Computer Lab 2 Computational Statistics

Phillip Hölscher

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
data1 = read.csv2("mortality_rate.csv")

# the LMR - natural logarithm of Rate
data1$LMR = log(data1$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):

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

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

# create a 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)
}
```

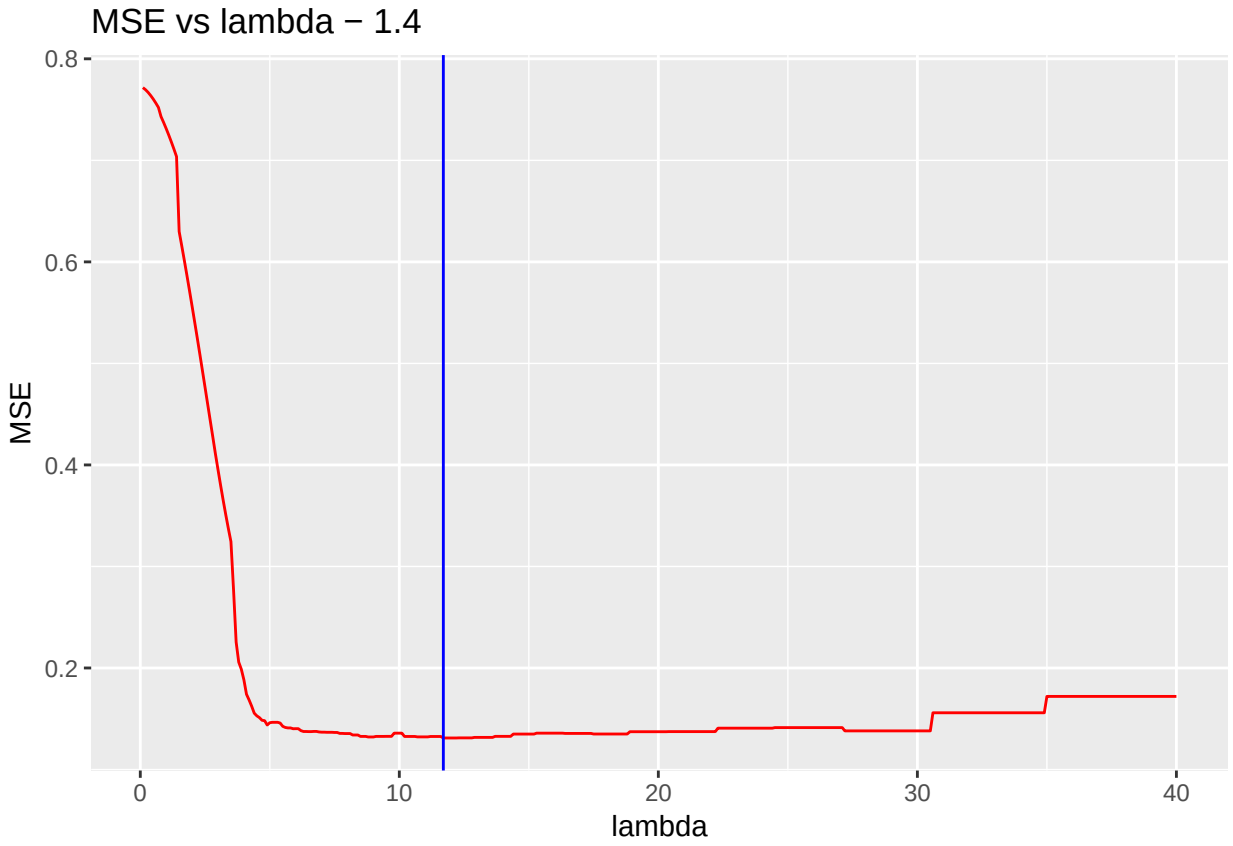
3. Use a simple approach: use function myMSE(), training and test sets with response LMR and predictor Day and the following λ values to estimate the predictive MSE values: $\lambda = 0.1,0.2,\ldots,40$

```
# initialize lambda
lambda = 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(lambda)) {
  mse_result[i] = myMSE(lambda[i], input_list)
}
```

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?

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(min(lambda),max(lambda)),
```

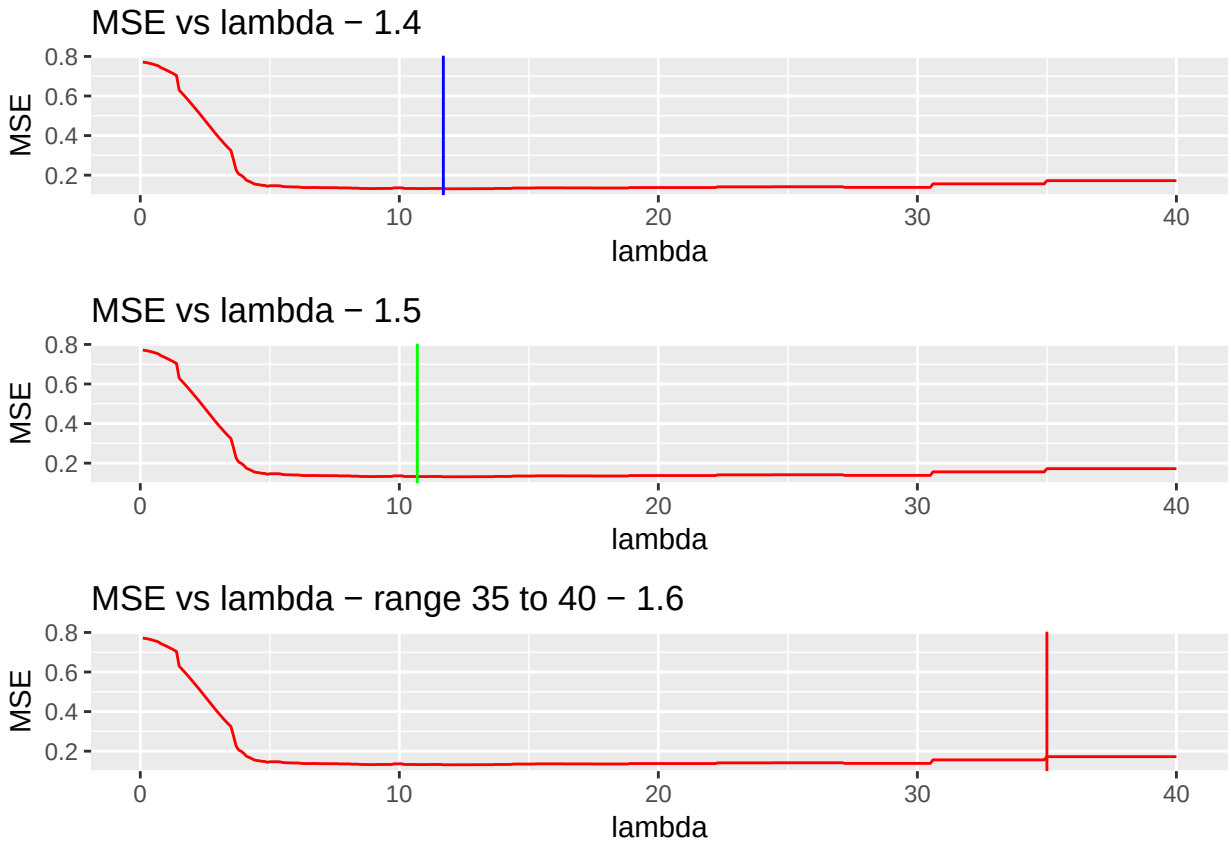
```
                pars = input_list, # specify the input for the function  pars
                tol = 0.01) # dedired accuracy
```

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()
# set the counter to 0 - not need this - counter in function included
opti2 = optim(par = 35,
              fn = myMSE,
              method = "BFGS",
              pars= input_list)
```

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



	Minumium.MSE	Optimal.lambda	Number.of.evaluations
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 to evaluate all the lambda values, did the needed 400 evaluations to find the best value for λ. At the 117 iteration did we find the min mse value.

Above can we see the results, the *optimize function* was able to find the optimal value of λ The number of iterations the function *myMSE* needed to find the best mse is 18.

In the last run did we evaluated the interval from λ 35 to 40. The table below shows that the function *optim* evaluates just ones and found the best value of λ in the point 35. Which is the optimal values for this specific interval, but not for the whole interval.

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 the data into the environment
# !!! ---------------- 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.

```
# a blog for the maximum likelihood estimator - normal distribution
#https://daijiang.name/en/2014/10/08/mle-normal-distribution/
```

The probability density function of normal distribution is:

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

100 observations - n = 100:

$$x_1, x_2, ..., x_n$$

The likelihood function:

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

Log-likelihood:

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

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

$$= -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2$$

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

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

Sovle the equation:

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

Derivate of the log-likelihood - σ

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

Sovle the equation:

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

```
#data
load("data.RData")
# fromula for mu

mu_hat =  sum(data)/length(data)

# formula for sigma
sigma_hat = sqrt(sum((data - mu_hat)^2)/length(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
# version 1 - wikipedia
minus_loglikelihood = function(paramater){
  mu = paramater[1]
  sigma = paramater[2]
  n = length(data)
  min_log = -(- (n/2) * log(2*pi*sigma^2) - (1/(2*sigma^2)) * sum((data - mu)^2))
  return(min_log)
}

# version 2 - version from the blog
minus_loglikelihood2 = function(paramater){
  mu = paramater[1]
  sigma = paramater[2]
  n = length(data)
  min_log = -(- (n/2) * log(2*pi) - n * log(sigma) - (1/(2*sigma^2)) * sum((data - mu)^2))
```

```
  return(min_log)
}
```

The gradient: Derivate of the minus log-likelihood - μ

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

Derivate of the minus log-likelihood - σ

$$\frac{\partial L}{\partial \sigma} = -\frac{n}{\sigma} + \frac{\sum_{i=1}^{n} x_i - \mu}{\sigma^3}$$

```
# the gradient function
gradient_function = function(parameter){
  mu = parameter[1]
  sigma = parameter[2]
  n = length(data)

  minus_mu = (sum(data) - mu * n)/sigma^2
  minus_sigma = -n/sigma + ((sum(data) - mu)/sigma^3)

  return(c(-minus_mu, -minus_sigma))
}
```

Optimize the minus log-likelihood function:

```
# optim - prev exercise
# opti2 = optim(par = 35,
#               fn = myMSE,
#               method = "BFGS",
#               pars= input_list)

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

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

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

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

BGFS method without gradient method:

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

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

BGFS method with gradient method:

```
## $par
## [1]  2.066098 -1.817616
##
## $value
## [1] 222.0055
##
## $counts
## function gradient
##       39        5
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Conjugate Gradient method without gradient method:

```
## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      297       45
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Conjugate Gradient method with gradient method

```
## $par
## [1] 1.020422 1.220422
##
## $value
## [1] 249.0815
##
```

```
## $counts
## function gradient
##       51        3
##
## $convergence
## [1] 0
##
## $message
## NULL
```

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?

	Method	Gradient.specified	convergence	optimal.mu	optimal.sigma	function.evaluation	gradient.evaluation
opti1	BFGS	FALSE	0	1.275527	2.005977	37	15
opti2	BFGS	TRUE	0	2.066098	-1.817616	39	5
opti3	CG	FALSE	0	1.275528	2.005977	297	45
opti4	CG	TRUE	0	1.020422	1.220422	51	3

Based on the information of the table would I choose the method with the minimum runs of function evaluations.

Appendix

```
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)
#install.packages("kableExtra") # to change the size of the table in 2.4
library(kableExtra)
library(knitr)
# import the data
data1 = read.csv2("mortality_rate.csv")

# the LMR - natural logarithm of Rate
data1$LMR = log(data1$Rate)
# split the data into train and test set
n=dim(data1)[1]
set.seed(123456)
id=sample(1:n, floor(n*0.5))
train=data1[id,]
test=data1[-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_list = list(X = train$Day, Y = train$LMR , Xtest = test$Day, Ytest = test$LMR)

# create a 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 lambda
lambda = 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(lambda)) {
  mse_result[i] = myMSE(lambda[i], input_list)
}
```

```r
# 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 = lambda[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 = lambda)

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,
  "Number of evaluations"=counter)
#knitr::kable(result14)
# set the counter to 0
counter = 0
opti = optimize(f = myMSE,
                interval = c(min(lambda),max(lambda)),
                pars = input_list, # 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,
  "Number of evaluations"=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()
# set the counter to 0 - not need this - counter in function included
opti2 = optim(par = 35,
              fn = myMSE,
              method = "BFGS",
              pars= input_list)

#create data frame with result values
result16 = data.frame(
  "Minumium MSE"=opti2$value,
  "Optimal lambda"=opti2$par,
  "Number of evaluations"=opti2$counts[1])
```

```r
#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
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)
result1 %>%
  kable()
# load the data into the environment
# !!! ---------------- data.RData ---------------- !!!
# a blog for the maximum likelihood estimator - normal distribution
#https://daijiang.name/en/2014/10/08/mle-normal-distribution/
#data
load("data.RData")
# fromula for mu

mu_hat =  sum(data)/length(data)

# formula for sigma
sigma_hat = sqrt(sum((data - mu_hat)^2)/length(data))
mu_hat
sigma_hat
# minus loglikelihood - function
# n - number of observations
# version 1 - wikipedia
minus_loglikelihood = function(paramater){
  mu = paramater[1]
  sigma = paramater[2]
  n = length(data)
  min_log = -(- (n/2) * log(2*pi*sigma^2) - (1/(2*sigma^2)) * sum((data - mu)^2))
  return(min_log)
}

# version 2 - version from the blog
minus_loglikelihood2 = function(paramater){
  mu = paramater[1]
  sigma = paramater[2]
  n = length(data)
  min_log = -(- (n/2) * log(2*pi) - n * log(sigma) - (1/(2*sigma^2)) * sum((data - mu)^2))
  return(min_log)
}

# the gradient function
gradient_function = function(parameter){
  mu = parameter[1]
  sigma = parameter[2]
  n = length(data)
```

```
  minus_mu = (sum(data) - mu * n)/sigma^2
  minus_sigma = -n/sigma + ((sum(data) - mu)/sigma^3)

  return(c(-minus_mu, -minus_sigma))
}

# optim - prev exercise
# opti2 = optim(par = 35,
#               fn = myMSE,
#               method = "BFGS",
#               pars= input_list)

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

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

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

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

opti_BFGS
opti_BFGS_gr
opti_CG
opti_CG_gr
# create a data frame of results
# vector of the column names
colnames_result_dataframe <- c("Method", "Gradient.specified", "convergence", "optimal.mu", "optimal.sig

# create data frame for each evaluation

#opti_BFGS
# opti_BFGS_vector = data.frame("BFGS", "False", opti_BFGS$convergence, opti_BFGS$par[1],opti_BFGS$par[

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

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

#opti_CG_gr
opti_CG_gr_df =  data.frame("CG", TRUE , opti_CG_gr$convergence, opti_CG_gr$par[1],opti_CG_gr$par[2],op
```

```r
colnames(opti_CG_gr_df) = colnames_result_dataframe

# create data frame with values of first evaluation
result_df = data.frame("Method" = "BFGS",
                       "Gradient specified" = FALSE,
                       "convergence" = opti_BFGS$convergence,
                       "optimal mu" = opti_BFGS$par[1],
                       "optimal sigma"= opti_BFGS$par[2],
                       "function evaluation" = opti_BFGS$counts[1],
                       "gradient evaluation" = 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)
# rename the rownames
rownames(result_df) = c("opti1","opti2","opti3","opti4")

result_df %>%
  kable() %>%
  kable_styling(latex_options="scale_down")
#%>%
  #kable_styling(full_width = F, position = "center")
#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
```