Maximum Likelihood Estimation (MLE)
Introduction to Statistics Using R (Psychology 9041B)

Paul Gribble

Winter, 2019

1 Coin Flipping

Assume somebody gives you a coin, and they claim that it is fair, i.e. the probabilty of a heads is the same as the probability of tails, i.e. both are 50%. Your task is to observe the behaviour of the coin and then determine what the probability of heads θ really is. Is it 50% or is it something else? (which would reflect a biased coin)

1.1 The Data

You flip the coin 20 times and you observe 12 heads and 8 tails.

1.2 The Model

We must now decide on a model. A convenient model for this case is the Binomial Distribution, which models a Bernoulli process (a sequence of binary random responses). The Binomial model is:

$$y_i \sim f(\theta, y_i) = \frac{N!}{y_i!(N-y_i)!}\theta^{y_i}(1-\theta)^{N-y_i} \tag{1}$$

The model has a single parameter, θ, and that is exactly what we want to estimate. In maximum likelihood estimation, we determine what value of the parameter θ would make the data that we observed most likely?

1.3 The Likelihood Function

For the Binomial model, there actually exists a closed-form analytic solution to the maximum likelihood estimate of θ, namely the number of heads divided by the number of coin flips, or in this case, $12/20 = 0.60$. Let's assume however that there isn't a closed form solution, or that we don't know of one. We will use optimization to find the value of θ that maximizes the likelihood of the data that we observed. To do this we need to compute the likelihood of our data, given a candidate value (i.e. a guess) for θ.

As we saw in the readings, after some insight into Bayes Theorem, we can compute the likelihood as:

$$\mathcal{L}(\theta|y) = p(y|\theta) \tag{2}$$

And we know that for the Binomial distribution,

$$p(heads|\theta) = \theta \tag{3}$$

$$p(tails|\theta) = (1-\theta) \tag{4}$$

The other trick we need to invoke is that the likelihood of the entire dataset is the product of the likelihoods of each individual data point in the dataset.

And finally, because multiplying a bunch of probabilities makes computers feel icky, we take the logarithm to convert the products into sums, and so instead of multiplying the likelihood of each observation in the data set, we add the log-likelihood of each data point.

And finally finally, because optimizers like to minimize things and not maximize things, we multiply the sum of log-likelihoods by (-1) to get the negative log likelihood of the dataset:

$$NLL = -\sum_{i=1}^{N} \ln p(y_i|\theta) \tag{5}$$

So now we can write a function in R to compute the negative log likelihood of a data vector y given a guess at the parameter θ:

```
> NLL <- function(theta,y) {
+   NLL <- 0
+   N <- length(y)
+   for (i in 1:N) {
+     if (y[i]==1) {p <- theta}   # heads
+     if (y[i]==0) {p <- 1-theta} # tails
+     NLL <- NLL + log(p)
+   }
+   -NLL
+ }
```

Now we can make use of a built-in optimizer in R to find the parameter value θ that minimizes the negative log likelihood. Note that one has to give the optimizer a starting guess at the parameter value, here I give it a starting guess of 0.5.

```
> flips <- c(1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0) # 12 heads, 8 tails
> out = optim(par=0.5, fn=NLL, method="Brent", lower=0.00001, upper=0.99999, y=flips)
> (theta <- out$par)

[1] 0.6
```

And so our best estimate of θ is 0.6, which matches the known closed-form analytic solution (12/20). Note for one-dimensional optimization problems the optim() function in R suggests that the "Brent" method is used, which is why I pass that argument in. I also chose to give the optim() function lower and upper bounds on the parameter estimate, to keep it constrained to sensible values.

2 Normal Distribution

Let's do another example, this time with a model that involves estimating two parameters. Imagine you are handed a dataset comprised of a list of 20 numbers x:

```
> x <- c(85,84,75,93,88,82,85,94,86,76,81,98,95,82,76,91,81,82,72,94)
```

Assume that you want to model these data as coming from a gaussian random process. That is, there is some unknown mean μ and variance σ^2. Thus our model has two unknown parameters.

Now set aside for the moment that we all know full well that there is a closed-form analytic solution to this problem, namely:

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{6}$$

$$\hat{\sigma} = \frac{1}{N} = \sum_{i=1}^{N}(x_i - \hat{\mu})^2 \tag{7}$$

The point of MLE is to show that as long as you have a forward model of the probability of the data given a guess at the parameter(s), you can use an optimizer to find the parameter value(s) that maximize the likelihood (minimize the negative log-likelihood) of the data given the parameter(s).

We can compute the probability of the data x given parameters μ and σ^2 using the equation for the normal distribution pdf (probability density function):

$$p(y|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{8}$$

After taking the log and some algebraic manipulation (not shown) we get:

$$\mathcal{L}(y|\mu,\sigma^2) = -\frac{N}{2}\ln(2\pi) - \frac{N}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 \tag{9}$$

Let's put this into an R function to compute the negative log-likelihood of the data x given parameter guesses μ and σ^2:

```
> NLL <- function(theta,data) {
+   mu = theta[1]
+   sigma = theta[2]
+   n = length(data)
+   NLL = -(n/2)*log(2*pi) - (n/2)*log(sigma**2)
+   tmp = 0
+   for (i in 1:n) {
+     tmp = tmp + (data[i]-mu)**2
+   }
+   NLL = NLL + -(1/(2*(sigma**2)))*tmp
+   -NLL
+ }
```

Now we can use an R optimizer to find the parameter values that minimize this function:

```
> out = optim(par=c(100,10), fn=NLL, data=x)
> (out$par)

[1] 84.998857  7.183545
```

Note that I still have to give an initial guess at the parameter values ... here I simply guessed $\mu = 100$ and $\sigma = 10$. The optimizer tells me that the best parameter estimates are in fact $\mu = 84.998857$ and $\sigma = 7.183545$.

So again, the point here is that no matter how complicated our model of the data is, as long as we can write down the likelihood function (which amounts to a probability model of how likely different values of data are, given model parameters), then we can use an optimizer to find the model parameters that maximize the likelihood of the data.

3 A more complex example

Let's say a colleague hands you a coin and you are told that it is fair, in other words the probability of heads and tails is the same, $p(heads) = p(tails) = 0.5$. You are asked to flip the coin, and each time give it back to your colleague who will put it in his pocket, and then take it out again and hand it back to you. You do this 100 times and you observe the following data:

```
> flips = c(0,1,1,0,1,1,1,0,0,1,0,0,1,1,1,0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,0,1,
+ 0,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,1,1,1,0,
+ 1,1,0,1,1,1,1,1,0,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,0,1,1,1,0)
```

You are then told that at some point, your colleague switched the coin from the fair coin to an unfair coin with $p(heads) = 0.9$. Your task is to try to estimate **when** the switch occurred.

Our first question will be, how are we going to model this process? Clearly like our first example, a Binomial model is appropriate... but we also have to model the switching of a coin (when?).

A natural Binomial model might look like this. For trials 1 to s, $p(heads) = 0.5$ and for trials $s + 1$ to 100, $p(heads) = 0.9$. Now we have a model of the process that involves the unknown parameter s, the switch trial. Let's write a function in R to compute the negative log-likelihood given the observed data and a guess at the s parameter:

```
> NLL <- function(s, data) {
+   n = length(data)
+   NLL = 0
+   for (i in 1:n) {
```

```
+     if (i<=s) { # still the unbiased coin
+       p = 0.5
+     }
+     else {      # coin has switched to biased coin
+       p = 0.9
+     }
+     if (data[i]==1) {NLL = NLL + log(p)}
+     if (data[i]==0) {NLL = NLL + log(1-p)}
+   }
+   -NLL
+ }
```

Now we can optimize:

```
> optim(par=50, fn=NLL, data=flips, method="Brent", lower=1, upper=99)

$par
[1] 40.27792

$value
[1] 51.62531

$counts
function gradient
      NA       NA

$convergence
[1] 0

$message
NULL
```

We can see that the best estimate is that $s = 40.28$, let's call it trial 40, is when the coin switch occurred. In fact this is a very good estimate, since I generated the data using the following, in which the flip happened at trial 41:

```
> flips = c(rbinom(n=40,size=1,prob=0.5), rbinom(n=60,size=1,prob=0.9))
```

You can imagine extending this example for the case in which you have to estimate both the time of the coin switch, and the new $p(heads)$.

4 Estimating Psychometric Functions

Another example to look at in which MLE is used to estimate a model of data, is the case of estimating a psychometric function.

A *psychometric function* relates a parameter of a physical stimulus to an experimental participant's subjective response. In a typical experiment, some parameter of a stimulus is varied across some range on each of several trials, and an experimental participant is asked on each to report their subjective response. For example in an experiment on visual perception, the experimenter might vary the relative brightness of two sine-wave gratings, and ask the participant which is brighter, the one on the left or the one on the right. In an experiment on auditory perception, the experimenter might vary the first formant frequency of a synthetic vowel sound along a continuous range, and ask the participant, do you hear an /i/ or an /a/? These are examples of a two-alternative-forced-choice (2-AFC) task. What we're interested in is how the probability of responding with one of the two choices varies with the stimulus.

Here we will consider the situation where the physical stimulus being varied is the direction of movement of a participant's (passive) arm, which is moved by a robotic device to the left or right of the participant's midline (to varying extents), and the participant's response is binary (*left* or *right*). This paradigm has been described in several of our recent papers [1, 2, 3, 4].

4.1 The Logistic Function

We assume the shape of the psychometric function is *logistic* (Figure 1). The function relates lateral hand position (relative to a participant's midline) to the probability of the participant responding that their hand is to the *right* of midline. The vertical dashed line indicates the hand position at which the probability of responding *right* is 50%, known as the perceptual *threshold* (in this case at the actual midline, 0 mm).

Equation 10 gives the logistic function:

$$p = 1/\left(1 + e^{-y}\right) \tag{10}$$

where $p = \Pr(\text{right}|x)$.

Here the value of y is a linear function of the lateral hand position x, given by Equation 11:

$$y = \beta_0 + \beta_1 x \tag{11}$$

The shape (steepness) and position (in terms of the threshold) of the logistic function can be directly related to the parameters β_0 and β_1. Another quantity that we sometimes compute is the distance between the 75th and 25th percentile (in other words the middle 50 %), as a measure of *acuity.*

$$\begin{aligned} \text{threshold} &= -\beta_0/\beta_1 && (12)\\ \text{slope} &= \beta_1/4 && (13)\\ \text{acuity} &= \text{ilog}(\beta, .75) - \text{ilog}(\beta, .25) && (14) \end{aligned}$$

where $\text{ilog}(\beta, p)$ is the inverse logistic function:

$$x = \left[\log\left(-p/(p-1)\right) - \beta_0\right]/\beta_1 \tag{15}$$

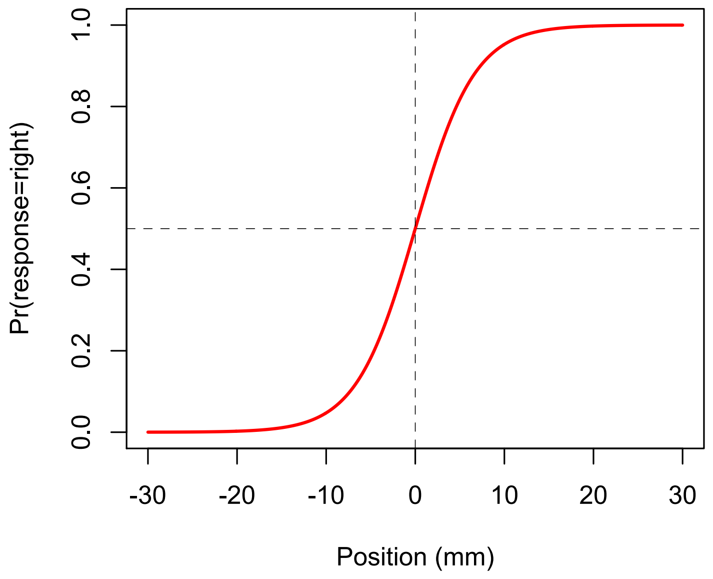


Figure 1: Logistic Function

and as before, $p = \Pr(\text{response} = \text{right}|x)$.

4.2 Maximum Likelihood Estimation

Assume we have empirical data consisting of binary responses R (r_1 through r_n) (each *left* or *right*) for some number of hand positions X (x_1 through x_n). To estimate a psychophysical function we need to find the values of β_0 and β_1 (Equation 11) that best fit the data.

How do we define a metric for determining "best fit"? Unfortunately we cannot use a criterion such as *least-squares* combined with standard linear regression approaches. Using least-squares and a standard linear model is inappropriate because it violates a number of the assumptions of linear regression using least-squares. First, the error term would be *heteroskedastic* which means that the variance of the dependent variable is not constant across the range of the independent variable. Second, the error term is not normally distributed, because it is binary and thus takes on only two possible values. Third, there is no straightforward way of guaranteeing that predicted probabilities will not be greater than 1 or less than 0, which of course is not meaningful.

Instead, we will define a new cost function. We will find the values of β_0 and $\beta 1$ that maximize the *likelihood of the data*.

Unlike estimation problems like linear regression, in which there is a closed-form expression for the parameters that minimize a cost function (e.g. least-squares), in this case using maximum likelihood estimation (MLE) there is no closed-form solution, and so we must use numerical optimization to find the parameter values β_0 and $\beta 1$ that result in the best fit, i.e. that maximize the likelihood of the data (the set of positions and binary responses) given the model (β_0 and $\beta 1$).

The probability of an individual binary response r, given a hand position x is given by:

$$\Pr(r = \text{right}|x) = p \tag{16}$$

$$\Pr(r = \text{left}|x) = 1 - p \tag{17}$$

where p depends on β_0, β_1 and position x according to Equations 10 and 11.

The likelihood of the data (all n responses taken together) is proportional to the product of the probabilities of each response $r_i|x_i$:

$$\text{L}(\text{data}|\text{model}) = \prod_{i=1}^{n} \Pr(\text{resp}_i|x_i) \tag{18}$$

The product of many probabilities (which range from 0 to 1) becomes very small very quickly, and for large datasets, the likelihood is a very small number — so small that computers will have difficulties representing it accurately. To fix this, we take the logarithm, which has the dual effect of changing small fractions into larger numbers, and also changing products into sums. So instead of degenerating into a smaller and smaller fraction, this quantity accumulates into a manageable-sized number. This is thus the log-likelihood of the data:

$$\log\left[\text{L}(\text{data}|\text{model})\right] = \sum_{i=1}^{n} \log\left[\text{L}(\text{resp}_i|x_i)\right] \tag{19}$$

Finally, since numerical optimizers are set up to find the minimum, not maximum, of a cost function, we multiply the log-likelihood by (-1). This gives us a cost function in terms of the *negative log-likelihood* of the data, given the model.

$$-\log\left[\text{L}\right] = -\sum_{i=1}^{n} \log\left[\Pr(\text{resp}_i|x_i)\right] \tag{20}$$

Equation 20 is our cost function. Now with Equations 10, 11, 16, 17 and 20, given our dataset (the set of hand positions x_i and binary responses r_i) and a candidate “model” (β_0, β_1), we can compute the cost, and use numerical optimization techniques to find the model (β_0, β_1) that minimizes the cost (the negative log-likelihood of the data).

Table 1 shows an example of the kind of data we are interested in fitting:

x (mm)	response
1.1150	left
10.2260	right
-9.2050	left
5.2180	right
...	...

Table 1: Example data

Typically we represent the response in a numerical format, where 0=left and 1=right. Also typically we have many more data points than shown, e.g. in a typical experiment we may test 7 positions each sampled 8 times for a total of 56 observations.

Assume we have an array of hand positions `X` in metres and an array of binary responses `R` (where left=0 and right=1).

First let's write our logistic function:

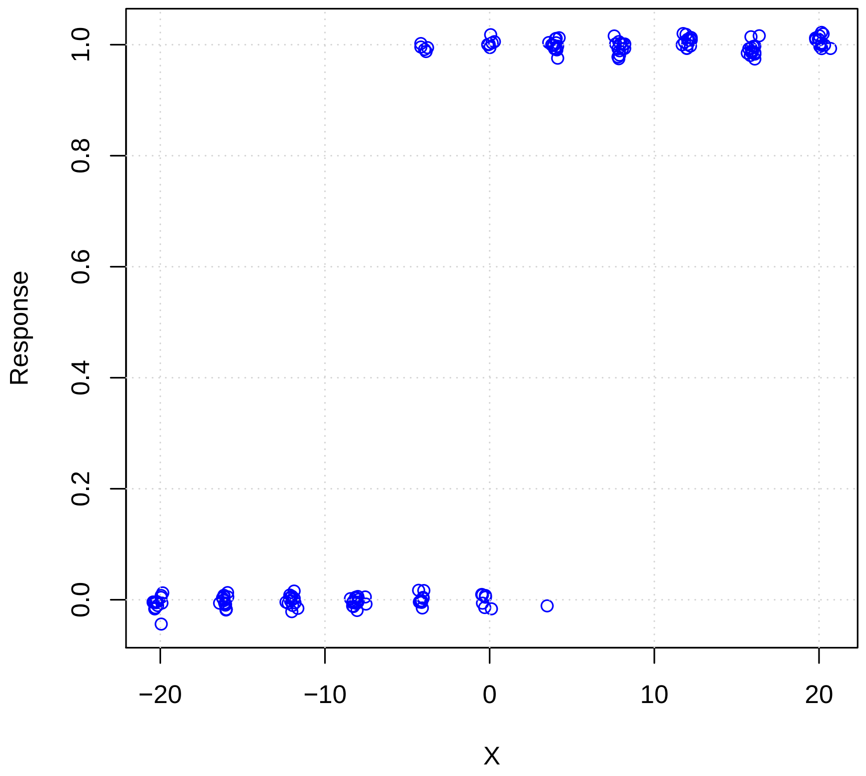
```
> logistic <- function(y) {
+   p = 1/(1+exp(-y))
+   p
+ }
```

We can write our cost function, called `nll` as follows:

```
> NLL <- function(B,X,R) {
+   y = B[1] + B[2]*X
+   p = logistic(y)
+   NLL = -sum(log(p[R==1])) - sum(log(1-p[R==0]))
+   NLL
+ }
```

Let's load in some example data and plot it (note I've added some random scatter along the y-axis to help visualize each individual response):

```
> pdata = read.table("https://www.gribblelab.org/stats2019/data/psychometric_data.txt",
+ sep=" ", header=FALSE)
> X = pdata$V2
> R = pdata$V3
> plot(X, R+rnorm(length(R),0,.01), type="p", col="blue",
+ xlab="X", ylab="Response")
> grid()
```



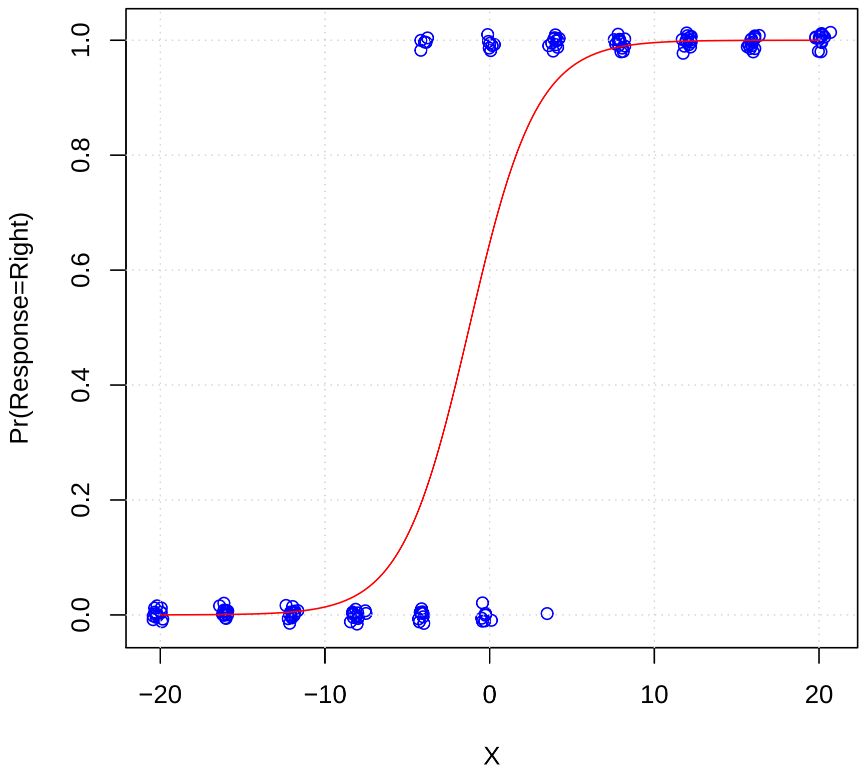
You can see that when X is to the far left (negative values), the response is always 0, which corresponds to "Left". On the far right, the response is always 1, or "Right". In between, the responses gradually change. In the middle somewhere, responses are about half "Left" and half "Right".

Now let's use MLE to fit a psychometric function to the data. Again, note that one has to give the optimizer an initial guess. Here my initial guess is $(-.1, .1)$.

```
> out = optim(par=c(-.1,.1), NLL, X=X, R=R)
> (Bfit = out$par)

[1] 0.6043109 0.4870757

> Xp = seq(-20,20,.1)
> p = logistic(Bfit[1] + Bfit[2]*Xp)
> plot(X, R+rnorm(length(R),0,.01), type="p", col="blue",
+ xlab="X", ylab="Pr(Response=Right)")
> grid()
> lines(Xp, p, type="l", col="red")
```



You can see that the fitted curve does a very good job of characterizing how the participant's responses change as the stimulus changes value from left (negative X) to right (positive X).

References

[1] Elizabeth T Wilson, Jeremy Wong, and Paul L Gribble. Mapping proprioception across a 2-d horizontal workspace. *PLoS One*, 5(7):e11851, 2010.

[2] David J Ostry, Mohammad Darainy, Andrew AG Mattar, Jeremy Wong, and Paul L Gribble. Somatosensory plasticity and motor learning. *The Journal of Neuroscience*, 30(15):5384–5393, 2010.

[3] Jeremy D Wong, Elizabeth T Wilson, and Paul L Gribble. Spatially selective enhancement of proprioceptive acuity following motor learning. *Journal of Neurophysiology*, 105(5):2512–2521, 2011.

[4] Jeremy D Wong, Dinant A Kistemaker, Alvin Chin, and Paul L Gribble. Can proprioceptive training improve motor learning? *Journal of Neurophysiology*, 108(12):3313–3321, 2012.