Lab 4: Linear Regressions

Hao Wang

February 12, 2017

Linear Regression

Definition

Linear regression attempts to model the relationship by fitting a linear equation to observed data. One variable is considered to be a dependent variable, and the others are considered to be explanatory variables. Linear regression with n explanatory variables have n + 1 parameters (with intercept).

$$\hat{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...\beta_n x_n$$

Or in the Matrix format

$$\hat{Y} = XB$$

Where the X matrix include the column $(1, x_{1i}, x_{2i}...)$ The B matrix is the parameter matrix

Variations of Linear Regression

Linear regression can take multiple formats:

1. Regression through the origin
$$\hat{y}_i = \beta_1 x_1 + \beta_2 x_2 + ...\beta_n x_n$$
code:

```
library(car)
mydata <- Prestige
#help("Prestige")
lm(prestige ~ income -1, data = mydata)
```

```
##
## Call:
## lm(formula = prestige ~ income - 1, data = mydata)
##
## Coefficients:
##   income
## 0.005777
```

2. Simple linear regression
$$\hat{y}_i = \beta_0 + \beta_1 x_1$$

```
lm(prestige ~ income , data = mydata)
```

```
##
## Call:
## lm(formula = prestige ~ income, data = mydata)
##
## Coefficients:
```

```
## (Intercept)       income
##   27.141176     0.002897
```

3. Multivariate linear regression

$$\hat{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...\beta_n x_n$$

```
lm(prestige ~ income + education, data = mydata)
```

```
##
## Call:
## lm(formula = prestige ~ income + education, data = mydata)
##
## Coefficients:
## (Intercept)       income    education
##   -6.847779     0.001361     4.137444
```

4. Ploynomial regression

$$\hat{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 {x_1}^2$$

note that the identity function I( ) allows terms in the model to include normal mathematical symbols. I() is needed for the concern of collinearity.

```
lm(prestige ~ income + I(income^2), data=mydata)
```

```
##
## Call:
## lm(formula = prestige ~ income + I(income^2), data = mydata)
##
## Coefficients:
## (Intercept)       income  I(income^2)
##   1.418e+01    6.154e-03   -1.433e-07
```

5. Interaction

5.a full interaction equation

$$\hat{y}_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$$

```
lm(prestige ~ income*education, data=mydata)
```

```
##
## Call:
## lm(formula = prestige ~ income * education, data = mydata)
##
## Coefficients:
##      (Intercept)            income         education  income:education
##       -2.207e+01         3.944e-03         5.373e+00        -1.961e-04
```

5.b the interaction term only

$$\hat{y}_i = \beta_0 + \beta_3 x_1 x_2$$

```
lm(prestige ~ income:education, data=mydata)
```

```
##
## Call:
## lm(formula = prestige ~ income:education, data = mydata)
##
## Coefficients:
##      (Intercept)  income:education
##        3.105e+01         1.982e-04
```

Loss Function: Least Squares

In linear regression, we want to minimize its loss funtion, which mostly known as the least squre method

Def:

$$L = \sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

Loss funtion can appear in other ways, for instance in LASSO https://onlinecourses.science.psu.edu/stat857/node/158, least square is penelized with λ.

Calculate Linear Regression

The idea is to minimize the loss function. We can achieve this in multiple ways. +

Let's begin with simple linear regression with prestige and income

By Hand through calculus

$$L = \sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

$$\hat{y_i} = \beta_0 + \beta_1 x_i$$

Take derivatives

$$L = \sum_{i=1}^{n}(y_i - \beta_0 + \beta_1 x_i)^2$$

$$\frac{d(L)}{d(\beta_0)} = -2\sum_{i=1}^{n}(y_i - \beta_0 + \beta_1 x_i) = 0$$

$$\frac{d(L)}{d(\beta_1)} = -2\sum_{i=1}^{n}(y_i - \beta_0 + \beta_1 x_i)x_i = 0$$

solve these equations, we get

$$\beta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

Let's do that in R

```r
attach(mydata)
b1.upper <- sum(
  (education-mean(education))*
    (prestige -mean(prestige))
)

b1.lower <- sum(
  (education -mean(education))^2
)

b1 <- b1.upper / b1.lower
b0 <- mean(prestige) - b1*mean(education)
b1
```

```
## [1] 5.360878
```

```r
b0
```

```
## [1] -10.73198
```

```r
lm(prestige ~ education)
```

```
##
## Call:
## lm(formula = prestige ~ education)
##
## Coefficients:
## (Intercept)    education
##     -10.732        5.361
```

- Practice question 1: calculate prestige ~ income based on the formula above, compare your result with lm(prestige ~ income)

By Matrix

$$Y = X\beta + \epsilon$$

$$\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \cdots \\ \epsilon_n \end{bmatrix}$$

We want to minimize

$$\sum \epsilon^2 = \epsilon'\epsilon$$

$$(Y - X\beta)'(Y - X\beta)$$

Take derivatives

$$\frac{d}{d\beta}(Y - X\beta)'(Y - X\beta) = -2X'(Y - X\beta) = 0$$

Therefore

$$X'Y = X'X\beta$$

and

$$\beta = (X'X)^{-1}X'Y$$

Let's do it in R

```
X <- data.frame(1, education) # 1 is needed for intercept
X <- as.matrix(X)
y <- mydata$prestige

beta <-solve(t(X)%*%X)%*%t(X)%*%y #solve function returns inverse, t() returns transpose
beta
```

```
##                  [,1]
## X1        -10.731982
## education   5.360878
```

- Practice question: calculate linear regression prestiage ~ income + education in matrix notation

```
attach(mydata)
# X <- data.frame(1, ?, ?)
# X <- ?
# y <- ?
```

(optional) By Gradient Descent

Gradient decent is a machine learning algorithm fitting data, the main idea is to take the partial derivative of the cost function with respect to theta. That gradient, multiplied by a learning rate, becomes the update rule for the estimated values of the parameters. Iterate and things should converge nicely.

cost funtion

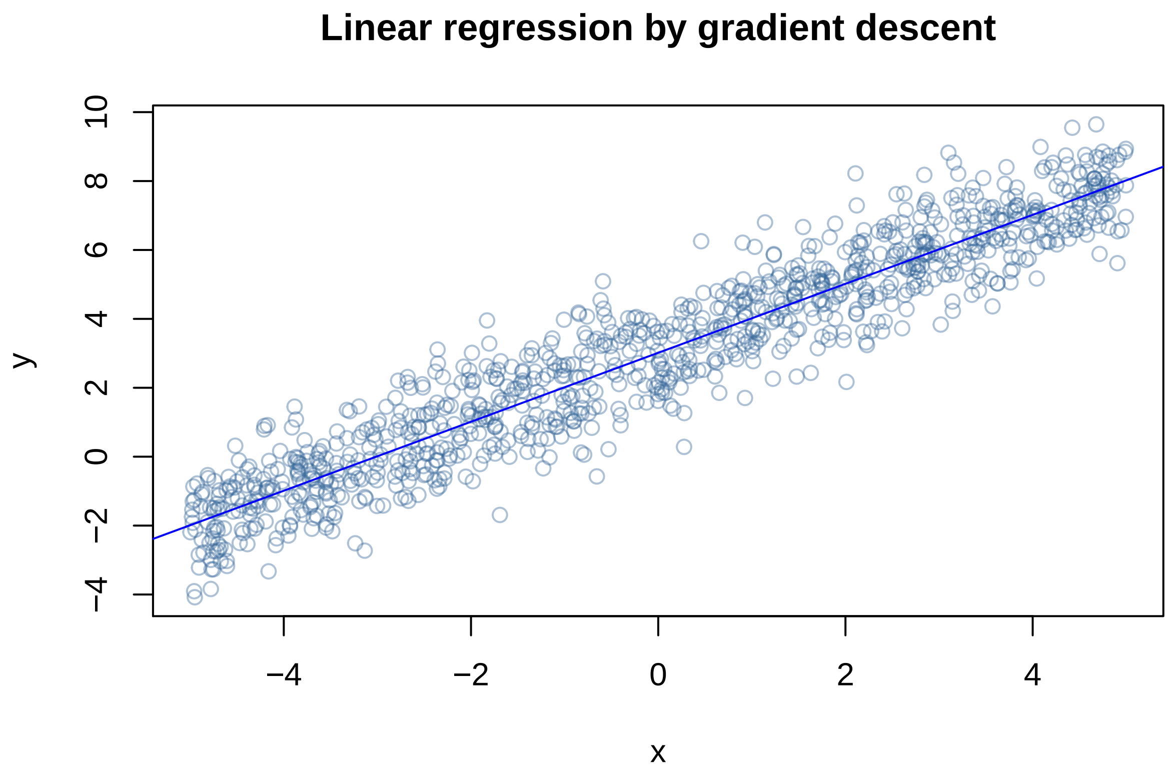
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2$$

In action, gradient descent gradually approaches optimal values for θ. How gradual depends on the learning rate, α. h_θ is the prediction funtion.

```
# generate random data in which y is a noisy function of x
x <- runif(1000, -5, 5)
y <- x + rnorm(1000) + 3

# fit a linear model
res <- lm( y ~ x )
print(res)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##       3.016        1.001
```

```
# plot the data and the model
plot(x,y, col=rgb(0.2,0.4,0.6,0.4), main='Linear regression by gradient descent')
abline(res, col='blue')
```

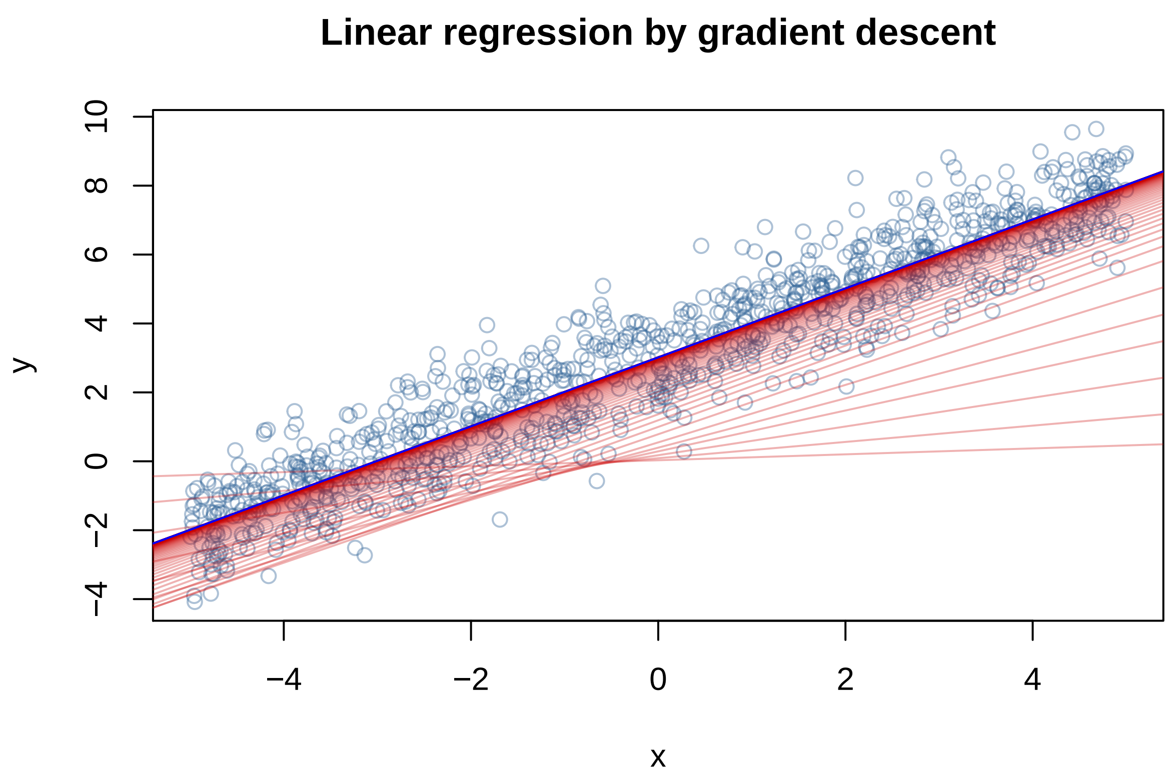



```
# squared error cost function
cost <- function(X, y, theta) {
  sum( (X %*% theta - y)^2 ) / (2*length(y))
}

# learning rate and iteration limit
alpha <- 0.01
num_iters <- 1000

# keep history
cost_history <- double(num_iters)
theta_history <- list(num_iters)

# initialize coefficients
theta <- matrix(c(0,0), nrow=2)

# add a column of 1's for the intercept coefficient
X <- cbind(1, matrix(x))

# gradient descent
for (i in 1:num_iters) {
  error <- (X %*% theta - y)
  delta <- t(X) %*% error / length(y)
  theta <- theta - alpha * delta
  cost_history[i] <- cost(X, y, theta)
  theta_history[[i]] <- theta
```

```
}

print(theta)
```

```
##          [,1]
## [1,] 3.015388
## [2,] 1.000779
```

```
# plot data and converging fit
plot(x,y, col=rgb(0.2,0.4,0.6,0.4), main='Linear regression by gradient descent')
for (i in c(1,3,6,10,14,seq(20,num_iters,by=10))) {
  abline(coef=theta_history[[i]], col=rgb(0.8,0,0,0.3))
}
abline(coef=theta, col='blue')
```



Interpret Linear Regression Results

Extract information form the summary()

```
#let's split our data into two parts first: train and test.
set.seed(99)
n <-nrow(mydata)
n1 <- floor(n/1.5) #train
n2 <- n -n1        #test
```

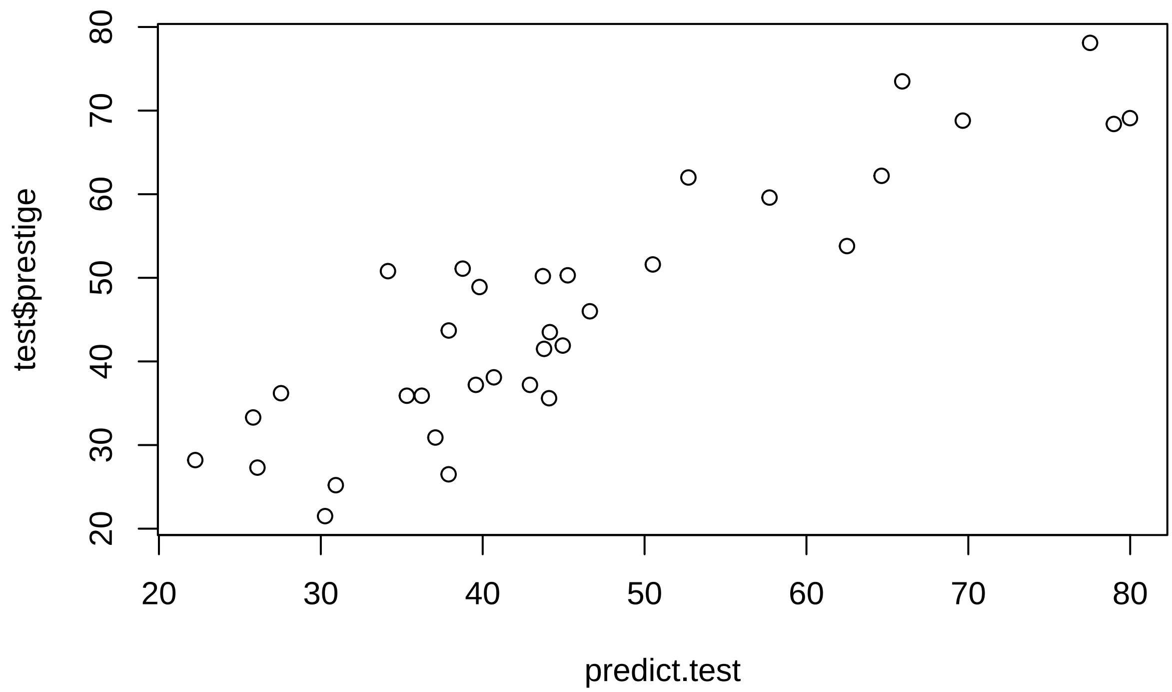
```r
ii <- sample(1:n, n) # a funtion of sample
train <- mydata[ii[1:n1],]
test  <- mydata[ii[n1+1:n2],]

lm <- lm(prestige ~ education + income, data =train)
summary(lm)
```

```
##
## Call:
## lm(formula = prestige ~ education + income, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.884  -5.253   1.166   4.688  18.741
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9.5441437  4.2205068  -2.261   0.0271 *
## education    4.3605163  0.4721306   9.236 1.91e-13 ***
## income       0.0013938  0.0003255   4.282 6.24e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.26 on 65 degrees of freedom
## Multiple R-squared:  0.8028, Adjusted R-squared:  0.7967
## F-statistic: 132.3 on 2 and 65 DF,  p-value: < 2.2e-16
```

```r
fit <- lm$fitted.values #fitted value
resid <- lm$residuals  #residuals
coef <- summary(lm)$coefficients #coefficients matrix
coef
```

```
##                 Estimate    Std. Error   t value     Pr(>|t|)
## (Intercept) -9.544143682 4.2205067702 -2.261374 2.708893e-02
## education    4.360516290 0.4721305624  9.235827 1.907554e-13
## income       0.001393831 0.0003255424  4.281566 6.242489e-05
```

```r
predict.test <- predict(lm, newdata = test) #use the previous fitted value to predict on test data.
plot(predict.test, test$prestige)
```




```
predict.point <- predict(lm, data.frame(education = 10, income=10000))
predict.point
```

```
##        1
## 47.99933
```

(Optional) Boostrap

The fundamental problem: we are fitting our regression as if we knew the population – which is not true.

- The idea: We have just one dataset. When we compute a statistic on the data, we only know that one statistic - we don't see how variable that statistic is. The bootstrap creates a large number of datasets that we might have seen and computes the statistic on each of these datasets. Thus we get a distribution of the statistic. Key is the strategy to create data that "we might have seen".
- Benefits: more consistent standard error

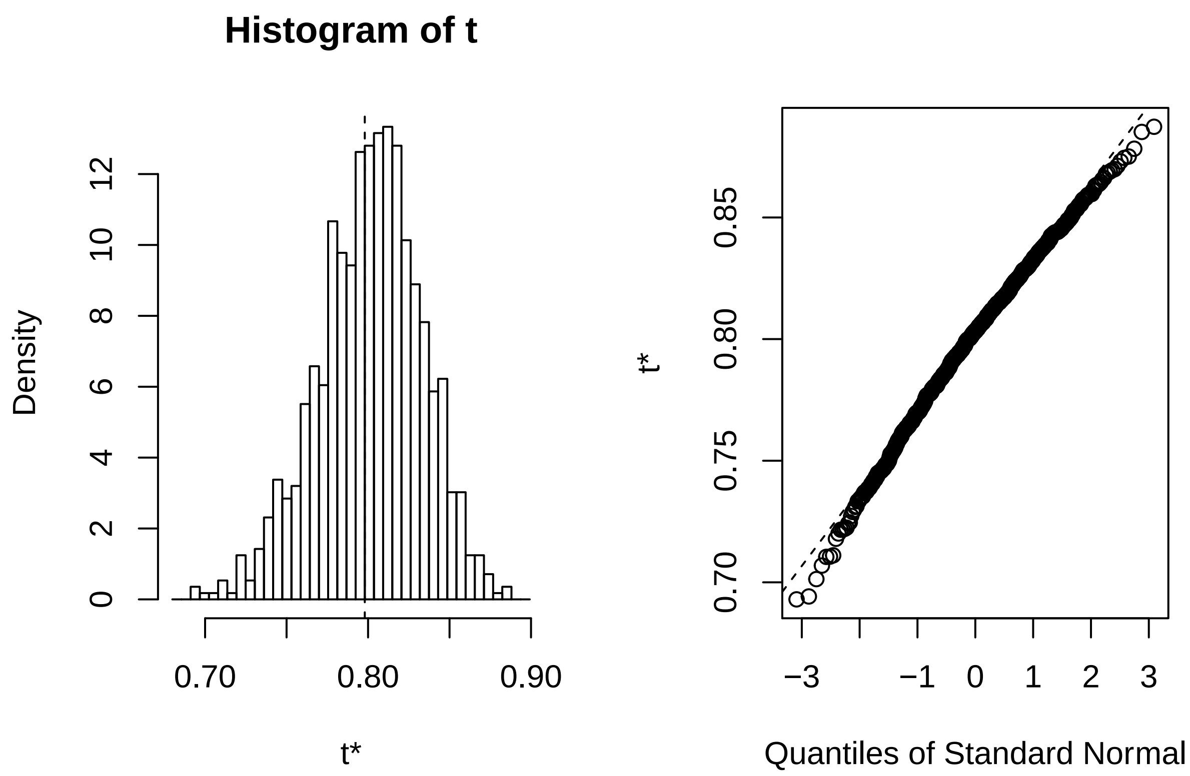
Bootstrap 95% CI for R-Squared

```
library(boot)

# Bootstrap 95% CI for R-Squared
# function to obtain R-Squared from the data
rsq <- function(formula, data, indices) {
  d <- data[indices,] # allows boot to select sample
```

```
  fit <- lm(formula, data=d)
  return(summary(fit)$r.square)
}

# bootstrapping with 1000 replications
set.seed(99)
results <- boot(data=mydata, statistic=rsq,
    R=1000, formula=prestige~ income+education)

# view results
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = mydata, statistic = rsq, R = 1000, formula = prestige ~
##     income + education)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.7980008 0.003355002  0.03154118
```

```
plot(results)
```



```r
# get 95% confidence interval
boot.ci(results, type="bca")
```

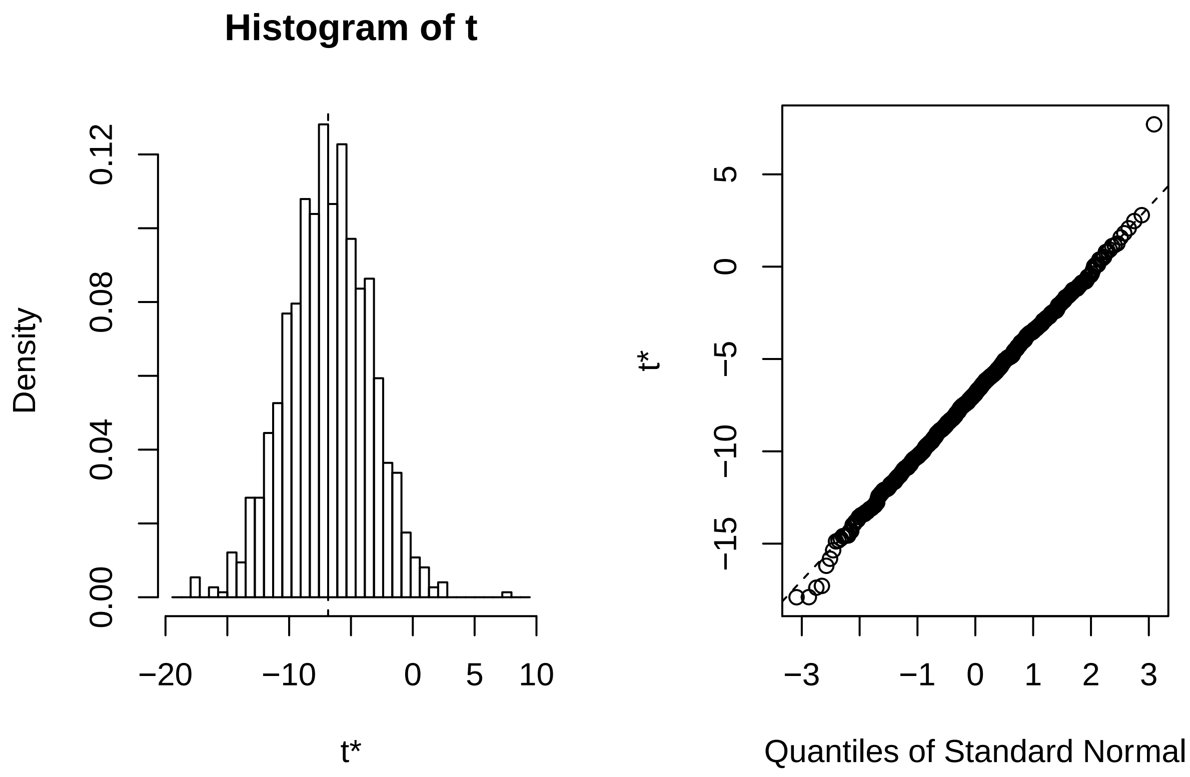
```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca")
##
## Intervals :
## Level       BCa
## 95%   ( 0.7217,  0.8490 )
## Calculations and Intervals on Original Scale
```

Bootstrap CI for regression coefficients

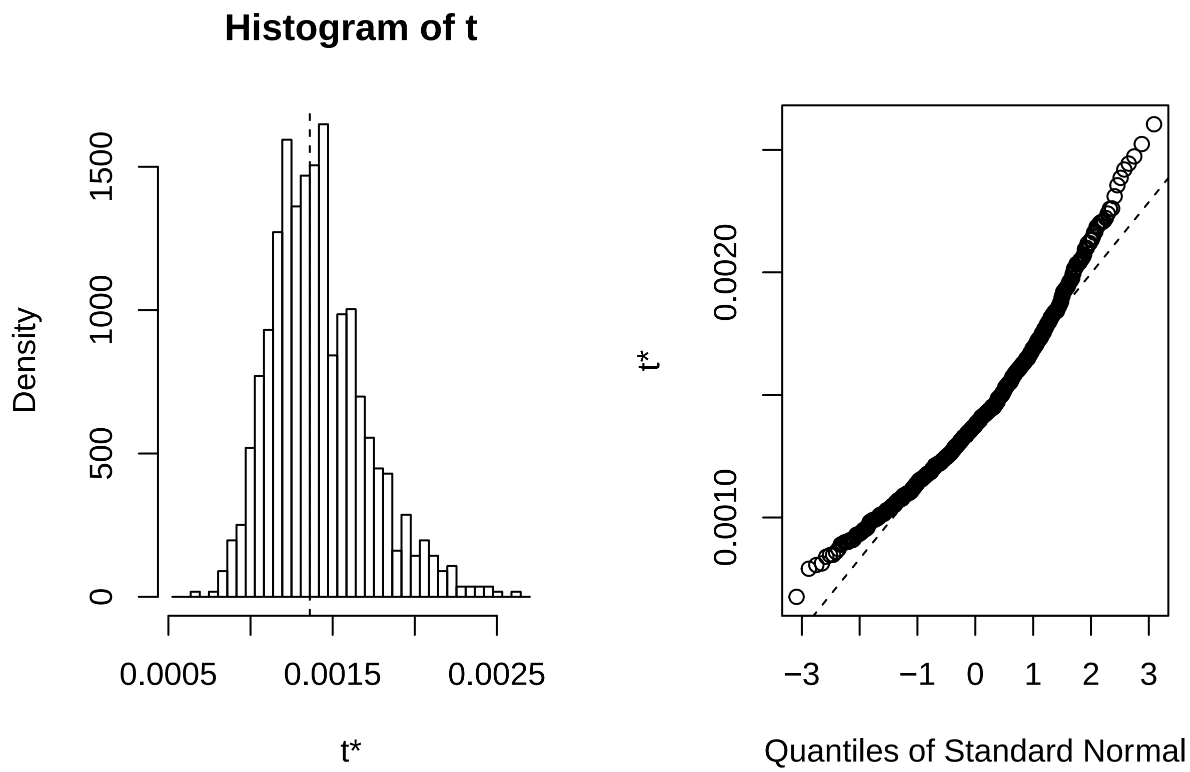
```r
# Bootstrap 95% CI for regression coefficients
# function to obtain regression weights
bs <- function(formula, data, indices) {
  d <- data[indices,] # allows boot to select sample
  fit <- lm(formula, data=d)
  return(coef(fit))
}
# bootstrapping with 1000 replications
set.seed(99)
results <- boot(data=mydata, statistic=bs,
    R=1000, formula=prestige~income+education)

# view results
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = mydata, statistic = bs, R = 1000, formula = prestige ~
##     income + education)
##
##
## Bootstrap Statistics :
##         original        bias     std. error
## t1* -6.847778720 -3.008593e-02 3.3711800814
## t2*  0.001361166  5.228186e-05 0.0002911381
## t3*  4.137444384 -2.483312e-02 0.3966468516
```

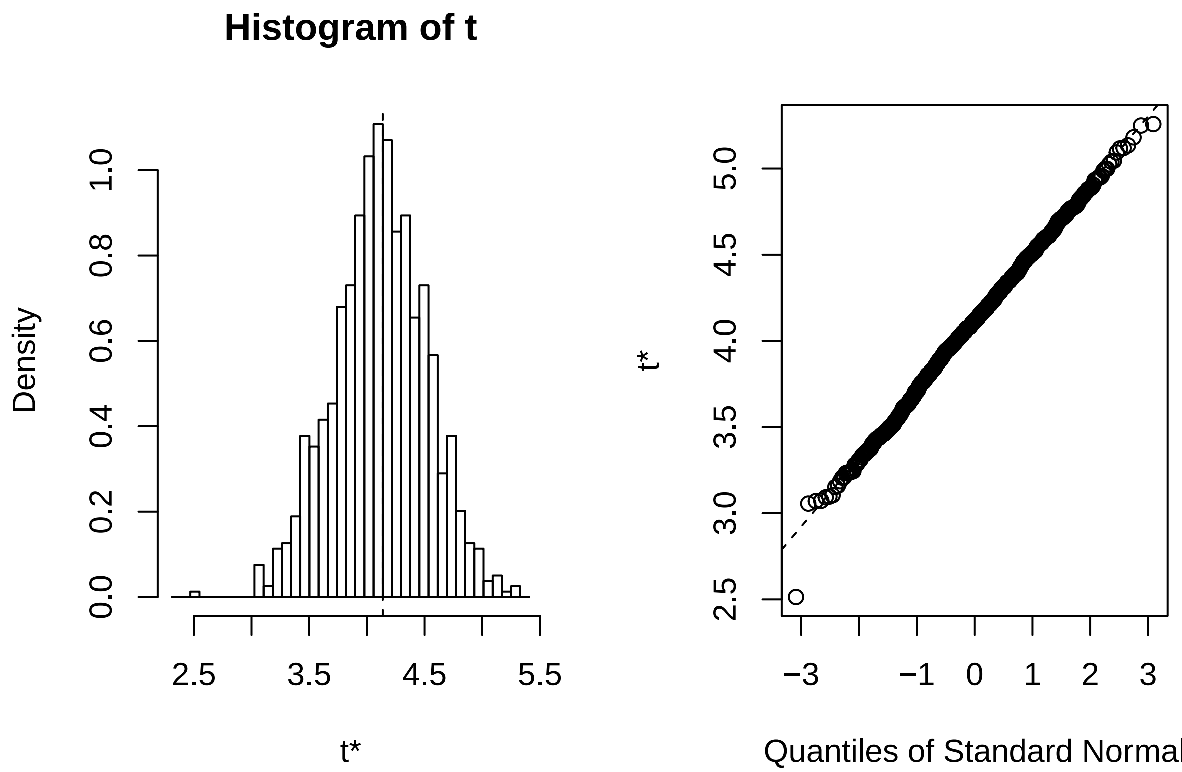
```r
plot(results, index=1) # intercept
```




```
plot(results, index=2) # income
```




```
plot(results, index=3) # education
```




```r
# get 95% confidence intervals
boot.ci(results, type="bca", index=1) # intercept
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca", index = 1)
##
## Intervals :
## Level       BCa
## 95%   (-13.374,  -0.376 )
## Calculations and Intervals on Original Scale
```

```r
boot.ci(results, type="bca", index=2) # income
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca", index = 2)
##
## Intervals :
## Level       BCa
## 95%   ( 0.0009,  0.0020 )
## Calculations and Intervals on Original Scale
```

```r
boot.ci(results, type="bca", index=3) # education
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca", index = 3)
##
## Intervals :
## Level       BCa
## 95%   ( 3.372,  4.938 )
## Calculations and Intervals on Original Scale
```

Report

'stargazer'

- **Stargazer** https://cran.r-project.org/web/packages/stargazer/vignettes/stargazer.pdf
- **stargazer cheatsheet** http://jakeruss.com/cheatsheets/stargazer.html
- **list of stats code** https://rdrr.io/cran/stargazer/

```r
lm <- lm(prestige ~income + education, data=train)
lm2 <- lm(prestige ~income + education, data=test)
```

```r
library(stargazer)
stargazer(lm, lm2, title = "Table with Stargazer", style = "apsr",  omit.stat = c("rsq", "f","ser"))
```

% Table created by stargazer v.5.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harvard.edu
% Date and time: Sun, Feb 12, 2017 - 7:29:46 PM

Table 1: Table with Stargazer

	prestige	
	(1)	(2)
income	0.001^{***}	0.001^{***}
	(0.0003)	(0.0003)
education	4.361^{***}	3.601^{***}
	(0.472)	(0.529)
Constant	-9.544^{**}	-0.731
	(4.221)	(5.010)
N	68	34
Adjusted R^2	0.797	0.785

$^{*}p < .1$; $^{**}p < .05$; $^{***}p < .01$