Updates to Assignment 3, Text Analysis

The assignment itself, with updates marked in orange, begins on the next page. On this "page 0", we also document the time, location, and nature of the updates, in reverse chronological order,

Sun Mar 26, 11:30pm: in `a3.py`, the commented line 225 in `trigram_generation` should read "(2) do something to get w3 added to your output", not `w2`.

Thu Mar 23, 11:00pm:

1. Section 3.1, page 2: Explain explicitly how to create a new Sample.
2. Section 3.4, page 4: Clarification and addition of a hint on how to pick a starting bigram.

CS 1110 Spring 2017, Assignment 3: Text Analysis*

Due on CMS **Thursday March 30th, 2017, 11:59pm**. As usual, we strongly recommend that you submit a preliminary version by 2pm the day deadline, for the usual reasons.[1]

We've included suggested "milestone" deadlines throughout.

Necessary files: download and unzip http://www.cs.cornell.edu/courses/cs1110/2017sp/assignments/assignment3/a3.zip

1 Introduction

You'll be writing code that analyzes text samples, and can produce output like this from real speech transcripts, such as U.S. State of the Union (SOTU) addresses:

```
****Analyzing differences between Obama SOTU 2013-2016 and Bush SOTU 2005-2008****

Only said by Obama (and at least 10 times):  ['folks', 'wage', 'cant', 'workforce', 'manufacturing',
'crisis', 'everybody', 'applause', 'minimum', 'planet', 'race', 'fact', 'doesnt']

Only said by Bush:  (and at least 10 times):  ['palestinian', 'victory', 'given', 'struggle', 'iraqi',
'iraqis', 'empower', 'enemy']

Ten most Obama-leaning joint words:  [['lets', 3.921], ['thats', 3.801], ['kids', 2.929], ['class',
2.838], ['republicans', 2.502], ['place', 2.502], ['worked', 2.433], ['student', 2.433], ['wont',
2.359], ['dont', 2.359]]

Ten most Bush-leaning joint words:  [['terror', -2.947], ['personal', -2.509], ['offensive', -2.286],
['liberty', -3.251], ['institutions', -2.404], ['honor', -2.914], ['grateful', -2.404], ['extremists',
-2.845], ['elections', -2.404], ['duty', -2.691]]
```

For example, this means that Obama used the word "folks" (possibly capitalized) at least 10 times in the listed SOTU addresses, whereas Bush never used that word at all in his listed SOTUs; whereas Bush used the word "palestinian" at least 10 times in his listed SOTUs, but it never occurs in the Obama speeches we collected. The word "lets" (with apostrophe removed) is used by both Obama and Bush's SOTUs, but much more in Obama's; the reverse is true of the word "terror".

It's also possible to use text samples as a *language model* to generate new text. For example, suppose we were to tire of writing new CS1110 exams. We might take the text from the 2014 spring exam and from it create a new exam automatically[2]:

> the average of a titem2 with gpa contribution 8 and id2 be the id of a list of titem2s threshold is a string representing a nonempty list each item either an int or a float or int highlist and lowlist are possibly nonempty lists for an example see text at the bottom of this page make sure you see this line and indent relative to it for item in sourcelist illustrative example let id1 be the id of a titem2 as its number of credits times its gradeval for example for the titem2 created by the length of the values in the diagrams below you are their grader please mark up each student's solution as follows 1 draw an x through anything that is present that should not be 2 circle anything that is missing you may wish to do this question by first drawing the relevant frames and objects yourself 3 11 points implement redact and after so that they meet their specifications def redacts returns a copy of string s where the all but the first occurrence of c examples after aloha a3 loh after aloha a3 loh after aloha a4 loha after bananaphone n 6 anapho 4 13 points assume that inside a module named transcript2 is the sum of

*Authors: Lillian Lee, Erik Andersen

[1] Reminders: you can replace older submissions with improved ones up to the actual deadline. 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 — by CMS's definition of time, which may differ from your watch's or your computer's — for whatever reason, including server delays stemming from many other students trying to submit at the same time.

[2] Albeit one that reads like the author was Lucky from *Waiting for Godot*.

the room when asked to write python code on this exam you may wish to do this question by first drawing the relevant frames and objects yourself 3 11 points implement redact and after so that they meet their specifications def redacts returns a copy of s that starts after the first page of the room when asked to write python code on this exam with students who are scheduled to take a later makeup academic integrity is expected of all students of cornell university at all

Contents

1 Introduction **1**

2 Rules (All The Same As For A2) **2**

3 Your Goal: Complete a3.py So That a3checks.py Gives Correct Output **2**

3.1 The class Sample in a3given.py . . . 2

3.1.1 TODO (aim for end of the day Friday): Complete `make_sample` in a3.py . . . 3

3.1.2 Debugging Hints . . . 3

3.2 TODO (aim for end of the day Friday or Monday): Complete `merge` in a3.py . . . 3

3.3 TODO (aim for end of the day Monday): Complete `diffs()` . . . 4

3.4 TODO: Complete the Generation Functions . . . 4

3.4.1 If You Are Running Low On Time, De-Prioritize The Trigram Problem . . . 4

4 Before You Submit **4**

2 Rules (All The Same As For A2)

This assignment, and all subsequent ones unless otherwise noted, will receive a single grade; there is no revise-and-resubmit like there was for Assignment 1.

You may do this assignment in groups of size **one or two**. All previous groups have been dissolved by CMS for this assignment; if you want to work with someone (whether or not you've worked with them before), you need to *form your group on CMS* before submitting.[3]

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

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 Your Goal: Complete a3.py So That a3checks.py Gives Correct Output

We have provided the output of our a3checks.py incorporating our solutions in the file A3_sample.txt so that you can check your output against ours.

3.1 The class Sample in a3given.py

We've provided class definitions and useful functions in a3given.py . Most important is the class Sample, which encodes information in a given piece of text.

First, read the class specification carefully. (The specification for `__init__` means that to create a new Sample, you use a constructor call with no arguments; for example, if you are in file a3.py and have an import statement `import a3given`, then here is such a constructor call: `a3given.Sample()`[4]. This new Sample will have `label`, `text`, `unigram_counts`, `bigram_dict`, `trigram_dict` all set to the empty string/list/dictionary, and `length` set to 0.)

[3] Reminder: This requires actions on the part of both parties. Brief instructions for how to form a CMS group are on the course Assignment page.

[4] We mention this because we haven't explained `__init__` methods in lecture yet.

Here is an example. Suppose we had the following 30-word string — which is the contents of the file short.txt in the directory sources in the zip file we've provided you, after being stripped of punctuation and lowercased, a.k.a. downcased:

> here is a small piece of text a small piece of text a small piece of text here is a small piece of text at nine oclock in the morning

Then, what we expect if you created a Sample `s` with label `"short"` for this string is that the attributes of `s` would be as follows:

- `s.label`: the string `"short"`
- `s.text`: the list `['here', 'is', 'a', 'small', 'piece', 'of', 'text', 'a', 'small', 'piece', 'of', 'text', 'a', 'small', 'piece', 'of', 'text', 'here', 'is', 'a', 'small', 'piece', 'of', 'text', 'at', 'nine', 'oclock', 'in', 'the', 'morning']`
- `s.unigram_counts`: the dictionary `{'a': 4, 'of': 4, 'is': 2, 'in': 1, 'here': 2, 'morning': 1, 'oclock': 1, 'nine': 1, 'at': 1, 'text': 4, 'small': 4, 'the': 1, 'piece': 4}`
- `s.bigram_dict`: the dictionary `{'a': ['small', 'small', 'small', 'small'], 'of': ['text', 'text', 'text', 'text'], 'is': ['a', 'a'], 'in': ['the'], 'here': ['is', 'is'], 'oclock': ['in'], 'nine': ['oclock'], 'at': ['nine'], 'text': ['a', 'a', 'here', 'at'], 'small': ['piece', 'piece', 'piece', 'piece'], 'the': ['morning'], 'piece': ['of', 'of', 'of', 'of']}`
- `s.trigram_dict`: the dictionary `{'text here': ['is'], 'nine oclock': ['in'], 'text at': ['nine'], 'small piece': ['of', 'of', 'of', 'of'], 'piece of': ['text', 'text', 'text', 'text'], 'here is': ['a', 'a'], 'text a': ['small', 'small'], 'oclock in': ['the'], 'a small': ['piece', 'piece', 'piece', 'piece'], 'at nine': ['oclock'], 'of text': ['a', 'a', 'here', 'at'], 'in the': ['morning'], 'is a': ['small', 'small']}`

3.1.1 TODO (aim for end of the day Friday): Complete `make_sample` in a3.py

Complete the aforementioned function according to its specification and the implementation hints and requirements given in the skeleton code.

Do note the comment in the body that this function is doing the job typically handled by an `__init__` method for a class. Our unusual design decision stems from the fact that we haven't yet covered writing methods in this course.

3.1.2 Debugging Hints

Debugging may be a bit challenging. To make this a bit easier for/on you, we've provided function `sample_from_source_file()` in a3given.py which calls your `a3.make_sample()` and then uses our `a3given.print_summmary_stats()` to print out the contents of the new Sample your code should have helped create.

So, run file a3checks.py, which tries to use your code to create a new Sample from the file short.txt, and then attempts to print out some information about your Sample. Do you get the same output as in our A3_sample.txt?

You may also need to do some extra printing to debug. Take a look at our code for `a3given.print_summary_stats()`; you may learn some useful techniques.

You'll also want to check whether you've made each attribute the right type.

3.2 TODO (aim for end of the day Friday or Monday): Complete `merge` in a3.py

This function is what allows us to combine several SOTU files, each by the same speaker, into one big sample for that speaker.

Take a look at the "Combining Obama SOTU and then Bush SOTU" lines in a3checks.py to see a compact way of doing so. And then, verify that you get the same output from your a3checks.py as that depicted in A3_sample.txt.

3.3 TODO (aim for end of the day Monday): Complete `diffs()`

Read the specification for this function in a3.py. Our implementation of this function was used to generate the Obama-vs-Bush output in the Introduction.

For this assignment, we'll use the *log probability ratio* to score a word w that occurs in both Samples under consideration[5]. For simplicity, let's say that c_1 is the number of times that w occurs in the first sample s_1, and c_2 is the number of times that w occurs in the second sample s_2. Finally, let n_1 and n_2 be the lengths of s_1 and s_2, respectively. Then, the log probability ratio is as follows, where, importantly, **all division operations must be done using float arithmetic**:

$$\log\left(\frac{c_1/n_1}{c_2/n_2}\right) \tag{1}$$

What does this function mean?

- If c_1/n_1 were the same as c_2/n_2 — so the word occurs with the same frequency in both samples — then the ratio is 1, and the log of the ratio is therefore 0, which seems like a "neutral" number.
- If $c_1/n_1 > c_2/n_2$ — so the word has a higher rate of occurrence in s_1 than in s_2 — then the ratio is greater than one, and the log of the ratio will then be positive.
- Finally, if $c_1/n_1 < c_2/n_2$, the score comes out negative.

So, the sign of the function tells us which sample the word "leans toward".

To avoid dividing by zero or taking the log of zero, we don't compute the score for any words such that either c_1 or c_2 is 0.

Implement the function according to its specification, obeying the hints and/or requirements given in the skeleton code. Then, use A3_sample.txt to verify your work, as before.

3.4 TODO: Complete the Generation Functions

The two functions `bigram_generation` and `trigram_generation` stitch together randomly chosen bits of the sample text to create a new text string in the "style" of the author(s) of the Sample.

The term *bigram* means a two-word sequence. The term *trigram* means a three-word sequence. The first function, `bigram_generation`, bases its decision of the next word to generate based just on the single previous word last generated; whereas the second function's decision is based on the *two* words last generated.[6] For `trigram_generation`, pick the starting bigram from the *text*, not the bigram dictionary. (Hint: while it may not be immediately obvious how to pick a bigram from the text, note that it suffices to pick the *index* of the starting bigram in the text.)

Follow the instructions given in the skeleton code.

Once you get these functions working, you might amuse yourself by finding other document files to generate from. Can you make your own new Shakespeare play?

3.4.1 If You Are Running Low On Time, De-Prioritize The Trigram Problem

The vast bulk of the credit for these two functions will be assigned to the bigram part of the challenge, so if pressed for time, concentrate on getting the bigram function working.

4 Before You Submit

Once again, check the results of running a3checks.py against A3_sample.txt.

Put your (and your partner's, if you have one) name and netid at the top of the submitted file(s). **Also acknowledge by name all help you received in comments at the top of your submitted file(s).**

If you try running your code on some interesting data files, we'd be happy to hear about any intriguing findings you make; feel free to email us professors to let us know!

[5] There are better scoring functions to use, but we'll stick to this one for simplicity. One undesirable property it has it that low-frequency words tend to come out as highly biased to one sample or the other.

[6] If you have heard of Markov models or probabilistic language models, these functions are non-standardly-implemented versions of those models, in that the choice of next state is uniform over all the given choices.