Self-Assessment 3

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, mail `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: `ggplot2`.

Part 1: Logistic regression

For some additional practice with logistic regression, we'll be looking at American election data from the National Election Study from 1948 through 2002.

A note on `caret`

In the following, you'll be using the `caret` package to obtain cross-validated estimates of α and λ for regularized logistic regression. Keep the following points in mind:

- Use 5-fold cross validation without any repeats.
- In order to tell `train()` that you want to perform two-class classification instead of standard regression, set `classProbs=TRUE` in the control parameters.

- In addition, in order to use area under the ROC curve as your metric of model quality, set `summaryFunction=twoClassSummary` in the control parameters and pass in `metric="ROC"` to `train()`.

Getting started

The dataset is located in the `nat-elections` directory as `elections-cleaned.dta`. Information about the data is located in `nes-glossary.txt`.

- `.dta` files are Stata data files which R cannot natively read. Load the `foreign` package and use `read.dta()` to load the dataset into R.

For your convenience, we've already cleaned the dataset by imputing missing values and properly renaming factor levels. In addition, we've restricted consideration to years with presdential elections and selected a subset of the original variables.

Exploring the data

Before you do any data analysis, it's typically a good idea to do some basic exploratory visualizations to build intuition around the dataset.

- Use `mosaicplot()` to make a couple mosaic plots from the cleaned and simplified dataset. For example, try `mosaicplot(table(df$income, df$presvote))`. Can you find any counterintuitive results?
- Considering the entire time period from 1948–2000, it there any relationship between what region a voter lives in and which presidential party they support? Is this relationship any different if you restrict to looking data from smaller timespans (*e.g.*, a single election year or 2 consecutive elections)? You can just look at a couple mosaic plots to answer this.

Analysis with logistic regression

We need to expand out the factor columns into a set of binary indicator variables in order to fit linear models.

Earlier, you learned that a factor with k levels should be expanded out into a set of $k-1$ indicator variables, because k indicator variables (one for each level) would suffer from multicollinearity. When using regularized models, we however *do* want to use k indicator variables. Here's why: intuitively, the multicollinearity arises from our being able to write one of the k indicator variables as a linear combination of the others. However, when we *regularize*, we

constrain the magnitudes of the model's coefficients and effectively overcome this problem. As such, adding in k instead of $k-1$ indicator variables for factors can improve the performance of a regularized model.

Thankfully, we don't need to write our own function to perform this expansion.

- Use `dummy.data.frame()` from the `dummies` package to create a *new* data frame with the factors expanded out into indicator variables. When calling `dummy.data.frame()`, set `sep="_"` to make the resulting column names more readable.

We're now ready to use regularized logistic regression to explore the dataset. As described in the regularization assignment, use `caret`'s `train()` function to search for the correct values of α and λ. It typically gives good initial results to search over $\alpha \in \{0, 0.1, \ldots, 1\}$ and $\lambda \in \{2^{-4}, 2^{-3}, \ldots, 2^{1}\}$; if you want further improvements, you can perform a finer grid search over a smaller range of values.

Note: In the following, `scale()`ing various subsets of the data might introduce `NAs` into the data frame because a constant column cannot be scaled (as it has standard deviation 0). Be sure to check and correct for this.

For each of the following questions, you should interpret the nonzero regression coefficients and calculate the area under the ROC curve. They are relatively open-ended; feel free to do any additional analysis which interests you or to explore questions which aren't listed here.

- Predict support for George H. W. Bush in the 1992 election. (Restrict consideration to people who actually voted!) Can you improve your model by adding interaction terms? Compare the performance of a regularized vs. unregularized model.
- For each election year, predict party support in terms of the features of the dataset. (Again, consider adding well-chosen interaction terms fo your model.) Make a data frame to store the coefficients of your model each year and graph how the coefficients have changed over time. Which demographic variables have increased or decreased the most in importance over time?
- Using a similar approach to the previous question, predict how voters who didn't vote *would have voted* each year and analyze how predicted non-voter preferences have changed over time.

Part 2: Probability

In Part 2, we'll look at computational approaches to a variety of probability-based interview questions.

Hashmap collisions

From *120 Interview Questions*, we have the following question:

> 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

Lastly, 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?

To solve this problem, we're going to simultaneously simulate a large number of amoeba lineages and incrementally step forward in time to determine how the probability of total extinction changes as we keep iterating forward.

- 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.)

- You'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. How does that polynomial relate to the problem?

Part 3: SQL