Computer Lab 2 Computational Statistics

Zijie Feng (zijfe244), Phillip Hölscher (phiho267)

24 1 2019

Contents

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. 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:

```
# import the data
data = read.csv2("mortality_rate.csv")

# the LMR - natural logarithm of Rate
data$LMR = log(data$Rate)
```

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):

$$predictiveMSE = \frac{1}{length(test)} \sum_{i \leq length(test)} (Ytest[i] - fYpred(X[i]))^2,$$

where fYpred(X[i]) is the predicted value of Y if X is X[i]. Read on R's functions for prediction so that you do not have to implement it yourself.

```
# the myMSE() function
# input of the function- parameter lambda & list(vector(X, Y, Xtest, Ytest)) pars
# list(vector(X, Y, Xtest, Ytest)) pars
input_vector = list(X = train$Day, Y = train$LMR , Xtest = test$Day, Ytest = test$LMR)

# create an empty counter
counter = 0

myMSE = function(lambda,pars){
# create the model
  model = loess(formula = pars$Y ~ pars$X,
                enp.target = lambda)# just use enp.target or
                #,span = 0.75) # default - smoothing parameter
  #make the prediction
  pred <- predict(object = model,
                  newdata = pars$Xtest)
  # calculate the mse
  predictiveMSE = (1/length(pars$Ytest)) * sum((pars$Ytest - pred)^2)

  #print and return the result
  counter <<- counter + 1
  #print(predictiveMSE)
  return(predictiveMSE)
}
```

3. Estimate MSEs by `myMSE()`

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: $\lambda = 0.1,0.2,\ldots,40$

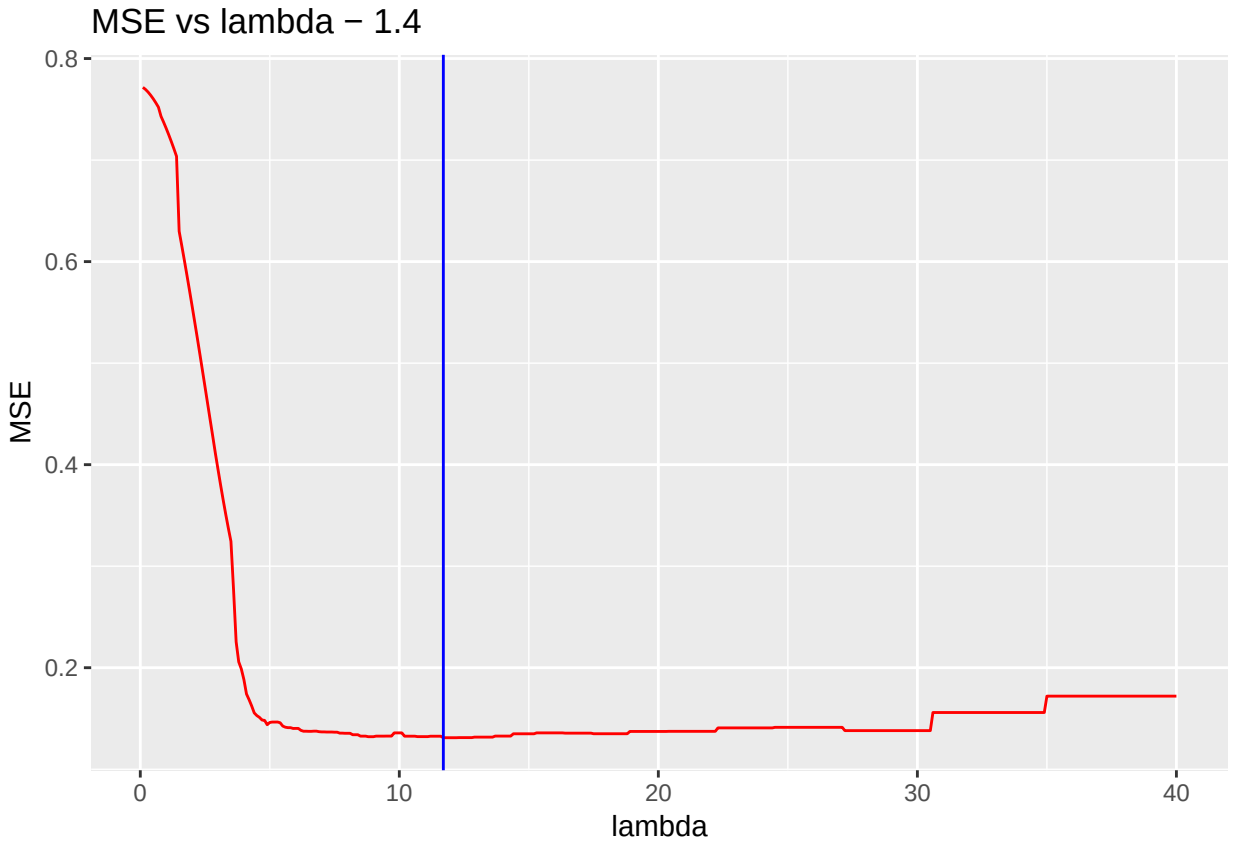
```
# initialize lambdas
lambdas = seq(from = 0.1, to = 40, by = 0.1)

# use the function myMSE - for all lambda values
mse_result = c()
for (i in 1:length(lambdas)) {
  mse_result[i] = myMSE(lambdas[i], input_vector)
}
```

4. Create a plot of the MSE vs. λ

and comment on which λ value is optimal. How many evaluations of myMSE() were required (read ?optimize) to find this value?

The plot:



This subtask is answered in section 1.6.

5. Use `optimize()` function

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

```
# set the counter to 0
counter = 0
opti = optimize(f = myMSE,
                interval = c(0.1,40),
                pars = input_vector, # specify the input for the function  pars
                tol = 0.01) # dedired accuracy
```

This subtask is answered in section 1.6.

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()
opti2 = optim(par = 35,
              fn = myMSE,
              method = "BFGS",
              pars= input_vector)
```

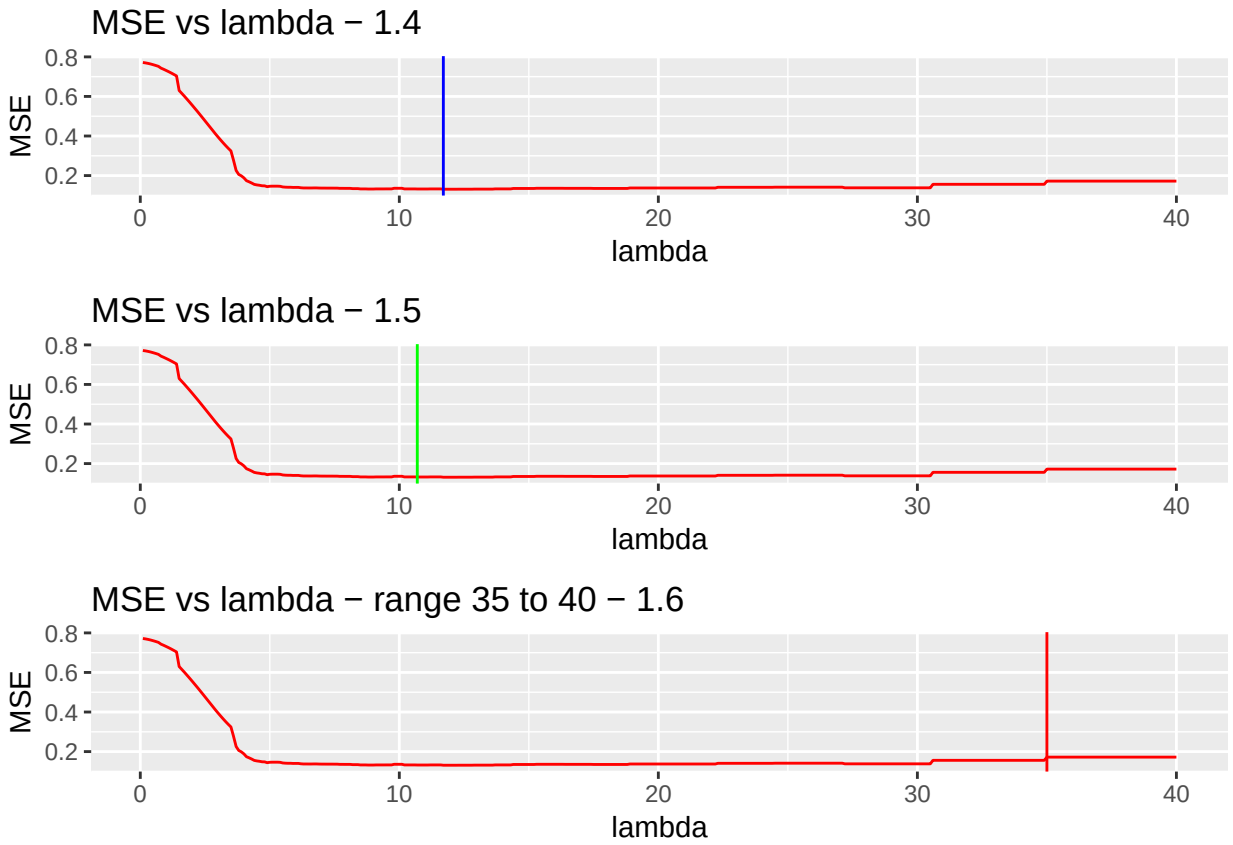
	Minumium.MSE	Optimal.lambda	Iter..of.myMSE
Result 1.4	0.1310470	11.70000	400
Result 1.5	0.1321441	10.69361	18
Result 1.6	0.1719996	35.00000	1

Above can we see the optimal MSE, optimal λ and the number of evaluations were used. Since we implemented a for loop with 400 iterations to evaluate all the lambda values for the best λ-value. At the 117 iteration did we find the minimal MSE value.

According to `help()`, the `optimize()` function was able to find the optimal value of λ by a combination of golden section search and successive parabolic interpolation. The number of iterations the function `myMSE()` needed is 18. Its minimum MSE is larger than the one in step 4, but `optimize()` is much quicker and effective.

In the last run did we evaluated the λ-interval from 35 to 40. Method "BFGS" is a quasi-Newton method, which could not only operate as fast as Newton method but also search the local minimum effciently. The output *counts* shows `optim()` evaluates `myMSE()` function once and its gradient once to build up a picture of the surface to be optimized. The table below shows that it found the best value of λ in the point 35. Which is the optimal values for this specific (local) interval, but not for the whole (global) interval. Thus, `optim()` is the fastest function to find optimal λ, but users should be careful with the initial values.

Plot *MSE vs* λ of task 1.4, 1.5 and 1.6 combined



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.

```
rm(list = ls())
# load the data into the environment
load("data.RData")
```

2. Write down the log-likelihood function for 100 observations

and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use the derived formulae to obtain parameter estimates for the loaded data.

The probability density function (PDF) of normal distribution is:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in X,$$

where $X = (x_1, x_2, ..., x_{100})$. The parameters μ and σ are based on the sample X.

Its corresponding likelihood function:

$$\begin{aligned} f(x_1, x_2, ..., x_n \mid \mu, \sigma) =& \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \\ =& (\frac{1}{\sigma\sqrt{2\pi}})^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}, \end{aligned}$$

and Log-likelihood:

$$\begin{aligned} log(f(x_1, x_2, ..., x_n \mid \mu, \sigma)) =& log((\frac{1}{\sigma\sqrt{2\pi}})^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2}) \\ =& nlog(\frac{1}{\sigma\sqrt{2\pi}}) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 \\ =& -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 \\ =& -\frac{n}{2}log(2\pi) - \frac{n}{2}log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2. \end{aligned}$$

Call $log(f(x_1, x_2, ..., x_n \mid \mu, \sigma))$ as L, the derivation of the log-likelihood - μ should be

$$\frac{\partial L}{\partial \mu} = -\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu) = 0,$$

and corresponding solution is

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n}.$$

Then we derivate of the log-likelihood - σ

$$\frac{\partial L}{\partial \sigma} = -\frac{n}{\sigma} + \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{\sigma^3} = 0,$$

and its solution thereby is

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{\mu})^2}{n}}.$$

```
#data
# Use the derived formule to obtain parameter estimates for the loaded data.
# fromula for mu
mymu <- function(x) sum(x)/length(x)
mu_hat =  mymu(data)

# formula for sigma
mysd <- function(x) sqrt(sum((x-mymu(x))**2)/length(x))
sigma_hat = mysd(data)
```

Value of $\hat{\mu}$:

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

Value of $\hat{\sigma}$:

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

3. Optimize the minus log-likelihood function

with initial parameters $\mu = 0$, $\sigma = 1$. Try both Conjugate Gradient method (described in the presentation handout) and BFGS (discussed in the lecture) algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log-likelihood?

Negative log-likelihood:

$$L(y) = -log(y)$$

```
# minus loglikelihood - function
# n - number of observations
minus_loglikelihood <- function(par,x){
  mu=par[1]
  sd=par[2]
  n=length(x)
  logllk <- -log(2*pi)*n/2-log(sd**2)*n/2-sum((x-mu)**2)/(2*sd**2)
  -logllk
}

#  the gradient function
gradient_function <- function(par,x){
  dmu <- sum(par[1]-x)/par[2]**2
  dsd <- length(x)/par[2]-sum((x-par[1])**2)/(par[2]**3)
  return(c(dmu,dsd))
}
```

```r
pars <- c(mymu(data),mysd(data))
minus_loglikelihood(pars, x=data)
```

```
## [1] 211.5069
```

Optimize the minus log-likelihood function:

```r
opti_BFGS = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      x=data) #quasi-Newton method

opti_BFGS_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      gr = gradient_function,
      x=data)

opti_CG = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      x=data) #conjugate gradients method

opti_CG_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      gr = gradient_function,
      x=data)
```

	Method	Gradient.specified	mu	sigma	Num.fun	Num.gr	convergence
opti_BFGS	BFGS	FALSE	1.275527	2.005977	37	15	0
opti_BFGS_gr	BFGS	TRUE	1.275527	2.005977	38	15	0
opti_CG	CG	FALSE	1.275528	2.005977	210	35	0
opti_CG_gr	CG	TRUE	1.275528	2.005977	53	17	0

Maximizing likelihood function is a question with exponent, calculating exponent and the value with power n are very expensive compared with calculating its logarithm, since its logarithm is additive directly. Thus, maximizing log-likelihood can be more economical and effecient than maximize likelihood.

4. Did the algorithms converge in all cases?

What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

Yes, all the algorithms converge to 0, which means the data is really normal distributed and such 4 methods work well based on our data. According to the table, all the estimated μ and σ are simiar. However, the evaluations are very different among these 4 methods. CG methods have more evaluations both in function and gradient, especially the one without specified gradient. It seems that CG method is not a good choice when the gradient is not given. On the contrary, the results of two BFGS methods are very close, which might mean that BFGS method is more robust on normal distributed sample.

References

```
#sources used
#https://daijiang.name/en/2014/10/08/mle-normal-distribution/
#https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/#nll
#https://en.wikipedia.org/wiki/Likelihood_function
#https://www.statlect.com/fundamentals-of-statistics/normal-distribution-maximum-likelihood
```

Appendix

```
rm(list=ls())
Sys.setlocale(locale = "english")
knitr::opts_chunk$set(echo = TRUE, eval = TRUE)
# list packages we use in this lab
library(ggplot2)
#install.packages("gridExtra") # to put the plots togeter
library(gridExtra)
# import the data
data = read.csv2("mortality_rate.csv")

# the LMR - natural logarithm of Rate
data$LMR = log(data$Rate)
# split the data into train and test set
n=dim(data)[1]
set.seed(123456)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
# the myMSE() function
# input of the function- parameter lambda & list(vector(X, Y, Xtest, Ytest)) pars
# list(vector(X, Y, Xtest, Ytest)) pars
input_vector = list(X = train$Day, Y = train$LMR , Xtest = test$Day, Ytest = test$LMR)

# create an empty counter
counter = 0

myMSE = function(lambda,pars){
# create the model
  model = loess(formula = pars$Y ~ pars$X,
                enp.target = lambda)# just use enp.target or
                #,span = 0.75) # default - smoothing parameter
  #make the prediction
  pred <- predict(object = model,
                  newdata = pars$Xtest)
  # calculate the mse
  predictiveMSE = (1/length(pars$Ytest)) * sum((pars$Ytest - pred)^2)

  #print and return the result
  counter <<- counter + 1
  #print(predictiveMSE)
  return(predictiveMSE)
```

```
}
# initialize lambdas
lambdas = seq(from = 0.1, to = 40, by = 0.1)

# use the function myMSE - for all lambda values
mse_result = c()
for (i in 1:length(lambdas)) {
  mse_result[i] = myMSE(lambdas[i], input_vector)
}

# the min mse value
mse_min = min(mse_result)
# the lamnda index of the min vlaue
mse_min_index = which.min(mse_result)
# the min lambda value
lambda_optimal = lambdas[mse_min_index]
# create the plot of the MSE values
# data frame for the plot
mse_plot_data = data.frame(mse_values = mse_result,
                           lamda_values = lambdas)

mse_plot = ggplot(mse_plot_data, aes(x = lamda_values, y = mse_values)) +
  geom_line(color = "red") +
  ggtitle("MSE vs lambda - 1.4") +
  geom_vline(xintercept = lambda_optimal, color = "blue", size = 0.5) +
  xlab("lambda") + ylab("MSE")
mse_plot
#create data frame with result values
result14 = data.frame(
  "Minumium MSE"=mse_min,
  "Optimal lambda"=lambda_optimal,
  "Iter. of myMSE"=counter)
#knitr::kable(result14)
# set the counter to 0
counter = 0
opti = optimize(f = myMSE,
                interval = c(0.1,40),
                pars = input_vector, # specify the input for the function  pars
                tol = 0.01) # dedired accuracy
#create data frame with result values
result15 = data.frame(
  "Minumium MSE"=opti$objective,
  "Optimal lambda"=opti$minimum,
  "Iter. of myMSE"=counter)
#knitr::kable(result15)

mse_plot2 = ggplot(mse_plot_data, aes(x = lamda_values, y = mse_values)) +
  geom_line(color = "red") +
  ggtitle("MSE vs lambda - 1.5 ") +
  geom_vline(xintercept = opti$minimum, color = "green", size = 0.5) +
  xlab("lambda") + ylab("MSE")
#?optim()
opti2 = optim(par = 35,
```

```
              fn = myMSE,
              method = "BFGS",
              pars= input_vector)

#create data frame with result values
result16 = data.frame(
  "Minumium MSE"=opti2$value,
  "Optimal lambda"=opti2$par,
  "Iter. of myMSE"=opti2$counts[1])
#knitr::kable(result16)
# create a table for the result for 1.4, 1.5, 1.6
result1 = rbind(result14,result15,result16)
rowname = c("Result 1.4", "Result 1.5", "Result 1.6")
rownames(result1) = rowname
knitr::kable(result1)
mse_plot3 = ggplot(mse_plot_data, aes(x = lamda_values, y = mse_values)) +
  geom_line(color = "red") +
  ggtitle("MSE vs lambda - range 35 to 40 - 1.6") +
  geom_vline(xintercept = opti2$par, color = "red", size = 0.5) +
  xlab("lambda") + ylab("MSE")
grid.arrange(mse_plot, mse_plot2, mse_plot3, nrow = 3)
rm(list = ls())
# load the data into the environment
load("data.RData")
#data
# Use the derived formule to obtain parameter estimates for the loaded data.
# fromula for mu
mymu <- function(x) sum(x)/length(x)
mu_hat =  mymu(data)

# formula for sigma
mysd <- function(x) sqrt(sum((x-mymu(x))**2)/length(x))
sigma_hat = mysd(data)
mu_hat
sigma_hat
# minus loglikelihood - function
# n - number of observations
minus_loglikelihood <- function(par,x){
  mu=par[1]
  sd=par[2]
  n=length(x)
  logllk <- -log(2*pi)*n/2-log(sd**2)*n/2-sum((x-mu)**2)/(2*sd**2)
  -logllk
}

#  the gradient function
gradient_function <- function(par,x){
  dmu <- sum(par[1]-x)/par[2]**2
  dsd <- length(x)/par[2]-sum((x-par[1])**2)/(par[2]**3)
  return(c(dmu,dsd))
}

pars <- c(mymu(data),mysd(data))
```

```r
minus_loglikelihood(pars, x=data)

opti_BFGS = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      x=data) #quasi-Newton method

opti_BFGS_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "BFGS",
      gr = gradient_function,
      x=data)

opti_CG = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      x=data) #conjugate gradients method

opti_CG_gr = optim(par =c(0,1),
      fn = minus_loglikelihood,
      method = "CG",
      gr = gradient_function,
      x=data)

# create a data frame of results
# vector of the column names
colnames_result_dataframe <- c("Method", "Gradient.specified", "convergence", "mu", "sigma", "Num.fun",

#opti_BFGS_gr
opti_BFGS_gr_df =  data.frame("BFGS", T , opti_BFGS_gr$convergence, opti_BFGS_gr$par[1],opti_BFGS_gr$pa
colnames(opti_BFGS_gr_df) = colnames_result_dataframe

#opti_CG
opti_CG_df =  data.frame("CG", F, opti_CG$convergence, opti_CG$par[1],opti_CG$par[2],opti_CG$counts[1],
colnames(opti_CG_df) = colnames_result_dataframe

#opti_CG_gr
opti_CG_gr_df =  data.frame("CG", T, opti_CG_gr$convergence, opti_CG_gr$par[1],opti_CG_gr$par[2],opti_C
colnames(opti_CG_gr_df) = colnames_result_dataframe

# create data frame with values of first evaluation
result_df = data.frame("Method" = "BFGS",
                       "Gradient specified" = F,
                       "convergence" = opti_BFGS$convergence,
                       "mu" = opti_BFGS$par[1],
                       "sigma"= opti_BFGS$par[2],
                       "Num.fun" = opti_BFGS$counts[1],
                       "Num.gr" = opti_BFGS$counts[2])
# combind the data frame by row
result_df = rbind(result_df, opti_BFGS_gr_df)
result_df = rbind(result_df, opti_CG_df)
result_df = rbind(result_df, opti_CG_gr_df)
result_df <- result_df[,c(1,2,4,5,6,7,3)]
```

```r
# vector of row names for the table
rownames_table = c("opti_BFGS", "opti_BFGS_gr","opti_CG", "opti_CG_gr")
rownames(result_df) = rownames_table

knitr::kable(result_df)
#sources used
#https://daijiang.name/en/2014/10/08/mle-normal-distribution/
#https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/#nll
#https://en.wikipedia.org/wiki/Likelihood_function
#https://www.statlect.com/fundamentals-of-statistics/normal-distribution-maximum-likelihood
```