LIN 353C: Introduction to Computational Linguistics, Spring 2015, Erk

Midterm exam

Take-home exam. Handed out Thursday March 5, 2015

Due midnight on Sunday March 8, 2015. Please submit this electronically on Canvas.

This take-home midterm exam comes with a file, `lastname_firstname_midterm.py`. This is a (basically empty) file, called a *stub file*. Please record all the answers in the appropriate places of this file. Please make sure that you save the file in *plain text*, not something like the Word format.

For submission, please rename the file such that it reflects your last name. That is, if your name was Ada Lovelace, you would rename it to

```
lovelace_ada_midterm.py
```

Please also remember to put your name and EID in the beginning of the file `lastname_firstname_midterm.py`

Please put your Python code into `lastname_firstname_midterm.py`, but do not put the results that your Python code produced, unless instructed otherwise. Also, please do not copy the line-initial ">>>" from the Python shell into your answer file. (The same holds for the ... symbols from the Python shell.)

Please make sure that you submit your Python code for all problems that ask for Python code, and not just the results that you got using your Python code. We need to see the code.

Please submit your solution to the midterm exam electronically using the Canvas Dropbox.

If any of these instructions do not make sense to you, please get in touch with the instructor right away.

A perfect solution to this midterm exam will be worth *100* points.

1. **Functions and lists (10 points)**

   Write a Python function `summed_wordlength` that takes as a parameter a list of strings (words) and returns the sum of the lengths of the strings. The usage of the function should be as follows:

```
>>> summed_wordlength( [ "hello", "world!" ] )
11
>>> summed_wordlength( [ "friends", "romans", "and", "countrymen" ] )
26
```

   Please note that to get full credit for this problem, your code must consist of a python *function*, and it must *return*, rather than print, its result.

2. **Regular expressions (15 points)**

   Download the novel *The Count of Monte Cristo* by Alexandre Dumas from Project Gutenberg. Please make sure to use the plain-text version. You do not need to remove the Gutenberg legalese from the beginning and the end of the file.

   Then use regular expressions (not the NLTK `concordance()` method) to find all *lines* in the text that contain the verb "stand". You need to find the following word forms:

   stand

   stood

   standing

   stands

   Make sure not to find the sequence "stand" embedded in other words, like "understand".

   Remember that you may want to keep Python from interpreting symbols in your regular expression string by marking the string as "raw", like this: `r`"...".

   Extra credit (5 points): For the extra credit, make a concordance for the verb "stand", but use the NLTK regular expressions, do not use the NLTK function `concordance()`. Display each line that contains the word "stand", but only display up to 20 characters to the left and up to 20 characters to the right of "stand" in the line in which it occurs in the text. (That is, you need not go beyond line breaks. But of the lines in which "stand" occurs, only display up to 20 characters on each side of "stand".) It does not matter what these characters are: spaces, letters, digits, whatever.

3. **Processing lists of words (25 points)**

   (a) Write a Python program that maps tagged words to simplified tags. Produce a list of pairs of a word and a part-of-speech tag as follows:

   ```
   import nltk
   from nltk.corpus import brown
   brown_tagged_words = brown.tagged_words(categories = "news")
   ```

   Now use the list of pairs (tuples) `brown_tagged_words` as follows: Map the list `brown_tagged_words` to a list of simplified tags. *The simplified tags should comprise the first two letters of the full part-of-speech tags.*

   The first six entries on `brown_tagged_words` are

   ```
   >>> brown_tagged_words[:6]
   [ ('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'),
    ('Grand', 'JJ-TL'), ('Jury', 'NN-TL'), ('said', 'VBD') ]
   ```

   For these, your code should produce the list

   ```
   ['AT', 'NP', 'NN', 'JJ', 'NN', 'VB' ]
   ```

   (b) Write a Python function that finds prefixed words and removes the prefix at the same time. Given a list of words, the result list should only retain the words that start with "in", and remove the "in" from the beginning of these words. Please do not retain the word "in" itself. (That is, we want a function that takes a list of words as input and returns a list of words as output.)

   To obtain a word list, please use again the novel "The Count of Monte Christo" that you downloaded for problem 2. Split it into words, and lowercase the words.

   The resulting list should start out like this:

   ```
   ['cluded', 'to', 'stinct', 'ner', 'to', 'to',
   'quired', 'to', 'quired', 'terrupted',...]
   ```

   Please state two words from your list where "in" is in fact a prefix, and two where it is not.

4. **Which verbs go with which prepositions? (25 points)**

   For this problem, collect the different prepositions that go with each verb in a corpus. As a corpus, use again the news section of the Brown corpus with part-of-speech tags. Please see problem 3 for how to obtain this corpus.

   For this problem, use Python dictionaries. Do not use the NLTK data structures `nltk.FreqDist` and `nltk.ConditionalFreqDist`. Apart from that, there are no restrictions on the Python functions you can use.

   In the Brown news corpus, look for verbs followed directly by a preposition, as in:

   > Stein , who **lives at** 3300 Lake Shore Dr.
   > we need guardians to **demonstrate against** the stupidity
   > will **march to** San Francisco for a rally in Union Square

   Identify verbs by a part-of-speech tag beginning with "V". Identify prepositions by their part-of-speech tag "IN". The corpus does not identify lemmas (dictionary forms) of the verbs, so you will not have access to those. So for this problem, collect prepositions separately for different forms of the same verb (like "live", "lives", "lived").

   Which verb occurs with the highest number of different prepositions directly after it? How many different prepositions are there for this verb, and what are they?

5. **Language models (25 points)**

For this problem, please use the novel "Dracula" by Bram Stoker, which is available online at Project Gutenberg. Make sure that you download a plain-text version.

You do not need to remove the legalese at the beginning and end of the file. But please convert all words to lowercase. Also remove all punctuation from the beginning and end of words.

(a) Compute how often each word occurs in *Dracula*, and how often each word bigram (pair) occurs.

What are the ten most frequent words, and what are their frequencies?

What are the ten most frequent bigrams, and what are their frequencies?

For this problem, you can either use Python dictionaries, or you can use NLTK data structures.

(b) Compute the probability of the sentence

there is a vampire in the room

in a bigram language model computed from *Dracula*.

You can use the NLTK data structure `nltk.FreqDist` to solve this problem, but not `nltk.ConditionalFreqDist` or `nltk.ConditionalProbDist`.

As a basis for your bigram language model, just use the word counts and bigram counts from part (a) of this problem.

To do so, multiply

- the probability of the word "there", computed as the relative frequency: $\frac{count(there)}{length\ of\ the\ novel\ Dracula\ in\ words}$
- the conditional probability $P(is|there)$ computed as $\frac{count(there\ is)}{count(there\ \_)}$
- the conditional probabilities $P(a|is)$, $P(vampire|a)$, $P(in|vampire)$, $P(the|in)$, $P(room|the)$, computed the same way.

(c) Like in part (b) of this problem, compute the probability of the sentence

there is a vampire in the room

in a bigram language model computed from *Dracula*.

But this time, use the NLTK data structure `nltk.ConditionalProbDist` to solve the problem.