Computational Statistics - Lab 02

Annalena Erhard (anner218) and Maximilian Pfundstein (maxpf364)

2019-02-04

Contents

1 Question 1: Optimizing a Model Param **1**

2 Question 2: Maximizing Likelihood **4**

3 Source Code **7**

1 Question 1: Optimizing a Model Param

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

Task: 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.

Day	Rate	LMR
1	0.0014	-6.571283
2	0.0040	-5.521461
3	0.0051	-5.278515
4	0.0064	-5.051457
5	0.0075	-4.892852
6	0.0098	-4.625373

Task: 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):

$$\text{predictive MSE} = \frac{1}{\text{length(test)}} \sum_{\text{ith element in test set}} (Y_{test}[i] - fY_{pred}(X[i]))^2$$

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.

```
# Implementation of myMSE -------------------
myMSE = function(lambda, pars) {
  model = loess(Y ~ X, data=pars, enp.target = lambda)
  prediction = predict(model, newdata = pars$Xtest)
  mse = sum((prediction - pars$Ytest)^2)/length(pars$Ytest)
  #print(".")
  return(mse)
}
```

Task: 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: $\lambda = 0.1, 0.2, ...40$.

```
# parameters for the myMSE function ----------------------
pars = list(X = train$Day, Y = train$LMR, Xtest = test$Day, Ytest = test$LMR)
lambdas = seq(from = 0.1, to = 40, by = 0.1)

# applying the myMSE function to all lambdas ------------
mses = sapply(X = lambdas, FUN = myMSE, pars = pars)
```

Task: 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?

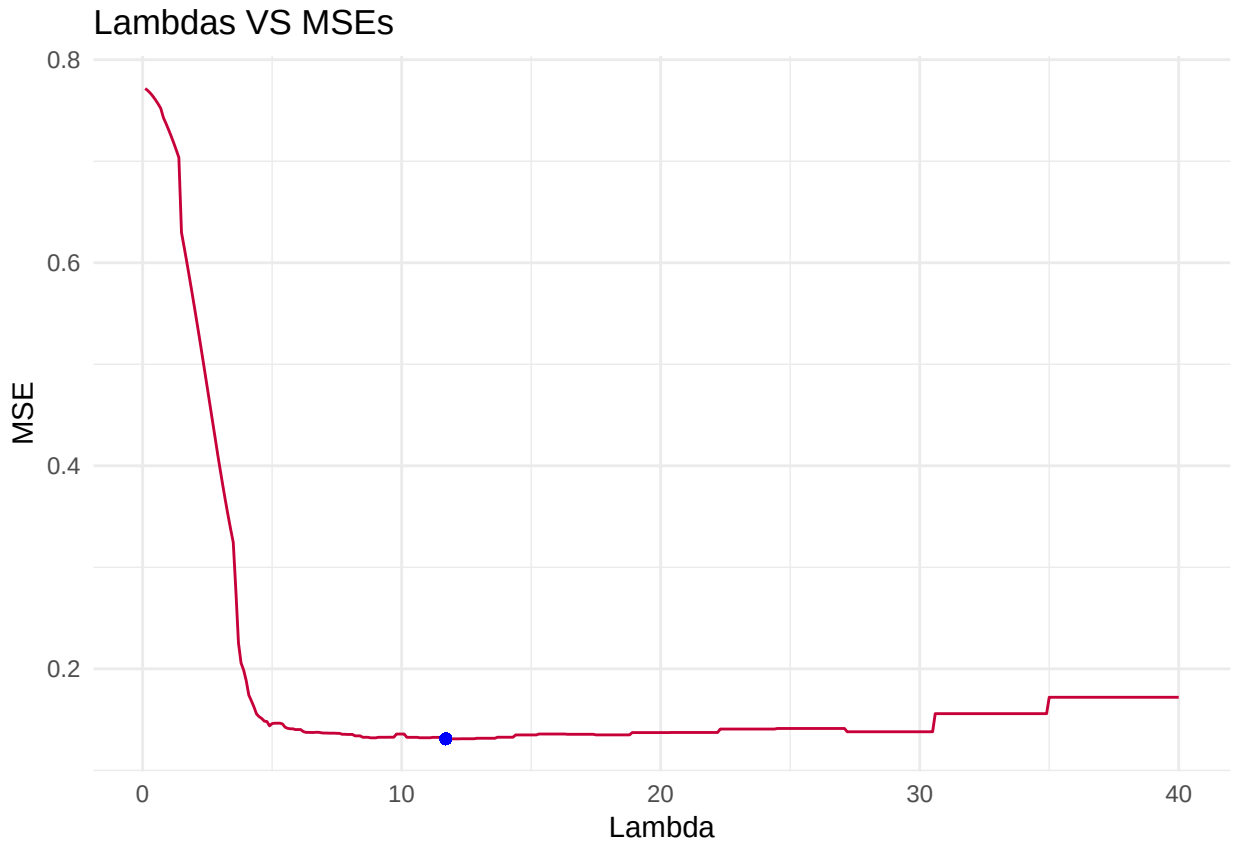
Answer: From looking at the plot, it seems like the optimal λ is between 10 and 30.

```
## [1] 11.7
```

is best λ (for the given input sequence). This minimum can also be seen at the blue point in the plot below. The function was called

```
## [1] 400
```

times.



Task: Use `optimize()` function for the same purpose, specify range for search `[0.1, 40.0]` 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.

Answer: The call for the build in `optimize()` function looks like this:

```r
o = optimize(myMSE, tol = 0.01, interval = c(0.1, 40), pars = pars)
```

This means the minimum could be found at a λ of

```
## [1] 10.69361
```

and an MSE of

```
## [1] 0.1321441
```

The function `myMSE()` was called 18 times. We counted the printed dots, we were too lazy to build a wrapper with a counter :).

Task: 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.

```r
optim(35, myMSE, method = "BFGS", pars = pars, control = list(fnscale = 1))
```

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

Answer: The function iterated only once at a starting value of `35` and is therefore at an "optimal"" lambda value of `0.1719996`. This means that the "real" minimum was not reached. This is due to teh fact that the gradient at this position is `0` and thus the algorithm stops.

Information: The function `optimize()` searches the whole interval using the golden selection search (description on the course slides), so it is looking for a local minima in the given interval. It uses a constant reduction factor α. For this to work the function has to be unimodal, which means that is only has one maxima/minima (in the given interval). So it's pretty basic and, depending on α, we need more or less iterations.

Newtons Method (Nelder-Mead) is memory heavy but therefore converges quickly. It computes an approximation for the Hessian and calculates the quasi–Newton condition or secant condition:

$$B_{k+1}(\vec{x}_{k+1} - \vec{x}_k) = \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}_k)$$

We want the current and the next itertion to be as close as possible. The computations for this can be found on the slides and won't be mentioned here. One last word about the BGFS (Broyden–Fletcher–Goldfarb–Shanno), which was used here. It needs in general more iterations than Newton, but each iteration is faster to compute, so it's actually better for large scale problems. The gradient in this case is calculated like this:

$$\nabla f(\vec{x}) = A\vec{x} - \vec{b} = r(\vec{x})$$

The gradient is therefore used to calculate an improved lambda value. Since this is `0` at the initial lambda value, the algorithm converges. In contrast to the optimize function, the minimum could not be found here. It is possible, however, that `optim()` works much more efficiently if no `0` gradients occur.

2 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.

Task: Load the data to R environment.

```
## [1]  3.8554793  2.7670692  3.3384582 -2.6535340 -0.3624106  4.4326187
```

Task: 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.

Answer: The negative log-likelihood function for the normal distribution is defined by:

$$\mathcal{L}(\mu, \sigma^2, x_1, ..., x_{100}) = \frac{n}{2} ln(2\pi) + \frac{n}{2} ln(\sigma^2) + \frac{1}{2\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2$$

The estimators are:

$$\hat{\mu}_n = \frac{1}{n} \sum_{j=1}^{n} x_j$$

and

$$\hat{\sigma}_n^2 = \frac{1}{n} \sum_{j=1}^{n} (x_j - \hat{\mu})^2$$

Taken from https://www.statlect.com/fundamentals-of-statistics/normal-distribution-maximum-likelihood, not derived by hand. How the formulas can be derived can be found on the page.

The implementation in R looks like the following:

```r
# c(mu, sigma)
neg_llik_norm = function(par) {
  n = nrow(as.matrix(data))
  p1 = (n/2)*log(2*pi)
  p2 = (n/2)*log(par[2]^2)
  sum = sum((data - par[1])^2)
  p3 = 1/(2*par[2]^2) * sum
  return(p1+p2+p3)
}
```

Task: 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?

Answer: The partial derivates for the negative log-likelihood are given by:

$$\frac{\partial \mathcal{L}(\mu, \sigma^2, x_1, ..., x_{100})}{\partial \mu} = -\frac{1}{n\sigma^2} \sum_{j=1}^{n} (x_j - \mu)$$

$$\frac{\partial \mathcal{L}(\mu, \sigma^2, x_1, ..., x_{100})}{\partial \sigma^2} = \frac{1}{2\sigma^2} \left(n - \frac{1}{\sigma^2} \sum_{j=1}^{n} (x_j - \mu)^2 \right)$$

We derived them on paper.

```
# c(mu, sigma)
neg_llik_norm_prime = function(par) {
    n = nrow(as.matrix(data))
    mu_prime = -1/(n*par[2]^2) * sum(data-par[1])
    sigma_prime = 1/(2*par[2]^2) * (n -
    (1/(par[2]^2)) * sum((data-par[1])^2))

  return(c(mu_prime, sigma_prime))
}
```

Conjugate Gradient Method:

```
optim(c(0, 1), neg_llik_norm, method = "CG")
```

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

```
optim(c(0, 1), neg_llik_norm, method = "CG", gr = neg_llik_norm_prime)
```

```
## $par
## [1] 1.275452 2.005959
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      309      101
##
## $convergence
## [1] 1
##
## $message
```

```
## NULL
```

BFGS:

```
optim(c(0, 1), neg_llik_norm, method = "BFGS")
```

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

```
optim(c(0, 1), neg_llik_norm, method = "BFGS", gr = neg_llik_norm_prime)
```

```
## $par
## [1] 1.275234 2.006185
##
## $value
## [1] 211.507
##
## $counts
## function gradient
##       37       26
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Answer: The likelihood is a product of really small values. This means that we reach the floiting point precision limits rather fast and it becomes basically impossible to calculate the likelihood at one point. Thus we use the loglikihood, because we convert the product to a (negative) sum. The make up for an optimization problem, we often use the negative loglikelihood the get rid of the negative signs.

Task: 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?

Answer: The results can be seen in the previous printouts. The property `$par` shows the optimized parameters for μ and σ. If the algorithm converged can be found in the parameter `$convergence`. Possible (intersting) results are:

- **1:** indicates that the iteration limit maxit had been reached.

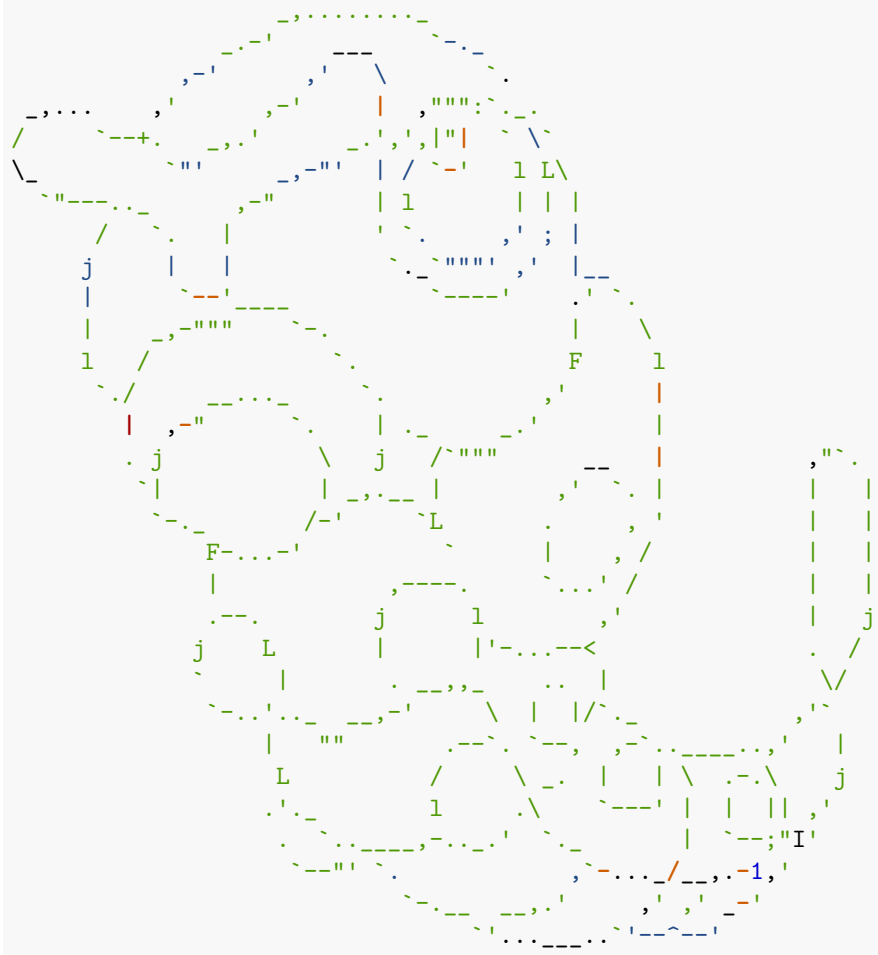
We can see, that they are always `0`, so all of them converged.

The amount of function and gradient descents can also be taken from the above printouts. They're found in the parameter `$counts`.

Therefore we recommend the following setting:

BFGS without specifying the gradient as this takes the fewest iterations.

```
Don't forget, always be nice to Cataperie!
```



3 Source Code

```
knitr::opts_chunk$set(echo = TRUE, cache = FALSE, include = TRUE, eval = TRUE)
library(ggplot2)
library(knitr)
library(gridExtra)

################################################################################
# Question 1: Optimizing a Model Param
################################################################################


# Data import --------------------------------
data = read.csv("mortality_rate.csv", sep = ";", dec = ",")
data$LMR = log(data$Rate)
```

```r
# first 6 rows of the full log- dataset --------
kable(head(data))

# division of data into train and test -------
n = dim(data)[1]
set.seed(123456)
id = sample(1:n, floor(n * 0.5))
train = data[id ,]
test = data[-id ,]


# Implementation of myMSE --------------------
myMSE = function(lambda, pars) {
  model = loess(Y ~ X, data=pars, enp.target = lambda)
  prediction = predict(model, newdata = pars$Xtest)
  mse = sum((prediction - pars$Ytest)^2)/length(pars$Ytest)
  #print(".")
  return(mse)
}

# parameters for the myMSE function ----------------------
pars = list(X = train$Day, Y = train$LMR, Xtest = test$Day, Ytest = test$LMR)
lambdas = seq(from = 0.1, to = 40, by = 0.1)

# applying the myMSE function to all lambdas ------------
mses = sapply(X = lambdas, FUN = myMSE, pars = pars)

lambdas[which.min(mses)]
length(lambdas)
df = data.frame(lambdas, mses)

ggplot(df) +
  geom_line(aes(x = lambdas, y = mses), color = "#C70039") +
  geom_point(aes(x = seq(0.1, 40, by = 0.1)[which.min(mses)],
                 y = mses[which.min(mses)], color = "min MSE"),
             colour = "blue") +
  labs(title = "Lambdas VS MSEs", y = "MSE", x = "Lambda") +
  theme_minimal()


o = optimize(myMSE, tol = 0.01, interval = c(0.1, 40), pars = pars)

o$minimum
o$objective

optim(35, myMSE, method = "BFGS", pars = pars, control = list(fnscale = 1))


##############################################################################
# Question 2: Maximizing Likelihood
##############################################################################

data2 = get(load("data.RData"))
```

```r
head(data2)


# c(mu, sigma)
neg_llik_norm = function(par) {
  n = nrow(as.matrix(data))
  p1 = (n/2)*log(2*pi)
  p2 = (n/2)*log(par[2]^2)
  sum = sum((data - par[1])^2)
  p3 = 1/(2*par[2]^2) * sum
  return(p1+p2+p3)
}

# c(mu, sigma)
neg_llik_norm_prime = function(par) {
    n = nrow(as.matrix(data))
    mu_prime = -1/(n*par[2]^2) * sum(data-par[1])
    sigma_prime = 1/(2*par[2]^2) * (n -
    (1/(par[2]^2)) * sum((data-par[1])^2))

  return(c(mu_prime, sigma_prime))
}


optim(c(0, 1), neg_llik_norm, method = "CG")
optim(c(0, 1), neg_llik_norm, method = "CG", gr = neg_llik_norm_prime)


optim(c(0, 1), neg_llik_norm, method = "BFGS")
optim(c(0, 1), neg_llik_norm, method = "BFGS", gr = neg_llik_norm_prime)


Don't forget, always be nice to Cataperie!
```