Maximum Likelihood Estimation

2020-01-17

Preparation

In this set of notes, you will learn about the method of maximum likelihood to estimate model parameters. In this set of notes, we will not be using a specific dataset.

```
# Load libraries
library(broom)
library(educate) #Need version 0.1.0.1
library(patchwork)
library(tidyverse)
```

Joint Probability Density

In the previous set of notes, we discussed the probability density of an observation X_i. Now we will extend this idea to the probability density of a set of observations, say x_1, x_2, …AND x_k. The probability density of a set of observations is referred to as the *joint probability density*, or simply *joint density*.

If we can make an assumption about INDEPENDENCE, then the joint probability density would be the product of the individual densities:

$$p(x_1, x_2, x_3, \ldots, x_k) = p(x_1) \times p(x_2) \times p(x_3) \times \ldots \times p(x_k)$$

Say we had three independent observations from our $\sim \mathcal{N}(50, 10)$ distribution, namely $x = \{60, 65, 67\}$. Then the joint density would be,

```
dnorm(x = 60, mean = 50, sd = 10) * dnorm(x = 65, mean = 50, sd = 10) * dnorm(x = 67, mean = 50, sd = 10)
```

```
[1] 0.000002947448
```

We could also shortcut this computation,

```
prod(dnorm(x = c(60, 65, 67), mean = 50, sd = 10))
```

```
[1] 0.000002947448
```

This value is the joint probability density. The joint probability density indicates the probability of observing the data ($x = \{60, 65, 67\}$) GIVEN (1) they are drawn from a normal distribution and (2) the normal distribution has a mean of 50 and a standard deviation of 10. In other words:

> The joint probability density is the probability of the data given the distribution and parameters.

Symbolically,

$$\text{Joint Density} = P(\text{Data} \mid \text{Distribution and Parameters})$$

Likelihood

Likelihood is the probability of a particular set of parameters GIVEN (1) the data, and (2) the data are from a particular distribution (e.g., normal). Symbolically,

$$\text{Likelihood} = P(\text{Parameters} \mid \text{Distribution and Data})$$

Likelihood takes the data as given and computes the probability of a set of parameters. Symbolically we denote likelihood with a scripted letter "L" ($\mathcal{L}$). For example, we might ask the question, given the observed data $x = \{30, 20, 24, 27\}$ come from a normal distribution, what is the likelihood (probability) that the mean is 20 and the standard deviation is 4? We might denote this as,

$$\mathcal{L}(\mu = 20, \sigma = 4 \mid x)$$

Note that although we need to specify the distribution (e.g., normal), this is typically not included in the symbolic notation; instead it is typically included in the assumptions.

The likelihood allows us to answer probability questions about a set of parameters. For example, what is the likelihood (probability) that the data ($x = \{30, 20, 24, 27\}$) were generated from a normal distribution with a mean of 20 and standard deviation of 4? To compute the likelihood we compute the joint probability density of the data under that particular set of parameters.

```r
prod(dnorm(x = c(30, 20, 24, 27), mean = 20, sd = 4))
```

```
[1] 0.0000005702554
```

What is the likelihood (probability) that the data ($x = \{30, 20, 24, 27\}$) were generated from a normal distribution with a mean of 25 and standard deviation of 4?

```r
prod(dnorm(x = c(30, 20, 24, 27), mean = 25, sd = 4))
```

```
[1] 0.00001774012
```

It is important to note that although we use the joint probability under a set of parameters to compute the likelihood of those parameters, theoretically joint density and likelihood are very different. Likelihood refers to the probability of the parameters and joint probability density refers to the probability of the data.

Maximum Likelihood

Which set of parameters,$\mathcal{N}(20, 4)$ or $\mathcal{N}(25, 4)$, was *more likely* to generate the given data? Since the second set of parameters produced a higher likelihood, the data was more likely to have been generated from the $\mathcal{N}(25, 4)$ distribution that the $\mathcal{N}(20, 4)$ distribution.

So now we come to the crux of Maximum Likelihood Estimation (MLE). The goal of MLE is to find a set of parameters that MAXIMIZES the likelihood given the data and a distribution. For example, given the observed data $x = \{30, 20, 24, 27\}$ were generated from a normal distribution, what are the values for the parameters of this distribution (mean and standard deviation) that produce the HIGHEST (or maximum) value of the likelihood?

There are different methods for determining the parameter values that produce the maximum value of the likelihood. One of these methods is a computational method called *grid search*.

Method 1: Grid Search

One method for finding the parameters (in our example, the mean and standard deviation) that produce the maximum likelihood, is to substitute several parameter values in the `dnorm()` function, compute the likelihood for each set of parameters, and determine which set produces the highest (maximum) likelihood.

In computer science, this method for finding the MLE is referred to as a *grid search*. Below is some syntax to carry out a grid search. The syntax creates several sets of parameter values (called the search space), computes the likelihood for each combination of parameter values, and then arranges the likelihoods in descending order.

```
crossing(
  mu = seq(from = 10, to = 30, by = 0.1),
  sigma = seq(from = 0, to = 10, by = 0.1)
  ) %>%
  rowwise() %>%
  mutate(
    L = prod(dnorm(c(30, 20, 24, 27), mean = mu, sd = sigma))
    ) %>%
  arrange(desc(L))
```

```
Source: local data frame [20,301 x 3]
Groups: <by row>

# A tibble: 20,301 x 3
      mu sigma         L
   <dbl> <dbl>     <dbl>
 1  25.2   3.7 0.0000183
 2  25.3   3.7 0.0000183
 3  25.3   3.8 0.0000182
 4  25.2   3.8 0.0000182
 5  25.4   3.7 0.0000182
 6  25.1   3.7 0.0000182
 7  25.2   3.6 0.0000182
 8  25.3   3.6 0.0000182
 9  25.1   3.8 0.0000182
10  25.4   3.8 0.0000182
# ... with 20,291 more rows
```

The parameters that maximize the likelihood (in our search space) are a mean of 25.2 and a standard deviation of 3.7.

Log-Likelihood

The likelihood values are quite small since we are multiplying several probabilities together. We could take the natural logarithm of the likelihood to alleviate this issue. So in our example, $\mathcal{L} = .00001829129$ and the log-likelihood would be

```
log(.00001829129)
```

```
[1] -10.90909
```

We typically denote log-likelihood using a scripted lower-case "l" (l). Going back to how we compute the likelihood, we assumed a set of parameters and then found the joint probability density, which assuming normality and independence is the product of the individual densities.

$$\mathcal{L}(\text{parameters}|\text{data}) = p(x_1) \times p(x_2) \times ... \times p(x_n)$$

If we compute the log of the likelihood instead:

$$l(\text{parameters}|\text{data}) = \ln\Big(\mathcal{L}(\text{parameters}|\text{data})\Big) = \ln\Big(p(x_1) \times p(x_2) \times ... \times p(x_n)\Big)$$

Using the rules of logarithms, the right-hand side of the equation can be manipulated to:

$$= \ln\Big(p(x_1)\Big) + \ln\Big(p(x_2)\Big) + ... + \ln\Big(p(x_n)\Big)$$

The log-likelihood is the *sum of the log-transformed densities*. This means we could re-write our grid search syntax to compute the log-likelihood. Since finding the log of the densities is so useful, there is even an argument in `dnorm()` of `log=TRUE` that does this for us. Our revised grid search syntax is:

```
crossing(
  mu = seq(from = 10, to =30, by = 0.1),
  sigma = seq(from = 0, to = 10, by = 0.1)
  ) %>%
  rowwise() %>%
  mutate(
    log_L = sum(dnorm(c(30, 20, 24, 27), mean = mu, sd = sigma, log = TRUE))
    ) %>%
  arrange(desc(log_L))
```

```
Source: local data frame [20,301 x 3]
Groups: <by row>

# A tibble: 20,301 x 3
      mu sigma log_L
   <dbl> <dbl> <dbl>
 1  25.2   3.7 -10.9
 2  25.3   3.7 -10.9
 3  25.3   3.8 -10.9
 4  25.2   3.8 -10.9
 5  25.1   3.7 -10.9
 6  25.4   3.7 -10.9
 7  25.3   3.6 -10.9
 8  25.2   3.6 -10.9
 9  25.1   3.8 -10.9
10  25.4   3.8 -10.9
# ... with 20,291 more rows
```

Maximizing the log-likelihood gives the same parameter values as maximizing the likelihood. Remember that the log computation keeps the same ordination of values as the original data, so maximizing the log-likelihood is the same as maximizing the likelihood.

Maximum Likelihood Estimation for Regression

In model fitting, the components we care about are the residuals. Those are the things we put distributional assumptions on (e.g., normality, homogeneity of variance, independence). Our goal in regression is to estimate a set of parameters (β_0, β_1) that maximize the likelihood for a given set of residuals that come from a normal distribution.

To understand this, let's use a toy example of $n = 10$ observations.

```
   x  y
1  4 53
2  0 56
3  3 37
4  4 55
5  7 50
6  0 36
7  0 22
8  3 75
9  0 37
10 2 42
```

To begin, we can enter these observations into two vectors, x and y.

```r
# Enter data into vectors
x = c(4, 0, 3, 4, 7, 0, 0, 3, 0, 2)
y = c(53, 56, 37, 55, 50, 36, 22, 75, 37, 42)
```

Next, we will write a function to compute the log-likelihood (or likelihood) of the residuals given particular `b0` and `b1` estimates that will be inputted to the function.

One issue is that in using the `dnorm()` function we need to specify the mean and standard deviation. The regression assumptions help with this task. The conditional mean residual value is 0. So we will set the mean value to 0. The assumption about the standard deviation is that the conditional distributions all have the same SD, but it doesn't specify what that is. However, the SD of the errors seems like a reasonable value, so let's use that.

Below, we will write a function called `log_likelihood()` that takes two arguments as input, `b0=` and `b1=`, and outputs the log-likelihood.

```r
log_likelihood = function(b0, b1){
  # Use the following x and y values
  x = c(4, 0, 3, 4, 7, 0, 0, 3, 0, 2)
  y = c(53, 56, 37, 55, 50, 36, 22, 75, 37, 42)

  # Compute the yhat and residuals based on the two input values
  yhats = b0 + b1*x
  errors = y - yhats

  # Compute the sd of the residuals
  sigma = sd(errors)

  # Compute the log-likelihood
  log_lik = sum(dnorm(errors, mean = 0, sd = sigma, log = TRUE))

  # Output the log-likelihood
  return(log_lik)
}
```

Now we read in our function by highlighting the whole thing and running it. Once it has been read in, we can use it just like any other function. For example to find the log-likelihood for the parameters $\beta_0 = 10$ and $\beta_1 = 3$ we use:

```
log_likelihood(b0 = 10, b1 = 3)
```

```
[1] -64.29224
```

We can also use our function in a grid search.

```
crossing(
  b0 = seq(from = 30, to = 50, by = 0.1),
  b1 = seq(from = -5, to = 5, by = 0.1)
) %>%
  rowwise() %>%
  mutate(
    log_L = log_likelihood(b0 = b0, b1 = b1)
    ) %>%
  arrange(desc(log_L))
```

```
Source: local data frame [20,301 x 3]
Groups: <by row>

# A tibble: 20,301 x 3
      b0    b1 log_L
   <dbl> <dbl> <dbl>
 1  40.1   2.7 -39.5
 2  40     2.7 -39.5
 3  40.2   2.7 -39.5
 4  39.9   2.8 -39.5
 5  39.8   2.8 -39.5
 6  40     2.8 -39.5
 7  39.9   2.7 -39.5
 8  39.7   2.8 -39.5
 9  40.3   2.7 -39.5
10  40.1   2.8 -39.5
# ... with 20,291 more rows
```

Here the parameter values that maximize the likelihood are $\beta_0 = 40.1$ and $\beta_1 = 2.7$. We can also compute what the standard deviation for the residual distributions was using the estimated parameter values. Remember, this value is an estimate of the RMSE.

```
errors = y - 40.1 - 2.7*x
sd(errors)
```

```
[1] 13.18665
```

In practice, there are a couple subtle differences, namely that the estimate for the SD value we use in `dnorm()` is slightly different. This generally does not have an effect on the coefficient estimates, but does impact the estimate of the RMSE. We will talk more about this when we talk about *Restricted Maximum Likelihood Estimation* (REML).

Large Search Spaces

So far, we have been using a very finite search space that has been defined for us. For example, we limited the search space to 20,301 combinations of β_0 and β_1.

```r
nrow(
  crossing(
    b0 = seq(from = 30, to = 50, by = 0.1),
    b1 = seq(from = -5, to = 5, by = 0.1)
  )
)
```

```
[1] 20301
```

This allowed us to find the coefficient estimates to the nearest tenth. If we instead needed to find the estimates to the nearest hundredth, we would need to expand the number of combinations:

```r
nrow(
  crossing(
    b0 = seq(from = 30, to = 50, by = 0.01),
    b1 = seq(from = -5, to = 5, by = 0.01)
  )
)
```

```
[1] 2003001
```

This leads to a search space of 2,003,001 parameter combinations. If we need them to the nearest thousandth, the search space is 200,030,001 combinations.

Furthermore, in practice you would not have any idea which values of β_0 and β_1 to limit the search space to. Essentially you would need to search an infinite number of values unless you could limit the search space in some way. For many common methods (e.g., linear regression) finding the ML estimates is mathematically pretty easy (if we know calculus; see the section Way, Way, Way too Much Mathematics). For more complex methods (e.g., mixed-effect models) there is not a mathematical solution. Instead, mathematics is used to help limit the search space and then a grid search is used to hone in on the estimates.

ML Estimation in Regression Using R

Recall that the `lm()` function uses Ordinary Least Squares (OLS) estimation—it finds the coefficient estimates and RMSE that minimize the sum of squared residuals.

```r
lm.1 = lm(y ~ 1 + x)

# Get coefficient estimates
tidy(lm.1)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)    40.0       6.34      6.31 0.000231
2 x               2.74      1.98      1.38 0.203
```

```
# Get estimate for RMSE
glance(lm.1)
```

```
# A tibble: 1 x 11
  r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
      <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>  <dbl> <dbl> <dbl>
1     0.193        0.0926  14.0      1.92   0.203     2  -39.5  84.9  85.8
# ... with 2 more variables: deviance <dbl>, df.residual <int>
```

Under OLS estimation:

- $\hat{\beta}_0 = 40.01$
- $\hat{\beta}_1 = 2.74$
- $\hat{\sigma}_\epsilon = 13.99$

To compute ML estimates of the coefficients we will use the `mle2()` function from the **bbmle** package. To use the `mle2()` function, we need to provide a user-written function that returns the *negative log-likelihood* given a set of parameter inputs.

For simple regression, recall that we need to estimate three parameters: β_0, β_1, and σ_ϵ (RMSE). Below we have a function that inputs values for each of the three parameters, uses the inputted coefficient values to compute the residuals given the data, and returns the negative log-likelihood value assuming normality, independence, and homoscedasticity.

```
regress.ll = function(b0, b1, rmse) {
  # Use the following x and y values
  x = c(4, 0, 3, 4, 7, 0, 0, 3, 0, 2)
  y = c(53, 56, 37, 55, 50, 36, 22, 75, 37, 42)

  # Compute yhats and residuals
  yhats = b0 + b1 * x
  errors = y - yhats

  # Compute the negative log-likelihood
  neg_log_L = -sum(dnorm(errors, mean = 0, sd = rmse, log = TRUE))
  return(neg_log_L)
}
```

Now we can implement the `mle2()` function. This function requires the argument, `minuslogl=`, which takes the user written function returning the negative log-likelihood. It also requires a list of starting values for the input parameters in the user-written function. (Here we give values close to the OLS estimates as starting values.)

```
# Fit model using ML
library(bbmle)
mle.results = mle2(minuslogl = regress.ll, start = list(b0 = 40.0, b1 = 2.7, rmse = 13.98))

# View results
summary(mle.results)
```

```
Maximum likelihood estimation

Call:
```

```
mle2(minuslogl = regress.ll, start = list(b0 = 40, b1 = 2.7,
    rmse = 13.98))

Coefficients:
     Estimate Std. Error z value     Pr(z)
b0    40.0075     5.6721  7.0533 1.747e-12 ***
b1     2.7361     1.7674  1.5481    0.1216
rmse  12.5097     2.7973  4.4721 7.744e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log L: 78.90883
```

Under ML estimation:

- $\hat{\beta}_0 = 40.01$
- $\hat{\beta}_1 = 2.74$
- $\hat{\sigma}_\epsilon = 12.51$

Comparing the coefficient estimates ($\hat{\beta}_0$ and $\hat{\beta}_1$) between the two methods of estimation, we find they are quite similar. The estimate of σ_ϵ is different between the two estimation methods (although they are somewhat close in value).

Why do the estimates of the RMSE differ depending on the method of estimation? This is because the two methods use different formulas for computing RMSE. In OLS estimation, recall that the estimate of $\hat{\sigma}_\epsilon$ was:

$$\hat{\sigma}_\epsilon = \frac{\left(Y_i - \hat{Y}_i\right)^2}{n-2},$$

For ML estimation, the estimate for $\hat{\sigma}_\epsilon$ is:

$$\hat{\sigma}_\epsilon = \frac{\left(Y_i - \hat{Y}_i\right)^2}{n},$$

The smaller denominator in OLS results in a higher estimate of the variation. This, in turn, affects the size of the SE estimates for the coefficients (and thus the t- and p-values). When n is large, the differences in the estimates of $\hat{\sigma}_\epsilon$ are minimal and can safely be ignored.

Lastly, we note that the value of -2(log-likelihood) is the same for both the ML and OLS estimated models. This is a useful result. It allows us to use `lm()` to estimate the coefficients from a model and then use its log-likelihood as if we had fitted the model using ML.

Compute the Likelihood and Log-Likelihood from a Model Fitted with OLS

We can use R to directly compute the log-likelihood after we fit a model using the `lm()` function. To do this, we use the `logLik()` function.

```
lm.1 = lm(y ~ 1 + x)
logLik(lm.1)
```

```
'log Lik.' -39.45442 (df=3)
```

To compute the likelihood, we can use the `exp()` function to back-transform the log-likelihood to the likelihood (although generally we will work with the log-likelihood).

```
exp(-39.45442)
```

```
[1] 7.330998e-18
```

Way, Way, Way too Much Mathematics

A second, more convenient method to determine the ML estimates of the regression parameters is to use mathematics; specifically calculus. Remember, we can express the likelihood of the regression residuals mathematically as:

$$\mathcal{L}(\beta_0, \beta_1 | \text{data}) = p(\epsilon_1) \times p(\epsilon_2) \times \ldots \times p(\epsilon_n)$$

where the probability density of each residual (assuming normality) is:

$$p(\epsilon_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(\epsilon_i - \mu)^2}{2\sigma^2}\right]$$

In addition to normality, which gives us the equation to compute the PDF for each residual, the regression assumptions also specify that each conditional error distribution has a mean of 0 and some variance (that is the same for all conditional error distributions). We can call it σ^2_ϵ. Substituting these values into the density function, we get,

$$\begin{aligned} p(\epsilon_i) &= \frac{1}{\sigma_\epsilon\sqrt{2\pi}} \exp\left[-\frac{(\epsilon_i - 0)^2}{2\sigma^2_\epsilon}\right] \\ &= \frac{1}{\sigma_\epsilon\sqrt{2\pi}} \exp\left[-\frac{(\epsilon_i)^2}{2\sigma^2_\epsilon}\right] \end{aligned}$$

Now we use this expression for each of the $p(\epsilon_i)$ values in the likelihood computation.

$$\begin{aligned} \mathcal{L}(\beta_0, \beta_1 | \text{data}) &= p(\epsilon_1) \times p(\epsilon_2) \times \ldots \times p(\epsilon_n) \\ &= \frac{1}{\sigma_\epsilon\sqrt{2\pi}} \exp\left[-\frac{\epsilon_1^2}{2\sigma^2_\epsilon}\right] \times \frac{1}{\sigma_\epsilon\sqrt{2\pi}} \exp\left[-\frac{\epsilon_2^2}{2\sigma^2_\epsilon}\right] \times \ldots \times \frac{1}{\sigma_\epsilon\sqrt{2\pi}} \exp\left[-\frac{\epsilon_n^2}{2\sigma^2_\epsilon}\right] \end{aligned}$$

We can simplify this:

$$\mathcal{L}(\beta_0, \beta_1 | \text{data}) = \left[\frac{1}{\sigma_\epsilon\sqrt{2\pi}}\right]^n \times \exp\left[-\frac{\epsilon_1^2}{2\sigma^2_\epsilon}\right] \times \exp\left[-\frac{\epsilon_2^2}{2\sigma^2_\epsilon}\right] \times \ldots \times \exp\left[-\frac{\epsilon_n^2}{2\sigma^2_\epsilon}\right]$$

Now we will take the natural logarithm of both sides of the expression:

$$\ln\Big(\mathcal{L}(\beta_0, \beta_1 | \text{data})\Big) = \ln\left(\left[\frac{1}{\sigma_\epsilon\sqrt{2\pi}}\right]^n \times \exp\left[-\frac{\epsilon_1^2}{2\sigma^2_\epsilon}\right] \times \exp\left[-\frac{\epsilon_2^2}{2\sigma^2_\epsilon}\right] \times \ldots \times \exp\left[-\frac{\epsilon_n^2}{2\sigma^2_\epsilon}\right]\right)$$

Using our rules for logarithms and re-arranging gives,

$$l(\beta_0, \beta_1 | \text{data}) = -\frac{n}{2} \times \ln(2\pi\sigma^2_{\epsilon}) - \frac{1}{2\sigma^2_{\epsilon}} \times \sum \epsilon_i^2$$

Examining this equation, we see that the log-likelihood is a function of n, σ^2_{ϵ} and the sum of squared residuals (SSE). The observed data define n (the sample size) and the other two components come from the residuals which are a function of the parameters and the data.

Once we have this function, calculus can be used to find the analytic maximum. Typically before we do this, we replace ϵ_i with $Y_i - \hat{\beta}_0 - \hat{\beta}_1(X_i)$; writing the residuals as a function of the parameters (which we are solving for) and the data.

$$l(\beta_0, \beta_1 | \text{data}) = -\frac{n}{2} \times \ln(2\pi\sigma^2_{\epsilon}) - \frac{1}{2\sigma^2_{\epsilon}} \times \sum \bigg(Y_i - \hat{\beta}_0 - \hat{\beta}_1(X_i)\bigg)^2$$

To find the analytic maximum, we compute the partial derivatives *with respect to* $\hat{\beta}_0$ and $\hat{\beta}_1$, and set these equal to zero. This gives us a system of two equations with two unknowns ($\hat{\beta}_0$ and $\hat{\beta}_1$). We can then solve this set of equations to obtain each of the parameter estimates. Then, based on these values, we can compute the estimate for the RMSE.