INFX 573 Lab: MLE

Your Name Here

November 3rd, 2016

Collaborators:

Don't forget to list the full names of your collaborators!

Instructions:

Before beginning this assignment, please ensure you have access to R and/or RStudio.

1. Download the `week6b_lab.Rmd` file from Canvas. Open `week6b_lab.Rmd` in RStudio (or your favorite editor) and supply your solutions to the assignment by editing `week6b_lab.Rmd`.
2. Replace the "Insert Your Name Here" text in the `author:` field with your own full name.
3. Be sure to include code chucks, figures and written explanations as necessary. Any collaborators must be listed on the top of your assignment. Any figures should be clearly labeled and appropriately referenced within the text.
4. When you have completed the assignment and have **checked** that your code both runs in the Console and knits correctly when you click `Knit`, rename the R Markdown file to `YourLastName_YourFirstName_lab6b.Rmd`, and knit it into a PDF. Submit the compiled PDF on Canvas.

In this lab, you will need access to the following R packages:

```
# Load some helpful libraries
library(tidyverse)
```

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.

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:

```
p.parameter <- 0.8  #assign a value to the parameter(in bernoulli distribution there is either p or (1-p))
sequence <- rbinom(10, 1, p.parameter)  #create an array of bernoulli distribution with size 10 and P(p =
```

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.

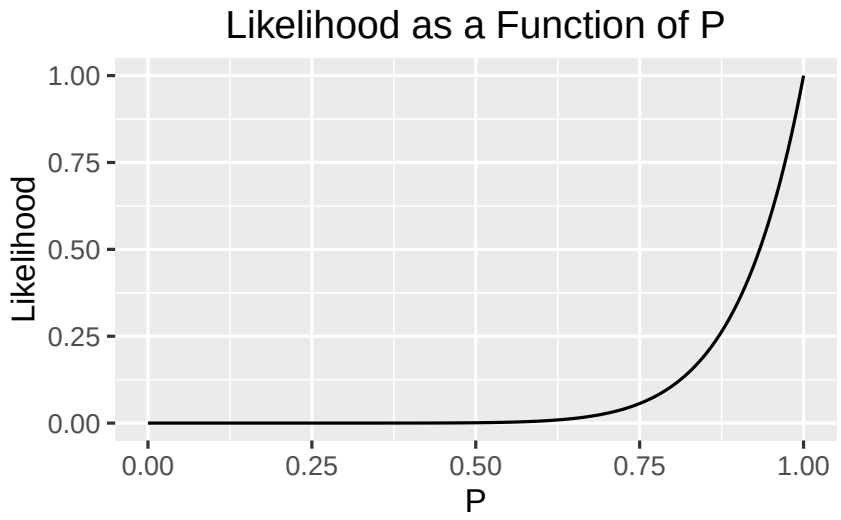
It's worth pointing out that the analytic solution to the maximum likelihood estimation problem is to use the sample mean. We'll therefore use `mean(sequence)` as a measure of the accuracy of our approximation algorithm.

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 given the observed data and parameter(of any value)
likelihood <- function(sequence, p.parameter)
{
  likelihood <- 1 #original likelihood of 1 means model perfectly describes the observation

  for (i in 1:length(sequence)) #loop through each observation to recursively multiply likelihoods of each
  {
    if (sequence[i] == 1) #if the observation is true, probability = parameter x previous likelihood
    {
      likelihood <- likelihood * p.parameter
    }
    else #if the observation is false, probability = (1 - parameter) x previous likelihood
    {
      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)  #all possible parameter value from 0 to 1 with an increment of 0.001
qplot(possible.p, sapply(possible.p, function(p) {
    likelihood(sequence, p)
}), geom = "line", main = "Likelihood as a Function of P",
    xlab = "P", ylab = "Likelihood")
```




```
# plot all possible parameter values versus
# their corresponding likelihood values using
# the likelihood function above.
```

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" in plot)
mle.results <- optimize(function(p) {likelihood(sequence, p)}, #a function with all possible likelihood gi
                        interval = c(0, 1), #likelihood 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()

for (n in 10:2500) #loop through all possible sample sizes from 10 to 2500
{
  sequence <- rbinom(n, 1, p.parameter) #create an array of bernoulli distribution with size n and a param

  likelihood.results <- optimize(function(p) {likelihood(sequence, p)},
                                 interval = c(0, 1),
                                 maximum = TRUE)
   #find the parameter value corresponds to the highest likelihood

  true.mle <- mean(sequence) #calculate sample mean to measure accuracy of the estimation

  likelihood.error <- true.mle - likelihood.results$maximum
  #calculate error by comparing the sample mean with the parameters with highest likelihood from each run

  error.behavior <- rbind(error.behavior,
                          data.frame(N = n,
                                     Error = likelihood.error,
                                     Algorithm = 'Likelihood'))
  #store all errors in a data.frame
}
#plot(error.behavior$N, error.behavior$Error) to look at the sample size versus error
```

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)  #create an array of bernoulli distribution with size 2500 and gi

likelihood.results <- optimize(function(p) {
    likelihood(sequence, p)
}, interval = c(0, 1), maximum = TRUE)
# find the parameter value corresponds to the
# highest likelihood
likelihood.results
```

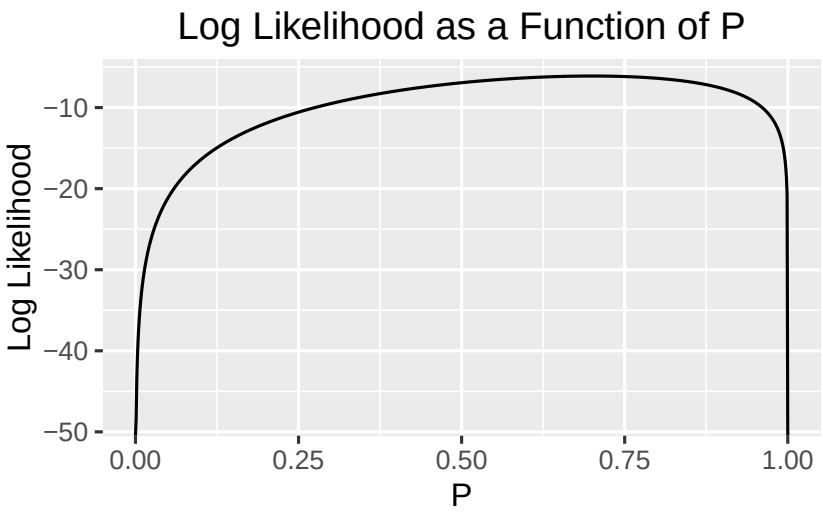
```
## $maximum
## [1] 0.9999339
##
## $objective
## [1] 0
```

The likelihood of our data is numerically indistinguishable from o 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:

```
# a function calculating logarithm of all
# likelihoods
log.likelihood <- function(sequence, p) {
    log.likelihood <- 0  #log(1) = 0, thus log(likelihood) starts from 0

    for (i in 1:length(sequence)) {
        if (sequence[i] == 1) {
            log.likelihood <- log.likelihood +
                log(p)  #log(likelihood x parameter) is equal to log(likelihood) + log(parameter)
        } else {
            log.likelihood <- log.likelihood +
                log(1 - p)  #same as above only the parameter/probability here is equal to (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)  #create a random array with 1 and 0s
possible.p <- seq(0, 1, by = 0.001)  #all possible parameter value from 0 to 1 with an increment of 0.001
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")
```

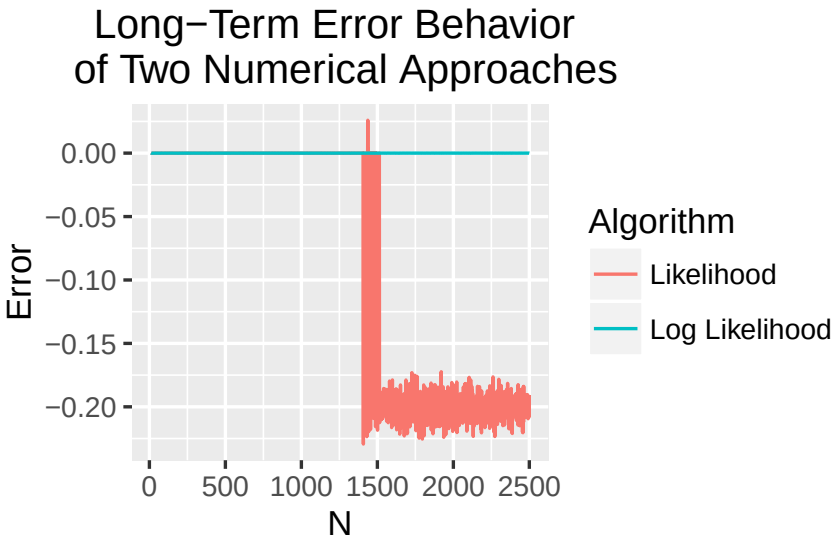



```
# plot all possible parameter values versus
# their corresponding log likelihood values
# using the log.likelihood function.
```

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.