Self-Assessment 2

Signal Data Science

We'll be having another self-assessment. As before,

- Type your answers in a new R script file with comments indicating where the answer to each question begins.
- Write down your starting time. When you finish, email `signaldatascience@gmail.com` with your R script attached along with the amount of time you spent on the self assessment.
- Work individually. You can however consult R documentation, look at old assignments, use the Internet, etc., but don't copy and paste code verbatim.
- Make your code as clear, compact, and efficient as possible. Use everything that you've learned! **Please comment and organize your code so we can easily tell how parts of your R script correspond to specific problems.**

Packages you may find useful: `dplyr`, `ggplot2`, and `glmnet`.

Regularization, grid search, and cross-validation

In this self-assessment, you'll be getting some independent practice with the `glmnet()` function. We'll be returning to the `msq` dataset from the `psych` package, which we looked at in the first self-assessment.

Like before, we'll be trying to predict Extraversion and Neuroticism ratings with the columns `"active"` through `"scornful"`. *Unlike* before, we'll be using *elastic net regression*, so we have two hyperparameters to optimize: `\alpha`, controlling the balance between L^1 and L^2 regularization, and λ, the strength of the regularization penalty.[1]

In short, our goal is to **find the best value for the hyperparameters** (α, λ) when predicting Extraversion and Neuroticism. In order to do so, you'll be using *n-fold cross-validation* to calculate RMSE scores associated with each possible pair of values for (α, λ).

[1] Consult the `glmnet` documentation if you need a refresher on how α works.

Putting it differently, you'll be performing a *grid search* over values of α while *re-implementing* the functionality of `cv.glmnet()` for each value of α tested. (You **should not be using `cv.glmnet()`** in this part of the self-asessment – just regular `glmnet()`.) In total, you'll be re-implementing the basic functionality of `caret`'s `train()` function.

Here are some general tips to keep in mind while working on this:

- Keep your code well-commented, so you can understand what each part does.
- Separate different sections of code with a single blank line of whitespace. This will allow you to *visually* see which code blocks correspond to what functionality.
- Use sensibly-named variables.

This is *more involved than it sounds*, so we'll give you a detailed outline of what your code should accomplish. We've organized the bullet points below so that each one corresponds to a specific task, usually not more than just a couple lines of code, and that they're sequentially ordered in the way that your code should be. Moreover, the nesting structure of the bullet points corresponds precisely to the correct nesting structure of your `for` loops.

Getting started

First, we'll get some preliminaries out of the way – loading the data, generating fold assignments which you'll use for cross-validation, etc.

- For reproducibility, set the seed to 1.
- Choose values of α and λ to iterate over. For consistency with other students' work, set $\alpha = 0, 0.1, 0.2, \ldots, 1$ and $\lambda = 10^s$ where s ranges from 1 to -3 with 50 different values spaced uniformly. Make sure that the sequence for λ values is *decreasing*, because `glmnet()` wants you to pass in decreasing sequences for its `lambda` parameter.
- Load the `msq` dataset and fill in `NA`s in the numeric columns with column means.
- Make a `features` data frame containing just the columns in `msq` from `active` to `scornful`. Similarly, make separate variables for the Extraversion and Neuroticism columns.
- Generate *fold assignments* for *n-fold cross-validation* with $n = 10$. We recommend shuffling the row numbers, taking them modulo n, and adding 1, but there are other ways of doing this which work as well.
- You'll soon implement n-fold cross validation, and because of the way you'll structure your code, you'll have to access the train/test data many

times. To *reduce on computation time*, it's best to *pre-compute* the subsets of data you need for each fold. To that end, initialize 4 lists of the correct length[2] in order to hold these precomputed subsets of data *for each cross-validation fold*: (1) the subset of features which you'll train `glmnet()` on (90% of the data), (2) the subset of features which you'll test the trained model on (10% of the data), (3) the subset of the Extraversion vector which you'll train `glmnet()` on, and (4) the subset of the Neuroticism vector which you'll train `glmnet()` on. Next, iterate through the 10 folds and fill in the elements of these 4 lists. When scaling the *test* data, be sure to pass in `center` and `scale` parameters to `scale()` corresponding to the attributes of the scaled *training* data.

- Create an empty data frame with `data.frame()` to store the results of your computations. You don't need to do anything aside from setting a variable equal to `data.frame()`, but in the future you'll be filling in each row with (1) a value of α, (2) a value of λ, (3) the cross-validated RMSE for predicting Extraversion with the selected (α, λ), and (4) the cross-validated RMSE for predicting Neuroticism with the selected (α, λ).
- Write a convenience function `rmse(x, y)` that takes in two vectors `x` and `y` and returns the associated RMSE. (It doesn't matter which vector represents the "actual" values, because $(x-y)^2 = (y-x)^2$.)

Grid search

Next, you'll be writing two nested `for` loops in order to look at every combination of λ and α values. Within these `for` loops, you'll be calculating a cross-validated estimate of the RMSE for every such combination.

- Iterate over every value of α. In each iteration, do the following:
  - Print the value of α which you're testing.
  - For each fold of the data, we'll be fitting a regularized linear model to both Extraversion and Neuroticism against the out-of-fold data. As such, initialize two lists of length n in which you'll store these fits for later.
  - Iterate over each fold of the data. In each iteration, do the following:
    * Use `glmnet()` to fit a regularized linear model for Extraversion with the selected value of α with the training data associated with that fold. Since `glmnet()` can fit a whole range of λ values simultaneously (it has a cool internal algorithm!), pass in your range of λ values to the `lambda` parameter as well.
    * Do the same for Neuroticism.

[2] Use `vector("list", list_length)` to initialize an empty list of length `list_length`.

        * Store both of those linear fits in the lists you previously created for storing `glmnet()` fits.
    - Next, iterate over every value of λ. In each iteration, do the following:
        * Print the value of λ which you're testing.
        * Initialize vectors of the appropriate length for *predictions* of both Extraversion and Neuroticism.
        * Iterate over each fold of the data. In each iteration, do the following:
            · Fill in the subset of the Extraversion predictions vector corresponding to the current fold with predictions made with the `glmnet()` model object previously trained on the out-of-fold data. Remember to pass in the current value of λ to the `s` parameter of `predict()`.
            · Do the same for Neuroticism.
        * Calculate the RMSE values corresponding to your predictions of Extraversion and Neuroticism.
        * `rbind()` the row `(alpha, lambda, rmse_extraversion, rmse_neuroticism)` into the results data frame you created earlier. (It *is* possible to initialize the data frame prior to the nested loops and to fill it out as we go, but the improvement in computation time required is so miniscule that either approach is fine.)
- Set the column names of your results data frame appropriately, to something like `c("alpha", "lambda", "rmse_extraversion", "rmse_neuroticism")`.
- Print out your results data frame and visually examine the results.

Finishing up

Now that we have cross-validated RMSE estimates for every (α, λ) pair, the next step is to find the pair with the *lowest* RMSE. With that optimal pair of hyperparameters, we'll train an elastic net regularized linear model on the entire dataset. Finally, we'll visualize the results of our regression analysis.

- Define a utility function `arg_min(v)` which returns the index of the vector `v` corresponding to its minimal value. (If there are multiple such indices, return the leftmost one.)
- Use your `arg_min(v)` function to help you concisely extract the rows of your results data frame corresponding to the minimal RMSE values for predicting Extraversion and Neuroticism. You should find $(\alpha, \lambda) =$

$(0.1, 0.2329952)$ for Extraversion and $(\alpha, \lambda) = (0.1, 0.3393222)$ for Neuroticism.

- On the *whole dataset*, train regularized linear models for Extraversion and Neuroticism using the optimal values of (α, λ) you just determined.
- Use `coef()` to extract the coefficients associated with these regularized linear models for Extraversion and Neuroticism. Don't forget to specify a value of λ for the `s` parameter of `coef()`.
- Bind the two columns of coefficients together into a single matrix. (You'll have to coerce the outputs of `coef()` into numeric vectors first.)
  - Set the column names of the matrix equal to `c("Extraversion", "Neuroticism")` (with the two flipped if necessary) and set the row names equal to the row names of the objects returned by `coef()`.
  - Remove the top row (corresponding to the intercept term).
  - Remove rows where both coefficients are 0.
  - For each column, call `quantile()` on its absolute values to get some statistics about the distribution of coefficient magnitudes.
  - Remove every row from the matrix where *both* of the magnitudes of the coefficients fall under the 75th pecentile for their respective columns.
- Use `corrplot()` with `is.corr=FALSE` to plot the matrix of coefficients. Interpret the results.