Assignment 2

Xiaoshu Gui

Setup

```
library(dplyr)
library(purrr)
library(magrittr)
library(tidyr)
library(tibble)
library(stringr)
library(reshape2)
library(lme4)
library(mfx)
```

Exercise 1 Data Creation

```
set.seed(12345)
X1 <- runif(10000, min = 1, max = 3)
X2 <- rgamma(10000, shape = 3, scale = 2)
X3 <- rbinom(10000, 1, 0.3)
esp <- rnorm(10000, 2, 1)
Y <- 0.5 + 1.2*X1 - 0.9*X2 + 0.1*X3 + esp
X = as.matrix(cbind(1, X1, X2, X3))
ydum <- ifelse(Y > mean(Y), 1, 0)
```

Exercise 3 Numerical Optimization

```
# Likelihood function:
likelihood <- function(beta) {
  prod(pnorm(X %*% beta)^ydum * (1 - pnorm(X %*% beta)^(1 - ydum)))
}
# Log likelihood fucntion:
ll <- function(beta) {
  sum(ydum * log(pnorm(X %*% beta)) + (1 - ydum)*log(1 - pnorm(X %*% beta)))
}
# gradient:
gradient <- function(beta) {
  u = X %*% beta
  Phi = pnorm(u)
  phi = dnorm(u)
  v = ifelse(ydum == 1, phi/Phi, -phi/(1 - Phi))
  g = t(v) %*% X
  g = t(g)
  return(g)
}
# set up for gradient ascent
beta_old = matrix(c(0.1, 0.1, 0.1, 0.1), nrow = 4, ncol = 1)
```

```r
L = ll(beta_old) # an array that stores log-likelihood for each iteration
iter = 0 # count interation
tol = 10^(-2)
step_size = 10^(-5) # scaling parameter
# gradiant ascent
while(TRUE) {
  grad = gradient(beta_old)   # compute gradient as the direction
  beta_new = beta_old + grad * step_size # update
  L = c(L, ll(beta_new))  # store results
  diff = abs(L[length(L)] - L[length(L) - 1])
  iter = iter + 1
  beta_old = beta_new
  # stopping criteria
  if(iter == 1000){
    message("reach max iterations")
    break()
  }
  if(diff < tol){
    message("log-likelihood has converged")
    break()
  }
}
#L[length(L)] # log-likelihood
beta_old # coefficient
```

```
##          [,1]
##     2.2470181
## X1  1.2787218
## X2 -0.8130116
## X3  0.1484534
```

```r
# They are quite close to the true parameters, which are 1.2, -0.9, 0.1.
```

Exercise 2 OLS

1. The correlation between Y and X1 is given by *corr*, which is quite far away from 1.2 (Correlation should take a value between -1 and 1 though, not sure if it meant be a 0.2 instead of 1.2 in the question…).

```r
cor(Y, X1)
```

```
## [1] 0.2018198
```

```r
# calculate correlation by hand:
corr <- sum((X1 - mean(X1))*(Y - mean(Y)))/(sd(X1)*sd(Y)*(10000-1))
print(corr)
```

```
## [1] 0.2018198
```

2. Calculate the coefficients of the regression of Y on X, where X = (1, X1, X2, X3)

```r
m1 <- lm(Y~ (1 + X1 + X2 + X3))
coef(m1)
```

```
## (Intercept)          X1          X2          X3
##  2.48667726  1.21809470 -0.89860058  0.08064399
```

```r
# by hand
X = as.matrix(X)
y = as.matrix(Y)
beta <- solve(t(X) %*% X) %*% t(X) %*% y # The restuls are the same with coef(m1)
print(beta)
```

```
##           [,1]
##     2.48667726
## X1  1.21809470
## X2 -0.89860058
## X3  0.08064399
```

3. Calculate the standard errors of the regression

```r
# Using standand fomulas from OLS using the varcov matrix
k = ncol(X)
n = nrow(X)
beta0 <- y - X %*% beta
# create means of each column:
X_mean <- matrix(beta0, nrow = n) %*% cbind(mean(X[,1]), mean(X[,2]), mean(X[,3]), mean(X[,4]))
D <- X - X_mean # create a difference matrix
C <- (n-1)^(-1) * t(D) %*% D # create the variance-covariance matrix
sqrt(diag(C))/sqrt(n)
```

```
##                      X1          X2          X3
## 0.014150289 0.028895921 0.091605739 0.006231439
```

```r
# using bootstrap with 49 and 499 replications
df = data.frame(X1, X2, X3, Y)
n = nrow(df)
coef_table = matrix(0, nrow = 499, ncol = 4)
for (B in 1:499) {
  # Bootstrap sampling
  boots_index = sample(1:n, n, replace = TRUE)
  df_boots = df[boots_index, ]
  X = as.matrix(cbind(1, df_boots[, 1:(ncol(df_boots) - 1)]))
  y = as.matrix(df_boots[,ncol(df_boots)])
  # compute coefficient, store result
  coef_table[B, ] = solve(t(X)%*% X) %*% t(X) %*% y
}
# se. with 49 replications:
apply(coef_table[1:49,], 2, sd)/sqrt(49)
```

```
## [1] 0.0056775823 0.0024143427 0.0004559675 0.0028293777
```

```r
# se. with 499 reps:
apply(coef_table, 2, sd)/sqrt(499)
```

```
## [1] 0.0017304176 0.0007718834 0.0001350985 0.0009578184
```

Exercise 4

The log likelihood function for probit, logit and OLS model are the same, dispite different function F(x):

$$logL(\beta) = \sum_{i=1}^{n} y_i logF(X_i\beta) + (1 - y_i)log(1 - F(X_i\beta))$$

The probit model specifies F(x) as a standard normal cdf, the logit model cdf of the logistic distribution and OLS specifies a linear function (βX). Now we optimize the log likelihood of each model using the *optim* function in R.

```
X = as.matrix(cbind(1, X1, X2, X3))
## Probit model ##
# log likelihood function
prob_lik <- function(beta) {  # likelihood
  u = X %*% beta # linear predictor
  p = pnorm(u) # probability
  logl <- sum(ydum * log(p) + (1 - ydum) * log(1- p))
  return(-logl) # negative log-likelihood
}
# gradient function
prob_gr <- function (beta) {
  u <- X %*% beta
  p <- pnorm(u)
  t <- dnorm(u) * (ydum - p) / (p * (1 - p)) # chainrule
  -crossprod(X, t)  # gradient
}
# use optim package to maximize log likelihood
fit_prob <- optim(c(0,0,0,0), prob_lik, gr = prob_gr, method = "BFGS", hessian = TRUE)
fit_prob$par
```

```
## [1]  2.98527577  1.12753109 -0.88192555  0.09332505
```

```
# compare with glm estimates:
m2 <- glm(ydum ~ X1 + X2 + X3, family = binomial(link = "probit"))
m2$coefficients
```

```
## (Intercept)          X1          X2          X3
##  2.98527659  1.12753085 -0.88192555  0.09332476
```

```
## Logit model ##
# log likelihood function
logit_lik <- function(beta) {
  u <- exp(X %*% beta)
  p <- u/(1 + u)
  logl <- sum(t(ydum) %*% log(p) + t(1 - ydum) %*% log(1 - p))
  return(-logl)
}
# gradient function
logit_gr <- function(beta) {
  grad <- beta*0
  u <- exp(X %*% beta)
  p <- u/(1 + u)
  for (i in 1:k) {
  grad[i] <- sum(X[,i] * (ydum - p))
  }
  return(-grad)
}
# optimize the log likelihood funtion using optim()
fit_logit <- optim(c(0,0,0,0), logit_lik, logit_gr, method = "BFGS",  hessian = TRUE)
fit_logit$par
```

```
## [1]  5.3809214  2.0368218 -1.5908075  0.1672326
```

```
# compare with glm estimates:
m3 <- glm(ydum ~ X1 + X2 + X3, family = "binomial")
m3$coefficients
```

```
## (Intercept)          X1          X2          X3
##   5.3809347   2.0368304  -1.5908126   0.1672334
```

```
## linear model ##
# log likelihood function
ols_lik <- function(beta) {
  u = X %*% beta
  p = u
  logl <- sum((ydum) %*% log(p) + (1 - ydum) %*% log(1 - p))
}
fit_ols <- optim(c(0.5,0.1,0,0), ols_lik, gr = NULL)
fit_ols$par
```

```
## [1]  5.230484e-01  1.589575e-01  2.477325e-05 -3.922535e-04
```

Interpret and compare the estimated coefficients. How significant are they?

```
est_coef <- rbind(fit_prob$par, fit_logit$par, fit_ols$par)
colnames(est_coef) <- c("Intercept", "X1", "X2", "X3")
rownames(est_coef) <- c("Probit", "Logit", "OLS")
est_coef
```

```
##        Intercept        X1            X2            X3
## Probit 2.9852758 1.1275311 -8.819255e-01  0.0933250545
## Logit  5.3809214 2.0368218 -1.590808e+00  0.1672325789
## OLS    0.5230484 0.1589575  2.477325e-05 -0.0003922535
```

Notice that OLS generates estimates with wrong signs for X2 and X3, indicating that OLS is a bad fit for our data (dummy variables). To examine how significant are these estimates, we calculate the standard deviations of estimates of each model by bootstraping coefficients.

```
df2 = data.frame(X1, X2, X3, ydum)
R = 10                         # number of bootstrap samples
n = nrow(df2)                   # sample size
k = ncol(df2)     # number of coefficients
## LOGIT ##
# set up a empty matrix B1
B1 = matrix(nrow = R, ncol = k,
           dimnames = list(paste("Sample",1:R), names(fit_logit$par)))
for(i in 1:10) {
  # sample credit data with replacement
  boot_index = sample(x = 1:n, size = n, replace = TRUE)
  boot.data = df2[boot_index, ]
  X = as.matrix(cbind(1, boot.data[, 1:(ncol(boot.data) - 1)]))
  ydum = as.matrix(boot.data[,ncol(boot.data)])
  # fit the model on the boostrapped sample:
  fit_logit2 <- optim(c(0,0,0,0), logit_lik, logit_gr, method = "BFGS", hessian = TRUE)
  # store the coefficients
  B1[i,] = fit_logit2$par
}
# get standard deviations from boostrap coefficients
sd_logit <- apply(B1, 2, sd)
## Probit ##
```

```r
B2 = matrix(nrow = R, ncol = k,
            dimnames = list(paste("Sample",1:R), names(fit_prob$par)))
for(i in 1:10) {
  boot_index = sample(x = 1:n, size = n, replace = TRUE)
  boot.data = df2[boot_index, ]
  X = as.matrix(cbind(1, boot.data[, 1:(ncol(boot.data) - 1)]))
  ydum = as.matrix(boot.data[,ncol(boot.data)])
  fit_prob2 <- optim(c(0,0,0,0), prob_lik, gr = NULL, method = "BFGS", hessian = TRUE)
  B2[i,] = fit_prob2$par
}
sd_probit <- apply(B2, 2, sd)
## OLS ##
B3 = matrix(nrow = R, ncol = k,
            dimnames = list(paste("Sample",1:R), names(fit_ols$par)))
for(i in 1:10) {
  boot_index = sample(x = 1:n, size = n, replace = TRUE)
  boot.data = df2[boot_index, ]
  X = as.matrix(cbind(1, boot.data[, 1:(ncol(boot.data) - 1)]))
  ydum = as.matrix(boot.data[,ncol(boot.data)])
  #fit_ols <- optim(c(0.5,0.1,0,0), ols_lik, gr = NULL)
  B3[i,] = fit_ols$par
}
#sd_ols <- apply(B3, 2, sd)
# fit_ols produces NaNs. Perhaps because OLS result is bad...
sd_coef <- rbind(sd_probit, sd_logit) %>% print()
```

```
##                [,1]       [,2]       [,3]       [,4]
## sd_probit 0.1007766 0.04912024 0.01196486 0.04768611
## sd_logit  0.2273279 0.05370631 0.03174054 0.08339592
```

Exercise 5

The marginal effects of probit and logit models are

$$\frac{\partial Pr(y_i = 1|Xi)}{\partial X_{ij}} = F'(X\beta)\beta_j$$

, namely, the pdf function times each coefficient($F'(X\beta)\beta_j$). with different specifications of $F(X\beta)$ (standard normal or logistic). Therefore, the marginal effect of probit model is given by $\phi(X\beta)\beta$ and that of logit model is $\frac{\exp(X\beta)\beta}{(1+exp(X\beta))^2}$. For simplicity, I then calculate the average of sample marginal effects.

```r
## Probit Model ##
pdf_probit <- dnorm(X %*% fit_prob$par)
probit_coef <- as.matrix(fit_prob$par)
marginef_probit <- pdf_probit %*% t(probit_coef)
# The marginal effects differ with the point of evaluation Xi for nonlinear models.
# I calculate average marginal effects to better interpret the result:
avg_marginef_probit <- mean(pdf_probit) %*% t(probit_coef)
print(avg_marginef_probit)
```

```
##           [,1]      [,2]       [,3]       [,4]
## [1,] 0.3708178 0.1400569 -0.1095489 0.01159243
```

```r
## Logit Model ##
pdf_logit <- dlogis(X %*% fit_logit$par)
logit_coef <- as.matrix(fit_logit$par)
marginef_logit <- pdf_logit %*% t(logit_coef)
# average marginal effect:
avg_marginef_logit <- mean(dlogis(pdf_logit)) %*% t(logit_coef)
print(avg_marginef_logit)
```

```
##          [,1]      [,2]       [,3]       [,4]
## [1,] 1.341332 0.5077297 -0.3965493 0.04168698
```

```r
# cross-check with marginal effects calculated from glm estimates:
mean(dnorm(predict(m2)))*coef(m2)
```

```
## (Intercept)          X1          X2          X3
##  0.36780949  0.13892065 -0.10866015  0.01149834
```

```r
mean(dlogis(predict(m3)))*coef(m3) # Note that there's a bit dif for logit estimates
```

```
## (Intercept)          X1          X2          X3
##  0.36639804  0.13869164 -0.10832145  0.01138724
```

```r
# Now try another way to get average marginal effect for logit models:
# Pi = mean(ydum), where Pi = exp(XB)/(1+ exp(XB))
marg_logit <- mean(ydum)*(1- mean(ydum))*fit_logit$par
marg_logit # produces similar results with ave_marginef_logit
```

```
## [1]  1.32441241  0.50132531 -0.39154730  0.04116115
```

Compute the standard deviations of marginal effect using the delta method:

```r
library(numDeriv)
## Logit Model ##
# using the Delta method, define the (average) marginal effect function g(.) = f(Xbeta)*beta
me_logit <- function (beta) {
  u <- exp(X %*% beta)
  p <- u/(1 + u)
  me = p %*% t(beta)
}
# calculate the Jacobian Matrix of g(.)
j_logit <- jacobian(me_logit, beta)
# get the variance of marginal effects:
var_me_logit <- j_logit %*% vcov(m3) %*% t(j_logit)
# Then the variance of average marginal effect (AME) is:
var_ame_logit <- sum(var_me_logit)/(ncol(var_me_logit))^2
sd_ame_logit <- sqrt(var_ame_logit) %>% print()  # 0.02238572
```

```
## [1] 0.02238572
```

```r
# compare it with :
sd(marginef_probit)
```

```
## [1] 0.294634
```

```r
## Probit Model ##
me_probit <- function(beta) {
  u = X %*% beta
  p = pnorm(u)
  me = mean(p) %*% t(beta)
```

```r
}
j_probit <- jacobian(me_probit, beta)
var_me_probit <- j_probit %*% vcov(m2) %*% t(j_probit)
var_ame_probit <- sum(var_me_probit)/(ncol(var_me_probit))^2
sd_ame_probit <- sqrt(var_ame_probit) %>% print()  # 0.01176799
```

```
## [1] 0.01176799
```

```r
# compare it with:
sd(marginef_logit)
```

```
## [1] 0.3057361
```

. . . using bootstrap:

```r
## Probit Nodel ##
me_table_probit = matrix(0, 10, ncol = k)
for(i in 1:10) {
  boot_index = sample(x = 1:n, size = n, replace = TRUE)  # sample credit data with replacement
  boot.data = df2[boot_index, ]
  X = as.matrix(cbind(1, boot.data[, 1:(ncol(boot.data) - 1)]))
  ydum = as.matrix(boot.data[,ncol(boot.data)])
  # fit the AME on the boostrapped sample:
  fit_prob <- optim(c(0,0,0,0), prob_lik, gr = prob_gr, method = "BFGS", hessian = TRUE)
  pdf_probit <- dnorm(X %*% fit_prob$par)
  probit_coef <- as.matrix(fit_prob$par)
  avg_marginef_probit <- mean(pdf_probit) %*% t(probit_coef)
  me_table_probit[i,] = avg_marginef_probit  # store the coefficients
}
apply(me_table_probit, 2, sd)
```

```
## [1] 0.012291351 0.005222634 0.001034898 0.003945527
```

```r
## Logit Model ##
me_table_logit = matrix(0, 10, ncol = k)
for(i in 1:10) {
  boot_index = sample(x = 1:n, size = n, replace = TRUE)
  boot.data = df2[boot_index, ]
  X = as.matrix(cbind(1, boot.data[, 1:(ncol(boot.data) - 1)]))
  ydum = as.matrix(boot.data[,ncol(boot.data)])
  fit_logit <- optim(c(0,0,0,0), logit_lik, logit_gr, method = "BFGS",  hessian = TRUE)
  pdf_logit <- dlogis(X %*% fit_logit$par)
  logit_coef <- as.matrix(fit_logit$par)
  avg_marginef_logit <- mean(dlogis(pdf_logit)) %*% t(logit_coef)
  me_table_logit[i,] = avg_marginef_logit  # store the coefficients
}
apply(me_table_logit, 2, sd)
```

```
## [1] 0.045214032 0.027839387 0.006124092 0.016121713
```