BIOS7345 Lab 8

Numerical optimization

Sarah Lotspeich

11/14/2019

Logistic regression

Consider a simple logistic regression model where you have a binary outcome Y and a single covariate X on a sample of n observations. This is equivalent to assuming that $Y \sim Bernoulli(p = (1 + \exp\{-(\alpha + \beta X\}))^{-1})$ based on the model

$$\log[\frac{P(Y=1)}{1-P(Y=1)}] = \alpha + \beta X.$$

The function $(1 + \exp\{-(\alpha + \beta X\}))^{-1}$ is referred to as the sigmoid function. Write a function `sigmoid(x,beta,alpha)` that returns this value.

```
# write a function for the sigmoid
sigmoid <- function(x, beta, alpha)
{
  z <- (beta*x + alpha)
  return(1/(1+exp(-z)))
}
```

By definition, the likelihood function of the regression model parameters is

$$L(\alpha, \beta) = \prod_{i=1}^{n} P(Y_i|\alpha, \beta)$$

and thus the log-likelihood is

$$l(\alpha, \beta) = \sum_{i=1}^{n} \log P(Y_i|\alpha, \beta).$$

Based on the distribution of Y (above), what is the detailed form of the log-likelihood?

$$l(\alpha, \beta) = \sum_{i=1}^{n} \log P(Y_i|\alpha, \beta) \tag{1}$$

$$= \sum_{i=1}^{n} \log\{(1 + \exp\{-(\alpha + \beta X\}))^{-1})_i^Y [1 - (1 + \exp\{-(\alpha + \beta X\}))^{-1})]^{(1-Y-i)}\} \tag{2}$$

$$= \sum_{i=1}^{n} Y_i \log\{(1 + \exp\{-(\alpha + \beta X\}))^{-1})\} + (1 - Y_i) \log\{1 - (1 + \exp\{-(\alpha + \beta X\}))^{-1})]^{(1-Y-i)}\} \tag{3}$$

Recognize that $\log P(Y_i|\alpha, \beta) = Y_i \log[S(X_i, \alpha, \beta)] + (1 - Y_i) \log[1 - S(X_i, \alpha, \beta)]$, where we denote the sigmoid function $S(X_i, \alpha, \beta) = (1 + \exp\{-(\alpha + \beta X_i\}))^{-1}$. Use this form to write a `log_likelihood(x, y, beta, alpha)` function that calls your `sigmoid()` function from above.

```
log_likelihood <- function(x, y, beta, alpha)
{
  sigmoid_probs <- sapply(X = x, sigmoid, beta = beta, alpha = alpha)
  return(sum(y*log(sigmoid_probs) + (1-y)*log(1-sigmoid_probs)))
}
```

And now to get the MLEs we take the derivatives of $l(\alpha, \beta)$ with respect to α and β, set them equal to 0, and solve... right? Wrong. There is no closed-form solution for the maximum in this setting. This is an exercise on numerical optimization, afterall.

Newton-Raphson Method

We begin with an initial guess for the MLEs $\hat{\theta}^{(0)} = (\hat{\alpha}^{(0)}, \hat{\beta}^{(0)})$, and update it by a Newton-Raphson "step":

$$\hat{\theta}^{(1)} = \hat{\theta}^{(0)} - H^{-1}(\hat{\theta}^{(0)}) \nabla l(\hat{\theta}^{(0)})$$

where $H(\hat{\theta}^{(0)})$ is the Hessian matrix of the log-likelihood and $\nabla l(\hat{\theta}^{(0)})$ the gradient.

Gradient

The gradient is a vector of partial derivatives of the same dimension as $\hat{\theta}^{(0)}$ defined $\nabla l(\hat{\theta}^{(0)}) = \left[\frac{\partial}{\partial \alpha} l(\theta) \quad \frac{\partial}{\partial \beta} l(\theta)\right]^T \Big|_{\theta=\hat{\theta}^{(0)}}$. Solve for the elements of the gradient and write a `gradient(x, y, beta, alpha)` function that returns the vector (hint: you should also be able to call the `sigmoid()` function).

```
gradient <- function(x, y, beta, alpha)
{
  sigmoid_probs <- sapply(X = x, sigmoid, beta = beta, alpha = alpha)
  grad_mat <- matrix(c(sum((y-sigmoid_probs)*x),
                       sum((y-sigmoid_probs)*1)), nrow = 2, ncol = 1)
  return(grad_mat)
}
```

Hessian

THe hessian is a matrix of double partial derivatives defined

$$H^{-1}(\hat{\theta}^{(0)}) = \left[\begin{array}{cc} \frac{\partial^2}{\partial \alpha^2} l(\theta) & \frac{\partial^2}{\partial \alpha \partial \beta} l(\theta) \\ \frac{\partial^2}{\partial \alpha \partial \beta} l(\theta) & \frac{\partial^2}{\partial \beta^2} l(\theta) \end{array}\right] \Bigg|_{\theta=\hat{\theta}^{(0)}}.$$

Derive the elements of the Hessian and write a `hessian(x, y, beta, alpha)` function that returns this matrix (hint: keep using the `sigmoid` function).

```
hessian <- function(x, y, beta, alpha)
{
  sigmoid_probs <- sapply(X = x, sigmoid, beta = beta, alpha = alpha)
  d1 <- sum((sigmoid_probs*(1-sigmoid_probs))*x*x)
  d2 <- sum((sigmoid_probs*(1-sigmoid_probs))*x*1)
  d3 <- sum((sigmoid_probs*(1-sigmoid_probs))*1*1)
  H <- matrix(c(d1, d2, d2, d3), nrow = 2, ncol = 2, byrow = TRUE)
  return(H)
}
```

Now, we have all the necessary functions to carry out a single iteration of the Newton-Raphson method. To arrive at the MLEs, we need to continue taking the Newton-Raphson "steps" until we reach convergence (i.e. until two consecutive iterations' estimates are within some small tolerance).

Put it all together

Use the functions you've created in the preceding sections to write your own logistic regression function called `my_logreg()`. In addition to your data `x` and `y`, your should take in arguments for the initial values `alpha0` and `beta0`, as well as the tolerance `TOL` and maximum number of iterations before the algorithm terminates `MAX_ITER`.

```
my_logreg <- function(x, y, alpha0 = 0, beta0 = 0, TOL = 1e-4, MAX_ITER = 200)
{
  alpha <- alpha0
  beta <- beta0
  Delta_l <- Inf
  l <- log_likelihood(x, y, beta, alpha)
  iter <- 0
  while(abs(Delta_l) > TOL & iter < MAX_ITER)
  {
    iter <- iter + 1
    g <- gradient(x, y, beta, alpha)
    hess <- hessian(x, y, beta, alpha)
    H_inv <- solve(hess)

    Delta <- H_inv %*% g
    Delta_beta <- Delta[1]
    Delta_alpha <- Delta[2]

    alpha <- alpha + Delta_alpha
    beta <- beta + Delta_beta

    l_new <- log_likelihood(x, y, beta, alpha)
    Delta_l <- l - l_new
    l <- l_new
  }
  return(matrix(c(alpha, beta), nrow =  2))
}
```

Test it out

Use the chunk below to simulate data. Then, compare the results from the `my_logreg()` function to those from `glm()`.

```
N <- 100
alpha <- 1
beta <- 2
X <- rnorm(N)
p <- 1/(1+exp(-(alpha + beta*X)))
Y <- rbinom(N, 1, p)

my_logreg(x = X, y = Y, alpha0 = 0, beta0 = 0)
```

```
##          [,1]
## [1,] 1.263184
## [2,] 1.938375
```

```
glm(Y~ X, family = "binomial")
```

```
##
## Call:  glm(formula = Y ~ X, family = "binomial")
##
## Coefficients:
## (Intercept)            X
##       1.263        1.938
##
## Degrees of Freedom: 99 Total (i.e. Null);  98 Residual
## Null Deviance:       130.7
## Residual Deviance: 94.39     AIC: 98.39
```

Starting values

In the test above, we use "non-informative" initial values; both `alpha0` and `beta0` were set to 0. We know that our data are generated such that the true values of α and β are 1 and 2, respectively. Test your function using initial values that are closer to the truth (`alpha0`=0.5 and `beta0`=1.5) and farther from the truth (`alpha0`=-1 and `beta0`=-1).

```
# test with initial values closer to the truth
my_logreg(x = X, y = Y, alpha0 = 0.5, beta0 = 1.5)
```

```
##          [,1]
## [1,] 1.263184
## [2,] 1.938375
```

```
# test with initial values farther from the truth
my_logreg(x = X, y = Y, alpha0 = -1, beta0 = -1)
```

```
##          [,1]
## [1,] 1.263184
## [2,] 1.938375
```

What do you observe about the estimates? We arrive at the correct coefficients regardless of what we choose for our starting values. So how should we choose them? You can sometimes make efficiency gains in choosing more informative initial values. Use `system.time()` to compare the performance of the three settings of initial values from above.

```
system.time(my_logreg(x = X, y = Y, alpha0 = 0, beta0 = 0))
```

```
##    user  system elapsed
##   0.010   0.000   0.014
```

```
system.time(my_logreg(x = X, y = Y, alpha0 = 0.5, beta0 = 1.5))
```

```
##    user  system elapsed
##   0.003   0.000   0.003
```

```
system.time(my_logreg(x = X, y = Y, alpha0 = -1, beta0 = -1))
```

```
##    user  system elapsed
##   0.004   0.000   0.005
```

References

-[https://www.stat.cmu.edu/~cshalizi/350/lectures/26/lecture-26.pdf]