Optimization using Conjugate-Gradient and BFGS

Sreenand Sasikumar

06/02/2020

Question 1: Optimizing a model parameter

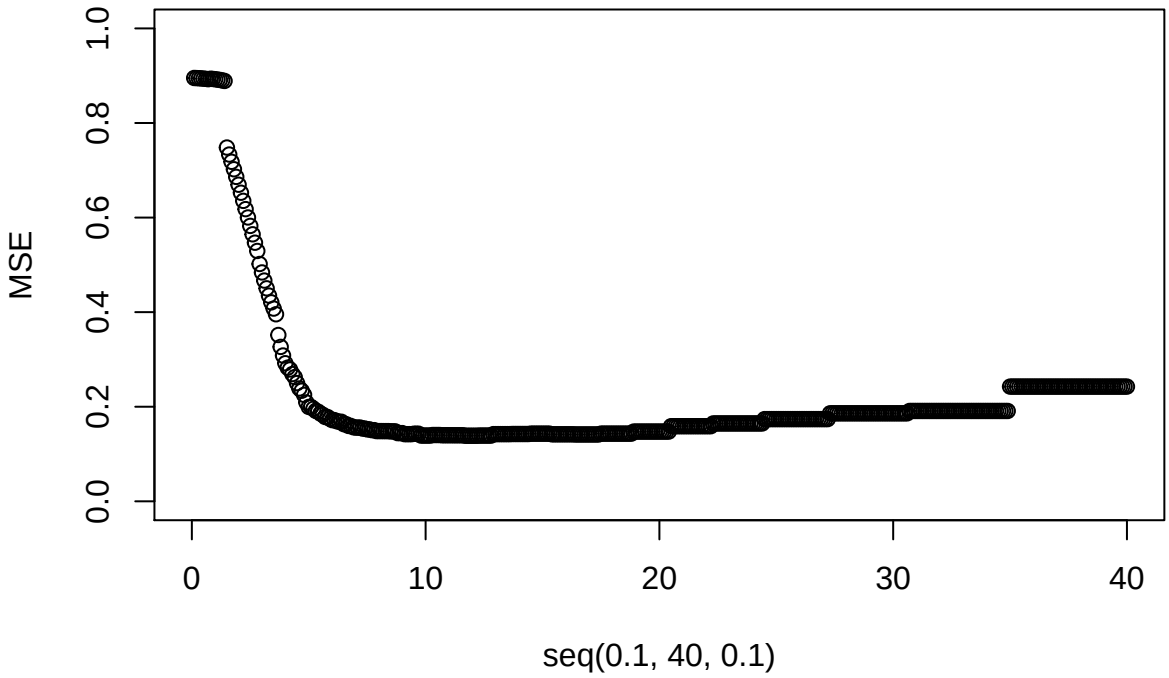
1.1

Importing the data and adding one more variable LMR to the data which is the natural logarithm of Rate.

1.2

Writing a function myMSE() that for given parameters 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 and then predicts the model for Xtest.

1.3,1.4




```
## The minimum MSE obtained is :
```

```
## [1] 0.1386422
```

```
## The optimal lambda values obtained for corresponding optimal MSE are
```

```
## [1] 11.7 11.8 11.9 12.0 12.1 12.2
```

From the plot it is evident that, the optimal lambda is the lambda with minimum Mean Square Error, in this case the Optimal lambda is found to be at 11.7,11.8,11.9,12.0,12.1,12.2. For all of the above lambda the MSE is same that is 0.1386422 and is the least of all. After 400 evaluations, optimal lambda was achieved at 117th evaluation.

1.5

```
## The below are the total number of evaluations in acheiving optimal MSE using optimize() function
```

```
## [1] 0.1430674
## [1] 0.1646174
## [1] 0.1420511
## [1] 0.139655
## [1] 0.1386422
## [1] 0.1420598
## [1] 0.138957
## [1] 0.1386422
## [1] 0.1386422
## [1] 0.1386422
## [1] 0.1386422
## [1] 0.1386422
## [1] 0.1386422
## [1] 0.1386422
## [1] 0.1386422
```

```
## $minimum
## [1] 11.9115
##
## $objective
## [1] 0.1386422
```

The optimize function was almost able to find the optimal MSE value. The MSE value obtained from aptimize function is 0.1386422 which is exactly the same MSE which was acheieved in previous step. The number of evaluations required to calculate this optimal MSE is around 18 which is very less comapred to step 4 evaluation.

1.6

```
## The below are the total number of evaluations in acheiving optimal MSE using optim() function
```

```
## [1] 0.2425965
## [1] 0.2425965
## [1] 0.2425965
```

```
## $par
## [1] 35
##
## $value
## [1] 0.2425965
##
## $counts
## function gradient
```

```
##        1        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The optim function results in optimal MSE value of 0.2425965 which is highest amongst the above two MSE calculations. By using optim function the number of evaluations is just 3 since it evaluates the function from lambda 35.

Comparing all the three evaluations to find optimal MSE, the optimize function does a good job by reducing number of evaluations and also maintaing the close to actual MSE value. The optim function failed in finding global minimum since the lambda is started from 35 and the function assumes that as minimum of all.

Question 2: Maximizing likelihood

2.1

Importing the data into R.

2.2

```
## The estimated value for Mean after deriving maximum likelihood formula and setting partial derivativ
## [1] 1.275528
## The estimated value for Sigma after deriving maximum likelihood formula and setting partial derivati
## [1] 2.005976
## The minus loglikelihood value is
## [1] 211.5069
```

Probablity density function for normal distributionu is given by:

$$f(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(x-\mu)^2/2\sigma^2$$

Log-liklihood for 100 observations is given by:

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

Maximum likelihood estimators for

$$\mu, \sigma$$

analytically by setting partial derivatives to zero is given by the formula:

$$\frac{\partial - (L(f(x \mid \mu, \sigma)))}{\partial \mu} = \frac{1}{\sigma^2}\sum_{n=1}^{100}(x_i - \mu))$$

$$\frac{\partial - (L(f(x \mid \mu, \sigma)))}{\partial \sigma} = -(-\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{n=1}^{100}(x_i - \mu)^2)$$

Setting the derivatives to zero we get,

$$\hat{\mu} = \frac{\sum_{n=1}^{n} (x_i - \mu)}{n}$$

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

2.3

```
## Conjugate Gradient without Gradient
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      180       33
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
## BFGS without Gradient
```

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

```
## Conjugate Gradient with Gradient
```

```
## $par
## [1] 1.275528 2.005976
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
```

```
##       53       17
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
## BFGS with Gradient
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       38       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

It is a bad idea to use maximum likelikehood instead of maximum loglikelihood because, the computations carried out while calculating likelihood might end up in zero values, for example 1 / (some product) ^ (big number in this case 100) results results in 0 which fails the whole purpose of calculating the likelihood.

2.4

```
##             mu_val sigma_val minloglik_val fun_count_val grad_count_val
## CG        1.275528  2.005977      211.5069           180             33
## BFGS      1.275528  2.005977      211.5069            37             15
## CG_grad   1.275528  2.005976      211.5069            53             17
## BFGS_grad 1.275528  2.005977      211.5069            38             15
##           conv_val
## CG               0
## BFGS             0
## CG_grad          0
## BFGS_grad        0
```

All the 4 methods, Conjugate gradient, BFGS with and without gradient function has converged to zero. The best amongst the 4 methods can be decided based on the least number of evluations funcion has taken and in this case, BFGS method without gradient has taken 37 evaluations to converge. It is also to be acknowledged that BGFS method with gradient function has taken 38 evaluations to converge.

Appendix

```
knitr::opts_chunk$set(echo = TRUE,warning = FALSE, message = FALSE)
#1.1
library(readr)
mort_data <- read_csv2("mortality_rate.csv")
n=dim(mort_data)[1]
mort_data[['LMR']] <- log(mort_data$Rate)
```

```r
RNGversion(min(as.character(getRversion()),"3.6.2")) ## with your R-version
set.seed(12345, kind = "Mersenne-Twister", normal.kind = "Inversion")
id=sample(1:n,floor(n*0.5))
train = mort_data[id,]
test = mort_data[-id,]
#pars <- list(train$Day,train$LMR,test$Day,test$LMR)
myMSE <- function(lambda,pars){
  loess.fit <- loess(unlist(pars[2]) ~ unlist(pars[1]), enp.target = lambda)
  loess.pred <- predict(loess.fit, newdata = unlist(pars[3]))
  pred_mse <- (1 / length(unlist(pars[3]))) * sum((unlist(pars[4]) - loess.pred) ^ 2)
  #print(pred_mse)
  return(pred_mse)
}
#myMSE(30,pars)
#1.3
pars <- list(train$Day,train$LMR,test$Day,test$LMR)
#length(unlist(pars[3]))
MSE <- vector()
j=1
for(i in seq(0.1,40,0.1))
{
  MSE[j] <- myMSE(i,pars)
  j <- j+1
}

# 1.4
plot(seq(0.1,40,0.1),MSE, type="p", xlim = c(0,40),ylim = c(0,1))
cat("The minimum MSE obtained is :")
min(MSE)
cat("The optimal lambda values obtained for corresponding optimal MSE are")
which(MSE == min(MSE))*0.1
# 1.5
myMSEopt <- function(lambda,pars)
{
  #print(lambda)
  #print(pars)

  loess.fit <- loess(unlist(pars[2]) ~ unlist(pars[1]), enp.target = lambda)
  loess.pred <- predict(loess.fit, newdata = unlist(pars[3]))
  pred_mse <- (1 / length(unlist(pars[3]))) * sum((unlist(pars[4]) - loess.pred) ^ 2)
  print(pred_mse)
  return(pred_mse)
}
cat("The below are the total number of evaluations in acheiving optimal MSE using optimize() function")
optimize(myMSEopt,seq(0.1,40),tol = 0.01,pars)
#myMSEopt(35,pars)

# 1.6
# myMSEopt(35,pars)
cat("The below are the total number of evaluations in acheiving optimal MSE using optim() function")
optim(par=35,fn=myMSEopt,method = "BFGS",pars=pars)

#2.1
```

```r
load("data.RData")
r_data <- data
mu <- sum(r_data)/length(data)
cat("The estimated value for Mean after deriving maximum likelihood formula and setting partial derivati
mu
sigma <- sqrt(sum((r_data - mu)^2) / length(data))
cat("The estimated value for Sigma after deriving maximum likelihood formula and setting partial deriva
sigma
n <- length(r_data)

minlog <- function(par)
{
  #minuslog <- -(n * log(1/sqrt(2 * pi * par[2]^2))) - (sum((r_data - par[1])^2) / (2 * par[2]^2))
  minuslog <- -(-(n * log(2 * pi) / 2) - (n * log(par[2]^2) / 2) - 1/(2*par[2]^2)*sum((r_data- par[1])^
  return(minuslog)
}
cat("The minus loglikelihood value is")
minlog(par=c(mu,sigma))
gradient <- function(par)
{
  grad_mu <- -(1 / par[2]^2) * sum(r_data - par[1])
  grad_sigma <- -((-n / par[2]) + (sum((r_data - par[1])^2) / par[2]^3))
  return(c(grad_mu,grad_sigma))
}
cat("Conjugate Gradient without Gradient")
CG <- optim(par = c(0,1), fn = minlog, method = "CG")
CG
cat("BFGS without Gradient")
BFGS <- optim(par = c(0,1), fn = minlog, method = "BFGS")
BFGS
cat("Conjugate Gradient with Gradient")
CG_grad <- optim(par = c(0,1), fn = minlog, method = "CG", gr = gradient)
CG_grad
cat("BFGS with Gradient")
BFGS_grad <- optim(par = c(0,1), fn = minlog, method = "BFGS", gr = gradient)
BFGS_grad
overall <- matrix(NA,nrow = 4,ncol = 6)
mu_val <- c(CG$par[1],BFGS$par[1],CG_grad$par[1],BFGS_grad$par[1])
sigma_val <- c(CG$par[2],BFGS$par[2],CG_grad$par[2],BFGS_grad$par[2])
minloglik_val <- c(CG$value,BFGS$value,CG_grad$value,BFGS_grad$value)
fun_count_val <- c(CG$count[1],BFGS$counts[1],CG_grad$counts[1],BFGS_grad$counts[1])
grad_count_val <- c(CG$count[2],BFGS$counts[2],CG_grad$counts[2],BFGS_grad$counts[2])
conv_val <- c(CG$convergence,BFGS$convergence,CG_grad$convergence,BFGS_grad$convergence)
overall <- cbind(mu_val,sigma_val,minloglik_val,fun_count_val,grad_count_val,conv_val)
rownames(overall) <- c("CG","BFGS","CG_grad","BFGS_grad")
overall
```