CSE199 Activity: Some Data Science Elements Fall 2017

Plotting, Correlation, Regression, and Textual Content Analysis

This activity gives a 'get-acquainted' experience with a few elemental data tools and concepts: plotting, correlation, linear regression, and quantitative analysis of large amounts of text—all using short programs and modules written in Python. No past programming experience in Python is needed and the required portion is shorter than last week's SQL activity. The goals this time are having fun playing around and discussion of data policy issues. The discussion includes what it means to build and train a *predictive model*. **Update:** This activity needs **Chrome** or **Firefox** or a downloaded Python system; the two browsers seem to work on any platform.

The first part continues the NFL teams example of last week. What factors are correlated with long playoff win droughts such as the Buffalo Bills have had since 1995? We will explore • the number of coaching changes since 1990 and • the sizes of the teams' media markets. Is coaching turnover symptomatic or causative of playoff failure? Are small-market teams starved for resources needed to win? Well, this is "small data"—there are only 32 NFL "apples" and we're not considering the "oranges" of other sports—but we can see everything going on with just 3 rows of 32 data points.

The second part belongs to a genre whose most immediately recognizable example may be the mass filtering of communications and webpages for terrorist indications and gauges of threat levels. "Hot words" and combinations of words are assigned numerical *scores* in various categories of threats and implications by the models underlying these filters. Another prominent example is the Google Ngram Viewer. It counts occurrences of N-word phrases in Google's entire collection of digitized books (searchable through 2008) but does not give scores. In the more-refined direction are various models of *stylometry* including Signature by Peter Millican of Oxford University.

We will use a recently-released "alpha" version (v0.5) of the Canadian *National Research Council Affect-Intensity Lexicon* (NRCAIL). It has almost 6,000 "intense" words scored in the general categories of *anger*, *fear*, *joy*, and *sadness*.[1] We will use it to make simple measures of the "emotional temperature" of various webpages. How to choose a set of measures based on scientific justification rather than personal whim—and to focus *what* is being *modeled*—is for discussion at the end. This also serves as a rudimentary example of "programming the Internet." Here's what's involved:

- **Homework** to complete *individually* before your recitation hour, including showing a score on the game guessthecorrelation.com to a grader in your recitation and reading the short paper "Word Affect Intensities" by Dr. Saif Mohammad of NRC Canada. Play the game *on the device you bring to recitation* because it uses cookies rather than a login.
- Allowing time for troubleshooting Python on individual machines, the *running* part is short.
- Emphasis is on discussion and interpretation, so again please form groups of 3-or-4.
- The project files are those beginning with `NFL` and with `heat` in the `https://www.cse.buffalo.edu/~regan/cse199/` folder.

1 Preparation and Python Setup

The main technical requirement is access to a working Python 3 installation. **Updating previous information**, issues with (the free demo level of) the "Python 3 Trinket" site

[1] You may recognize these from the 2015 Pixar movie *Inside Out*. Only *disgust* as played by Mindy Kaling is missing. They say they will add it soon; doing so ourselves is an optional programming topic at the end. Also note the role of Google's Ngrams in their model's foundation in the reading assignment below.

https://trinket.io/python3 have been solved **in Chrome and Firefox** on Windows and Mac platforms. *Hence please download Chrome if you have neither of those browsers* (or download and work with your own Python3 system). For simple Web use, no login or registration apparently needed:

1. Go to https://trinket.io/python3 ("Python 3 Trinket" free demo level). (Alternate page)
2. Copy and paste the `NFLtest.py` file into the window for "`main.py`"—no need to change the filename or add any new files. (Unlike with SQLizer last week, there is no file upload.)
3. Click the triangle "play" button to run.
4. After it runs, slide the vertical divider bar left to widen the output window.

Troubleshooting: If it works for you, leave it alone—often running it a second time has led to its doing nothing. If it does nothing the first time, *wait* and try again. While waiting, you can try making it load `NFLTeams.xml` directly rather than via HTTP: Hit the '`+`' sign at upper right to add a file, name it `NFLTeams.xml`, and copy and paste the XML file's contents. Then go back into the code, comment-*out* line 95, comment-*in* line 94 to use a simple filename as the location, and try again. *If it keeps twirling at upper right, try my alternate Trinket page*, which is set up for part II but can be used for the NFL part by just pasting that code over `main.py`.

If it really doesn't work (or if the site bogs down from simultaneous use during class), first try another browser. As a last resort—which you can also use as a first resort—we'll try the CSE machines or a Python download. *If Python 3 Trinket does work you can skip the rest of the section.*

Recommended Option: Setting up your own Python system

It would still ultimately be worth your while to download and install a complete Python3 system such as Anaconda (https://www.anaconda.com/distribution/) and/or learn to use `python3` on the CSE undergraduate machines if you have an account for CSE115 or another course. Directions will be given for all three, with Python 3 Trinket first and in full detail.

The libraries `html, re, sys, traceback, urllib, xml` and their child packages are standard in Python 3 (3.4 or later). The ones to check, in order of need, are `numpy`, `sklearn`, `pandas`, `matplotlib`, and `scipy`. The last two already have work-arounds in place so are not required for this activity; `numpy` is virtually standard and `pandas` can be worked around. So the onus falls on `sklearn` for linear regression—the alternatives in `scipy` or the less-common `statsmodels` packages are more complicated so please verify your access to `sklearn` beforehand. The lines invoking the problematic five packages all work in Python 3 Trinket and on the CSE machines:

```
import numpy as np
import scipy
import mathplotlib
import pandas as pd
from sklearn import linear_model
```

We've put them all in the `NFLtest.py` file so you can get a one-shot test of everything you'll probably need for other courses as well. The installation should take care of all the system paths you need so that placing `NFLtest.py` in the base folder for code and entering `python NFLtest.py` in a command window is all you need do.

On the CSE machines you can use any folder you like for work. It's simplest if you first go there and enter

```
cp ~regan/cse199/NFLtest.py .
```

to copy the file over. Then enter

```
python3 NFLtest.py
```

Or you may first load the Python 3 environment directly—as is also possible on the CSE machines by typing just `python3`. In that case, you can both load and run the file by entering (at Python's own prompt which might be `>>>`):

```
from NFLtest import *
```

Once loaded, repeating that command has no effect, but you can re-run by re-pasting lines of the top-level code under the comment `# main` beginning at or after the creation of arrays called `X`, `Y`, and `Z`. And/or, you can change those lines to use different formulas...

The CSE machines will not show the `matplotlib` color plots, nor might your home system, but both will save them as PNG pictures `XYplot.png` and `ZYplot.png` for separate viewing. My code also prints crude ASCII plots of the data points, though not the regression lines. The points are enough to get the "point," hehe. Python 3 Trinket *does* show the PNG pictures—a pleasant feature that usually takes a separate "notebook" setup to see.

2 Pre-Recitation Homework

Besides testing out Python 3, please do the following **before** your recitation. Some of it may be shown in class.

1. Play a round or two of the game "Guess the Correlation" to gain some visual context for the NFL data plots.

2. Please read the webpage "Regression with scikit.learn" (which `sklearn` further abbreviates). Be warned that the webpage leaves some Python lines incomplete on purpose and that the loading syntax is a little different from ours. You're already familiar with the XML format of the supplied data file `NFLTeams.xml` from last week.

3. For the "heat index" part, please skim-read the 7-page "Word Affect Intensities" paper. You need not understand all the technical bits but should appreciate the following points:

   - Even at professional level this kind of work is in early stages.
   - This is almost the first lexicon with numerical rather than binary (i.e. 0-1 or yes/no) data on words for general purposes.
   - The numbers were computed (to three decimal places) not by whim but by some kind of regular scientific procedure based on examining large amounts of data and using "cloud and crowd" methods.
   - The crowd-sourced numbers were checked against human annotators. They were found to correlate significantly highly with an average of human opinions. The correlation measures mentioned on page 2 are not the same as the "R^2" from the NFL part but are related.

4. Download the project code files from `https://www.cse.buffalo.edu/~regan/cse199/` via web or copy them from `~regan/cse199` on the CSE machines. In full they are:

```
NFLtest.py
NFLTeams.xml
heatlib.py
heatindex.py
NRC-AffectIntensity-Lexicon.txt   ---download or just let code access
   https://www.cse.buffalo.edu/~regan/cse199/deepweb/NRC-AffectIntensity-Lexicon.txt
```

5. Please finally skim-read the Python code. Note especially the lines with `location =` in `heatindex.py` and with `pageStr` and uses of `re.sub` in the `heatScore` function in `heatlib.py`, where there are alternatives to comment in or out.

NSFW and Copyright Notice

The NRCAIL begins with the highest scoring words for *anger* and includes slurs as well as compounds of the F-word and its ilk. Well, *you* don't have to read it—your (or our) computer will. The word `trigger` does not appear while `warning` scores 0.297 in the *fear* category.

The copyright is held by Saif M. Mohammad under the aegis of the National Research Council of Canada. We have written permission from him and his co-worker Pierre Charron to use it in CSE199. In its current v0.5 form it is even freely downloadable at its http://saifmohammad.com/WebPages/AffectIntensity.htm home page. However, the Terms-of-Use there include a directive not to *redistribute* it, and at some point it will be covered by the same EULA (please view) as the other NRC Lexicons.

Depending on possible network contention for the web-posted copy (1/1/18: link removed) and how many of you have access to Python on the CSE machines, we will encourage downloading it to private Python systems but urge it not be circulated further than there.

3 Part 1 (NFL) Activity Directions

The programming part of the activity runs in one shot—nothing more to do. Please, however, take it slow and discuss the following train of thought in a small group:

1. What factors might *cause*, *affect*, and/or be *symptomatic* of Y = long playoff droughts such as the Buffalo Bills are woefully experiencing?
2. One might be Z = the size of the local media market. The Bills are a small-market team. (Unless, that is, we can appeal to Toronto, which in-toto would jump the 2.9 million figure to over 12 million.)
3. Another could be X = high coaching turnover. It might not just be a symptom of losing seasons. Frequent changes could upset the "team chemistry" needed to win.
4. Would you expect X to be strongly correlated with Y?
5. The first linear regression tests Y against X. It gives numbers s and i (for *slope* and *intercept*) that can be interpreted as *projecting*

$$y = i + s \cdot x,$$

where x is the number of coaches any one team has had (since 1990) and y is the *predicted* playoff drought.

6. Look at the s and i for the first regression (drought vs. coaches) at the bottom. Do they make sense? Is it weird for i to be negative? (Well, for the Patriots, the whole drought is zero.)

7. Look at the R^2 figure printed at the end for the "number of coaches since 1990" run. Also look at the printed plot (the crude ASCII plot you can scroll-up for is good enough; if your system allows making a proper `matplotlib` plot, all the better) and compare with your "Guess the Correlation" experience. Would you say the correlation is strong?

8. Now examine the second regression, Y versus Z. Does it show a strong effect? any effect?

9. Eyeball the second plot too. Ummm... do the Bills have any excuse?

10. Finally, the third and last line shows the results of regressing Y against both X and Z. That is, it predicts
$$y = i + s_1 \cdot x + s_2 \cdot z,$$
with two "slope" coefficients. How did the respective slopes and the R^2 score change?

For a quick further experiment, look at the coaching-change data point for the Cincinnati Bengals. They've had only 5 coaches since 1990 despite not even making the playoffs again until 2005 and losing 7 straight first-round games since then. So let's consider Cincinnati a "bonzo outlier" (look, they even let us swipe their "Shark Girl" statue). *Scrub* their data point—either by changing the code to load `http://www.cse.buffalo.edu/ regan/cse199/NFLTeamsNoCIN.xml` or by commenting out the CIN line in your copy of the `NFLTeams.xml` file. (To do the latter, you need only change the initial `<` to `<!--` and the final `/>` to `/-->` as exemplified by other lines in the file.) Re-run. How much do the s, i, and especially the R^2 figures change?

4 Part 2 (Heat Index) Directions

Again the directions are simple. If using https://trinket.io/python3 (or the alternate Trinket page):

1. *Create a new window* in your browser to go there.

2. Copy-and-paste `heatindex.py` into "main.py" there. Again, no need to change the name `main.py` and no file upload.

3. Then click the '`+`' to add a new file, name it `heatlib.py`, and copy-and-paste the text from your own download or browser view of `heatlib.py`.

4. Click the triangle to run. You will be prompted to enter a URL. You can eithe type something like `www.cnn.com` right there or copy and paste the URL of any webpage you like. Again you can widen the output window to see lines whole.

On a home Python3 system or the CSE machines the command options are:

```
python3 heatindex.py                          or give a URL argument at the command line, say
python3 heatindex.py www.cnn.com
```

Or from within the Python environment, enter `from heatindex import *` on the first run, and if you quit the menu (but not Python) and want to restart, enter `processUserInput(url,heatDict,mulDict)`.

1. Play around with a few webpages. You can copy and paste URLs from the browser window. Twitter pages will give you the person's last yea-many Tweets.

2. Can you find a webpage with a negative score? Negative means more 'joy'—

3. Stop—! We need a "coach's huddle" before we get carried away...

The most important thing to know about the numerical results at the very end is that they are **not designed by Dr. Mohammad** but rather by me, KWR. This project design performs a "reduction" on his work. The final number comes from adding up the 'anger,' 'fear,' and 'sadness' scores and *subtracting* the total 'joy' score. Then my code divides by the total number of words read and—totally arbitrarily for better "eye candy"—multiplies that by 1,000. In spot-tests this puts the final numbers roughly on a familiar 0-to-10 scale, concentrated more near zero, and possibly negative—when 'joy' outweighs the three other categories. So it has some street sense. But it's still doing "push a button and get a number"—which is what everybody wants in order to make their jobs simpler but which comes with blinkers and dangers.

Put another way, I have slapped a layer of "multipliers" onto his model. The default multipliers pass the "Occam's Razor" test of being simple: +1 each for *anger*, *fear*, and *sadness*, and −1 for *joy*. The code allows you to change them at will, but they are still *arbitrary*—that is, not justified by theoretical considerations and/or empirical testing. They also represent some presumptions about *what* we want to model:

- They presume we are trying to model 'light'-versus-'dark' in some way.
- If we want to model *intensity* of any kind, we should use +1 for *joy* too.
- At least the +1 multipliers are leaving Dr. Mohammad's carefully-crafted numbers alone...
- But adding them up and dividing by the total number of words is still "reducing" his work. What is that ratio supposed to represent?
- If we append to the bottom of a page an equal amount of completely neutral text, we will *halve* the score. Does this make sense—shouldn't we give more weight to text at the top? Think of what "TL-DR" means. (On the other hand, a Twitter sample might be best treated as uniformly weighted over time.)
- Perhaps 'anger' should be regarded as generating more "heat" than 'fear' (which is more passive) and certainly 'sadness.' (Indeed, those who have seen *Inside Out* may recall its upshot about sadness.)
- Try multipliers of +10 for anger, +6 for fear, +2 for sadness and (really "winging it" now) −5 for joy. Repeat some earlier URLs. Do any scores shift notably? (Note: Just doubling the multipliers will not double the score, because the code has a further "normalizing" division by the sum of the absolute values of the multipliers.)
- Try other multipliers too. Each combo constitutes a different *model* of "webpage intensity." Then reflect on the most important question:

From the myriad parameter combinations, how can we determine **one** that is *correct*, *best*, or at least *neutrally justifiable*? In particular, how could we *train* the model?

Answering this question first requires determining what we want to model and what the purpose is. Suppose we adopt the "light-versus-dark" purpose. The first thing we might try is to identify a corpus of pages that represent the "neutral" middle. Then we want to define our scale and train our multipliers so that those pages get a score of 0. This still does only a quarter of the work we need—in technical jargon, it leaves three "degrees of freedom" in the four parameters we are fitting.

Further progress needs asking, what are we trying to *predict*? Here we might come full circle to the "threat levels" application mentioned at the outset. Suppose we have a corpus of pages that were reliably associated with the same threat level in past history. Some pages might have more 'anger,' others more 'fear' and so on. We can train our multipliers to equalize those pages.

A general methodology emerges from the idea that how morose or bubbly a webpage written today is can predict how morose or bubbly a page written by the same person(s) tomorrow will be. Or we can apply this prediction idea from sentence to sentence or phrase to phrase. Now we are in the world of those N-grams. Predicting a probability range on the nature of the $(N+1)$st item from the previous N in a continuing series involves building a so-called *Markov model* (or *Markov chain*, named for the Russian mathematician Andrey Andreyevich Markov). The Markov model M can use the principle that words continuing the score trend of the previous N are most likely. Its results thus depend on the scores and hence on the multipliers we choose. The likelihood principle in this context yields the training policy:

> Find the combination of parameters that maximizes the probability that M projects for the record of what actually happened.

This sounds like "making yourself look good by predicting the past" but the power of this principle carries into the future. It still, however, takes quite a bit of work to build M and then some computer muscle to carry out the *maximum likelihood estimation* (MLE).

5 Final Discussion Points

If you think of Data Science as 'sexy' then it has been my purpose to spend a lot of time metaphorically on safety and STDs—and even literally on matters of personal consent and privacy. Instead of brilliant results with clear correlations, we've seen:

- Inconclusive results from small data (when that is all the data that is);
- How to "cheat" with Cincinnati—or rather, how sensitive results can be to one data point;
- How quantities for which there is obvious and immediate demand are currently *inchoate*;
- How the desire for simplicity induces arbitrary choices that can be fraught with biases;
- How fixing these issues requires lots of attentive hard work.

6 More Fiddling Around (optional)

Note that the Python code is pretty 'modular' not only in the library module `heatlib.py` but even in the client file—it avoids referencing the four categories by name as much as possible. Thus it is ready to handle updates of the lexicon to include other categories. Try this yourself by adding the category *disgust* and a dozen words like `gross` and `yuck` and `nausea` right to the lexicon file, giving them scores at-your-whim. Make the few edits needed to `heatindex.py` and see the results.