf{P}(y \neq \phi(\mathbf{x}))$ the expected misclassification rate.
    * $R(\hat{\phi})$: Bayes risk

- We can have similar result when $y \in \{-1, 1\}$ such that the classifier becomes
$$\hat{\phi}(\mathbf{x}) = \arg\max_{k\in\{-1,1\}} \eta_k(\mathbf{x}).$$

4.4.2 Logistic loss: $y \in \{0, 1\}$ case

- For a regression function $f(\mathbf{x})$, the logistic loss is defined by
$$L(y, f(\mathbf{x})) = -yf(\mathbf{x}) + \log\big(1 + \exp(f(\mathbf{x}))\big).$$
  - As we see above, it is the same as the negative log-likelihood under Bernoulli distribution assumption.
  - (Homework) The risk becomes $R(f) = EL(y, f(\mathbf{x})) = E(h_f(\mathbf{x}))$, where
$$h_f(\mathbf{x}) = E(-yf(\mathbf{x}) + \log\big(1 + \exp(f(\mathbf{x}))\big)|\mathbf{x}).$$
  - (Homework) It is easy to see that
$$h_f(\mathbf{x}) = \log\big(1 + \exp(f(\mathbf{x}))\big) - f(\mathbf{x})\eta_1(\mathbf{x}) \geq h_{\hat{f}}(\mathbf{x}),$$
for any $\mathbf{x} \in \mathbb{R}^p$ if
$$\hat{f}(\mathbf{x}) = \log(\eta_1(\mathbf{x})/\eta_0(\mathbf{x})).$$
  - (Homework) The classifier can be defined by $\hat{\phi}(\mathbf{x}) = I[\hat{f}(\mathbf{x}) > 0]$.

4.4.3 Logistic loss: $y \in \{-1, 1\}$ case

- For a function $f(\mathbf{x})$, the logistic loss is defined by
$$L(y, f(\mathbf{x})) = \log\big(1 + \exp(-yf(\mathbf{x}))\big).$$
  - (Homework) The risk becomes $R(f) = E(h_f(\mathbf{x}))$ and satisfies
$$\begin{aligned} h_f(\mathbf{x}) &= E\big(\log\big(1 + \exp(-yf(\mathbf{x}))\big)|\mathbf{x}\big) \\ &= \textstyle\sum_{k\in\{-1,1\}} \eta_k(\mathbf{x}) \log\{1 + \exp(-kf(\mathbf{x}))\} \geq h_{\hat{f}}(\mathbf{x}) \end{aligned}$$
for any $\mathbf{x} \in \mathbb{R}^p$ if
$$\hat{f}(\mathbf{x}) = \log\big(\eta_1(\mathbf{x})/\eta_{-1}(\mathbf{x})\big).$$
  * (Homework) The classifier can be defined by $\hat{\phi}(\mathbf{x}) = \text{sign}(\hat{f}(\mathbf{x}))$.

4.4.4 Fisher consistency of loss functions: $y \in \{-1, 1\}$ case

- Note the classifier from the logistic loss satisfies

$$\hat{\phi}^{Logistic} = \hat{\phi}^{Bayes}.$$

  - We say that a loss function L is Fisher consistent if the minimizer $\hat{f}$ of the risk $R(f) = EL(f(X), Y)$ leads to a classifier $\hat{\phi}$ that is the same as $\hat{\phi}^{Bayes}$.
    - In this sense, we call a loss function that is Fisher consistent a surrogate of 0-1 loss.

- Other surrogate loss functions:
  - exponential: $L(y, f(\mathbf{x})) = \exp(-yf(\mathbf{x}))$
  - square: $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2 = (1 - yf(\mathbf{x}))^2$
  - hinge: $L(y, f(\mathbf{x})) = \max\{0, 1 - yf(\mathbf{x})\}$ (support vector machine)
  - ψ: $L(y, f(\mathbf{x})) = I[yf(\mathbf{x}) \leq 0] + \max\{0, 1 - yf(\mathbf{x})\}I[yf(\mathbf{x}) > 0]$
  - * Plot a graph that shows $L(y, f(\mathbf{x}))$ vs $yf(\mathbf{x})$!

```
##### various surrogate functions of zero-one loss

### plotting
x.vec <- seq(-2,2,length.out=100)
plot(x.vec,log(1+exp(-x.vec)),ylim=c(-0.1,2),type="n")
lines(x.vec,log(1+exp(-x.vec)),col=1,lty=1,lwd=2)
lines(x.vec,exp(-x.vec)       ,col=2,lty=2,lwd=2)
lines(x.vec,(1-x.vec)^2       ,col=3,lty=3,lwd=2)
lines(x.vec,x.vec<0           ,col=4,lty=4,lwd=2)
lines(x.vec,pmax(1-x.vec,0)   ,col=5,lty=5,lwd=2)
lines(x.vec,(x.vec<0)+(x.vec>0)*pmax(1-x.vec,0),col=6,lty=6,lwd=2)
```

4.5 Unbalanced data and controlled sampling

- unbalanced data
  - Unbalanced data often shows $n_1 \ll n_0$, if we use random sampling.
  - It is hard to construct a nice classifier (lack of samples).
  - Classifiers based on random sampling are often bad or unrealistic.
  - E.g., in fraud detection problems, almost all the samples are normal cases.

- controlled sampling
  - Controlled sampling is a way of making random samples from cases ($y = 1$) and from non-cases ($y = 0$), respectively
  - e.g., case-control study
  - Note, sample distribution is differ from the population distribution.

4.5.1 Bayes classifier for controlled sampling

- For samples from controlled sampling, the Bayes classifier is not a good choice and we often use

$$\hat{\phi}_c^{Bayes}(\mathbf{x}) = I[\eta_1(\mathbf{x}) > c]$$

for some $0 < c < 1$ considering the size of n_1 and n_2.

- **Theorem 1** Assume that we have n_1 samples from cases ($y = 1$) and n_0 samples from non-cases ($y = 0$) by using the controlled sampling. Given $\mathbf{x} \in \mathbb{R}^p$, let $\mathbf{P}^o(y = 1|\mathbf{x})$ be the posterior probabilities based on the controlled samples. Then the Bayes classifier becomes

$$\hat{\phi}^{Bayes}(\mathbf{x}) = I\left[\mathbf{P}^o(y = 1|\mathbf{x}) > n_1\pi_0/(n_0\pi_1 + n_1\pi_0)\right],$$

where $\pi_1 = \mathbf{P}(y = 1)$ and $\pi_0 = \mathbf{P}(y = 0)$ are prior probabilities.

  - Theorem 1 shows how to construct the Bayes classifier from the controlled sampling when π_0 and π_1 are known.
  - Often, π_1 and π_0 are unknown (or hard to know) and it itself can be a main problem.
  - Hence, we cannot use the Bayes classifier.
  - This implies the 0-1 loss is may not the best in practice.

- **Theorem 2** Let the conditional population risk of a classifier ϕ given $\mathbf{x}$ be

$$R_{\mathbf{x}}(\phi) = E_{\mathbf{x}}\Big[L_1\eta_1(\mathbf{x})I[\phi(\mathbf{x}) = 0] + L_0\eta_0(\mathbf{x})I[\phi(\mathbf{x}) = 1]\Big],$$

where $L_k, k \in \{0, 1\}$ are given incorrect classification losses. Then the optimal classifier is obtained with $c = L_0/(L_0 + L_1)$ such that

$$\tilde{\phi}(\mathbf{x}) = I(\eta_1(\mathbf{x}) > c).$$

  - We can see that we need to know L_0 and L_1 to obtain an optimal c.
  - Instead, we try to calculate various measures for determining c.

4.6 Classification assessment measure

- Assume that we have a classifier $\phi_c(\mathbf{x}) = I(\eta_1(\mathbf{x}) > c)$ for some $0 < c < 1$.

4.6.1 Usual measures

- training and test prediction errors and information criteria
  - training/test 0-1 error: $\sum_{(y_i,\mathbf{x}_i)\in \text{ training/test set}} I[y_i = \phi_c(\mathbf{x}_i)]$

```
##### prediction errors

### logistic regression based on the first 8 variables
rx.mat <- x.mat[,1:8]
rnx.mat <- nx.mat[,1:8]
rlog.fit <- glm(y.vec~.,data=as.data.frame(cbind(y.vec,rx.mat)),family="binomial")

### 0-1 error/full model/c=0.5
f.vec <- as.vector(cbind(1,x.mat)%*%log.fit$coef)
p.vec <- exp(f.vec)/(1+exp(f.vec))
yh.vec <- as.numeric(p.vec>0.5)
```

```
chisq.test(y.vec,yh.vec)$obs
mean(abs(y.vec-yh.vec))

nf.vec <- as.vector(cbind(1,nx.mat)%*%log.fit$coef)
np.vec <- exp(nf.vec)/(1+exp(nf.vec))
nyh.vec <- as.numeric(np.vec>0.5)
chisq.test(ny.vec,nyh.vec)$obs
mean(abs(ny.vec-nyh.vec))

### 0-1 error/reduced model/c=0.5
rf.vec <- as.vector(cbind(1,rx.mat)%*%rlog.fit$coef)
rp.vec <- exp(rf.vec)/(1+exp(rf.vec))
ryh.vec <- as.numeric(rp.vec>0.5)
chisq.test(y.vec,ryh.vec)$obs
mean(abs(y.vec-ryh.vec))

rnf.vec <- as.vector(cbind(1,rnx.mat)%*%rlog.fit$coef)
rnp.vec <- exp(rnf.vec)/(1+exp(rnf.vec))
rnyh.vec <- as.numeric(rnp.vec>0.5)
chisq.test(ny.vec,rnyh.vec)$obs
mean(abs(ny.vec-rnyh.vec))

### 0-1 error/full model/c=0.4
f.vec <- as.vector(cbind(1,x.mat)%*%log.fit$coef)
p.vec <- exp(f.vec)/(1+exp(f.vec))
yh.vec <- as.numeric(p.vec>0.4)
chisq.test(y.vec,yh.vec)$obs
mean(abs(y.vec-yh.vec))

nf.vec <- as.vector(cbind(1,nx.mat)%*%log.fit$coef)
np.vec <- exp(nf.vec)/(1+exp(nf.vec))
nyh.vec <- as.numeric(np.vec>0.4)
chisq.test(ny.vec,nyh.vec)$obs
mean(abs(ny.vec-nyh.vec))

### 0-1 error/reduced model/c=0.4
rf.vec <- as.vector(cbind(1,rx.mat)%*%rlog.fit$coef)
rp.vec <- exp(rf.vec)/(1+exp(rf.vec))
ryh.vec <- as.numeric(rp.vec>0.4)
chisq.test(y.vec,ryh.vec)$obs
mean(abs(y.vec-ryh.vec))

rnf.vec <- as.vector(cbind(1,rnx.mat)%*%rlog.fit$coef)
rnp.vec <- exp(rnf.vec)/(1+exp(rnf.vec))
rnyh.vec <- as.numeric(rnp.vec>0.4)
chisq.test(ny.vec,rnyh.vec)$obs
mean(abs(ny.vec-rnyh.vec))
```

– training/test loss: $\sum_{(y_i,\mathbf{x}_i)\in \text{ training/test set}} L(y_i, f(\mathbf{x}_i))$

  * Training/test loss is invariant to the choice of c!

```
##### prediction errors
```

```
### negative log-likelihood (loss) of the full model
log0.fun(cbind(1, x.mat), y.vec,log.fit$coef)
log0.fun(cbind(1,nx.mat),ny.vec,log.fit$coef)

### negative log-likelihood (loss) of the reduced model
log0.fun(cbind(1, rx.mat), y.vec,rlog.fit$coef)
log0.fun(cbind(1,rnx.mat),ny.vec,rlog.fit$coef)
```

- generalized information criterion (GIC) indexed by a λ

$$\text{GIC}_\lambda(\mathcal{S}) = -2nR_n(\hat{\boldsymbol{\beta}}_{\mathcal{S}}) + \lambda|\mathcal{S}|$$

  - $\mathbf{X}_{\mathcal{S}}$: sub-matrix consists of columns $\mathbf{X}_j, j \in \mathcal{S} \subset \{1, \ldots, p\}$
  - $\hat{\boldsymbol{\beta}}_{\mathcal{S}} = \arg\min_{\beta_j=0: j\in\mathcal{S}^c} R_n(\boldsymbol{\beta})$: maximum likelihood estimator based on $\mathbf{X}_{\mathcal{S}}$
  - $|\mathcal{S}|$: the cardinality of $\mathcal{S}$
  - λ: a predefined positive sequence of the sample size n and model size p
  - GIC is invariant to the choice of c!

```
##### AIC and BIC for the logistic regression

### constants
n.num <- length(y.vec)
p.num <- dim(x.mat)[2]+1
rp.num <- dim(rx.mat)[2]+1

### AIC and BIC of the full model
2*n.num*log0.fun(cbind(1,x.mat),y.vec,log.fit$coef)+2*p.num
2*n.num*log0.fun(cbind(1,x.mat),y.vec,log.fit$coef)+log(n.num)*p.num

### AIC and BIC of the reduced model
2*n.num*log0.fun(cbind(1,rx.mat),y.vec,rlog.fit$coef)+2*rp.num
2*n.num*log0.fun(cbind(1,rx.mat),y.vec,rlog.fit$coef)+log(n.num)*rp.num
```

4.6.2 Sensitivity and specificity

- When the classification methods give the estimated conditional probabilities we can use various probabilities.
  - probability of correct classification: $\text{Cor}_{\phi_c} = \mathbf{P}(\phi_c(\mathbf{x}) = y)$
  - probability of incorrect classification: $\text{Inc}_{\phi_c} = \mathbf{P}(\phi_c(\mathbf{x}) \neq y)$
  - sensitivity: $\text{Sen}_{\phi_c} = \mathbf{P}(\phi_c(\mathbf{x}) = 1|y = 1)$
  - specificity: $\text{Spc}_{\phi_c} = \mathbf{P}(\phi_c(\mathbf{x}) = 0|y = 0)$
- We can use training or test samples to estimate the probabilities as follows.
  - $\hat{\mathbf{P}}(\phi_c(\mathbf{x}) = y) = (n_{00} + n_{11})/n$,
  - $\hat{\mathbf{P}}(\phi_c(\mathbf{x}) \neq y) = (n_{01} + n_{10})/n$,
  - $\hat{\mathbf{P}}(\phi_c(\mathbf{x}) = 1|y = 1) = n_{11}/n_1$,
  - $\hat{\mathbf{P}}(\phi_c(\mathbf{x}) = 0|y = 0) = n_{00}/n_0$,

```
##### probability measures

### training samples/c=0.4/reduced model
mean(y.vec==ryh.vec)
mean(y.vec!=ryh.vec)
mean(ryh.vec[y.vec==1])
1-mean(ryh.vec[y.vec==0])

### test samples/c=0.4/reduced model
mean(ny.vec==rnyh.vec)
mean(ny.vec!=rnyh.vec)
mean(rnyh.vec[ny.vec==1])
1-mean(ryh.vec[ny.vec==0])
```

where $n = n_1 + n_0$, $n_1 = n_{11} + n_{10}$, $n_0 = n_{00} + n_{01}$ and n_{st} is the number of samples that satisfy $y = s$ and $\phi_c(\mathbf{x}) = t$ for $s, t \in \{0, 1\}$.

4.6.3 ROC and AUC

- Receiver operating characteristic (ROC) curve
  - The ROC shows a functional relationship between 1-Spc$_{\phi_c}$ (false positivity) and Sen$_{\phi_c}$ (true positivity) for all $0 \leq c \leq 1$.
  - It shows how the classifier work as c varies.
- R example

```
##### ROC plot for the reduced and full model

### R package "AUC"
library(AUC); par(mfrow=c(2,2))
plot(roc(p.vec,as.factor(y.vec)))
plot(roc(p.vec,as.factor(ny.vec)))
plot(roc(rp.vec,as.factor(y.vec)))
plot(roc(rp.vec,as.factor(ny.vec)))

auc(roc(p.vec,as.factor(y.vec)))
auc(roc(p.vec,as.factor(ny.vec)))
auc(roc(rp.vec,as.factor(y.vec)))
auc(roc(rp.vec,as.factor(ny.vec)))

### ROC function
roc.fun <- function(p.vec,y.vec,c.num){
    c.vec <- seq(0,1,length.out=c.num)
    fp.vec <- tp.vec <- rep(0,c.num)
    for(c.pos in 1:c.num){
        y.hat.vec <- p.vec>=c.vec[c.pos]
        fp.vec[c.pos] <- mean(y.hat.vec[y.vec==0])
        tp.vec[c.pos] <- mean(y.hat.vec[y.vec==1])
    }
    plot(fp.vec,tp.vec,type="l")
    return(list(tp.vec=tp.vec,fp.vec=fp.vec))
```

```
}

### ROC from the full/reduced model over training/test samples
roc.fun(p.vec,y.vec,100)
roc.fun(p.vec,ny.vec,100)
roc.fun(rp.vec,y.vec,100)
roc.fun(rp.vec,ny.vec,100)
```

4.6.4 Rift chart

- (Homework)

4.7 Perfect fit problem

- Assume there are $n = 2$ samples $(y_1, x_1) = (1, a)$ and $(y_2, x_2) = (0, -a)$, where $a > 0$.
  - For the samples, we fit a simple logistic regression model $f(x) = \beta x$, by minimizing (homework)

$$R_n(\beta) = -a\beta + \log(1 + \exp(a\beta)) + \log(1 + \exp(-a\beta)).$$

  - It is easy to see that (homework) $R_n(\beta)$ is a decreasing function of β so that the minimizer of $R_n(\beta)$ is $\hat{\beta} = \infty$.

```
rn.fun <- function(a,beta){
    exp.ab <- exp(a*beta); -a*beta+log(1+exp.ab)+log(1+1/exp.ab) }
a <- 1; beta <- seq(-10,10,length.out=1e+3)
plot(beta,rn.fun(a,beta))
```

  - Note the training error is zero (and hence called perfect fit) since

$$\hat{y}_1 = \hat{\phi}(a) = I[a\hat{\beta} > 0] = 1 = y_1,$$
$$\hat{y}_2 = \hat{\phi}(-a) = I[-a\hat{\beta} > 0] = 0 = y_2.$$

  - In fact, (homework) any $\hat{\beta} > 0$ produces the same classification results!

- Perfect fit problem can happen in general when the samples can be perfectly separated by a regression function $f(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\beta}$!
  - Note the (i, i) entry of the Hessian matrix $d^2R_n(\boldsymbol{\beta})/d\boldsymbol{\beta}^2$ is proportional to $p(\mathbf{x}_i)(1-p(\mathbf{x}_i)) = 0$ for any $i \leq n$, which implies the Hessian is semi-definite!
  - When a model fits the samples perfectly, we should be careful to use it, and according to my experience, the model do not give nice results in data analysis.
  - To avoid perfect fit, one may try variable selection or regularized methods such as the ridge and LASSO.

- R example

```
rm(list=ls())
n.num <- 30
x.mat <- matrix(runif(3*n.num,-1,1),nrow=n.num)
y.vec <- x.mat[,1]>0
log.fit <- glm(y.vec~.,data=as.data.frame(cbind(y.vec,x.mat)),family="binomial")
plot(x.mat[,1],y.vec)
```

4.8 R practice

4.8.1 Simulation 1

- Consider the model below.

$$y \sim B(1, p(\mathbf{x})), \tag{23}$$

  where $p(\mathbf{x}) = \exp(f(\mathbf{x}))/(1 + \exp(\mathbf{x}))$ and $f(\mathbf{x}) = \beta_0 + \mathbf{x}^T\boldsymbol{\beta}$.

- Do the followings.
  - Generate training and test samples of sizes $n = 100, 1000$, respectively, from the model when $\mathbf{x} \in \mathbb{R}^p \sim N(\mathbf{0}_p, \mathbf{I}_{p\times p})$ and $\beta_j = 1/2j, j \leq p$ for $p = 5$.
  - Estimate the model $\hat{f}$ from the training samples and construct the classifier $\hat{\phi}$ by using $\hat{f}$.
  - Calculate the followings by using training samples.
    - posterior probabilities
    - 0-1 error and loss
    - AIC and BIC
    - sensitivity and specificity
  - Construct 10 bins for lift chart.
    - Draw lift chart including %R, %CR and Lift
  - Construct the followings by using training samples.
    - Draw ROC and calculate AUC.
  * Do the same things by using test samples.