Computational Statistics
732A90

Lab 2

Albin Västerlund
albva223

Eric Herwin
Erihe068

Contents

Assigment 1. Optimizing a model parameter

1

We start to import the data and add the variable LMR to the data set. We also split the data in training and test data.

```
data <- read_delim("mortality_rate.csv",";")
```

```
## Parsed with column specification:
## cols(
##   Day = col_integer(),
##   Rate = col_character()
## )
```

```
data$Rate<-as.numeric(gsub(",",".",data$Rate))
data$LRM<-log(data$Rate)

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

2

We make a function that we call `myMSE()` that use the function `loess()` to the traning data and then calculate the MSE for the test data and then return it.

```
# Make the para list ####
para<-list()
para$X<-train %>% select(Day) %>% as.matrix()
para$Y<-train %>% select(LRM) %>% as.matrix()

para$Xtest<-test %>% select(Day) %>% as.matrix()
para$Ytest<-test %>% select(LRM) %>% as.matrix()


# make function my_MSE ########
my_MSE<-function(lambda,para){
  model<-loess(para$Y~para$X,enp.target=lambda)

  fitted_test<-predict(model,newdata=(para$Xtest))
  MSE_test<-1/length(fitted_test)*sum((as.numeric(para$Ytest)-fitted_test)^2)
  MSE_test
}
```

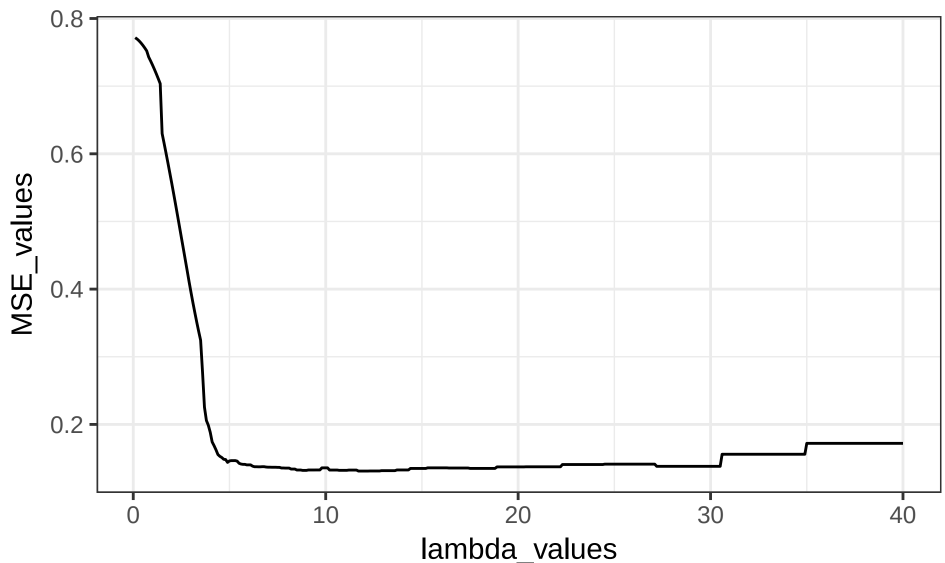
3 and 4

Now we want to use the `myMSE()` function to test which `lambda` value the minimize the MSE. We going to set $\lambda = 0.1, 0.2, ..., 40$

```
# make plot of diffrent lambda ####
MSE_values<-data.frame(lambda_values=seq(0.1,40,0.1),
                       MSE_values=0)

j<-0
for(i in MSE_values$lambda_values){
  j<-j+1
  MSE_values[j,2]<-my_MSE(i,para)

}
ggplot(MSE_values,aes(x=lambda_values,y=MSE_values))+
  geom_line()+
  theme_bw()
```




```
minsta<-which.min(MSE_values$MSE_values)
MSE_values$lambda_values[minsta]
```

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

By looking at the graph we see that the MSE for the test data is minimized when $\lambda = 11.7$. For us to come up to this conclusion we evaluated `myMSE()` 400 times.

5

Insted of testing all possible values for λ we can use the function `optimize()` in `r` to check which value of λ that minimize the MSE value for the test data set.

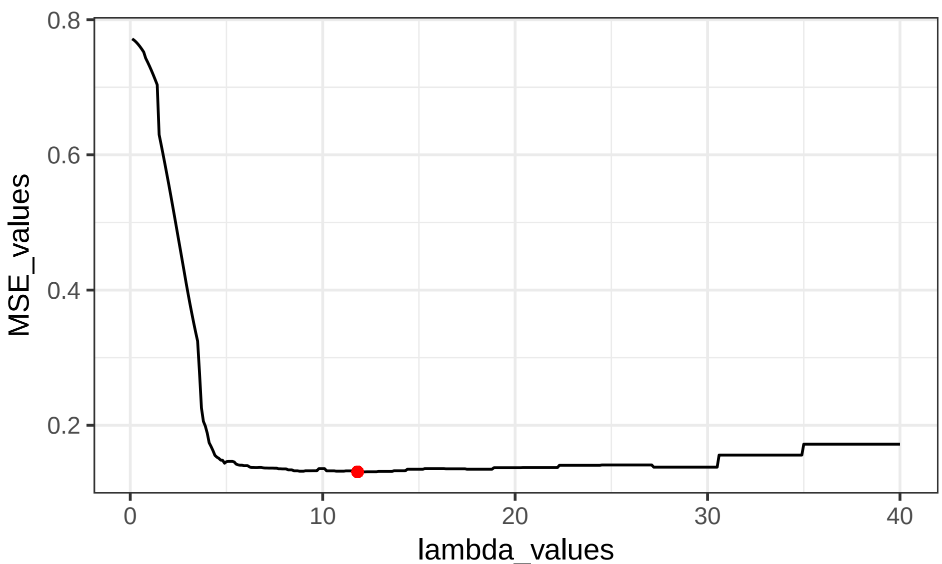
```r
# Optimize lambda ####
my_optim<-optimize(f =my_MSE,
          interval = c(0.1,41),
          tol=0.1,
          para=para )
```

```
## [1] "Lambda=15.7224098601293  MSE=0.135801837464901"
## [1] "Lambda=25.3775901398707  MSE=0.141280865926753"
## [1] "Lambda=9.7551802797414  MSE=0.13265807363536"
## [1] "Lambda=6.0672295803879  MSE=0.140216062782004"
## [1] "Lambda=11.7515476776897  MSE=0.131046964831872"
## [1] "Lambda=11.9546032738958  MSE=0.131046964831872"
## [1] "Lambda=11.8530754757928  MSE=0.131046964831872"
## [1] "Lambda=11.8142953077203  MSE=0.131046964831872"
## [1] "Lambda=11.8142953077203  MSE=0.131046964831872"
```

```r
my_optim
```

```
## $minimum
## [1] 11.8143
##
## $objective
## [1] 0.131047
```

```r
ggplot(MSE_values,aes(x=lambda_values,y=MSE_values))+
  geom_line()+
  geom_point(mapping = aes(x=my_optim$minimum,y=my_optim$objective),col="red")+
  theme_bw()
```



The function `optimize()` return that the value for λ that minimize the MSE for the test data is 11.81. The function `optimize()` evaluated `myMSE()` 9 times and started at 15.72. So by using the `optimize()` function insted of using the simple approach we evaluated `myMSE()` 391 times less. Which could be a huge speed up if the `myMSE()` was a function that takes long time to run.

6

Now are we going to use the function `optim()` with method=BFGS and starting point for $\lambda = 35$ insted of the `optimize()` function that we used in assigment 1.5.

```
optim(par = 35,fn =my_MSE,para=para,method = "BFGS")
```

```
## [1] "Lambda=35  MSE=0.171999614458471"
## [1] "Lambda=35.001  MSE=0.171999614458471"
## [1] "Lambda=34.999  MSE=0.171999614458471"
```

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

By using `optim()` we dont find the value for λ that minimize the MSE for the test data. The function `optim()` evaluates the function with $\lambda = 35$, $\lambda = 35.001$ and $\lambda = 34.999$ and finding that we get the same results for the MSE value and therefore decides to stop the function. That because we are stuck in a local minimum.

Assigment 2. Maximizing likelihood

1

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

2

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$L(\mu,\sigma) = f(x_1|\mu,\sigma)\cdot ... \cdot f(x_n|\mu,\sigma) = \prod_{i=1}^{n}\frac{1}{\sqrt{2\pi\sigma^2}}\cdot e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} = \frac{1}{(2\pi\sigma^2)^n}e^{-\frac{\sum(x_i-\mu)^2}{2\sigma^2}}$$

$$l(\mu,\sigma) = lnL(\mu,\sigma) = -\frac{n}{2}ln(2\pi) - \frac{n}{2}ln(\sigma^2) - \frac{\sum(x_i-\mu)^2}{2\sigma^2}$$

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

MLE for mu

$$\frac{dl(\mu)}{d\mu} = 0$$

$$\frac{dl(\mu)}{d\mu} = \frac{2\sum x_i - 2n\mu}{2\sigma^2} = 0$$

$\frac{dl(\mu)}{d\mu}$ can onley be 0 if $2\sum x_i - 2n\mu = 0$. Which means that:

$$2\sum x_i - 2n\mu = 0 \Rightarrow \hat{\mu} = \frac{\sum x_i}{n} = \bar{x}$$

MLE for sigma

$$\frac{dl(\sigma)}{d\sigma} = 0$$

$$\frac{dl(\sigma)}{d\sigma} = -\frac{n}{\sigma} - \frac{\sum(x_i-\mu)^2}{2}\cdot\frac{-2}{\sigma^3} = -\frac{n}{\sigma} + \frac{\sum(x_i-\mu)^2}{\sigma^3} = \frac{1}{\sigma}\cdot\left(\frac{\sum(x_i-\mu)^2}{\sigma^2} - n\right)$$

$\frac{dl(\sigma^2)}{d\sigma^2}$ can onley be 0 if $\frac{\sum(x_i-\mu)^2}{\sigma^2} - n = 0$. Which means that $\sigma^2 \neq 0$ and:

$$\frac{\sum(x_i-\mu)^2}{\sigma^2} - n = 0 \Rightarrow \sigma = \sqrt{\frac{1}{n}\cdot\sum(x_i-\mu)^2}$$

Use MLE on our data

```
n<-length(data)
mu_mle<-sum(data)/n
mu_mle
```

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

$$\hat{\mu} = 1.276$$

```
sigma_mle<-sqrt(1/n*sum((data-mu_mle)^2))
sigma_mle
```

```
## [1] 2.005976
```

$$\hat{\sigma} = 2.006$$

3

In this assigment we will use the function `optim()` with method=CG (Conjugate Gradient method) and method=BFGS to optimize the value σ and μ.

To do so we need to make a function called log_lik that return the minus log likelihood for a given σ and μ.

We will use both the standard setting for calculate the gradient and make a function called `log_lik_grr()` that calculate the gradient.

```
log_lik<-function(x,data){

  mu<-x[1]
  sigma_2<-x[2]^2
  n<-length(data)
  nr1<--n/2

  first<-nr1*log(2*pi)
  sec<-nr1*log(sigma_2)
  third<--(sum((data-mu)^2))/(2*sigma_2)
  tillbaka<-first+sec+third

  #print(paste("mu=",mu,"  ","sigma=",sigma_2,"  ","log_like=",-tillbaka,sep=""))

  -tillbaka
}
```

```
log_lik_grr<-function(x,data){

  mu<-x[1]
  sigma<-x[2]
  n<-length(data)
  grr_mu<-(2*sum(data)-2*n*mu)/(2*sigma^2)
```

```
  first<--1/(sigma)
  sec<-((sum((data-mu)^2))/sigma^2)-n
  grr_sigma<-first*sec
  -c(grr_mu,grr_sigma)

}
```

Now when we have the function we can use `optim()` to optimize μ and σ.

```
CG_normal<-optim(par = c(0,1),fn = log_lik,data=data,method = "CG")
CG_with_gr<-optim(par = c(0,1),fn = log_lik,gr = log_lik_grr,data=data,method = "CG")
```

```
BFGS_normal<-optim(par = c(0,1),fn = log_lik,data=data,method = "BFGS")
BFGS_with_gr<-optim(par = c(0,1),fn = log_lik,gr = log_lik_grr,data=data,method = "BFGS")
```

```
##             CG_normal CG_with_gr BFGS_normal BFGS_with_gr
## mu              1.276      1.276       1.276        1.276
## Sigma           2.006      2.006       2.006        2.006
## function      210.000     53.000      37.000       38.000
## gradient       35.000     17.000      15.000       15.000
## convergence     0.000      0.000       0.000        0.000
```

It's a bad ide to maximize likelihood rather than maximizing log likelihood because $e^{-\frac{\sum(x_i-\mu)^2}{2\sigma^2}}$ can get very samll so the computer do a underflow mistake when we maximize likelihood.

4

Both CG and BFGS algorithms did converge and find the optimal values for σ and μ in all cases. The CG algorithm got a big speed up when we used or gradient function while BFGS algorithm was almost the same.

The algorithms evaluated the `log_lik()` function and calculated the gradient diffrent number of times. The algorithm that evaluated both the function and the gradient fewest times was the BFGS algorithm.

We would recommend the BFGS method in this case. Conjugate gradient methods will generally be more fragile than the BFGS method, but as they do not store a matrix they may be successful in much larger optimization problems.