Computational Statistics (732A90) Lab2

Anubhav Dikshit(anudi287) and Thijs Quast(thiqu264)

23 January 2019

Contents

Question 1: Optimizing a model parameter

1. Import this file to R and add one more variable LMR to the data which is the natural logarithm of Rate. Afterwards, divide the data into training and test sets by using the following code:

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

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

2. Write your own function myMSE() that for given parameters lambda and list pars containing vectors X, Y, Xtest, Ytest fits a LOESS model with response Y and predictor X using loess() function with penalty lambda (parameter enp.target in loess()) and then predicts the model for Xtest. The function should compute the predictive MSE, print it and return as a result. The predictive MSE is the mean square error of the prediction on the testing data. It is defined by the following Equation (for you to implement): predictive MSE = (1/length(test)) * sum (Ytest[i] - fYpred(X[i]))^2 where fYpred(X[i]) is the predicted value of Y if X is X[i]. Read on R's functions for prediction so that you do not have to implement it yourself.

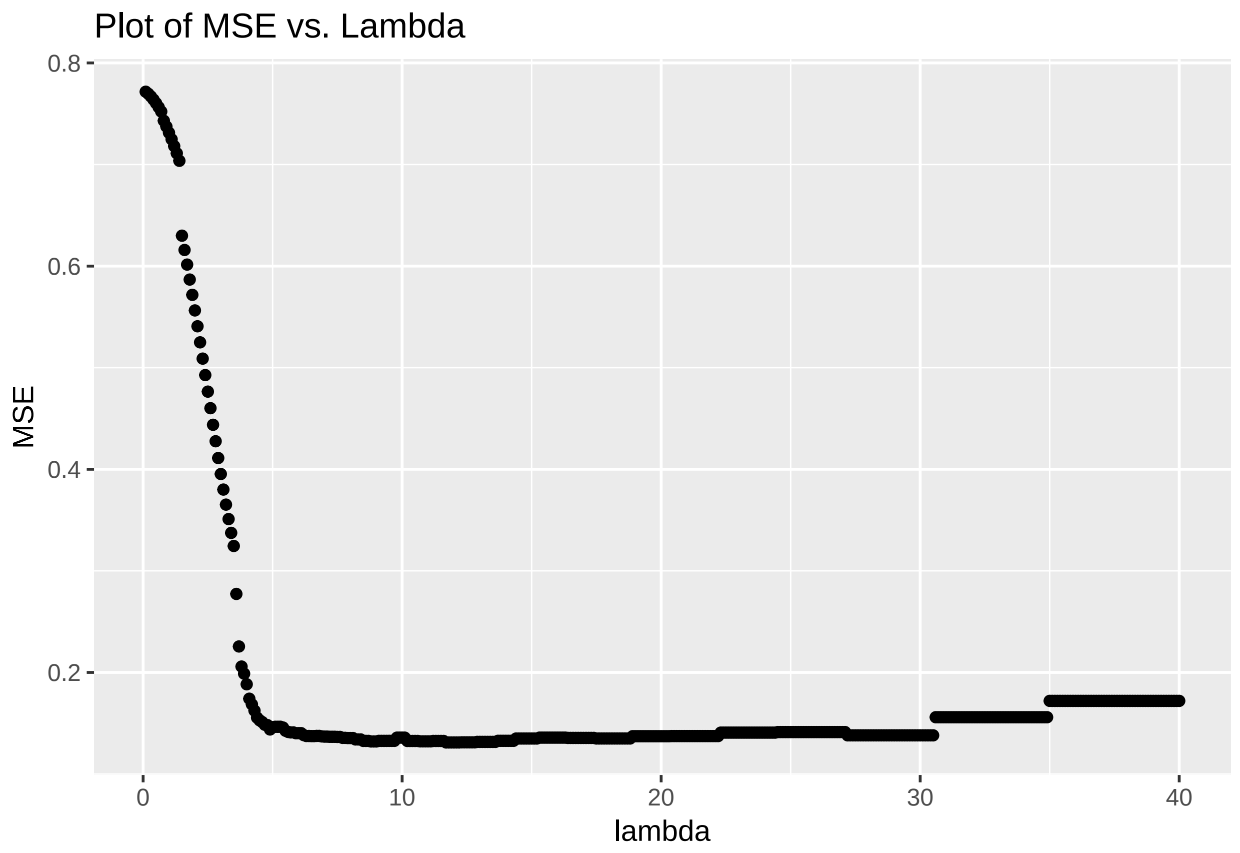
```
myMSE <- function(pars, lambda){

  X <- pars$X
  Y <- pars$Y
  Xtest <- pars$Xtest
  Ytest <- pars$Ytest

model <- loess(Y ~ X, enp.target=lambda)
predicted <- predict(model, Xtest)
n <- length(Ytest)

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

answer_mse <- (1/n) * sum(exp)
     count <<- count + 1
return(answer_mse)
}
```

3. Use a simple approach: use function myMSE(), training and test sets with response LMR and predictor Day and the following lambda values to estimate the predictive MSE values: lambda = 0.1, 0.2, 0.3,. . . ,40

```r
library(dplyr)

X <- train %>% select(c(Day)) %>% as.matrix()
Y <- train %>% select(c(LMR)) %>% as.matrix()

Xtest <- test %>% select(c(Day)) %>% as.matrix()
Ytest <- test %>% select(c(LMR)) %>% as.matrix()

pars <- list(X=X, Y=Y , Xtest=Xtest, Ytest=Ytest)

final <- NULL
count <- 0
for(lambda in seq(from = 0.1, to = 40, by = 0.1)){
temp <- myMSE(pars, lambda=lambda)
temp <- cbind(temp, lambda)
final <- rbind(temp, final)
}

colnames(final) <- c("MSE", "lambda")
```

4. Create a plot of the MSE values versus lambda and comment on which lambda value is optimal. How many evaluations of myMSE() were required (read ?optimize) to find this value?

```r
library(ggplot2)

ggplot(data=data.frame(final), aes(x = lambda, y=MSE)) + geom_point() +
  ggtitle("Plot of MSE vs. Lambda")
```



Analysis: 400 evaluations were run, however given that MSE is lowest when lambda is at around 11.7, we would need 110 iterations to get reach this value of lambda.

5. Use optimize() function for the same purpose, specify range for search [0:1; 40] and the accuracy 0:01. Have the function managed to and the optimal MSE value? How many myMSE() function evaluations were required? Compare to step 4.

```r
count <- 0
optimize(interval = c(0.1, 40), myMSE, pars=pars, tol=0.01)
```

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

```r
cat("iternations: ", count)
```

```
## iternations:  18
```

Answer: Yes the function managed to find the an approximate optimal MSE value which we can visually compare from step4. It took 18 evaluations, this is not the global minima but very close to one. From a practical point of view I would consider this point to be good enough.

6. Use optim() function and BFGS method with starting point lambda = 35 to and the optimal value. How many myMSE() function evaluations were required (read ?optim)? Compare the results you obtained with the results from step 5 and make conclusions.

```r
count <- 0
optim(35, myMSE, pars=pars, method = c("BFGS"))
```

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

```r
cat("iternations: ", count)
```

```
## iternations:  3
```

Answer: 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.

Although the number of iterations is only 3, the lambda value returned is 35. from the graph of lambda vs. MSE we can see that indeed this is a inflection point, hence the number of iterations being less is purely due to starting point.

Secondly the global minimum is around 11 while this method yields 35, thus I would prefer 10.5 over this, however choosing a optimal starting point may change the conclusions.

Question 2: Maximizing likelihood

1. Load the data to R environment.

```r
rm(list=ls())
load("data.RData")
```

2. Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for Mean, Sigma analytically by setting partial derivatives to zero. Use the derived formulae to obtain parameter estimates for the loaded data.

The expression for the log likehood would be:

$$L_x = -(\frac{100}{2}\ln(2\pi) + \frac{100}{2}\ln_(\sigma^2) + \frac{1}{2\sigma^2}\sum_{j=1}^{100}(x_j-\mu)^2)$$

Paritally differenitating with μ we get

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$$

$$\sigma^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j-\mu)^2$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_j-\mu)^2}$$

```
obtained_mean <-  sum(data)/length(data)
obtained_sigma <- sqrt(sum((data - obtained_mean)^2)/length(data))
```

3. Optimize the minus log-likelihood function with initial parameters Mean = 0, Sigma = 1. Try both Conjugate Gradient method (described in the presentation handout) and BFGS (discussed in the lecture) algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log-likelihood?

```
my_log_likehood <- function(pars){
  x<-data
  mu <- pars[1]
  sigma <- pars[2]
  n <- length(x)
answer <- n*0.5*log(2*pi*sigma^2) + (0.5/sigma^2)* sum((x-mu)^2)

return(answer)
}

gradient <- function(pars){
    x<-data
  mu <- pars[1]
  sigma <- pars[2]
  n <- length(x)
  grad_mu <- - (1/sigma^2)* sum(x-mu)
  grad_sig <- (n/sigma) - (1/sigma^3) * sum((x-mu)^2)
  return(c(grad_mu, grad_sig))
}

run1 <- optim(c(0,1), fn = my_log_likehood, gr=NULL, method = c("BFGS"))
run2 <- optim(c(0,1), fn = my_log_likehood, gr=NULL, method = c("CG"))
run3 <- optim(c(0,1), fn = my_log_likehood, gr=gradient, method = c("BFGS"))
run4 <- optim(c(0,1), fn = my_log_likehood, gr=gradient, method = c("CG"))
```

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

4. Did the algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

```r
final <- NULL
final$algorithm <- c("BFGS", "CG", "BFGS+gradient", "CG+gradient")
final$parameters <- rbind(run1$par, run2$par, run3$par, run4$par)
final$counts <- rbind(run1$counts, run2$counts, run3$counts, run4$counts)
final$convergence <- rbind(run1$convergence, run2$convergence, run3$convergence, run4$convergence)
final$value <- rbind(run1$value, run2$value, run3$value, run4$value)


knitr::kable(as.data.frame(final), caption = "Table showing the summary from various optimization techni
```

Table 1: Table showing the summary from various optimization techniques

algorithm	parameters.1	parameters.2	counts.function	counts.gradient	convergence	value
BFGS	1.275527	2.005977	37	15	0	211.5069
CG	1.275528	2.005977	297	45	0	211.5069
BFGS+gradient	1.275528	2.005977	39	15	0	211.5069
CG+gradient	1.275528	2.005977	53	17	0	211.5069

Analysis: Number of function/gradient evaluations taken by BFGS+no gradient supplied -> 37/15

Number of function/gradient evaluations taken by CG+no gradient -> 297/45

Number of function/gradient evaluations taken by BFGS+gradient -> 39/15

Number of function/gradient evaluations taken by CG+gradient -> 53/17

All the functions converged, and the parameters values and the values obtained are practically the same. The difference is only in the number of iterations (runtime) where BFGS without gradient specified is the fastests.

Given the findings, I would say the choosing BFGS without gradient is fastest method however for more complicated cases specifying the gradient might be the optimal choice.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)

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

n=NROW(data)
```

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


myMSE <- function(pars, lambda){

  X <- pars$X
  Y <- pars$Y
  Xtest <- pars$Xtest
  Ytest <- pars$Ytest

model <- loess(Y ~ X, enp.target=lambda)
predicted <- predict(model, Xtest)
n <- length(Ytest)

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

answer_mse <- (1/n) * sum(exp)
     count <<- count + 1
return(answer_mse)
}

library(dplyr)

X <- train %>% select(c(Day)) %>% as.matrix()
Y <- train %>% select(c(LMR)) %>% as.matrix()

Xtest <- test %>% select(c(Day)) %>% as.matrix()
Ytest <- test %>% select(c(LMR)) %>% as.matrix()

pars <- list(X=X, Y=Y , Xtest=Xtest, Ytest=Ytest)

final <- NULL
count <- 0
for(lambda in seq(from = 0.1, to = 40, by = 0.1)){
temp <- myMSE(pars, lambda=lambda)
temp <- cbind(temp, lambda)
final <- rbind(temp, final)
}

colnames(final) <- c("MSE", "lambda")

library(ggplot2)

ggplot(data=data.frame(final), aes(x = lambda, y=MSE)) + geom_point() +
  ggtitle("Plot of MSE vs. Lambda")
count <- 0
optimize(interval = c(0.1, 40), myMSE, pars=pars, tol=0.01)
```

```r
cat("iternations: ", count)
count <- 0
optim(35, myMSE, pars=pars, method = c("BFGS"))
cat("iternations: ", count)

rm(list=ls())
load("data.RData")
obtained_mean <-  sum(data)/length(data)
obtained_sigma <- sqrt(sum((data - obtained_mean)^2)/length(data))


my_log_likehood <- function(pars){
  x<-data
  mu <- pars[1]
  sigma <- pars[2]
  n <- length(x)
answer <- n*0.5*log(2*pi*sigma^2) + (0.5/sigma^2)* sum((x-mu)^2)

return(answer)
}

gradient <- function(pars){
    x<-data
  mu <- pars[1]
  sigma <- pars[2]
  n <- length(x)
  grad_mu <- - (1/sigma^2)* sum(x-mu)
  grad_sig <- (n/sigma) - (1/sigma^3) * sum((x-mu)^2)
  return(c(grad_mu, grad_sig))
}

run1 <- optim(c(0,1), fn = my_log_likehood, gr=NULL, method = c("BFGS"))
run2 <- optim(c(0,1), fn = my_log_likehood, gr=NULL, method = c("CG"))
run3 <- optim(c(0,1), fn = my_log_likehood, gr=gradient, method = c("BFGS"))
run4 <- optim(c(0,1), fn = my_log_likehood, gr=gradient, method = c("CG"))

final <- NULL
final$algorithm <- c("BFGS", "CG", "BFGS+gradient", "CG+gradient")
final$parameters <- rbind(run1$par, run2$par, run3$par, run4$par)
final$counts <- rbind(run1$counts, run2$counts, run3$counts, run4$counts)
final$convergence <- rbind(run1$convergence, run2$convergence, run3$convergence, run4$convergence)
final$value <- rbind(run1$value, run2$value, run3$value, run4$value)


knitr::kable(as.data.frame(final), caption = "Table showing the summary from various optimization techni
```