

Lecture 8: Multiple linear regression and beyond

Stats 32: Introduction to R for Undergraduates

Harrison Li

April 25, 2024

Agenda

- 1 Multiple linear regression
- 2 Categorical predictors
- 3 Polynomial regression
- 4 Locally weighted regression

Reading:

Multiple linear regression

Multiple linear regression

Last lecture, we introduced simple linear regression to explore the relationship between an outcome variable y and a single predictor x .

Multiple linear regression extends this idea to relate an outcome variable to more than one predictor variable.

For instance, we may want to model how someone's blood pressure depends on multiple physical factors: height, weight, age, race, etc.

Multiple linear regression

Let p be the total number of predictor variables, which we label x_1, \dots, x_p . Then the multiple linear regression model takes the form

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p + e$$

where again e is random noise centered around 0.

We now have $p + 1$ different coefficients, or *parameters*, in the model: $\beta_0, \beta_1, \dots, \beta_p$.

OLS (again)

Just like in simple linear regression, the coefficients in multiple linear regression are typically computed using ordinary least squares, i.e. they are chosen to minimize

$$\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_{i,1} - \dots - \beta_p x_{i,p})^2$$

where the i -th observation consists of $p + 1$ data points: the p predictor values $x_{i,1}, \dots, x_{i,p}$ and the outcome y_i .

Multiple linear regression

Just like with simple linear regression, we can use the built-in `lm()` function to fit a multiple linear regression.

Let's try to understand the relationship between `Length` and the predictors `Shucked.wt`, `Viscera.wt`, and `Shell.wt` in the `abalone` data (let's call these x_1 , x_2 , and x_3 , respectively). Note the `+` signs separating the different predictors on the right hand side of the formula.

```
library(tidyverse)
library(moderndivide)
abalone <- read_csv("abalone.csv")
model <- lm(formula=Length ~ Shucked.wt + Viscera.wt + Shell.wt, data=abalone)
model %>%
  get_regression_table()
```

```
## # A tibble: 4 x 7
##   term      estimate std_error statistic p_value lower_ci upper_ci
##   <chr>      <dbl>    <dbl>    <dbl>  <dbl>  <dbl>  <dbl>
## 1 intercept  0.331    0.005    69.3    0      0.322  0.34
## 2 Shucked.wt 0.144    0.03     4.75    0      0.084  0.204
## 3 Viscera.wt 0.231    0.079    2.95    0.003  0.077  0.386
## 4 Shell.wt   0.42     0.047    8.97    0      0.328  0.512
```

Prediction

From the regression table on the previous slide, we see $\beta_0 = 0.331$, $\beta_1 = 0.144$, $\beta_2 = 0.231$, $\beta_3 = 0.420$.

Just like with simple linear regression, we can use `predict()` to compute a prediction based on the regression equation $y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$. The `newdata` argument should be a tibble with p appropriately named columns.

For instance, let's say we find two new abalones with the following weight measurements, and want to predict their lengths:

```
abalone_new <- tibble(Shucked.wt=c(0.5, 0.3),  
                     Viscera.wt=c(0.2, 0.2),  
                     Shell.wt=c(0.4, 0.35))
```

```
abalone_new
```

```
## # A tibble: 2 x 3  
##   Shucked.wt Viscera.wt Shell.wt  
##   <dbl>      <dbl>    <dbl>  
## 1     0.5      0.2      0.4  
## 2     0.3      0.2      0.35
```

Prediction

```
predict(object=model, newdata=abalone_new)
```

```
##           1           2  
## 0.6172920 0.5674962
```

We can manually verify the prediction for the first abalone:

```
0.331+0.144*0.5+0.231*0.2+0.420*0.4  
## [1] 0.6172
```

Coefficient interpretation

From the regression equation $y = \beta_0 + \beta_1 * x_1 + \dots + \beta_p * x_p$, we see that β_0 is the prediction for y when all predictors x_1, \dots, x_p are 0.

You want to be careful interpreting the other coefficients. The proper interpretation is that β_1 is the predicted change in y corresponding to a one-unit increase in x_1 , *keeping the other predictors fixed*.

It is important to include the “keeping the other predictors fixed” part, because the predictors themselves might be positively correlated! That is, an increase in x_1 might tend to correspond to an increase (or decrease) in x_2 , and/or the other predictors.

Categorical predictors

Categorical predictors

Last lecture, we saw that binary predictors can be used in simple linear regression, using a 0/1 encoding. They can also be included as such in a multiple linear regression.

Now, we will examine how general categorical predictors – which can take on an arbitrary (but finite) number of different values – can be used in (multiple or simple) linear regression.

Dummy variables

Suppose a categorical predictor x takes on K possible values, where K is some positive integer (at least 2).

The standard way to represent x for regression is to generate $K - 1$ different 0/1 **dummy variables**, which we label $x^{(1)}, \dots, x^{(K-1)}$.

We designate one possible value for x as the *control level*. If x is equal to the control level, then we encode $x^{(1)} = \dots = x^{(K-1)} = 0$. Otherwise, x is equal to one of the other $K - 1$ possible values, which is encoded by $x^{(j)} = 1$ (for some j in $1, \dots, K - 1$) and the other dummy variables equal to 0.

The regression equation for a *simple* linear regression of y onto x becomes $y = \beta_0 + \beta_{1,1} \cdot x^{(1)} + \dots + \beta_{1,K-1} \cdot x^{(K-1)}$, which is fit like a *multiple* linear regression of y onto $x^{(1)}, \dots, x^{(K-1)}$.

Dummy variables

A simple non-binary categorical predictor is the `origin` airport from the `flights` data (one of EWR, JFK, or LGA, so $K = 3$). If we fit a simple linear regression for `arr_delay` versus `origin`, R will automatically generate the 2 dummy variables for `origin`:

```
library(nycflights13)
lm(arr_delay ~ origin, data=flights) %>%
  get_regression_table()
```

A tibble: 3 x 7

##	term	estimate	std_error	statistic	p_value	lower_ci	upper_ci
##	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
## 1	intercept	9.11	0.13	69.9	0	8.85	9.36
## 2	origin: JFK	-3.56	0.188	-18.9	0	-3.92	-3.19
## 3	origin: LGA	-3.32	0.191	-17.4	0	-3.70	-2.95

Dummy variables

By default, R chooses the first level alphabetically (EWR) to be the control level. We see that R has generated two dummy variables: `origin: JFK` and `origin: LGA`.

`origin: JFK` is the dummy variable that is 1 when `origin = JFK` (and 0 otherwise). Likewise, `origin: LGA` is 1 when `origin = LGA` (and 0 otherwise). When `origin = EWR`, both dummy variables are 0.

The control level can be changed using `fct_relevel()`. The first level will be the control.

Prediction

What is the predicted `arr_delay` for a flight leaving from EWR?
How about JFK?

From the regression equation $y = 9.107 - 3.556 * x^{(1)} - 3.324 * x^{(2)}$, (here $x^{(1)}$ corresponds to `origin: JFK` and $x^{(2)}$ corresponds to `origin: LGA`), we see that the prediction for EWR must be the intercept, 9.107 (as EWR corresponds to $x^{(1)} = x^{(2)} = 0$).

Meanwhile, for JFK the prediction is $9.107 - 3.556 = 5.551$.

Prediction

Of course, we can also use the `predict()` function to avoid this manual computation:

```
model <- lm(arr_delay ~ origin, data=flights)
predict(model, newdata=tibble(origin="EWR"))
```

```
##           1
## 9.107055
```

```
predict(model, newdata=tibble(origin="JFK"))
```

```
##           1
## 5.551481
```

Interpretation

We can interpret the intercept as the predicted value for y when x equals its control level.

Meanwhile, $\beta_{1,j}$ is the predicted value for y when x equals level j , *minus* the predicted value for y when x equals its control level. Can you see why?

Categorical predictors in multiple linear regression

Finally, we note that categorical predictors can be included in a multiple linear regression, alongside one or more other predictors (quantitative and/or categorical)!

For instance, if x_1 is quantitative but x_2 is categorical with 4 levels, we have the regression equation

$$y = \beta_0 + \beta_1 \cdot x_1 + \beta_{2,1} \cdot x_2^{(1)} + \beta_{2,2} \cdot x_2^{(2)} + \beta_{2,3} \cdot x_2^{(3)}$$

where $x_2^{(1)}$, $x_2^{(2)}$, and $x_2^{(3)}$ are the dummy variables corresponding to x_2 .

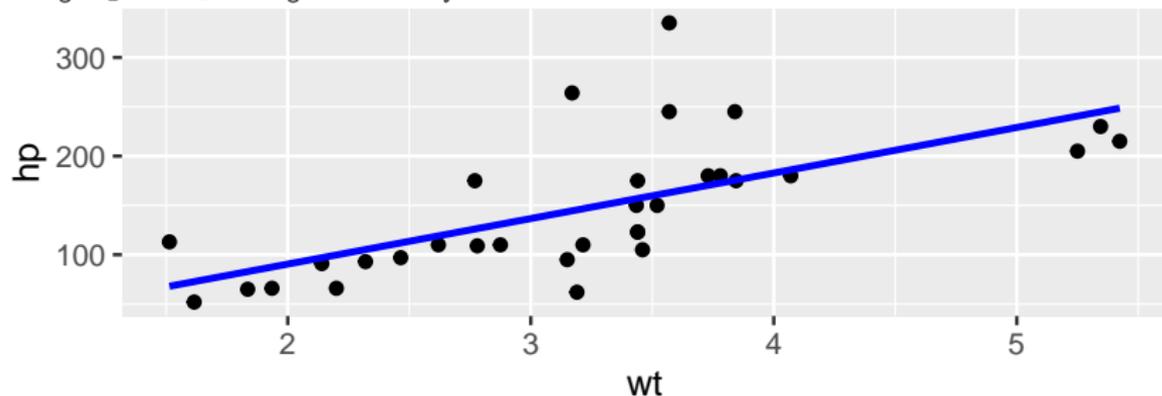
Polynomial regression

Polynomial regression

Let's look at a scatterplot of the horsepower `hp` versus the `wt` (weight) of various cars in the built-in `mtcars` dataset, along with the line from simple linear regression (recall from Lab 7 that `geom_smooth()` in `ggplot2` condenses the model fitting, prediction, and plotting steps into one):

```
mtcars %>%  
  ggplot(aes(x=wt, y=hp)) +  
  geom_point() +  
  geom_smooth(method="lm", color="blue", se=FALSE)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Quadratic regression

We observe a line may not fit the points above that well, since the increasing trend seems to “flatten” out for higher values of wt .

One solution is quadratic regression, which models y as a *quadratic* function of x (plus noise):

$$y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + e$$

We can fit this via a *multiple* regression of y onto the two predictors x and x^2 !

Quadratic regression in R

The formula to fit a quadratic regression in R will look like $y \sim x + I(x^2)$. Do NOT try $y \sim x + x^2$, which has unexpected behavior. That is, the `I()` wrapper around x^2 is critical.

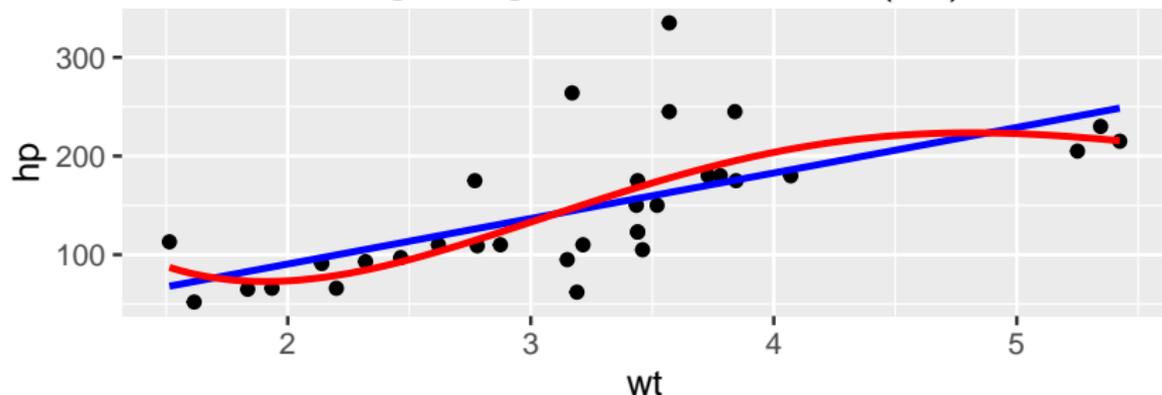
```
lm(formula=hp ~ wt+I(wt^2), data=mtcars) %>%  
  get_regression_table()  
  
## # A tibble: 3 x 7  
##   term      estimate std_error statistic p_value lower_ci upper_ci  
##   <chr>      <dbl>    <dbl>    <dbl> <dbl>    <dbl>    <dbl>  
## 1 intercept  -76.7     83.4     -0.92  0.365   -247.    93.8  
## 2 wt         93.8     49.8      1.88  0.07    -8.04   196.  
## 3 I(wt^2)    -6.94     7.12     -0.975 0.338   -21.5    7.62
```

The regression equation is $y = -76.734 + 93.765 \cdot x - 6.938 \cdot x^2$.

Polynomial regression

Of course, we can extend the idea of quadratic regression to polynomials of higher order, e.g. $y = \beta_0 + \beta_{1,1} \cdot x + \dots + \beta_{1,p} \cdot x^p$ for a polynomial of degree p , again by fitting a multiple regression with the p predictors x, x^2, \dots, x^p .

Here's what a 4-th degree regression fit looks like (red):



Locally weighted regression

Locally weighted regression

Sometimes, you may have a complicated-looking relationship between y and x that you want to fit. There may be no obvious polynomial relationship.

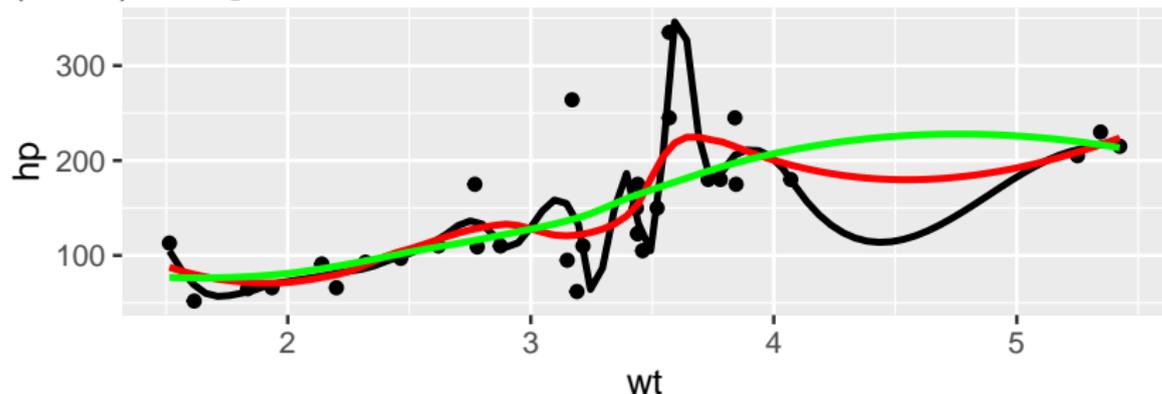
Luckily, there are still regression-based tools to deal with that! One popular method to fit a generic curve to your data is locally weighted regression, implemented using the built-in function `loess()`.

The interface for `loess()` is very similar to that of `lm()`. The big difference is that `loess()` does not give you a regression equation, as it does not assume a simple mathematical form for the relationship between y and the predictors. This allows it to do much more flexible “curve fitting”. However, you can still use `predict()`!

span

An important argument to `loess()` is the `span`, which controls the degree of “smoothing”, i.e. how “smooth” the fitted curve will look. `span` ranges from 0 to 1; higher values indicate a smoother curve.

Let’s look at `loess()` fits with `span` 0.2 (black), 0.5 (red), and 0.8 (green) for `hp` versus `wt`, as above:



span

There's no single correct way to choose span. Choosing span too low will make the curve fit the data too closely, which could cause your curve to start fitting the “noise” in the data rather than the actual trend.

On the other hand, choosing span too high could cause the curve to miss important patterns in the data.

Next week, we will explore the concept of *cross validation* to provide a data-driven way to choose model parameters like span.