HW2

Jie Ren

February 8, 2019

```
rm(list=ls())
setwd("C:/Users/jiere/Dropbox/Spring 2019/ECON 613/ECON613_HW/HW2_output")
set.seed(613)
```

Exercise 1: Data generation —

```
obs <- 10000
X1 <- runif(obs, max = 3, min = 1)
X2 <- rgamma(obs,3,scale = 2)
X3 <- rbinom(obs,1,0.3)
eps <- rnorm(obs, mean = 2, sd = 1)
Y <- 0.5 + 1.2*X1 - 0.9*X2 + 0.1*X3 + eps
ydum <- ifelse(Y > mean(Y),1,0)
mydata <- cbind(Y,ydum,X1,X2,X3,eps)
```

Exercise 2: —

Correlation between Y and X1

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

```
## [1] 0.224856
```

The correlation is limited from -1 to 1, so only thing we can tell is that there is a strong positive correlation between these two variables, which match the fact that the coef on X1 is possitive.

Regression of Y on X where X = (1,X1,X2,X3) (Mannually)

```
ols <- function(X,y,se=F){
  n <- nrow(X)
  k <- ncol(X)
  ols.coef <- solve(t(X)%*%X)%*%(t(X)%*%y) #coefficient
  ols.res <- (y-X%*%ols.coef) # residual
  ols.V <- 1/(n-k) * as.numeric(t(ols.res)%*%ols.res)*solve(t(X)%*%X) #covariance matrix
  ols.se <- as.matrix(sqrt(diag(ols.V))) #standard error
  ifelse(se == T, return(data.frame(b_hat = ols.coef,se = ols.se)),return(data.frame(b_hat =
 ols.coef))) # output coef only by default
}

X <- cbind(1,X1,X2,X3)
y <- as.matrix(Y)
ols.result <- ols(X,y,se=T)
ols.result
```

```
##         b_hat          se
##     2.4664985 0.040936074
## X1  1.2344523 0.017393381
## X2 -0.9058537 0.002881981
## X3  0.1240774 0.021997461
```

Check with lm function

```
lm.result <- lm(Y~X1+X2+X3, data.frame(mydata))
coef(summary(lm.result))[,c(1,2)] #coefficient & standard error
```

```
##               Estimate  Std. Error
## (Intercept)  2.4664985 0.040936074
## X1           1.2344523 0.017393381
## X2          -0.9058537 0.002881981
## X3           0.1240774 0.021997461
```

Boodstrap mannually

```
bootstrapse <- function(n){
  # Input times of bootstrap
  boot.coef <- data.frame(matrix(nrow=4, ncol=0))
  for (i in 1:n) {
    # sample from existing data to get X.s, y.s
    df.s <- mydata[sample(nrow(mydata),size = nrow(mydata),replace = T),]
    y.s <- df.s[,1]
    X.s <- cbind(1,df.s[,c(3:5)])
    # calculate ols coefficient in each sample
    boot.coef <- cbind(boot.coef,ols(X.s,y.s))
  }
  # Report the SE over all the obtained coefficients
  return(data.frame(se = apply(boot.coef,1,sd)))
}

boot.se_49 <- bootstrapse(49)
boot.se_499 <- bootstrapse(499)
boot.se_49
```

```
##             se
##    0.050086521
## X1 0.018716541
## X2 0.003642031
## X3 0.025282905
```

```
boot.se_499
```

```
##             se
##    0.039802565
## X1 0.017253930
## X2 0.003001135
## X3 0.021328256
```

Exercise 3: —

likelihood function

```
probit.llike <- function(b. = b, y. = ydum,X. = X){
  phi <- pnorm(X.%*%b.)
  phi[phi==1] <- 0.9999 # avoid NaN of log function
  phi[phi==0] <- 0.0001
  f <- sum(y.*log(phi))+sum((1-y.)*log(1-phi))
  f <- -f
  return(f)
}
```

Optimizer using steepest descent

First define a function to calcuate gradient/jacobian

```
jacobian <- function(fun,par){
  d <- 1e-8
  par. <- matrix(par,length(par),length(par)) # generate a matrix that repeating par vector
  J <- (apply(par. + diag(d,length(par)),2,fun)-apply(par.,2,fun))/d
  return(J)
}
```

By input a inial guess of parameter, relative stopping criteria (percentage change in function), and function, using gradient decent method featuring backtracking line search, you can get the parameter that minimize the funtion. Note: First arguement of fun. must be the parameter you want to get, X and y must be set into the default value.

```
graddes <- function(b,stop,fun){
  d <- 1e-8 # delta in the finite difference method
  alpha <- 0.01 # any initial alpha
  delta <- Inf
  while (delta > stop){
    # Calculate gradient for each regressor by Finite Difference Method
    g <- jacobian(probit.llike,b)
    # Backtracking (Armijo-Goldstein condition tests to make sure the size of alpha)
    if(fun(b - alpha*g)>fun(b)-0.1*alpha*t(g)%*%g){
      alpha <- 0.5*alpha
      next
    }
    # Make step forward (t+1) and saved in bn
    bn <- b - alpha*g
    # Stopping Criteria
    delta <- (abs(fun(bn)-fun(b))/abs(fun(b)))
    b <- bn
  }
  return(b)
}
result.gd <- graddes(c(0,0,0,0),1e-5,probit.llike)
result.gd
```

```
## [1]  2.39494605  1.27595516 -0.82536805  0.08740896
```

Except the coefficient on constant, others are close, but still have a big difference.

Exercise 4 —

Optimize Probit

```
b.p <- c(0,0,0,0)
result.p <- optim(par = b.p, probit.llike)
result.p$par
```

```
## [1]  3.02500737  1.15932567 -0.88828104  0.04192845
```

Optimizing Logit

```
logit.llike <- function(b., y. = ydum,X. = X){
  gamma <- exp(X.%*%b.)/(1+exp(X.%*%b.))
  f <- sum(y.*log(gamma))+sum((1-y.)*log(1-gamma))
  f <- -f
  return(f)
}
b.l <- c(0,0,0,0)
result.l <- optim(par = b.l, logit.llike)
result.l$par
```

```
## [1]  5.41688230  2.07285400 -1.58992423  0.07182841
```

Optimizing Linear Probability

```
result.lp <- lm(ydum~X1+X2+X3)
summary(result.lp)
```

```
## 
## Call:
## lm(formula = ydum ~ X1 + X2 + X3)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.87837 -0.26854  0.05906  0.24697  1.81983 
## 
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.8965251  0.0134968   66.42   <2e-16 ***
## X1           0.1430257  0.0057347   24.94   <2e-16 ***
## X2          -0.1032952  0.0009502 -108.71   <2e-16 ***
## X3           0.0094310  0.0072527    1.30    0.194    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.3305 on 9996 degrees of freedom
## Multiple R-squared:  0.5563, Adjusted R-squared:  0.5562 
## F-statistic:  4177 on 3 and 9996 DF,  p-value: < 2.2e-16
```

Check significance with glm function

```
result.p.glm <- glm(ydum ~ X1 + X2 + X3, family = binomial(link = "probit"),
                    data = data.frame(mydata))
coef(summary(result.p.glm))
```

```
##                Estimate Std. Error    z value      Pr(>|z|)
## (Intercept)  3.02375700 0.09999510  30.2390518 7.267853e-201
## X1           1.15995818 0.04319531  26.8537961 7.616121e-159
## X2          -0.88829516 0.01814131 -48.9653239  0.000000e+00
## X3           0.04239708 0.04724673   0.8973547  3.695297e-01
```

```
result.l.glm <- glm(ydum ~ X1 + X2 + X3, family = binomial(link = "logit"),
                    data = data.frame(mydata))
coef(summary(result.l.glm))
```

```
##                Estimate Std. Error    z value      Pr(>|z|)
## (Intercept)  5.41720775 0.18674718  29.0082438 5.178847e-185
## X1           2.07443382 0.07992105  25.9560387 1.554178e-148
## X2          -1.59043702 0.03598611 -44.1958600  0.000000e+00
## X3           0.07133257 0.08463913   0.8427848  3.993488e-01
```

The coefficient varies largely across these three methods, but the sign on the coefficients are the same, and all X3 coefs are not significant.For Probit and logit, without calcuating the marginal effect, we can only interprete the sign. Higher X2 can decrease the probability of ydum = 1, which means higher X2 is likely to generate Y below mean. Higher X1 and X3 can increase the probability of ydum = 1, which means that higher X1 and X3 is likely to genrate Y above mean. This match the sign in the data generating process. For linear probability model, X1: one unit increase in X1 likely to increase the likelihood of ydum = 1 by 14%; X2: one unit increase in X2 likely to decrease the likelihood of ydum = 1 by 10%; X3: one unit increase in X3 likely to increase the likelihood of ydum = 1 by 1%.

Exercise 5 —

Compute Average Marginal Effect (AME) of X of probit

```
AME.p <- function(result) mean(dnorm(X%*%result))*result # result are the parameters
AME.p(coef(result.p.glm))
```

```
##  (Intercept)          X1          X2          X3
##  0.365469414  0.140199505 -0.107364683  0.005124365
```

Compute Average Margeinal Effect of X of Logit

```
AME.l <- function(result) mean(dlogis(X%*%result))*result
AME.l(coef(result.l.glm))
```

```
##  (Intercept)          X1          X2          X3
##  0.364919891  0.139740287 -0.107136763  0.004805183
```

Comment: Except the constant term, both models' average marginal effects are close to linear probability

Compute the Standard Error of AME by delta method (Probit)

```
J <- jacobian(AME.p, coef(result.p.glm)) # "jacobian" defined in EX3
cov_matrixv <- vcov(result.p.glm)
se.p <- sqrt(diag(J%*%cov_matrixv%*%t(J)))
se.p
```

```
## [1] 0.0095822375 0.0044077138 0.0004010621 0.0057091892
```

Compute the Standard Error of AME by delta method (Logit)

```
J <- jacobian(AME.l, coef(result.l.glm))
cov_matrixv <- vcov(result.l.glm)
se.l <- sqrt(diag(J%*%cov_matrixv%*%t(J)))
se.l
```

```
## [1] 0.0095501434 0.0044069389 0.0003987282 0.0056998702
```

Compute the SE of AME by Bootstrap (Probit and Logit)

```
bootstrapse2 <- function(n,l){
  # input bootstrap times and type of link function (connect to different pdf)
  boot.me <- data.frame(matrix(nrow=4, ncol=0))
  pdf <-ifelse(l == "probit", dnorm, dlogis) # note to myself: ifelse can only return one val
ue
  for (i in 1:n) {
    # sample from existing data to get X.s, y.s
    df.s <- mydata[sample(nrow(mydata),size = nrow(mydata),replace = T),]
    result <- glm(ydum ~ X1 + X2 + X3, family=binomial(link = l),data.frame(df.s))
    # calculate Average marginal effect
    me <- mean(pdf(X%*%coef(result)))*coef(result)
    boot.me <- cbind(boot.me,me)
  }
  # Report the SE over all the obtained average margianl effect
  return(data.frame(se = apply(boot.me,1,sd)))
}
bootstrapse2(49,"probit")
```

```
##                       se
## (Intercept) 0.0083011737
## X1          0.0037640063
## X2          0.0004455203
## X3          0.0054863894
```

```
bootstrapse2(49,"logit")
```

```
##                       se
## (Intercept) 0.0096116087
## X1          0.0042942151
## X2          0.0004294998
## X3          0.0055303077
```