STAT 428 Homework 6

Yiming Gao (NetID: yimingg2)

2017/4/2

Contents

Exercise 1

We have data in pairs (x_i, y_i) for $i = 1, 2, ..., 25$. Conditional on x_i, y_i is Bernoulli with success probability

$$p_i = P[y_i = 1|x_i] = \frac{e^{\beta_0+\beta_1 x_i}}{1 + e^{\beta_0+\beta_1 x_i}}$$

and the log-likelihood function is

$$\ell(\beta) = \sum_{i=1}^{n}[y_i log(p_i) + (1 - y_i) log(1 - p_i)]$$

First we define the log-likelihood function and input the given data in the following cell.

```r
loglik <- function(beta0, beta1, x,  y) {
  prob = exp(beta0 + beta1 * x) / (1 + exp(beta0 + beta1 * x)) # success probability
  f = sum(y * log(prob) + (1 - y) * log(1 - prob)) # log-likelihood
  return(f)
}

X = c(1.34, -1.38, -0.19, -0.44, 1.90, -0.80, 0.91, 0.26, 1.37, -1.62, -0.96, 1.90, 0.99, 1.96

Y = c(1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0)

# likelihood at the given data
lik_beta <- function(beta) {
  loglik(beta[1], beta[2], X, Y)
}
```

Part a

Use the function `optim()`to compute $\hat{\beta}$ using initial value (.25,.75).

Note that by default this function performs minimization, but it will maximize if `control$fnscale` is negative.

```
optim1 = optim(c(0.25, 0.75), lik_beta, control = list(fnscale = -1))
beta.hat = optim1$par
lik = optim1$value
beta.hat
```

```
## [1] 0.1159 1.0286
```

Starting with initial value (0.25, 0.75), the estimated $\hat{\beta}$ is 0.1159, 1.0286 and the log-likelihood at that point is -13.0024.

Part b (bonus)

We will find out the next value when using the Newton-Raphson algorithm.

```
library(numDeriv)
init = c(0.25, 0.75)
# numerical gradient of a function, i.e., score function
grad = grad(lik_beta, init)
hess_matrix = hessian(lik_beta, init) # Hessian matrix
next_value = init - solve(hess_matrix) %*% grad
next_value
```

```
##         [,1]
## [1,] 0.1269
## [2,] 0.9985
```

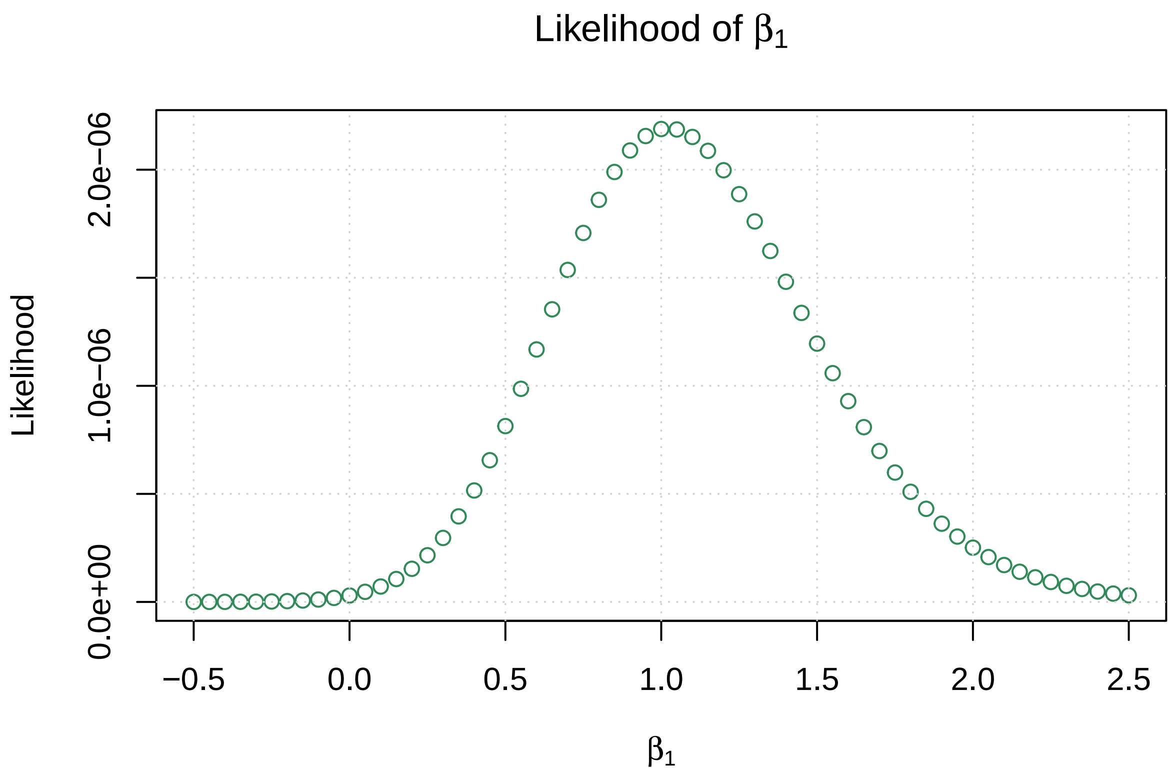
The next value for estimated β is (0.1269, 0.9985).

Part c

Assume that $\beta_0 = 0$, and plot the likelihood function $L(\beta_1)$ as a function of β_1.

```r
function_c <- function(beta1) {
  loglik(0, beta1, X, Y)
}

beta1 = seq(-0.5, 2.5, 0.05)
lbeta1 = sapply(beta1, function_c)
```



Part d

From the plot in part c, we will use 1 as the initial value of β_1, and assume $\beta_0 = 0$ to find $\hat{\beta_1}$ by using three different methods.

- **uniroot()**

```r
# the derivatives of the likelihood functions
derivative <- function(beta){
  p <- exp(beta[1] + beta[2] * X)/(1+exp(beta[1] + beta[2] * X))
  deriv <- c(sum(Y - p),sum(X * (Y - p)))
  return(deriv)
}

beta0 = 0
```

```
Lbeta1 = function(beta) {derivative(c(0, beta))[2]}
beta1_hat_uni = uniroot(Lbeta1, c(0, 2))$root
beta1_hat_uni
```

```
## [1] 1.022
```

- **Grid Search**

We calculate `f` 10000 times.

```
grid = seq(0, 2, length.out = 10000)
f = sapply(grid, function(beta) {loglik(0, beta, X, Y)})
beta1_hat_grid = grid[f == max(f)]
beta1_hat_grid
```

```
## [1] 1.022
```

- **Newton-Raphson algorithm**

```
# N-R function
NewtonRaph <- function(f, tol=1e-7, x0, N) {
  h = 1e-7
  i = 1
  p = numeric(N)
  while(i <= N) {
    dfdx = (f(x0 + h) - f(x0)) / h
    x1 = (x0 - (f(x0) / dfdx))
    p[i] = x1
    i = i + 1
    if(abs(x1 - x0) < tol) break
    x0 = x1
  }
  return(p[1: (i-1)])
}

roots = NewtonRaph(Lbeta1, x0 = 1, N = 1000)

beta1_hat_NR = roots[length(roots)]
```

The `uniroot` estimate is 1.0219, the grid search estimate is 1.0219 and the N-R method esimate is 1.0219, which gives same estimates.

Exercise 2

We assume

$$Y_i \sim Poisson(\mu) \quad for \quad i = 1, 2, ..., k$$

$$Y_i \sim Poisson(\lambda) \quad for \quad i = k+1, ..., n$$

With some simplifying, the log-likelihood is

$$l(\mu, \lambda) = \sum_{i=1}^{k}[-\mu + y_i log(\mu) - log(y_i!)] + \sum_{i=k+1}^{n}[-\lambda + y_i log(\lambda) - log(y_i!)]$$

```
library(boot)
y = floor(coal[[1]])
y = tabulate(y)
y = y[1851: length(y)]
```

```
# log ikelihood function defined as a function in the parameter and data
loglike_change <- function(k, mu, lambda, y) {
  item1 = -mu + y * log(mu) - log(factorial(y))
  item2 = -lambda + y * log(lambda) - log(factorial(y))
  l = sum(item1[1: k]) + sum(item2[(k+1): length(y)])
  return(-l) # optim() performs minimization by default
}

# Likelihood at the given k, data
LL_given <- function(param, k) {
  loglike_change(k, param[1], param[2], y)
}
```

First we set k is the length of half of the data, and initial value for (μ, λ) is $(2.5, 1)$.

```
# now we set k and initial values
k = length(y)/2 # k = 56
param0 = c(2.5, 1)

optim(param0, LL_given, k = k)$par
```

```
## [1] 2.5178 0.8927
```

For the given dataset, we can find the MLE of (μ, λ) given k is the midpoint of timeline is (2.5178, 0.8927).

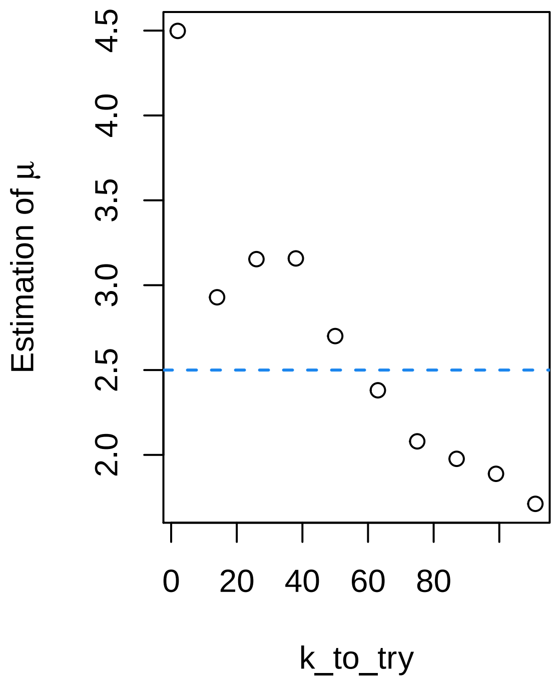
Just for fun, let's learn how MLE of parameters change as k changes. We set 10 k's here, ranging from 2 to total years minus 1.

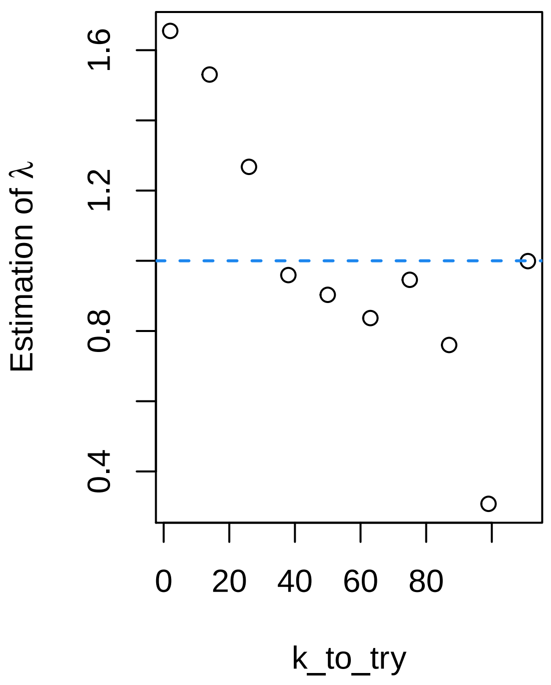
```
k_to_try = round(seq(2, length(y)-1, length.out = 10))

MLE_matrix = matrix(0, nrow = length(k_to_try), ncol = 2)
for (i in 1:10) {
  ki = k_to_try[i]
  MLE_matrix[i, ] = optim(param0, LL_given, k = ki)$par
}

rownames(MLE_matrix) = paste("k =", k_to_try)
print(MLE_matrix)
```

```
##            [,1]   [,2]
## k = 2   4.498 1.6548
## k = 14  2.929 1.5306
## k = 26  3.153 1.2677
## k = 38  3.158 0.9593
## k = 50  2.700 0.9030
## k = 63  2.381 0.8368
## k = 75  2.080 0.9458
## k = 87  1.977 0.7602
## k = 99  1.889 0.3077
## k = 111 1.712 0.9991
```





```r
# mean of estimates
apply(MLE_matrix, 2, mean)
```

```
## [1] 2.648 1.017
```

From the plot we notice that the estimates of parameters fall as k goes up, and when k is somewhere around midpoint, both estimates will come close to the true value $(2.5, 1)$.

Then we calculate the mean of estimates for (μ, λ) is (2.6476, 1.0165).