# Math for Data Science: Lab 8

Prof. Asya Magazinnik

2023-14-11

## Matrix Algebra and Calculus: An Application to Linear Regression

Today we will practice matrix algebra and calculus with an application that you will get to know very well: linear regression. For now, we will put aside many (very important) issues around causal and statistical inference, and just think of regression as a predictive or descriptive exercise: what is the best linear model that we can fit to our data?

To make things concrete, let's use a real dataset of median home values in Boston. Load the dataset `BostonHousing` from the `mlbench` package. Take a look at the data. You can read the documentation here. What are the included variables? What is the unit of analysis?

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
## Warning: package 'mlbench' was built under R version 4.3.2
```

Now, let's write down our linear model. We are interested in modeling the median home price in a Census tract (`medv`) as a function of the following variables:

- `crim`: per capita crime rate in the town
- `chas`: Charles River dummy variable (1 if tract is on the water, 0 if not)
- `nox`: nitric oxides concentration (parts per 10 million)
- `dis`: weighted distances to five Boston employment centres
- `ptratio`: pupil-teacher ratio by town
- `rad`: index of accessibility to radial highways

We require that our model be linear in the parameters, which means that for any tract $i$, we can express the *response* $y_i$ (the median home value) as a *linear combination* of the variables plus some error term.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i \tag{1}$$

Our goal is to find the "best" values of $\beta_0, \beta_1, \beta_2, ..., \beta_p$, where "best" has a specific meaning: the ones that minimize the sum of squared errors over the dataset.[1]

$$\min_{\beta_0,\beta_1,...,\beta_p} \sum_{i=1}^{n} \varepsilon_i^2 = \min_{\beta_0,\beta_1,...,\beta_p} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - ... - \beta_p x_{ip})^2 \tag{2}$$

Before we go on, let's set this up as a maximization problem in the way we are familiar with: as a system of equations to solve.

$$\begin{aligned}
\frac{\partial}{\partial \beta_0} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - ... - \beta_p x_{ip})^2 = 0 \\
\frac{\partial}{\partial \beta_1} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - ... - \beta_p x_{ip})^2 = 0 \\
\frac{\partial}{\partial \beta_2} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - ... - \beta_p x_{ip})^2 = 0 \\
... \\
\frac{\partial}{\partial \beta_n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - ... - \beta_p x_{ip})^2 = 0
\end{aligned}$$

But we won't go this route today. Instead, we'll solve the same problem using linear algebra and calculus. And you'll see that, once you know the basic rules, the matrix route is the more efficient one.

Let's write what is represented by Equation (2) above in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & ... & x_{1p} \\ 1 & x_{21} & x_{22} & ... & x_{2p} \\ ... & & & & \\ 1 & x_{n1} & x_{n2} & ... & x_{np} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ ... \\ \beta_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ ... \\ \varepsilon_n \end{bmatrix} \tag{3}$$

Let's collapse this into:

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon \tag{4}$$

where:

- $\mathbf{Y}$ $(n \times 1)$ is called the **response vector**
- $\mathbf{X}$ $(n \times p + 1)$ is called the **design matrix**
- $\beta$ $(p + 1 \times 1)$ is our vector of **coefficients**
- $\varepsilon$ $(n \times 1)$ is our vector of **error terms**

Now, let's write the sum of squared error minimization problem in matrix form:

[1] Why minimize this quantity and not something else? The short answer is that it has some nice statistical properties. But there are many other *loss functions* you might minimize, which you will meet later in this class and in Machine Learning.

$$\sum_{i=1}^{n} \varepsilon_i^2 = \begin{bmatrix} \varepsilon_1 & \varepsilon_2 & ... & \varepsilon_n \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ ... \\ \varepsilon_n \end{bmatrix} = \varepsilon^T \varepsilon \tag{5}$$

And now we can do some matrix manipulation:

$$\begin{aligned} \varepsilon^T \varepsilon &= (\mathbf{Y} - \mathbf{X}\beta)^T (\mathbf{Y} - \mathbf{X}\beta) \\ &= \mathbf{Y}^T\mathbf{Y} - \mathbf{Y}^T\mathbf{X}\beta - \beta^T\mathbf{X}^T\mathbf{Y} + \beta^T\mathbf{X}^T\mathbf{X}\beta \end{aligned} \tag{6}$$

We'll still take the first derivative and set it equal to 0 to solve for our error-minimizing $\beta$'s, but we'll do it with the whole vector of $\beta$'s in one shot:

$$\frac{\partial}{\partial\beta}(\mathbf{Y}^T\mathbf{Y} - \mathbf{Y}^T\mathbf{X}\beta - \beta^T\mathbf{X}^T\mathbf{Y} + \beta^T\mathbf{X}^T\mathbf{X}\beta) = \frac{\partial}{\partial\beta}\mathbf{Y}^T\mathbf{Y} - \frac{\partial}{\partial\beta}\mathbf{Y}^T\mathbf{X}\beta - \frac{\partial}{\partial\beta}\beta^T\mathbf{X}^T\mathbf{Y} + \frac{\partial}{\partial\beta}\beta^T\mathbf{X}^T\mathbf{X}\beta \tag{7}$$

Now, please take the following steps:

1. Use the handy Matrix Cookbook[2] to show that Equation 7 reduces to:

$$-2\mathbf{X}^T\mathbf{Y} + 2\mathbf{X}^T\mathbf{X}\beta \tag{8}$$

2. Set Equation 8 equal to 0 and solve for $\hat{\beta}$. Show that:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \tag{9}$$

3. Using the `BostonHousing` data, create your design matrix and response vector.[3]

```
one_col <- 1

selected <- BostonHousing %>%
  select(dis, rad)

df <- cbind(one_col, selected)

matrix <- as.matrix(df)
response <- as.matrix(BostonHousing$medv)
```

4. Manually compute the $\hat{\beta}$ vector that you derived in Equation 9.[4]

[2] Section 2.4, Derivatives of Matrices, Vectors and Scalar Forms

[3] Hint: for this next part, your design matrix needs to be a matrix, not a data frame. Some annoying little data manipulations might be needed to get it into that form.

[4] Hint: you can use the `solve` function to compute a matrix inverse.

```
beta <- solve((t(matrix)%*%matrix))%*%(t(matrix)%*%response)

beta
```

```
##                [,1]
## one_col 24.6355610
## dis      0.3537546
## rad     -0.3607835
```

5. Check your answer using R's built-in regression function `lm`.

```
betalm <- lm(response ~ matrix)
```

6. Using Equation 4 above, compute the predicted values: $\hat{\mathbf{Y}} = \mathbf{X}\beta$. Compare to the predictions generated using the built-in `predict` function in R.

```
predictions_predict <- predict(betalm)

predictions <- matrix%*%beta

head(predictions_predict)
```

```
##        1        2        3        4        5        6
## 25.72163 25.67113 25.67113 25.69774 25.69774 25.69774
```

```
head(predictions)
```

```
##       [,1]
## 1 25.72163
## 2 25.67113
## 3 25.67113
## 4 25.69774
## 5 25.69774
## 6 25.69774
```