732A90: Lab 2
Computational Statistics, Group 6

Chih-Yuan Lin, Sarah Walid Alsaadi, Carles Sans Fuentes, Joshua Burrata

February 10, 2017

Question 1

Exercise 1

In this exercise, the datafile **mortality rate.csv** is used, which contains information about mortality rates of the fruit flies during a certain period.

On the data, an extra column called "lograte" is added to it (being the logarithm of our data column "Rate") and a partition is performed with the code provided by the activity in train and test data.

Then, a function myMSE(with parameters "pars" and "lambda") is implemented to predict the performance of the loess model on our test data given different lambdas (from 0.1 to 40 by 0.1). A plot of the MSE with given different lambdas can be seen below in Figure 1:

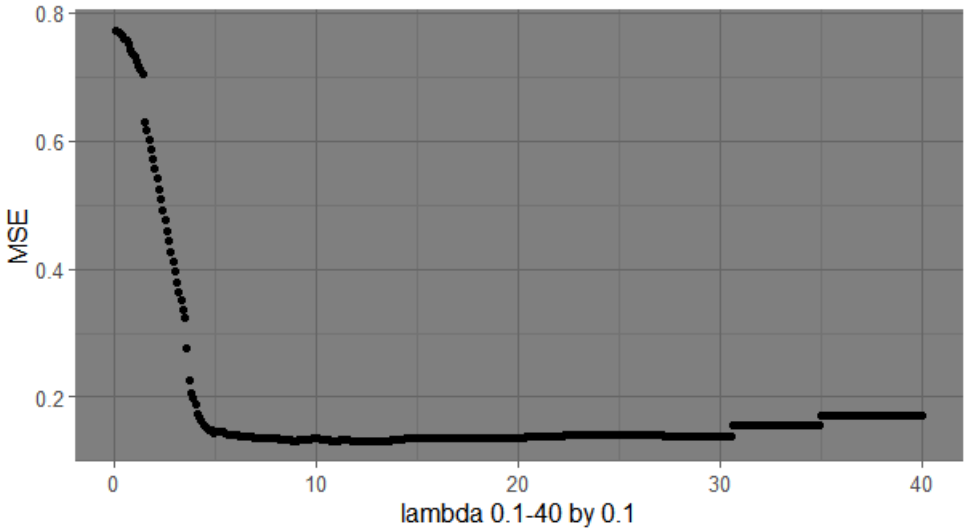

Figure 1: Plot of the MSE of the LOESS model for lambdas from 0.1 to 40 by 0.1

The goal of it is to find the optimal value of lambda for which our model is best. It can be seen that the best lambda used is 11.7 with an MSE of 0.131047. In order to find this solution, the number of iterations needed is the length of the vector lambda (in this case 400 different values for lambda were evaluated).

In order to improve time on optimization, the optimize() and the optim() formula from R are used.

When using the optimize() function for accuracy equals to 0.01, the number of iterations performed are 18 giving the following result:

```
$minimum
[1] 10.69361

$objective
[1] 0.1321441
```

When using the optim() function starting from lambda equals to 35 and using the BFGS method, the number of iterations performed are only 3, getting the following result:

```
$par
[1] 35

$value
[1] 0.1719996

$counts
function gradient
       1        1

$convergence
[1] 0

$message
NULL
```

In both cases the function has found some local optimum, but not the best one because the function is a polynomial of more than one degree. For this reason, none of both functions is good to find a local optima if you run it once, because it can get stuck in local optima. For this reason, if this kind of optimization is to be used, then it would be good to try different starting points when optimizing the function for the parameter.

Question 2

In this exercise, the datafile **data.RData**, which contains a sample from the normal distribution with some parameters μ, σ, is used.

We derive the maximum likelihood estimates for these parameters as follows. We start from the probability density function of the normal distribution:

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

So the likelihood function for N observations is:

$$L(\mu,\sigma|x) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x_n-\mu)^2}{2\sigma^2}} \tag{2}$$

Taking the natural logarithm gives us the log-likelihood function:

$$l(\mu,\sigma|x) = -\frac{N}{2} log(2\sigma^2\pi) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 \tag{3}$$

To find the maximum likelihood estimates for each parameter, we first find the partial derivatives of this log-likelihood function with respect to each parameter:

$$\begin{aligned}
\frac{\partial l}{\partial \mu} &= -\frac{1}{2\sigma^2}\sum_{n=1}^{N}\frac{\partial}{\partial \mu}(x_n-\mu)^2 \\
&= -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(-2)(x_n-\mu) \\
&= \frac{1}{\sigma^2}\left(-N\mu + \sum_{n=1}^{N} x_n\right)
\end{aligned} \tag{4}$$

$$\frac{\partial l}{\partial \sigma} = -\frac{N}{2}\frac{\partial}{\partial \sigma}(log(2\sigma^2\pi)) - \sum_{n=1}^{N}(x_n-\mu)^2\frac{\partial}{\partial \sigma}(\frac{1}{2\sigma^2})$$
$$= -\frac{N}{2}\frac{4\sigma\pi}{2\sigma^2\pi} - \left(-\frac{4\sigma}{(2\sigma^2)^2}\right)\sum_{n=1}^{N}(x_n-\mu)^2$$
$$= -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{n=1}^{N}(x_n-\mu)^2 \tag{5}$$

Setting the partial derivatives to zero and re-arranging gives us the following ML estimators for μ and σ:

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N}x_n \tag{6}$$

$$\sigma_{ML} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_n-\mu)^2} \tag{7}$$

A function that calculates these maximum likelihood estimators for a normal distribution is implemented. The following results for the data are provided below:

```
> loglikelihood(newdata)
$mu
[1] 1.275528

$var
[1] 4.023942

$sd
[1] 2.005976

$loglikelihood
[1] 211.5069
```

Then, we are asked to optimize the minus loglikelihood function with initial parameters $\mu = 0, \sigma = 1$ using the Conjugate Gradient (CG)) and "BFGS" method with gradient specified and without.

For specifying the gradient, we created a function which takes the same arguments as our minus loglikelihood function and returns the gradient of this function. We know that the gradient of the log-likelihood function here is given by:

$$\nabla l = \left(\frac{\partial l}{\partial \mu}, \frac{\partial l}{\partial \sigma}\right)^T \tag{8}$$

Therefore we can use the partial derivative expressions obtained in equations 4 and 5 and just take the negative as we are optimizing the minus log-likelihood function here.

The usage of the log-likelihood in front of the likelihood is just a matter of convenience: (1) the logarithm is a monotonically increasing function, which means that it achieves its maximum value at the same points as the function itself. (2) having everything in logarithm eases the derivative work when try to optimize and find maximum or minimums. (3) it helps on improving the quality of the calculations computationally speaking reducing the problems that arise when using large numbers.

Here we provide with the results of each data:

```
> myframe
                  CG with gradient CG without gradient BFGS with gradient BFGS without gradient
mu                        1.275528              1.275528           1.275528                1.275528
sigma                     2.005976              2.005976           2.005977                2.005977
loglikelihood           211.506949            211.506949         211.506949              211.506949
counts.function          53.000000            210.000000          38.000000               37.000000
counts.gradient          17.000000             35.000000          15.000000               15.000000
convergence               0.000000              0.000000           0.000000                0.000000
```

The algorithm converges in all cases giving the same output (as provided above) which is the same as in the case when calculating the maximum likelihood with the expected parameters. It is necessary to mention that the counter for the optim function without gradient does not count well iterations since it excludes those calls needed to compute the Hessian, if requested, and any calls to the function to compute a finite-difference approximation to the gradient. Thus, the real number of times that the function was called has been computed being 97 times for BFGS and 350 times for th CG method.

Nevertheless, it can be seen that when the gradient function is provided, the optimization is performed much faster and the number of iterations is lowered a lot. The fastest model then is the BFGS model, being called 38 times for the loglikelihood function and 15 times for the gradient function.

Contributions

All results and comments presented have been developed and discussed together by the members of the group.

Appendix

Assignment 1

```
### Assignment 2

## Exercise 1

data<- read.csv2("C:/Users/M/Desktop/Statistics and Data Mining Master/Semester 2/Computationa
    Statistics/Lecture 2/mortality_rate.csv", sep = ";", stringsAsFactors = FALSE)

# Adding loragithmic of rate

lograte<- log(data[,2])
#binding data
data<- cbind( data, lograte)


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

mylist<- list(X = train[,1], Y = train[,3], Xtest= test[,1], Ytest= test[,3])

myseq<- seq(0.1,40,0.1)
##1.3 doing the function
counter<- 0
myMSE <- function(lambda= myseq, pars = mylist){
  mytrain<- data.frame(X =pars[["X"]],Y = pars[["Y"]])


  pred_MSE<- integer(length(lambda)) ##variable to store values
  counter <<- counter+1
  print(counter)
  for(count in 1:length(lambda)){
    model <- loess(formula = Y ~ X, data = mytrain, enp.target = lambda[count])
    fYpred<- predict(model, pars[["Xtest"]])
    pred_MSE[count]<- sum((pars[["Ytest"]]-fYpred)**2)/length(fYpred)

    }
  return(pred_MSE)
}

MSEresults<-myMSE(pars = mylist, lambda= myseq)

##best lambda
best<- c(best_lambda = myseq[which.min(MSEresults)], MSE = min(MSEresults))


## 1.4 plot
myplot<-function(vector1, vector2){
  library(ggplot2)
  ggplot2::ggplot(data = data.frame(X1 =vector1, X2= vector2))+
  geom_point(aes(x = X1, y= X2))+
  xlab("lambda 0.1-40 by 0.1")+ ylab("MSE")+
  theme_dark()}
myplot(myseq, MSEresults)


## 1.5


myoptimize<- optimize(myMSE , interval = c(0.1,40), tol = 0.01, pars = mylist) ##iterations =
    18

## 1.6

myoptim <- optim(35 , myMSE , method = "BFGS", pars = mylist) #iterations = 3



#####2

load("C:/Users/M/Desktop/Statistics and Data Mining Master/Semester 2/Computationa Statistics/
    Lecture 2/data.RData")

newdata<- data

##2.2

Nloglikelihood<- function(x){
  n<- length(x)
```

```
  mu<- sum(x)/n
  var<- (1/n)*sum((x-mu)**2)
  sd<- sqrt(var)
  loglikelihood<- n*log(1/(sqrt(2*var*pi))) - sum((x-mu)**2)/(2*var)

  res<- list(mu = mu, var = var, sd = sd, loglikelihood= loglikelihood)

  return(res)

}

Nloglikelihood(newdata)

##2.3
count<- 0
optimloglikelihood<- function(par, x){

  n<- length(newdata)
  loglikelihood<- -(-n/2*log(2*pi)-n/2*log(par[2]**2) - sum((x-par[1])**2)/(2*par[2]**2))
  count<<- count+1
  print(count)
  return(loglikelihood)

}


###97 iterations
BFGS_no_grad<-optim(par = c(0, 1), optimloglikelihood, x= newdata , method = "BFGS")
##350 iterations
CG_no_grad<-optim(par = c(0, 1), optimloglikelihood, x= newdata , method = "CG")


##gradient
count<-0
mycount<- 0
gradient <- function(par, x){
  n<- length(newdata)
  gradient_mu<- -((1/(par[2]**2))*(sum(x)-n*par[1]))
  gradient_sigma <- -(-n/par[2] + (1/par[2]**3)*sum((x-par[1])**2))

  result <- c(gradient_mu = gradient_mu, gradient_sigma= gradient_sigma )
  mycount<<- mycount+1
  print(mycount)
  return(result)
}

##38 iterations, 15 times the gradient function
BFGS_grad <-optim(par = c(0, 1), optimloglikelihood, x= newdata, method = "BFGS", gr = gradient
    )
##53 iterations, 17 times the gradient function
CG_grad   <-optim(par = c(0, 1), optimloglikelihood ,x= newdata, method = "CG", gr = gradient)

##Creating a data frame for answers
myframe <- data.frame(unlist(CG_grad), unlist(CG_no_grad), unlist(BFGS_grad), unlist(BFGS_no_
    grad))
rownames(myframe)<- c("mu", "sigma", "loglikelihood", "counts.function", "counts.gradient", "
    convergence")
colnames(myframe)<- c("CG with gradient", "CG without gradient", "BFGS with gradient", "BFGS
    without gradient")


myframe <- rbind(myframe)
myframe
```