90-lab2-report

Fengjuan Chen(fench417) Zhixuan Duan(zhidu838)

2/6/2020

Question 1

1 Import the file and add one variable LMR

```
data=read.csv2("mortality_rate.csv")
data=cbind(data,LMR=log(data$Rate))
```

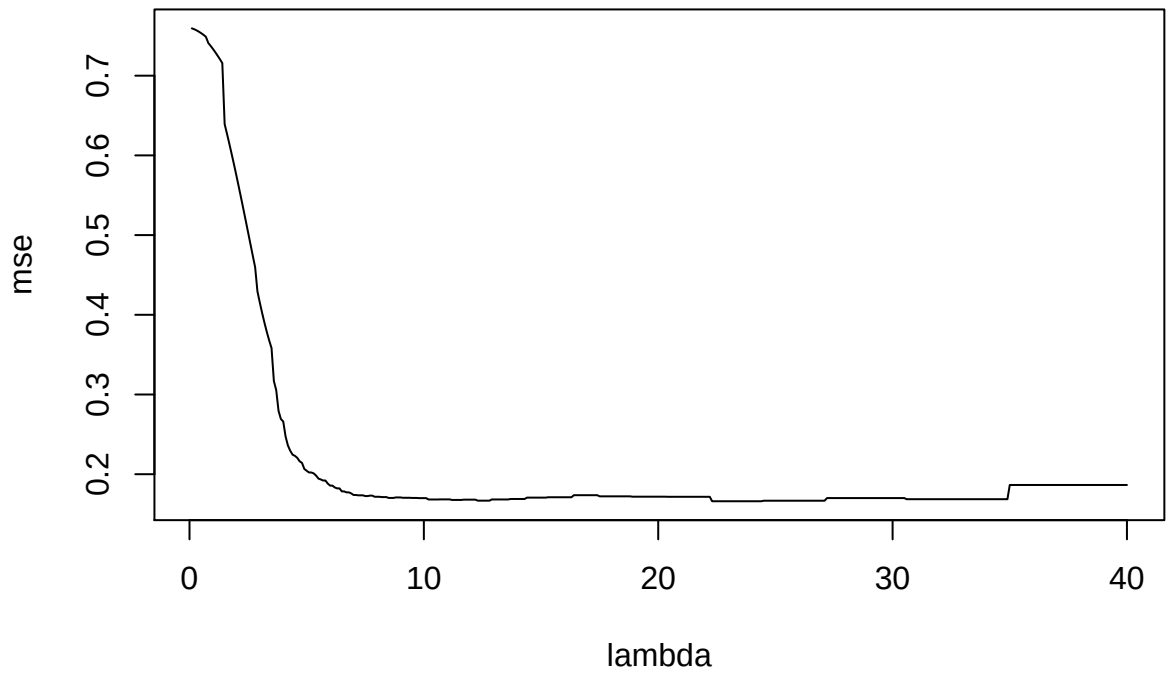
2. Our function myMSE()

```
myMSE <- function(lambda,pars){
  train=data.frame(pars[1],pars[2])
  colnames(train)=c("X","Y")
  test=data.frame(pars[3],pars[4])
  colnames(test)=c("X","Y")
  # fit the LOESS model
  model=loess(Y~X,data=train,enp.target = lambda)
  pre=predict(model,newdata =test)
  # calculate predictive MSE
  a=(pre-test$Y)^2
  mse=sum(a,na.rm = TRUE)/nrow(test)
  # print mse for counting the evaluation number
  print(mse)
  return(mse)
}
```

3 & 4 Plot results of myMSE()

In our function myMSE(), we evaluated 400 values of λ from $0.1, 0.2, 0.3, ..., 40$. From the plot and returned MSE values, we found that the minimal MSE is 0.1660973, the corresponding values of λ are from 22.3 to 24.4. These 22 λs all have the minimum of the function myMSE().

The plot of MSE versus λ is shown below.



5 Use optimize() function

```
minimum=optimize(myMSE,c(0.1,40),tol = 0.01,pars=list)
```

The optimize() can find the optimal MSE value,0.166097 and the position of the minimum is on $\lambda = 22.28461$. Optimize() function evaluated 18 times of myMSE() function.

Comparing with the evaluations number in step 4, we can see that optimize() function needs less evaluations.

6 Use optim() function and BFGS method

```
minimum2=optim(par = 35,myMSE,method = "BFGS",pars=list)
```

The optim() function only used 3 myMSE() functions evalustions to obtain the optimal MSE value. However, it just got 0.1864996 as the optimal MSE value at $\lambda = 35$. It is far away from the mimimum value, 0.1660973, since the MSE is between 0.1660973 and 0.7593852.

From the results of step 5 and step 6, it is clear that in one dimensional situation, the optimize() will evaluate more points than optim() function. Howerver, the optim() can not return the best optimal in this scenario.

In fact, optim() didn't even moved away from the initialisation.

Question 2

1 Load the data

```
data1=get(load("data.RData"))
```

2 Log-likelihood function and MLE estimates

```
loglikelihood <- function(x,y){
  x1=x[1]
  x2=x[2]
  a=length(y)*log(1/(x2*sqrt(2*pi)))
  b=sum((y-x1)^2)/(2*x2^2)
  loglike=a-b
  # min(-loglikelihood)=max(loglikelihood)
  return(-loglike)

}
```

μ. I guess you should better learn to type it in whatever IDE you are using.

The maximum likelihood estimates of u and δ in normal distrubion are $u = \frac{1}{n}\sum_i^n y_i$ and $\delta = \sqrt{\frac{1}{n}\sum_i^n (y_i - u)^2}$. We use R to calculate these MLE estimates on the loaded data.

```
mle_u=mean(data1)
n=length(data1)
mle_delta=sqrt(sum((data1- mle_u)^2)/n)
```

We got the estimated u and δ, 1.27553 and 2.00598.

3 Use optim() function to optimize the function

We first use optim() with Conjugate Gradient method and BFGS method to optimize minus log-likelihood function without gradient specified.

```
cg1=optim(c(0,1),loglikelihood, method = "CG",y=data1)
bfgs1=optim(c(0,1),loglikelihood, method = "BFGS",y=data1)
```

Then we define the gradient function of minus log-likelihood function.

```
grlog <- function(x,y){
  x1=x[1]
  x2=x[2]
  # partial derivative of u from loglikelihood
  a=sum(y-x1)/(x2^2)
  # partial derivative of delta from loglikelihood
  b=sum((y-x1)^2)/(x2^3)-length(y)/x2
  # -a and -b is the partial derivatitve of -log likelihood
  return(c(-a,-b))
}
```

We use optim() with Conjugate Gradient method and BFGS method to optimize minus log-likelihood function again. But this time we specify the gradient of the minum log-likelihood function.

```
cg2=optim(c(0,1),loglikelihood,grlog, method = "CG",y=data1)
bfgs2=optim(c(0,1),loglikelihood,grlog, method = "BFGS",y=data1)
```

In this case the likelihood is no more complex than log-likelihood. Also in general, if the likelihood is multi-model, the log-likelihood will also be so, although perhaps a bit "smoother".

The likelihood function is more complex than the log-likelihood function. And the gradient of likelihood is very hard to calculate. Since the optim() function with method CG and BFGS both use gradient in optimazation process, it is a bad idea to maximize likelihood rather than maximizing log-likelihood.

The main reason to use log-likelihood is that it avoids floating point problems. Note that the likelihood is typically very close to zero when there are multiple data points.

4 Results of optim() function

We show the results of optim() using methods CG and BFGS without gradient specified in table 1.

Table 1: CG and BFGS without gradient specified

	par1	par2	value	count_function	count_gradient	convergence
CG	1.2755	2.006	211.51	171	29	0
BFGS	1.2755	2.006	211.51	37	15	0

The we show the results of optim() using methods CG and BFGS with gradient specified in table 2.

Table 2: CG and BFGS with gradient specified

	par1	par2	value	count_function	count_gradient	convergence
CG	1.2755	2.006	211.51	53	17	0
BFGS	1.2755	2.006	211.51	37	15	0

From the results shown in table 1 and 2, we can see that all these optimization methods converge because all the returned values of convergence are 0.

The columns of par1 and par2 in table1 and table 2 represent the optimal u and δ respectively.

The columns of count_function and count_gradient in table1 and table 2 show the times of function and gradient evaluations for each algorithm to converge.

After comparing with these results in table 1 and table 2, we recommend BFGS method since it needs less evaluations. Further more, the number of evaluations on function and gradient are same with and without gradient specified. Furthermore

But each evaluation of finite-difference gradient approximation takes multiple function evaluations.

Appendix

```r
# Question 1
RNGversion(min(as.character(getRversion()),"3.6.2"))
options(digits = 22)
## 1

data=read.csv2("mortality_rate.csv")
data=cbind(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,]

## 2

myMSE <- function(lambda,pars){
  train=data.frame(pars[1],pars[2])
  colnames(train)=c("X","Y")
  test=data.frame(pars[3],pars[4])
  colnames(test)=c("X","Y")
  model=loess(Y~X,data=train,enp.target = lambda)
  pre=predict(model,newdata =test)
  a=(pre-test$Y)^2
  mse=sum(a,na.rm = TRUE)/nrow(test)
  print(mse)
  return(mse)
}

## 3

list=list(train$Day,train$LMR,test$Day,test$LMR)
lambda=seq(0.1,40,0.01)
mse=vector(length = length(lambda))
j=1
for (i in lambda) {
  mse[j]=myMSE(i,list)
  j=j+1
}

## 4
plot(lambda, mse,ylim=c(0,0.7))
plot(lambda,mse,type = 'l')

## 5
minimum=optimize(myMSE,c(0.1,40),tol = 0.01,pars=list)

## 6
minimum2=optim(par = 35,myMSE,
               method = "BFGS",pars=list)
```

```r
# Question 2

## 1

data1=get(load("data.RData"))

## 2

## log-likelihood function

loglikelihood <- function(x,y){
  x1=x[1]
  x2=x[2]
  a=length(y)*log(1/(x2*sqrt(2*pi)))
  b=sum((y-x1)^2)/(2*x2^2)
  loglike=a-b
  #  mim(-loglikelihood)=max(loglikelihood)
  return(-loglike)

}

## the maximum likelihood of u and delta
##       in normal distrubion is
##   u=mean=y_bar=sum(yi)/n
##   delta=sqrt(sum((yi-u)^2)/n)

mle_u=mean(data1)
n=length(data1)
mle_delta=sqrt(sum((data1- mle_u)^2)/n)


## 3

## gradient of -log likelihood
##
grlog <- function(x,y){
  x1=x[1]
  x2=x[2]
  # partial derivative of u from loglikelihood
  a=sum(y-x1)/(x2^2)
  # partial derivative of delta from loglikelihood
  b=sum((y-x1)^2)/(x2^3)-length(y)/x2
  # -a and -b is the partial derivatitve of -log likelihood
  return(c(-a,-b))
}

## optimize the minus log-likelihood using CG

##  without gradient

cg1=optim(c(0,1),loglikelihood,
          method = "CG",y=data1)
```

```r
## with gradient

cg2=optim(c(0,1),loglikelihood,grlog,
          method = "CG",y=data1)

## optimize the minus log-likelihood using BFGS

## without gradient

bfgs1=optim(c(0,1),loglikelihood,
            method = "BFGS",y=data1)

## with gradient
bfgs2=optim(c(0,1),loglikelihood,grlog,
            method = "BFGS",y=data1)

## 4
# results showing in two tables

comp1=data.frame(c(cg1$par[1],bfgs1$par[1]),
                 c(cg1$par[2],bfgs1$par[2]),
                 c(cg1$value,bfgs1$value),
                 c(cg1$counts[1],bfgs1$counts[1]),
                 c(cg1$counts[2],bfgs1$counts[2]),
                 c(cg1$convergence,bfgs1$convergence))
colnames(comp1)=c("par1","par2","value","count_function",
                  "count_gradient","convergence")

comp2=data.frame(c(cg2$par[1],bfgs2$par[1]),
                 c(cg2$par[2],bfgs2$par[2]),
                 c(cg2$value,bfgs2$value),
                 c(cg2$counts[1],bfgs2$counts[1]),
                 c(cg2$counts[2],bfgs2$counts[2]),
                 c(cg2$convergence,bfgs2$convergence))
colnames(comp2)=c("par1","par2","value","count_function",
                  "count_gradient","convergence")
```