Computational Statistics Lab2

Min-Chun Shih (shimi077), Saewon Jun(saeju204)

Question 1: Optimizing a model parameter

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

1.1 Import the data and add one more varaible 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

```
data <- read.csv2("mortality_rate.csv")
#in this case using read.csv2 is easier since they automatically substr , into .
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,]
```

1.2

- 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()), 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. (fYpred(X[i]) is the predicted value of Y if X is X[i])

$$predictiveMSE = \frac{1}{length(test)} \sum_{ith\ element\ in\ test\ set} (Ytest[i] - fYpred(X[i]))^2$$

```
my_MSE <- function(lambda, pars){
        #lambda is a given parameter
        #pars is a list containing vectors X,Y,Xtest,Ytest
        #loess is LOcal regrESSion - predict the best predicton for each interval

        model <- loess(pars$Y ~ pars$X, enp.target=lambda)
        pred <- predict(model, newdata=pars$Xtest)


        predMSE <- sum((pars$Ytest-pred)^2)/nrow(test)
        it <<- it+1

        return(predMSE)


}
```

1.3 Use a simple approach: use function myMSE(), training and test sets with reponse LMR and predictor Day and the following lambda values to estimate the predictive MSE values

- λ=0.1, 0.2, ...,40

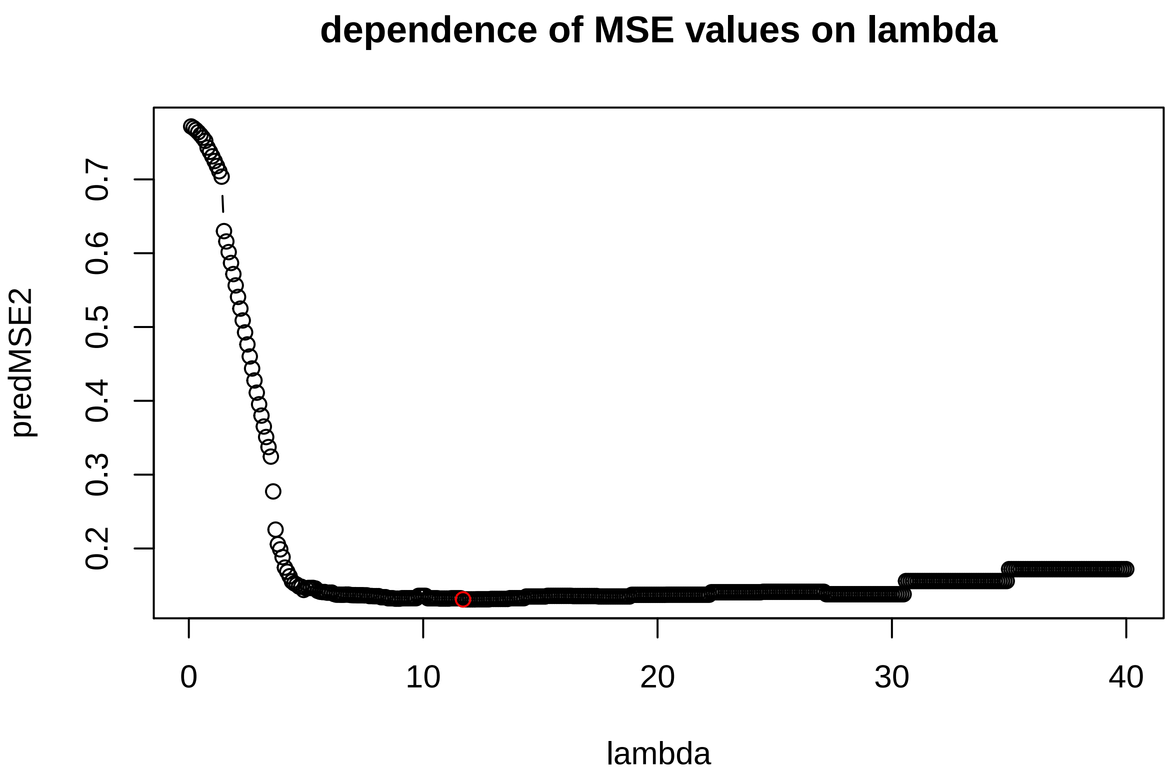
```
lambda <- seq(0.1,40,0.1)
pars <- list(Y=train$LMR, X=train$Day, Ytest=test$LMR, Xtest=test$Day)

predMSE2 <- c()
it <- 0

for (i in 1:length(lambda)){
        predMSE2[i] <- my_MSE(lambda[i], pars)

}
```

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

```
which.min(predMSE2)
```

```
## [1] 117
```

```
plot(lambda, predMSE2, type="b", main="dependence of MSE values on lambda")
points(lambda[117], predMSE2[117],col="red")
```



```
list("opimal_lambda(minimum)"=lambda[117],
     "MSE_for_optimal_lambda(objective)"=predMSE2[117],
     "#_of_evaluations"=it)
```

```
## $`opimal_lambda(minimum)`
## [1] 11.7
##
## $`MSE_for_optimal_lambda(objective)`
## [1] 0.131047
##
## $`#_of_evaluations`
## [1] 400
```

The optimal lambda is 11.7 , and as we simply went through all the lambdas so the number of evaluation is 400 which is equal to the number of lambda. We can check the minimum MSE only after going through all the possible cases we have for lambda.

1.5 Use optimize() function for the same purpose

- range for search to [0.1,40]
- accuracy to 0.01 Have the function managed to find the optimal MSE value? How many myMSE() function were required this time? compare.

```
it<-0
optimize(my_MSE, lower=0.1, upper=40, tol=0.01, pars=pars)
```

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

```
it
```

```
## [1] 18
```

The lambda value is 10.69361 which is quite similar to the result we got from step 1.4, but the number of function evaluated is decreased a lot, which is more efficient - less time consuming.

1.6 Use optim() fucntion and BFGS method

- starting point $\lambda = 35$ to find the optimal λ value Compare the results from step 5 and make conclusion.

```
it <- 0
optim(35, my_MSE, method="BFGS", pars=pars)
```

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

```
it
```

```
## [1] 3
```

Method "BFGS" is a quasi-Newton method. Here, the value we got is local minimum. Compare to previous exercises we can see that the number of evaluation has decreased a lot, so from this exercise we can figure out that if you choose the reasonable starting parmater, you can save your time.

Question 2: Maximizing likelihood

The file data.RData contains a sample from *normal distribution* with some parameters μ, σ.

2.1 Load the data to R environment.

```
load("data.RData")
x <- data #sample with 100obs from normal distribution
```

2.2 Write down the log-liklihood function for 100 observations and derive maximum likelihood estimators for mu and sigma analytically by setting partial derivatices to zero. Use the derived formulae to obtain parameter estimates for the loaded data.

Gaussian distribution,

$$f_{\mu,\sigma}(x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(\frac{-(x_i-\mu)^2}{2\sigma^2}) \qquad \theta = (\mu, \sigma)$$

$$\mathcal{L}(\theta) = \prod f_{\mu,\sigma}(x_i) = \prod \frac{1}{\sqrt{2\pi\sigma^2}} exp(\frac{-(x_i-\mu)^2}{2\sigma^2})$$

$$\mathcal{L}^*(\theta) = -\frac{n}{2} log 2\pi - n log\sigma - \frac{1}{2\sigma^2}\sum_i (x_i-\mu)^2$$

1. $$\frac{\partial}{\partial\mu}\mathcal{L}^*(\theta) = \frac{1}{\sigma^2}\sum_i (x_i-\mu) = \frac{1}{\sigma^2}(\sum_i x_i - n\mu) := 0$$

$$Then, we\ get \quad \hat{\mu} = \frac{\sum_i x_i}{n}$$

2. $$\frac{\partial}{\partial\sigma}\mathcal{L}^*(\theta) = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_i (x_i-\mu)^2 := 0$$

$$Then, we\ get \quad \hat{\sigma^2} = \frac{\sum_i (x_i-\mu)^2}{n}$$

```
loglik <- function(par){
        n = length(x)
        mu=par[1]
        sigma=par[2]
        ll = -n*log(2*pi)*(1/2)-n*log(sigma)-sum((x-mu)^2)*(1/(2*sigma^2))
        #ll=-(n/2)*log(2*pi)-nlog(sigma)-(1/(2*sigma^2))*sum((x-mu)^2)
        return(-ll)
        #we are minimizing the function so we need -logliklihood
}
```

```
mu_hat <- sum(x)/length(x)
sigma_squared_hat <- sqrt(sum((x-mu_hat)^2)/length(x))

list(estimated_mu=mu_hat,estimated_sigma=sigma_squared_hat)
```

```
## $estimated_mu
## [1] 1.275528
##
## $estimated_sigma
## [1] 2.005976
```

2.3 Optimize the minus log-likelihood function with initial parameters mu=0 and sigma=1. Try both Conjugate Gradient method and BFGS algorithm with gradient specified and without.

Gradient is the partial derivative with respect to μ and σ

$$Gradient \quad \nabla f(\vec{x}) = (\frac{\partial f(\vec{x_1})}{\partial x_1}, ..., \frac{\partial f(\vec{x_n})}{\partial x_n})^T$$

$$\frac{\partial}{\partial \mu}\mathcal{L}^*(\theta) = \frac{1}{\sigma^2}\sum_i (x_i - \mu)$$

$$\frac{\partial}{\partial \sigma}\mathcal{L}^*(\theta) = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_i (x_i - \mu)^2$$

fuction for gradient

```
#$$\frac{\partial}{\partial\sigma^2}\mathcal{L}^*(\theta)=-\frac{1}{2\sigma^2}+\frac{1}{2\sigma^3}\sum_
gradient <- function(par){
        n = length(x)
        mu = par[1]
        sigma = par[2]
        c(-(1/sigma^2)*sum(x-mu),-(-(1/sigma)*n+(1/sigma^3)*sum((x-mu)^2)))

}
```

Conjugate Gradient method - with gradient specified

```
CG_gr <-optim(par=c(0,1), fn=loglik, gr=gradient, method="CG")
```

```
## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced
```

```
CG_gr
```

```
## $par
## [1] 1.275528 2.005976
##
## $value
## [1] 211.5069
##
```

```
## $counts
## function gradient
##       53       17
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Conjugate Gradient method - without gradient specified(default Gaussian Markov kernel is used)

```
CG <- optim(c(0,1), gr=NULL, loglik, method="CG")
```

```
## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced
```

```
CG
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      201       35
##
## $convergence
## [1] 0
##
## $message
## NULL
```

BFGS - with gradient specified

```
BFGS_gr <- optim(c(0,1), loglik, gr=gradient, method="BFGS")
```

```
## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced
```

```
BFGS_gr
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
```

```
##
## $counts
## function gradient
##       38       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

BFGS - without gradient specified(default Gaussian Markov kernel is used)

```
BFGS <- optim(c(0,1), loglik, gr=NULL, method="BFGS")
```

```
## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced

## Warning in log(sigma): NaNs produced
```

```
BFGS
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       37       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Comparative table

```
df_table = data.frame("Algorithm" = c("CG_gr", "CG", "BFGS_gr", "BFGH"),
                       "value" = c(CG_gr$value, CG$value, BFGS_gr$value, BFGS$value),
                       "count_function" = c(CG_gr$counts[1], CG$counts[1],
                                            BFGS$counts[1], BFGS_gr$counts[1]),
                      "count_gradient" = c(CG_gr$counts[2], CG$counts[2],
                                           BFGS$counts[2], BFGS_gr$counts[2]))
library(knitr)
library(kableExtra)

kable(df_table, booktabs=T,
      col.names=c("Algorithms", "Value",
```

Table 1: comparative table for algorithms

Algorithms	Value	count_Function	count_gradient
CG_gr	211.5069	53	17
CG	211.5069	201	35
BFGS_gr	211.5069	37	15
BFGH	211.5069	38	15

```
                  "count_Function", "count_gradient"),
      caption = "comparative table for algorithms")
```

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

Logarithm is always increasing function, so when you are maximixing the value, you will get the same result rather you are maximzing liklihood or maximizing log-liklihood. The reason why it is better to maximize the log-liklihood is when you are taking logarithm, it decreases the magnitude of value, so it simplifies the computational procedure, which will prevent the computational from underflow or overflow.

For all 4 algorithms, we got the same parameter values 1.275528 and 2.005977, and the same value for the Value. For each algorithm we were able to find out that with setting gradient argument, the number or function called and the count of gradient is lower then without setting the argument. Especially for Conjugate Gradient method, we can see that the count for Function is decreased a lot with setting gradient argument. From the table above, BFGS method with gradient argument specified seems to be the best choice.