Problem Set 3

Due Wednesday Oct. 2, 10 am

Comments

- This covers material in Unit 5.
- It's due at 10 am (Pacific) on Wednesday October 2, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please see PS1 for formatting and attribution requirements.

Problems

1. Let's investigate the structure of the `statsmodels` package to get some experience with the structure of a large Python package and with how `import` and the `__init__.py` file(s) are used. You'll need to go into the `statsmodels` source code (see Unit 5). Also note that the following cases may involve functions, classes, and class methods. Be sure to be clear to say which of those you are talking about and if it's a class, describe any inheritance structure.

    a. For this subpart only, consider doing `import statsmodels`. What is in the `statsmodels` namespace that is created? Where (what module file) is the version number for statsmodels stored in? What is the absolute path to the package on the machine you are working on?
    b. The remaining subparts all relate to using the standard `import statsmodels.api as sm` invocation. First, describe briefly what happens when this is run (what files are accessed). Then, describe what kind of object `MICE` is, how it is imported and where it is found. Do the same for `GLM`.
    c. Consider `sm.gam`. What is in the namespace? Describe how the importing works and in what modules the objects in the namespace are defined.
    d. Consider `sm.distributions.monotone_fn_inverter`. What is it, how it is imported and what file it is defined in?

    Hints:

    i. `grep -R <pattern> <directory>` will search all files within a directory recursively.
    ii. As you work on this, you may want to be able to modify one or more of the `__init__.py` or other files to better understand what is happening (e.g., by commenting out a line of code or adding a print statement). A good way to do this is to create a Conda environment in which `statsmodels` is installed, so you isolate any changes you make, e.g., `conda create -n test_env python=3.12 statsmodels`. Then you can edit code files in the environment and when you start Python and import statsmodels, you should see the effects of your

changes. Alternatively, you could use the debugger to set breakpoint(s) in a file in the package.

2. The website Commission on Presidential Debates has the text from recent debates between the candidates for President of the United States. (As a bit of background for those of you not familiar with the US political system, there are usually three debates between the Republican and Democratic candidates at which they are asked questions so that US voters can determine which candidate they would like to vote for.) Your task is to process the information and produce data on the debates. Note that while I present the problem below as subparts (a)-(d), your solution does not need to be divided into subparts in the same way, but you do need to make clear in your solution where and how you are doing what. For the purposes of this problem, please work on the the debates I've selected (see code below) for the years 2000, 2004, 2008, 2012, 2016, and 2020. (I've tried to select debates that cover domestic policy in whole or in part to control one source of variation, namely the topic of the debate.) I'll call each individual response by a candidate to a question a "chunk". A chunk might just be a few words or might be multiple paragraphs.

   The goal of this problem is two-fold: first to give you practice with regular expressions and string processing and the second to have you thinking about writing well-structured, readable code (similar to question 4 of PS1). You can choose to use either a functional programming approach or an object-oriented approach. I strongly recommend that you use the approach that you are **less** familiar with so as to gain more experience. Please think about writing short, modular functions or methods. Explore the use of `map`, list comprehension or other techniques to avoid having a lot of nested for loops. Think carefully about how to structure your objects to store the spoken chunks so that the structure works well with your functions/methods. Note that for this problem, for the sake of time, you do not need extensive docstrings, but it should still be clear what each function does. In parts (a)-(c), add simple sanity checks that you are getting reasonable results.

   Given that in earlier problem sets, you already worked on downloading and processing HTML, I'm giving you the code (in the file `ps/ps3prob3.py` in the class repository) to download the HTML and do some initial processing, so you can dive right into processing the actual debate text.

   a. Convert the text so that for each debate, the spoken text is split up into individual chunks of text spoken by each speaker (including the moderator). If there are two chunks in a row spoken by a candidate, combine them into a single chunk. Make sure that any formatting and non-spoken text (e.g., the tags for 'Laughter' and 'Applause') is stripped out. Report the number of chunks per speaker.

   b. Extract the individual words. To do the extraction I suggest you use existing Python natural language processing packages that can "tokenize" text, but you're also welcome to use regular expressions, particularly if you want to practice them more. Doing this with 100% accuracy will be hard (particularly for spoken text). Try to handle various issues, but don't try to be perfect.

   c. For each candidate, for each debate, count the number of words and characters and compute the average word length for each candidate. Compare these between candidates and over time in some basic fashion.

d. Do some checking of your functions on simple test inputs. For the sake of time, you don't need to set up formal testing, and this doesn't need to be extensive.

e. (Extra credit) For each candidate, count the following words or word stems (feel free to replace some or all of these with words or expressions of your choice) and store in a data structure: I, we, America{,n}, democra{cy,tic}, republic, Democrat{,ic}, Republican, free{,dom}, terror{,ism}, safe{,r,st,ty}, {Jesus, Christ, Christian}. Make a plot or two and comment briefly on the results.

3. Now sketch out a design for a functional programming (FP) approach (if your solution to Problem 2 used OOP) or an OOP approach (if your solution to Problem 2 used functional programming). If you're designing an OOP approach, decide what the classes would be and the fields and methods of those classes. If you're designing a FP approach, decide what the functions would be and what inputs/output they would use. **To be clear, you do not have to write any of the code for the methods/classes/functions; the idea is just to design the code.** As your response in the OOP case, for each class, please provide a bulleted list of methods and bulleted list of fields and for each item briefly comment what the purpose is. Or in the FP case, for each function, provide a bulleted list of inputs and output and briefly comment on the purpose of each function.