Supervised learning: Regression 3

Contents

Introduction

In this practical, we will learn about nonlinear extensions to regression using basis functions and how to create, visualise, and interpret them. Parts of it are adapted from the practicals in ISLR chapter 7.

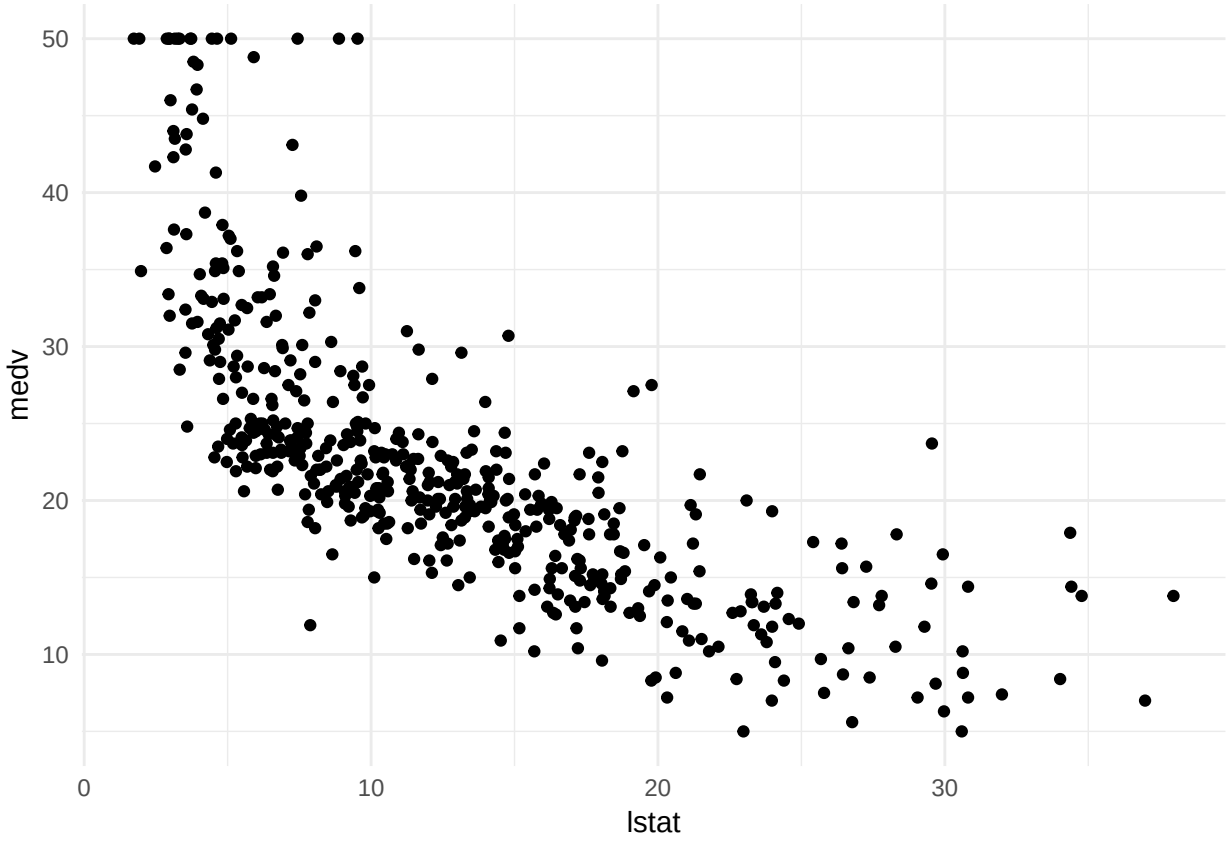
One of the packages we are going to use is `splines`. For this, you will probably need to `install.packages("splines")` before running the `library()` functions.

```
library(MASS)
library(splines)
library(ISLR)
library(tidyverse)
```

Prediction plot

Median housing prices in Boston do not have a linear relation with the proportion of low SES households. Today we are going to focus exclusively on *prediction*.

```
Boston %>%
  ggplot(aes(x = lstat, y = medv)) +
  geom_point() +
  theme_minimal()
```



First, we need a way of visualising the predictions.

---

1. **Create a function called `pred_plot()` that takes as input an `lm` object, which outputs the above plot but with a prediction line generated from the model object using the `predict()` method.**

---

2. **Create a linear regression object called `lin_mod` which models `medv` as a function of `lstat`. Check if your prediction plot works by running `pred_plot(lin_mod)`. Do you see anything out of the ordinary with the predictions?**

Polynomial regression

The first extension to linear regression is polynomial regression, with basis functions $b_j(x_i) = x_i^j$ (ISLR, p. 270).

3. **Create another linear model `pn3_mod`, where you add the second and third-degree polynomial terms `I(lstat^2)` and `I(lstat^3)` to the formula. Create a `pred_plot()` with this model.**

The function `poly()` can automatically generate a matrix which contains columns with polynomial basis function outputs.

---

4. **Play around with the poly() function. What output does it generate with the arguments `degree = 3` and `raw = TRUE`?**

---

5. **Use the poly() function directly in the model formula to create a 3rd-degree polynomial regression predicting `medv` using `lstat`. Compare the prediction plot to the previous prediction plot you made. What happens if you change the poly() function to `raw = FALSE`?**

Piecewise regression

Another basis function we can use is a step function. For example, we can split the `lstat` variable into two groups based on its median and take the average of these groups to predict `medv`.

---

6. **Create a model called `pw2_mod` with one predictor: `I(lstat <= median(lstat))`. Create a pred_plot with this model. Use the coefficients in `coef(pw2_mod)` to find out what the predicted value for a low-lstat neighbourhood is.**

---

7. **Use the `cut()` function in the formula to generate a piecewise regression model called `pw5_mod` that contains 5 equally spaced sections. Again, plot the result using `pred_plot`.**

Note that the sections generated by `cut()` are equally spaced in terms of `lstat`, but they do not have equal amounts of data. In fact, the last section has only 9 data points to work with:

```
table(cut(Boston$lstat, 5))
```

```
##
## (1.69,8.98] (8.98,16.2] (16.2,23.5] (23.5,30.7]   (30.7,38]
```

```
##          183          183           94           37            9
```

---

8. **Optional: Create a piecewise regression model `pwq_mod` where the sections are not equally spaced, but have equal amounts of training data. Hint: use the `quantile()` function.**

Piecewise polynomial regression

Combining piecewise regression with polynomial regression, we can write a function that creates a matrix based on a piecewise cubic basis function:

```
piecewise_cubic_basis <- function(vec, knots = 1) {
  if (knots == 0) return(poly(vec, degree = 3, raw = TRUE))

  cut_vec <- cut(vec, breaks = knots + 1)

  out <- matrix(nrow = length(vec), ncol = 0)

  for (lvl in levels(cut_vec)) {
    tmp <- vec
    tmp[cut_vec != lvl] <- 0
    out <- cbind(out, poly(tmp, degree = 3, raw = TRUE))
  }

  out
}
```

---

9. **This function does not have comments. Copy - paste the function and add comments to each line. To figure out what each line does, you can first create "fake" `vec` and `knots` variables, for example `vec <- 1:20` and `knots <- 2` and try out the lines separately.**

---

10. **Create piecewise cubic models with 1, 2, and 3 knots (`pc1_mod` - `pc3_mod`) using this piecewise cubic basis function. Compare them using the `pred_plot()` function.**

Splines

We're now going to take out the discontinuities from the piecewise cubic models by creating splines. First, we will manually create a cubic spline with 1 knot at the median by constructing a truncated power basis as per ISLR page 273, equation 7.10.

11. **Create a data frame called `boston_tpb` with the columns `medv` and `lstat` from the `Boston` dataset.**

12. **Now use `mutate` to add squared and cubed versions of the `lstat` variable to this dataset.**

13. **Use `mutate` to add a column `lstat_tpb` to this dataset which is 0 below the median and has value `(lstat - median(lstat))^3` above the median. Tip: you may want to use `ifelse()` within your `mutate()` call.**

Now we have created a complete truncated power basis for a cubic spline fit.

14. **Create a linear model `tpb_mod` using the `lm()` function. How many predictors are in the model? How many degrees of freedom does this model have?**

The `bs()` function from the `splines` package does all the work for us that we have done in one function call.

15. **Create a cubic spline model `bs1_mod` with a knot at the median using the `bs()` function. Compare its predictions to those of the `tpb_mod` using the `predict()` function on both models.**

16. **Create a prediction plot from the `bs1_mod` object using the `plot_pred()` function.**

Note that this line fits very well, but at the right end of the plot, the curve slopes up. Theoretically, this is unexpected – always pay attention to which predictions you are making and whether that behaviour is in line with your expectations.

The last extension we will look at is the natural spline. This works in the same way as the cubic spline, with the additional constraint that the function is required to be linear at the boundaries. The `ns()` function from the `splines` package is for generating the basis representation for a natural spline.

17. **Create a natural cubic spline model (`ns3_mod`) with 3 degrees of freedom using the `ns()` function. Plot it, and compare it to the `bs1_mod`.**

18. **Plot `lin_mod`, `pn3_mod`, `pw5_mod`, `pc3_mod`, `bs1_mod`, and `ns3_mod` and give them nice titles by adding + `ggtitle("My title")` to the plot. You may use the function `plot_grid()` from the package `cowplot` to put your plots in a grid.**

Programming assignment (optional)

19. **Use 12-fold cross validation to determine which of the 6 methods (`lin`, `pn3`, `pw5`, `pc3`, `bs1`, and `ns3`) has the lowest out-of-sample MSE.**