Homework 3: Working with Files

Due February 7, 11:59 pm

Worth 10 points

Read this first. A few things to bring to your attention:

1. **Important:** If you have not already done so, please request a Flux Hadoop account. Instructions for doing this can be found on Canvas.
2. Start early! If you run into trouble installing things or importing packages, it's best to find those problems well in advance, not the night before your assignment is due when we cannot help you!
3. **Make sure you back up your work!** I recommend, at a minimum, doing your work in a Dropbox folder or, better yet, using `git`, which is well worth your time and effort to learn.

Instructions on writing and submitting your homework.

Failure to follow these instructions will result in lost points. Your homework should be written in a jupyter notebook file. I have made a template available on Canvas, and on the course website at `http://www-personal.umich.edu/~klevin/teaching/Winter2018/STATS701/hw_template.ipynb`. You will submit, via Canvas, a `.zip` file called `yourUniqueName_hwX.zip`, where `X` is the homework number. So, if I were to hand in a file for homework 1, it would be called `klevin_hw1.zip`. Contact the instructor or your GSI if you have trouble creating such a file.

When I extract your compressed file, the result should be a directory, also called `yourUniqueName_hwX`. In that directory, at a minimum, should be a jupyter notebook file, called `yourUniqueName.hwX.ipynb`, where again `X` is the number of the current homework. You should feel free to define supplementary functions in other Python scripts, which you should include in your compressed directory. So, for example, if the code in your notebook file imports a function from a Python file called `supplementary.py`, then the file `supplementary.py` should be included in your submission. In short, I should be able to extract your archived file and run your notebook file on my own machine. Please include all of your code for all problems in the homework in a single Python notebook unless instructed otherwise, and please include in your notebook file a list of any and all people with whom you discussed this homework assignment. Please also include an estimate of how many hours you spent on each of the three sections of this homework assignment.

These instructions can also be found on the course webpage at `http://www-personal.umich.edu/~klevin/teaching/Winter2018/STATS701/hw_instructions.html`. Please direct any questions to either the instructor or your GSI.

1 More Fun with Tuples (3 points)

In this problem, you'll do a bit more with tuples.

1. You may recall that the functions `min` and `max` take any (positive) number of arguments, but that `sum` does not behave similarly. Write a function called `my_sum` that takes any number of numeric (ints and floats) arguments, and returns the sum of its arguments. Your function should correctly handle the case of zero arguments. You need not perform any error checking for this function. **Reminder:** by convention, an empty sum is taken to be 0.

2. Write a function called `reverse_tuple` that takes a tuple as its only argument and returns a tuple that is the reverse of the input. That is, the output should have as its first entry the last entry of the input, the second entry of the output should be the second-to-last entry of the input, and so on. You need not perform any error checking for this function.

3. Write a function called `rotate_tuple` that takes two arguments: a tuple and an integer, in that order. Letting n be the integer given in the input, your function should return a tuple of the same length as the input tuple, but with its entries "rotated" by n. If n is positive, this should mean to "push forward" all the entries of the input tuple by n entries, with entries that "go off the end" of the tuple being wrapped around to the beginning, so that the i-th entry of the input tuple becomes the $(i+n)$-th entry of the output, wrapping around to the beginning of the tuple if this index goes off the end. It should be clear that if n is negative, then this merely corresponds to rotating the entries in the other direction, with entries of the input tuple being "pushed backward". Your function should perform error checking to ensure that the inputs are of appropriate types. If the user supplies a non-integer, print a message to alert the user that the input was not as expected, and try to recover by casting it to an integer. **Hint:** a try/catch statement will likely be useful here.

2 More Fun with Vectors (3 points)

In your previous homework assignment, you implemented matrix and vector operations using tuples to represent vectors. In many applications, it is common to have vectors of dimension in the thousands or millions, but in which only a small fraction of the entries are nonzero. Such vectors are called *sparse* vectors, and if we tried to represent them as tuples, we would be using thousands of entries just to store zeros, which would quickly get out of hand if we needed to store hundreds or thousands of such vectors.

A reasonable solution is to instead represent a sparse vector (or matrix) by only storing its non-zero entries, with (index, value) pairs. We will take this approach in this problem, and represent vectors as dictionaries with positive integer keys (so we index into our vectors starting from 1, just like in MATLAB and R). A *valid* sparse vector will be a dictionary that has the properties that (1) all its indices are positive integers, and (2) all its values are floats.

1. Write a function `is_valid_sparse_vector` that takes one argument, and returns `True` if and only if the input is a valid sparse vector, and returns `False` otherwise. **Note:** your function should *not* assume that the input is a dictionary.

2. Write a function `sparse_inner_product` that takes two "sparse vectors" as its inputs, and returns a float that is the value of the inner product of the vectors that the inputs represent. Your function should raise an appropriate error in the event that either of the inputs is not a valid sparse vector.

   Note: This may be your first foray into *algorithm design*, so here's something I'd like you to think about: there are several distinct ways to perform this inner product operation, depending on how one chooses to iterate over the entries of the two dictionaries. For this specific problem, it doesn't much matter which you choose, but there is an important point that you should consider: if the indices of our vectors were sorted, there would be an especially fast way to perform this operation that would require that we look at each entry of the two vectors at most once. Unfortunately, there is no guarantee about order of dictionary keys, so we can't take advantage of this fact, but we'll come back to it. You **do not** need to write anything about this, but please give it some thought.

3 Counting Word Bigrams (4 points)

In your previous homework, you wrote a function for counting character bigrams. Now, let's write a function for counting word bigrams. That is, for each pair of words, say, `cat` and `dog`, we want to count how many times the word "cat" occurred immediately before the word "dog". We will represent this bigram by a tuple, `('cat', 'dog')`. For our purposes, we will ignore all spaces, newlines, punctuation and capitalization in our counting. So, as an example, the fragment of poem,

> Half a league, half a league,
> Half a league onward,
> All in the valley of Death
> Rode the six hundred.

includes the bigrams `('half', 'a')` and `('a', 'league')` both three times, the bigram `('league', 'half')` appears twice,while the bigram `('in', 'the')` appears only once.

1. Write a function `count_bigrams_in_file` that takes a filename as its only argument. Your function should read from the given file, and return a dictionary whose keys are bigrams (given in the tuple form above), and values are the counts for those bigrams. Again, your function should ignore punctuation, spaces, newlines and capitalization. The strings in your key tuples should be lower-case. Your function should use a try-catch statement to raise an error with an appropriate message to alert the user in the event that the given file cannot be opened, and a different error in the event that the provided argument isn't a string at all. **Hint:** you will find the Python function `str.strip()`, along with the string constants defined in the string documentation (`https://docs.python.org/3/library/string.html`), useful in removing punctuation. **Hint:** be careful to check that your function handles newlines correctly. For example, in the poem above, one of the `('league', 'half')` bigrams spans a newline, but should be counted nonetheless. **Note:** be careful that your function does not accidentally count the empty string as a word (this is a common bug if you aren't careful about splitting the input text). Solutions that merely delete "bad" keys from the dictionary at the end will not receive full credit, as all edge cases should handled by correctly splitting the input.

2. Download the file `mobydick.txt` from the course webpage: `http://www-personal.umich.edu/~klevin/teaching/Winter2018/STATS701/mobydick.txt`. Run your function on this file, and pickle the resulting dictionary in a file called `mb.bigrams.pickle`. Please include this file in your submission, along with `mobydick.txt`, so that we can run your notebook directly from your submission.

3. We say that word A is *collocated* with word B in a text if words A and B occur immediately one after another (in either order). That is, words A and B are collocated if and only if either of the tuples `(A, B)` or `(B, A)` are present in the text. Write a function `collocations` that takes a filename as its only argument and returns a dictionary. Your function should read from the given file (raising an appropriate error if the file cannot be opened or if the argument isn't a string at all) and return a dictionary whose keys are all the strings appearing in the file (again ignoring case and stripping away all spaces, newlines and punctuation) and the value of word A is a Python set containing all the words collocated with A. Again using the poem fragment above as an example, the string `'league'` should appear as a key, and should have as its value the set `{'a', 'half', 'onward'}`, while the string `'in'` should have the set `{'all', 'the'}` as its value. **Hint:** we didn't discuss Python sets in lecture, because they are essentially just dictionaries without values. See the documentation at `https://docs.python.org/3/tutorial/datastructures.html#sets` for more information.

4. Run your function on the file `mobydick.txt` and pickle the resulting dictionary in a file called `mb.colloc.pickle`. Please include this file in your submission.