STAT 600: Homework 2

Hyungjoon Kim, Colorado State University

February 15, 2024

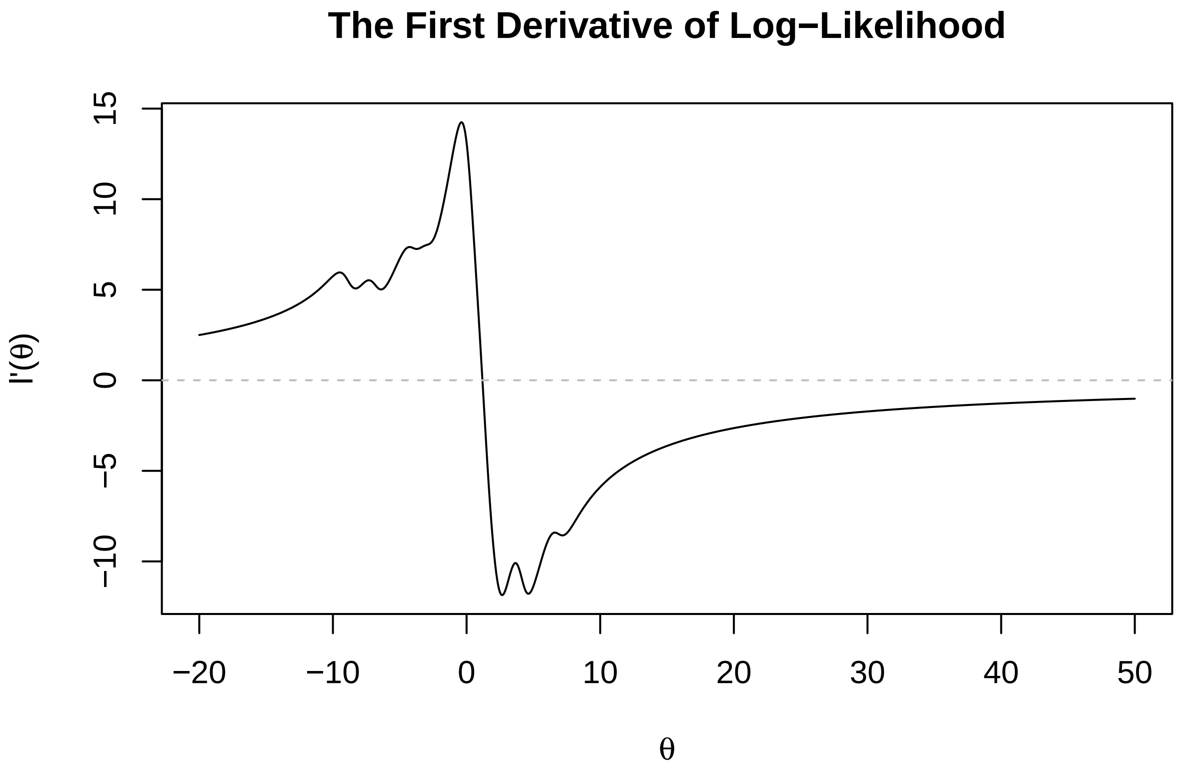
Problem 1

a) Since $X_1, \cdots, X_{25}$ are *i.i.d.*, the likelihood function is the product of density functions, $L(\theta) = \prod_{i=1}^{25} f(x_i \mid \theta)$, which gives

$$\begin{aligned}\ell'(\theta) &= \frac{\partial}{\partial\theta}\log L(\theta)\\ &= -\sum_{i=1}^{25}\frac{\partial}{\partial\theta}\log\pi\left(1+(x_i-\theta)^2\right)\\ &= \sum_{i=1}^{25}\frac{2(x_i-\theta)}{1+(x_i-\theta)^2}\end{aligned}$$

```cpp
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

using namespace Rcpp;

// Calculate the first derivative of the log-density of Cauchy distribution
//
// @param theta double
// @param x double
// @return first derivative of log-density of Cauchy for the given theta and x
// @export
// [[Rcpp::export]]
double get_ell_prime(double theta, double x){
  return 2.0 * (x - theta) / (1.0 + pow(x - theta, 2.0));
}

// Calculate the first derivative of the log-likelihood
// using \ell = \sum log(density)
//
// @param theta double
// @param x arma::vec
// @param ftn Function
// @return the first derivative of the log-likelihood
// @export
// [[Rcpp::export]]
double get_log_likelihood(double theta, arma::vec x, Function ftn){
  int n = x.n_elem;
  double res = 0.0;

  for(int i = 0; i < n; i++){
    res += as<double> (ftn(theta, x(i)));
  }

  return res;
}
```

```
# R code for drawing a graph
data_ <- c(-8.86, -6.82, -4.03, -2.84, 0.14, 0.19, 0.24, 0.27, 0.49, 0.62,
           0.76, 1.09, 1.18, 1.32, 1.36, 1.58, 1.58, 1.78, 2.13, 2.15, 2.36,
           4.05, 4.11, 4.12, 6.83)

x_grid <- seq(-20, 50, length.out = 1000)
y_value <- numeric(1000)

for(i in 1:1000){
  y_value[i] <- get_log_likelihood(x_grid[i], data_, get_ell_prime)
}

plot(x_grid, y_value, type = "l",
     main = "The First Derivative of Log-Likelihood",
     xlab = expression(theta),
     ylab = expression(paste("l'(", theta, ")")))
abline(h = 0, lty = 2, col = "grey")
```



b) 1) Using bisection method, the MLE estimate is 1.1816.

```
// Implement Bisection method to find the root of given `ftn`
//
// @param ftn Function
// @param x arma::vec
// @param a double
// @param b double
// @param tol double
// @return the root of given function, ftn
// @export
// [[Rcpp::export]]
List do_bisection(Function ftn,
                  arma::vec x,
                  double a, double b, double tol){
  double c;
  double f_a;
  double f_c;
  double eps = 1.0;

  int niter = 1;
  while(eps >= tol){
    c = (a + b) / 2.0;
    f_a = get_log_likelihood(a, x, ftn);
    f_c = get_log_likelihood(c, x, ftn);
    if(f_c == 0.0){
      break;
    }else if(f_c * f_a < 0.0){
      eps = abs(b - c) / abs(c);
      b = c;
    }else{
      eps = abs(a - c) / abs(c);
      a = c;
    }
    niter += 1;
  }
  return(List::create(
    Named("root") = c,
    Named("niter") = niter,
    Named("convergence") = eps
  ));
}
```

```
res_bisection <- do_bisection(get_ell_prime, data_, 0.0, 5.0, 0.01)
```

2) Using Newton-Raphson algorithm, the MLE estimate is 1.1879.

```
// Implement Newton-Raphson to find the root of given `ftn`
//
// @param ell_prime Function
// @param ell_double_prime Function
// @param x arma::vec
// @param init double
// @param tol double
// @return the root of given function, ftn
// @export
// [[Rcpp::export]]
List do_newton_raphson(Function ell_prime,
                       Function ell_double_prime,
                       arma::vec x,
                       double init, double tol){
  double new_val;
  double eps = 1.0;

  int niter = 1;
  while(eps >= tol){
    double l1 = get_log_likelihood(init, x, ell_prime);
    double l2 = get_log_likelihood(init, x, ell_double_prime);
    new_val = init - l1 / l2;
    eps = abs(new_val - init) / abs(init);
    init = new_val;
    niter += 1;
  }
  return(List::create(
      Named("root") = init,
      Named("niter") = niter,
      Named("convergence") = eps
  ));
}
```

```
res_newton <- do_newton_raphson(get_ell_prime,
                                get_ell_double_prime,
                                data_, 1.0, 0.01)
```

3) Using Fisher Scoring algorithm, the MLE estimate is 1.1879.

```
// Implement Fisher Scoring to find the root of given `ftn`
//
// @param ell_prime Function
// @param ell_double_prime Function
// @param x arma::vec
// @param init double
// @param tol double
// @return the root of given function, ftn
// @export
// [[Rcpp::export]]
List do_fisher_scoring(Function ell_prime,
                       Function ell_double_prime,
                       arma::vec x,
                       double init, double tol){
  double new_val;
  double eps = 1.0;

  int niter = 1;
  while(eps >= tol){
    double l1 = get_log_likelihood(init, x, ell_prime);
    double I = -get_log_likelihood(init, x, ell_double_prime);
    new_val = init + l1 / I;
    eps = abs(new_val - init) / abs(init);
    init = new_val;
    niter += 1;
  }
  return(List::create(
      Named("root") = init,
      Named("niter") = niter,
      Named("convergence") = eps
  ));
}
```

```
res_fisher <- do_fisher_scoring(get_ell_prime,
                                get_ell_double_prime,
                                data_, 1.0, 0.01)
```

4) Using secant method, the MLE estimate is 1.1889.

```
// Implement Secant method to find the root of given `ftn`
//
// @param ftn Function
// @param x arma::vec
// @param x0 double
// @param x1 double
// @param tol double
// @return the root of given function, ftn
// @export
// [[Rcpp::export]]
List do_secant(Function ftn,
               arma::vec x,
               double x0, double x1, double tol){
  double x2;
  double eps = 1.0;

  int niter = 1;
  while(eps >= tol){
    double f_x0 = get_log_likelihood(x0, x, ftn);
    double f_x1 = get_log_likelihood(x1, x, ftn);
    x2 = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0);
    eps = abs(x2 - x1) / abs(x1);
    x0 = x1;
    x1 = x2;
    niter += 1;
  }
  return(List::create(
      Named("root") = x2,
      Named("niter") = niter,
      Named("convergence") = eps
  ));
}
```

```
res_secant <- do_secant(get_ell_prime, data_,
                        0.0, 2.0, 0.01)
```

c)

```
library(dplyr)

knitr::kable(
  rbind(data.frame(res_bisection),
        data.frame(res_newton),
        data.frame(res_fisher),
        data.frame(res_secant)) %>%
```

```r
  `rownames<-`(c("Bisection", "Newton-Raphson",
                 "Fisher Scoring", "Secant")) %>%
  `colnames<-`(c("$\\hat{\\theta}$", "#(Iterations)", "Convergence"))
)
```

	$\hat{\theta}$	#(Iterations)	Convergence
Bisection	1.181641	10	0.0082645
Newton-Raphson	1.187943	3	0.0022827
Fisher Scoring	1.187943	3	0.0022827
Secant	1.188929	3	0.0076541

d) I used the relative convergence criterion to stop the iteration.

$$\frac{\mid x^{(t+1)} - x^{(t)} \mid}{\mid x^{(t)} \mid} < \epsilon = 0.01$$

As the root of the first derivative of the log-likelihood function is small, so I did not want to be worried about the unit.

e) Using the fact that $\text{Var}\left(\theta\right) = -\mathbb{E}\left[\ell''\left(\theta\right)\right]^{-1}$, we can obtain the variance of our estimate by calculating

$$\text{Var}\left(\hat{\theta}\right) = -\left[\ell''\left(\hat{\theta}\right)\right]^{-1}$$

In my opinion, the estimate calculated by Newton-Raphson algorithm is the best even though its standard error is slightly bigger than that of secant method. The reason is that Newton-Raphson algorithm gives a more stable result. So, the standard error of my estimate is

$$\sqrt{\text{Var}\left(\hat{\theta}_{\text{Newton}}\right)} = \sqrt{-\left[\ell''\left(\hat{\theta}_{\text{Newton}}\right)\right]^{-1}} = 0.2813329$$

However, if we assume that the estimate with smaller standard error is the better estimate, then $\hat{\theta}_{\text{Secant}}$ is the best, because $\sqrt{\text{Var}\left(\hat{\theta}_{\text{Secant}}\right)} = 0.2813195$.

```r
sqrt(-1 / get_log_likelihood(res_newton$root, data_, get_ell_double_prime))
```

```
## [1] 0.2813329
```

```r
sqrt(-1 / get_log_likelihood(res_secant$root, data_, get_ell_double_prime))
```

```
## [1] 0.2813195
```

f) To initialize, I first checked the graph and set the values near the suspected root as the inital values. More specifically, if I choose initial values of 10 and 20, then both of the evaluated values are negative, so bisection method cannot be used. In addition to that, secant method cannot be used because it will diverge. And suppose that my initial value is 10, then both Newton-Raphson and Fisher Scoring cannot be used as the root diverges.

```r
do_bisection(get_ell_prime, data_, 10.0, 20.0, 0.01)
```

```
## $root
## [1] 19.84375
##
## $niter
## [1] 7
##
## $convergence
## [1] 0.007874016
```

```r
do_secant(get_ell_prime, data_, 10.0, 20.0, 0.01)
```

```
## $root
## [1] 1.649634e+154
##
## $niter
## [1] 734
##
## $convergence
## [1] 0
```

```r
do_newton_raphson(get_ell_prime, get_ell_double_prime, data_, 10.0, 0.01)
```

```
## $root
## [1] NaN
##
## $niter
## [1] 256
##
## $convergence
## [1] NaN
```

```r
do_fisher_scoring(get_ell_prime, get_ell_double_prime, data_, 10.0, 0.01)
```

```
## $root
## [1] NaN
##
## $niter
## [1] 256
##
## $convergence
## [1] NaN
```

g) Using 25 more observations, the MLE estimate will be near 1.47. The results obtained by each method are attached below. I still prefer to use Newton-Raphson algorithm to find MLE, as it is stable. However, bisection method yields the estimate with the smallest standard error.

```r
data_ <- c(data_,
           c(-8.34, -1.73, -0.40, -0.24, 0.60, 0.94, 1.05, 1.06, 1.45, 1.50,
             1.54, 1.72, 1.74, 1.88, 2.04, 2.16, 2.39, 3.01, 3.01, 3.08,
             4.66, 4.99, 6.01, 7.06, 25.45))

res_bisection <- do_bisection(get_ell_prime, data_, 0.0, 5.0, 0.01)

res_newton <- do_newton_raphson(get_ell_prime,
                                get_ell_double_prime,
                                data_, 1.0, 0.01)

res_fisher <- do_fisher_scoring(get_ell_prime,
                                get_ell_double_prime,
                                data_, 1.0, 0.01)

res_secant <- do_secant(get_ell_prime, data_,
                        0.0, 2.0, 0.01)

knitr::kable(
  rbind(data.frame(res_bisection),
        data.frame(res_newton),
        data.frame(res_fisher),
        data.frame(res_secant)) %>%
    mutate(se =
             c(sqrt(-1 / get_log_likelihood(res_bisection$root,
                                            data_, get_ell_double_prime)),
               sqrt(-1 / get_log_likelihood(res_newton$root,
                                            data_, get_ell_double_prime)),
               sqrt(-1 / get_log_likelihood(res_fisher$root,
                                            data_, get_ell_double_prime)),
               sqrt(-1 / get_log_likelihood(res_secant$root,
                                            data_, get_ell_double_prime)))) %>%
    `rownames<-`(c("Bisection", "Newton-Raphson",
                   "Fisher Scoring", "Secant")) %>%
    `colnames<-`(c("$\\hat{\\theta}$", "#(Iterations)",
                   "Convergence", "Standard Error"))
)
```

	$\hat{\theta}$	#(Iterations)	Convergence	Standard Error
Bisection	1.474609	10	0.0066225	0.1970263
Newton-Raphson	1.471299	4	0.0000400	0.1970398
Fisher Scoring	1.471299	4	0.0000400	0.1970398
Secant	1.471241	4	0.0050902	0.1970401

Problem 2

Note that the order of convergence β_{Secant} is the solution for the following equation:

$$\lim_{t\to\infty} \mid \epsilon^{(t)} \mid^{1-\beta+1/\beta} = \frac{c^{1+1/\beta}}{d}$$

As $\epsilon^{(t+1)} \approx d^{(t)}\epsilon^{(t)}\epsilon^{(t-1)}$, a 2-step secant method will have $\epsilon^{(t+2)} \approx d^{(t+1)}d^{(t)}\left(\epsilon^{(t)}\right)^2\epsilon^{(t-1)}$. That is, β_{Secant^2} must satisfy

$$\lim_{t\to\infty} \mid \epsilon^{(t)} \mid^{2-\beta+1/\beta} = \frac{c^{1+1/\beta}}{d}$$

Equivalently, we have to find the root of a equation, $2-\beta+1/\beta=0$. Hence, the order of convergence for a 2-step secant method is $\beta_{\text{Secant}^2} = 1+\sqrt{2} \approx 2.41 > 2$. Since the order of convergence for Newton-Raphson algorithm is 2, a 2-step secant method converges faster than Newton-Raphson algorithm.

Problem 3

a) As $Y_i \sim \text{Bernoulli}\left(p_i\right)$, for $\beta = (\beta_0, \beta_1, \beta_2)$,

$$\begin{aligned}
\ell\left(\beta\right) &= \sum_{i=1}^{n}\left[y_i \log p_i + (1-y_i)\log\left(1-p_i\right)\right] \\
&= \sum_{i=1}^{n}\left[y_i \log\left(\frac{\exp\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}\right)}{1+\exp\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}\right)}\right) + (1-y_i)\log\left(\frac{1}{1+\exp\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}\right)}\right)\right] \\
&= \sum_{i=1}^{n}\left[y_i\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}\right) - \log\left(1+\exp\left(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}\right)\right)\right]
\end{aligned}$$

b)

```
// [[Rcpp::export]]
List logit(arma::vec y,
           arma::mat X,
           arma::vec beta_init,
           double tol){
  int n = y.n_elem;
  int p = X.n_cols;
  double eps = 1.0;

  arma::vec beta_new(p, arma::fill::zeros);
  arma::mat hessian(p, p, arma::fill::zeros);
  arma::vec yhat(n, arma::fill::zeros);
  arma::vec Xbeta(n, arma::fill::zeros);

  int niter = 1;
  while(eps >= tol){
    Xbeta = X * beta_init;
    for(int i = 0; i < n; i++){
      yhat(i) = exp(Xbeta(i)) / (1 + exp(Xbeta(i)));
    }
    hessian = X.t() * diagmat(yhat % (1.0 - yhat)) * X;
    beta_new = beta_init + inv(hessian) * X.t() * (y - yhat);

    double d1 = 0.0;
    double d0 = 0.0;
    for(int j = 0; j < p; j++){
      d1 += pow(beta_new(j) - beta_init(j), 2.0);
      d0 += pow(beta_init(j), 2.0);
    }
    eps = d1 / d0;
    beta_init = beta_new;
    niter += 1;
  }

  return(List::create(
    Named("coefs") = beta_new,
    Named("hessian") = hessian,
    Named("yhat") = yhat,
    Named("niter") = niter
  ));
}
```

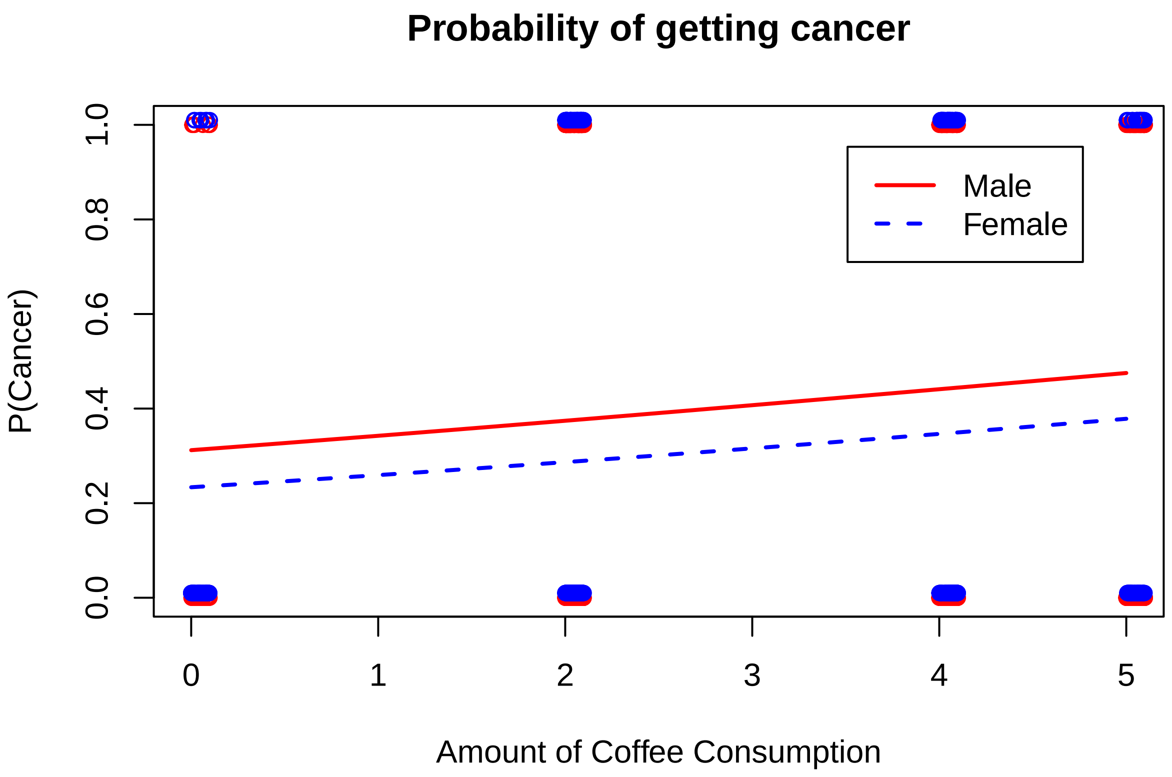
```
# male = 1; female = 0;
df <- data.frame(gender = c(1, 1, 1, 1, 0, 0, 0, 0),
                 n = c(41, 213, 127, 142, 67, 211, 133, 76),
                 r = c(9, 94, 53, 60, 11, 59, 53, 28),
                 x1 = c(0, 2, 4, 5, 0, 2, 4, 5))

df <- rbind(
  as.data.frame(lapply(df, rep, df$r)) %>%
    mutate(y = 1),
  as.data.frame(lapply(df, rep, df$n - df$r)) %>%
    mutate(y = 0)) %>%
```

```r
  select(y, x1, gender)

fit <- logit(df$y, as.matrix(cbind(1, df[, 2:3])),
             c(-1.0, 0.1, 0.3), 0.01)

knitr::kable(
  data.frame(coef = fit$coefs,
             se = sqrt(diag(solve(fit$hessian)))) %>%
    mutate(z = coef / se,
           p = round(pnorm(abs(coef / se),
                           lower.tail = FALSE), 3)) %>%
    `rownames<-`(c("$\\beta_{0}$", "$\\beta_{1}$", "$\\beta_{2}$")) %>%
    `colnames<-`(c("Estimate", "Std. Error",
                   "z-value", "Pr(>abs(z))"))
)
```

	Estimate	Std. Error	z-value	Pr(>abs(z))
β_0	-1.1877880	0.1572146	-7.555199	0.000
β_1	0.1382984	0.0427119	3.237935	0.001
β_2	0.3972508	0.1336597	2.972105	0.001

c) According to the graph and the fitted result given above, we can say that a male has higher probability of cancer than a female, and a person who consumes more coffee has a higher probability of getting cancer.

```r
X <- cbind(1, rep(seq(0, 5, length.out = 100), 2),
           c(rep(1, 100), rep(0, 100)))
Xbeta <- X %*% fit$coefs
prob <- exp(Xbeta) / (1 + exp(Xbeta))

plot(X[X[, 3] == 1, 2], prob[X[, 3] == 1],
     ylim = c(0, 1),
     type = "l", col = "red", lwd = 2,
     xlab = "Amount of Coffee Consumption",
     ylab = "P(Cancer)",
     main = "Probability of getting cancer")
lines(X[X[, 3] == 0, 2], prob[X[, 3] == 0],
      col = "blue", lwd = 2, lty = 2)
points(df[df$gender == 1, 2] + runif(sum(df$gender == 1), 0, 0.1),
       df[df$gender == 1, 1], col = "red")
points(df[df$gender == 0, 2] + runif(sum(df$gender == 0), 0, 0.1),
       df[df$gender == 0, 1] + 0.01,
       col = "blue")
legend(x = "topright", inset = 0.08, legend = c("Male", "Female"),
       lty = c(1, 2), col = c("red", "blue"), lwd = 2)
```



d) Yes, we can say that the amount of coffee consumption and the gender affect the chance of cancer, as p-value of each regression coefficient is very small. In detail, a male has higher probability of getting cancer than a female, and the more a person consumes coffee, the higher probability of getting cancer, because both coefficient estimates are positive.