IMT 573 Lab: Maximum Liklihood Estimation

Miloni Desai

November 14th, 2019

Collaborators

Objectives

Today we will explore maximum likelihood estimation. Your task is to run through the examples discussed by J.M. White and *add code comments to all code chunks*. Please add any comments or discussion of results as you go along as well.

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 Server.

1. Download the `lab7_mle.rmd` file from Canvas or save a copy to your local directory on RStudio Server. Open `lab7_mle.rmd` in RStudio and supply your solutions to the assignment by editing `lab7_mle.rmd`.
2. Replace the "Insert 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 chucks, 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 no need four different visualizations of the same pattern.
4. Collaboration on problem sets is fun and useful, and we 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 PDF`, rename the knitted PDF file to `lab7_YourLastName_YourFirstName.pdf`, and submit the PDF file on Canvas.

Setup

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

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

Doing Maximum Likelihood Estimation by Hand in R

By John Myles White on 4.21.2010

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:

```
#Assign a value to the parameter (either p or (1-p) as it is a Bernoulli distribution)
p.parameter <- 0.8
#Create a distribution sample of size 10 and P(p=1) = 0.8
sequence <- rbinom(10, 1, p.parameter)
```

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.

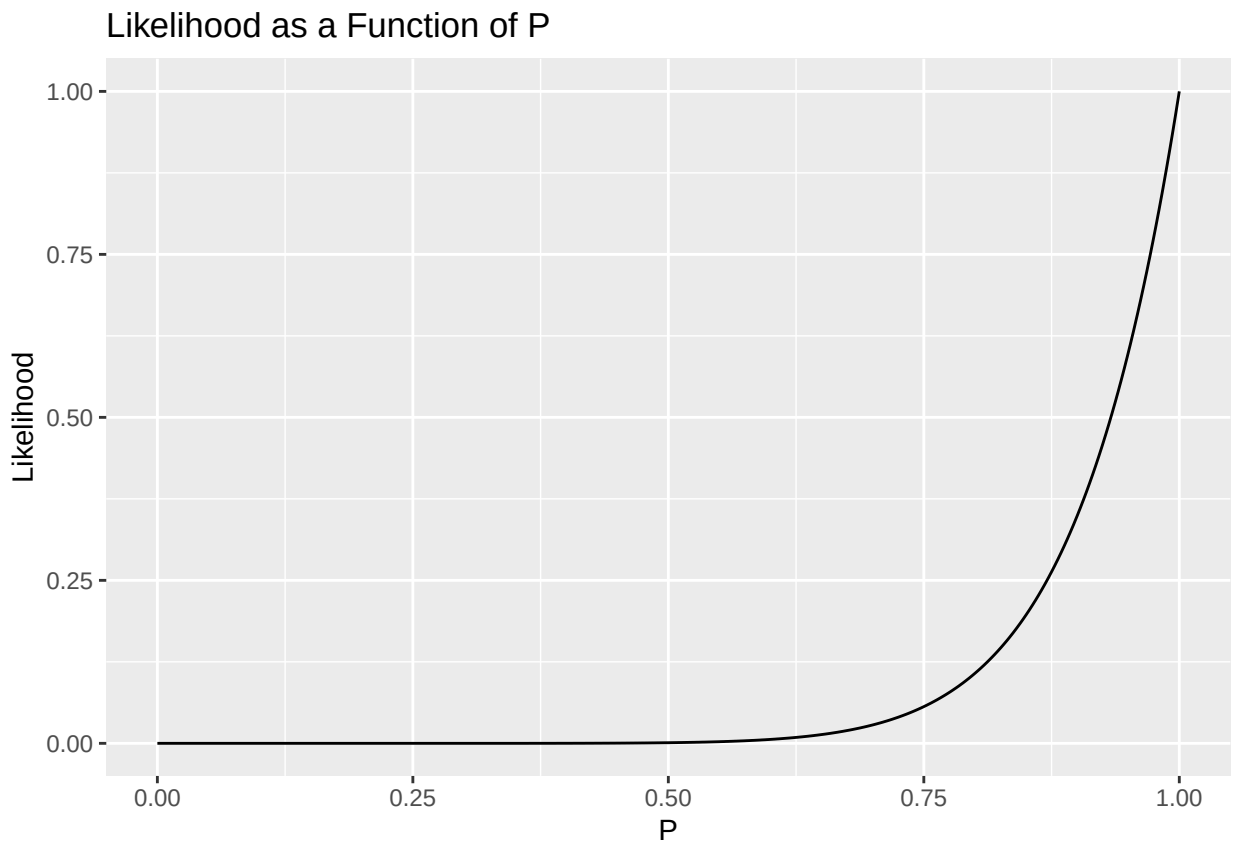
How do we find the parameter numerically? First, we want to define a function that specifies the probability of our entire data set. We assume that each observation in the data is independently and identically distributed, so that the probability of the sequence is the product of the probabilities of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

```
#likelihood function with the given data and parameter(give it any value)
likelihood <- function(sequence, p.parameter)
{
  likelihood <- 1 #likelihood of 1 means that the model perfectly describes the observation.

  #for loop to loop through each observation and multiply likelihoods of each observation
  for (i in 1:length(sequence))
  {
    #if observation is true, probability of likelihood = previous likelihood X parameter
    if (sequence[i] == 1)
    {
      likelihood <- likelihood * p.parameter
    }
    #if the observation is false, probability = (1 - parameter) x previous likelihood
    else
    {
      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:

```
#All possible values that the parameter can take from 0 to 1 with an increment of 0.001
possible.p <- seq(0, 1, by = 0.001)
#Plot all possible values that the parameter can take with their corresponding likelihood values using
qplot(possible.p,
```

```
      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:

```
#find the parameter value corresponds to the highest likelihood ("peak value" in plot)
mle.results <- optimize(function(p) {likelihood(sequence, p)}, #a function that finds all the given lik
                        interval = c(0, 1), #the likelihood value is a number between 0 and 1
                        maximum = TRUE) #find the maximum likelihood

mle.results
```

```
## $maximum
## [1] 0.9999339
##
## $objective
## [1] 0.9993391
```

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()
#loop through all the samples sizes from 10 to 2500
for (n in 10:2500)
{
  sequence <- rbinom(n, 1, p.parameter) #create a bernoulli distribution of size n

  likelihood.results <- optimize(function(p) {likelihood(sequence, p)},
                                 interval = c(0, 1),
                                 maximum = TRUE)
 #find the parameter value that corresponds to the highest likelihood and then use mean of the sample t
  true.mle <- mean(sequence)

  likelihood.error <- true.mle - likelihood.results$maximum
 #calculate error by comparing the sample mean with the parameters with highest likelihood from each ru
  error.behavior <- rbind(error.behavior,
                          data.frame(N = n,
                                     Error = likelihood.error,
                                     Algorithm = 'Likelihood')) #store all error values in a dataframe
}
```

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:

```
#create an array of bernoulli distribution with size 2500 and given parameter
sequence <- rbinom(2500, 1, p.parameter)
#find the parameter value corresponds to the highest likelihood
likelihood.results <- optimize(function(p) {likelihood(sequence, p)},
                               interval = c(0, 1),
                               maximum = TRUE)

likelihood.results
```

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

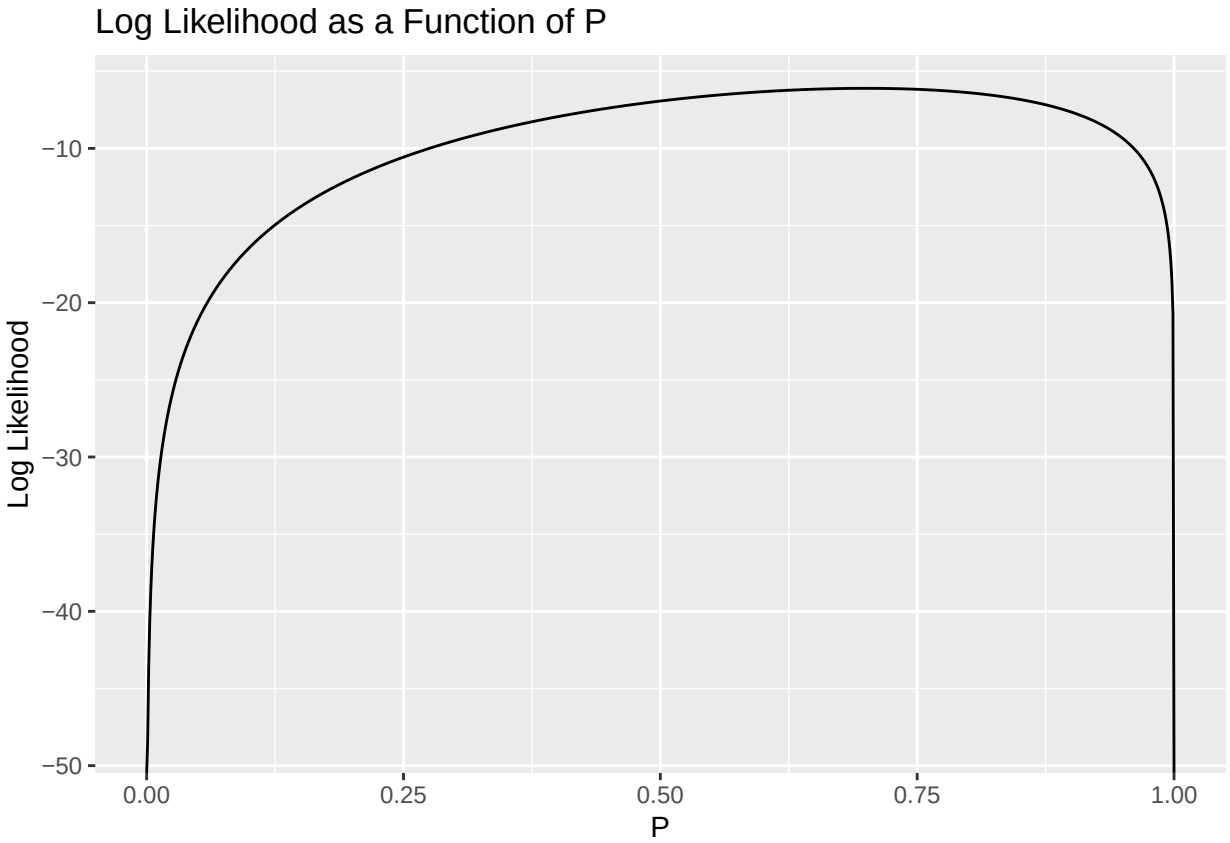
The likelihood of our data is numerically indistinguishable from 0 given the precision of my machine's floating point values. Multiplying thousands of probabilities together is simply not a viable approach without infinite precision. Thankfully, there's a very simple solution: replace all of the probabilities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

```
#compute the logarithmic values of likelihood
log.likelihood <- function(sequence, p)
{
  log.likelihood <- 0 # Since log(1) = 0, log(likelihood) starts from 0 here.

  for (i in 1:length(sequence))
  {
    if (sequence[i] == 1)
```

```
    {
      #log(likelihood x parameter) is equal to log(likelihood) + log(parameter)
      log.likelihood <- log.likelihood + log(p)
    }
    else
    {
      #log(likelihood x parameter) is equal to log(likelihood) + log(1-parameter)
      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:

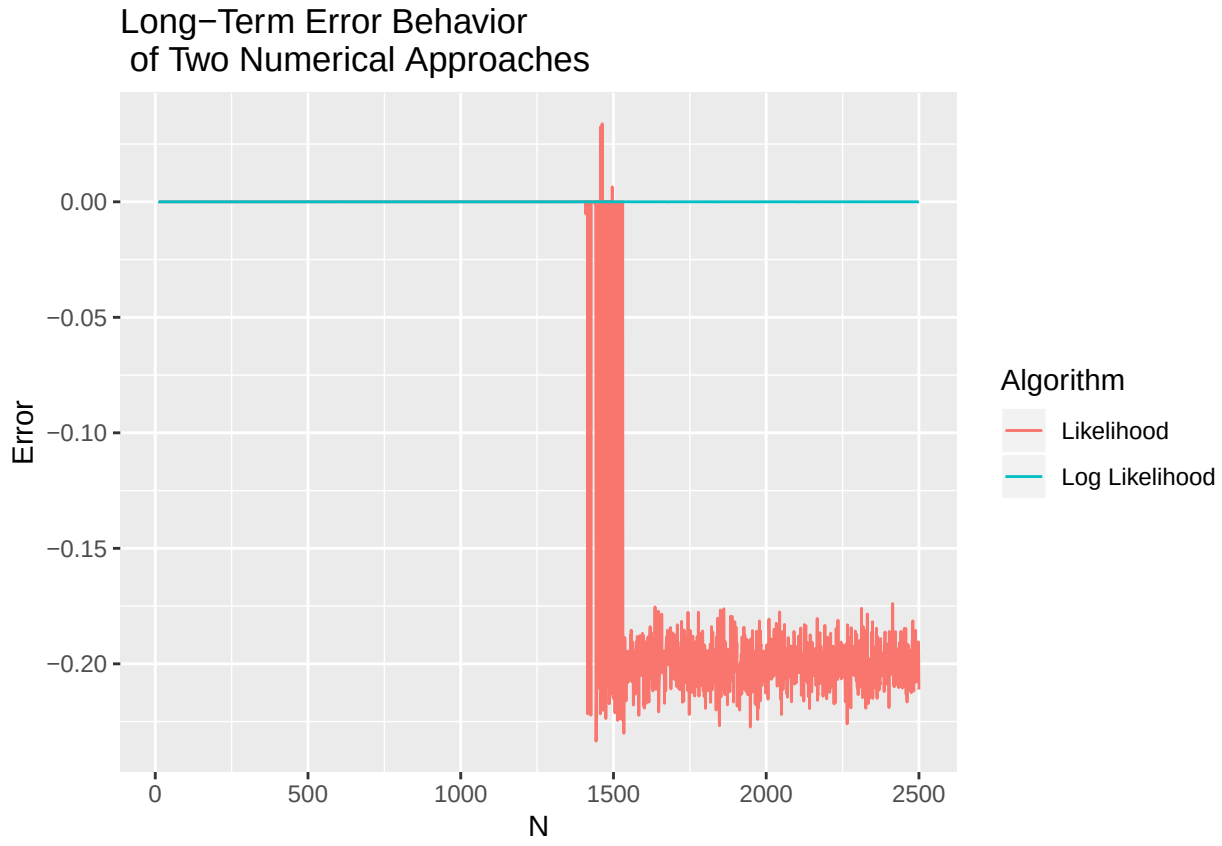
```
#create a random array of 1s and 0s
sequence <- c(0, 1, 1, 1, 1, 1, 1, 0, 1, 0)
#all possible parameter value from 0 to 1 with an increment of 0.001
possible.p <- seq(0, 1, by = 0.001)
#Plot of all parameter values and their corresponding log (likelihood) values
qplot(possible.p,
      sapply(possible.p, 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()

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'))
}

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.