Computational Statistics Lab 2

Jooyoung Lee, Vasileia Kampouraki, Weng Hang Wong

30/1/2020

Question 1: Optimizing a model parameter

The file "mortality_rate.csv" contains information about mortality rates of the fruit fies during a certain period.

1. Import this file to R and add one more variable LMR to the data which is the natural logarithm of Rate. Afterwards, divide the data into training and test sets by using the following code:

```
RNGversion("3.5.1")
```

```
## Warning in RNGkind("Mersenne-Twister", "Inversion", "Rounding"): non-
## uniform 'Rounding' sampler used
```

```
data <- read.csv2("~/Desktop/732A90_VT2020_Materials/mortality_rate.csv")
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. Write your own function myMSE() that for given parameters λ and list pars containing vectors X, Y, Xtest, Ytest fits a LOESS model with response Y and predictor X using loess() function with penalty λ (parameter enp.target in loess()) and then predicts the model for Xtest. The function should compute the predictive MSE, print it and return as a result. The predictive MSE is the mean square error of the prediction on the testing data. It is defined by the following Equation (for you to implement):

Predictive MSE $= \frac{1}{length(test)} \sum_{\text{i}} (Ytest[i] - fYpred(X[i]))^2$

where $fYpred(X[i])$ is the predicted value of Y if X is X[i], such that i is the the index number of elements in the test data set.

Read on R's functions for prediction so that you do not have to implement it yourself.

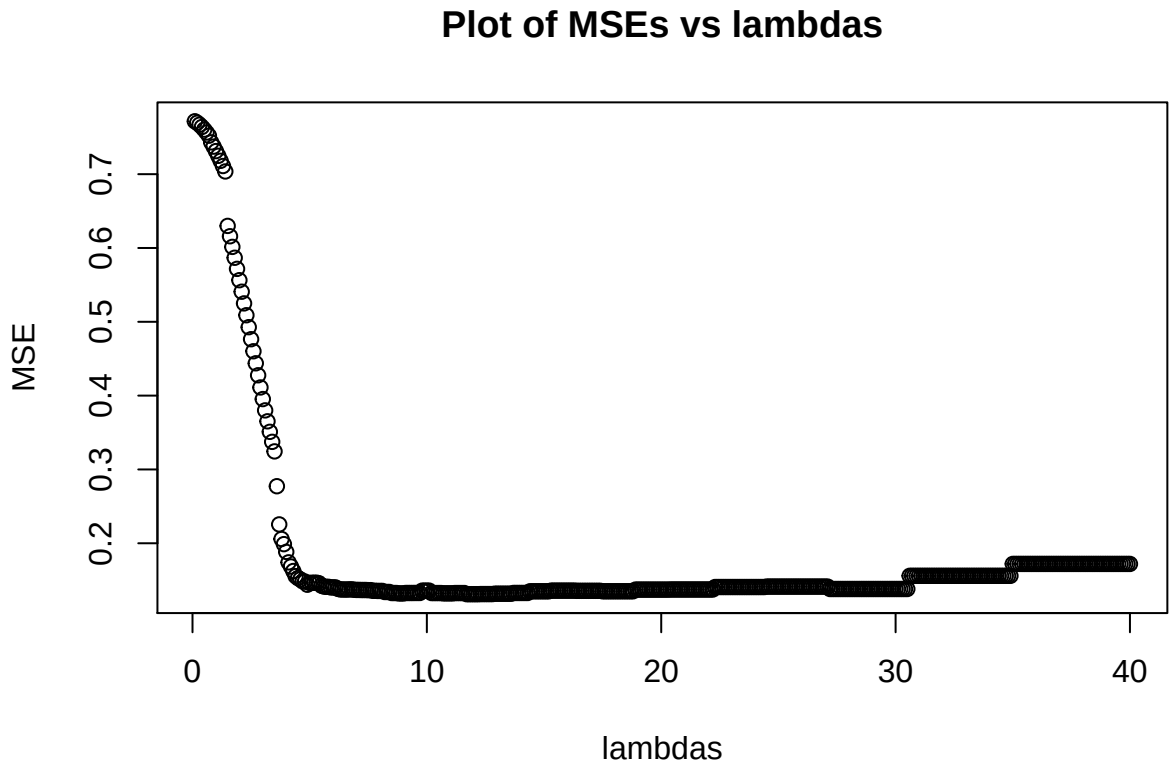
```
myMSE <- function(lambda,pars){
  model <- loess(pars$Y~pars$X, enp.target = lambda)
  fitted <- predict(model,newdata=pars$Xtest)

  MSE <- mean((pars$Ytest-fitted)^2)
  print(MSE)
  return(MSE)
}
```

3. Use a simple approach: use function myMSE(), training and test sets with response LMR and predictor Day and the following λ values to estimate the predictive MSE values: λ =0.1, 0.2, . . . , 40

4. Create a plot of the MSE values versus λ and comment on which λ value is optimal. How many evaluations of myMSE() were required (read ?optimize) to find this value?

```
plot(lambdas,MSE, main="Plot of MSEs vs lambdas")
```



By investigating the plot, it is possible to find that the minimum MSE is given for a value of λ which is around 11. Further investigation is performed as below too check it precisely.

```
MSE[which.min(MSE)]
```

```
## [1] 0.131047
```

```
check <- (MSE==min(MSE))
which(check %in% TRUE)
```

```
## [1] 117 118 119 120 121 122
```

According to the result from the above code, the minimum (optimal) MSE value is 0.131047 and such value of MSE is obtained when λ= 11.7,11.8,11.9,12.0,12.1 and 12.2. Multiple λs are returned, because of rounding issue in the computer.

The number of evaluations needed to find the optimal values of λ is 400, as first, the MSE values for all the 400 different λ has to be computed and then the optimal one that gives the minimum MSE value is extracted (in above case, more than one gave minimum value).

5. Use optimize() function for the same purpose, specify range for search [0.1, 40] and the accuracy 0.01. Has the function managed to find the optimal MSE value? How many myMSE() function evaluations were required? Compare to step 4.

```
#1D optimization
optimize(myMSE,c(0.1,40),pars=list,tol=0.01)
```

```
## [1] 0.1358018
## [1] 0.1412809
## [1] 0.1326581
## [1] 0.1401702
## [1] 0.1325391
## [1] 0.1321441
## [1] 0.1325542
## [1] 0.1321441
## [1] 0.1325391
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
## [1] 0.1321441
```

```
## $minimum
## [1] 10.69361
##
## $objective
## [1] 0.1321441
```

```
#number of evaluations
```

By using **tol=0.01** slightly different value for the minimum MSE (0.1321441) is obtained compared to the one found using the command **MSE[which.min(MSE)]**.

The function was evaluated 18 times which is way lesser than in step 4(400 evaluations) and gave very similar result.

6. Use optim() function and BFGS method with starting point $\lambda = 35$ to find the optimal λ value. How many myMSE() function evaluations were required (read ?optim)? Compare the results you obtained with the results from step 5 and make conclusions.

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

```
## [1] 0.1719996
## [1] 0.1719996
## [1] 0.1719996
```

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

optim() returned an optimal value of MSE, the value 0.1719996 and the function was evaluated only once. The optimal value of MSE differs quite a lot compared to the one obtained in step 5 (0.1321441) and doesn't seem accurate. What **optim()** did was to find a local minimum and stop, while the **optimize** function found the global minimum. Generally, trying to fing global optima can be computationally expensive as it requires in many cases a large number of iterations.

In conclusion, according to above investigations, step 5 gave better results as it found almost precisely the optimal value of MSE and was relatively quick (18 evaluations).

Question 2: Maximizing likelihood

The file data.RData contains a sample from normal distribution with some parameters μ,σ.For this question read ?optim in detail.

1. Load the data to R environment.

```
load("~/Desktop/732A90_VT2020_Materials/data.RData",verbose = T)
```

```
## Loading objects:
##   data
```

2.

Here, the theory behind the maximum likelihood estmation for a normal distribution with parameters μ,σ will be presented.

Probability density function

The probability density function of Normal distribution is:

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

The assumptions for the maximum likelihood estimation are that the data are independently and identically distributed (i.i.d.).

Let $\theta = \{\mu, \sigma\}$ be the parameters set.

The goal here is to maximize the probability density, which can be written by multiplying each probability density, under the assumption of independency.

$f(x_1, ..., x_n|\theta) = f(x_1|\theta) \cdot f(x_2|\theta) \cdot ... \cdot f(x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta)$

Likelihood and log-likelihood functions

The likelihood function is:

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

and the log-likelihood function is:

$l(x_1, ..., x_n|\theta) = log(L(x_1, ..., x_n|\theta))$
$= -\frac{n}{2} ln(2\pi) - \frac{n}{2} ln(\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n}(x_i - \mu)^2$

which is derived by applying the logarithm function on the likelihood function.

Note: The ml estimators will be symbolized as $\hat{\mu}$ and $\hat{\sigma}$.

Maximum likelihood estimators for μ and σ

The maximum likelihood estimators for mean and variance are derived by setting the derivative of the log-likelihood function with respect to μ and σ, respectvely, equal to zero.

$0 = \frac{\partial l(x_1,...,x_n|\theta)}{\partial \mu}$
$= -\frac{1}{2\sigma^2} \sum_{i=1}^{n} -2(x_i - \mu)$
$= \frac{1}{\sigma^2} \sum_{i=1}^{n}(x_i - \mu)$
$\Rightarrow \frac{1}{\sigma^2} \sum_{i=1}^{n}(x_i - \mu) = 0$
$\Rightarrow \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$

$0 = \frac{\partial l(x_1,...,x_n|\theta)}{\partial \sigma}$
$= -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^{n}(x_i - \mu)^2$
$\Rightarrow -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^{n}(x_i - \mu)^2 = 0$
$\Rightarrow \sum_{i=1}^{n}(x_i - \mu)^2 = n\sigma^2$
$\Rightarrow \hat{\sigma^2} = \frac{1}{n} \sum_{i=1}^{n}(x_i - \mu)^2$

3.

```
n <- length(data)
mu_ml <- sum(data)/n
sigma_ml <- sqrt(sum((data-mu_ml)^2)/n)
mu_ml
```

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

```
sigma_ml
```

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

```r
### defining function for -log-likelihood
par=c(0,1)
minus_llik <- function(par, data) {
  n <- length(data)
  mu <- par[1]
  sigma <- par[2]
  llik <- ((-n/2)*log(2*pi*(sigma^2))) +
    ((-1/(2*(sigma^2)))*sum((data-mu)^2))
  return(-llik)
}

### defining function for gradient
grad <- function(par, data) {
  n <- length(data)
  mu <- par[1]
  sigma <- par[2]
  grad_mu <- -(sum(data)-(mu*n))/(sigma^2)
  grad_sigma <- (n/sigma) - (sum((data-mu)^2)/(2*(sigma^3)))
  return(c(grad_mu, grad_sigma))
}
```

```r
CG_without <- optim(par=c(0,1), fn=minus_llik, method="CG", data=data)
CG_with <- optim(par=c(0,1), fn=minus_llik, method="CG", data=data, gr=grad)

BFGS_without <- optim(par=c(0,1), fn=minus_llik, method="BFGS", data=data)
BFGS_with <- optim(par=c(0,1), fn=minus_llik, method="BFGS", data=data, gr=grad)
```

why is it bad to use just likelihood?

Since calculating the derivative of likelihood is more complicated and costly. Using log-likelihood in the function we will have the same result while the deviative of log-likelihood is simplier and easier than likelihood.

4.

Did the algorithms converge in all cases?

Both the CG and BFGS algorithms converge in all cases.

Optimal values and function and gradients were required.

The real mean and standard deviation from the data set are 1.275528 and 2.005976 respectively.

When the Conjugate Gradient method is used *without specified gradient*, the optimal values are: μ=1.275528, and σ=2.005977 using 297 function and 45 gradient as evaluations, of which the optimal parameters are closely the same with the real values. However, when it *specified with gradient*, the result returned μ=1.678282 and σ=1.956658 using 283 function and 29 gradient, even though this result is close but worse than the above method without gradient.

The same situation also applied to BFCS method, when the function is run with *unspecified the gradient* the optimal parameters are μ=1.275528, σ=2.005977 using 37 function and 15 gradient. But with *specified gradient* in BFCS, μ=1.299622 and σ=1.900686 using 86 function and 13 gradient are returend.

According to the above result, the BFCS without specified the gradient would be recommended for having the same accuracy result compare to the CG method without gradient. Furthermore, since the BFCS cost less iteration and use fewer steps for convergence, it is more efficient.

Appendix

```
RNGversion("3.5.1")
data <- read.csv2("~/Desktop/732A90_VT2020_Materials/mortality_rate.csv")
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,]
myMSE <- function(lambda,pars){
  model <- loess(pars$Y~pars$X, enp.target = lambda)
  fitted <- predict(model,newdata=pars$Xtest)

  MSE <- mean((pars$Ytest-fitted)^2)
  print(MSE)
  return(MSE)
}
lambdas <- seq(0.1,40,0.1)
list <- list(X=train$Day,Y=train$LMR,Xtest= test$Day,Ytest=test$LMR)
#calculate MSEs using myMSE
MSE <- vector(length = length(lambdas))
for (i in 1:length(lambdas)){
  MSE[i] <- myMSE(lambdas[i],list)
}
plot(lambdas,MSE, main="Plot of MSEs vs lambdas")
MSE[which.min(MSE)]
check <- (MSE==min(MSE))
which(check %in% TRUE)
#1D optimization
optimize(myMSE,c(0.1,40),pars=list,tol=0.01)
#number of evaluations
optim(35,myMSE,NULL,method = "BFGS",pars=list)
load("~/Desktop/732A90_VT2020_Materials/data.RData",verbose = T)
n <- length(data)
mu_ml <- sum(data)/n
sigma_ml <- sqrt(sum((data-mu_ml)^2)/n)
mu_ml
sigma_ml

### defining function for -log-likelihood
par=c(0,1)
minus_llik <- function(par, data) {
  n <- length(data)
  mu <- par[1]
  sigma <- par[2]
  llik <- ((-n/2)*log(2*pi*(sigma^2))) +
```

```
    ((-1/(2*(sigma^2)))*sum((data-mu)^2))
  return(-llik)
}

### defining function for gradient
grad <- function(par, data) {
  n <- length(data)
  mu <- par[1]
  sigma <- par[2]
  grad_mu <- -(sum(data)-(mu*n))/(sigma^2)
  grad_sigma <- (n/sigma) - (sum((data-mu)^2)/(2*(sigma^3)))
  return(c(grad_mu, grad_sigma))
}
CG_without <- optim(par=c(0,1), fn=minus_llik, method="CG", data=data)
CG_with <- optim(par=c(0,1), fn=minus_llik, method="CG", data=data, gr=grad)

BFGS_without <- optim(par=c(0,1), fn=minus_llik, method="BFGS", data=data)
BFGS_with <- optim(par=c(0,1), fn=minus_llik, method="BFGS", data=data, gr=grad)
```