Stats 201B: Statistical Modeling and Learning
Problem Set 2

January 21, 2015

Due Tuesday 2 February

Please note the following instructions:

- Your solutions should be printed out and handed in during class on 5 February
- Responses should be typeset in LaTeX, or through Rstudio/Rmarkdown or similar.
- Coding tasks should be done in `R`
- Please include a copy of your code in the write-up (e.g. using markdown or the "verbatim" environment) **and also** upload your code on the course website.

Problem 1

(20pt) Suppose that we have a random sample of size n from the population of the U.S. eligible voters. Let Y_i be a binary variable indicating whether respondent i voted in the previous presidential election. Let T_i be a binary predictor of interest representing i's level of education, where $T_i = 1$ indicates that i has high education and $T_i = 0$ low education. Finally, let $W_i = [W_{1i} \cdots W_{Ki}]^\top$ represent the set of K control variables for respondents' other demographic characteristics (e.g. age, gender, etc.). For notational convenience, we use X_i to represent all the right-hand-side variables and an intercept, i.e., $X_i = [1 \; T_i \; W_i^\top]^\top$.

Assume that we use the following logistic regression to model this data:

$$\Pr(Y_i = 1 \mid X_i) \;=\; \frac{\exp(X_i^\top \beta)}{1 + \exp(X_i^\top \beta)} \;=\; \frac{\exp(\alpha + \gamma T_i + W_i^\top \delta)}{1 + \exp(\alpha + \gamma T_i + W_i^\top \delta)},$$

where $\beta = [\alpha \; \gamma \; \delta^\top]^\top$ is the vector of coefficients of length $K + 2$. Note that α and γ are scalars, while δ is a K-dimensional column vector.

1. (6pt) Consider the following quantity called the *odds ratio*:

$$OR(w) \;=\; \frac{\Pr(Y_i = 1 \mid T_i = 1, W_i = w) / \Pr(Y_i = 0 \mid T_i = 1, W_i = w)}{\Pr(Y_i = 1 \mid T_i = 0, W_i = w) / \Pr(Y_i = 0 \mid T_i = 0, W_i = w)}.$$

   What does this quantity represent in substantive terms? Explain in plain English.

This particular odds ratio represents the odds that the respondents voted given that they have high level education, compared to the odds that the respondents voted given that they have low level education. If the odds ratio is much greater than 1, high level education is considered to contribute to whether the respondents voted.

2. (7pt) Derive $OR(w)$ for the logistic regression model. What does the result imply about this model?

$$OR(\omega) = \frac{\frac{1}{1+e^{-(\alpha+\gamma+W_i^T\omega)}} \Big/ \frac{e^{-(\alpha+\gamma+W_i^T\omega)}}{1+e^{-(\alpha+\gamma+W_i^T\omega)}}}{\frac{1}{1+e^{-(\alpha+W_i^T\omega)}} \Big/ \frac{e^{-(\alpha+W_i^T\omega)}}{1+e^{-(\alpha+W_i^T\omega)}}} = \frac{e^{\alpha+\gamma+W_i^T\omega}}{e^{\alpha+W_i^T\omega}} = e^{\gamma}$$

If γ is much greater than 0, the odds ratio is much greater than 1. So having high level education may contributes to the probability that respondents voted. But there may be chance that both factors are associated with a third factor.

3. (7pt) Suppose we estimate the model to get consistent estimates for γ as well as its standard error, which we will call $\hat{\gamma}$ and $\hat{\sigma}$, respectively. Using the Delta method, derive the asymptotic variance of the estimated odds ratio, $\widehat{OR}$.

$$\widehat{\gamma} - \gamma \sim N(0, \widehat{\sigma}^2)$$

$$\widehat{OR} - OR = e^{\widehat{\gamma}} - e^{\gamma} \approx e^{\gamma} + e^{\gamma}(\widehat{\gamma} - \gamma) \qquad \text{since } \frac{\partial e^{\gamma}}{\partial \gamma} = e^{\gamma}$$

$$= e^{\gamma} + e^{\gamma} N(0, \widehat{\sigma}^2)$$

$$= N(e^{\gamma}, e^{2\gamma}\widehat{\sigma}^2)$$

The asymptotic variance is $e^{2\gamma}\sigma^2$.

Problem 2

(20pts) Consider a random sample of size n, with a binary outcome $Y_i \in \{0, 1\}$. We assume $Y_i \sim Bern(\pi_i)$.

Furthermore, we use a linear probability model so that $\pi_i = X_i^{\top}\beta$, where X_i is a predictor vector of length P. Here we consider estimation of β by maximum likelihood, though, rather than least squares.

4. (1pt) In the conventional terminology of GLMs, what "link function" is used here?

Identity function, $f(x) = x$.

5. (5pt) Show that the error term from the linear regression of Y_i on X_i is always heteroskedastic. If you were to use OLS for this model, could you correct for this problem?

$\text{Var}(Y_i) = \pi_i(1-\pi_i) = X_i^T\beta(1-X_i^T\beta) = f(X_i)$

So once X_i is changed, $\text{Var}(Y_i)$ is changed.
OLS cannot correct the problem, since OLS is exactly GLM whose link function is identity function, $f(x) = x$.

6. (4pt) Write down the likelihood function for this problem.

$$L(\beta|Y_i, X_i, i = 1, \ldots, n) = \prod_i^n \pi_i^{y_i}(1-\pi_i)^{1-y_i} = \prod_i^n (X_i^T\beta)^{y_i}(1-X_i^T\beta)^{1-y_i}$$

7. (5pt) Find the log-likelihood and then the score. Show that the score function for the sample is given by

$$S(\beta) \;=\; \sum_{i=1}^{n} \frac{Y_i - X_i^\top \beta}{X_i^\top \beta(1-X_i^\top \beta)} X_i.$$

$$l(\beta|Y_i, X_i, i = 1, \ldots, n) = \sum_i^n \left[Y_i \ln(X_i^T\beta) + (1-Y_i)\ln(1-X_i^T\beta)\right]$$

$$\frac{\partial l}{\partial \beta} = \sum_i^n \left[Y_i \frac{1}{X_i^T\beta} X_i + (1-Y_i)\frac{1}{1-X_I^T\beta}(-X_i)\right]$$

$$= \sum_i^n \frac{Y_i(1-X_i^T\beta) + (1-Y_i)(-X_i^T\beta)}{X_i^T\beta(1-X_i^T\beta)} X_i$$

$$= \sum_i^n \frac{Y_i - X_i^T\beta}{X_i^T\beta(1-X_i^T\beta)} X_i$$

8. (5pt) Derive the asymptotic variance of the maximum likelihood estimate of β assuming correct specification. (**Hint 1:** You need not do this from first principles – you can use what you know about the relationship between the Hessian and the asymptotic variance under correct specification assumptions; **Hint 2:** You'll need to realize that $\mathbb{E}[Y_i|X_i] = X_i^\top\beta$ by

definition of the model, under correct specification.)

$$\frac{\partial^2 l}{\partial\beta\partial\beta^T} = \sum_i^n X_i \frac{-X_i^T[X_i^T\beta(1-X_i^T\beta)] - (y_i - X_i^T\beta)[X_i^T - 2(X_i^T\beta)X_i^T]}{[X_i^T\beta(1-X_i^T\beta)]^2}$$

$$= \sum_i^n X_i X_i^T \left\{ \frac{-[X_i^T\beta(1-X_i^T\beta)] - (y_i - X_i^T\beta)(1-2(X_i^T\beta))}{[X_i^T\beta(1-X_i^T\beta)]^2} \right\}$$

$$I(\beta) = -\mathbb{E}\frac{\partial^2 l}{\partial\beta\partial\beta^T} = \sum_i^n X_i X_i^T \left\{ \frac{[X_i^T\beta(1-X_i^T\beta)] + (\mathbb{E}[y_i] - X_i^T\beta)(1-2(X_i^T\beta))}{[X_i^T\beta(1-X_i^T\beta)]^2} \right\}$$

$$= \sum_i^n X_i X_i^T \left\{ \frac{1}{[X_i^T\beta(1-X_i^T\beta)]} \right\}$$

$$\text{asymptotic variance} = I(\beta)^{-1} = \left\{ \sum_i^n \frac{X_i X_i^T}{[X_i^T\beta(1-X_i^T\beta)]} \right\}^{-1}$$

Problem 3

9. (5pt) Consider again a binary outcome. Write down the likelihood function (for individuals and for the sample) and the log-likelihood (also for individuals and sample), but this time, assume the probit link function, $\pi_i = \Phi(X_i^\top\beta)$.

$$L(\beta|Y_i, X_i, i = 1, \ldots, n) = \prod_i^N [\Phi(X_i^T\beta)]^{y_i}[1 - \Psi(X_i^T\beta)]^{1-y_i}$$

$$l(\beta|Y_i, X_i, i = 1, \ldots, n) = \sum_i^N y_i \ln[\Psi(X_i^T\beta)] + (1-y_i)\ln[1 - \Psi(X_i^T\beta)]$$

10. (12pt) Write your own function in `R` to fit probit models. Your function should accept: (a) a vector of responses, `y`; (b)a matrix of covariates (and possibly an intercept), `X`. This `X` must be allowed to have multiple covariates (columns). It should return: (a) the fitted coefficients, $\hat{\beta}_{MLE}$; (b) the log-likelihood at the solution; and (c) an estimate of Var(β), which may use the numerically estimated hessian provided by `optim`. *To get full credit, you must comment your code such that a colleague of yours would be able to understand each steps.*

```
# Since we use BFGS method in optim(), there may be errors due to gradient
#calculation. In that case, reuse probit(), because I use randomized intial
#paramenters in optim() to guarantee the program runs well for most initial
#values
#
#y is a n-length vector, X is a n*p matrix
probit <-function(y, X)
{
```

```
  #switch y to a n*1 matrix
  Y = matrix(y, ncol=1)
  #check if the dimensions of y and X are compatible
  if(nrow(y) != nrow(X))
    return("the number of responds and the number of
           observations are not consistent")
  #set the initial coefficients
  initial_beta = matrix(rnorm(ncol(X)),ncol=1)
  #check if X contains intercept.  If not, add it to X and modify the dimension
  # of initial parameter.
  if(prod(apply(X, 2, var))!= 0)
  {
    X = cbind(1, X)
    initial_beta = c(rnorm(1), initial_beta)
  }



  #establish the targetd log-likelyhood function
  loglik <- function(beta)
  {
    psi_value = pnorm(X %*% beta)  #use matrix multiplication for conveniece
    return( t(Y)%*%log(psi_value) + t(1-Y) %*% log(1-psi_value))
  }
  #optimization
  probit_optim = optim(par = initial_beta, fn=loglik,
                       method = "BFGS", control = list(fnscale = -1),
                       hessian = TRUE)
  #result
  result = list("coefficients"=probit_optim$par, "log-likelihood value"=
                  probit_optim$value, "variance"=
                  solve(-probit_optim$hessian))
  return(result)
}
```

11. (8pt) Now let's setup a test for your probit function to prove that it works. You should:

    - Create your `X` matrix, with 200 observations on three covariates, each with a standard normal distribution.
    - Generate your `y` vector assuming that the probit models is correct for your data. I.e., choose some values of β, and use these together with your `X` values and your probit link function to *stochastically* generate a set of outcomes, `y`.

- Run your model on these data and compare your coefficient estimate to the true results you used to generate the data.

```
#generate matrix X
X = matrix(NaN,nrow=200, ncol=3)
for(i in 1:3)
  X[,i] = rnorm(nrow(X))
#choose beta as (1, 2, 1)
beta = matrix(c(1, 2, 1), ncol=1)
#generate y
prob = pnorm(X%*%beta)
y = matrix(NaN, nrow=nrow(X), ncol=1)
for(i in (1:nrow(X)))
  y[i,] = rbinom(1, 1, prob[i])
#calculate the result
probit(y, X)
> result = probit(y, X)
> result
$coefficients
[1] -0.02037657  1.00415335  1.96019498  0.97004038

$`log-likelihood value`
[1] -50.57152

$variance
               [,1]          [,2]        [,3]        [,4]
[1,]  0.0199549385 -0.0009915645 0.001697011 0.001217589
[2,] -0.0009915645  0.0320526534 0.025087862 0.013979006
[3,]  0.0016970107  0.0250878622 0.074822581 0.023843131
[4,]  0.0012175891  0.0139790056 0.023843131 0.029248632
```

As we see, the coefficient of intercept is close to 0, and others are close to 1, 2, 1 respectively.

12. (8pt) Test whether the asymptotic distribution for $\hat{\beta}$ obtained using the standard variance covariance matrix is similar to the result using the bootstrap, and whether these are similar to the "true" expected sampling variability. Specifically, overlay three density plots: (a) the distribution of β estimates that you get from the asymptotic distribution; (b) the distribution of β estimates that you get from bootstrapping on the original sample; (c) the distribution of β estimates you get by drawing entirely new samples from the known data generating process.

Comment on how these results compare to each other.

```
#generate beta from the asymptotic distribution
#
library(MASS)
#generate beta from the asymptotic distribution
beta_a = mvrnorm(n = 8000, mu = result$coefficients[-1],
        Sigma = result$variance[-1, -1])


#generate beta gotten from bootstrapping
#
Itera = 8000
#the set of all the estimators
beta_b = matrix(NaN, nrow = Itera, ncol = 3)
i=1
N = nrow(X)
while(i <= Itera)
{
  #There may be errors due to method BFGS, so use tryCatch
  tryCatch(
    {
      #bootstrap sample from X, y
      index = sample(1:N, N, replace = TRUE)
      X_new = X[index,]
      y_new = y[index, , drop=FALSE]
      #calculate the new beta
      beta_b[i, ] = probit(y_new, X_new)$coefficients[-1]
      i = i + 1
    },
    error = function(e){}
  )
}



#Distribution gotten from drawing entirely new samples
#
Itera2 = 8000
```

```
beta_c = matrix(NaN, nrow = Itera2, ncol = 3)
i = 1

while(i <= Itera2)
{
  tryCatch(
    {
      #generate X, Y
      new_X = matrix(NaN,nrow=200, ncol=3)
      for(j in 1:3)
        new_X[,j] = rnorm(nrow(new_X))
      #choose beta as (1, 2, 1)
      new_beta = matrix(c(1, 2, 1), ncol=1)
      #generate y
      prob = pnorm(new_X%*%beta)
      new_y = matrix(NaN, nrow=nrow(new_X), ncol=1)
      for(j in (1:nrow(new_X)))
        new_y[j,] = rbinom(1, 1, prob[j])
      beta_c[i, ] = probit(new_y, new_X)$coefficients[-1]
      i = i + 1
    },
    error = function(e){}
  )
}
#plot beta_1
plot(density(beta_a[,1], , bw = 0.05), col = 1, lty = 1,
     xlim=c(0.25, 1.75), ylim=c(0, 2.5), main = "beta_1")
lines(density(beta_b[,1], , bw = 0.05), col = 2, lty = 2)
lines(density(beta_c[,1], , bw = 0.05), col = 4, lty = 4)
legend(x = 1.3, y = 2.5, legend = c("method a", "method b", "method c"),
       col = c(1, 2, 4), lty = c(1, 2, 4))

#plot beta_2
plot(density(beta_a[,2], , bw = 0.05), col = 1, lty = 1,
     xlim=c(1.25, 3.5), ylim=c(0, 1.5), main = "beta_2")
lines(density(beta_b[,2], , bw = 0.05), col = 2, lty = 2)
lines(density(beta_c[,2], , bw = 0.05), col = 4, lty = 4)
legend(x = 2.8, y = 1.5, legend = c("method a", "method b", "method c"),
       col = c(1, 2, 4), lty = c(1, 2, 4))
```

```
#plot beta_3
plot(density(beta_a[,3], , bw = 0.05), col = 1, lty = 1,
     xlim=c(0.4, 2.25), ylim=c(0, 2.2), main = "beta_3")
lines(density(beta_b[,3], , bw = 0.05), col = 2, lty = 2)
lines(density(beta_c[,3], , bw = 0.05), col = 4, lty = 4)
legend(x = 1.7, y = 2.0, legend = c("method a", "method b", "method c"),
       col = c(1, 2, 4), lty = c(1, 2, 4))
```

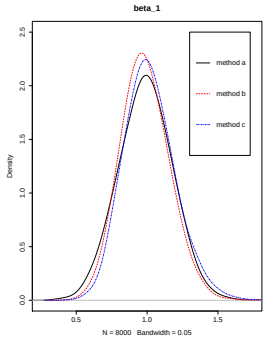

Figure 1: β_1

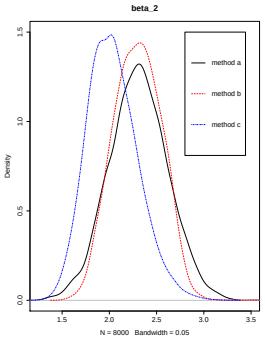

Figure 2: β_2

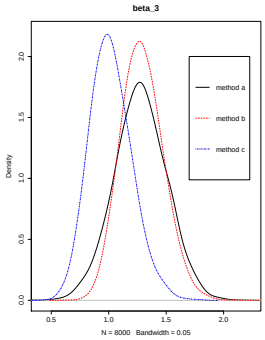

Figure 3: β_3

From figure 1, the distribution of β_1 gotten from these three method are similar. They all approximately have mean 1 and standard error 0.18. But from figure 2 and figure 3, we see method (a) and (b) have biased mean. Because (a) and (b) can only calculate β based on known information, but (c) calculate β with new generated data, so (c) could be more accurate. Again, their standard errors are similar, and 0.26 for figure 2 and 0.19 for figure 3. For method (a) and (b), bootstrapping is a little more robust than getting β from asymptotic distribution.

Problem 4

(32pt) In their article in *Quarterly Journal of Political Science*, Atkinson, Enos and Hill (2009) assess the effect of candidate appearance on vote choices using exit poll data in the 1992–2006 Senate elections. You can find the paper at Seth Hill's website. The authors first measure "facial competency" for the candidates, by asking a group of participants to judge how "competent" political candidates appeared to be based only on photographs of their faces. They then analyze whether the facial competency measure significantly influences voting behavior.

For this problem set, we will re-evaluate these findings, but with a different data set. Specifically, the facial competency data from that paper have been merged with survey data from the 1992-2004 American National Election Study (ANES).[1] Each observation (row) represents an individual survey respondent who voted in the election for Senate. We will examine how each respondent's

[1]Credit goes to Teppei Yamamoto at MIT for providing this merged dataset.

self-reported vote in the Senate election relates to the facial competency of the candidates and other features of the race.

The file `PS2_data.csv` (available on the course CCLE website) contains a modified version of the data. The dataset includes the following variables:

- `vote_incumbent`: Binary variable, equal to 1 if the respondent voted for the incumbent
- `same`: Binary variable, equal to 1 if the respondent identifies with the same party as the incumbent
- `diff`: Binary variable, equal to 1 if the respondent identifies with a different party from the incumbent's party
- `comp_challenger`: Facial competence score for the challenger candidate
- `comp_incumbent`: Facial competence score for the incumbent
- `id`: Identifier coding state and year of election
- `cook_inc_risk`: Measure of competitiveness of the senate race, ranging from 0 to 3
- `pop`: the population of the state, in millions
- `inc_tenure`: the number of years that the incumbent senator has been in office
- `inc_age`: the age of the incumbent senator
- `expenditures`: the log of the campaign expenditures of the challenger

Please answer the following questions:

13. (8pt) Estimate the probit model represented in column 3 of Table 2 (p.237) *without* using the clustered standard errors. Describe the coefficient and standard error for the incumbent and challenger competency variables, and how they compare to the original estimates in the paper. Describe the additional assumptions required for the choice of standard errors used here to be appropriate. (**Note:** As noted on page 233, AEH standardize the competence scores by subtracting the mean and dividing by the standard deviation. You will want to do that here as well. You might want to use the `scale` function in `R`.)

```
#read_data
PS2_data = read.csv(file = "E:\\PS2_data.csv", header = TRUE)

cook_inc_risk = PS2_data$cook_inc_risk
diff = PS2_data$diff
same = PS2_data$same
scaled_comp_challenger = scale(PS2_data$comp_challenger) #standardize the data
```

```
scaled_comp_incumbent = scale(PS2_data$comp_incumbent) #standardize the data
inc_tenure = PS2_data$inc_tenure
tenure_squared = inc_tenure^2
inc_age = PS2_data$inc_age
age_squared = inc_age^2
vote = PS2_data$vote_incumbent
#construct data frame
Senate_1 = data.frame(cook_inc_risk, diff, same, scaled_comp_challenger,
                      scaled_comp_incumbent, inc_tenure,
                      inc_age, vote)
glm_Senate_1 = glm(vote~., data = Senate_1,
                   family = binomial(link = "probit"))
summary(glm_Senate_1)
> summary(glm_Senate_1)

Call:
glm(formula = vote ~ ., family = binomial(link = "probit"), data = Senate_1)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6161  -0.6865   0.3401   0.5436   2.1216

Coefficients:
                        Estimate Std. Error z value Pr(>|z|)
(Intercept)            -0.070605   0.295367  -0.239 0.811074
cook_inc_risk           0.159333   0.030695   5.191 2.09e-07 ***
diff                   -1.073598   0.076329 -14.065  < 2e-16 ***
same                    1.155582   0.084634  13.654  < 2e-16 ***
scaled_comp_challenger -0.120528   0.033679  -3.579 0.000345 ***
scaled_comp_incumbent   0.028296   0.033611   0.842 0.399852
inc_tenure             -0.004940   0.005738  -0.861 0.389221
inc_age                 0.001498   0.005615   0.267 0.789638
---

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2831.9  on 2088  degrees of freedom
Residual deviance: 1911.9  on 2081  degrees of freedom
AIC: 1927.9
```

```
Number of Fisher Scoring iterations: 5
```

Notation: I tested the coefficients including tenure_square and age_square, then the optimum log-likelihood is much smaller than the one without these two factors. So I set the coefficients of them as zero.
the standard deviation of intercept is much smaller than the original one in the paper, while other factors have larger standard deviations. The coefficients of intercept, comp_challenger, comp_incumbent, inc_tenure and incumbent_age are larger than the original one. The coefficient of cook_inc_risk becomes positive. The coefficients of diff and same are similar. Except that the standard error of intercept is smaller, other standard errors don't change so much.
Assumptions: all respondents are independent especially in regard to year and state. Also we assume the comp_challenger is a continuous outcome instead of order outcome.

14. (8pt) Estimate the same model, but using the standard errors clustered by state-election. Does the result confirm their original finding?

```
library(sandwich)
library(zoo)
library(lmtest)
#regenerate a data frame about all the factors.
vote = PS2_data$vote_incumbent
state = PS2_data$state
Senate_2 = data.frame(cook_inc_risk, diff, same, scaled_comp_challenger,
                            scaled_comp_incumbent, inc_tenure,
                            inc_age, vote, state)
#perform glm with binomial and link function, probit
glm_Senate_2 = glm(vote~.-state, data = Senate_2,
                 family = binomial(link = "probit"))
unique_state = unique(state)
m = length(unique_state)
p = ncol(model.matrix(glm_Senate_2))
#calculate score
scores=estfun(glm_Senate_2)
#clustered score
Senate_clust=matrix(NA,nrow=m,ncol=p)
for(j in 1:p)
  Senate_clust[,j]= tapply(scores[,j], state, sum)
meat.cl = (m/(m-1)) * t(Senate_clust) %*% Senate_clust
vcov.cl = vcov(glm_Senate_2)
vcov.cl = vcov.cl %*% meat.cl %*% vcov.cl
```

```
#retest.
round(coeftest(glm_Senate_2, vcov=vcov.cl),4)

z test of coefficients:

                       Estimate Std. Error z value Pr(>|z|)
(Intercept)             -0.0706     0.2880 -0.2452   0.8063
cook_inc_risk            0.1593     0.0347  4.5892   <2e-16 ***
diff                    -1.0736     0.1113 -9.6460   <2e-16 ***
same                     1.1556     0.1143 10.1116   <2e-16 ***
scaled_comp_challenger  -0.1205     0.0429 -2.8074   0.0050 **
scaled_comp_incumbent    0.0283     0.0404  0.6996   0.4842
inc_tenure              -0.0049     0.0086 -0.5735   0.5663
inc_age                  0.0015     0.0066  0.2257   0.8214
---
```

Not exactly. Since using the standard errors clustered by state-election doesn't change the estimation of parameters, parameters differ a little with their finding except intercept. But for their finding , scaled_comp_challenger and scaled_comp_incumbent are both negative which means people are prone not to vote incumbents with good facial competence and to vote challengers with good facial competence. That's unreasonable with common sense. To this point, clustered GLM performs better. Except intercept, their results have overall standard errors than clustered GLM. For their finding, only scaled_comp_challenger are significant with $\alpha = 0.05$, but for the clustered GLM, cook_inc_risk, same(Respondent shares incumbent party) and diff(Respondent shares challenger party) are also significant.

15. (8pt) A very "incompetent looking" white male politician, whose facial competency is 3 standard deviations below the mean, approaches you and asks what vote share he can expect if he runs as a challenger in a senate race. He has not decided on where he will run or which party he will run under, so he just wants to know the average predicted vote share conditional on his appearance. Answer this gentleman's question using the model in the previous problem.

Once, I planned to calculate the expectation through conditional probability, but it needs to know the distribution of all the factors, which we don't know. So I use simulation approximation for the white politicians based on empirical distribution.

```
comp_3_sigma = pnorm(-3)
r_comp_challenger =  qnorm(runif(length(scaled_comp_challenger)) * comp_3_sigma)
Senate_3 = cbind(1, cook_inc_risk, diff, same,
                      scaled_comp_challenger = r_comp_challenger,
```

```
                      scaled_comp_incumbent, inc_tenure,
                      inc_age)
#simulate the vote
prob = pnorm(Senate_3 %*% matrix(glm_Senate_2$coefficients, ncol =1))
y = rbinom(length(prob), 1, prob)
mean(y)
> mean(y)
[1] 0.6701771
```

16. (8pt) Using the same model, compute the average marginal effect of increasing the challenger's facial quality from the 25th percentile to 75th percentile on the probability of voting for the incumbent. Plot the estimated effect size with its 95% confidence interval, and provide a brief interpretation of the result. Be sure to account for the clustering.

```
itera4 = 8000
Senate_4 = data.frame(cook_inc_risk, diff, same,
                 scaled_comp_challenger,
                 scaled_comp_incumbent, inc_tenure,
                 inc_age)
Senate_4_25 = cbind(1, cook_inc_risk, diff, same, qnorm(0.25),
                 scaled_comp_incumbent, inc_tenure,
                 inc_age)
Senate_4_75 = cbind(1, cook_inc_risk, diff, same, qnorm(0.75),
                    scaled_comp_incumbent, inc_tenure,
                    inc_age)
N = nrow(Senate_4)
marginal_effect = matrix(rep(NaN, itera4), ncol =1)
for(i in 1:itera4)
{
  #generate
  index = sample(N, N, replace = TRUE);
  glm_Senate_4 = glm(vote[index]~., data = Senate_4[index,],
                     family = binomial(link = "probit"))
  coefficients = matrix(glm_Senate_4$coefficients, ncol =1)
  prob_25 =  pnorm(Senate_4_25%*%coefficients)
  y_25 = rbinom(length(prob_25), 1, prob_25)
  prob_75 =  pnorm(Senate_4_75%*%coefficients)
  y_75 = rbinom(length(prob_75), 1, prob_75)
  marginal_effect[i,] = mean(prob_75) - mean(prob_25)
```

```
}

dens = density(marginal_effect)
plot(dens, main = "bootstrap")
q_25 = quantile(marginal_effect, 0.025)
q_75 = quantile(marginal_effect, 0.975)
x1 = min(which(dens$x >= q_25))
x2 = max(which(dens$x < q_75))
with(dens, polygon(x=c(x[c(x1, x1:x2, x2)]), y=c(0, y[x1:x2], 0), col = "gray"))


marginal_effect2 = matrix(rep(NaN, itera4), ncol =1)
for (i in 1:itera4){
  beta_new = mvrnorm(1,glm_Senate_2$coefficients, vcov.cl)
  prob_25=pnorm(Senate_4_25%*%beta_new)
  prob_75=pnorm(Senate_4_75%*%beta_new)
  Y_25=rbinom(length(prob_25),1,prob_25)
  Y_75=rbinom(length(prob_75),1,prob_75)
  marginal_effect2[i] = mean(Y_75-Y_25)
}

dens2 = density(marginal_effect2)
plot(dens2, main = "Simulation Approximation")
q_25 = quantile(marginal_effect2, 0.025)
q_75 = quantile(marginal_effect2, 0.975)
x1 = min(which(dens$x >= q_25))
x2 = max(which(dens$x < q_75))
with(dens, polygon(x=c(x[c(x1, x1:x2, x2)]), y=c(0, y[x1:x2], 0), col = "gray"))
```

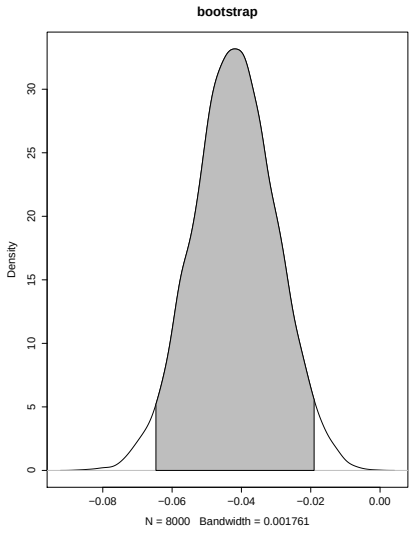

Figure 4: bootstrap

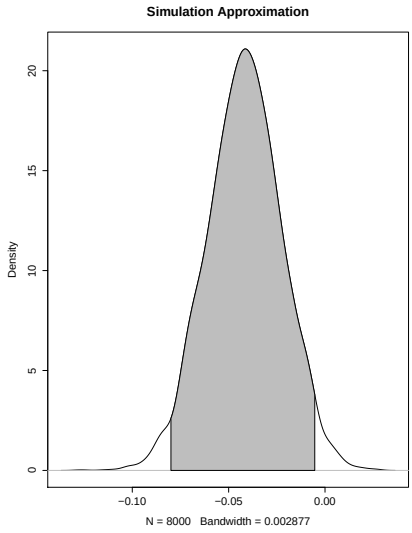

Figure 5: simulation approximation

Bootstrapping and simulation approximation lead to similar results. But simulation approximation considers the clustered state while bootstrapping doesn't use it. From the plots, if a challenger's facial quality increased from the 25th percentile to 75th percentile, the probability that respondents voted for the incumbent, would decrease 4 percent averagely. And Usually, the probability would decrease some percent between 0.005 and 0.08 with significant level $\alpha = 0.05$.