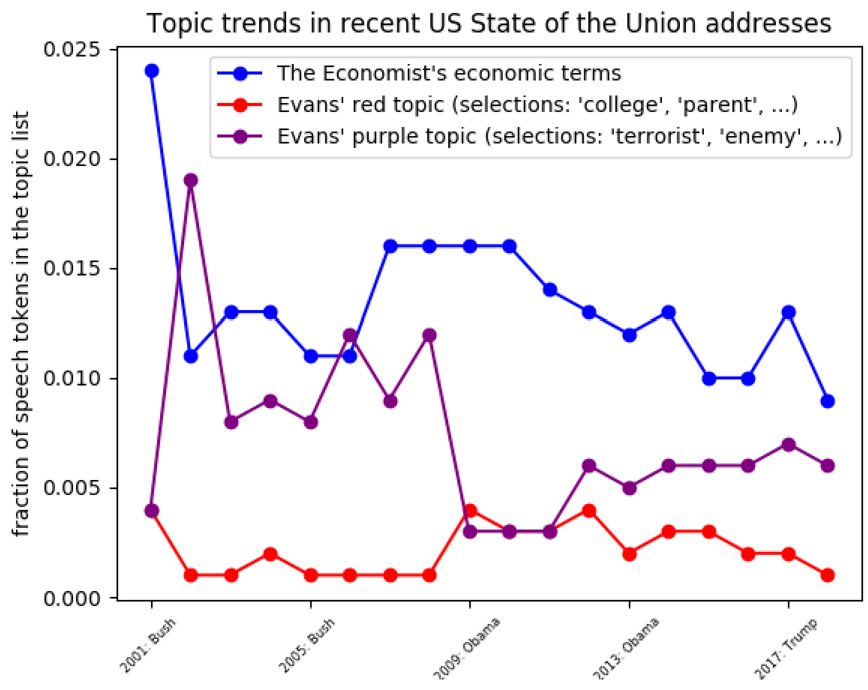
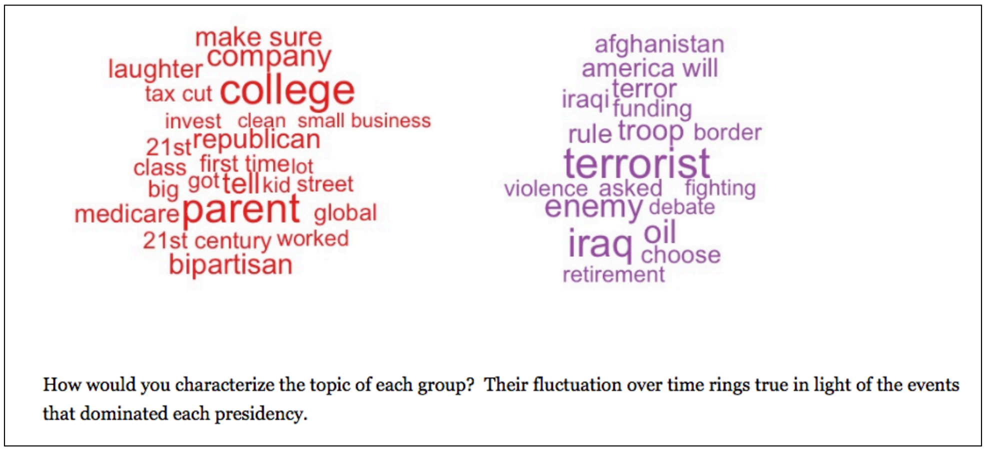
CS 1110 Spring 2018, Assignment 3: Topic tracking in political speeches*
March 22, 2018

1 Motivation, or, "It's the economy, stupid"

The assignment is motivated by an interdisciplinary question: *Are there significant trends in political discourse over time*? We narrow down this big question to consider trends in three specific topics over recent U.S. Presidential State of the Union addresses. When you complete this assignment, your code will be able to generate this figure:



where the "economics" topic was created by processing webpages posted by *The Economist*[1], and the "red" and "purple" topics were manually drawn from Topic Modeling the State of the Union: TV and Partisanship, Frank D. Evans Jan 10, 2016[2]:



How would you characterize the topic of each group? Their fluctuation over time rings true in light of the events that dominated each presidency.

*Authors: Lillian Lee. Thanks to Victoria Litvinova, Cristian Danescu-Niculescu-Mizil, and Jon Kleinberg for helpful discussions.
[1] And you know how to do that from Assignment A1! Go, you!
[2] https://www.exaptive.com/blog/topic-modeling-the-state-of-the-union. Figure is a cropped screenshot

Contents

2 Rules (all the same as for previous assignment)

There is no revise-and-resubmit for this or any subsequent assignment unless otherwise noted.

2.1 You need to re-group (so to speak) on CMS, regardless of prior groupings

You may work alone or with just one other person, who can be someone you've grouped with before in CS1110, or a different person.

If you are partnering, regardless of whether you were grouped in a previous assignment, the two of you must still form a new group *specifically for this assignment* on CMS before submitting.[3]

If your partnership dissolves, see the course Academic Integrity description about "group divorce" for what to do.

2.2 (Dis-)allowed collaborations and documentation requirements

Our policies are laid out in full on the course Academic Integrity page, but we re-state here **the main rules**: where "you" means you and, if there is one, your one CMS-registered group partner,

1. Never look at, access or possess any portion of another group's work in any form.
2. Never show or share any portion of your work in any form to anyone except a member of the course staff.
3. Never request solutions from outside sources; for example, on online services like StackOverflow.
4. DO specifically acknowledge by name all help you received, whether or not it was "legal" according to (1)-(3).

3 Meta-commentary and strategies (TODO: EoD Friday Mar 23)

Chances are you may find yourself a bit intimidated when browsing the skeleton code: there are some complexities and subtleties. Understanding the following points should help make this project more manageable and more educational.

Throughout, we'll use the notation TODO: to indicate target milestone deadlines. "EoD" stands for "end of day".

1. At heart, the educational goals for A3 are (1) for-loop practice, both with iterating over list values and iterating over list indices; (2) exemplifying the design process for a larger scale task (see Section 5); (3) careful consideration of specifications and test cases; (4) debugging for a subtle(-ish) task Regarding for-loops:

[3]This links your submission "portals".

(a) TRY: EOD Friday Mar 23. Have lab 08 completed.

(b) Good references: Ch. 9, 10.7, and 13 of the text; lecture 11's files and added writeup; the completed for-loops we've given you in the A3 code; 2017 Spring's A3 solutions, a Python 3 version of which are posted at this semester's Assignments page. Regarding specifications: we cannot stress enough the importance of first solidifying one's grasp of the task, in part by writing test cases based on the specification *before* implementation. This assignment hammers this home by **including purposely incorrect test cases in the test functions we've given you**:[4] you will need to examine the specs and our test code carefully in order to fix those bugs.

Regarding debugging for a challenging task:

(a) Many bugs are caused by unintentionally changing the semantics of a variable. Pick informative variable names and/or comment what your intents are. Make sure you update variable values correctly when the situation changes.[5]

(b) Section 13.10 of the text ("...especially if you are working on a hard bug") is good advice.

(c) Only implement a little bit at a time and test incessantly. Add temporary print statement to check your partial progress as necessary. You don't want an uncaught bug early one messing up a lot of things downstream.

2. Navigating complex files:

   (a) You can "fold" unction bodies in Komodo Edit to temporarily hide them. Click just to the left of `def` lines to cause this fold to happen. You may have your Editor→Smart Editing→Folding preferences set to "Use horizontal line on folds", which is useful. Click on such horizontal lines to unfold code.

   (b) When you've got the first two or three functions working, switch from testing via `python a3test.py` on the command line to `python a3test.py quiet` to speed through the tests functions you've already passed.

3. You *can* complete this assignment without taking the time to understand the bigger picture, such as the code we provided for you. If pressed for time, the bare minimum would be to only look at what is marked **STUDENTS:** in the code.

4. Other aspects of the code introduce new techniques or bigger examples of material from this class. Explore at your leisure. We may provide more documentation later. Some highlights:

   (a) Real-life recursive function, exemplifying important helper-function pattern: a3.get_econ_vocab_helper(), called by a3.get_econ_vocab()

   (b) use of if __name__== '__main__': depending on whether a3.py is treated as a module or a script

   (c) command-line argument passed to various functions, in this case, to minimize the output a3test.py when you've got most of your code working: sys.argv in script code of a3test.py

   (d) file opening and writing operations: a3.get_content_lines and a3.downlaod_econ_vocab_data()

   (e) for-loop over characters in a string for a real-life problem: a3.download_econ_vocab_data()

4 Add AND **FIX** test cases (TODO: mid-day Mon Mar 26)

Although implementing the conversion functions (described in the next section) may well be challenging, the *fixing of intentionally incorrect test cases* might be the part of this assignment that really throws you for a loop (ha). We are stressing that you should do this *first*, both because it's important, and because fixing given test cases may be an unfamiliar task to you.[6]

[4] There is the possibility that we also have *unintentional* errors, so please ask us if you aren't sure about any cases!

[5] Example: in a3.get_econ_vocab_helper(), there's a variable called work_text. When the first part of work_text is processed, I update work_text to chop off the completed part, so that the "meaning" of the variable is once again, "all and only the stuff that still needs doing."

[6] In which case — assuming you do any testing at all — I deeply, seriously envy you.

5 Code and files organization

Download the zip file http://www.cs.cornell.edu/courses/cs1110/2018sp/assignments/assignment3/a3assignmentfiles.zip into a new directory and unzip it. *Warning for windows users*: you may to move the files out of original Explorer window and into another folder.

Remember that our goal is to look at topic trends in political discourse. So, a3.py contains code to do so. a3test.py exists to test the functions in a3.py.

The directory sources contains text of recent US Presidential State of the Union (SOTU) addresses taken with almost no preprocessing from The UCSB American Presidency Project[7] It also contains small files useful for testing on; examples are lonely_as_a_cloud.txt, imbeciles.txt,[8] shortest.txt. Some of these files contain *comments*, marked by a preceding '#', that for our purposes are not to be considered part of the content of the files.

Back to a3.py. We want to track topics over a series of speeches or utterances. That's the job of the main a3 function track_topic(docs_list, vocab_list). Getting the right arguments means two main subtasks.

5.1 Create a list of documents

Create a list of documentsdocs_list from some files. Since we know about string processing, it's convenient for each document to be represented as a single string.

Importantly, text files are typically structures as sequences of lines, so that's the kind of input we must consider. Let's call such input a linelist, which is created from a file by get_content_lines().

5.1.1 Converting all a file's lines into a single string (TODO: EoD Mon Mar 26)

If we want to track topics over documents represented as different files, then we don't need to do any special treatment of particular lines within linelist, we just need to merge them together into a single string. That's the job of convert_lines_to_string(linelist) and convert_lines_to_string2(linelist) — we're asking to implement the same functionality with two different types of for-loops.

5.1.2 Blank lines treated as paragraph breaks (TODO: EoD Thu Mar 29)

If we want to track topics over paragraphs within a given speech, then we choose to recognize paragraph breaks as special, blank lines. That's the job of convert_lines_to_paragraphs().

5.2 (OPTIONAL READING) Create a vocabulary list

This was sufficiently complicated that we did all of this for you. So, if pressed for time, you can skip this section. How do we get a list of words having to do with economics topics? Some web-searches[9] reveal that The Economist's web-pages have well-structured webpages that define terms, and the URLs for these webpages are amenable to automatic processing; for example, `https://www.economist.com/economics-a-to-z/a` defines "antitrust", "arbitrage", and so on, and `https://www.economist.com/economics-a-to-z/u` defines "unemployment", "utility", and so forth.

But, it would be bad on a number of fronts if all CS1110 students were frequently hitting those webpages, so we wrote a function , download_econ_vocab_data() that would store the contents of those Economist webpages in a file on one's local drive.

The processing of this data can be done (arguably elegantly) recursively. We do this via main function get_econ_vocab() which calls recursive helper get_econ_vocab_helper(). As we delve more into recursion in this class, you may find it helpful to study those two functions.

Once we have collected a list of economic terms from this data, we copied that list in a module econ_terms in directory sources so that no one would need to waste time re-computing it again.

5.3 Compute the topic statistics (TODO: EoD Fri Mar 30, keeping in mind the 2pm checkpoint)

If you have all the other functions done, completing track_topic() should feel pretty straightforward.

[7]Led by John Woolley and Gerhard Peters. `http://www.presidency.ucsb.edu/sou.php`. Italicized material has been retained, even though it often doesn't strictly reflect what was said by the President: I thought it might be interesting to trace, say, the occurrence of laughter, which is marked *[Laughter]* on the webpages. Other stray material might be in the files, too

[8]There's some comments in the file about its interesting origins. Find out about the Workshop on Potential Literature.

[9]I guess you could claim this is the "natural cleverness" part of "artificial intelligence".

5.4 OPTIONAL, but for fun and satisfaction: Recreate the graph on the first page

Once completed, `a3.py` can itself be run as a script. Try `python a3.py` on the command line.

6 Code cleanup and submission checklist

Before submitting, ensure your code obeys the following.[10]

- Lines are short enough (˜80 characters) that horizontal scrolling is not necessary. (We've sometimes violated this for readability)
- You have indented with spaces, not tabs (this is not an issue if using Komodo Edit).
- You have removed any debugging `print` statements.
- You have removed all `pass` statements that were provided with the initial code.
- You have removed "instruction" comments, typically marked "`# STUDENTS:`".
- If you added any helper functions, these have good docstring specifications and you have put sufficient testing code for your functions.

Make sure the following are all true before you submit.

1. You (and your partner) have included your names and NetIDs in the header of all files.
2. You've also indicated in the file headers the names of any (non-staff) people whose help should be acknowledged.
3. The date in the header comments has been changed to when the files were last edited.
4. You have set your CMS notifications settings to receive email regarding grade changes, and regarding group invitations.
5. (reminder) If working with a partner, you have grouped on CMS. (One has invited on CMS, and one has accepted on CMS.)

6.1 Due dates

1. If you are partnering: well before submission, follow the instructions in the "How to form a group" instructions on the course Assignments page. Both parties need to act on CMS in order for the grouping to take effect.
2. By 2pm on Fri Mar 30, submit whatever you have done at that point to CMS, following steps 1-3 in the "Updating, verifying, and documenting assignment submission" section of the course Assignments page. It is OK if you haven't finished working on the files yet.
3. By **11:59pm on Fri Mar 30**, make your final submission, again following the aforementioned steps 1-3.[11]

[10] One reason for these requirements is that they speed up the process of reading hundreds of files.

[11] The 2pm checkpoint on Fri Mar 30 provides you a chance to alert us during business hours if any problems arise. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time, for whatever reason. There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy. Of course, if some special circumstances arise, contact the instructor(s) immediately.