MLE & Numerical Maximization

Boyie Chen

11/17/2019

Maximized Likelihood Function

Concept

After learning Newton's Method, we know how to do a numerical maximization with computer.(Simply use `nlm()` or `optim()`) Now we're applying the idea of maximization to find the MLE estimators.

If $Y_i \overset{iid}{\sim} N(\mu, \sigma^2)$, $i = 1, 2, \ldots, n$ Then we can find the likelihood function by the density function of Y

$$f_Y(y_i; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2}(\frac{y_i - \mu}{\sigma})^2}$$

Thus, the likelihood function is the product of density function. Note that the exogenous variables for density function are μ and σ^2. The exogenous variables for likelihood function is y_i.

$$\mathcal{L}(\mu, \sigma^2; y) = \prod_{i=1}^{n} f_Y(y_i) = \prod_{i=1}^{n} (2\pi\sigma^2)^{\frac{-1}{2}} e^{\sum_{i=1}^{n} \frac{-1}{2}(\frac{y_i - \mu}{\sigma})^2}$$

Since we're doing maximization, we can do a monotonic trasformation on it by taking log.

$$\ell(\mu, \sigma^2; y) = log\mathcal{L}(\mu, \sigma^2; y) = \sum_{i=1}^{n} log(2\pi\sigma^2)^{\frac{-1}{2}} - \frac{1}{2}\sum_{i=1}^{n}(\frac{y_i - \mu}{\sigma^2})^2 = \frac{-1}{2}\sum_{i=1}^{n} log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2$$

This is what we will form our likelihood function in the numeric maximization approach.

Note

We can take derivatives with respect to μ and σ^2, and find the MLE estimators for μ and σ^2. Which are $\hat{\mu}_{MLE} = \frac{1}{n}\sum_{i=1}^{n} Y_i = \bar{Y_n}$ and $\hat{\sigma}^2_{MLE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \mu)^2$, respectively.

Sometimes the MLE estimators (such as Logistic and Poisson Regression) have no closed form. In these cases, we have to adopt numerical maximization.

Finding Maximized Likelihood Numerically

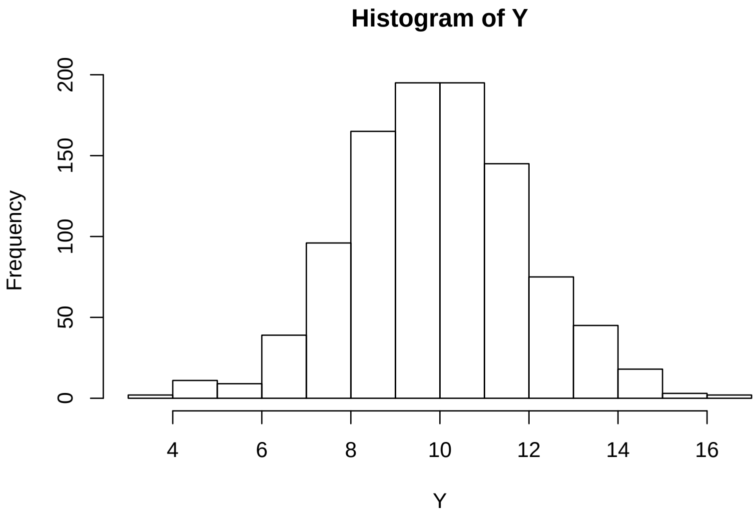
We have a very convenient function `dnorm()` which gives us the density function of normal distribution. Simply adopt it and define our likelihood function.

```
#MLE for mu & sigma^2 of normal dist.
#simply use the `dnorm()` function to write the log-likelihood fcn
mll = function(mu, sigma){
  logLikelihood = sum(log(dnorm(Y, mean = mu, sd = sigma)))
  return(-logLikelihood)
}
```

Since we're going to use `nlm()` which is doing the minimizing, so we put negative sign in `return()`.

We can use self-generating data to test whether our maximization is right or wrong. The true parameters are $\mu = 10, \sigma = 2$ in this example.

```r
#then we generate a series of pseudo data
n = 1000
set.seed(1234); Y = rnorm(n, 10, 2) #True parameters
hist(Y); mean(Y)
```




```
## [1] 9.946806
```

We can see that when the given parameters are different, the likelihoods are different. Our goal is to find the parameters θ that maximized the likelihood (or minimizing the negative likelihood). Since we generated data by ourselves, we know $\theta = (10, 2)$

```r
mll(10, 2) #the "maximum" => actually, find the negative minimum
```

```
## [1] 2109.283
```

```r
mll(11, 2); mll(9, 2); mll(10, 0); mll(10, 1)
```

```
## [1] 2247.582

## [1] 2220.985

## [1] Inf

## [1] 2907.729
```

Let the Computer Do the Job

we want to let the computer find the μ & σ s.t. `mll(mu, sigma)` is the smallest. We can use `nlm()`. And `mle()` in package `stats4` gives us the actual parameters of the MLE estimators. We can compare their results in the following code.

Using `mle()`

We first use `mle()`. We have to set initial guess as one of the inputs. Here we guess $\hat{\theta} = (0, 1)$

```r
library(stats4) #in order to use `mle()`
MaxLikeEst = mle(mll, start = list(mu = 0, sigma = 1)) ##
summary(MaxLikeEst) ##
```

```
## Maximum likelihood estimation
##
## Call:
```

```
## mle(minuslogl = mll, start = list(mu = 0, sigma = 1))
##
## Coefficients:
##        Estimate Std. Error
## mu      395.602   125.6702
## sigma 4053.905         NaN
##
## -2 log L: 18461.8
```

The parameters estimated are : $\hat{\mu} = 395.602$ and $\hat{\sigma} = 4053.905$. The $\hat{\theta}$ we found is very far away from $\hat{\theta} = (10, 2)$. This may result in bad initial guess since our maximization is non-linear.

Change initial guess

This time we guess $\hat{\theta} = (9, 1)$

```r
MaxLikeEst = mle(mll, start = list(mu = 9, sigma = 1)) ##
summary(MaxLikeEst) ##
```

```
## Maximum likelihood estimation
##
## Call:
## mle(minuslogl = mll, start = list(mu = 9, sigma = 1))
##
## Coefficients:
##        Estimate Std. Error
## mu    9.946806 0.06304564
## sigma 1.993678 0.04457996
##
## -2 log L: 4217.839
```

```r
mean(Y) #Recall mean(Y) is the MLE estimate for mu
```

```
## [1] 9.946806
```

The parameters estimated are : $\hat{\mu} = 9.946$ and $\hat{\sigma} = 1.9936$. The $\hat{\theta}$ we found is not far away from $\hat{\theta} = (10, 2)$

Using `nlm()`

We can also use `nlm()`. We also need to guess a initial point and put it as one of the inputs. I'll leave the message from the function below, it tells about how the estimated parameters change over iterations. Here we guess $\hat{\theta} = (0, 1)$ as well.

```r
nlm(mll, 0,1, print.level=2, hessian=TRUE) #we can find mu by non-linear minimization
```

```
## iteration = 0
## Step:
## [1] 0
## Parameter:
## [1] 0
## Function Value
## [1] 52375.79
## Gradient:
## [1] -9946.805
##
## iteration = 1
## Step:
## [1] 10
```

```
## Parameter:
## [1] 10
## Function Value
## [1] 2907.729
## Gradient:
## [1] 53.1994
##
## iteration = 2
## Parameter:
## [1] 9.946801
## Function Value
## [1] 2906.314
## Gradient:
## [1] -2.697359e-06
##
## Relative gradient close to zero.
## Current iterate is probably solution.
```

```
## $minimum
## [1] 2906.314
##
## $estimate
## [1] 9.946801
##
## $gradient
## [1] -2.697359e-06
##
## $hessian
##      [,1]
## [1,] 1000
##
## $code
## [1] 1
##
## $iterations
## [1] 2
```

You can see that the message from `nlm()` function tells us that it iterated 2 times. The estimated parameters $\hat{\mu}$ are $0, 10, 9.946801$ over iterations.

Matrix Form (Optional)

Writing likelihood function with bare hands...

Our goal is to find the MLE estimator for μ in multiple explainatory variable model. In a simple linear regression, we have to estimate μ_1 & μ_2, which are the intercept and the slope. In a multiple linear regression, we will have μ_1 and $\mu_2, \mu_3, \ldots, \mu_{k-1}$ represents the "intercept" and "slopes" in higher dimension space. . ### Notations

We have our regression model:

$$Y = X\beta + \epsilon$$

where $\epsilon \sim N(0, \sigma^2 I)$ and Y, X, β, ϵ are matrices with the following dimensions.

$Y_{n\times 1}$: contains n observations representing the realizations of explained variable.

$X_{n\times k}$: contains n observations for k explanatory variables.

For example, $k = 1$ means the first explanatory variable, and its n observations are: $X_{1,1}, X_{2,1}, X_{3,1}, \ldots, X_{n,1}$

Note that we denote X_1 as the first observations for k explanatory variables.

$$X_1 = [X_{1,1}, X_{1,2}, X_{1,3}, \ldots, X_{1,k}]$$

$\beta_{k\times 1}$: represents k parameters

$\epsilon_{n\times 1}$: represents n error terms

Given

$$\mu = X\beta$$

With the above notations, we can write the regression model as following:

$$y_i \sim N(\mu_i, \sigma^2)$$

where $\mu_i = X_i\beta$

Rewrite Likelihood Function

Thus we can rewrite our density function & likelihood function as following:

$$f_Y(y_i; X, \beta, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - X_i\beta)^2}{2\sigma^2}}$$

Then the likelihood function $\mathcal{L}$ is:

$$\mathcal{L}(X, \beta, \sigma^2; y_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - X_i\beta)^2}{2\sigma^2}}$$

And then with matrix form:

$$\mathcal{L} = (\frac{1}{\sqrt{2\pi\sigma^2}})^n e^{\frac{-1}{2\sigma^2}(y - X\beta)'(y - X\beta)}$$

So the log-likelihood function ℓ is:

$$\ell = log(\mathcal{L}) = \frac{-n}{2} log(2\pi) - \frac{n}{2} log(\sigma^2) - \frac{1}{2} \sum_{i=1}^{n} \frac{(y_i - X_i\beta)^2}{\sigma^2}$$

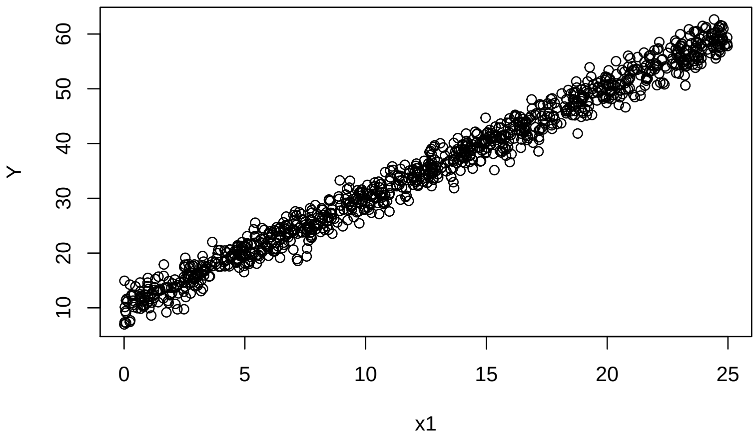
or in matrix form:

$$\ell = log(\mathcal{L}) = \frac{-n}{2} log(2\pi) - \frac{n}{2} log(\sigma^2) - \frac{1}{2\sigma^2}(y - X\beta)'(y - X\beta)$$

Generating Pseudo Data

We can generate the true $\beta = (1, 2)$. It means that our simple linear regression has an intercept 10 and slope 2.

```
#beta = c(mu_1, mu_2)
n = 1000
beta = c(10,2) #intercept is 10 & slope is 2
set.seed(1234)
x1 = runif(n, 0, 25) #x1 can come from any dist.
X = cbind(rep(1,n), x1)
Y = X %*% beta + rnorm(n, mean = 0, sd = 2) #where espilon comes from normal
plot(x1, Y) #intercept is 10 & slope is 2
```



Write the matrix form likelihood function:

```
mll = function(beta){
  logLikelihood = -n/2*log(2*pi) -n/2*log(beta[2]^2) - 1/(2*beta[2]^2)*t((Y-X%*%beta))%*%(Y-X%*%beta)
  return(-logLikelihood)
}
(t((Y-X%*%beta))) %*% (Y-X%*%beta) #note that this is a scalar
```

```
##          [,1]
## [1,] 3596.898
```

Numerical Maximization with `optim()` & `nlm()`

Similarly, we have to give the above function an initial guess. If our parameter β is a $k \times 1$ vector, then our initial guess should also be a $k \times 1$ vector.

Here, we first guess $\beta = (0, 1)$, i.e. we guess $\mu_1 = 0$ and $\mu_2 = 1$

```
est_opt = optim(c(0, 1), mll)
est_nlm = nlm(mll, c(10,3), print.level=2, hessian=TRUE) #we can find mu_1 & mu_2
```

And the parameters estimated are:

```
est_opt[1]
```

```
## $par
## [1] 10.127444  1.988197
```

```
est_nlm[2]
```

```
## $estimate
## [1] 10.132183  1.987735
```

which are not far from $\beta = (10, 2)$

Verifing our result in numerical maximization

We have a function `lm()` for linear regression model:

```
#compare the result of `lm()`
reg1 = lm(Y~X-1)
summary(reg1)
```

```
##
## Call:
```

```
## lm(formula = Y ~ X - 1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8387 -1.2591  0.0217  1.2764  5.4636
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## X   10.088982   0.120551   83.69   <2e-16 ***
## Xx1  1.991142   0.008245  241.50   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.897 on 998 degrees of freedom
## Multiple R-squared:  0.9975, Adjusted R-squared:  0.9975
## F-statistic: 2.027e+05 on 2 and 998 DF,  p-value: < 2.2e-16
```

```r
coef(reg1) #gives us the parameters estimated
```

```
##         X       Xx1
## 10.088982  1.991142
```

which are telling us what β should be. Note that `lm()` is based on OLS method, and there no garantee that the OLS estimator should always be the same as the MLE estimator. But they are the same here.

Extend to multiple explainatory variables

We have demonstrated an approach for estimating MLE estimators in simple linear regression. Now we're going to show the same approach in multiple linear regression.

Similarly, we generate pseudo data for 2 explainatory variables.

```r
#when beta is a (k by 1) vector, k = 3 here
n = 1000
beta = c(1,2,3)
set.seed(1234)
x1 = runif(n, 0, 100) #x1 can come from any dist.
x2 = rbinom(n, 10, 0.22)
X = cbind(rep(1,n), x1, x2)
```

Since we need to do the matrix multiplication, we have to adjust the `data type` of `X` with `as.matrix()`.

```r
class(beta)
```

```
## [1] "numeric"
```

```r
beta = as.matrix(beta)
dim(X)
```

```
## [1] 1000    3
```

```r
dim(beta)
```

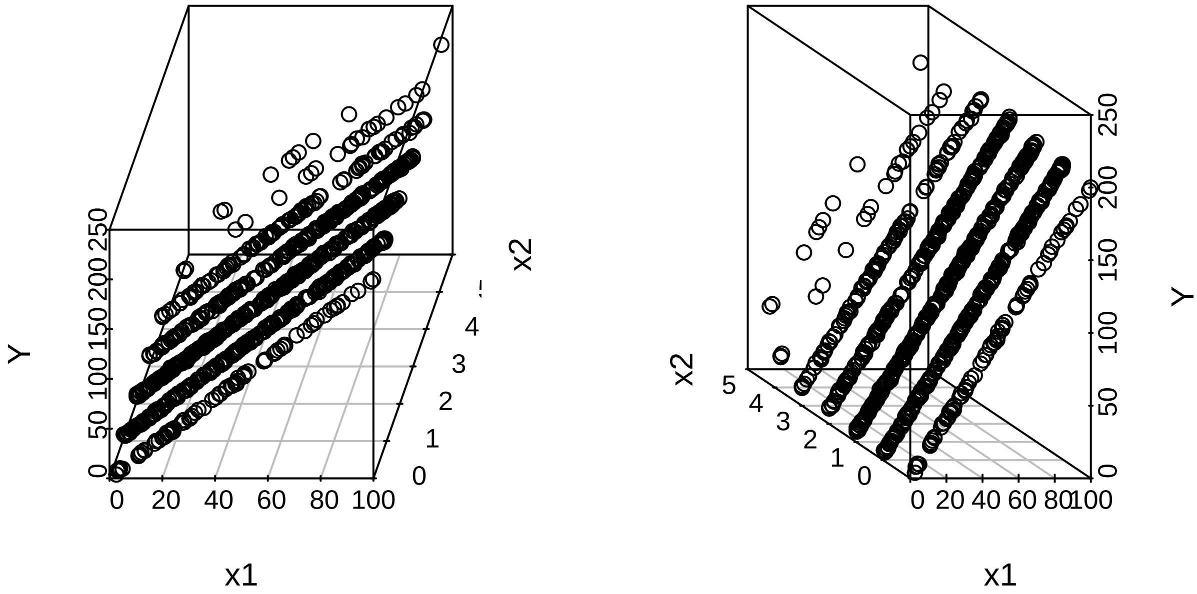
```
## [1] 3 1
```

Then we can do the matrix multiplication.

```r
Y = X %*% beta + rnorm(n, mean = 0, sd = 1) #where espilon comes from normal
```

We can look at the scatterplot.

```r
library(scatterplot3d)
par(mfrow = c(1,2))
scatterplot3d(x1, x2, Y, angle = 67.5)
scatterplot3d(x1, x2, Y, angle = 157.5)
```



```r
graphics.off()
```

Find the MLE Estimates

What we're going to find is the intercept and the slopes of the plane in 3-D space. Our likelihood function is exactly the same as above.

```r
mll = function(beta){
  logLikelihood = -n/2*log(2*pi) -n/2*log(beta[2]^2) - 1/(2*beta[2]^2)*t((Y-X%*%beta))%*%(Y-X%*%beta)
  return(-logLikelihood)
}
```

Give an initial guess $\beta = (1, 2, 3)$

```r
est_opt = optim(c(1,1,1), mll)
est_nlm = nlm(mll, c(1,1,1), print.level=2, hessian=TRUE)
```

```r
est_opt[1]
```

```
## $par
## [1] 0.1934479 2.0062585 3.1650868
```

```r
est_nlm[2]
```

```
## $estimate
## [1]   1.511413 160.129684   2.162356
```

The results deviate a lot from true $\beta = (1, 2, 3)$.

What happen to the `optim()` and `nlm()`? Because our log-likelihood function is non-linear, we have to assign initial guesses randomly.

Randomly assign initial guesses

We can use `rnorm(3)` to generate a 3×1 vector as an initial guess for β. We can use `runif()` as well.

```
set.seed(12345)
est_opt = optim(rnorm(3), mll)
set.seed(1234567)
est_nlm = nlm(mll, rnorm(3), print.level=2, hessian=TRUE)
```

The numeric method gives us more trustable estimates after randomly assign initial guesses.

```
est_opt[1]
```

```
## $par
## [1] 1.166377 1.997198 3.000919
```

```
est_nlm[2]
```

```
## $estimate
## [1] 1.151018 1.998070 2.981869
```

We can always compare the results.

```
#Let's cheat, use `lm()` to look at the answer
lm(Y~X-1)
```

```
##
## Call:
## lm(formula = Y ~ X - 1)
##
## Coefficients:
##     X    Xx1    Xx2
## 1.062  2.000  2.981
```

Our numerical method is not that bad.