Model reliability

Katie Schuler

2024-11-12

⚠ Under Construction

0.1 Basic idea

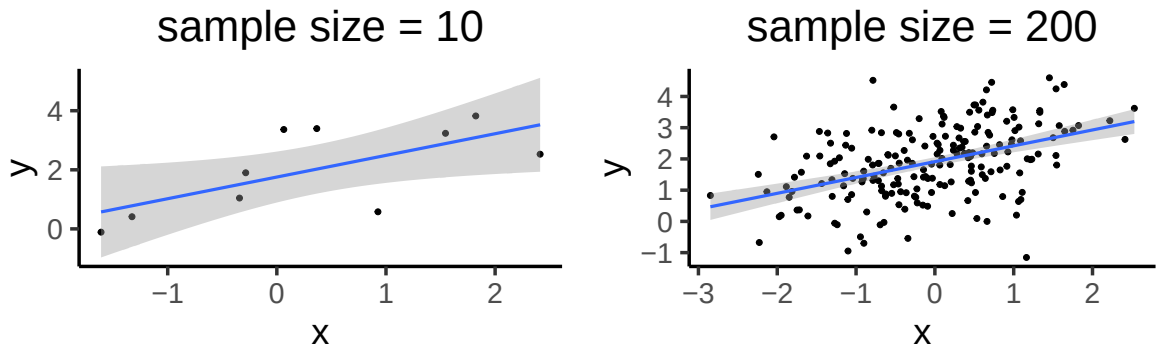
Imagine you've just fit a model to some data, and now you have values for its parameters. But how sure can we be about those parameter values? Well, this uncertainty has to do with the difference between the data you sampled and the larger population it represents. We're not really interested in the model parameters that just fit your sample—our real goal is to find parameters that would best describe the entire population from which that sample came.

Recall from Sampling Distribution week that, for any statistic computed on a sample from a population there's some sampling error. We add error bars to show the range of values we'd expect our statistic to take on if we took new samples from the population. In previous weeks we were working with single summary statistics like the mean, but these same principles apply to models: model parameters estimated from a sample will also vary because of sampling error. By putting error bars on these parameters, we indicate how these estimates might change with different samples.

Model reliability, then, is about how stable those parameter estimates are in the face of sampling error. And the more data we collect, the more reliable our estimates become, reducing uncertainty in our model's parameters.

```
ggplot(data_n10, aes(x = x, y = y)) +
    geom_point() +
    labs(title = "sample size = 10") +
    geom_smooth(method = "lm", formula = "y ~ x")
ggplot(data_n200, aes(x = x, y = y)) +
    geom_point() +
    labs(title = "sample size = 200") +
```

```
    geom_smooth(method = "lm", formula = "y ~ x")
    model_n10 <- lm(y ~ x, data = data_n10)
summary(model_n10)
model_n200 <- lm(y ~ x, data = data_n200)
summary(model_n200)
```




```
Call:
lm(formula = y ~ x, data = data_n10)

Residuals:
    Min      1Q  Median      3Q     Max
-1.8557 -0.6285 -0.0113  0.6370  1.5624

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.7548     0.3740   4.692  0.00156 **
x             0.7333     0.2862   2.562  0.03352 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.138 on 8 degrees of freedom
Multiple R-squared:  0.4508,    Adjusted R-squared:  0.3821
F-statistic: 6.566 on 1 and 8 DF,  p-value: 0.03352
```

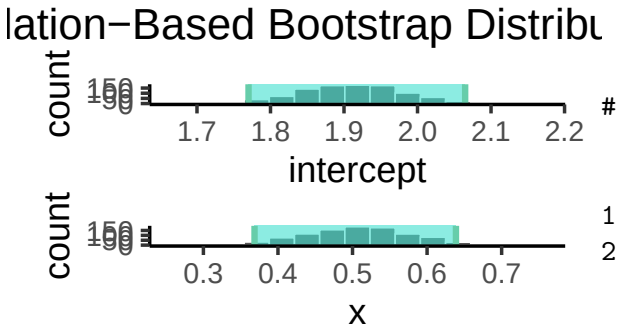
```
Call:
lm(formula = y ~ x, data = data_n200)

Residuals:
    Min      1Q  Median      3Q     Max
-3.6565 -0.6757  0.0689  0.6032  3.0019

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.91308    0.07233  26.448  < 2e-16 ***
x            0.50704    0.07236   7.007 3.72e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.021 on 198 degrees of freed
Multiple R-squared:  0.1987,    Adjusted R-squared:  0
F-statistic: 49.1 on 1 and 198 DF,  p-value: 3.724e-1
```

0.2 Calculating reliability

To calculate error bars on our model parameter estimates, we can use bootstrapping, a non-parametric approach that makes minimal assumptions and uses the data itself to construct a sampling distribution. You already know how to do this: we used bootstrapping earlier when we explored sampling distributions! Here's a reminder of the process: we generate many "bootstrap" samples by resampling our original data (with replacement). Then, we fit our model to

each of these samples and observe the distribution of parameter estimates across them. This distribution provides a way to assess the variability in parameter estimates if we were to draw different samples from the population.

```
# get the observed fit
observed_fit <- data_n200 %>%
    specify(y ~ x) %>%
    fit()

# constructe the sampling distribution
boot_fits <- data_n200 %>%
    specify(y ~ x) %>%
    generate(reps = 1000, type = "bootstrap") %>%
    fit()

# get the confidence interval
ci <- boot_fits %>%
    get_confidence_interval(
        point_estimate = observed_fit, # from above
        level = 0.95
    )

# visualize the sitribution and ci
boot_fits %>%
    visualize() +
    shade_ci(endpoints = ci)
ci
```




```
# A tibble: 2 x 3
  term      lower_ci upper_ci
  <chr>        <dbl>    <dbl>
1 intercept    1.77     2.06
2 x            0.369    0.639
```

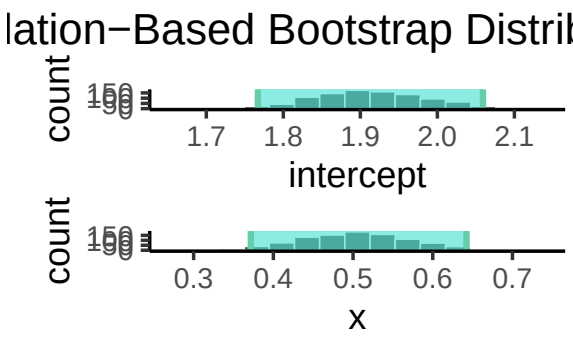
When we fit a model using lm() in R, for example, the summary output already provides us with standard errors (S.E.) for each parameter. These standard errors are based on a parametric approach, where we assume that the residuals (errors) follow a normal distribution. Parametric methods like this can work well for large, normally distributed data, providing

standard error estimates directly from the model. Bootstrapping, in contrast, doesn't assume normality and is more flexible for small or non-normal data, allowing us to approximate error distributions without relying on these assumptions.

```
# get the observed fit
observed_fit <- data_n200 %>%
    specify(y ~ x) %>%
    fit()

# constructe the sampling distribution
boot_fits <- data_n200 %>%
    specify(y ~ x) %>%
    generate(reps = 1000, type = "bootstrap") %>%
    fit()

# get the confidence interval
ci <- boot_fits %>%
    get_confidence_interval(
        point_estimate = observed_fit, # from above
        level = 0.95,
        type = "se"
    )

# visualize the sitribution and ci
boot_fits %>%
    visualize() +
    shade_ci(endpoints = ci)
ci
summary(model_n200)
```

What if we want to understand not just individual parameters but also a combined metric or the overall function described by our mode? Bootstrapping can be used here well: we compute whatever quantity we care about for each bootstrap sample, then examine how that quantity varies across all iterations.

There are other methods resampling techniques beyond bootstrapping that you might hear about in the context of model reliability, like **jackknifing** and **split-half analysis**. Jackknifing is faster but generally less accurate and flexible: we leave out one observation at a time and calculate our estimates on the remaining data. Split-half analysis - where we split the data and analyze each half separately — gives a rough sense of sampling error. However, unless computation is a limiting factor, bootstrapping is usually preferred because it offers a balance of accuracy and adaptability.




```
# A tibble: 2 x 3
  term      lower_ci upper_ci
  <chr>        <dbl>    <dbl>
1 intercept    1.77     2.06
2 x            0.372    0.642
```

```
Call:
lm(formula = y ~ x, data = data_n200)

Residuals:
    Min      1Q  Median      3Q     Max
-3.6565 -0.6757  0.0689  0.6032  3.0019

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.91308    0.07233  26.448  < 2e-16 ***
x            0.50704    0.07236   7.007 3.72e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.021 on 198 degrees of freedom
Multiple R-squared:  0.1987,    Adjusted R-squared:  0.1947
F-statistic:  49.1 on 1 and 198 DF,  p-value: 3.724e-11
```

0.3 Correlated errors

When a model has multiple parameters, the errors in their estimates may be correlated. This happens because, in models with more than one parameter, sampling error isn't just a single value; it's part of a multidimensional distribution. So, putting separate error bars on individual parameters can miss the bigger picture, especially when parameters interact.

Imagine a simple case with a linear regression model with two correlated predictors (a child's age and height, for example). If these predictors are highly correlated, then increasing the weight of one can often be offset by decreasing the weight of the other. This results in a kind of "trade-off" between the weights, where an error in one weight often corresponds to an opposite error in the other. When we look at how these weights vary across samples, we might see a negative correlation in the errors between the two weights.

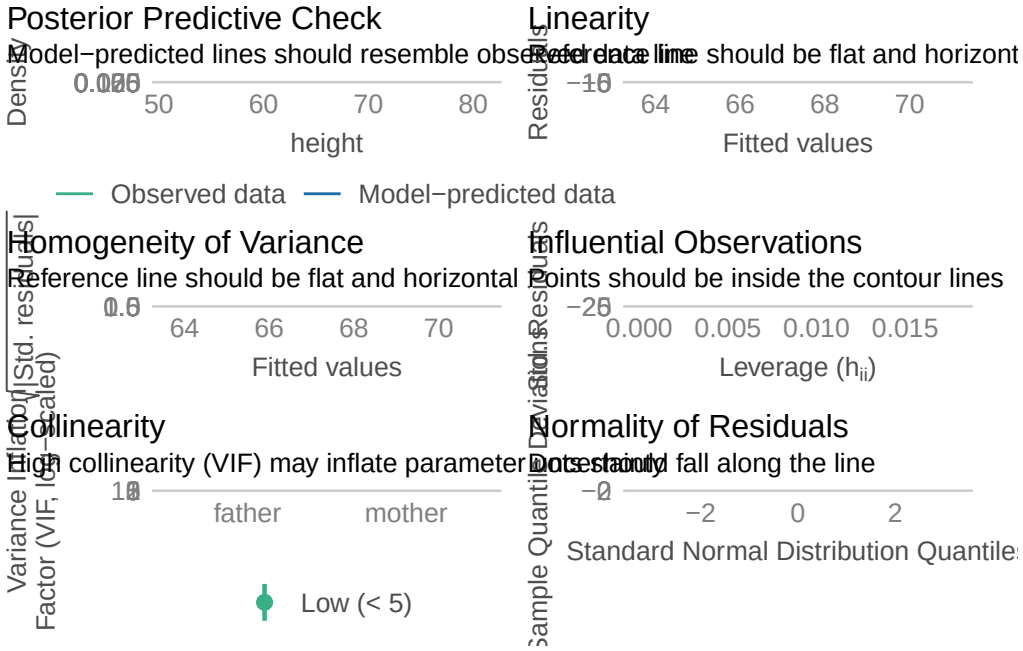
We can use a simple function `check_model()` that is part of the `easystats` package to check our model for correlated errors. `check_model()` returns a number of important checks of model assumptions. Colinearity checks for correlated errors and returns a VIF (Variance Inflation Factor):

- VIF = 1: no correlation with other predictors
- VIF > 1 and < 5: Moderate correlation with other predictors (acceptable)
- VIF > 5: Correlation with other predictors is problematic (high multicolinearity)

```
library(easystats)
```

```
# Attaching packages: easystats 0.7.3
✓ bayestestR  0.15.0   ✓ correlation 0.8.6
✓ datawizard  0.13.0   ✓ effectsize  0.8.9
✓ insight     0.20.5   ✓ modelbased  0.8.9
✓ performance 0.12.4   ✓ parameters  0.23.0
✓ report      0.5.9    ✓ see         0.9.0
```

```
galton_model <- lm(height ~ father + mother, data = Galton)
check_model(galton_model)
```



For highly correlated regressors, these individual weights may have low reliability, meaning their exact values can fluctuate. However, the model as a whole may still be reliable. Even if the individual weights vary, the combined model output (a weighted sum of the predictors) can remain stable across different samples.

0.4 Accuracy v. Reliability

Model acuracy and model reliability are closely related concepts in model building, but they aren't the same. Accuracy refers to how close a model's predictions are to the true values we want to predict. Reliability, on the other hand, is about the model's stability—how consistent the model's parameters and outputs are when new data is sampled.

Let's break down the difference with four scenarios:

1. **Reliable and accurate**: The model is both close to the true model and stable across different samples. This is the ideal case, indicating we have enough data to produce both a precise and consistent model fit.
2. **Reliable but inaccurate**: The model parameters are stable across samples, meaning it's reliable, but it's far from the true model. This could happen if our model is structurally limited or misses some aspect of the data, even if we have plenty of data for stable estimates.

3. **Unreliable but accurate**: This situation is unlikely. Without enough data, the model's predictions will fluctuate widely from sample to sample, making it hard to consistently approximate the true model. So, without reliability, achieving accuracy is improbable.
4. **Unreliable and inaccurate**: Here, the model's estimates are unstable and far from the true model. This could be due to either insufficient data or an inappropriate model choice that doesn't match the data's structure. With limited data, it's hard to tell which factor is to blame.

In short, accuracy tells us how correct a model is, while reliability tells us how consistently it performs across samples. Both qualities are essential but address different aspects of model quality.