Supervised learning: Regression 1

Contents

Introduction

In this practical, you will learn how to perform regression analysis, how to plot with confidence and prediction intervals, how to calculate MSE, perform train-test splits, and write a function for cross validation.

Just like in the practical at the end of chapter 3 of the ISLR book, we will use the `Boston` dataset, which is in the `MASS` package that comes with `R`.

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

Regression in R

Regression is performed through the `lm()` function. It requires two arguments: a `formula` and `data`. A `formula` is a specific type of object that can be constructed like so:

```
some_formula <- outcome ~ predictor_1 + predictor_2
```

You can read it as "the outcome variable is a function of predictors 1 and 2". As with other objects, you can check its class and even convert it to other classes, such as a character vector:

```
class(some_formula)
```

```
## [1] "formula"
```

```r
as.character(some_formula)
```

```
## [1] "~"                             "outcome"
## [3] "predictor_1 + predictor_2"
```

A linear model can thus be generated by specifying the outcome variable and the predictors in a formula and specifying the dataset these variables should be taken from.

Create a linear model object called `lm_ses` that estimates the formula `medv ~ lstat` using the `Boston` dataset

---

You have now run a regression with `medv` (housing value) as the outcome/dependent variable and `lstat` (socio-economic status) as the predictor / independent variable.

Remember that a regression estimates β_0 (the intercept) and β_1 (the slope) in the following equation:

$$\boldsymbol{y} = \beta_0 + \beta_1 \cdot \boldsymbol{x}_1 + \boldsymbol{\epsilon}$$

Use the function `coef()` to extract the intercept and slope from the `lm_ses` object. Interpret the slope coefficient.

Use `summary()` to get a summary of the `lm_ses` object. What do you see? You can use the help file `?summary.lm`.

Because we now have a model object that represents the formula

$$\text{medv}_i = 34.55 - 0.95 * \text{lstat}_i + \epsilon_i$$

we can predict a new `medv` value by simply inputting any `lstat` value. The `predict()` method allows us to quickly do this for the `lstat` values in the original dataset.

Save the predicted y values to a variable called `y_pred`

Create a scatter plot with `y_pred` mapped to the x position and the true y value (`Boston$medv`) mapped to the y value. What do you see? What would this plot look like if the fit were perfect?

We can also generate predictions from new data using the `newdat` argument. For that, we need to prepare a data frame with new values for the original predictors.

Use the `seq()` function to generate a sequence of 1000 equally spaced values from 1 to 40. Store this vector in a data frame (`data.frame()` or `tibble()`) with as its column name `lstat`. Name the data frame `pred_dat`

Use the newly created data frame as the `newdata` argument to a `predict()` call for `lm_ses`. Store it in a variable named `y_pred_new`.

Plotting lm() in `ggplot`

A good way of understanding your model is by visualising it. We are going to walk through the construction of a plot with a fit line and prediction / confidence intervals from an `lm` object.

Create a scatter plot from the `Boston` dataset with `lstat` mapped to the x position and `medv` mapped to the y position. Store the plot in an object called `p_scatter`.

Now we're going to add a prediction line to this plot.

Add the vector `y_pred_new` to the `pred_dat` data frame with the name `medv`.

Add a geom_line() to `p_scatter`, with `pred_dat` as the `data` argument. What do you see?

The `interval` argument can be used to generate confidence or prediction intervals. Create a new object called `y_pred_95` using `predict()` (again with the `pred_dat` data) with the `interval` argument set to "confidence". What is in this object?

Create a data frame with 4 columns: `medv`, `lstat`, `lower`, and `upper`.

Add a `geom_ribbon()` to the plot, below the `geom_line()` and the `geom_points()` of before. Give it a nice colour and clean up the plot, too!

Do the same thing, but now with the prediction interval instead of the confidence interval.

Mean square error

Write a function called `mse()` that takes in two vectors: true y values and predicted y values, and which outputs the mean square error. Start like so:

```
mse <- function(y_true, y_pred) {
  # your function here
}
```

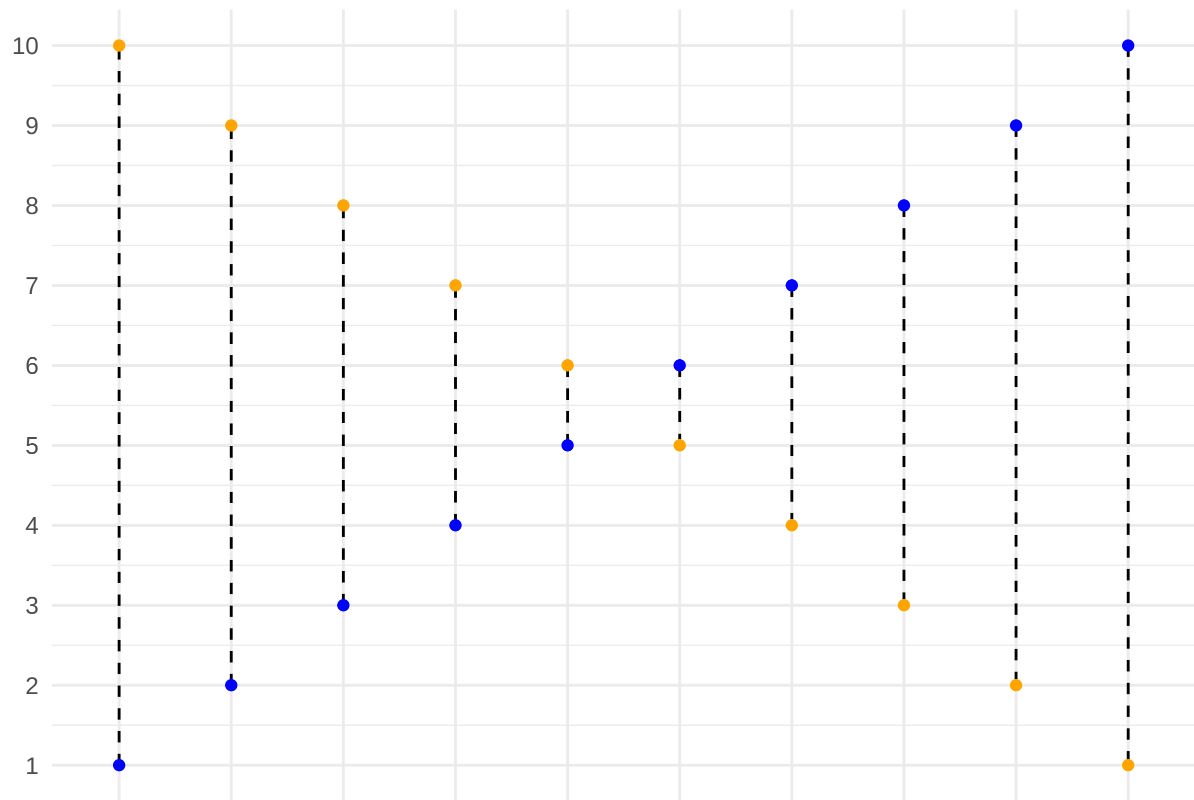
Wikipedia may help for the formula.

Make sure your `mse()` function works correctly by running the following code.

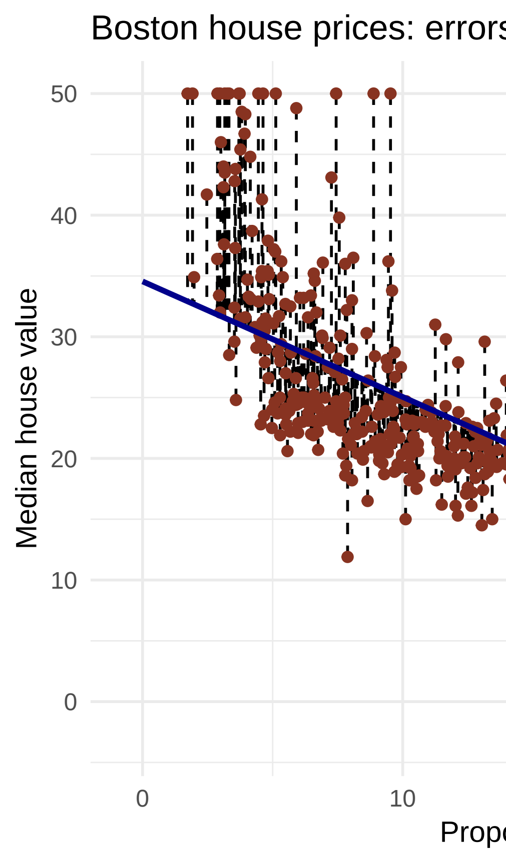
```
mse(1:10, 10:1)
```

```
## [1] 33
```

You have now calculated the mean squared length of the dashed lines below.



Calculate the mean square error of the `lm_ses` model. Use the `medv` column as `y_true` and use the `predict()` method to generate `y_pred`.



You have calculated the mean squared length of the dashed lines in the plot below

Train-validation-test split

Now we will use the `sample()` function to randomly select observations from the `Boston` dataset to go into a training, test, and validation set. The training set will be used to fit our model, the validation set will be used to calculate the out-of sample prediction error during model building, and the test set will be used to estimate the true out-of-sample MSE.

The `Boston` dataset has 506 observations. Use `c()` and `rep()` to create a vector with 253 times the word "train", 152 times the word "validation", and 101 times the word "test". Call this vector `splits`.

Use the function `sample()` to randomly order this vector and add it to the `Boston` dataset using `mutate()`. Assign the newly created dataset to a variable called `boston_master`.

Now use `filter()` to create a training, validation, and test set from the `boston_master` data. Call these datasets `boston_train`, `boston_valid`, and `boston_test`.

We will set aside the test dataset for now, until the moment we may want to compare models of different

---

Create a linear regression object called `model_1` using the training dataset. Use the formula `medv ~ lstat` like in the first `lm()` exercise. Use `summary()` to check that this object is as you expect.

Calculate the MSE with this object. Save this value as `model_1_mse_train`.

Now calculate the mean squared error on the validation set and assign it to variable `model_1_mse_valid`. Use the newdata argument to `predict()`. You can reach the correct help file by typing `?predict.lm`.

This is the estimated out-of-sample mean squared error.

Create a second model `model_2` which includes `age` and `tax` as predictors. Calculate the train and validation MSE.

Compare model 1 and model 2 in terms of their training and validation MSE. Which would you choose and why?

Calculate the test MSE for the model of your choice in the previous question. What does this number tell you?

Programming exercise: cross-validation

This is an advanced exercise. Some components we have seen before in this and previous practicals, but some things will be completely new. Try to complete it by yourself, but don't worry if you get stuck. If you don't know about `for loops` in `R`, read up on those before you start the exercise.

Use help in this order:

- R help files
- Internet search & stack exchange
- Your peers
- The answer, which shows one solution

You may also just read the answer and try to understand what happens in each step.

Create a function that performs k-fold cross-validation for linear models.

Inputs: - `formula`: a formula just as in the `lm()` function - `dataset`: a data frame - `k`: the number of folds for cross validation - any other arguments you need necessary

Outputs: - Mean square error averaged over folds

Use your function to perform 9-fold cross validation with a linear model with as its formula `medv ~ lstat + age + tax`. Compare it to a model with as formulat `medv ~ lstat + I(lstat^2) + age + tax`.
