Assignment 2: Linear and Logistic Regression
Introduction to Applied Data Science
2022-2023

Bas Machielsen
a.h.machielsen@uu.nl

April 2023

Assignment 2: Linear and Logistic Regression to Predict Poverty

1. Preparation

In this assignment, you will build your own linear and logistic regression models, using the gradient descent algorithms we talked about in the lecture. Similarly to the previous assignment, you will fill up the code chunks left empty in this document, and you will interpret them in this document. To start with, please replace my name and e-mail address with yours. Then, remove the lines:

```
output:
  pdf_document:
    includes:
      in_header: "preamble.tex"
    latex_engine: xelatex
```

from the document, and replace them with:

```
output: pdf_document
```

2. Obtain the required data

Next, we are going to make use of the `wbstats` package, which you have seen before. If you haven't done so already, you can install the package with `install.packages('wbstats')`. Make sure not to put this in your Rmarkdown document, as R will then attempt to install this package every time you knit your document.

The `wbstats` package allows you to navigate the World Bank database, and download datasets without having to visit the World Bank website, download the data into a spreadsheet, and subsequently load it into R. With the help of this package, we just download the data into R right away.

Question 1: Read this so-called vignette to find out how to navigate the World Bank data using the `wbstats` package. Download the variable that measures the poverty headcount index, the proportion of the population with daily per capita income (in 2011 PPP) below the poverty line at $1,90.

```
library(wbstats)
all_poverty_variables <- wbstats::wb_search("poverty")

View(all_poverty_variables)
```

Change `eval = FALSE` to `eval = TRUE` below once you've found the right answer.

In addition, we will also load the `tidyverse` package, as we're going to do some data wrangling to glue various pieces of data together:

```
library(tidyverse)
```

```
poverty_data <- wb_data("1.0.HCount.1.90usd")
```

Question 2: For which countries is this data available? Print a vector of the unique country names. Do not type a string variable of all the country names yourself, but use code to extract this.

```
poverty_data %>% select(country) %>% pull() %>% unique()
```

```
##  [1] "Argentina"          "Bolivia"            "Brazil"
##  [4] "Chile"              "Colombia"           "Costa Rica"
##  [7] "Dominican Republic" "Ecuador"            "Guatemala"
## [10] "Honduras"           "Mexico"             "Nicaragua"
## [13] "Panama"             "Peru"               "Paraguay"
## [16] "El Salvador"        "Uruguay"            "Venezuela, RB"
```

Next, save the *unique* `iso2c` and `iso3c` codes in two separate vectors:

```
iso2c <- unique(poverty_data %>% select(iso2c)) %>% pull()

iso3c <- unique(poverty_data %>% select(iso3c)) %>% pull()
```

For convenience, we might want to rename our poverty variable. One way to do this is as follows:

```
poverty_data <- poverty_data %>%
  rename(poverty = `1.0.HCount.1.90usd`)
```

The rename function follows the `new = old` syntax, which you can verify by typing `?rename` in the console.

Next, we will download datasets related to varieties of democracy. This package is not available on the official R package repository CRAN, but it a custom package which is downloadable from somebody's github repository. To download packages from Github, we need the `devtools` package, and then, we can install the package using the `install_github()` function.

```
#install.packages("devtools")
```

After installing the package, we want to load the dataset from the package.

Question x: Find out what the median is of the $1.90 poverty head count variable in the dataset you downloaded. Make a new dummy variable `povyesno`, meaning:

$$\text{povyesno}_i = \begin{cases} 1 & \text{if 1.0.HCount.1.90usd } > \text{median(1.0.HCount.1.90usd)} \\ 0 & \text{otherwise} \end{cases}$$

You can easily do so using the `if_else` function from `dplyr`. Check `?if_else` for its syntax.

```
poverty_data <- poverty_data %>%
  mutate(povyesno = if_else(poverty > median(poverty, na.rm = TRUE), 1, 0))
```

Building a linear regression model

Now, we want to predict poverty by means of two factors: (i) historical educational investment, and (ii) political stability. We will undertake various efforts to obtain these data from different sources.

Linear regression can also be performed using an analytical solution for the minimum loss. This function is implemented in R in the `lm` function. Check the syntax for the `lm` method using `?lm`.

Question x: Verify that the solution from the analytical model (`lm(y ~ x1 + x2, data = dataset)`) is identical to the solution you obtained when performing gradient descent.

Question x: you can also report a summary of the model estimated using the `modelsummary` package and function. Report a summary of the model below.

```
model1 <- lm(y ~ x1 + x2, data = dataset)

modelsummary::modelsummary(model1)
```

Question x: Give an interpretation of the model. What happens when a country becomes more democratic, and what would happen to poverty if a country had invested more in education?

Question x: Generate the predicted values from the model. Find the *highest* predicted value. For which country is this? And what is the predicted value? Use code to find all of this information.

Building a logistic regression model

Finally, we'll analyze the same data using a logistic regression model. A key element in a logistic regression model is the so-called *sigmoid* function. The sigmoid function $S(x)$ is defined as follows:

$$S(x) = \frac{1}{1 + e^{-x}}$$

Question x: Implement the sigmoid function as a function in R

```
sigmoid <- function(x){
  1 / (1 + exp(-x))
}
```

Question x: Differentiate the sigmoid function with respect to x. Rewrite this derivative to show that it is equal to $S(x)(1 - S(x))$ using the definition of $S(x)$ given above.

Question x: Implement the derivative of the sigmoid function in R.

```
sigmoid_derivative <- function(x){

  sigmoid(x) * (1-sigmoid(x))

}
```

A nice thing about this sigmoid function is that its outcomes always fall between 0 and 1. We'll use the sigmoid function to build a logistic regression model. In particular, instead of a simple argument x, we are predicting the outcome, poverty, on the basis of the same two independent variables as before, and two corresponding coefficients. Our sigmoid function would thus look like this:

$$\text{Poverty}_i = \frac{1}{1 + e^{-[\alpha + \beta_1 \cdot \text{educ}_i + \beta_2 \cdot \text{stab}_i]}}$$

Question x: Use the chain rule from calculus to identify the derivatives of the above particular sigmoid function with respect to education$_i$ and stability$_i$ and α. Hint: use the result from above, and use that:

$$\frac{\partial S(.)}{\partial \text{education}_i} = \frac{\partial S(x)}{\partial x} \cdot \frac{\partial x}{\partial \text{education}_i}$$

In the next few steps, we want to implement these functions in our gradient descent algorithm. We can largely use the same setup as we used in the linear model, with the exception of the derivates. Please fill the derivatives in this algorithm yourself:

```
gradient_descent <- function(x, y, learn_rate, conv_threshold = 0.00001, n = length(y), max_iter = 5000
  #plot(x, y, col = "blue", pch = 20)
  #initialize constant term, beta1 and beta2
  alpha <- runif(1, 0, 10)
  beta1 <- runif(1, 0, 10)
  beta2 <- runif(1, 0, 10)

  #create the predicted values
  yhat <- 1 / (1+sigmoid(alpha + beta1*x[1] + beta2*x[2]))
  MSE <- sum((y - yhat) ^ 2) / n

  converged = F
  iterations = 0

  while(converged == F) {
    ## Implement the gradient descent algorithm
    alpha_new <- alpha - learn_rate * ((1 / n) * (sum(sigmoid_derivative(alpha + beta1*x[1] + beta2*x[2]
    beta1_new <- beta1 - learn_rate * ((1 / n) * (sum(sigmoid_derivative(alpha + beta1*x[1] + beta2*x[2]
    beta2_new <- beta2 - learn_rate * ((1 / n) * (sum(sigmoid_derivative(alpha + beta1*x[1] + beta2*x[2]

    ## Write the updated coefficients and compute predictions again
    alpha <- alpha_new
    beta1 <- beta1_new
    beta2 <- beta2_new

    yhat <- 1 / (1+sigmoid(alpha + beta1*x[1] + beta2*x[2]))
    MSE_new <- sum((y - yhat) ^ 2) / n

    if(MSE - MSE_new <= conv_threshold) {
      #abline(c, m)
      converged = T
      return(paste("Optimal intercept:", alpha,
                   "Optimal slope x1:", beta1,
                   "Optimal slope x2:", beta2))
    }
    iterations = iterations + 1
    if(iterations > max_iter) {
      #abline(c, m)
      converged = T
      return(paste("Optimal intercept:", alpha,
```

```
                    "Optimal slope x1:", beta1,
                    "Optimal slope x2:", beta2))
    }
  }
  }
```

```
gradient_descent(mtcars[1:2], mtcars$vs, 1)
```

```
## [1] "Optimal intercept: 0.371381987351924 Optimal slope x1: 8.12523144064471 Optimal slope x2: 5.6038
```