Appendix R8: Non-Linear Models

Interaction, Polynomial, Piecewise and Spline Models

J. Alberto Espinosa

3/2/2023

Contents

```
RNGkind(sample.kind = "default") # To use the R default RNG
```

Non-Linearity

This appendix provides a number of modeling alternatives to use when the OLS assumption of linearity does not hold. This can be evaluated visually with plots and also tested with ANOVA or CV tests to evaluate whether some non-linear models have more predictive accuracy that the corresponding linear model. In this section, I discuss **interaction**, **polynomial**, **piecewise splines** and **MARS (spline)** models.

Binary x Continuous (B x C) Interaction Models

Overview

An interaction model contains at least 2 predictors and a multiplicative term between the two predictors. The regression coefficients for the 2 predictors are called **main effects** and the regression coefficient for the **multiplicative** term is called the **interaction effect**. These models are necessary when business intuition suggests that the effect of one predictor is influenced by the presence of the other predictor. Interaction effects cannot be interpreted by themselves, but they need to be interpreted jointly with their respective main effects. The interpretation of effects in an interaction model can be done from the perspective of either of the main predictors. If you have an interaction model like `y ~ x1 + x2 + x1 * x2`, the total effect of `x1` will be equal to the **main effect** of `x1` plus the interaction effect of `x1 * x2`. That is, the effect of `x1` will be influenced by the presence of `x2`. If the sign of the main effect of `x1` is the same as the sign of the interaction term, the interaction effect of `x2` on `x1` is **positive** because the interaction effect **enhances** the effect of `x1`, regardless of whether these effects are positive or negative. The fact that the main and interaction effects have the same signs means that the main effect is enhanced by the interaction. Conversely, if the sign of the main effect of `x1` and the interaction effect have opposite sign, then the interaction is **negative** because it **offsets** the main effect. The fact that the main and interaction effects have opposite signs means that the main effect is diminished or offset by the interaction. Similarly, if you look at the interaction from the perspective of `x2`, its main effect is either enhanced or offset by the presence of `x1` in the interaction term.

Binary x Continuous (B x C) Interactions

Essentially, the interaction term changes the slope of the main effect. When both variables are continuous, the total effect of `x1` changes for every value of `x2`, which makes **Continuous x Continuous (C x C)** interactions more difficult to interpret and require additional modeling transformations. I provide some guidance of how to build C x C interactions below and in the main chapter for this appendix.

In this section, I focus on **B x C** interactions for two main reasons. First, **B x C** interactions are very popular, widely used and very useful to help understand effect variations across categories. Second, understanding how to specify and interpret B x C interactions is a necessary precondition to understand **C x C** interactions. **B x C** interactions have a simple and intuitive interpretation because the interaction term measures how much the slope of the main effect of the C variable changes when the B variable changes from 0 to 1. Think of it as 2 regression lines, just like with the dummy variable example discussed in Chapter 3, except that the regression lines are no longer parallel, and it is the interaction term that causes the slope or effect of the regression line to change.

Let's first load the **{ISLR}** library and do some setup and data transformations with the **Auto** data set. We will use this data set to predict the gas mileage performance in vehicles

```r
library(ISLR) # Contains the Auto data set

options(scipen = 4)
```

The **Auto** data set has an **origin** variable that takes the value of 1 for American vehicles and 2 and 3 for European and Japanese respectively. To illustrate the interaction model, I will convert this variable into a **foreign** variable with the value of 0 for American vehicles and 1 for all other vehicles.

```r
Auto$foreign <- ifelse(Auto$origin == 1, 0, 1)
```

Let’s start with a plain **linear model** without interaction terms.

```r
fit.linear <- lm(mpg ~ horsepower + weight + foreign + year,
                 data = Auto)

summary(fit.linear)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower + weight + foreign + year, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.4008 -2.1335 -0.0628  1.8284 13.6148
##
## Coefficients:
##                Estimate  Std. Error t value   Pr(>|t|)
## (Intercept) -17.5775640   4.1432258  -4.242 0.00002769 ***
## horsepower   -0.0075412   0.0091924  -0.820      0.413
## weight       -0.0056080   0.0004338 -12.928    < 2e-16 ***
## foreign       2.1155598   0.4369972   4.841 0.00000187 ***
## year          0.7596272   0.0507256  14.975    < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.335 on 387 degrees of freedom
## Multiple R-squared:  0.8193, Adjusted R-squared:  0.8174
## F-statistic: 438.6 on 4 and 387 DF,  p-value: < 2.2e-16
```

The regression output indicates that, other things being equal, each additional pound of vehicle weight is associated a decrease of 0.0056 miles per gallon. Also, on average, foreign vehicles are more fuel efficient, yielding 2.115 more miles per gallon than domestic vehicles.

Let’s add one B x C interaction term **foreign x weight** to evaluate if the effect of weight varies by vehicle origin.

```r
fit.inter.1 <- lm(mpg ~ horsepower + weight + foreign + year +
                        foreign * weight,
                        data = Auto)
```

R is smart and knows that interaction effects should be interpreted jointly with the main effects, so if you omit the main effects of **weight** and **foreign**, R will include them because they are in the interaction term, which simplifies the model specification. This model will yield the same results as above, but if for some strange reason you want to exclude the main effects you can use `foreign : weight` instead of `foreign * weight`.

```
fit.inter.1 <- lm(mpg ~ horsepower + year +
                        foreign * weight,
                        data = Auto)

summary(fit.inter.1)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower + year + foreign * weight, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.7988 -2.0742 -0.1083  1.6348 12.8584
##
## Coefficients:
##                  Estimate  Std. Error t value Pr(>|t|)
## (Intercept)    -21.7924516   4.1018525  -5.313 1.83e-07 ***
## horsepower      -0.0119039   0.0089534  -1.330    0.184
## year             0.7896218   0.0495323  15.942  < 2e-16 ***
## foreign         10.9062409   1.7838010   6.114 2.38e-09 ***
## weight          -0.0048765   0.0004446 -10.969  < 2e-16 ***
## foreign:weight  -0.0035444   0.0006987  -5.073 6.09e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.234 on 386 degrees of freedom
## Multiple R-squared:  0.8306, Adjusted R-squared:  0.8284
## F-statistic: 378.4 on 5 and 386 DF,  p-value: < 2.2e-16
```

In the output above the item `foreign:weight` is the interaction term. The output shows that, on average, for each additional pound of weight, the miles per gallon performance decreases by 0.0049. The `foreign:weigth` coefficient has the same sign, indicating that the effect of weight is stronger for foreign vehicles than domestic vehicles, by 0.0035. In other words, the negative effect of weight is stronger for foreign vehicles (- 0.0049 - 0.0035 = - 0.0084) than for domestic vehicles (- 0.0049). Naturally, you can interpret the interaction effect from the perspective of the **foreign** predictor. In simple terms, foreign vehicles have a higher gas mileage, but this effect is offset as vehicles get heavier.

Let's add another B x C interaction term **foreign x year**

```
fit.inter.2 <- lm(mpg ~ horsepower + weight + foreign + year +
                        foreign * weight +
                        foreign * year,
                        data = Auto)

summary(fit.inter.2)
```

```
##
## Call:
## lm(formula = mpg ~ horsepower + weight + foreign + year + foreign *
##     weight + foreign * year, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8995 -1.9335 -0.0871  1.5693 12.0042
##
## Coefficients:
##                    Estimate  Std. Error t value   Pr(>|t|)
## (Intercept)      -7.2627664   5.1818018  -1.402    0.16184
## horsepower       -0.0153070   0.0087795  -1.743    0.08205 .
## weight           -0.0050669   0.0004364 -11.611    < 2e-16 ***
## foreign         -21.6332954   7.5624162  -2.861    0.00446 **
## year              0.6112655   0.0629915   9.704    < 2e-16 ***
## weight:foreign   -0.0033460   0.0006839  -4.892 0.00000147 ***
## foreign:year      0.4167118   0.0942411   4.422 0.00001275 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.158 on 385 degrees of freedom
## Multiple R-squared:  0.8388, Adjusted R-squared:  0.8362
## F-statistic: 333.8 on 6 and 385 DF,  p-value: < 2.2e-16
```

The effect of **weight** is still negative, but now the effect of **foreign** is negative and quite large. Let's analyze what is happening. The interaction effect of **weight x foreign** is still negative, so the negative effect of weight is stronger for foreign vehicles, like in the model above. On average, foreign vehicles have lower gas mileage performance and this effect is enhanced for heavier vehicles.

Now, let's analyze the other interaction term. As one would expect, newer vehicles have better gas mileage, that is 0.6113 more miles per gallon for each model year. This effect is enhanced by 0.4167 for foreign vehicles. That is newer vehicle models are more fuel efficient and more so for foreign vehicles. Looking at it from the perspective of vehicle origin, foreign vehicles have a lower gas mileage, but not so much with newer vehicles.

Now let's do some ANOVA testing to see which model has more predictive power.

```
anova(fit.linear, fit.inter.1, fit.inter.2)
```

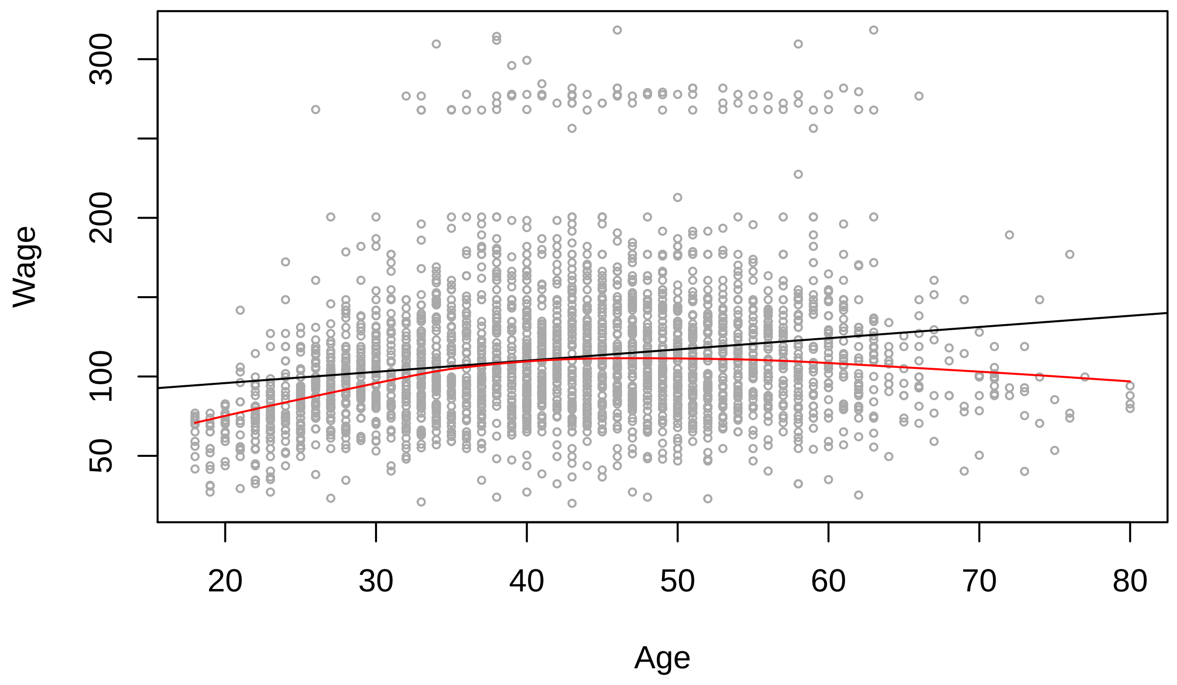
```
## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower + weight + foreign + year
## Model 2: mpg ~ horsepower + year + foreign * weight
## Model 3: mpg ~ horsepower + weight + foreign + year + foreign * weight +
##     foreign * year
##   Res.Df    RSS Df Sum of Sq      F       Pr(>F)
## 1    387 4304.9
## 2    386 4035.8  1    269.10 26.975 0.0000003348 ***
## 3    385 3840.8  1    195.05 19.552 0.0000127533 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Both ANOVA p-values are highly significant, so the model with one interaction term is superior to the linear model, but the model with two interaction terms is even better.

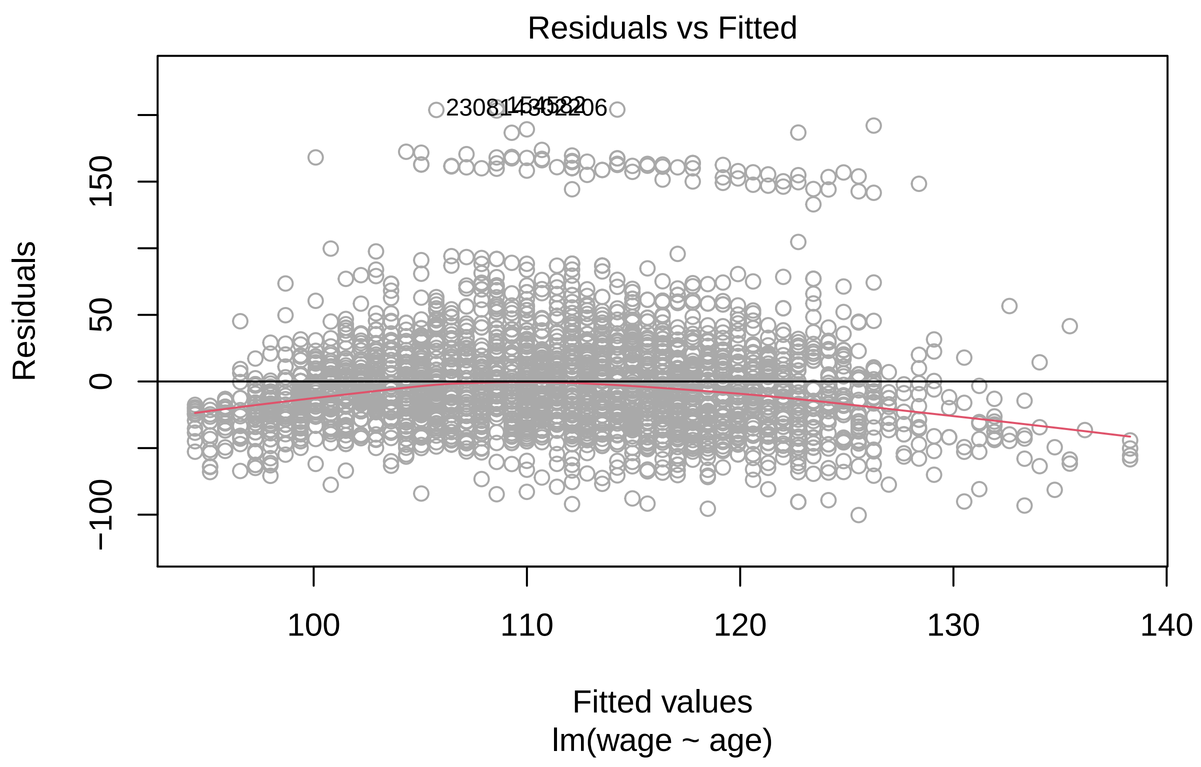
Polynomial Regression

A first step before considering a polynomial model is to plot the data. In this case, we are using the **Wage** data set from the **{ISLR}** library. I first render a bare scatterplot and I then use the `lines()` and `lowess()` functions to draw a trend line. The lowess function performs a **locally weighted smoothing**, which is like dividing the data into small segments and fitting mini-regressions in each, and then use the fitted values to draw a "smooth" line on the plot:

```
library(ISLR)

lm.fit <- lm(wage ~ age, data = Wage) # Linear Model

plot(Wage$age, Wage$wage,
     xlab = "Age",
     ylab = "Wage",
     cex = 0.5,
     col = "darkgrey")

abline(lm.fit) # Straight line

lines(lowess(Wage$age, Wage$wage),
      col = "red") # Trend curve
```



```
plot(lm.fit, which = 1,
     col = "darkgrey") # Residual Plot

abline(h = 0) # Draw a horizontal line at 0
```



You can change the smoothing on the line with the parameter **f**, which is the proportion of data points in each segment. The default is **f = 2/3**. Try a very small value (e.g., 0.01) for little smoothing (i.e., a jagged line) and a large one (e.g. 0.90) for a lot of smoothing (1 give a straight line).

A casual inspection of the plot suggests that the relationship between wage and age is not linear

Let's start simple, with a linear model

```
fit.poly.1 <- lm(wage ~ age,
                 data = Wage)

summary(fit.poly.1)
```

```
##
## Call:
## lm(formula = wage ~ age, data = Wage)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -100.265  -25.115   -6.063   16.601  205.748
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 81.70474    2.84624   28.71   <2e-16 ***
## age          0.70728    0.06475   10.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 40.93 on 2998 degrees of freedom
## Multiple R-squared:  0.03827,    Adjusted R-squared:  0.03795
## F-statistic: 119.3 on 1 and 2998 DF,  p-value: < 2.2e-16
```

The linear model has a good fit, although with a veyr low R-Square. The output shows that wage increases with age. For each additional year of age, the wage goes up by $707 (wages are in thousands).

Let's try a quadratic model

```
fit.poly.2 <- lm(wage ~ poly(age, 2, raw = T),
                 data = Wage)

summary(fit.poly.2)
```

```
##
## Call:
## lm(formula = wage ~ poly(age, 2, raw = T), data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.126 -24.309  -5.017  15.494 205.621
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -10.425224   8.189780  -1.273    0.203
## poly(age, 2, raw = T)1   5.294030   0.388689  13.620   <2e-16 ***
## poly(age, 2, raw = T)2  -0.053005   0.004432 -11.960   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.99 on 2997 degrees of freedom
## Multiple R-squared:  0.08209,    Adjusted R-squared:  0.08147
## F-statistic:   134 on 2 and 2997 DF,  p-value: < 2.2e-16
```

This model also has a good fit with a highly significant p-value for the full model. The R-Squared increased a small amount. Notice that I used the attribute `raw = T` to obtain the raw, rather than the orthogonal coefficients. The linear effect is positive and significant, and the quadratic term is negative and also highly significant.This means that salaries start going up at the rate of $5,294 for each additional year of age, but the negative effect of age squared makes this rate diminishing as age goes up, reaching a peak at - 5,294 / (2 * (- 53)) = 49.9 or almost 50 years of age, then diminishing after that.

Let's fit a cubic polynomial

```r
fit.poly.3 <- lm(wage ~ poly(age, 3, raw = T),
                 data = Wage)

summary(fit.poly.3)
```

```
##
## Call:
## lm(formula = wage ~ poly(age, 3, raw = T), data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.693 -24.562  -5.222  15.096 206.119
##
## Coefficients:
##                           Estimate  Std. Error t value Pr(>|t|)
## (Intercept)            -75.2439142  22.1837300  -3.392 0.000703 ***
## poly(age, 3, raw = T)1  10.1899915   1.6052282   6.348 2.51e-10 ***
## poly(age, 3, raw = T)2  -0.1680286   0.0368602  -4.559 5.36e-06 ***
## poly(age, 3, raw = T)3   0.0008495   0.0002702   3.143 0.001687 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.93 on 2996 degrees of freedom
## Multiple R-squared:  0.0851, Adjusted R-squared:  0.08419
## F-statistic: 92.89 on 3 and 2996 DF,  p-value: < 2.2e-16
```

The linear and quadratic effects in the cubic polynomial model are similar than those of the quadratic model, except that the linear and quadratic effects are larger, but the cubic term is also positive and significant, indicating that when the wage goes up, the cubic effect starts pulling the curve upwards, although the cubic effect is too small to observe a valley in the present data range.

Let's evaluate the 3 models with an ANOVA test

```r
anova(fit.poly.1, fit.poly.2, fit.poly.3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ age
## Model 2: wage ~ poly(age, 2, raw = T)
## Model 3: wage ~ poly(age, 3, raw = T)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786 143.4679 < 2.2e-16 ***
## 3   2996 4777674  1     15756   9.8801  0.001687 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first ANOVA test is highly significant, indicating that the quadratic model has much higher predictive power. The second ANOVA test is also significant, but not as much. Nevertheless, the cubic model has the highest predictive power. You can keep trying higher polynomials and testing them with ANOVA, but a better test would be to evaluate the models with the CV test error.

A note about **raw polynomials**: by default, `raw = F`, which causes the poly() to create **orthogonal polynomial** terms. These orthogonal polynomials are simply the principal components of the raw polynomial variables, which is meant to help reduce multi-collinearity. Setting `raw = T` computes back the raw coefficients, which are what we need for interpretation.

Alternatively, you could fit the same model above the long way with the I() (identity) function and get identical results, but with a lot more hand coding:

```
fit.poly.3I <- lm(wage ~ age + I(age ^ 2) + I(age ^ 3),
                  data = Wage)

summary(fit.poly.3I)
```

```
##
## Call:
## lm(formula = wage ~ age + I(age^2) + I(age^3), data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.693 -24.562  -5.222  15.096 206.119
##
## Coefficients:
##               Estimate  Std. Error t value Pr(>|t|)
## (Intercept) -75.2439142  22.1837300  -3.392 0.000703 ***
## age          10.1899915   1.6052282   6.348 2.51e-10 ***
## I(age^2)     -0.1680286   0.0368602  -4.559 5.36e-06 ***
## I(age^3)      0.0008495   0.0002702   3.143 0.001687 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.93 on 2996 degrees of freedom
## Multiple R-squared:  0.0851, Adjusted R-squared:  0.08419
## F-statistic: 92.89 on 3 and 2996 DF,  p-value: < 2.2e-16
```

Predictions and CV Testing with Polynomials

You can use the `cv.glm()` function to compute LOOCV, KFCV or any other CV test errors. In this section, I will use random subset sampling to do quick tests and compare the three polynomials, and also to illustrate how to do predictions.

```
RNGkind(sample.kind = "default") # R default RNG
set.seed(1) # Set any random seed
```

Let's construct a 80% train and 20% test subsets and make predictions

```r
train <- sample(nrow(Wage),
                0.8 * nrow(Wage))

Wage.train <- Wage[train, ] # Train subset
nrow(Wage.train) # 2,400 observations
```

```
## [1] 2400
```

```r
Wage.test <- Wage[-train, ] # Test subset
nrow(Wage.test) # 600 observations
```

```
## [1] 600
```

Now let's train, predict and test the 3 polynomial models

```r
# Train the models

fit.poly.1.train <- lm(wage ~ age,
                       data = Wage.train)

fit.poly.2.train <- lm(wage ~ poly(age, 2, raw = T),
                       data = Wage.train)

fit.poly.3.train <- lm(wage ~ poly(age, 3, raw = T),
                       data = Wage.train)

# Predict with test data with each of the 3 trained models

wage.pred.1 <- predict(fit.poly.1.train, Wage.test)
wage.pred.2 <- predict(fit.poly.2.train, Wage.test)
wage.pred.3 <- predict(fit.poly.3.train, Wage.test)

# Compute the MSE for each

mse.test.1 <- mean( (Wage.test$wage - wage.pred.1) ^ 2)
mse.test.2 <- mean( (Wage.test$wage - wage.pred.2) ^ 2)
mse.test.3 <- mean( (Wage.test$wage - wage.pred.3) ^ 2)

cbind("CV MSE Linear" = mse.test.1,
      "CV MSE Quadratic" = mse.test.2,
      "CV MSE Cubic" = mse.test.3)
```

```
##      CV MSE Linear CV MSE Quadratic CV MSE Cubic
## [1,]      2068.485         1989.221     1986.052
```

The CV test MSE yield very similar results to the ANOVA tests we did earlier. The quadratic model is superior to the linear model, and the cubic model is slightly better than the quadratic model.

Step Functions

A step function fits a bunch of horizontal lines in different **segments** of the data. The points where the partitions begin and end are the **knots**. To fit a Step function, we can use the **cut()** function to arbitrarily divide the age data into, say 4 partitions.

```
fit.step <- lm(wage ~ cut(age, 4),
               data = Wage)

summary(fit.step) # Take a look (coefficients are relative to intercept)
```

```
##
## Call:
## lm(formula = wage ~ cut(age, 4), data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -98.126 -24.803  -6.177  16.493 200.519
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)              94.158      1.476  63.790   <2e-16 ***
## cut(age, 4)(33.5,49]     24.053      1.829  13.148   <2e-16 ***
## cut(age, 4)(49,64.5]     23.665      2.068  11.443   <2e-16 ***
## cut(age, 4)(64.5,80.1]    7.641      4.987   1.532    0.126
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 40.42 on 2996 degrees of freedom
## Multiple R-squared:  0.0625, Adjusted R-squared:  0.06156
## F-statistic: 66.58 on 3 and 2996 DF,  p-value: < 2.2e-16
```

Let's do predictions and CV testing on this model

```
fit.step.train <- lm(wage ~ cut(age, 4),
                     data = Wage.train)

pred.step <- predict(fit.step.train, Wage.test)

mse.step <- mean( (Wage.test$wage - pred.step) ^ 2)
mse.step # Take a look
```

```
## [1] 2021.074
```

The Step function has more predictive accuracy than the linear model, but less accuracy than the quadratic or the cubic models. Makes sense.

Piecewise Linear Spline Regression

Let's use the same segments discussed in the main chapter. Note that I use the `I()` function. This is the **Identity** function, which basically doesn't do anything and doesn't change anything, but it is useful because sometimes the operations inside the functions can have other meaning in R. The identity function simply says "do this". The `I()` function creates the respective dummy variables and interaction terms. The term (age - 25) is equivalent to moving the Y axis to the knot at `age = 25` and then gets multiplied by `(age > 25)`, which will yield **0** for `age <= 25` and **1** `age > 25`.

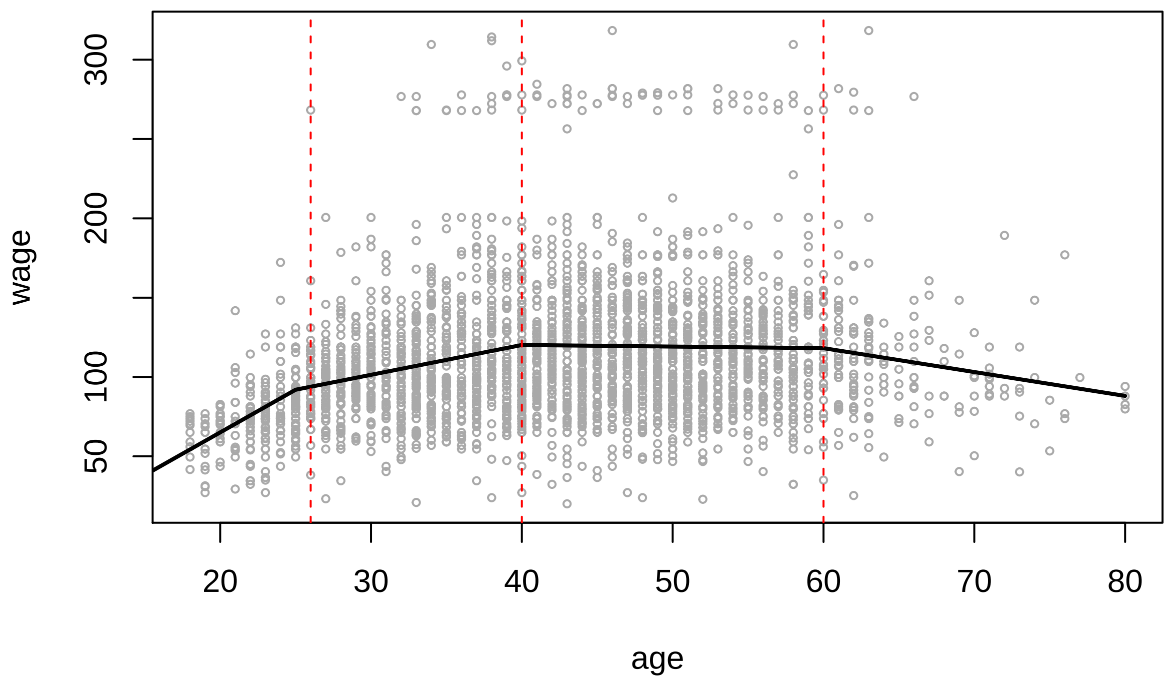
```
fit.piecewise <- lm(wage ~ age +
                           I((age - 25) * (age > 25)) +
                           I((age - 40) * (age > 40)) +
                           I((age - 60) * (age > 60)),
                    data = Wage)

summary(fit.piecewise) # Take a look
```

```
##
## Call:
## lm(formula = wage ~ age + I((age - 25) * (age > 25)) + I((age -
##     40) * (age > 40)) + I((age - 60) * (age > 60)), data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.795 -24.686  -4.856  15.344 204.671
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -42.4688    23.2695  -1.825  0.06809 .
## age                          5.3779     0.9739   5.522 3.64e-08 ***
## I((age - 25) * (age > 25))  -3.4977     1.0800  -3.239  0.00121 **
## I((age - 40) * (age > 40))  -1.9800     0.2970  -6.667 3.10e-11 ***
## I((age - 60) * (age > 60))  -1.4041     0.5424  -2.589  0.00968 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.91 on 2995 degrees of freedom
## Multiple R-squared:  0.08665,    Adjusted R-squared:  0.08543
## F-statistic: 71.03 on 4 and 2995 DF,  p-value: < 2.2e-16
```

The results indicate that the linear regression line starts with `$5.38K` more wage dollars for every additional year of age, up to age **25**. At that point the slope changes to `5.38 - 3.50 = 1.88K` higher wage per year of age, up to age **40**. It then changes to `1.88 - 1.98 = - 0.10K` lower wage for every additional year of age (i.e., almost flat), up to age **60**. After that, the wage declines at the rate of `- 0.10 - 1.40 = -1.50K` fewer dollars for each additional year of age.

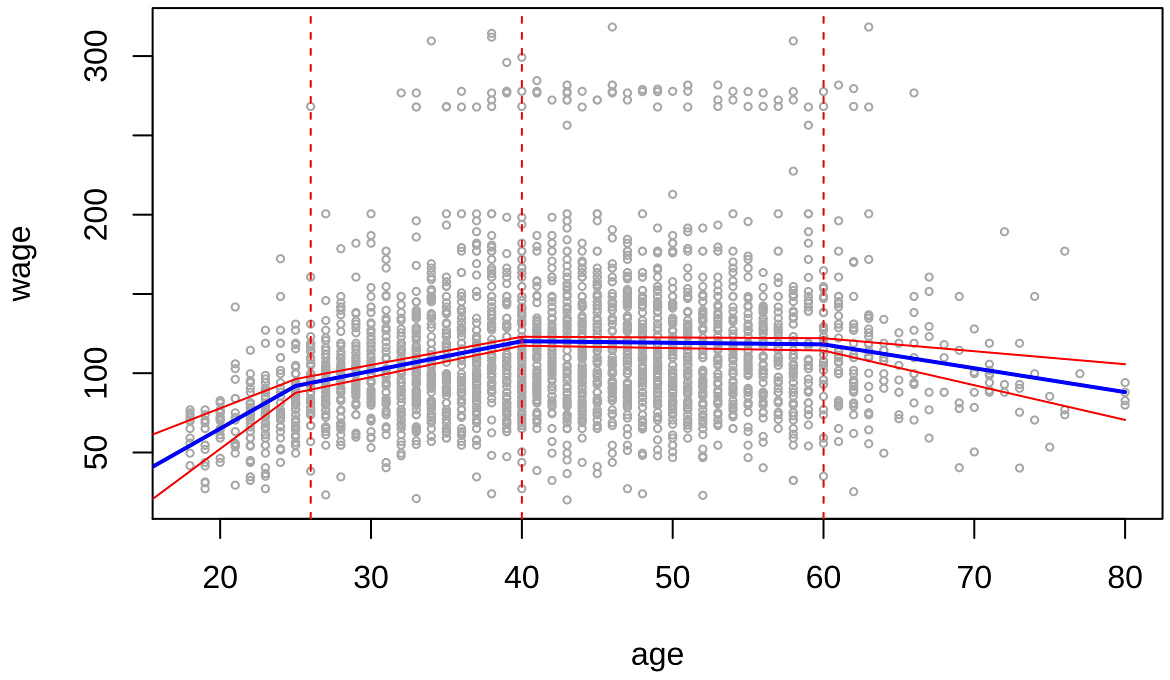
Piecewise linear splines are useful to visualize. First, let's create a vector with age values of interest -> 0, 25, 40, 60 and 80. Let's then predict wages at the knots and use the results to plot a graph.

```r
age.knots <- c(0, 25, 40, 60, 80)

# se = T computes standard errors

preds <- predict(fit.piecewise,
                 list(age = age.knots),
                 se = T)

plot(Wage$age, Wage$wage,
     cex = 0.5,
     col = "darkgrey",
     xlab = "age",
     ylab = "wage")

lines(age.knots, preds$fit, lwd = 2)

# Let's add vertical red lines at each knot

abline(v = 26, col = "red", lty = "dashed") # At age 26
abline(v = 40, col = "red", lty = "dashed") # At age 40
abline(v = 60, col = "red", lty = "dashed") # At age 60
```



Now let's plot the same graph, but with with 95% confidence interval bands using the standard errors in the `se` parameter, that i `2 * preds$se.fit`.

```r
plot(Wage$age, Wage$wage,
     cex = 0.5,
     col = "darkgrey",
     xlab = "age",
     ylab = "wage")

lines(age.knots, preds$fit,
      col = "blue",
      lwd = 2)

# Let's add vertical red lines at each knot

abline(v = 26, col = "red", lty = "dashed") # At age 26
abline(v = 40, col = "red", lty = "dashed") # At age 40
abline(v = 60, col = "red", lty = "dashed") # At age 60

lines(age.knots,
      preds$fit - 2 * preds$se.fit,
      col = "red",
      lwd = 1)

lines(age.knots,
      preds$fit + 2 * preds$se.fit,
      col = "red", lwd = 1)
```



Now let's do CV testing

```
fit.piecewise.train <- lm(wage ~ age +
                                 I((age - 25) * (age > 25)) +
                                 I((age - 40) * (age > 40)) +
                                 I((age - 60) * (age > 60)),
                          data = Wage.train)

pred.piecewise <- predict(fit.piecewise.train, Wage.test)

mse.piecewise <- mean( (Wage.test$wage - pred.piecewise) ^ 2)
mse.piecewise # Take a look
```

```
## [1] 1978.821
```

Interestingly, the piecewise linear spline is more accurate than the quadratic model, but a little less accurate than the cubic model. But the piecewise linear spline is much easier to interpret.

Multivariate Adaptive Regression Spline (MARS)

The piecewise linear spline model above is very interpretable, and tend to has better fit than polynomials and step functions because we can set as many knots as we want and therefore control the slopes in the various resulting segments. But if we are not so interested in interpretation, and are more interested in making predictions, MARS **B-Spline** models are a good alternative option. MARS models are easier to formulate and we can not only fit linear splines, but also polynomial splines connecting knots with curves, rather than with straight lines. I will use the **basis spline** function `bs()` from the **{splines}** package for this purpose. You can use the `summary()` function to view the regression output, if you wish, but the interpretation is not very intuitive.

Linear B-Spline

Let's train the model, make predictions and do CV testing

```
library(splines) # Needed to fit spline models

# Linear B-Spline (degree = 1)

fit.bs1.train <- lm(wage ~ bs(age,
                              knots = c(25, 40, 60),
                              degree = 1),
                    data = Wage.train)

pred.bs1 <- predict(fit.bs1.train, Wage.test)

mse.bs1 <- mean( (Wage.test$wage - pred.bs1) ^ 2)
mse.bs1 # Take a look
```

```
## [1] 1978.821
```

It's like magic !! Exact same result as the piecewise linear spline above

Polynomial and Cubic B-Splines

Let's now try a **Cubic B-Spline**. You can try other polynomials by changing the `degree=` parameter. Notice that I don't use the `degree=` parameter for the cubic spline because it is the default.

```
fit.bs3.train <- lm(wage ~ bs(age,
                              knots = c(25, 40, 60)),
                              data = Wage.train)

pred.bs3 <- predict(fit.bs3.train, Wage.test)

mse.bs3 <- mean( (Wage.test$wage - pred.bs3) ^ 2)
mse.bs3 # Take a look
```

```
## [1] 1991.037
```

A tad more accurate than the linear spline model

Polynomial and Cubic N-Splines

N-Splines are identical to B-Splines, except for the first and last segment, but they tend to give very similar predictions. Let's fit a linear and a cubic N-Spline, and do predictions and CV testing. The only thing that we need to do differently is to use the natural spline function `ns()` instead of the basis spline function `bs()`. Note: the linear B-Spline and linear N-Spline are identical, so I don't model the linear N-Spline below.

```r
# Cubic N-Spline (default is degree = 3)

fit.ns3.train <- lm(wage ~ ns(age,
                              knots = c(25, 40, 60)),
                    data = Wage.train)

pred.ns3 <- predict(fit.ns3.train, Wage.test)

mse.ns3 <- mean( (Wage.test$wage - pred.ns3) ^ 2)

# Print ALL MSE's

row1 <- cbind("CV MSE Linear" = mse.test.1,
              "Poly(2)" = mse.test.2,
              "Poly(3)" = mse.test.3,
              "Step" = mse.step,
              "Piecewise" = mse.piecewise)

row2 <- cbind("Linear B-Spline" = mse.bs1,
              "Cubic B-Spline" = mse.bs3,
              "Cubic N-Spline" = mse.ns3)

# Print all CV Test MSE results

row1
```

```
##      CV MSE Linear  Poly(2)  Poly(3)     Step Piecewise
## [1,]      2068.485 1989.221 1986.052 2021.074  1978.821
```

```r
row2
```

```
##      Linear B-Spline Cubic B-Spline Cubic N-Spline
## [1,]        1978.821       1991.037       1986.576
```

Of all the models, the **Cubic B-Spline** is the most accurate in this example.