Maximum Likelihood: Theory and Computation

Foundations of Statistical Inference (PLSC 503)

Simple Linear Model

Recall the Simple Linear Regression Model,

$$Y_i = \beta_0 + \beta_1 X_i = \epsilon_i$$

where $\mathbb{E}[\epsilon_i | X_i] = 0$ and $\text{Var}[\epsilon_i | X_i] = \sigma^2$. We previously defined the **residuals** as

$$\hat{\epsilon}_i = Y_i - \hat{Y}_i = Y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 X_i\right)$$

and showed the Ordinary Least Squared (OLS) estimates are the $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the **residual sum of squares**: $\sum_{i=1}^n \hat{\epsilon}_i^2$. That is,

$$\begin{aligned}\hat{\beta}_1 &= \frac{\sum_{i=1}^n \left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{\sum_{i=1}^n \left(X_i - \bar{X}\right)^2} \\ \hat{\beta}_0 &= \bar{Y} - \hat{\beta}_1 \bar{X}\end{aligned}$$

We also showed $\hat{\sigma}^2 = \frac{1}{n-p}\sum_{i=1}^n \hat{\epsilon}_i^2$ is an unbiased estimator for σ^2 ($p = 2$ here because we have 2 predictors in the model; $n - p$ is called the "degrees of freedom"). Suppose we impose the assumption that $\epsilon_i | X_i \sim \mathcal{N}(0, \sigma^2)$, or equivalently $Y_i | X_i \sim \mathcal{N}(\beta_0 + \beta_1 X_i, \sigma^2)$. We can then write out the likelihood function as,

$$\mathcal{L}(\beta_0, \beta_1, \sigma^2) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(Y_i - (\beta_0 + \beta_1 X_i))^2}{2\sigma^2}\right\}$$

or the greatly simplified log-likelihood,

$$\ell(\beta_0, \beta_1, \sigma^2) = -\frac{n}{2}\log 2\pi - n\log\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^n (Y_i - (\beta_0 + \beta_1 X_i))^2$$

which is much easier to work with when taking derivatives,

$$\begin{aligned}\frac{\partial \ell(\beta_0, \beta_1, \sigma^2)}{\partial \beta_0} &= -\frac{1}{2\sigma^2}\sum_{i=1}^n 2\left(Y_i - (\beta_0 + \beta_1 X_i)\right)(-1) \\ \frac{\partial \ell(\beta_0, \beta_1, \sigma^2)}{\partial \beta_1} &= -\frac{1}{2\sigma^2}\sum_{i=1}^n 2\left(Y_i - (\beta_0 + \beta_1 X_i)\right)(-X_i)\end{aligned}$$

when setting these derivatives equal to zero and solving for $\hat{\beta}_0$ and $\hat{\beta}_1$ we can ignore all the constants (e.g. $1/2\sigma^2$),

$$\begin{aligned}\sum_{i=1}^n Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i) &= 0 \\ \sum_{i=1}^n \left(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)\right) X_i &= 0\end{aligned}$$

with a bit of tedious algebra, you can show that the these are the same as the OLS estimators for β_0 and β_1. Thus, under the assumption that $\epsilon_i|X_i \sim \mathcal{N}(0, \sigma^2)$, OLS and MLE are equivalent. Another way to see this is to note that OLS minimizes $\sum_{i=1}^n \left(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)\right)^2$, whereas MLE **maximizes** $-\sum_{i=1}^n \left(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)\right)^2$. In general, maximizing a function over its argument is equivalent to minimizing that function over the same argument with a sign change. Finally, we can also take the derivative with respect to σ,

$$\frac{\partial \ell(\beta_0, \beta_1, \sigma^2)}{\partial \sigma} = -\frac{n}{\sigma} + 2\frac{1}{2\sigma^3}\sum_{i=1}^n (Y_i - (\beta_0 + \beta_1 X_i))^2$$

If we set this equal to zero and multiply through by σ^3 we have,

$$\begin{aligned}\hat{\sigma}^2 &= \frac{1}{n}\sum_{i=1}^n \left(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)\right)^2 \\ &= \frac{1}{n}\sum_{i=1}^n \hat{\epsilon}^2\end{aligned}$$

The **Generalized Linear Model** (GLM) is a generalization of the simple linear model that allows for more exotic model specifications. For example, suppose we want a parametric regression method for a binary outcome, e.g. $Y_i \in \{0, 1\}$. The GLM generalizes the linear model to situations like this by allowing the linear model to be related to the outcome variable via a **link function**. In the simplest scenario, the link function is the function applied to the mean outcome to return a linear predictor. In general, a GLM involves three components:

1. An exponential family of probability distributions (Normal, Binomial, Poisson, etc.).
2. A linear predictor, e.g. $X\beta$.
3. A link function g such that $\mathbb{E}[Y|X] = \mu = g^{-1}(\mu)$

You may have heard of "logit" or "probit" models before. We will see below that these terms are used to distinguish between two different link functions – the "logit" or "probit" (short for "probability unit"). We can fit pretty much any GLM in base `R` using the `glm()` function. All you need to do is tell `R` 1) what family of probability distribution to use for the error distribution; 2) the link function to be used in constructing the model.

Computation

Let's start by fitting a linear model that uses 1) a Normal (or "Gaussian") error distribution; and 2) an "identity" link. In the simple linear model, the "identity" function links the mean of the outcome variable to the linear term, e.g. $\mathbb{E}[Y_i] = \mu_i = \beta_0 + \beta_1 X_i$. In matrix notation the identity link function is $g(\mu) = \boldsymbol{X\beta}$, and its inverse or "mean function" is $\mu = \boldsymbol{X\beta}$.

```
# Setup: finite population of n = 10000 units
set.seed(503)
n <- 10000
X <- runif(n, min = -2, max = 2)

# Simple linear relationship w/ b0 = 1 and b1 = 3
e <- rnorm(n)
Y <- 1 + 3*X + e

# Take a random sample of n = 100 units
S <- sample(n, 100)
sample_df <- data.frame(Y = Y[S], X = X[S])

# Fit using glm(). Here we tell R that the error distribution is described by
# a "Gaussian" or Normal with an "identity" link function.
glm_fit <- glm(Y ~ X, family = gaussian(link = "identity"), data = sample_df)

# Extract coefficients
(glm_est <- coef(glm_fit))
```

```
## (Intercept)           X
##    1.162848    2.981715
```

```
# Extract SEs
(glm_ses <- sqrt(diag(vcov(glm_fit))))
```

```
## (Intercept)           X
##  0.10473665  0.09498855
```

As expected, these are the exact same results we obtain from OLS!

```
# Comparison to OLS:
lm_fit <- lm(Y ~ X, data = sample_df)
(lm_est <- coef(lm_fit))
```

```
## (Intercept)           X
##    1.162848    2.981715
```

```
(lm_ses <- sqrt(diag(vcov(lm_fit))))
```

```
## (Intercept)           X
##  0.10473665  0.09498855
```

We can use the `predict()` function in `R` to make predictions for Y given any value of X. For example, we can use either model to obtain the expected value of Y for an individual with $X = 2$, which is equal to $\hat{\beta}_0 + \hat{\beta}_1 \cdot 2$:

```
predict(glm_fit, newdata = data.frame(X = 2))
```

```
##        1
## 7.126277
```

```
sum(c(1, 2)*glm_est)
```

```
## [1] 7.126277
```

```
predict(lm_fit, newdata = data.frame(X = 2))
```

```
##        1
## 7.126277
```

```
sum(c(1, 2)*lm_est)
```

```
## [1] 7.126277
```

The Logistic Model

Suppose that we now have a binary outcome variable $Y_i \in \{0, 1\}$ and want to model the conditional probability $\Pr(Y_i = 1|X_i = x_i)$ as a function of a single covariate X_i and estimate the model using maximum likelihood. We can use **logistic regression** to solve this problem. The model with one covariate is,

$$p_i = \Pr(Y_i = 1|X_i = x_i) = \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)} = \frac{1}{1 + \exp[-(\beta_0 + \beta_1 x_i)]}$$

Note that $f(x) = \frac{\exp(x)}{1+\exp(x)}$ is called the logistic function. Further, if p is a probability then $p/(1-p)$ is the corresponding "odds" and the logit or "log-odds" is the logarithm of the odds. The inverse of the logistic function is,

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

This is where the model gets its name. The inverse of the logit function is the logistic function. The expression for p_i from above is equivalent to,

$$\text{logit}(p_i) = \beta_0 + \beta_1 x_i$$

Note that since the Y_i's are binary, the data are Bernoulli, e.g. $Y_i|X_i = x_i \sim \text{Bern}(p_i)$. Therefore, the likelihood function is,

$$\mathcal{L}(\beta_0, \beta_1) = \prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i}$$

We don't need to bother taking the log-likelihood and trying to derive the MLE because there is no analytic solution. Instead, we have to solve numerically. Fortunately, the `glm()` function in `R` handles this for us under the hood. Recall that a GLM has three components. For the logistic model the relevant exponential family is Binomial, the link function is the logit (i.e. $\text{logit}(p_i) = \beta_0 + \beta_1 x_i$), and the mean function is the inverse logit (i.e. $\frac{\exp(\beta_0+\beta_1 x_i)}{1+\exp(\beta_0+\beta_1 x_i)}$).

Computation

```r
# Setup: finite population of n = 10000 units
set.seed(503)
n <- 10000
X <- runif(n, min = -2, max = 2)

#  Apply inverse logit function to generate probabilities
p <- 1/(1 + exp(-(1 + 3*X)))

# Simulate Y | X ~ Bern(pi)
Y <- rbinom(n = n, size = 1, prob = p)

# Take a random sample of n = 100 units
S <- sample(n, 100)
sample_df <- data.frame(Y = Y[S], X = X[S])

# Here we use the Binomial family for the likelihood & a logit link function.
logit_fit <- glm(Y ~ X, family = binomial(link = "logit"), data = sample_df)

# Extract coefficients
(logit_est <- coef(logit_fit))
```

```
## (Intercept)           X
##   0.9082855   2.4264793
```

```r
# Extract SEs
(logit_ses <- sqrt(diag(vcov(logit_fit))))
```

```
## (Intercept)           X
##   0.3325801   0.4831294
```

Note that the logistic regression models the **log-odds** of the outcome, so the estimates we get back cannot be interpreted as probabilities. Rather, we'd say something like "according to the model, a one unit increase in X corresponds to an increase in the log-odds that $Y = 1$ by 2.8 units". If we instead want to obtain probabilities then we need to apply the logistic or "inverse logit" function. For example, suppose we want to know the predicted probability that $Y = 1$ for $X = 1$,

```r
# Calculate by hand using inverse-logit:
exp(sum(logit_est))/(1+exp(sum(logit_est)))
```

```
## [1] 0.9656024
```

```r
# Use the predict function w/ type = "response"
predict(logit_fit, newdata = data.frame(X = 1), type = "response")
```

```
##         1
## 0.9656024
```

Aside: the Linear Probability Model

What would happen if we just used OLS instead? This is often called a "linear probability model" and though you will hear people oppose the use of it in practice, typically on the grounds that the estimated coefficients can imply probabilities outside of the unit inteval $[0, 1]$. In practice, it is mostly harmless[1].

```
# Comparison to OLS:
lm_fit <- lm(Y ~ X, data = sample_df)

# Calculate E[Y=1|X=1] by hand
sum(coef(lm_fit))
```

```
## [1] 0.917647
```

```
# Use the predict function
predict(lm_fit, newdata = data.frame(X = 1))
```

```
##        1
## 0.917647
```

[1]See Chapter 3.4.2 of *Mostly Harmless Econometrics* for an elaboration; or just read this blog post http://www.mostlyharmlesseconometrics.com/2012/07/probit-better-than-lpm/ and Winston Lin's comment

The Probit Model

Suppose we have a binary outcome variable $Y_i \in \{0, 1\}$, but instead want to represent the conditional probability $\Pr(Y_i = 1|X_i = x_i)$ using a Probit Model. The model with one covariate is,

$$\Pr(Y_i = 1|X_i = x_i) = \Phi(\beta_0 + \beta_1 x_i)$$

The major difference between Probit and Logit is that Probit uses a different link function. You guessed it, the probit link function! This is just another name for the quantile function associated with the Normal CDF (`qnorm()` in `R`), which is the inverse of the Normal CDF,

$$\text{probit}(p) = \Phi^{-1}(p)$$

The likelihood function for this Probit Model is,

$$\mathcal{L}(\beta_0, \beta_1) = \prod_{i=1}^{n} \Phi(\beta_0 + \beta_1 x_i)^{y_i} \cdot (1 - \Phi(\beta_0 + \beta_1 x_i))^{1-y_i}$$

and the log-likelihood is

$$\ell(\beta_0, \beta_1) = \sum_{i=1}^{n} y_i \Phi(\beta_0 + \beta_1 x_i) + (1 - y_i) \log (1 - \Phi(\beta_0 + \beta_1 x_i))$$

Again, don't bother trying to solve this analytically we have to used numerical methods to find $\hat{\beta}_0$ and $\hat{\beta}_1$. Our friend `glm()` is happy to help with this. The syntax here is very similar to what we used for the logit model, we just need to specify that we instead want a probit link function.

Computation

```r
# Setup: finite population of n = 10000 units
set.seed(503)
n <- 10000
X <- runif(n, min = -2, max = 2)

#  Apply Normal CDF (inverse probit) to generate probabilities
p <- pnorm(1 + 3*X)

# Simulate Y | X ~ Bern(pi)
Y <- rbinom(n = n, size = 1, prob = p)

# Take a random sample of n = 100 units
S <- sample(n, 100)
sample_df <- data.frame(Y = Y[S], X = X[S])

# Here we use the Binomial family for the likelihood & a probit link function.
probit_fit <- glm(Y ~ X, family = binomial(link = "probit"), data = sample_df)

# Extract coefficients
(probit_est <- coef(probit_fit))
```

```
## (Intercept)           X
##   0.9035874   2.5610030
```

```r
# Extract SEs
(probit_ses <- sqrt(diag(vcov(probit_fit))))
```

```
## (Intercept)           X
##   0.2747525   0.5530643
```

Again, direct interpretation of the coefficients is ugly. We can instead use the predict function to generate predicted probabilities. For example, suppose we want to know the predicted probability that $Y = 1$ for $X = 1$,

```r
# Calculate by hand using inverse-probit:
pnorm(sum(probit_est))
```

```
## [1] 0.9997345
```

```r
# Use the predict function w/ type = "response"
predict(probit_fit, newdata = data.frame(X = 1), type = "response")
```

```
##         1
## 0.9997345
```