“Clarify-Style” Confidence Intervals Using Monte Carlo Sampling

Michael DeCrescenzo

April 16, 2020

This document describes an approach to generating confidence intervals without fussing with too much math. In many situations, confidence intervals can be calculated *analytically* (meaning, with a mathematical formula). For more complicated models or more complicated predictions, we can approximate confidence bounds without having to do a lot of math ourselves. We can do this using *numerical* (simulation-based) methods instead of analytical methods.

Intuition

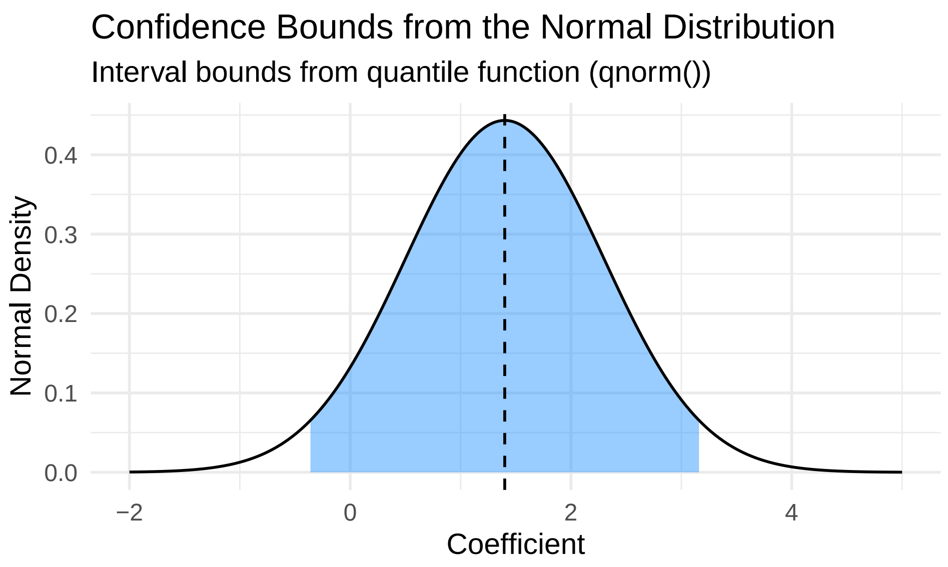
We will show how to approximate confidence bounds by simulating draws from the *normal distribution*.

Imagine that we estimate a linear model with a large number of observations (so our T distribution looks like a Normal distribution) and get a coefficient of 1.4 with standard error 0.9. We can build a confidence interval using this information like so:

$$\begin{aligned} \text{Confidence interval} &= \hat{\beta} \pm z(1-\alpha/2) \times \text{se}\left(\hat{\beta}\right) && (1) \\ &= 1.4 \pm 1.96 \times 0.9 && (2) \\ &= (-0.36, 3.16) && (3) \end{aligned}$$

where $z(1-\alpha/2)$ is the critical $z - value$ for our given confidence level α. In this is example, if we set α to be .05 then the z-value is 1.96.

What’s crucial to see is that these are the same bounds as we would get if we (a) specify a normal distribution with mean 1.4 and standard deviation of 0.9, grabbing the quantile values at .025 and .975.



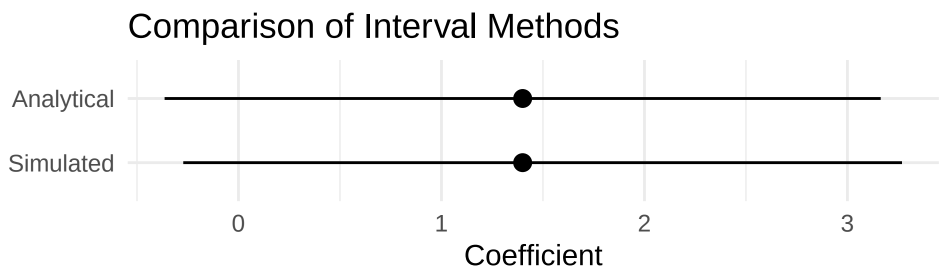
We can approximate the same bounds by *simulating a large number of Normal draws*, and grabbing the "empirical quantiles" from those simulations. Here is a simple demo of how we could do that in R.

```r
coef_value <- 1.4
se_value <- 0.9
```

```r
# simulate normal draws
sim_values <- rnorm(n = 1000, mean = coef_value, sd = se_value)

# extract bounds of the "inner 95%"
empirical_int <-
  quantile(sim_values, probs = c(0.025, 0.975)) %>%
  print()
##       2.5%      97.5%
## -0.2719749  3.2688940
```

If I plot the two intervals side by side, you can see that they are very similar (but not guaranteed to be exact).



Multivariate Example

Generating a confidence interval from a single parameter is simple even without simulating. The simulation method comes in handy when we have to demonstrate uncertainty in more complicated quantities, such as model predictions (which are functions of multiple model parameters), differences in predictions, and other transformations of model predictions. I will demonstrate with the Data Exercise 3 data (salaries of men and women).

```
# program needs to be source()d first
d3 <-
  create_exercise_3(seed = 813) %>%
  as_tibble() %>%
  print()
## # A tibble: 60 x 4
##    Rating Credits   Sex Salary
##     <dbl>   <dbl> <dbl>  <dbl>
##  1     77      11     0   63.8
##  2     53      11     0   58.6
##  3     48      14     0   58.7
##  4     73      15     0   62.8
##  5     54      20     0   58.9
##  6     83      19     0   65.8
##  7     75       0     0   64.1
##  8     71       8     0   62.2
##  9     88       4     0   65.7
## 10     63      17     0   61.6
## # ... with 50 more rows
```

I estimate the model that you should have done for the exercise.

```
mod <- lm(
  Salary ~ Sex + Rating + Sex*Rating + Credits,
  data = d3
)
```

The Multivariate Normal Distribution

In order to simulate draws of multiple model parameters at the same time, you use the *multivariate normal distribution*, which is a higher dimensional normal distribution. The mean of a multivariate normal isn't just one value; it is a vector of values. Similarly, the variance isn't one value either; it is a matrix of variances and covariances defined for every dimension of the distribution. To simulate coefficient draws from a model, we simulate from a multivariate normal distribution using our *coefficient estimates* as the mean vector…

```
# rounding for the sake of prettier output.

# the means
coef(mod) %>% round(2)
## (Intercept)         Sex      Rating     Credits  Sex:Rating
##       48.89       -4.36        0.20        0.00        0.15
```

…and the *variance-covariance matrix* of the coefficients. The elements that appear as 0 aren't actually 0; they're only *rounding down* to 0.

```
# variance-covariance matrix
vcov(mod) %>% round(2)
##             (Intercept)   Sex Rating Credits Sex:Rating
## (Intercept)        1.53 -1.35  -0.02   -0.01       0.02
## Sex               -1.35  1.68   0.02    0.01      -0.02
## Rating            -0.02  0.02   0.00    0.00       0.00
## Credits           -0.01  0.01   0.00    0.00       0.00
## Sex:Rating         0.02 -0.02   0.00    0.00       0.00
```

There are a few packages with functions to simulate multivariate Normal draws. I use {mvtnorm}. The function is rmvnorm(), r for "random" draws, mvnorm for "multivariate Normal." Here is an example using only a small number of draws.

```
mvtnorm::rmvnorm(
  n = 10,
  mean = coef(mod),
  sigma = vcov(mod)
)
##       (Intercept)       Sex    Rating      Credits Sex:Rating
##  [1,]    50.28498 -5.768011 0.1809320 -0.017071920  0.1724174
##  [2,]    50.33655 -5.701656 0.1817793 -0.026208635  0.1661655
##  [3,]    47.51275 -3.672307 0.2130140  0.012187571  0.1418649
##  [4,]    50.82571 -6.558309 0.1794635 -0.034762587  0.1845102
##  [5,]    49.44161 -4.267590 0.1882282  0.004454289  0.1521620
##  [6,]    49.18015 -5.506763 0.1944158 -0.001578588  0.1689113
##  [7,]    48.14831 -4.315415 0.2032750  0.035537631  0.1563224
##  [8,]    49.15469 -4.600125 0.1936414 -0.013213864  0.1616612
##  [9,]    50.78781 -6.507414 0.1764310 -0.010620787  0.1816309
## [10,]    48.36344 -4.531076 0.2053517  0.017686532  0.1528337
```

Predictions from simulated draws

Remember the matrix-form equation for the regression and model predictions,

$$Y = X\beta + \varepsilon, \tag{4}$$
$$\hat{Y} = X\hat{\beta}, \tag{5}$$

where Y, $\hat{Y}$, and ε are $n \times 1$, X is $n \times p$ including a column of 1s, and β is $p \times 1$ including the intercept.

To describe predictions from M simulated coefficient draws, let's introduce notation for $\tilde{Y}$, an $n \times M$ matrix of predictions, and $\tilde{\beta}$, a $p \times M$ matrix of simulated coefficients, each with *one column for each simulated draw*.

$$\tilde{Y} = X\tilde{B} \tag{6}$$

To make $\tilde{Y}$, we need to build X, simulate $\tilde{\beta}$, and then multiply.

```
# X must same variables in the same order as coefs
# transmute() is a combo of mutate() and select().
# I fit Credits at the mean for this set of predictions.
X <- d3 %>%
  transmute(
    const = 1,
    Sex,
    Rating,
    Credits = mean(Credits),
    Sex_Rating = Sex * Rating
  ) %>%
  print()
## # A tibble: 60 x 5
##     const   Sex Rating Credits Sex_Rating
##     <dbl> <dbl>  <dbl>   <dbl>      <dbl>
##  1      1     0     77    10.2          0
##  2      1     0     53    10.2          0
##  3      1     0     48    10.2          0
##  4      1     0     73    10.2          0
##  5      1     0     54    10.2          0
##  6      1     0     83    10.2          0
##  7      1     0     75    10.2          0
##  8      1     0     71    10.2          0
##  9      1     0     88    10.2          0
## 10      1     0     63    10.2          0
## # ... with 50 more rows
```

```
# we transpose (t()) this result
#   from M x p to p x M
sim_beta <-
  mvtnorm::rmvnorm(
    n = 1000,
    mean = coef(mod),
    sigma = vcov(mod)
  ) %>%
  t()

# check dimensions
dim(sim_beta)
## [1]    5 1000

# calculate Yhat.
# convert X to matrix so that it works with %*%,
#   which is matrix multiplication
Ys <- as.matrix(X) %*% sim_beta

# check dimensions
dim(Ys)
## [1]   60 1000
```

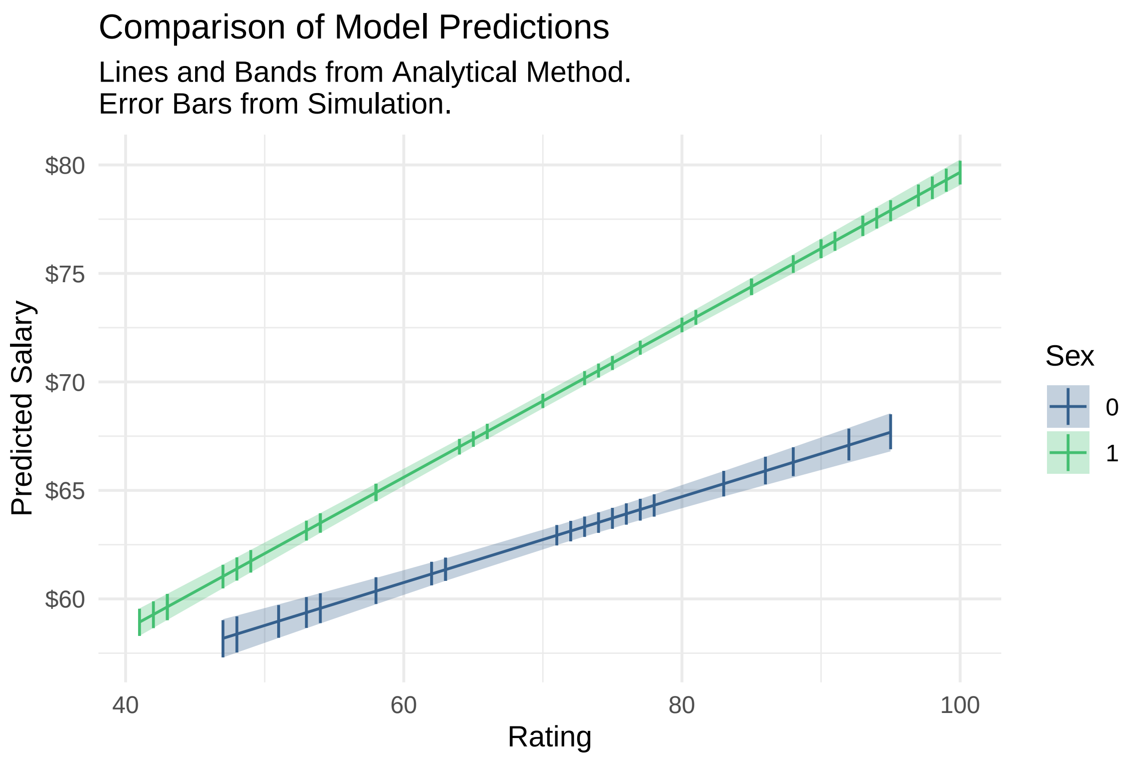
Now that we have M simulated predictions, grab the bounds at the quantiles for whichever α level you desire.

```
# apply() gives 2 x N array.
# I take the additional steps to
#   transpose, convert to data frame,
#   and rename columns
bounds <-
  apply(X = Ys, MARGIN = 1, FUN = quantile, c(.025, .975)) %>%
  t() %>%
  as_tibble() %>%
  set_names(c("sim.conf.low", "sim.conf.high")) %>%
  print()
## # A tibble: 60 x 2
##    sim.conf.low sim.conf.high
##           <dbl>         <dbl>
##  1         63.6          64.6
##  2         58.7          60.1
##  3         57.5          59.2
##  4         62.9          63.8
##  5          58.9          60.3
##  6          64.7          65.9
##  7          63.2          64.2
##  8          62.5          63.4
##  9          65.7          67.0
## 10          60.8          61.9
## # ... with 50 more rows
```

I could add the right-hand data as well as the point estimate from this design matrix (X) by using the point estimates of the coefficients instead of the simulations.

```
sim_predictions <- bounds %>%
  mutate(
    yhat = as.vector(as.matrix(X) %*% coef(mod))
  ) %>%
  bind_cols(X) %>%
  select(-const) %>%
  print()
## # A tibble: 60 x 7
##    sim.conf.low sim.conf.high  yhat   Sex Rating Credits Sex_Rating
##           <dbl>         <dbl> <dbl> <dbl>  <dbl>   <dbl>      <dbl>
##  1          63.6          64.6  64.1     0     77    10.2          0
##  2          58.7          60.1  59.4     0     53    10.2          0
##  3          57.5          59.2  58.4     0     48    10.2          0
##  4          62.9          63.8  63.3     0     73    10.2          0
##  5          58.9          60.3  59.6     0     54    10.2          0
##  6          64.7          65.9  65.3     0     83    10.2          0
##  7          63.2          64.2  63.7     0     75    10.2          0
##  8          62.5          63.4  62.9     0     71    10.2          0
##  9          65.7          67.0  66.3     0     88    10.2          0
## 10          60.8          61.9  61.3     0     63    10.2          0
## # ... with 50 more rows
```

How do these predictions compare to the predictions you would get from `augment()` or some other method?



Closing Notes

To see why this could be useful, imagine that instead of plotting separate trends for men and women, you wanted to plot the *difference in the predictions* for men and women. How would you calculate the confidence interval for the *difference* in these predictions? With a simulation-based method, you wouldn't have to. You could simply simulate predictions for men and women, calculate the difference in each iteration, and then summarize the distribution of differences by calculating the quantiles as we did above.

It's worth reiterating that this is an approximation to the proper confidence intervals. The approximation error comes from two sources: using the *Normal* distribution instead of the appropriate T distribution, and the imprecision of calculating CI bounds using a finite number of simulations. Using a large number of simulations should minimize the numerical error, but the distributional error will always be there. For this reason, you may want to avoid this method for smaller samples.