

IMT 573: Module 8 Lab

Estimation

Jenny Skytta

Due: May 22, 2022

Collaborators: List collaborators here.

Objectives

As you continue your data science career, and proceed to the rest of the data science sequence here at the UW iSchool, you will encounter more cases where estimation is a critical component of modeling data. Today, we get a first look at how these procedures work “under the hood”. This programming exercise is different from our past assignments in that I have provided all the code needed. Your job is to read and understand this code, and to demonstrate this understanding through detailed code comments. Good luck!

Instructions

Before beginning this assignment, please ensure you have access to R and RStudio; this can be on your own personal computer or on the IMT 573 R Studio Cloud.

1. Open the `08_lab_estimation.Rmd` and save a copy to your local directory. Supply your solutions to the assignment by editing `08_lab_estimation.Rmd`.
2. First, replace the “YOUR NAME HERE” text in the `author:` field with your own full name. Any collaborators must be listed on the top of your assignment.
3. Be sure to include well-documented (e.g. commented) code chunks, figures, and clearly written text chunk explanations as necessary. Any figures should be clearly labeled and appropriately referenced within the text. Be sure that each visualization adds value to your written explanation; avoid redundancy – you do not need four different visualizations of the same pattern.
4. Collaboration on problem sets is fun and useful, and I encourage it, but each student must turn in an individual write-up in their own words as well as code/work that is their own. Regardless of whether you work with others, what you turn in must be your own work; this includes code and interpretation of results. The names of all collaborators must be listed on each assignment. Do not copy-and-paste from other students’ responses or code.
5. All materials and resources that you use (with the exception of lecture slides) must be appropriately referenced within your assignment.
6. When you have completed the assignment and have **checked** that your code both runs in the Console and knits correctly when you click **Knit**. When the PDF report is generated rename the knitted PDF file to `lab8_YourLastName_YourFirstName.pdf`, and submit the PDF file on Canvas.

In this lab you will need, at minimum, the following R packages.

```
# Load standard libraries
library(tidyverse)
```

```
library(knitr) # this will keep code on the page!
opts_chunk$set(tidy.opts=list(width.cutoff=60),tidy=TRUE)
```

Doing Maximum Likelihood Estimation by Hand in R

By John Myles White on 4.21.2010

White, J.M. (2010). "Doing Maximum Likelihood Estimation by Hand in R." Available Online.

First, let's start with a toy example for which there is a closed-form analytic solution. We'll ignore that solution and use optimization functions to do the estimation. Starting with this toy example makes it easy to see how well an approximation system can be expected to perform under the best circumstances - and also where it goes wrong if you make poor programming decisions.

Suppose that you've got a sequence of values from an unknown Bernoulli variable like so:

```
p.parameter <- 0.8 # probability variable
sequence <- rbinom(10, 1, p.parameter) # random binomial with input of N, # of trials, and probability
```

Given the sequence, we want to estimate the value of the parameter, p , which is not known to us. The maximum likelihood approach says that we should select the parameter that makes the data most probable. For a Bernoulli variable, this is simply a search through the space of values for p (i.e. $[0, 1]$) that makes the data most probable to have observed.

It's worth pointing out that the analytic solution to the maximum likelihood estimation problem is to use the sample mean. We'll therefore use `mean(sequence)` as a measure of the accuracy of our approximation algorithm.

How do we find the parameter numerically? First, we want to define a function that specifies the probability of our entire data set. We assume that each observation in the data is independently and identically distributed, so that the probability of the sequence is the product of the probabilities of each value. For the Bernoulli variables, this becomes the following function:

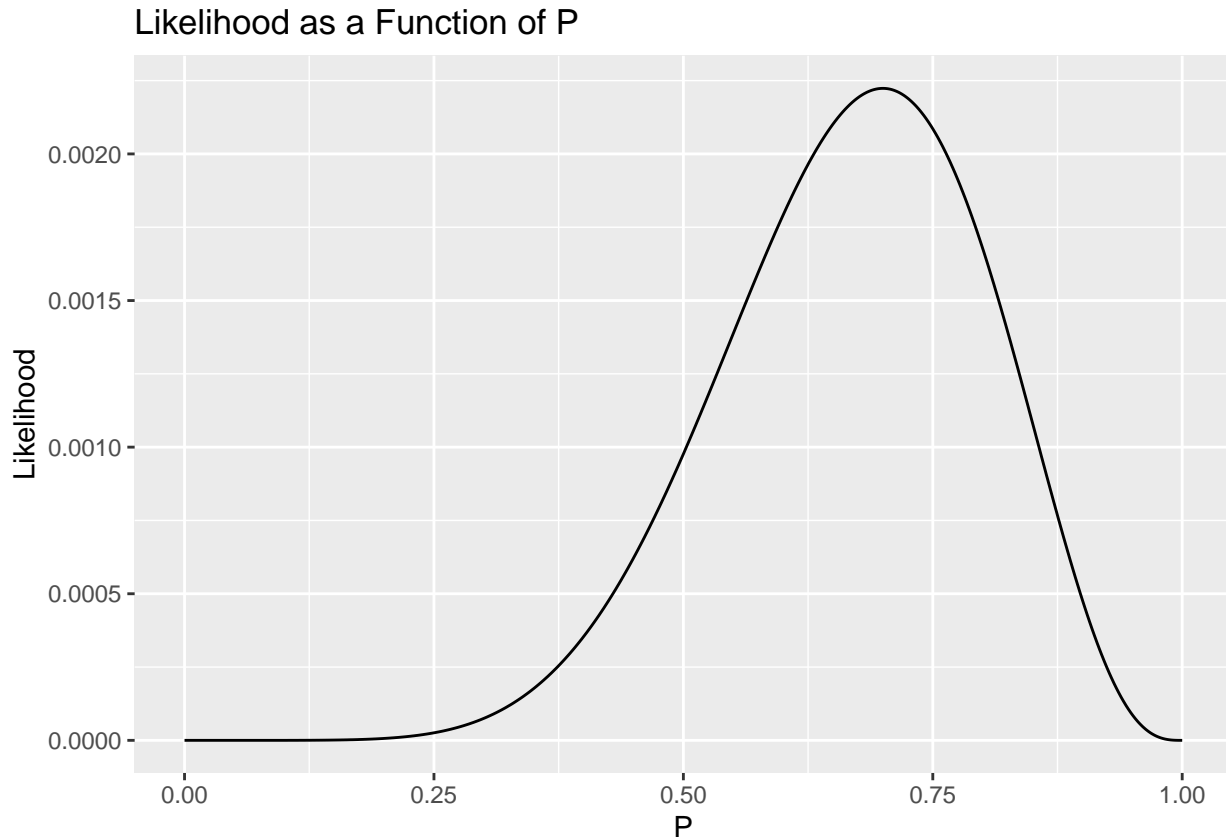
```
likelihood <- function(sequence, p.parameter) # creating function called likelihood
{
  likelihood <- 1 # a for loop that asks if sequence i = 1, multiply by p.parameter

  for (i in 1:length(sequence))
  {
    if (sequence[i] == 1)
    {
      likelihood <- likelihood * p.parameter
    }
    else # otherwise multiply input times 1 minus p.parameter
    {
      likelihood <- likelihood * (1 - p.parameter)
    }
  }

  return(likelihood)
}
```

To do maximum likelihood estimation, we therefore only need to use an optimization function to maximize this function. A quick examination of the likelihood function as a function of p makes it clear that any decent optimization algorithm should be able to find the maximum:

```
possible.p <- seq(0, 1, by = 0.001) # defining possible.p as sequence with sequence 0 to 1 in increments of 0.001
qplot(possible.p, # creating a possible.p plot
      sapply(possible.p, function(p) {likelihood(sequence, p)}),
      geom = 'line',
      main = 'Likelihood as a Function of P',
      xlab = 'P',
      ylab = 'Likelihood')
```



For single variable cases, I find that it's easiest to use R's base function `optimize` to solve the optimization problem:

```
mle.results <- optimize(function(p) {likelihood(sequence, p)}, # creating function
                        interval = c(0, 1), # optimize that searches defined interval from
                        maximum = TRUE) # lower to upper for a max of function(p)

mle.results # calling mle.results
```

```
## $maximum
## [1] 0.6999843
##
## $objective
## [1] 0.002223566
```

Here I've used an anonymous function that returns the likelihood of our current data given a value of `p`; I've also specified that the values of `p` must lie in the interval `[0, 1]` and asked `optimize` to maximize the result, rather than minimize, which is the default behavior. Examining the output of `optimize`, we can see

that the likelihood of the data set was maximized very near 0.7, the sample mean. This suggests that the optimization approximation can work. It's worth noting that the objective value is the likelihood of the data set for the specified value of p . The smallness of the objective for large problems can become a major problem. To understand why, it's worth seeing what happens as the size of the sample grows from 10 to 2500 samples:

```
error.behavior <- data.frame()  # creating dataframe

for (n in 10:2500)  # with for loop 10-2500 samples
{
  sequence <- rbinom(n, 1, p.parameter)  # random binomials, 1 and the p.parameter

  likelihood.results <- optimize(function(p) {likelihood(sequence, p)},
                                interval = c(0, 1),
                                maximum = TRUE)

  true.mle <- mean(sequence)  # mean of sequence defines true.mle

  likelihood.error <- true.mle - likelihood.results$maximum  # subtract likelihood.results optimization
  # from the true.mle and defining as likelihood.error
  error.behavior <- rbind(error.behavior,  # adding rows into dataframe with
                          data.frame(N = n, # row N, row Error, and row Algorithm
                                    Error = likelihood.error,
                                    Algorithm = 'Likelihood'))
}
```

As you can see, our approximation approach works great until our data set grows, and then it falls apart. This is exactly the opposite of what asymptotical statistical theory tells us should be happening, so it's clear that something is going very wrong. A quick examination of the results from the last pass through our loop makes clear what's wrong:

```
sequence <- rbinom(2500, 1, p.parameter)  # random binomial n of 2500, max 1 and p.parameter

likelihood.results <- optimize(function(p) {likelihood(sequence, p)}, # another optimization loop
                              interval = c(0, 1),
                              maximum = TRUE)

likelihood.results

## $maximum
## [1] 0.9999339
##
## $objective
## [1] 0
```

The likelihood of our data is numerically indistinguishable from 0 given the precision of my machine's floating point values. Multiplying thousands of probabilities together is simply not a viable approach without infinite precision. Thankfully, there's a very simple solution: replace all of the probabilities with their logarithms. Instead of maximizing the likelihood, we maximize the log likelihood, which involves summing rather than multiplying, and therefore stays numerically stable:

```
log.likelihood <- function(sequence, p) # redoing with log of p
{
  log.likelihood <- 0
```

```

for (i in 1:length(sequence))
{
  if (sequence[i] == 1)
  {
    log.likelihood <- log.likelihood + log(p)
  }
  else
  {
    log.likelihood <- log.likelihood + log(1 - p)
  }
}

return(log.likelihood)
}

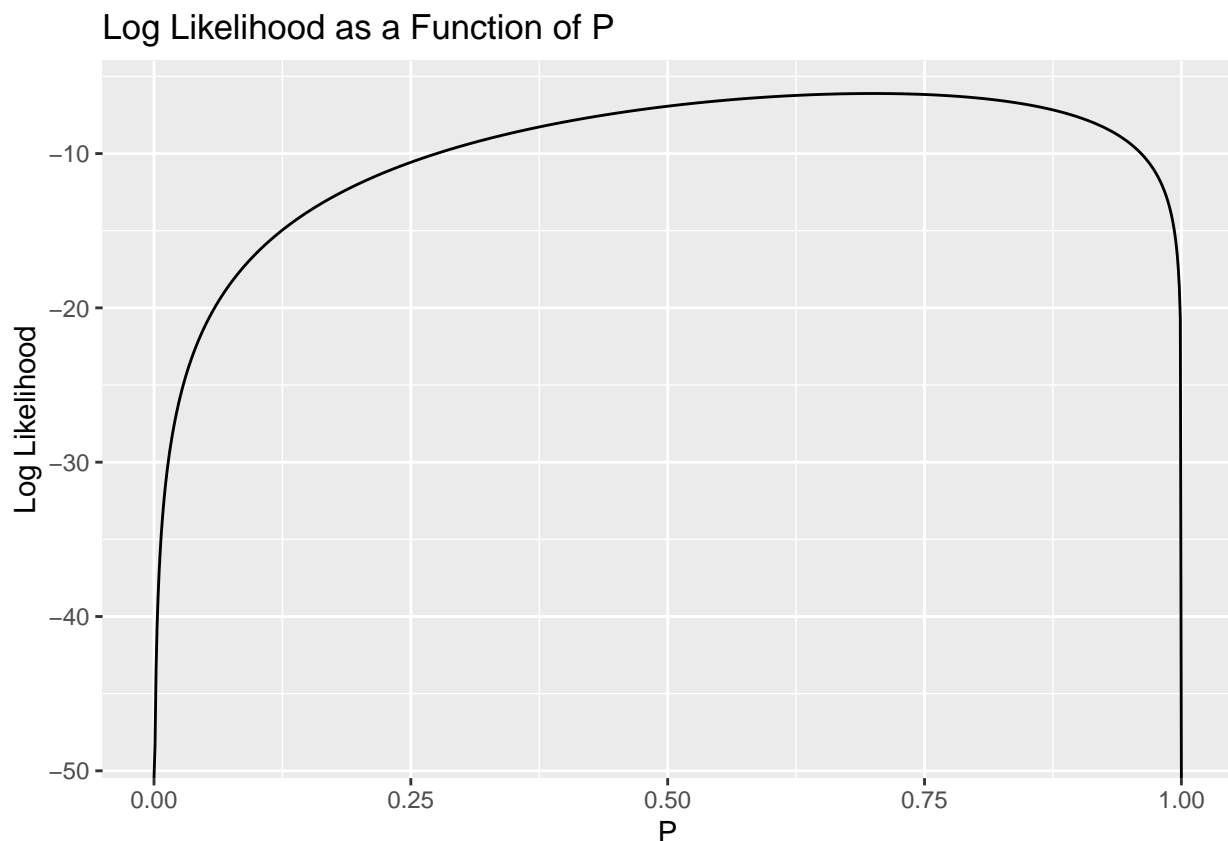
```

You can check that this problem is as easily solved numerically as the original problem by graphing the log likelihood:

```

sequence <- c(0, 1, 1, 1, 1, 1, 1, 0, 1, 0) # creating vector of numbers
possible.p <- seq(0, 1, by = 0.001) # setting sequence 0-1 by 0.001
qplot(possible.p, # plotting
      sapply(possible.p, function (p) {log.likelihood(sequence, p)}),
      geom = 'line',
      main = 'Log Likelihood as a Function of P',
      xlab = 'P',
      ylab = 'Log Likelihood')

```



And then you can rerun our error diagnostics using both approaches to confirm that the log likelihood approach does not suffer from the same numerical problems:

```
error.behavior <- data.frame() # rerunning error diagnostics

for (n in 10:2500)
{
  sequence <- rbinom(n, 1, p.parameter)

  likelihood.results <- optimize(function(p) {likelihood(sequence, p)},
                                interval = c(0, 1),
                                maximum = TRUE)

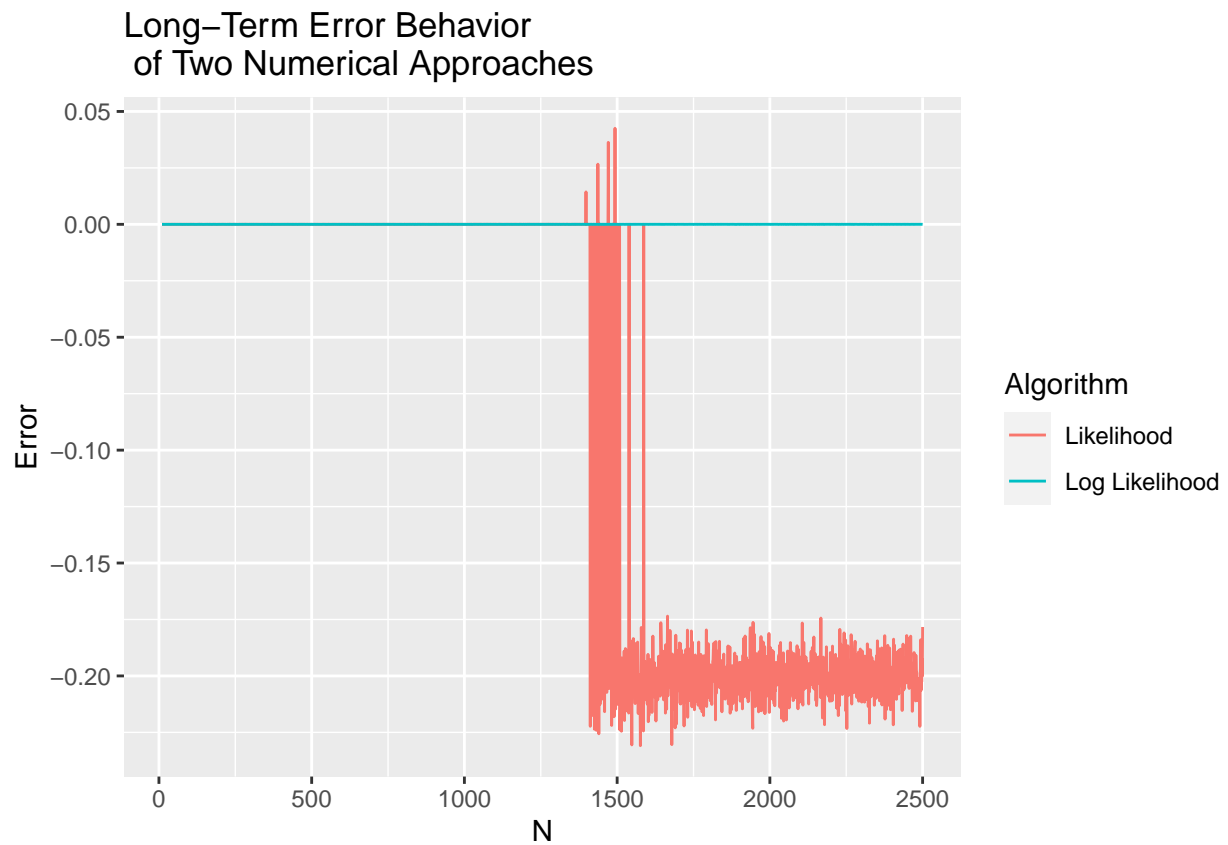
  log.likelihood.results <- optimize(function(p) {log.likelihood(sequence, p)},
                                    interval = c(0, 1),
                                    maximum = TRUE)

  true.mle <- mean(sequence)

  likelihood.error <- true.mle - likelihood.results$maximum
  log.likelihood.error <- true.mle - log.likelihood.results$maximum

  error.behavior <- rbind(error.behavior,
                          data.frame(N = n,
                                     Error = likelihood.error,
                                     Algorithm = 'Likelihood'),
                          data.frame(N = n,
                                     Error = log.likelihood.error,
                                     Algorithm = 'Log Likelihood'))
}

# plotting new error.behavior
ggplot(error.behavior, aes(x = N, y = Error)) +
  geom_line(aes(group = Algorithm, color = Algorithm)) +
  labs(title = 'Long-Term Error Behavior \n of Two Numerical Approaches',
       xlab = 'Sample Size', ylab = 'Deviation from True MLE')
```



More generally, given any data set and any model, you can - at least in principle - solve the maximum likelihood estimation problem using numerical optimization algorithms.