Linear Regression: Infant Mortality

Signal Data Science

In this assignment, you'll learn how to do a linear regression, the simplest of all machine learning techniques! We'll use data from the United Nations about infant mortality in different countries for this analysis.

Refer to the code template in `day1Example.R` as you work through this assignment. You'll go through the existing code, alternating between figuring out what it does, answering questions, and writing your own code to supplement what's already there.

Getting started

First, we have some preliminaries to take care of.

- Install the packages `car`, `Ecdat`, `HistData`, `ggplot2`, `dplyr`, `Rmisc`, and `GGally`. Load `day1Example.R` in RStudio.
- Load the packages `car`, `ggplot2`, and `GGally`. Set `df = UN`.
- Print out the data contained in `df` by just typing `df` into the console. Run `View(df)` (case-sensitive!) to bring up a nicer GUI for viewing the dataset. Does anything stand out to you? If there are any questions you'd like to answer with this data, write them down as comments in your R file.
- R packages are extensively documented online. Look at the reference manual for the `car` package to read about the `UN` data.

Viewing correlations and cleaning the data

The first step in any data analysis is to clean the data! Data is often messy and incomplete, which gets in the way of doing any analysis. After data cleaning, it's typically a good idea to look at some basic summary statistics before jumping directly into predictive modeling.

- Use `cor()` to find the correlation between infant mortality and GDP. There will be an error – what's wrong, and why is this happening? Look in the documentation for `cor()` and figure out how to tell it to ignore entries with missing values.
- For readability, multiply the correlation matrix by 100 and `round()` it to whole numbers.
- Wrap the above code into a function, `cor2()`, which outputs a correlation matrix (ignoring entries with missing values) with rounded whole numbers.

Instead of making each function we use handle missing values (`NA`s) in its own way, we can just remove all the rows in a dataset which contain missing values. If there are many such rows, this will decrease our sample size dramatically and potentially skew the results as well. However, if the proportion of rows with missing values is low, the effect of doing so will be minor.

- Type `?na.fail` and read the documentation on `NA`-related functions; find one appropriate for the job and use it to make `df2`, a data frame identical to `df` except without rows which contain missing values.

Visualizing distributions

Next, we'll do some visualizations and transformations of the data as a prelude to linear modeling.

- Use `ggpairs()` (from the `GGally` package) on the data frame.
  - What do you notice about the distributions of GDP and infant mortality?
  - Use `log()` on `df` to take the natural logarithm of every value in the data frame; assign the resulting data frame to `ldf`. Examine `ldf` with `ggpairs()`.
  - Note the differences and reflect on the appropriateness of a linear model for the untransformed vs. transformed data.

Running linear regressions

If you don't remember much about linear regressions, briefly skim the relevant sections in *Applied Predictive Modeling*.

- Run the lines in the example code which use `lm()` to generate linear fits of infant mortality against GDP. You can type `linear_fit` and `summary(linear_fit)` in the console to get summaries of the results.

Note that `summary()` will print out a statistic denoted **Adjusted R-squared**, which can be interpreted as the *proportion of variance in the target variable explained by the predictors*.

- Briefly skim StackExchange to see how Adjusted R-squared is calculated.

In general, a higher Adjusted R-squared is better, because it means that our regression model captures more of the variance in the target variable.

Plotting linear regressions

To plot the results of a simple linear regression, it's actually easier to let `ggplot2` fit the model for you.

- Run the line starting with `ggplot...` to plot a scatterplot of the data in `df2` along with a linear fit of infant mortality to GDP.
  - What happens when you remove the `method` argument in `geom_smooth()`?
  - Look at the documentation for `geom_smooth()` and determine what method it defaults to.
  - Find the documentation online for that method, **briefly** read about it, and explicitly call it in the `method` argument instead of `"lm"`. How good of an approximation is a linear model?
  - Make the same plots for the linear fit of log(infant mortality) vs. log(GDP). (*Hint:* To access column `c` of a dataframe `df`, use the syntax `df$c`.)

Looking at the residuals

A residual is a fancy word for prediction error; it's the difference given by `actual - predicted`. After we fit a model to our data, visualizing the residuals allows us to easily check for evidence of heteroskedasticity.

- Run the first `qplot()` command, which plots the residuals (`actual - predicted` values) of the simple linear fit. Is there evidence of heteroskedasticity?
- Run the second `qplot()` command, which plots the residuals of the linear fit of the log-transformed data. Is the log-log transformation an improvement? Why or why not?

One of the assumptions of linear regression is that the variances of the distributions from which the errors are drawn have the same variance. If that's the case,

then we shouldn't see much structure in the plot of the residuals. As such, if the residual plot looks very different from random noise, that's a warning sign indicating that the linear model may not be working very well. For example, compare the top and bottom plots here (top has structure, bottom doesn't).

- Using the documentation and experimenting in the console, make sure you understand what `df$infant.mortality - exp(fitted(loglog_fit))` does.
- Generate and analyze a plot of residuals for the linear fit of log(infant mortality) vs. GDP.

Anscombe's quartet

We'll finish up by briefly looking at *Anscombe's quartet*, a famous dataset constructed in 1973 by Francis Anscombe.

- Use `read.csv()` to load `anscombe.csv`, in the `anscombe` dataset, into a variable called `df`. Again, type `df` into the console to see the data and the column names; column `y2` can be accessed with `df$y2`, etc.
- Compute the `mean()` and variance of each of the `x` and `y` columns.
- Compute the correlation of each of the four pairs of `x` and `y` columns.
- Compute the parameters of a linear regression of each `y` column against its corresponding `x` column.
- Use `plot()` to plot each of the four pairs of `x` and `y` columns.

Visualization is important! Summary statistics don't tell the whole story.