Splines, model complexity, and prediction error

September 15, 2022

Fitting, plotting, and predicting with splines

Let's first see how some of the plots from the lecture slides were generated. These plots are based on the `Wage` data from the `ISLR2` package, a subset of which is given in `income.csv`.

```
income <- read_csv("income_data.csv")
income
```

```
## # A tibble: 2,920 x 3
##      age income  year
##    <dbl>  <dbl> <dbl>
##  1    18   75.0  2006
##  2    24   70.5  2004
##  3    45  131.   2003
##  4    43  155.   2003
##  5    50   75.0  2005
##  6    54  127.   2008
##  7    44  170.   2009
##  8    30  112.   2008
##  9    41  119.   2006
## 10    52  129.   2004
## # ... with 2,910 more rows
## # i Use `print(n = ...)` to see more rows
```

Fitting linear regression models

Let's focus on the data from 2007:

```
income_2007 <- income %>% filter(year == 2007) %>% select(-year)
income_2007
```

```
## # A tibble: 376 x 2
##      age income
##    <dbl>  <dbl>
##  1    45  117.
##  2    34   81.3
##  3    56  129.
##  4    40  161.
##  5    27  110.
##  6    57  177.
##  7    42   85.4
##  8    41   88.0
##  9    38  129.
## 10    51   94.1
## # ... with 366 more rows
## # i Use `print(n = ...)` to see more rows
```

We fit linear models using the `lm()` function (built into base R). We specify a *formula object* as well as the dataset to use. For example:

```
lm(income ~ age, data = income_2007)
```

```
##
## Call:
## lm(formula = income ~ age, data = income_2007)
##
## Coefficients:
## (Intercept)          age
##     83.4995       0.5765
```

Note that, by default, the intercept is included in addition to the predictors listed in the formula object. If we want the intercept-only model, we can use the special notation `1`:

```
lm(income ~ 1, data = income_2007)
```

```
##
## Call:
## lm(formula = income ~ 1, data = income_2007)
##
## Coefficients:
## (Intercept)
##       108.2
```

If we want a model *without* intercept, we can write

```
lm(income ~ age - 1, data = income_2007)
```

```
##
## Call:
## lm(formula = income ~ age - 1, data = income_2007)
##
## Coefficients:
##   age
## 2.396
```

Polynomial and spline models are linear models as well, so they can be computed via `lm()`. We just need the special functions `poly()` (built into base R) and `ns()` (in the `splines` package). For polynomials, we just need to specify the degree:

```
lm(income ~ poly(age, degree = 2), data = income_2007)
```

```
##
## Call:
## lm(formula = income ~ poly(age, degree = 2), data = income_2007)
##
## Coefficients:
##            (Intercept)  poly(age, degree = 2)1  poly(age, degree = 2)2
##                  108.2                   127.7                  -151.0
```

For natural splines, we just need to specify the degrees of freedom (the number and location of knots are chosen automatically):

```
lm(income ~ ns(age, df = 5), data = income_2007)
```

```
##
## Call:
## lm(formula = income ~ ns(age, df = 5), data = income_2007)
```

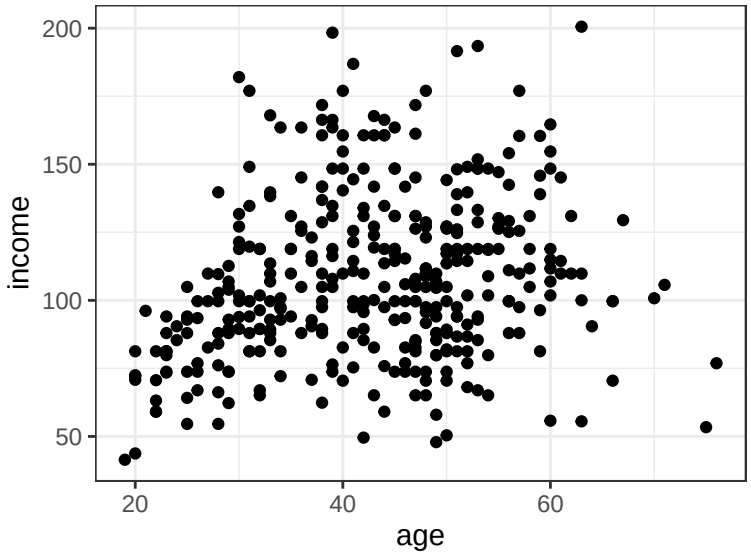
```
##
## Coefficients:
##      (Intercept)  ns(age, df = 5)1  ns(age, df = 5)2  ns(age, df = 5)3
##           65.348            69.939            29.958            59.448
## ns(age, df = 5)4  ns(age, df = 5)5
##           67.113            -8.112
```

Careful! Note that `df = 5` *excludes the intercept.* So there are a total of 6 degrees of freedom in this fit.

Plotting

Let's first plot the data:

```
income_2007 %>%
  ggplot(aes(x = age, y = income)) +
  geom_point()
```

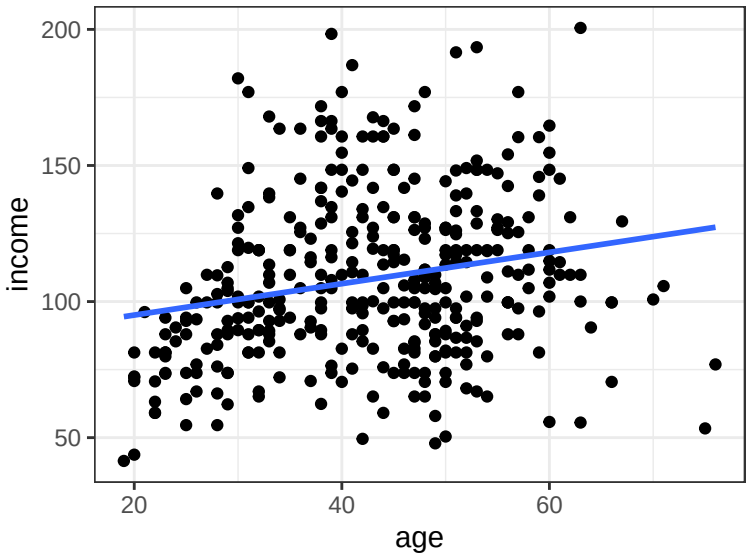


Ok, so how do we plot those fitted curves? We can use another geom called `geom_smooth()`. The relevant arguments are

- `method` (we will usually use `lm`)
- `formula` (specifying the linear regression formula)
- `se` (whether to draw error bars; we will usually use `FALSE`).

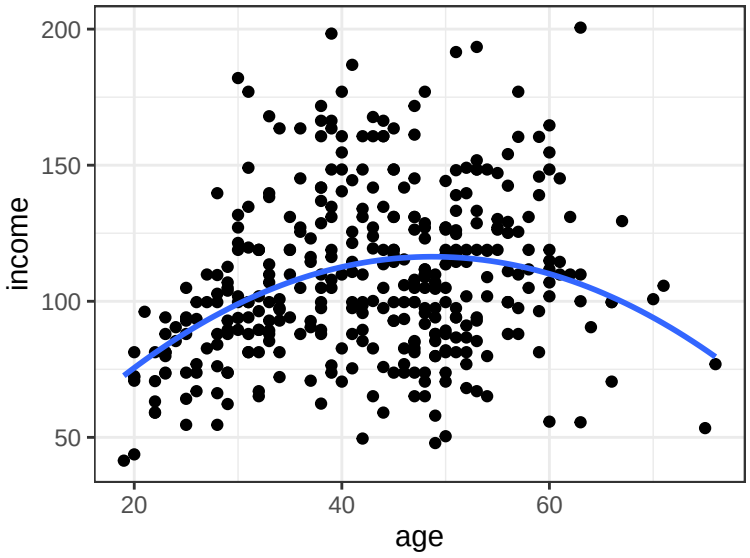
Let's take some examples. Here is a linear model fit:

```
income_2007 %>%
  ggplot(aes(x = age, y = income)) +
  geom_point() +
  geom_smooth(method = "lm",
              formula = "y ~ x", # NOTE: use y and x instead of income and age!
              se = FALSE)
```



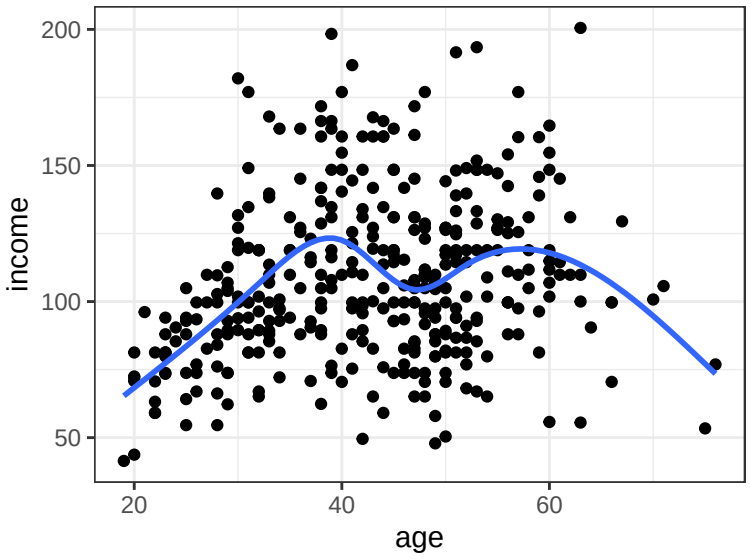
Here is a polynomial fit:

```
income_2007 %>%
  ggplot(aes(x = age, y = income)) +
  geom_point() +
  geom_smooth(method = "lm",
              formula = "y ~ poly(x,2)", # NOTE: use y and x instead of income and age!
              se = FALSE)
```



Here is a natural cubic spline fit:

```
income_2007 %>%
  ggplot(aes(x = age, y = income)) +
  geom_point() +
  geom_smooth(method = "lm",
              formula = "y ~ ns(x, df = 5)", # NOTE: use y and x instead of income and age!
              se = FALSE)
```



Prediction

In order to make predictions for a test dataset (e.g. the data from the year 2008), we need to first save the fit object, and then use the `predict()` function:

```
income_2008 <- income %>% filter(year == 2008) %>% select(-year)
spline_fit <- lm(income ~ ns(age, df = 5), data = income_2007)
predicted_incomes_2008 <- predict(spline_fit, newdata = income_2008)
head(predicted_incomes_2008)
```

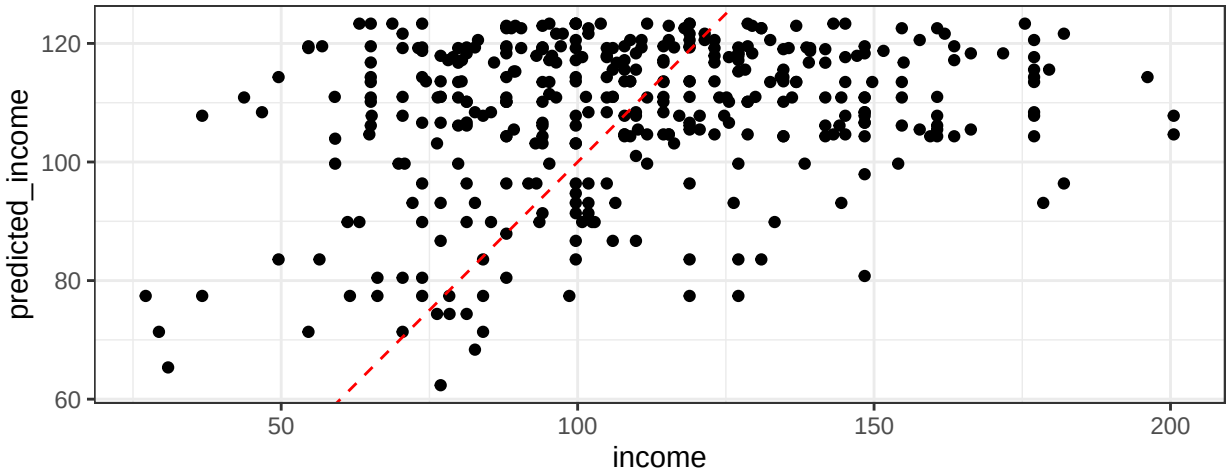
```
##         1         2         3         4         5         6
## 117.17964  99.72272 119.32551 110.16106 113.49437  91.36378
```

```
length(predicted_incomes_2008)
```

```
## [1] 377
```

Let's take a look at how well the predictions match the true incomes for 2008:

```
income_2008 %>%
  mutate(predicted_income = predicted_incomes_2008) %>%
  ggplot(aes(x = income, y = predicted_income)) +
  geom_point() +
  geom_abline(slope = 1, colour = "red", linetype = "dashed") +
  coord_fixed()
```



To quantify how far the predictions are from the truth, we can compute the root mean squared test error:

```
actual_incomes_2008 <- income_2008$income
sqrt(mean((actual_incomes_2008-predicted_incomes_2008)^2))
```

```
## [1] 31.6266
```

By contrast, let's take a look at the RMS training error:

```
actual_incomes_2007 <- income_2007$income
fitted_incomes_2007 <- spline_fit$fitted.values
sqrt(mean((fitted_incomes_2007-actual_incomes_2007)^2))
```

```
## [1] 27.51901
```

Now, let's calculate the training and test errors for `df` = 1,2,3,4,5:

```
num_df_values <- 5
df_values <- 1:num_df_values
train_errors <- numeric(num_df_values)
test_errors <- numeric(num_df_values)
for(i in 1:length(df_values)){
  df <- df_values[i]
  lm_fit <- lm(income ~ ns(age, df = df), data = income_2007)
  fitted_incomes_2007 <- lm_fit$fitted.values
  predicted_incomes_2008 <- predict(lm_fit, newdata = income_2008)
  actual_incomes_2007 <- income_2007 %>% pull(income)
  actual_incomes_2008 <- income_2008 %>% pull(income)
  train_errors[i] <- sqrt(mean((fitted_incomes_2007 - actual_incomes_2007)^2))
  test_errors[i] <- sqrt(mean((predicted_incomes_2008 - actual_incomes_2008)^2))
}
```

Let's take a look at these errors:

```
train_errors
```

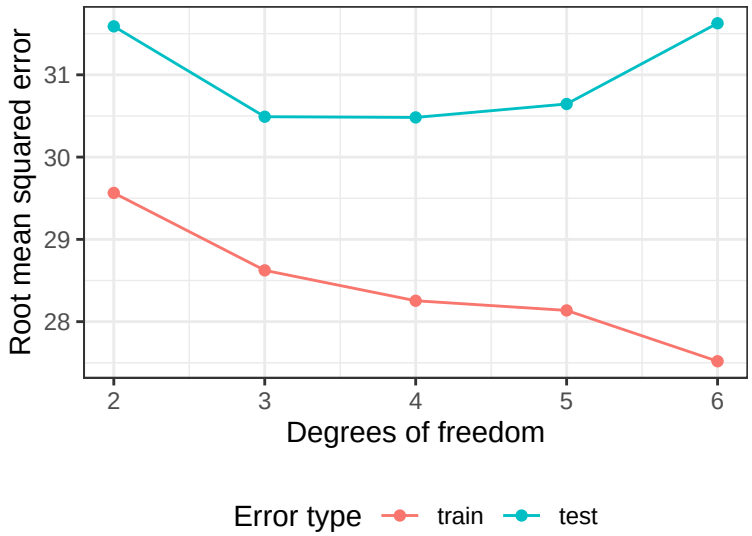
```
## [1] 29.56470 28.62369 28.25417 28.13682 27.51901
```

```
test_errors
```

```
## [1] 31.58897 30.49050 30.48212 30.64534 31.62660
```

Now let's plot them:

```
tibble(df = df_values+1,
       error_test = test_errors,
       error_train = train_errors) %>%
  pivot_longer(-df,
               names_to = "Error type",
               names_prefix = "error_",
               values_to = "error") %>%
  mutate(`Error type` = factor(`Error type`, levels = c("train", "test"))) %>%
  ggplot(aes(x = df, y = error, colour = `Error type`)) +
  geom_point() + geom_line() +
  labs(x = "Degrees of freedom",
       y = "Root mean squared error") +
  theme(legend.position = "bottom")
```



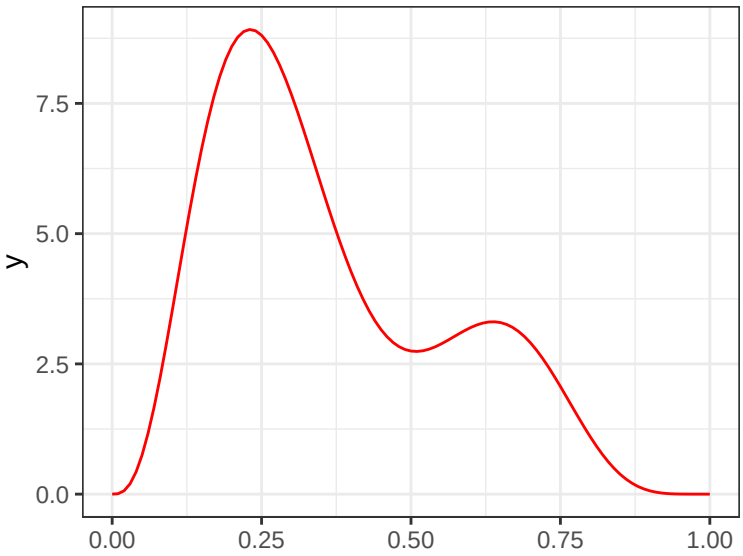
Exercise: Exploring model complexity in a simulation

In the `income` data, we don't know what the "true" trend is. It may be illuminating, therefore, to replace this data with *simulated* data (i.e. data we generate ourselves). Suppose that $Y = f(X) + \epsilon$, and let us assume f take the following form:

```
f <- function(x){
  0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
}
```

It's hard to understand what this function is but we can plot it:

```
ggplot() +
  stat_function(fun = f, colour = "red")
```



Now, let's create 500 training data points (X, Y) as follows. First, we generate 500 equally spaced values of `x` between 0 and 1.

```
num_observations <- 500
x <- seq(from = 0, to = 1, length.out = num_observations)
```

Then we generate the response vector `y` based on the formula above:

```
y <- f(x) + rnorm(n = num_observations, mean = 0, sd = 2)
```

Finally, we put `x` and `y` into a tibble called `data_train`:

```
data_train <- tibble(x = x, y = y)
data_train
```

```
## # A tibble: 500 x 2
##          x        y
##      <dbl>    <dbl>
##  1 0       -1.30
##  2 0.00200  2.10
##  3 0.00401  0.00726
##  4 0.00601  1.15
##  5 0.00802  0.924
##  6 0.0100  -0.309
##  7 0.0120   0.144
##  8 0.0140   1.29
##  9 0.0160   0.243
## 10 0.0180  -3.61
## # ... with 490 more rows
## # i Use `print(n = ...)` to see more rows
```

1. Create a scatter plot of these data, overlaying the function `f` in red.
2. Take a look at what natural spline fits look like for this data, using different degrees of freedom. Plot natural spline fits with degrees of freedom (excluding intercept) equal to 2,10,25,50 (use separate plots for each value of df). For each of these plots, superimpose the true trend as a red line. Comment on what happens as you increase the degrees of freedom. Which of these four values seems to fit the scatter plot best?
3. Create a test data set (named `data_test`) with 500 additional points from the same distribution and

compute the root mean square test error for each of the four spline fits above. Does the degrees of freedom that gives the closest fit to the underlying trend also lead to the lowest test error?