Lab02

Thijs Quast

25-1-2019

Contents

Question 1 - Optimizing a model parameter

1.1

```
data <- read.csv2("mortality_rate.csv")
data$LMR <- log(data$Rate)
```

```
n <- dim(data) [1]
set.seed(123456)
id <- sample(1:n, floor(n*0.5))
train <- data[id ,]
test <- data[-id ,]
```

1.2

```
X <- as.matrix(train$Day)
Y <- as.matrix(train$LMR)
Xtest <- as.matrix(test$Day)
Ytest <- as.matrix(test$LMR)
```

```
myMSE <- function(lambda, pars, iterCounter = FALSE){
  fit <- loess(pars$Y ~ pars$X, enp.target = lambda)
  predict <- predict(fit, Xtest, se=TRUE)$fit

  n <- length(Ytest)

  mse <- c()
  for (i in 1:n){
    mse[i] <- (Ytest[i] - predict[i])^2
  }

  predictive_mse <- (1/n) * sum(mse)

  if(iterCounter){
    if(!exists("iterForMyMSE")){
      assign("iterForMyMSE",
             value = 1,
             globalenv())
    } else {
      currentNr <- get("iterForMyMSE")
      assign("iterForMyMSE",
             value = currentNr + 1,
             globalenv())
    }
  }

  return(predictive_mse)
}
```

1.3

```
input_list <- list(X = X, Y = Y, Xtest = Xtest, Ytest = Ytest)
```

```
lambdas <- seq(from = 0.1, to = 40, by = 0.1)
```

```
result <- c()
for (lambda in lambdas){
  result[lambda/0.1] <- myMSE(lambda, input_list)
}

df <- as.data.frame(cbind(result, lambdas))
colnames(df) <- c("MSE", "lambda")
```

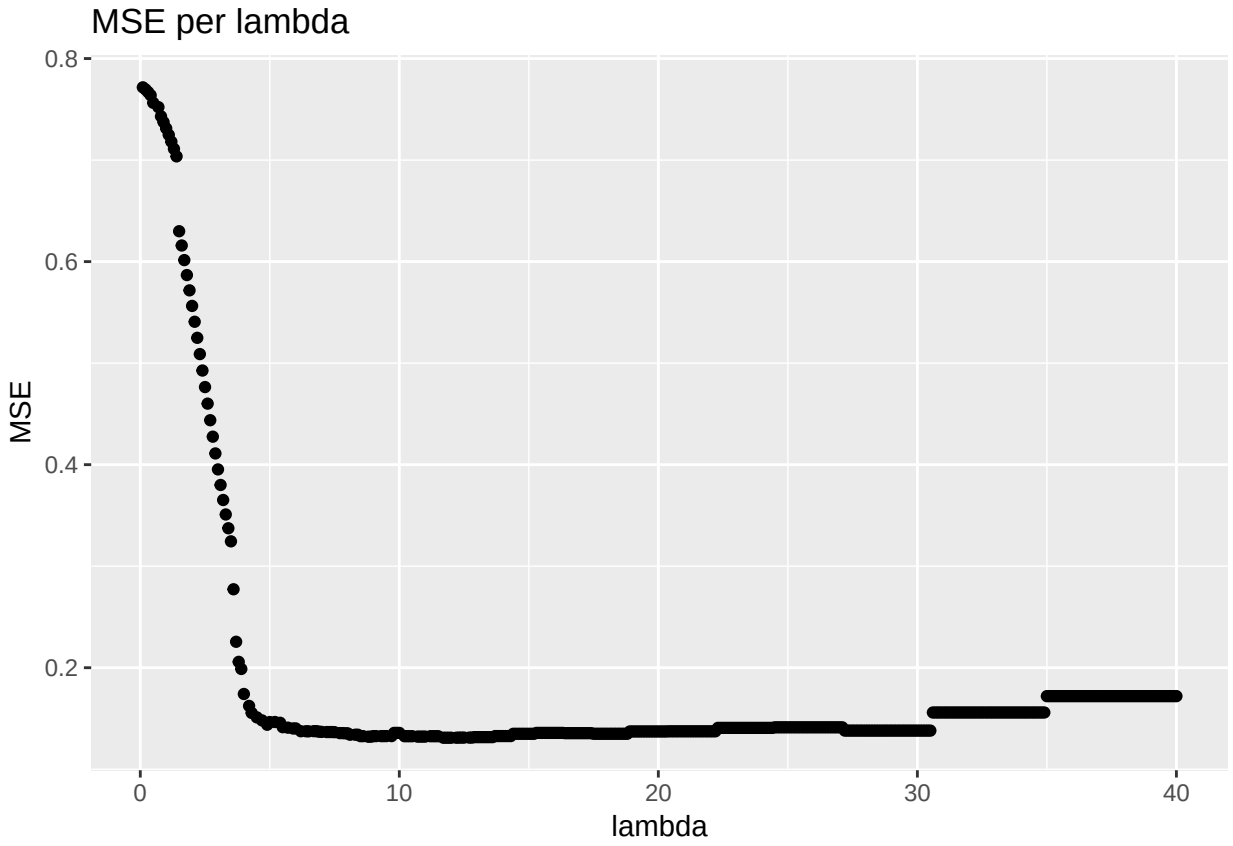
It is a bad idea to optimize likelihood instead of log-likelihood because the log likelihood is easier to differentiate. We can maximize the log likelihood because this function is monotonoically increasing, therefore the optimum of the likelihood occurs at the same point as the optimum for the log likelihood.

(source: https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdbb1)

1.4

```
library(ggplot2)
plot <- ggplot(df, aes(x = lambda, y = MSE)) +
  geom_point() +
  ggtitle("MSE per lambda")
plot
```

```
## Warning: Removed 24 rows containing missing values (geom_point).
```




```
lambdas[which.min(df$MSE)]
```

```
## [1] 11.7
```

The optimal value of lambda is 11.7. Since the sequence is from 0.1 to 40, we run the function a total of 400 times.

1.5

```
optimize(myMSE, pars = input_list, lambda, lower = 0.1, upper = 40, tol = 0.01,
                  maximum = FALSE, iterCounter=TRUE)
```

```
## $minimum
## [1] 10.69361
##
## $objective
## [1] 0.1321441
```

```
print(iterForMyMSE)
```

```
## [1] 18
```

18 function iterations were required to find the optimum, this is a significiant lower number than running it through all the 400 possibilities as in question 1.4.

1.6

```
optim(35, myMSE, pars = input_list, method = c("BFGS"), iterCounter = TRUE)
```

```
## $par
## [1] 35
##
## $value
## [1] 0.1719996
##
## $counts
## function gradient
##        1        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
print(iterForMyMSE)
```

```
## [1] 21
```

The number of iterations of this optimization is 3. For some reason, R stores the number of iterations as an addition to the previously specified iterForMyMSE. Therefore the number of iterations can be found by 21 - 18 = 3.

Since the number of iterations in question 1.6 is much lower, we would say this method is more efficiently in finding the optimum. Also because a starting value lambda = 35 is specified.

Question 2 - Maximum Likelihood

2.1

```
rm()
load("data.RData")
```

2.2

Log-likelihood function:

$$log(L(\mu, \sigma)) = -\frac{100}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{100}(x_i - \mu)^2$$

Computing derivatives:

$$0 = \frac{\partial}{\partial\mu}\log(L(\mu, \sigma)) = 0 - \frac{-2n(\bar{x} - \mu))}{2\sigma^2}$$

Solving results in:

$$\hat{\mu} = \bar{x} = \sum_{i=1}^{100} \frac{x_i}{n}$$

$$\hat{\sigma^2} = \frac{1}{n} \sum (x_i - \mu)^2$$

(source: https://en.wikipedia.org/wiki/Maximum_likelihood_estimation)

```
mu <- mean(data)
sigma <- sqrt(var(data))
mu
```

```
## [1] 1.275528
```

```
sigma
```

```
## [1] 2.016082
```

2.3

Here I create the minus log-likelihood function based on the given parameters.

```
minus_log_likelihood <- function(parameters){
  x_minus_mu <- c()
  mu <- parameters[1]
  sigma <- parameters[2]

  for (i in 1:100){
    x_minus_mu[i] <- data[i] - mu
  }

  outcome <- (100/2)*log(2*pi*sigma^2) + (1/(2*sigma^2))*sum(x_minus_mu^2)
  return(outcome)
}
```

```
optim(c(0,1), fn = minus_log_likelihood, gr = NULL, method = c("BFGS"))
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       37       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
optim(c(0,1), fn = minus_log_likelihood, gr = NULL, method = c("CG"))
```

```
## $par
## [1] 1.275528 2.005977
```

```
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      297       45
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Specifying the gradient. The gradient is the derivative of the log-likelihood function

```
gradient <- function(parameters){
  mu <- parameters[1]
  sigma <- parameters[2]
  n <- 100

  x_minus_mu <- c()
  for (i in 1:100){
    x_minus_mu <- data[i] - mu
  }

  gr_mu <- -(n/(sigma^2)*x_minus_mu)
  gr_sigma <- -(-100/(2*sigma^2)*(1-(1/(sigma^2))*(x_minus_mu^2)))

  outcome <- c(gr_mu, gr_sigma)
  return(outcome)
}
```

(source: http://www.notenoughthoughts.net/posts/normal-log-likelihood-gradient.html)

```
optim(c(0,1), fn = minus_log_likelihood, gr = gradient, method = c("BFGS"))
```

```
## $par
## [1]  1.140850 -2.279175
##
## $value
## [1] 213.1815
##
## $counts
## function gradient
##       60        5
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
optim(c(0,1), fn = minus_log_likelihood, gr = gradient, method = c("CG"))
```

```
## $par
## [1]  2.365791 -5.158382
```

```
##
## $value
## [1] 265.751
##
## $counts
## function gradient
##      145       13
##
## $convergence
## [1] 0
##
## $message
## NULL
```

2.4

Yes the functions converged in all cases. The first two optimizations are without any gradient specification. The optimal parameter values with BFGS method are 1.275528, 2.005977 and requires 37 function evaluations and 15 gradient evaluations. For the CG method, optimal parameters are 1.275528 2.005977 and requires 297 function evaluations and 45 gradient evaluations.

When specifying the gradient, the BFGS methods arrives at 1.140850, -2.279175 for optimal parameter values and requires 60 function evaluations and 5 gradient evaluations. The CG method arrives at 2.365791, -5.158382 for optimal parameter values and requires 145 function evaluations and 13 gradient evaluations.

Given the findings, I would say the settings with gradient and BFGS method are the best, because in general I think if possible specifying the gradient is better. From the BFGS and CG method with gradient specification, the BFGS method requires the least amount of both function and well as gradient evaluations.