Residual Analysis and Model Improvements

October 8, 2024

Table of contents

Generating *Toy* Data

For this notebook, we'll work with a simulated data set. This data will result in commonly occurring issues with *residuals* (prediction errors) from a straight-line linear regression model. We'll see what those issues look like and how to adjust for them.

Let's generate the data below. Can you see what kinds of associations there are between `x1`, `x2`, and `x3` with the response variable `y`?

```
set.seed(123)

nobs <- 100
my_data <- tibble(
```

```
  x1 = runif(nobs, -2, 2),
  x2 = rnorm(nobs, 0, 1),
  x3 = runif(nobs, 0, 2*pi),
  y = exp(0.5*x1 + 0.4*x2^2 - 0.6*sin(x3) + rnorm(nobs, 0, 0.3))
    )
```

We'll assume that this is our *training* data for today's class meeting.

Plot the Data

Produce plots between `y` and each of the `xi` *covariates* (predictors/variables/features). I'm building the first one for you. Store the plots as `p1`, `p2`, and `p3` and then use `{patchwork}` (already loaded) to arrange them as `(p1 + p2) / p3`.

```
p1 <- my_data %>%
  ggplot() +
  geom_point(aes(x = x1,
                 y = y),
             alpha = 0.75)

#Add the other plots here...

#Arrange them using {patchwork} here...
```

Proposed Model Form and Model Fit

Given our last class discussion on multiple linear regression, we could propose a model form of

$$\mathbb{E}[y] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

Construct and fit that model using the code chunk below. Hint, you'll need a model *specification*, a *recipe*, then a *workflow* to package the specification and recipe together, and finally you'll fit your workflow to the data.

Model Quality Assessments

As in our most recent class discussions, once we have a fitted model, we'll assess the model quality. There are three main types of assessments we'll engage in.

i) Global Test for Model Utility
ii) Term-Based Tests for Significance of Individual Terms
iii) Residual Analyses

We'll conduct each of those tests below.

Global Test for Model Utility

Use `glance()` to access your "global" (overall) model performance metrics. Determine the results of the test for global model utility:

$$\begin{array}{rcl} H_0 & : & \beta_1 = \beta_2 = \beta_3 = 0 \\ H_a & : & \text{At least one of the } \beta_i\text{'s is non-zero} \end{array}$$

```
#Conduct the global test for model utility
```

Individual Term-Based Tests

Use `extract_fit_engine()` and `tidy()` to access the individual term-based metrics. Determine the results of the hypothesis tests examining the significance of each model term

$$\begin{array}{rcl} H_0 & : & \beta_i = 0 \\ H_a & : & \beta_i \neq 0 \end{array}$$

in the code chunk below.

```
#Conduct the tests for significance of individual model terms
```

Before we decide to reduce the model and remove insignificant predictors, what does it mean that a predictor is not significant for this model?

Residual Analyses

Let's take a look at the *residuals* (model prediction errors). We'll generally look at five types of plot:

i) the distribution of residuals (we want normality)
ii) residuals versus response (we want randomness and constant standard deviation…no patterns)
iii) residuals versus predictions (we want randomness and constant standard deviation…no patterns)
iv) residuals versus each predictor (we want randomness and constant standard deviation…no patterns)

Remove the `#| eval: false` chunk option from the code chunk below and run the code to produce the residual plots. If you named your fitted model object something other than `mlr_fit`, then update the model name for each of the plots and re-run the code chunk. Analyse the results.

> **Note**
>
> Setting `#| eval: false` prevents the code in the chunk from being run when the notebook is rendered. I did this here because this code could not be run until you constructed your model.

```
my_data_residuals <- mlr_fit %>%
  augment(my_data)

p1 <- my_data_residuals %>%
  ggplot() +
  geom_histogram(aes(x = .resid,
                     y = ..density..),
                 color = "black",
                 fill = "purple") +
  geom_density(aes(x = .resid),
               fill = "purple",
               alpha = 0.75)

p2 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x1, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
```

```
              lwd = 2) +
  geom_smooth(aes(x = x1, y = .resid))

p3 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x2, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x2, y = .resid))

p4 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x3, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x3, y = .resid))

p5 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = y, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = y, y = .resid))

p6 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = .pred, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = .pred, y = .resid))

(p1 + p2) / (p3 + p4) / (p5 + p6)
```

Reacting to Residual Plots

There is quite a bit of *badness* here across the plots.

i) The distribution of residuals is not normal.
ii) There are patterns in all of the plots of residuals against predictors.
iii) There are associations between the residuals and both the response and predicted values.

We'll address these in-order.

Non-Normally Distributed Residuals

Because of the skew in the distribution of the residuals, let's try predicting `log(y)` instead of predicting `y`. This is a common technique for dealing with a skewed response variable.

```
my_data <- my_data %>%
  mutate(log_y = log(y))

mlr_log_rec <- recipe(log_y ~ x1 + x2 + x3, data = my_data)

mlr_log_wf <- workflow() %>%
  add_model(mlr_spec) %>%
  add_recipe(mlr_log_rec)

mlr_log_fit <- mlr_log_wf %>%
  fit(my_data)
```

Now, let's check out our new residual plots!

```
my_data_residuals <- mlr_log_fit %>%
  augment(my_data)

p1 <- my_data_residuals %>%
  ggplot() +
  geom_histogram(aes(x = .resid,
                     y = ..density..),
                 color = "black",
                 fill = "purple") +
  geom_density(aes(x = .resid),
               fill = "purple",
               alpha = 0.75)
```

```r
p2 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x1, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x1, y = .resid))

p3 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x2, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x2, y = .resid))

p4 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x3, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x3, y = .resid))

p5 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = y, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = y, y = .resid))

p6 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = .pred, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
```

```
              lwd = 2) +
  geom_smooth(aes(x = .pred, y = .resid))

(p1 + p2) / (p3 + p4) / (p5 + p6)
```

What have we "fixed"?

Associations Between Residuals and Predictors

Associations between the residuals and the `x2` and `x3` predictors still seem to exist. We'll address this by making transformations of these predictors. We'll talk more about how this works later in our course but, for now, inspect the code and then run it.

```
mlr_log_tx2x3_rec <- recipe(log_y ~ x1 + x2 + x3, data = my_data) %>%
  step_poly(x2, degree = 2, options = list(raw = TRUE)) %>%
  step_mutate(x3 = sin(x3))

mlr_log_tx2x3_wf <- workflow() %>%
  add_model(mlr_spec) %>%
  add_recipe(mlr_log_tx2x3_rec)

mlr_log_tx2x3_fit <- mlr_log_tx2x3_wf %>%
  fit(my_data)
```

Again, let's check out our new residual plots!

```
my_data_residuals <- mlr_log_tx2x3_fit %>%
  augment(my_data)

p1 <- my_data_residuals %>%
  ggplot() +
  geom_histogram(aes(x = .resid,
                     y = ..density..),
                 color = "black",
                 fill = "purple") +
  geom_density(aes(x = .resid),
               fill = "purple",
               alpha = 0.75)

p2 <- my_data_residuals %>%
```

```
  ggplot() +
  geom_point(aes(x = x1, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x1, y = .resid))

p3 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x2, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x2, y = .resid))

p4 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = x3, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = x3, y = .resid))

p5 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = y, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
  geom_smooth(aes(x = y, y = .resid))

p6 <- my_data_residuals %>%
  ggplot() +
  geom_point(aes(x = .pred, y = .resid)) +
  geom_hline(yintercept = 0,
             color = "red",
             linetype = "dashed",
             lwd = 2) +
```

```
  geom_smooth(aes(x = .pred, y = .resid))

(p1 + p2) / (p3 + p4) / (p5 + p6)
```

What have we "fixed" this time?

Assessing Our Updated Model

Now that we've built this new and improved model, let's assess its quality and compare it to our original model.

Global Model Utility

I'll provide the code we used to assess our original model. Add your own to assess the quality of our improved `mlr_log_tx2x3_fit` model.

```
mlr_fit %>%
  glance()
```

Conduct and assessment of overall model quality for our new model in the code chunk below. Note that our updated model form differs from the original model. The new model we've hypothesized is:

$$\mathbb{E}\left[\log\left(y\right)\right] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2 + \beta_4 \sin\left(x_3\right)$$

```
#Extract the results for our newest model here...
```

Individual Term-Based Assessments

Again, I'll remind you of the individual model term assessments from our original model.

```
mlr_fit %>%
  extract_fit_engine() %>%
  tidy()
```

Extract the individual term-based metrics for our updated model.

```
#Extract the results for our newest model here...
```

Summary

Add what you've learned here...