Computational statistics Lab 02 Report

Karthikeyan Devarajan - Karde799

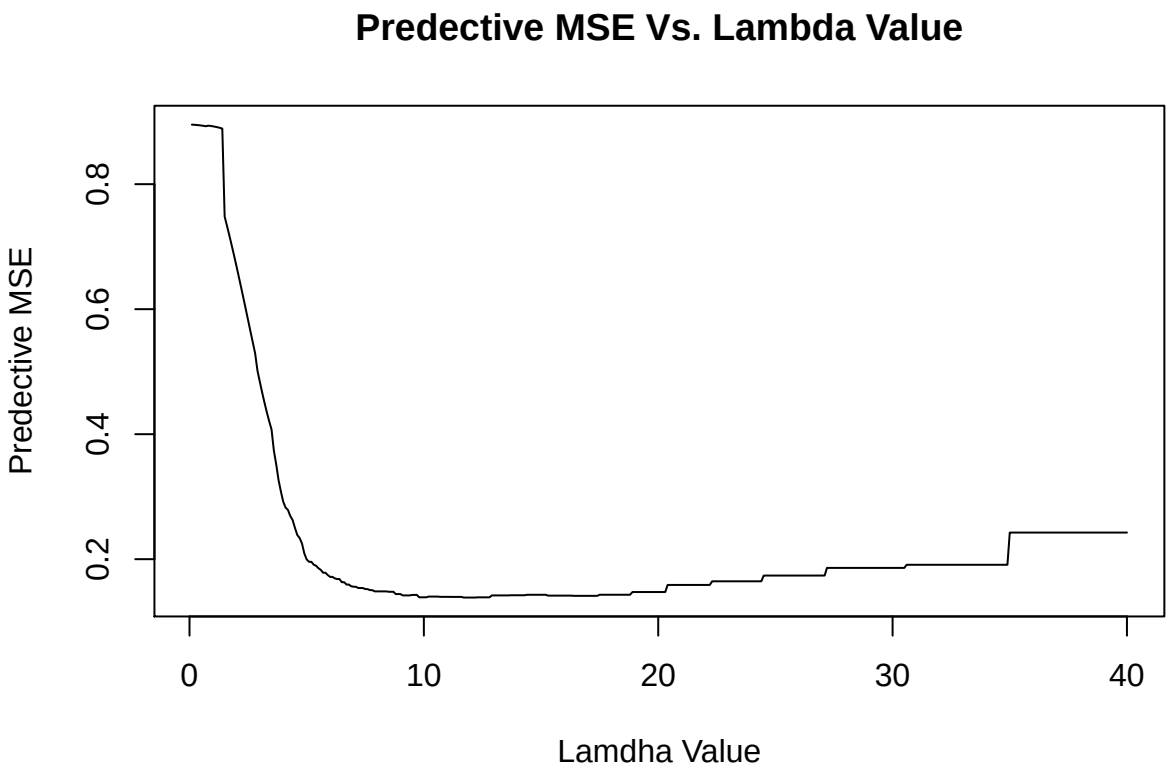
Question 1: Optimizing a model parameter

Part 02

```
# Part 02

myMSE = function(Lambda, pars, iterCount = FALSE) {
  fit = loess(formula = Y ~ X,data = pars,enp.target = Lambda)

  fYpred = predict(fit,newdata = pars$Xtest)

  sum_of_dif = 0
  for (index in 1:length(fYpred)) {
    predError = ((pars$Ytest[index] - fYpred[index]) ** 2)
    if (!is.na(predError)) {
      sum_of_dif = sum_of_dif + predError
    }
  }

  pred_MSE = (1 / length(pars$Xtest)) * sum_of_dif

  # --- Iteration Counter --- #
  if (iterCount) {
    if(!exists("iterForMyMSE")){
      assign("iterForMyMSE",
             value = 1,
             globalenv())
    } else {
      currentVal = get("iterForMyMSE")
      assign("iterForMyMSE",
             value = currentVal + 1,
             globalenv())
    }
  }
  # --- Iteration Counter --- #

  return(pred_MSE)
}
```

Part 03

Part 04




```
## Optimal Lambda Value are :- 11.7 11.8 11.9 12 12.1 12.2

## Minimum MSE value is :- 0.1386422

## No of iterations :-  400
```

There were 6 optimal values which have the Minimun MSE value (0.1386422)

We need to go through with the all possible lambda value in order to find optimal 'Lambda' Value (400 echouations)

Part 05

```
## No of iterations :-  13

## $minimum
## [1] 12.68492
##
## $objective
## [1] 0.138957
```

13 echouations were required to calculate optimal Lambda value. But for the step 4, 400 iterations were required to calculate optimal Lambda value. Hence 'optimize()' function have less computational time.

The optimize() functions normally uses Nelder-Mead in which the maximum number of iteration is 500 for it.In step 4, the iteration is manually done so it is not optimized. Whereas in optimize() function, the number of iteration is optimized based on the time for each iteration. So it will excute the number of iterations based on that.

Part 06

```
## No of iterations :-  3

## $par
## [1] 35
##
## $value
## [1] 0.2425965
##
## $counts
## function gradient
##        1        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

BFGS follows the Quasi Newton method. This generates an approxiamtion value. This calculation does not converges. It stops after one calculation (iteration).

Question 2: Maximizing likelihood

Part 01

Part 02

$$f_x(x_i;\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} exp(-\frac{(x_i-\mu)^2}{2\sigma^2})$$

The likelihood function is:-

$$L(f_x(x_i;\mu,\sigma^2)) = \prod_{i=1}^{n} f_x(x_i;\mu,\sigma^2)$$

$$L(f_x(x_i;\mu,\sigma^2)) = (2\pi\sigma^2)^{-n/2} exp(-\frac{1}{2\sigma^2}\sum_{x=1}^{n}(x_i-\mu)^2)$$

The log-likelihood function is:-

$$ln(L(f_x(x_i;\mu,\sigma^2))) = ln[(2\pi\sigma^2)^{-n/2} exp(-\frac{1}{2\sigma^2}\sum_{x=1}^{n}(x_i-\mu)^2)]$$

$$ln(L(f_x(x_i;\mu,\sigma^2))) = ln[(2\pi\sigma^2)^{-n/2}] + ln[exp(-\frac{1}{2\sigma^2}\sum_{x=1}^{n}(x_i-\mu)^2)]$$

$$ln(L(f_x(x_i;\mu,\sigma^2))) = -\frac{n}{2}ln[(2\pi\sigma^2)] - \frac{1}{2\sigma^2}\sum_{x=1}^{n}(x_i-\mu)^2$$

Apply n = 100;

$$ln(L(f_x(x_i;\mu,\sigma^2))) = -\frac{100}{2}ln[(2\pi\sigma^2)] - \frac{1}{2\sigma^2}\sum_{x=1}^{100}(x_i-\mu)^2$$

$$ln(L(f_x(x_i;\mu,\sigma^2))) = -\frac{100}{2}ln(2\pi) - \frac{100}{2}ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{x=1}^{100}(x_i-\mu)^2$$

maximum likelihood estimators for μ

Lets start by taking the Gradient with respect to μ

$$-0 - 0 - \frac{1*2}{2\sigma^2}\sum_{x=1}^{100}(x_i-\mu) = 0$$

$$-\frac{1}{\sigma^2}\sum_{x=1}^{100}(x_i-\mu) = 0$$

$$\sum_{x=1}^{100}(x_i-\mu) = 0$$

$$(x_1-\mu) + (x_2-\mu) + ... + (x_{100}-\mu) = 0$$

$$(x_1-\mu) + (x_2-\mu) + ... + (x_{100}-\mu) = 0$$

$$\sum_{x=1}^{100}(x_i) - 100\mu = 0$$

$$\mu = \frac{1}{100}\sum_{x=1}^{100}(x_i)$$

$$\hat{\mu}_{MLE} = \frac{1}{100}\sum_{x=1}^{100}(x_i)$$

maximum likelihood estimators for σ

Lets start by taking the Gradient with respect to σ

$$-0 - (\frac{100}{2} * \frac{2}{\sigma}) + \frac{2}{2\sigma^3} \sum_{x=1}^{100} (x_i - \mu)^2 = 0$$

$$-(\frac{100}{\sigma}) + \frac{1}{\sigma^3} \sum_{x=1}^{100} (x_i - \mu)^2 = 0$$

$$\frac{1}{\sigma^3} \sum_{x=1}^{100} (x_i - \mu)^2 = \frac{100}{\sigma}$$

$$\frac{1}{\sigma^2} \sum_{x=1}^{100} (x_i - \mu)^2 = 100$$

$$\frac{1}{100} \sum_{x=1}^{100} (x_i - \mu)^2 = \sigma^2$$

$$\hat{\sigma}_{MLE} = \sqrt{\frac{1}{100} \sum_{x=1}^{100} (x_i - \hat{\mu})^2}$$

```
## MLE for Mu =  1.275528
```

```
## MLE for Sigma =  2.005976
```

Part 03

```
## [1] "Conjugate Gradient method without gradient"
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      180       33
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
## [1] "BFGS method without gradient "
```

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

```
## [1] "Conjugate Gradient method with gradient"
```

```
## $par
## [1] 1.275528 2.005976
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       53       17
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
## [1] "BFGS method with gradient"
```

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##       38       15
##
## $convergence
## [1] 0
##
## $message
## NULL
```

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

When we check the likelihood function, it does have the $\frac{1}{(2\pi\sigma^2)^{n/2}}$ section. So likelihood is becoming so small when the number of data items are getting higher. So in this particular example n = 100. Hence value will be 0 when it calculate computationally. So it's better to check the log-likelihood.

For example, In $\prod_{i=0}^{N} f(y_i, x_i, \theta)$ the y_i and x_i are depending on the conditions of the parameter θ. When you take log for the above function then the function will turn into $\sum_{i=0}^{N} log(f(y_i, x_i, \theta))$.Since log is monotonic function, optimizing it will be same as optimizing the original function but whereas the shape of likelihood function is complicated. The log function will rapidly concave than the likelihood function.

Part 04

Nelder-Mead

This method try to minimize the function $f(x)$ for $x \in R^n$.The shrinking will be done based on $x_i = x_o + \sigma(x_i - x_1)$. The x_o is the centroid from (x_1, x_n). Here $n = n - 1$. The first point will be found based on $x_\gamma = x_o + \alpha(x_o + x_n)$, alpha should be always positive. Then it will be expanded based on $x_e = x_o + \gamma(x_\gamma + x_n)$, $\gamma > 1$. This will checking for the better points and contraction will be done based on if $f(x_c) > f(x_n)$ and the point will be $x_c = x_o + \rho(x_n - x_0)$. Then it will be iterated many time untill the worst points are replaced by optimized function.

BFGS
This is based on the neton formula:
$B_k p_k = \Delta f(x_k)$ where B_k is the Hessian Matrix and $\Delta f(x_k)$ is the gradient of the function. The direction is used given by the line p_k, the minimizing is done for $f(x_k + \gamma p_k)$ where $\gamma > 0$. With α_k as the minimixzation value of above formula, one dimension optimization is done, $s_k = \alpha_k p_k$. It is updated by $x_{k+1} = x_k + s_k$. Then the $y_k = \Delta f(x_{k+1}) - \Delta f(x_k)$.
With the previous value,
$B_{k+1} = B_k + (y_k y_k^T / y_k^T s_k) - (B_k s_k s_k^T B_k^T / s_k^T B_k s_k)$. The first time, the matrix B_o is intialized as I matrix.

L-BFGS-B
This is modified form of BFGS to reduce the memory space. $l \leq x \leq u$ and $x, l, u \in R^n$. The objective function is a quadratic function of the points is
$m_k(x) = f(x_k) + g_k(x-x_k) + 1/2(x-x_k)^TB_k(x-x_k)$
Appendix

```
knitr::opts_chunk$set(echo = TRUE)
RNGversion(min(as.character(getRversion()),"3.6.2"))

# Part 01

mortality_data = read.csv2(file.choose())

LMR = log(mortality_data$Rate) # natural logarithm of Rate
mortality_data = cbind(mortality_data, LMR)

n = dim(mortality_data)[1]
#set.seed(123456)
set.seed(12345, kind = "Mersenne-Twister", normal.kind = "Inversion")
id = sample(1:n, floor(n * 0.5))
train = mortality_data[id ,]
test = mortality_data[-id ,]
# Part 02
```

```r
myMSE = function(Lambda, pars, iterCount = FALSE) {
  fit = loess(formula = Y ~ X,data = pars,enp.target = Lambda)

  fYpred = predict(fit,newdata = pars$Xtest)

  sum_of_dif = 0
  for (index in 1:length(fYpred)) {
    predError = ((pars$Ytest[index] - fYpred[index]) ** 2)
    if (!is.na(predError)) {
      sum_of_dif = sum_of_dif + predError
    }
  }

  pred_MSE = (1 / length(pars$Xtest)) * sum_of_dif

  # --- Iteration Counter --- #
  if (iterCount) {
    if(!exists("iterForMyMSE")){
      assign("iterForMyMSE",
             value = 1,
             globalenv())
    } else {
      currentVal = get("iterForMyMSE")
      assign("iterForMyMSE",
             value = currentVal + 1,
             globalenv())
    }
  }
  # --- Iteration Counter --- #

  return(pred_MSE)
}


# Part 03

Lambda_array = seq(from = 0.1,
                   to = 40,
                   by = 0.1)

morta_list = list('X' = train$Day,
                  'Y' = train$LMR,
                  'Xtest' = test$Day,
                  'Ytest' = test$LMR)

my_MSE_vect = c()

iterForMyMSE = 0
for (item in 1:length(Lambda_array)) {
  my_MSE_vect[item] = myMSE(Lambda_array[item],morta_list,iterCount = TRUE)
}

# Part 04
```

```
plot(x = Lambda_array,
     y = my_MSE_vect,
     type = 'l',
     xlab = 'Lamdha Value',
     ylab = 'Predective MSE',
     main = "Predective MSE Vs. Lambda Value")

# Optimal Lambda Value
optimal_lambda = Lambda_array[which(my_MSE_vect == min(my_MSE_vect))]
cat("Optimal Lambda Value are :-", optimal_lambda, '\n')
cat("Minimum MSE value is :-", min(my_MSE_vect), '\n')
cat("No of iterations :- ", iterForMyMSE, '\n')
#min(my_MSE_vect)
iterForMyMSE = 0
min_val <- optimize(f = myMSE,
                    interval = Lambda_array,
                    pars = morta_list,
                    iterCount = TRUE,
                    tol = 0.01)
cat("No of iterations :- ", iterForMyMSE, '\n')
print(min_val)
iterForMyMSE = 0
opt_val <- optim(par = 35,fn = myMSE,method = 'BFGS',iterCount = TRUE,pars = morta_list)

cat("No of iterations :- ", iterForMyMSE, '\n')
print(opt_val)
load(file.choose())
# Part 02

getMeanMLE = function(showData = TRUE) {
  dataSum = 0
  for (val in data) {
    dataSum = dataSum + val
  }
  meanMLE = dataSum / length(data)
  if (showData == TRUE) {
    return(cat('MLE for Mu = ', meanMLE, '\n'))
  } else {
    return(meanMLE)
  }
}

getVarianceMLE = function() {
  res = 0
  mu_hat = getMeanMLE(showData = FALSE)
  for (val in data) {
    res = res + ((val - mu_hat)^2)
  }
  varianceMLE = sqrt(res / 100)
  return(cat('MLE for Sigma = ', varianceMLE, '\n'))
}

getMeanMLE()
```

```r
getVarianceMLE()
# Part 03
minusLogLik = function(x) {
  mu = x[1]
  sigma = x[2]

  return(((length(data) / 2) * log(pi*2)) +
           ((length(data) / 2) * log(sigma ^ 2)) +
           ((1 / (2 * (sigma ^ 2))) * sum((data - mu) ^ 2))
  )
}

grr = function(x) {
  ## Gradient of 'minusLogLik'
  mu = x[1]
  sigma = x[2]

  first_grad = -((1 / (sigma ^ 2)) * sum(data - mu))
  second_grd = (length(data) / sigma) - ( (1 / (sigma ^ 3)) * sum((data - mu) ^ 2))
  return(c(first_grad, second_grd))
}

cg_wo_grad = optim(par = c(0,1),
                   fn = minusLogLik,
                   method = 'CG')

bfgs_wo_grad = optim(par = c(0,1),
                   fn = minusLogLik,
                   method = 'BFGS')

cg_wi_grad = optim(par = c(0,1),
                   fn = minusLogLik,
                   method = 'CG',
                   gr = grr)

bfgs_wi_grad = optim(par = c(0,1),
                     fn = minusLogLik,
                     method = 'BFGS',
                     gr = grr)


# cat('Conjugate Gradient method without gradient ', cg_wo_grad, '\n')
# cat('BFGS method without gradient ', bfgs_wo_grad, '\n')
# cat('Conjugate Gradient method with gradient', cg_wi_grad, '\n')
# cat('BFGS method with gradient', bfgs_wi_grad, '\n')

print("Conjugate Gradient method without gradient")
print(cg_wo_grad)
print("BFGS method without gradient ")
print(bfgs_wo_grad)
print("Conjugate Gradient method with gradient")
print(cg_wi_grad)
print("BFGS method with gradient")
```

```r
print(bfgs_wi_grad)
```