Lab2,Group15

Naveen Gabriel ,Sridhar Adhirkala

2019-03-05

Contents

1 Assignment 1

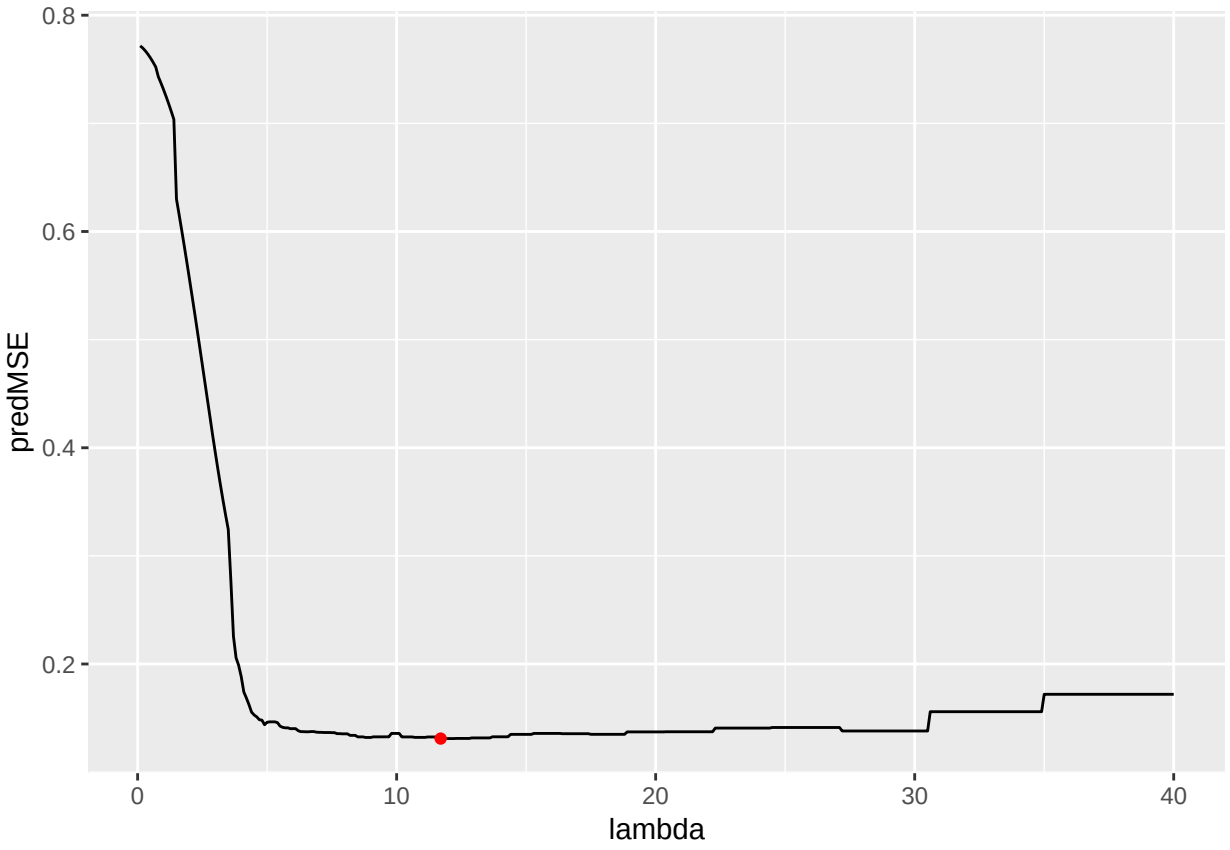
1.1 Reading Data

```
  Day   Rate       LMR
1   1 0.0014 -6.571283
2   2 0.0040 -5.521461
3   3 0.0051 -5.278515
4   4 0.0064 -5.051457
5   5 0.0075 -4.892852
6   6 0.0098 -4.625373
```

Reading Data into variable mortRate and adding log of the rate column to the data frame.

1.2 Created My_MSE function

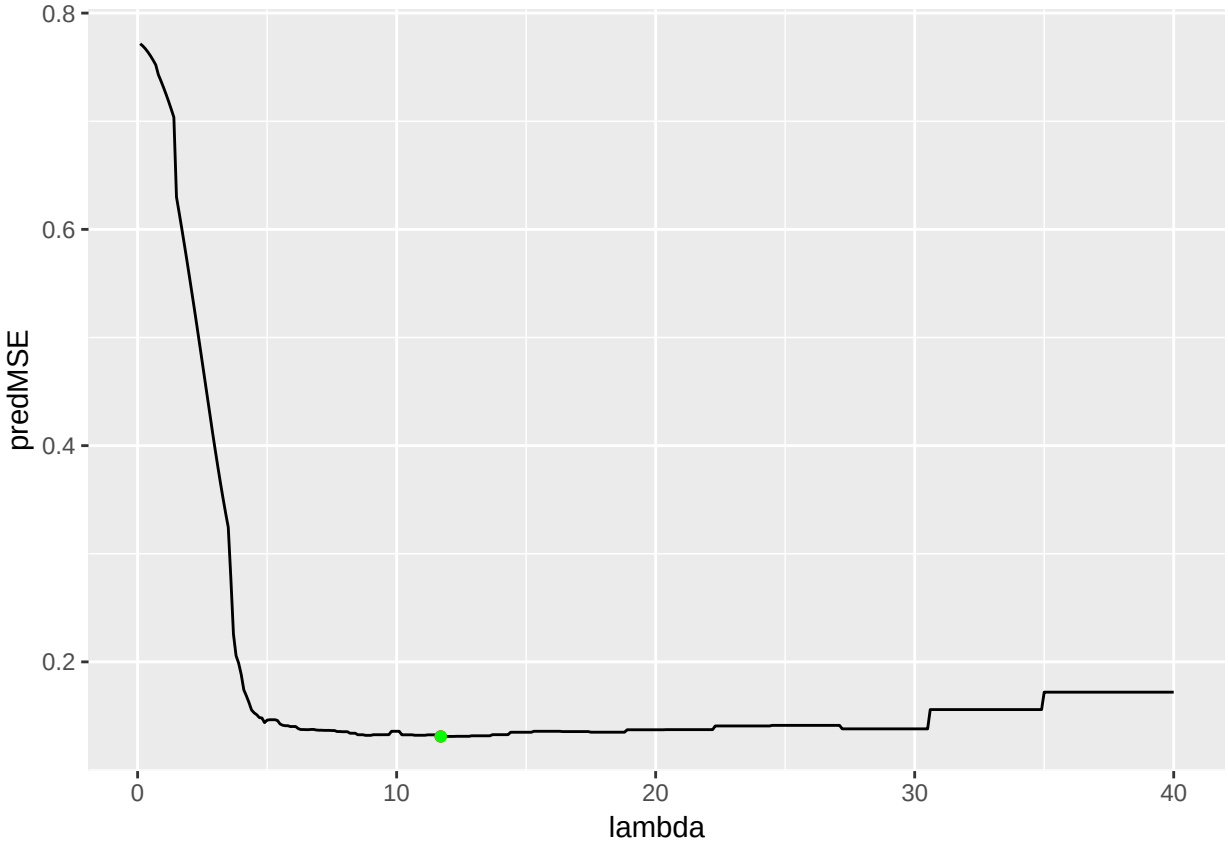
1.3 Using the Function Created for different values of lambda.




```
The minimun value of MSE found by the irerative method is :  0.131046964831872
The corresponding value of lambda at which this minimum was found is :  11.7
```

The minimum found by the iterative method is shown on the plot with a red point.

1.4 Using the My_MSE to find minimum iteratively with a threshold




```
The minimun value of MSE found by the irerative method is :  0.131046964831872
The corresponding value of lambda at which this minimum was found is :  11.7
The method had tolarence value of 0 and it took  245  steps to converge.
```

We use the counts to keep count of the number of times the function has been called. The function MyMSE was called 245 times until convergence, using the tolarence value of (0.01). The loop stops as soon as we get a increment in the MSE value greater than the tolarence value. The step size we are taking is very small 0.1, this is the reason it took 245 steps to converge.

The min value of MSE and the corresponding lambda found in this way is exactly the same to the actual minimum. This is the reason the points overlap in the plot.

1.5 Using Optimize with threshold on Accuracy : 0.01

```
Number of function calls taken for the optimize function to  find minimum :  18

The minimum found by the optimize function :

MIN MSE :  0.132144141920378
LAMBDA  :  10.6936106512018
```

The function takes fewer steps to converge when we introduce a threshold value on accuracy. With the threshold(0.01) in the iterative method it took 245 steps and with the threshold(0.01) using optimize it took just 18 steps to converge. I think the function optimize takes a larger step that lets it get to the min value faster, it is not considering all the intervals we provided it with. This is the reason the min value found by this function is not the global minimum, it is close to it, but not the exact minimum.

The minimum found in the previous case was the actual minimum and in this case the algorithm was forced to declare convergence as the threshold value of accuracy was satisfied.

1.6 Using optim()function and BFGS to find lambda value.

```
Number of function calls taken for the optimize function to  find minimum :  3

The minimum found by the optimize function :

MIN MSE :  0.171999614458471
LAMBDA  :  35

Number of Iterations :  3
```

The initialization point we set for the algorithm was 35, there is a steep downhill at that point, so the algorithm takes that step, but then the function is plateaued towards the lower bound from that point and has a peak as lambda value increase from that point. This is the reason the algorithm declares convergence at the local optima, and takes just three steps to declare convergence.

This function depends heavily on the initial point we specify as the start. If our initialization is bad the function will get stuck at a local optima. The previous optimize funcion took in a list of values to search over and find the minimum, this is the reason the convergence found by the previous function was better.

2 Maximum Likelyhood

2.1 Log likelyhood and deriving maximum likelyhood estimator

$$log(x_1, x_2...x_{100}|\mu, \sigma) = -\frac{1}{2 * \sigma^2} * \sum_{i=1}^{100}(x_i - \mu)^2 - \frac{100}{2}(log 2\pi\sigma^2)$$

Let P = $log(x_1, x_2...x_{100}|\mu, \sigma)$

$$\frac{\partial P}{\partial \mu} = \frac{1}{\sigma^2} * \sum_{i=1}^{100}(x_i - \mu)$$
$$\frac{\partial P}{\partial \sigma} = \frac{\sum_{i=1}^{100}(x_i - \mu)^2}{\sigma^3} - \frac{100}{\sigma}$$

We can maximize the estimator μ and σ by equating both the above derivative to zero and solving it. .

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

`Below parameters estimates the normal distribution from where the sample is taken:`

`Mean(mu): 1.28`

`Sigma: 2.01`

2.2 Optimizing the minus log likelihood function

Below is the minus log likelhood function

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

Why it is a bad idea to maximize likelihood rather than maximizing log likelihood?

Likelyhood terms are multiplicative and most of the distribution for whose finding the parameter is of keen interest includes exponential terms. Maximizing the likelyhood by finding a derivative and equating to zero would require cumbersome mathematics and is often computationally expensive because the right hand side of the equation contains lot of multiplicative and exponential term. Since log is monotonically increasing function, the parameter that maximizes the log of a function would be same as the one that maximizes the likelyhood. Log tranforms the multiplicative terms into addition terms and exponential term into multiplicative term which is less computationally intensive. Moreover the log reduces the number in interest and computer work much better with smaller numbers thereby reducing the precision error as well.

```
#Minus log likelyhood
minusloglikely <- function(par) {
    mu <- par[1]
    sigma <- par[2]
    1/((2*sigma^2))*sum((data-mu)^2)+(length(data)/2)*(log(2*pi*sigma^2))
}
```

```r
#gradient function calculates gradient for mu and sigma which is basically
#negative of dP/d(mu) and dp/d(sigma)
gradient_func <- function(par) {
    mu <- par[1]
    sig <- par[2]
    grad_mu <- - (sum(data-mu))/sig^2
    grad_sig <- (length(data)/sig) - sum((data-mu)^2)/sig^3

    return(c(grad_mu,grad_sig))
}

#Optimizing negative log likelyhood using conjugate gradient and BFGS method
#with and without gradient

optimum_val <- mapply(FUN=function(first_arg,second_arg) {
    r = optim(par=c(0,1),fn = minusloglikely,
                          gr=first_arg,
                          method = second_arg)

                   data.frame("mu"= round(r$par[1],2),
                              "sigma" = round(r$par[2],2),
                              "Function_Counts" = r$counts[1],
                              "Gradient_Counts" = r$counts[2])
                  },

                  second_arg = list("BFGS","BFGS","CG","CG"),
                  first_arg = list(NULL,gradient_func,NULL,gradient_func),
                  SIMPLIFY = TRUE,USE.NAMES = TRUE)


colnames(optimum_val) = c("BFGS(NULL_Gradient)", "BFGS",
                          "CG(NULL Gradient)", "CG")

optimum_val <- as.data.frame(optimum_val)
```

2.3 Analysis

Table 1: Analysis of various optimizing algorithm

	BFGS(NULL_Gradient)	BFGS	CG(NULL Gradient)	CG
mu	1.28	1.28	1.28	1.28
sigma	2.01	2.01	2.01	2.01
Function_Counts	37	39	297	53
Gradient_Counts	15	15	45	17

Maximizing log likelyhood,conjugate gradient and BFGS, converges the mean and sigma estimators for the data to same value to 2 decimal precision. We did employ with and without gradient for BFGS and CG. From the above table, it seems that BFGS without gradient function take least iteration of minuslikelyhood fuction to evaluate the same value of mu and sigma.

3 Appendix

```
knitr::opts_chunk$set(
    echo = FALSE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)

library(ggplot2)
#Ass 1
#Q1
mortRate = read.csv("mortality_rate.csv", sep=';', dec = ',')
mortRate$LMR = log(mortRate$Rate)
head(mortRate)

n = dim(mortRate)[1]
set.seed(123456)
id = sample(1:n, floor(n*0.5))
train = mortRate[id,]
test = mortRate[-id,]
#Q2
my_MSE = function(lambda, pars, iterCounter=FALSE){
  model = loess(pars$Y~pars$X, enp.target = lambda)
  fYpred = predict(model, newdata = pars$Xtest)
  n_test = length(pars$Ytest)
  predictiveMSE = sum((pars$Ytest - fYpred)^2)/n_test
  if(iterCounter){
    if(!exists("iterForMyMSE")){
      assign("iterForMyMSE",
             value = 1,
             globalenv())
    } else {
      currentNr <- get("iterForMyMSE")
      assign("iterForMyMSE",
             value = currentNr + 1,
             globalenv())
    }
  }
  return(predictiveMSE)
}


#Q3
lambda = seq(0.1,40,by=0.1)
pars = list()
pars$X = train[, 1]
pars$Y = train[, 3]
pars$Xtest = test[, 1]
pars$Ytest = test[, 3]

predMSE = matrix(0, ncol = 1, nrow = length(lambda))
```

```
counts = matrix(0, ncol = 1, nrow = length(lambda))
for(i in 1:length(lambda)){
  iterForMyMSE = 0
  predMSE[i] = my_MSE(lambda[i], pars, TRUE)
  counts[i] = iterForMyMSE
}

min_lbd = lambda[which.min(predMSE)]
min_mse = min(predMSE)

ggplot() + geom_line(aes(lambda, predMSE)) + geom_point(aes(min_lbd, min_mse), col='red')

cat(paste("The minimun value of MSE found by the irerative method is : ", min_mse, "\nThe corresponding

#Q4
bestMSE = -100
newMSE = -1
counts = 0
thresh = 0.01
bestThresh = 0
for(i in 1:length(lambda)){
  newMSE = my_MSE(lambda[i], pars, TRUE)
  counts = i
  if(bestMSE == -100){
    bestMSE = newMSE
    bestThresh = counts
  }else if(bestMSE > newMSE){
    bestMSE = newMSE
    bestThresh = counts
  }else{
    diffMSE = newMSE - bestMSE
    if(diffMSE>thresh){
      break
    }
  }
}

ggplot() + geom_line(aes(lambda, predMSE)) + geom_point(aes(min_lbd, min_mse), col='red') + geom_point(a

cat(paste("The minimun value of MSE found by the irerative method is : ", bestMSE, "\nThe corresponding
iterForMyMSE = 0
val = optimise(my_MSE, interval = lambda, pars=pars, iterCounter=TRUE, tol = 0.01)
cat(paste("Number of function calls taken for the optimize function to  find minimum : ", iterForMyMSE,
cat(paste("The minimum found by the optimize function : \n"))
cat(paste("MIN MSE : ", val$objective, "\nLAMBDA  : ", val$minimum, "\n"))

#Q5
iterForMyMSE = 0
val = optim(35, fn=my_MSE, pars=pars, iterCounter=TRUE, method="BFGS")
cat(paste("Number of function calls taken for the optimize function to  find minimum : ", iterForMyMSE,
cat(paste("The minimum found by the optimize function : \n"))
cat(paste("MIN MSE : ", val$value, "\nLAMBDA  : ", val$par, "\n"))
cat(paste("Number of Iterations : ", iterForMyMSE, "\n"))
```

```
knitr::opts_chunk$set(
    echo = TRUE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)

library(dplyr)
compare_data <- matrix(ncol=4,nrow=0)
colnames(compare_data) <- c("Mean","Sigma","Count_function_eval","Count_gradient_eval")

load("data.RData")
mu <- round(sum(data)/length(data),2)
sigma <- round(sqrt(sum((data-mu)^2)/length(data)),2)

cat("\nBelow parameters estimates the normal distribution from where the sample is taken: \n\n")
cat("\nMean(mu):",mu)
cat("Sigma:",sigma)

compare_data <- rbind(compare_data,c(mu,sigma,"-","-"))
#Minus log likelyhood
minusloglikely <- function(par) {
    mu <- par[1]
    sigma <- par[2]
    1/((2*sigma^2))*sum((data-mu)^2)+(length(data)/2)*(log(2*pi*sigma^2))
}

#gradient function calculates gradient for mu and sigma which is basically
#negative of dP/d(mu) and dp/d(sigma)
gradient_func <- function(par) {
    mu <- par[1]
    sig <- par[2]
    grad_mu <- - (sum(data-mu))/sig^2
    grad_sig <- (length(data)/sig) - sum((data-mu)^2)/sig^3

    return(c(grad_mu,grad_sig))
}

#Optimizing negative log likelyhood using conjugate gradient and BFGS method
#with and without gradient

optimum_val <- mapply(FUN=function(first_arg,second_arg) {
    r = optim(par=c(0,1),fn = minusloglikely,
                          gr=first_arg,
                          method = second_arg)

                  data.frame("mu"= round(r$par[1],2),
                             "sigma" = round(r$par[2],2),
                             "Function_Counts" = r$counts[1],
                             "Gradient_Counts" = r$counts[2])
                },
```

```
                  second_arg = list("BFGS","BFGS","CG","CG"),
                  first_arg = list(NULL,gradient_func,NULL,gradient_func),
                  SIMPLIFY = TRUE,USE.NAMES = TRUE)


colnames(optimum_val) = c("BFGS(NULL_Gradient)", "BFGS",
                          "CG(NULL Gradient)", "CG")

optimum_val <- as.data.frame(optimum_val)
knitr::kable(optimum_val,align = 'c',"latex",
             caption ="Analysis of various optimizing algorithm") %>%
    kableExtra::kable_styling(latex_options = "hold_position")
```