BIFX 553 - Discussion 4

Randy Johnson

February 9, 2017

Modeling and testing complex interactions

As a reminder, let's write out our model from a few weeks back:

$$nodes_i = \beta_0 + \beta_1 age_i + \beta_2 size_i + \beta_3 grade_i + \varepsilon_i$$

- $nodes_i$ is our outcome variable
- $\beta_0, \beta_1, \ldots$ are our regression coefficients
- $age_i, size_i, \ldots$ are our predictors
- ε_i are the error terms for the model (i.e. how far off is the model prediction from what we observed)

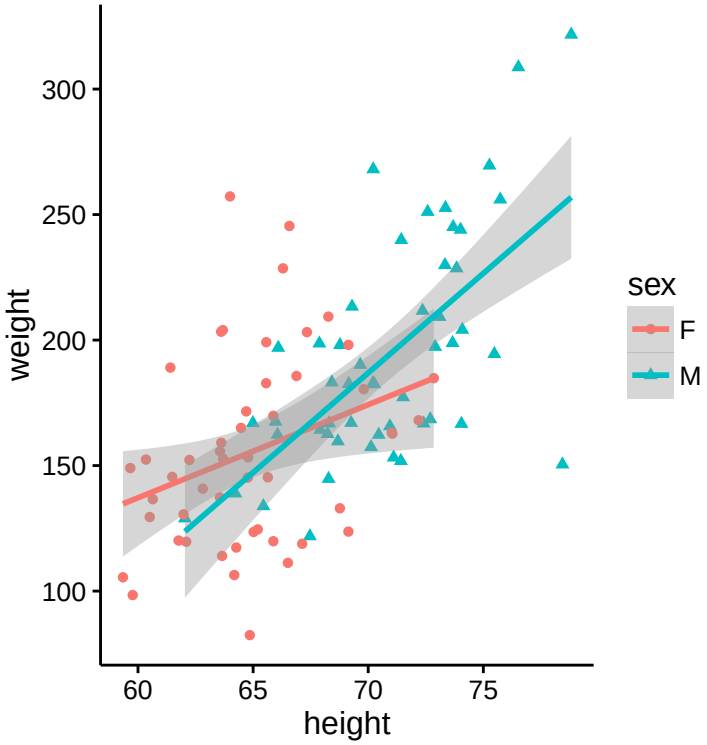
Fitting categorical predictors

For variables that are categorical (e.g. sex, race), we normally include dummy variables for them in the model.

$$weight_i = \beta_0 + \beta_1 height_i + \beta_2 (sex_i = M) + \varepsilon_i$$

```
set.seed(293847)
# data for our categorical predictor example
bmi <- data_frame(sex = c(rep(c('M', 'F'), each = 50)),
                  height = rnorm(100, mean = ifelse(sex == 'M', 70, 65), # average US height
                                 sd = ifelse(sex == 'M', 4, 3.5)),
                  bmi = ifelse(sex == 'M', 2^rnorm(100, 4.7004, 0.2305), # ave BMI - men
                                           2^rnorm(100, 4.6439, 0.3086)), # ave BMI - women
                  weight = ((height / 12 / 3.28)^2 * bmi) * 2.2) # calculate weight

# take a graphical look at the relationsihp between height and weight
ggplot(bmi, aes(x = height, y = weight, group = sex)) +
  geom_point(aes(color = sex, shape = sex)) +
  geom_smooth(method = 'lm', aes(color = sex))
```




```
# and now for a statistical look
lm(weight ~ height + sex, data = bmi) %>%
  tidy()
```

```
##          term    estimate std.error  statistic      p.value
## 1 (Intercept) -236.278051 69.799873 -3.3850785 1.028381e-03
## 2      height    6.032593  1.071821  5.6283572 1.764689e-07
## 3        sexM    2.437032  9.625050  0.2531969 8.006519e-01
```

In the background R is doing the following:

```
# create a dummy variable for "sex == 'M'"
bmi$male <- as.integer(bmi$sex == 'M')

# this model is exactly the same as the model above
lm(weight ~ height + male, data = bmi) %>%
  tidy()
```

```
##          term    estimate std.error  statistic      p.value
## 1 (Intercept) -236.278051 69.799873 -3.3850785 1.028381e-03
## 2      height    6.032593  1.071821  5.6283572 1.764689e-07
## 3        male    2.437032  9.625050  0.2531969 8.006519e-01
```

When we write this out in a formula, we get

$$\begin{aligned} weight_i &= \beta_0 + \beta_1 height_i + \beta_2 (sex_i = M) + \varepsilon_i \\ &= \beta_0 + \beta_1 height_i + \beta_2 (1) + \varepsilon_i \\ &= \beta_0 + \beta_1 height_i + \beta_2 + \varepsilon_i \end{aligned}$$

when sex_i is 'M' and

$$\begin{aligned} weight_i &= \beta_0 + \beta_1 height_i + \cancel{\beta_2(0)} + \varepsilon_i \\ &= \beta_0 + \beta_1 height_i + \varepsilon_i \end{aligned}$$

So, exactly what does β_2 represent? It is interpreted as the difference in height between a male and a female who are otherwise similar (i.e. in our case, they are the same height). If we work this out, we get:

$$\begin{aligned} \mathrm{E}(weight_i - weight_j | height_i = height_j) =& \beta_0 + \beta_1 height_i + \beta_2(sex_i = M) \\ & - (\beta_0 + \beta_1 height_j + \beta_2(sex_j = M)) \\ =& \cancel{\beta_0} + \cancel{\beta_1 height_i} + \beta_2(1) \\ & - \left(\cancel{\beta_0} + \cancel{\beta_1 height_j} + \cancel{\beta_2(0)}\right) \\ =& \beta_2 \\ =& 2.4 \text{ in} \end{aligned}$$

An asside on the interpretation of the intercept

What does the intercept (β_0) really mean? In the model above, β_0 is what we get when $sex_i =$ 'F' and $height_i = 0$. So, the predicted weight of a Female who is 0 inches tall would be

$$\begin{aligned} \mathrm{E}(weight_i) &= \beta_0 + \beta_1 height_i + \beta_2(sex_i = M) \\ &= \beta_0 + \cancel{\beta_1(0)} + \cancel{\beta_2(0)} \\ &= \beta_0 \\ &= -236 \text{ lbs} \end{aligned}$$

This, of course, makes no sense, neither the height nor the weight. For this reason, the intercept is often ignored. If we wanted the intercept to make sense, we could recenter $height_i$ like so:

```
# lets center height at the sample median (or there abouts)
median(bmi$height)
```

```
## [1] 67.89892
```

```
bmi <- mutate(bmi,
              ht_med = height - 67) # call it an even 5'7"

lm(weight ~ ht_med + sex, data = bmi) %>%
  tidy()
```

```
##          term   estimate std.error  statistic      p.value
## 1 (Intercept) 167.905685  5.608466 29.9378991 8.506954e-51
## 2      ht_med   6.032593  1.071821  5.6283572 1.764689e-07
## 3        sexM   2.437032  9.625050  0.2531969 8.006519e-01
```

Now, the intercept has a more useful meeing. It represents the expected weight of a 5'7" female:

$$\begin{aligned} \mathrm{E}(weight_i) &= \beta_0 + \beta_1(height_i - 67) + \beta_2(sex_i = M) \\ &= \beta_0 + \cancel{\beta_1(67 - 67)} + \cancel{\beta_2(0)} \\ &= \beta_0 \\ &= 168 \text{ lbs} \end{aligned}$$

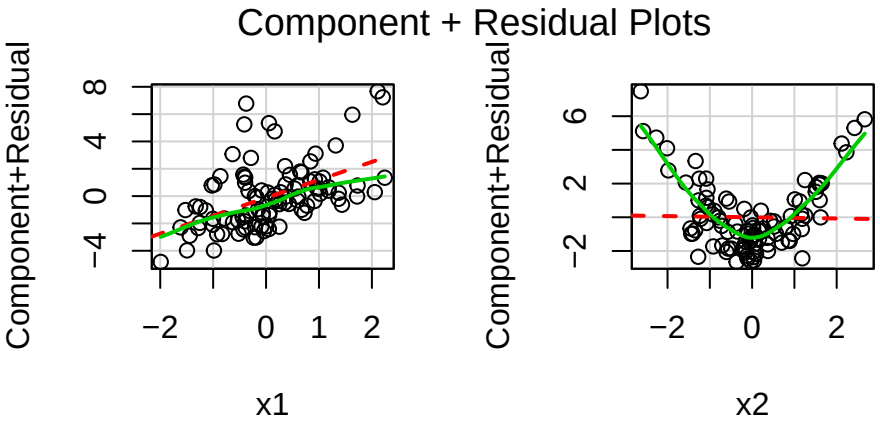
Dealing with Non-linearity

Simple non-linear terms

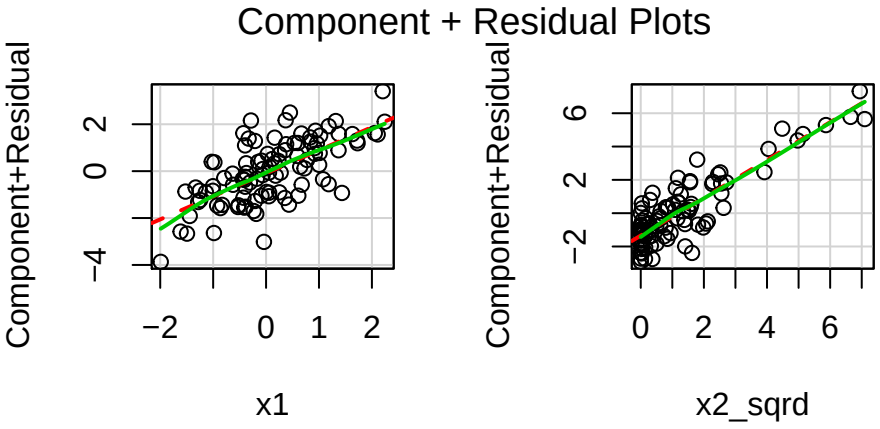
We saw in Discussion 3 that the number of nodes in the `gbsg` data set was more log-normally than normally distributed. Also, we looked at an example where the relationsihp between a predictor was a quadratic relationship, rather than a linear relationship. What do we do in these cases?

```
# Example 1: quadratic relationship between predictor and outcome
set.seed(293847)
tmp <- data_frame(x1 = rnorm(100),
                  x2 = rnorm(100),
                  y = x1 + x2^2 + rnorm(100))

# wrong model... y ~ x2 is a quadratic relationsihp
lm(y ~ x1 + x2, data = tmp) %>%
  crPlots()
```



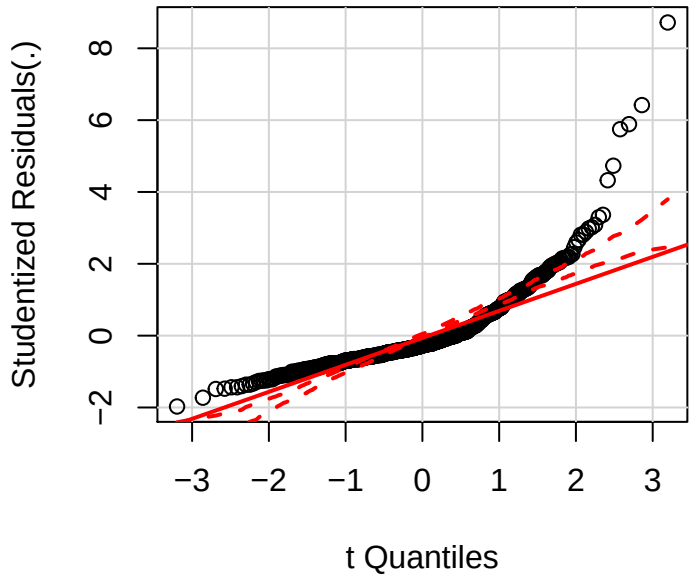

```
# create a new variable and try this
mutate(tmp, x2_sqrd = x2^2) %>%
  lm(formula = y ~ x1 + x2_sqrd) %>%
  crPlots() # looks much better!
```



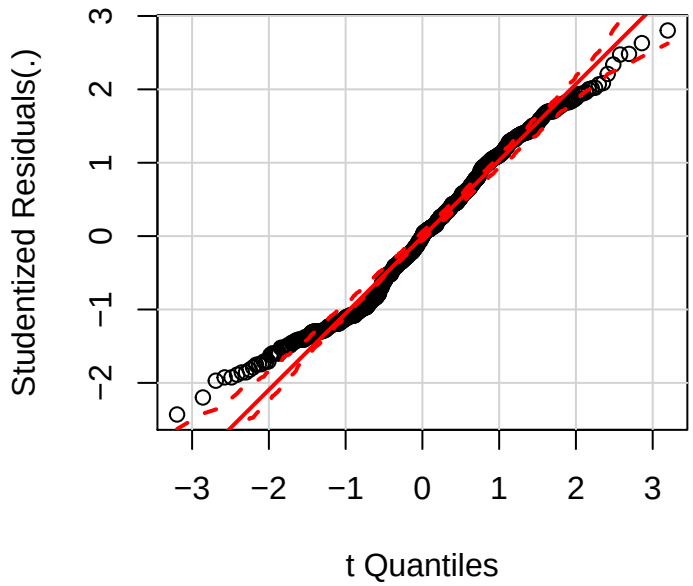

```
# Example 2: log-normally distributed outcome variable
load('../1-26/gbsg.RData')

# wrong model... nodes is log-normally distributed
lm(nodes ~ size + grade, data = gbsg) %>%
```

```
  qqPlot()
```




```
# lets try with log(nodes) ... still not perfect, but much better
mutate(gbsg, lnodes = log(nodes)) %>%
  lm(formula = lnodes ~ size + grade) %>%
  qqPlot()
```



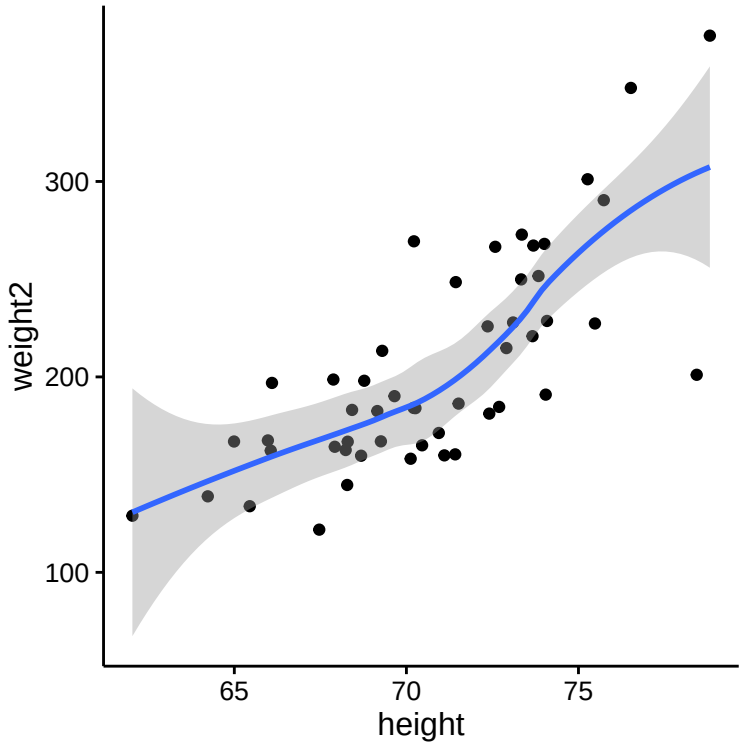
Splines

Sometimes, the relationship between a predictor and the outcome is neither linear, nor does it conform to a nice function that we can use for a simple transformation. In such a case, splines can help.

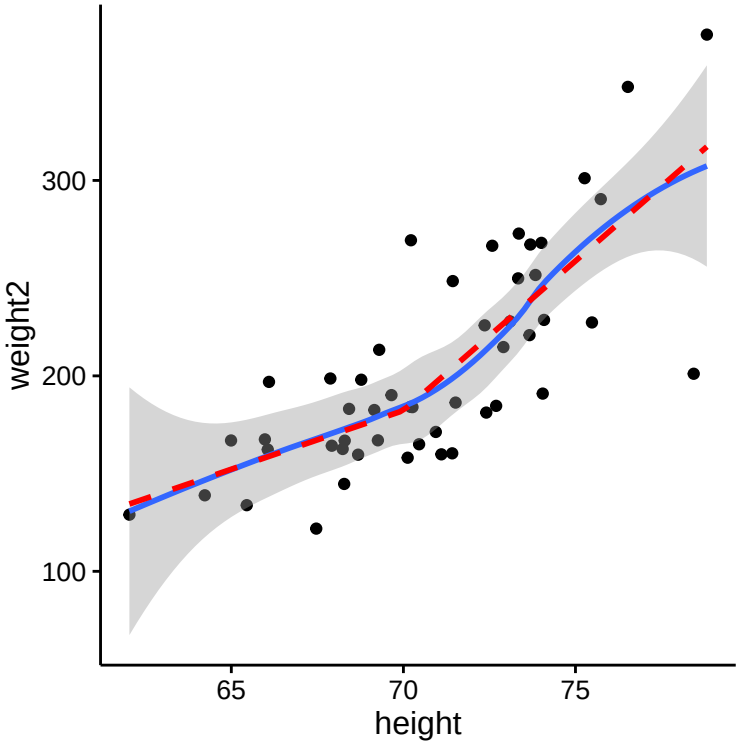
```
# add some additional weight for individuals over 70 inches tall
bmi <- mutate(bmi, weight2 = ifelse(height < 70, weight, weight + 6*(height - 70)))

# what does this look like?
```

```
g <- filter(bmi, sex == 'M') %>%
     ggplot(aes(x = height, y = weight2)) +
     geom_point() +
     geom_smooth()
g
```




```
# we can include a 1st degree spline with a knot at height = 70
g + geom_smooth(method = 'lm', se = FALSE, color = 'red', linetype = 2,
                formula = y ~ bs(x, knots = 70, degree = 1))
```




```
# the linear regression model would look like this:
mdl1 <- lm(weight2 ~ bs(height, knots = 70, degree = 1), data = subset(bmi, sex == 'M'))
mdl2 <- lm(weight2 ~ bs(height, knots = 70, degree = 1) + sex, data = bmi)
```

Interpreting the regression coefficients directly is beyond the scope of this class, but there is an easy way to use this model to predict. This should work with any of the models we will be generating in this class. All you need is the model output from `lm` and a data frame with the values needed to predict. In `mdl1`, we only need to include `height`, but we also need to include `sex` for `mdl2`.

```
# for the male-only model
data_frame(height = 70) %>%
  predict(object = mdl1)
```

```
##        1
## 182.2475
```

```
# for the full model
data_frame(height = c(70, 67),
           sex = c('M', 'F')) %>%
  predict(object = mdl2)
```

```
##        1        2
## 179.8799 163.0864
```

You can also use the `bs` function to fit higher order polynomial splines (the default is a cubic spline).

Non-parametric regression

You may have noticed that the default output of `geom_smooth()` is a smooth function that looks sort of like a moving average. The default smoother used for data sets with less than 1000 observations is called loess, and a generalized additive model is used with larger data sets. You won't get p-values from this, but it can be a valuable tool to graphically see what is going on.

```r
# make up some data and plot the smoothing line
data_frame(x = rnorm(500, 0, 3),
           y = x - .1*x^2 + .005*x^3 + rnorm(500)) %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_smooth()
```