MLE vs OLS

Inhwan Ko

July 29, 2021

```
rm(list=ls())
#install.packages(c("MASS","ggplot2","tidyverse"))

library(MASS)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.0.5
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v tibble  3.1.2     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## v purrr   0.3.4
```

```
## Warning: package 'tibble' was built under R version 4.0.5
```

```
## Warning: package 'tidyr' was built under R version 4.0.5
```

```
## Warning: package 'readr' was built under R version 4.0.5
```

```
## Warning: package 'dplyr' was built under R version 4.0.5
```

```
## Warning: package 'forcats' was built under R version 4.0.5
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::select() masks MASS::select()
```

```
#URL_tile <- c("https://faculty.washington.edu/cadolph/software/tile_0.4.14.tar.gz")
#URL_simcf <- c("https://faculty.washington.edu/cadolph/software/simcf_0.2.17.tar.gz")
#install.packages(URL_tile, repos = NULL, type="source")
#install.packages(URL_simcf, repos = NULL, type="source")

library(simcf)
library(tile)
```

```
## Loading required package: grid
```

Normal MLE

Let's say our dependent variable, y_i, follows this Normal distribution:

$$y_i \sim f_{\mathcal{N}}(\mu_i, \sigma^2)$$

with a mean μ_i defined by the linear equation of covariate matrix, $\mathbf{x}_i$, multiplied by the coefficient matrix, β:

$$\mu_i = \mathbf{x}_i \beta$$

This is called homoskedastic Normal distribution. However, if we allow the variance to vary across observations, say:

$$y_i \sim f_{\mathcal{N}}(\mu_i, \sigma_i^2)$$

when we define both parameters as:

$$\mu_i = \mathbf{x}_i \beta, \quad \sigma_i^2 = \exp(\mathbf{x_i} \gamma)$$

this is now heteroskedastic Normal, which we cannot model with OLS assumptions. The MLE for the heteroskedastic Normal should be:

$$\begin{aligned}
\mathrm{P}\left(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2\right) &= \prod_{i=1}^{n} f_{\mathcal{N}}\left(y_i|\mu_i, \sigma_i^2\right) && \text{[Joint probability]} \\
\mathrm{P}\left(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2\right) &= \prod_{i=1}^{n} \left(2\pi\sigma_i^2\right)^{-1/2} \exp\left[\frac{-\left(y_i - \mu_i\right)^2}{2\sigma_i^2}\right] && \text{[Expressed in Normal distribution]} \\
\log \mathcal{L}\left(\boldsymbol{\beta}, \boldsymbol{\sigma}^2|\mathbf{y}\right) &\propto -\frac{1}{2}\sum_{i=1}^{n} \log \sigma_i^2 - \frac{1}{2}\sum_{i=1}^{n} \frac{\left(y_i - \mathbf{x}_i\boldsymbol{\beta}\right)^2}{\sigma_i^2} && \text{[Converted to log likelihood; simplify]} \\
\log \mathcal{L}(\boldsymbol{\beta}, \boldsymbol{\gamma}|\mathbf{y}) &\propto -\frac{1}{2}\sum_{i=1}^{n} \log \mathbf{z}_i\gamma - \frac{1}{2}\sum_{i=1}^{n} \frac{\left(y_i - \mathbf{x}_i\boldsymbol{\beta}\right)^2}{\exp\left(\mathbf{x}_i\gamma\right)} && \text{[Substitute in systematic components]}
\end{aligned}$$

Let's try to simulate the heteroskedastic Normal data and fit with both the OLS and MLE.

$$\begin{aligned}
y_i &\sim f_{\mathcal{N}}(\mu_i, \sigma_i^2) \\
\mu_i &= 0 + 5x_{1i} + 15x_{2i} \\
\sigma_i^2 &= \exp(1 + 0x_{1i} + 3x_{2i})
\end{aligned}$$

Set up the variables

```
set.seed(2020)
n <- 1000

y <- x0 <- x1 <- x2 <- vector(mode="numeric", 1000)
x0 <- rep(1, n)
x1 <- runif(n, min=0, max=1)
```

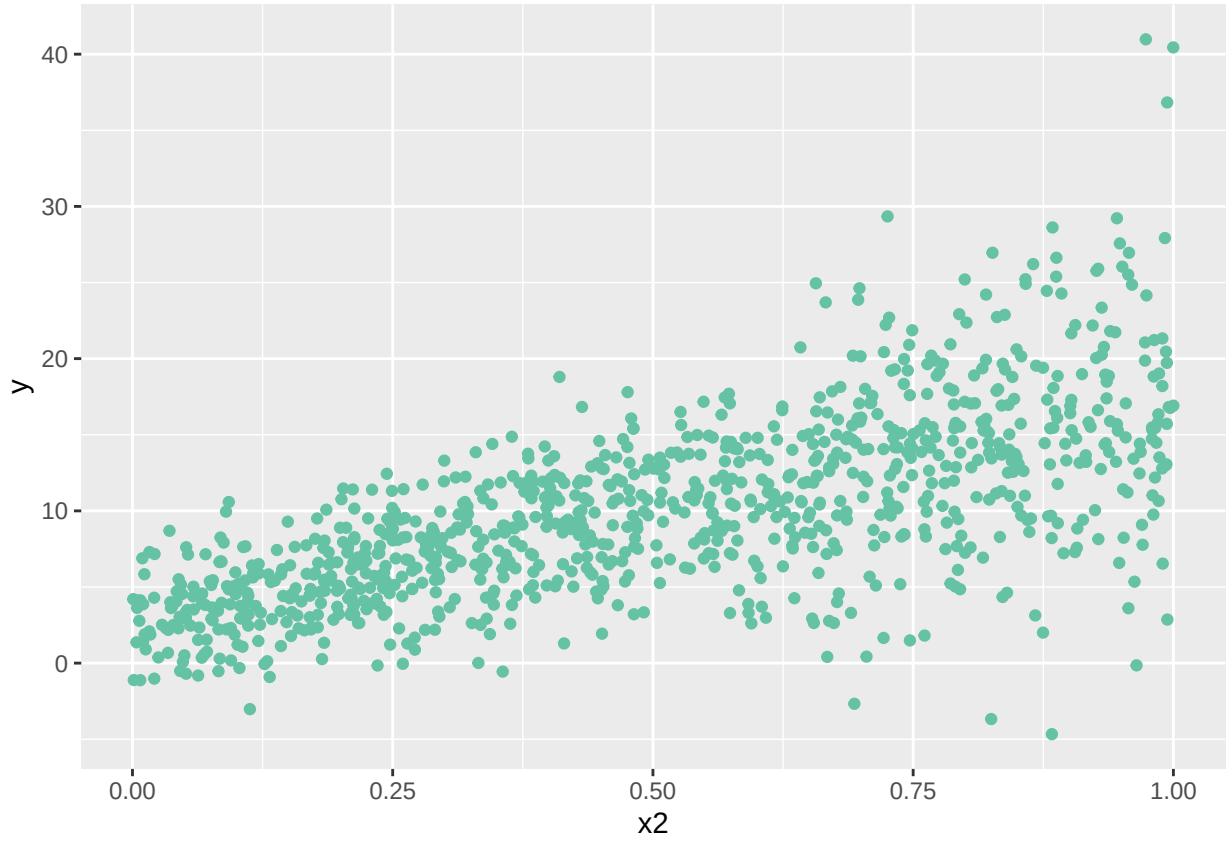
```r
x2 <- runif(n, min=0, max=1)

x <- cbind(x0,x1,x2)

beta <- c(0,5,15)
gamma <- c(1,0,3)
```

Create y variables according to our model

```r
mu <- x %*% beta
sigma2 <- exp(x %*% gamma)

y <- rnorm(n,
           mean=mu, sd=sqrt(sigma2))

data <- cbind(y,x1,x2) %>% as_tibble
```

Let's plot the y_i over x_{2i}.

```r
col <- RColorBrewer::brewer.pal(6, "Set2")

true <- ggplot(data, aes(x2, y)) +
  geom_point(color=col[1]) +
  theme(legend.position = "none")

true
```



What happens if we use OLS for this data?

```
lm <- lm(y ~ x1 + x2, data=data)
summary(lm)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.3971  -2.2322  -0.1876   2.1940  25.9313
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.1633     0.3678   0.444    0.657
## x1            4.8264     0.4626  10.433   <2e-16 ***
## x2           14.8285     0.4660  31.823   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.199 on 997 degrees of freedom
## Multiple R-squared:  0.5157, Adjusted R-squared:  0.5147
## F-statistic: 530.7 on 2 and 997 DF,  p-value: < 2.2e-16
```

What if we use MLE?

```r
llk.hetnormlin <- function(param, y, x, z) {
  x <- as.matrix(x)                       # x (some covariates) as a matrix
  z <- as.matrix(z)                       # z (some covariates) as a matrix
  os <- rep(1, nrow(x))                   # Set the intercept as 1 (constant)
  x <- cbind(os, x)                       # Add intercept to covariates x
  z <- cbind(os, z)                       # Add intercept to covariates z

  b <- param[1 : ncol(x)]                          # Parameters for x
  g <- param[(ncol(x) + 1) : (ncol(x) + ncol(z))] # Parameters for z

  xb <- x %*% b                           # Systematic components for mean
  s2 <- exp(z %*% g)                      # Systematic components for variance

  sum(0.5 * (log(s2) + (y - xb)^2 / s2)) # Likelihood we want to maximize
                                          # optim is a minimizer by default
                                          # To maximize lnL is to minimize -lnL
                                          # so the +/- signs are reversed
}
```

```r
stval <- c(0, 0, 0, 0, 0, 0)

xcovariate <- cbind(x1,x2)

# Run ML, and get the output we need
hetnorm.result <- optim(stval,           # Initial guesses
                        llk.hetnormlin,  # Likelihood function
                        method = "BFGS", # Gradient method
                        hessian = TRUE,  # Return Hessian matrix
                        y = y,           # Outcome variable
                        x = xcovariate, # Covariates x (w/o constant)
                        z = xcovariate  # Covariates z (w/o constant)
                        )
```

```r
pe <- hetnorm.result$par            # Point estimates
round(pe, 3)
```

```
## [1]  0.059  4.846 15.086  1.154 -0.218  2.925
```

```r
vc <- solve(hetnorm.result$hessian) # Var-cov matrix (for computing s.e.)
round(vc, 5)
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,]  0.05610 -0.06059 -0.05223  0.00087 -0.00068 -0.00107
## [2,] -0.06059  0.10771  0.01117 -0.00064  0.00098  0.00032
## [3,] -0.05223  0.01117  0.16069 -0.00184  0.00055  0.00315
## [4,]  0.00087 -0.00064 -0.00184  0.01570 -0.01387 -0.01376
## [5,] -0.00068  0.00098  0.00055 -0.01387  0.02406  0.00406
## [6,] -0.00107  0.00032  0.00315 -0.01376  0.00406  0.02355
```

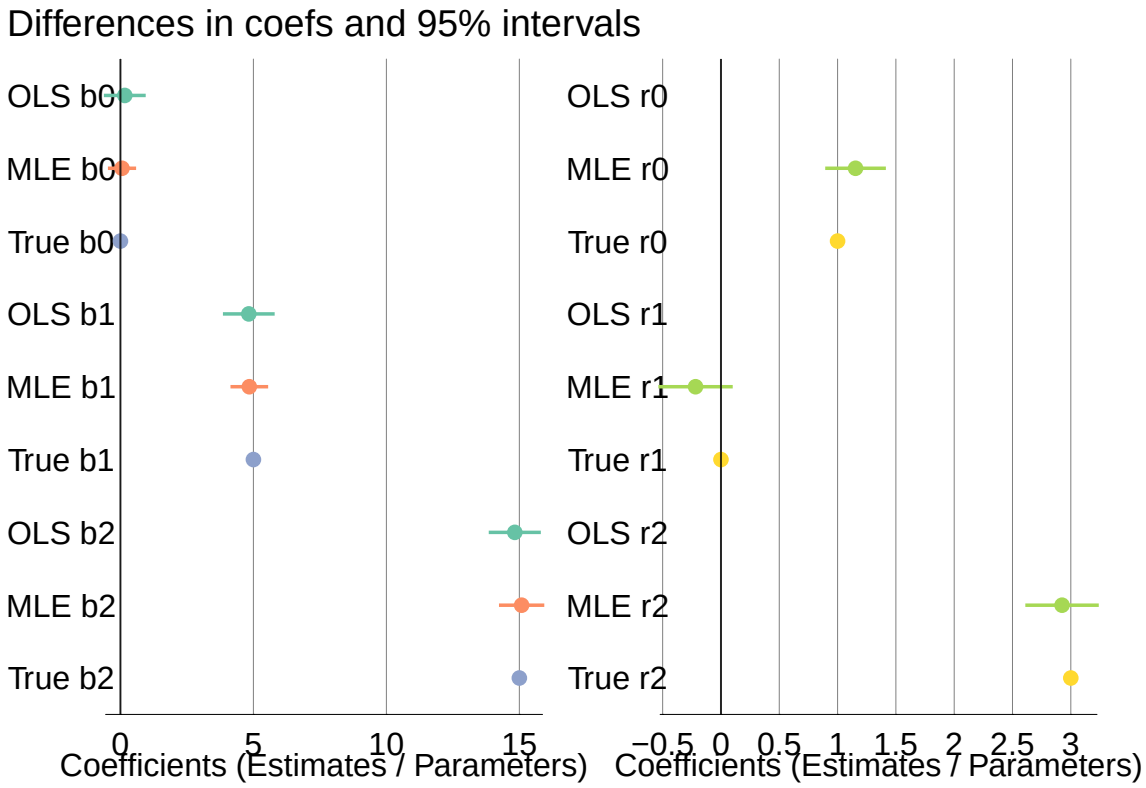
```r
se <- sqrt(diag(vc))                # To compute standard errors (s.e.)
                                    # take the diagonal of the Hessian;
                                    # then take square root
round(se, 3)
```

```
## [1] 0.237 0.328 0.401 0.125 0.155 0.153
```

Comparison between MLE and OLS? Let's show with a ladderplot:

```
lm_coef <- c(summary(lm)$coefficients[,1], rep(NA,3))
lm_se <- c(summary(lm)$coefficients[,2], rep(NA,3))
mle_coef <- round(pe,3)
mle_se <- round(se,3)
true_coef <- c(0,5,15,1,0,3)

lm_lower <- lm_coef+lm_se*-1.96
mle_lower <- mle_coef+mle_se*-1.96
lm_upper <- lm_coef+lm_se*1.96
mle_upper <- mle_coef+mle_se*1.96

pe_all <- c(lm_coef,mle_coef,true_coef)
pe_all <- pe_all[c(1,7,13,2,8,14,3,9,15,4,10,16,5,11,17,6,12,18)]

lower_all <- c(lm_lower,mle_lower,true_coef)
lower_all <- lower_all[c(1,7,13,2,8,14,3,9,15,4,10,16,5,11,17,6,12,18)]

upper_all <- c(lm_upper,mle_upper,true_coef)
upper_all <- upper_all[c(1,7,13,2,8,14,3,9,15,4,10,16,5,11,17,6,12,18)]

result <- data.frame(label = c("OLS b0", "MLE b0", "True b0",
                               "OLS b1", "MLE b1", "True b1",
                               "OLS b2", "MLE b2", "True b2",
                               "OLS r0", "MLE r0", "True r0",
                               "OLS r1", "MLE r1", "True r1",
                               "OLS r2", "MLE r2", "True r2"),
                     pe = pe_all, lower = lower_all, upper = upper_all)


B012 <- ropeladder(x=result[1:9,]$pe, lower=result[1:9,]$lower, upper=result[1:9,]$upper,
                   labels = c("OLS b0", "MLE b0", "True b0",
                              "OLS b1", "MLE b1", "True b1",
                              "OLS b2", "MLE b2", "True b2"),
                   size=0.65,
                   lex=1.75,
                   lineend="square", plot=1, col=col[c(1:3)])

R012 <- ropeladder(x=result[10:18,]$pe, lower=result[10:18,]$lower, upper=result[10:18,]$upper,
                   labels = c("OLS r0", "MLE r0", "True r0",
                              "OLS r1", "MLE r1", "True r1",
                              "OLS r2", "MLE r2", "True r2"),
                   size=0.65,
                   lex=1.75,
                   lineend="square", plot=2, col=col[c(4:6)])

# Make reference line trace for first diffs (at 0)
line1 <- linesTile(x=c(0,0), y=c(0,1), plot=1)
line2 <- linesTile(x=c(0,0), y=c(0,1), plot=2)

# Set tick marks for x axis
```

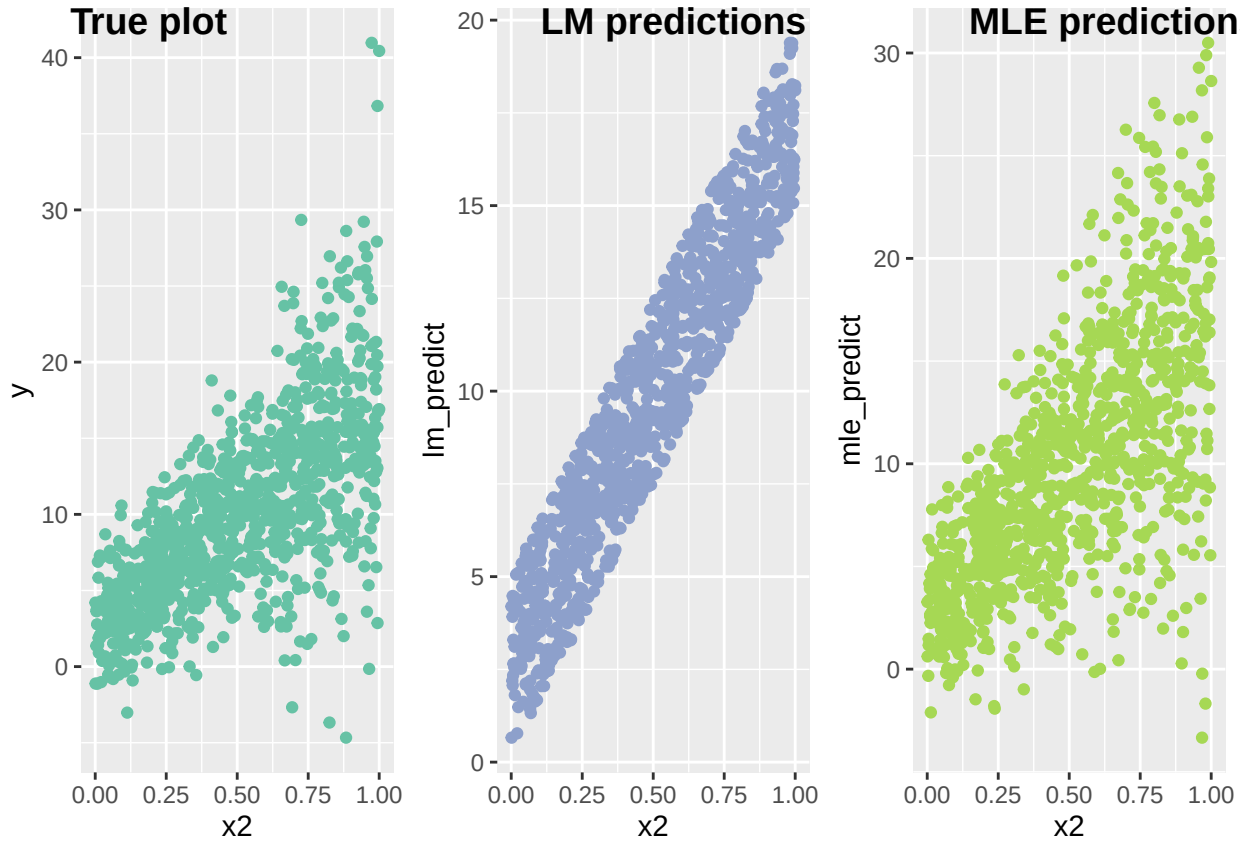
```
xat <- c(0,5,10,15)
xlab <- c(0,5,10,15)

tile(B012, R012, line1, line2,
     plottitle=list(labels="Differences in coefs and 95% intervals"),
     xaxistitle=list(labels="Coefficients (Estimates / Parameters)"),
     width=list(null=5),
     height=list(plottitle=6, xaxistitle=5),
     gridlines=list(type="xt"))
```



Finally, let's compare the prediction result between OLS and MLE:

```
# Linear regression prediction
lm_predict <- predict(lm, data)

mu_sim <- x %*% pe[1:3]
sigma_sim <- exp(x %*% pe[4:6])
sd_sim <- sqrt(sigma_sim)

mle_predict <- rnorm(1000, mu_sim, sd_sim)

plotdata <- data.frame(x1=x1, x2=x2,
                       lm_predict=lm_predict,
                       mle_predict=mle_predict)

lm_plot <- ggplot(plotdata, aes(x2, lm_predict)) +
```

```
  geom_point(color=col[3])

mle_plot <- ggplot(plotdata, aes(x2, mle_predict)) +
  geom_point(color=col[5]) +
  theme(legend.position="none")


final <- ggpubr::ggarrange(true, lm_plot, mle_plot,
                           labels=c("True plot", "LM predictions", "MLE predictions"),
                           ncol=3, nrow=1)


final
```



GLM vs MLE logit

Technically, GLM fits a logit model with MLE methods. The basic difference between the linear and logit model is the existence of an error term:

$$y_i \sim f_{Bern}(\pi_i)$$
$$\pi_i = \text{logit}^{-1}(X_i\beta) = \frac{e^{X_i\beta}}{1+e^{X_i\beta}} = \frac{1}{1+e^{-X_i\beta}}$$

The MLE for the logit model is as follows:

$$\mathcal{L}(\pi|\mathrm{y}) \propto \prod_{i=1}^{n} \pi_i^{y_i}(1-\pi_i)^{1-y_i}$$

$$\mathcal{L}(\beta|\mathrm{y}) \propto \prod_{i=1}^{n} (\frac{1}{1+\exp(-\mathrm{x}_i\beta)})^{y_i}(1-\frac{1}{1+\exp(-\mathrm{x}_i\beta)})^{1-y_i}$$

$$\mathcal{L}(\beta|\mathrm{y}) \propto \prod_{i=1}^{n} (1+\exp(-\mathrm{x}_i\beta))^{-y_i}(1+\exp(\mathrm{x}_i\beta))^{-(1-y_i)}$$

$$\log\mathcal{L}(\beta|\mathrm{y}) \propto \sum_{i=1}^{n} y_i\log(1+\exp(-\mathrm{x}_i\beta)) - (1-y_i)\log(1+\exp(\mathrm{x}_i\beta))$$

Let's just compare the results between GLM and MLE. Let's say we have a true model of a Bernoulli random variable which is:

$$\pi_i = \text{logit}^{-1}(0.5 + x_{1i} + 1.5x_{2_i}) = \frac{e^{X_i\beta}}{1+e^{X_i\beta}} = \frac{1}{1+e^{-X_i\beta}}$$

```
n <- 1000
x1 <- runif(n, 0, 1)
x2 <- runif(n, 0, 1)
y <- pi <- vector(mode="numeric", length=n)

# rbernoulli(1, 0.5)

for (i in 1:n) {
  xb <- 0.5 + x1[i] + 1.5*x2[i]
  pi[i] <- 1 / (1+exp(-xb))
  y[i] <- rbernoulli(1, pi[i])
}

sum(y)/length(y)
```

```
## [1] 0.857
```

Let's fit the model with `glm()`

```
glm <- glm(y ~ x1 + x2, family="binomial")

summary(glm)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5640   0.3342   0.4576   0.6007   1.0301
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.3180     0.2161   1.471 0.141272
```

```
## x1            1.1873     0.3215   3.693 0.000222 ***
## x2            2.1057     0.3450   6.103 1.04e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 820.74  on 999  degrees of freedom
## Residual deviance: 766.10  on 997  degrees of freedom
## AIC: 772.1
##
## Number of Fisher Scoring iterations: 5
```

What about MLE?

```
llk.logit <- function(param,y,x) {
  os <- rep(1,length(x[,1]))
  x <- cbind(os,x)
  b <- param[ 1 : ncol(x) ]
  xb <- x%*%b
  sum( y*log(1+exp(-xb)) + (1-y)*log(1+exp(xb)));
               # optim is a minimizer, so min -ln L(param|y)
}

stval <- glm$coefficients

xcovariates <- cbind(x1, x2)

mle_logit <- optim(stval,
                           llk.logit,
                           method="BFGS",
                           hessian=TRUE,
                           y=y,
                           x=xcovariates)
```

```
pe_2 <- mle_logit$par
se_2 <- sqrt(diag(solve(mle_logit$hessian)))

results_logit <- rbind(pe_2, se_2,
                       summary(glm)$coefficients[,1],
                       summary(glm)$coefficients[,2])
rownames(results_logit) <- c("Optim coefficients", "Optim SE", "GLM coefficients", "GLM SE")

results_logit %>% knitr::kable()
```

	(Intercept)	x1	x2
Optim coefficients	0.3179567	1.1873193	2.1057311
Optim SE	0.2161397	0.3215426	0.3450204
GLM coefficients	0.3179567	1.1873193	2.1057311
GLM SE	0.2161396	0.3215425	0.3450202

They are just same! What a surprise!