

ECON-613 HW #2

Min Chul Kim

2019-02-09

Exercise1

```
#Set Seed
set.seed(1)

#Drawing Samples from Distributions
n = 10000

X1 = runif(n, min = 1, max = 3)
X2 = rgamma(n, shape = 3, scale = 2)
X3 = rbinom(n, size = 1, prob = 0.3)
eps = rnorm(n, mean = 2, sd = 1)

#Creating Y and Dummy Y Variables
Y = 0.5 + 1.2*X1 - 0.9*X2 + 0.1*X3 + eps
ydum = ifelse(Y > mean(Y), 1, 0)
```

Exercise2

```
#Calculating the Correlation between Y and X1, How Different is it from 1.2
correlation = cor(Y, X1)
difference_corr = abs(cor(Y, X1) - 1.2) %>% as.data.frame
colnames(difference_corr) = "Difference between Correlation and 1.2"

kable(difference_corr, digits = 4, format = "markdown")
```

| Difference between Correlation and 1.2 |
|--|
| 0.9837 |

```
#Regression
X = data.frame(
  1,
  X1,
  X2,
  X3
) %>%
  as.matrix()

#Calculating the Regression of Y on X
reg1 = lm(Y ~ X1 + X2 + X3)

#Calculating the Coefficients of this Regression
```

```
reg1_coef = solve((t(X) %*% X)) %*% t(X) %*% Y

df_reg1_coef = reg1_coef %>% as.data.frame
colnames(df_reg1_coef) = "Coefficients of Reg"

kable(df_reg1_coef, digits = 4,
      format = "markdown", caption = "Coefficients of Regression")
```

| Coefficients of Reg | |
|---------------------|---------|
| X1.1 | 2.4919 |
| X1 | 1.1965 |
| X2 | -0.8978 |
| X3 | 0.1251 |

```
#Calculating the Standard Errors Using Standard Formula
k = length(reg1_coef)
est_var = sum( (Y - (X %*% reg1_coef))^2 ) / (n - k)
est_std = est_var * solve( (t(X) %*% X) ) %>%
  diag() %>%
  sqrt()

df_est_std = est_std %>% as.data.frame
colnames(df_est_std) = "SE from Standard Formula"

kable(df_est_std, digits = 4,
      format = "markdown", caption = "SE from Standard Formula")
```

| SE from Standard Formula | |
|--------------------------|--------|
| X1.1 | 0.0410 |
| X1 | 0.0174 |
| X2 | 0.0029 |
| X3 | 0.0221 |

```
#Bootstrap n = 49
boot_x1 = boot_x2 = boot_x3 = boot_eps = rep(NA, n)
boot_Y = list()
boot_coef = list()

for(i in 1:49){

  boot_x1 = sample(X1, n, replace = TRUE)
  boot_x2 = sample(X2, n, replace = TRUE)
  boot_x3 = sample(X3, n, replace = TRUE)

  boot_X = cbind(1, boot_x1, boot_x2, boot_x3)
  boot_Y = sample(Y, n, replace = TRUE)

  boot_coef[[i]] = solve((t(boot_X) %*% boot_X)) %*% t(boot_X) %*% boot_Y
}
```

```

#Making Bootstrap Values into a Dataframe
boot_temp_df = map(boot_coef, data.frame)
boot_coef_df = do.call("cbind", boot_temp_df)
colnames(boot_coef_df) = paste("sample", 1:49, sep = "")

#Calculating SE
bootstrap_se1 = apply(boot_coef_df, 1, FUN = sd) %>% as.data.frame()
colnames(bootstrap_se1) = "SE of Bootstrap 49"
rownames(bootstrap_se1) = c("Int", "X1", "X2", "X3")
kable(bootstrap_se1, digits = 4, format = "markdown")

```

| | SE of Bootstrap 49 |
|-----|--------------------|
| Int | 0.1227 |
| X1 | 0.0477 |
| X2 | 0.0100 |
| X3 | 0.0656 |

```

#Bootstrap n = 499
boot_x1 = boot_x2 = boot_x3 = boot_eps = rep(NA, n)
boot_Y = list()
boot_coef = list()

for(i in 1:499){

  boot_x1 = sample(X1, n, replace = TRUE)
  boot_x2 = sample(X2, n, replace = TRUE)
  boot_x3 = sample(X3, n, replace = TRUE)

  boot_X = cbind(1, boot_x1, boot_x2, boot_x3)
  boot_Y = sample(Y, n, replace = TRUE)

  boot_coef[[i]] = solve((t(boot_X) %*% boot_X)) %*% t(boot_X) %*% boot_Y
}

```

```

#Making Bootstrap Values into a Dataframe
boot_temp_df = map(boot_coef, data.frame)
boot_coef_df = do.call("cbind", boot_temp_df)
colnames(boot_coef_df) = paste("sample", 1:499, sep = "")

#Calculating SE
bootstrap_se1 = apply(boot_coef_df, 1, FUN = sd) %>% as.data.frame()
colnames(bootstrap_se1) = "SE of Bootstrap 499"
rownames(bootstrap_se1) = c("Int", "X1", "X2", "X3")
kable(bootstrap_se1, digits = 4, format = "markdown")

```

| | SE of Bootstrap 499 |
|-----|---------------------|
| Int | 0.1461 |
| X1 | 0.0610 |
| X2 | 0.0094 |
| X3 | 0.0721 |

Exercise3

In the following R chunk, I created ordinary likelihood function, as well as log likelihood function. For the ordinary likelihood function, expect 0 for the answer, since probabilities less than or equal to 1 are being multiplied for 10000 times.

```
#Creating Likelihood Function
probit_likelihood = function(x, beta, ydum){

  #Calculating XB
  X_Beta = x %*% beta

  #Calculating Phi(XB)
  phi = map(X_Beta, pnorm) %>% unlist()

  #Calculating the Likelihood Using Likelihood Formula
  prob = phi^ydum * (1 - phi)^(1 - ydum)
  likelihood = prod( prob )

  return(likelihood)
}

#The following will be 0 since probabilities <= 1 are being multiplied.
pro_df = data.frame(
  probit_likelihood(X, reg1_coef, ydum)
)
colnames(pro_df) = "Probit Likelihood"

kable(pro_df, digits = 4, format = "markdown")
```

| Probit Likelihood |
|-------------------|
| 0 |

```
#Creating Log Likelihood Function
pro_log_likelihood = function(x, beta, ydum){

  #Calculating XB
  X_Beta = x %*% beta

  #Calculating Phi(XB) and Preventing the Obtained Results from Being 0 and 1
  phi = map(X_Beta, pnorm) %>% unlist()
  phi[phi < 0.0001] = 0.0001
  phi[phi > 0.99] = 0.99

  #Calculating the Log Likelihood Using the Formula
  log_likelihood = sum( ydum * log(phi) + (1 - ydum) * log((1 - phi)) )

  return(-log_likelihood)
}

pro_log_df = data.frame(
  pro_log_likelihood(X, reg1_coef, ydum)
)
```

```
colnames(pro_log_df) = "Probit Log Likelihood"

kable(pro_log_df, digits = 4, format = "markdown")
```

| Probit Log Likelihood |
|-----------------------|
| 2519.584 |

```
#Steepest Ascent Function
steepest_ascent =
  function(x, betas_old, h, ydum_, learning_rate, epsilon){

#Initial Values (betas_new is initialized to start the while loop)
    sum_sq_betas_diff = 10
    betas_new = rep(0.5, 4)

    while( sum_sq_betas_diff > epsilon ){

      #Computing Matrices to Be Used in Derivative
      betas_old_matrix = matrix(betas_old, nrow = 4, ncol = 4)
      betas_new_matrix = betas_old_matrix + h * diag(1, nrow = 4, ncol = 4)

      #Calculating Partial Derivatives of Log Likelihood
      derivative = c()
      for(i in 1:4){
        derivative[i] = ( pro_log_likelihood(x, betas_new_matrix[i,], ydum_) -
                          pro_log_likelihood(x, betas_old_matrix[i,], ydum_) ) / h
      }

      #Updating Betas
      betas_old = betas_new
      betas_new = betas_old - learning_rate * derivative

      #Calculating sum_betas_diff
      sum_sq_betas_diff = sum((betas_new - betas_old)^2)

    }

    return(betas_new)
  }

#Implementing Steepest Ascent
betas = rep(0.5, 4)

steepest_ascent(X, betas_old = betas,
                h = 10^(-6), ydum, learning_rate = 0.000001, epsilon = .00000001)

## [1] 0.5000263 0.5000394 0.5000333 0.5000009
```

Exercise 4

```
##Writing and Optimizing Linear Probability Model
linear_reg = lm(ydum ~ X1 + X2 + X3)

dat = data.frame(
  X,
  ydum
)
colnames(dat) = c("Int", "X1", "X2", "X3", "Y")

##RSS to Minimize Using Optim Function
min_RSS = function(data, par){
  with(data, sum( (par[1] + par[2] * X1 + par[3] * X2 + par[4] * X3 - ydum)^2 ))
}

linear_coef_df = data.frame(
  optim(par = c(0, 1, 1, 1), min_RSS, data = dat)$par,
  linear_reg$coefficients
)
colnames(linear_coef_df) = c("Optim Linear Function", "lm Linear Function")

kable(linear_coef_df, digits = 4, format = "markdown")
```

| | Optim Linear Function | lm Linear Function |
|-------------|-----------------------|--------------------|
| (Intercept) | 0.9108 | 0.9133 |
| X1 | 0.1315 | 0.1312 |
| X2 | -0.1021 | -0.1023 |
| X3 | 0.0231 | 0.0232 |

```
##Writing and Optimizing Logit Model
logit_reg = glm(ydum ~ X1 + X2 + X3, family = binomial(link = "logit"))

sigmoid = function(x){
  return( 1 / (1 + exp(-x)) )
}

logit_cost_function = function(par){

  len = length(X)
  g = sigmoid(X %*% par)
  C = -(1 / len) * sum( ydum * log(g) + (1 - ydum) * log(1 - g))
  return(C)

}

initial_par = rep(0, ncol(X))

logit_coef_df = data.frame(
  optim(par = initial_par, fn = logit_cost_function)$par,
  logit_reg$coefficients
)
```

```
colnames(logit_coef_df) = c("Optim logit Function", "glm logit Function")

kable(logit_coef_df, digits = 4, format = "markdown")
```

| | Optim logit Function | glm logit Function |
|-------------|----------------------|--------------------|
| (Intercept) | 5.5030 | 5.5030 |
| X1 | 2.0297 | 2.0299 |
| X2 | -1.5910 | -1.5912 |
| X3 | 0.3277 | 0.3281 |

```
##Writing and Optimizing Probit Model
probit_reg = glm(ydum ~ X1 + X2 + X3, family = binomial(link = "probit"))

probit_cost_function = function(par){

  len = length(X)
  phi = pnorm(X %*% par)
  C = -(1 / len) * sum ( ydum * log(phi) + (1 - ydum) * log(1 - phi) )
  return(C)

}

probit_coef_df = data.frame(
  optim(par = initial_par, fn = probit_cost_function)$par,
  probit_reg$coefficients
)
colnames(probit_coef_df) = c("Optim Probit Function", "glm Probit Function")

kable(probit_coef_df, digits = 4, format = "markdown")
```

| | Optim Probit Function | glm Probit Function |
|-------------|-----------------------|---------------------|
| (Intercept) | 3.0839 | 3.0846 |
| X1 | 1.1296 | 1.1296 |
| X2 | -0.8884 | -0.8885 |
| X3 | 0.1837 | 0.1830 |

Interpretation:

Linear Regression:

Values from the optim function and the base package lm function are not different at all, with the maximum difference in coefficients being 0.0025. This maximum quantity can be considered negligible.

- X1: An increase in X1 increases the ydum variable.
- X2: An increase in X2 decreases the ydum variable.
- X3: An increase in X3 increases the ydum variable.

Logistic Regression:

Values from the optim function and the base package glm function are not different at all, with the maximum difference in coefficients being 0.0004. This maximum quantity can be considered negligible.

X1: An increase in X1 increases the likelihood of $y_{dum} = 1$.

X2: An increase in X2 decreases the likelihood of $y_{dum} = 1$.

X3: An increase in X3 increases the likelihood of $y_{dum} = 1$.

Probit Regression:

Values from the optim function and the base package glm function are not different at all, with the maximum difference in coefficients being 0.0007. This maximum quantity can be considered negligible.

X1: An increase in X1 increases the likelihood of $y_{dum} = 1$.

X2: An increase in X2 decreases the likelihood of $y_{dum} = 1$.

X3: An increase in X3 increases the likelihood of $y_{dum} = 1$.

Exercise 5

```
#Probit Model Marginal Effects
probit_x_beta = predict(probit_reg, type = "link")
probit_pdf = dnorm(probit_x_beta) %>% mean()
probit_marginal = probit_pdf * probit_reg$coefficients

#logit Model Marginal Effects
logit_x_beta = predict(logit_reg, type = "link")
logit_pdf = dlogis(logit_x_beta) %>% mean()
logit_marginal = logit_pdf * logit_reg$coefficients

#Summary Table
marginals_df = data.frame(
  probit_marginal,
  logit_marginal
)
colnames(marginals_df) = c("Probit", "Logit")

kable(marginals_df, digits = 4, format = "markdown")
```

| | Probit | Logit |
|-------------|---------|---------|
| (Intercept) | 0.3741 | 0.3726 |
| X1 | 0.1370 | 0.1374 |
| X2 | -0.1078 | -0.1077 |
| X3 | 0.0222 | 0.0222 |

```
#Bootstrap for Probit
boot_probit_coef = list()
for(i in 1:49){
```



```

boot_probit_x_beta = sample(probit_x_beta, n, replace = TRUE)
boot_probit_pdf = dnorm(boot_probit_x_beta) %>% mean()
boot_probit_marginal = boot_probit_pdf * probit_reg$coefficients

boot_probit_coef[[i]] = boot_probit_marginal
}

#Bootstrap for Logit
boot_logit_coef = list()
for(i in 1:49){

  boot_logit_x_beta = sample(logit_x_beta, n, replace = TRUE)
  boot_logit_pdf = dnorm(boot_logit_x_beta) %>% mean()
  boot_logit_marginal = boot_logit_pdf * logit_reg$coefficients

  boot_logit_coef[[i]] = boot_logit_marginal
}

#Making Bootstrap Values into a Dataframe
boot_probit_temp_df = map(boot_probit_coef, data.frame)
boot_probit_coef_df = do.call("cbind", boot_probit_temp_df)
colnames(boot_probit_coef_df) = paste("sample", 1:49, sep = "")

boot_logit_temp_df = map(boot_logit_coef, data.frame)
boot_logit_coef_df = do.call("cbind", boot_logit_temp_df)
colnames(boot_logit_coef_df) = paste("sample", 1:49, sep = "")

#Calculating SD for Probit
bootstrap_probit_sd = apply(boot_probit_coef_df, 1, FUN = sd) %>% as.data.frame()
colnames(bootstrap_probit_sd) = "Probit SD of Bootstrap 49"
rownames(bootstrap_probit_sd) = c("Int", "X1", "X2", "X3")
kable(bootstrap_probit_sd, digits = 4, format = "markdown")

```

| Probit SD of Bootstrap 49 | |
|---------------------------|--------|
| Int | 0.0051 |
| X1 | 0.0019 |
| X2 | 0.0015 |
| X3 | 0.0003 |

```

#Calculating SD for Logit
bootstrap_logit_sd = apply(boot_logit_coef_df, 1, FUN = sd) %>% as.data.frame()
colnames(bootstrap_logit_sd) = "Logit SD of Bootstrap 49"
rownames(bootstrap_logit_sd) = c("Int", "X1", "X2", "X3")
kable(bootstrap_logit_sd, digits = 4, format = "markdown")

```

| Logit SD of Bootstrap 49 | |
|--------------------------|--------|
| Int | 0.0061 |
| X1 | 0.0023 |
| X2 | 0.0018 |

| Logit SD of Bootstrap 49 | |
|--------------------------|--------|
| X3 | 0.0004 |

```
#Delta Method

##Computing Covariance Matrix of Probit Regression
v = vcov(probit_reg)

##Function to be used for Gradient
g = function(x){

  probit_pdf = dnorm(X %*% x) %>% mean()
  probit_marginal = probit_pdf * x

  return(probit_marginal)
}

##Gradient
gradient_g = numDeriv::jacobian(g, probit_reg$coefficients)

##Table of Delta Method SD
Delta_df = gradient_g %*%
  v %*%
  t(gradient_g) %>%
  diag() %>%
  sqrt() %>%
  as.data.frame()
colnames(Delta_df) = "Delta Method SD"

kable(Delta_df, digits = 4, format = "markdown")
```

| Delta Method SD |
|-----------------|
| 0.0094 |
| 0.0044 |
| 0.0004 |
| 0.0057 |