Self-Assessment

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.
- Email `signaldatascience@gmail.com` with your R script attached when you finish.
- 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 your code and write it cleanly so I know what each part of your script is supposed to do.**
- If it's 12:30 and you haven't finished: go eat lunch, come back, and keep working after a break!

We expect this to take up to 3 hours, but we may be wrong – we don't have any calibration data, after all! We'll determine the stopping point based on people's progress through the self-assessment at lunchtime.

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

Part 1: Regularization

In Part 1, 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]

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

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).

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()`.)

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:

- 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` variable with 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.
- 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

[2]Remember `vector("list", list_length)`.

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 is the "actual" values, because $(x - y)^2 = (y - x)^2$.)
- 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.
    - 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.

- 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 take a look!
- 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 any of them.)
- 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.
- 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. Interpret the results.

Finally, take a short (5 minute) break if you've made it all the way here! Get some water, stretch, etc.

Part 2: Probability

In Part 2, we'll take a break from the usual data science problems and work on some more probability!

We **won't** be going over the theoretical solutions to these questions during class time. However:

- If you have time to spare, please do think about *why* the numerical results are what they are.
- **After 7 PM**, we'll give a short (30-minute) explanation of all three probability questions to anyone who sticks around

Hashmap collisions

From "120 Interview Questions", we have the following:

> Your hash function assigns each object to a number between 1:10, each with equal probability. With 10 objects, what is the probability of a hash collision? What is the expected number of hash collisions? What is the expected number of hashes that are unused?

Use `sample(..., replace=TRUE)` to give estimates of all three.

(*Clarification:* If n objects are assigned to the same hash, that counts as $n - 1$ hash collisions.)

Rolling the dice

Here's a nice question from one of the first cohort's students:

> Given a fair, 6-sided dice, what's the *expected number of rolls* you have to make before each number (1, 2, . . . , 6) shows up at least once?

Write code to estimate the answer.

Bobo the Amoeba

Here's a problem commonly found in quantitative finance interviews, an easier version of which sometimes appears in data science interviews:

> Bobo the amoeba can divide into 0, 1, 2, or 3 amoebas with equal probability. (Dividing into 0 means that Bobo dies.) Each of Bobo's descendants have the same probabilities. What's the probability that Bobo's lineage eventually dies out?

I'll outline a computational path for you to estimate the solution to this problem. In particular, we're going to simulate a large number of amoeba lineages over time to determine how the probability of total extinction changes as we iterate forward in time.

- Write a function `next_gen(n)` that takes in an initial number of amoebas `n`, determines how many amoebas are in the next generation according to the probability above, and returns that value. Sanity check: `next_gen(1)` should return 0, 1, 2, or 3 with equal probability.
- Note the enormous computation time required for `next_gen(n)` when `n` is very large. If there are a *large* of amoebas, we can assume (with reasonable confidence) that the lineage isn't going to die out. Pick a reasonably large value of `n`, like 500 – let's call it `N` – and modify `next_gen(n)` to just return `N+1` when `n > N`. (This is fine because we just want to know if the lineage will *die out* or not – how huge the population can get in cases where it doesn't don't really matter to us.)
- We're going to simulate `num_lineages` lineages for `n_gens` generations, so set `num_lineages = 10000` and `n_gens = 30`.
- Initialize a matrix of appropriate size and dimensions, where each column represents a single lineage of amoebas and every row represents a different generation. Next, set the initial generation to a population of 1.
- Iterate over the number of generations. For each iteration, apply `next_gen(n)` to the population of the most recent generation to get the population for the next generation, filling in the values of your population matrix.
- Turn your population matrix into a matrix of 1s and 0s corresponding to whether the lineage was still alive or died out at each step of the simulation.
- Calculate the probabilities of lineage extinction from your population matrix. *Hint:* Take the `rowSums()` of the matrix.
- `qplot()` the time evolution of the extinction probability. What do you think it is? Give a numerical estimate using your calculations.

Now, use WolframAlpha to solve the cubic equation $p = \frac{1}{4} + \frac{1}{4}p + \frac{1}{4}p^2 + \frac{1}{4}p^3$. One of the solutions will numerically correspond to your calculated probability. Which one? Why?