CSCI 4140: Natural Language Processing

CSCI/DASC 6040: Computational Analysis of Natural Languages

Spring 2023

Homework 1 - Tokenization and segmentation

Due Sunday, January 29, at 11:59 PM

Part 1: Tokenizer basics - 30 points

Part 1(a) - 10 points

Write a function called `get_words` that takes a string `s` as its only argument. The function should return a list of the words in the same order as they appeared in `s`. Note that in this question a "word" is defined as a "space-separated item". For example:

```
get_words('The cat in the hat ate the rat in the vat')

['The', 'cat', 'in', 'the', 'hat', 'ate', 'the', 'rat', 'in',
'the', 'vat']
```

Hint: If you don't know how to approach this problem, read about str.split().

In [ ]:

```
# First writing a function from the scratch to practice what we want
# for this question. And then after playing with that, we're gonna write a cl
# get_words function easily. For this simple function, I get a sentence as an
# input then we change that sentence the lopwercase and lastly we're gonna sp
# and get the words.

def get_words():
    sentence = input("Write a sentence:")
    sentence = sentence.lower()
    split_sentence = str.split(sentence)
    print(split_sentence)

get_words()
```

```
['i', 'am', 'farnoosh', 'koleini']
```

In [ ]:
```
# get_words function

def get_words(s, do_lower=False):
    words = s.split()
    if (do_lower):
        s = s.lower()
    return s.split()

print (get_words('The cat in the hat ate the rat  in the vat'))
```

```
['The', 'cat', 'in', 'the', 'hat', 'ate', 'the', 'rat', 'in', 'the', 'vat']
```

Part 1(b) - 10 points

Write a function called `count_words` that takes a list of the words of `s` as its only argument and returns a `collections.Counter` that maps a word to the frequency that it occurred in `s`. Use the output of the `get_words` function as the input to this function.

```
s = 'The cat in the hat ate the rat in the vat'
words = get_words(s)
count_words(words)

Counter({'the': 3, 'in': 2, 'The': 1, 'cat': 1, 'hat': 1, 'ate':
1, 'rat': 1, 'vat': 1})
```

Notice that this is somewhat unsatisfying because **the** is counted separately from **The**. To fix this, have your `get_words` function be able to lower-case all of the words before returning them. You won't want to break any previous code you wrote, though (backwards compatibility is important!), so add a new parameter to `get_words` with a default value:

```
def get_words(s, do_lower=False)
```

Now, if `get_words` is called the way we were using it above, nothing will change. But if we call `get_words(s, do_lower=True)` then `get_words` should lowercase the string before getting the words. You can make use of `str.lower` to modify the string. When you're done, the following should work:

```
s = 'The cat in the hat ate the rat in the vat'
words = get_words(s, do_lower=True)
count_words(words)

Counter({'the': 4, 'in': 2, 'cat': 1, 'hat': 1, 'ate': 1, 'rat':
1, 'vat': 1})
```

In [ ]:

```
# Again for writing the count word function, first we need to write a simple
# similiar function and get a general input like any sentences, then using th
# previous small function, get_words, and then adding another part to that wh
# is counting the number of each word.

import collections # importing the collection module.

def count_words():
    sentence = input("Write a sentence:")
    sentence = sentence.lower()
    split_sentence = str.split(sentence)
    return(collections.Counter(split_sentence))

count_words()
```

Out[ ]:

```
Counter({'there': 1,
         'are': 1,
         'many': 1,
         'students': 1,
         'in': 2,
         'our': 1,
         'class': 1,
         'who': 1,
         'like': 1,
         'get': 1,
         'a': 1,
         'great': 1,
         'job': 1,
         'the': 1,
         'future': 1,
         'soon': 1})
```

In [ ]:
```
# count_words function

import collections

def get_words(s, do_lower=False):
    words = s.split()
    if (do_lower):
        s = s.lower()
    return s.split()


def count_words(words):
   count_words = collections.Counter(words)
   return (count_words)



s = 'The cat in the hat ate the rat in the vat'
words = get_words(s, do_lower=True)
print(count_words(words))
```

```
Counter({'the': 4, 'in': 2, 'cat': 1, 'hat': 1, 'ate': 1, 'rat': 1, 'vat': 1})
```

Part 1(c) - 10 points

Write a function called `words_by_frequency` that takes a list of words as its only required argument. The function should return a list of `(word, count)` tuples sorted by count such that the first item in the list is the most frequent item. Items with the same frequency should be in the same order they appear in the original list of words.

`words_by_frequency` should, additionally, take a second parameter `n` that specifies the maximum number of results to return. If `n` is passed, then only the `n` most frequent words should be returned. If `n` is not passed, then all words should be returned in order of frequency.

```
words_by_frequency(words)

[('the', 4), ('in', 2), ('cat', 1), ('hat', 1), ('ate', 1),
('rat', 1), ('vat', 1)]


words_by_frequency(words, n=3)

[('the', 4), ('in', 2), ('cat', 1)]
```

In [ ]:

```
# Words_by_frequency function: for this section, again we need to write a sim
# function which find the most common words with high frequency in a sentence
# a text file. For example, simply we can get a sentence as an input and then
# finding the three most common word in that sentence.

import collections

def words_by_frequency():
    sentence = input("Write a sentence:")
    n = int(input("Maximum number of results to return: "))
    sentence = sentence.lower()
    split_sentence = str.split(sentence)
    freq = collections.Counter(split_sentence)
    return freq.most_common(n)

words_by_frequency()
```

Out[ ]:

```
[('are', 2), ('there', 1), ('a', 1)]
```

In [ ]:

```
# words_by_frequency function

from collections import Counter

def words_by_frequency(word, n = None):
    word_count = Counter(words)
    return count_words(words).most_common(n)

print (words_by_frequency(words, n=3))
```

```
[('the', 4), ('in', 2), ('cat', 1)]
```

Part 2: Through the rabbit hole - 50 points

Next, you will explore some files from Project Gutenberg, a library of free eBooks for texts outside of copyright.

Some of the Gutenberg texts are all available in the `data/gutenberg/` directory.

Part 2(a) - 10 points

Let's use the copy of Lewis Carroll's "Alice's Adventures in Wonderland" from **data/gutenberg/carroll-alice.txt**. Use your `words_by_frequency` and `count_words` functions from Part 1 to explore the text. For the rest of this exercise, you will always lowercase when getting a list of words. You should find that the five most frequent words in the text are:

```
the      1603
and       766
to        706
a         614
she       518
```

Note: If your numbers were right in the previous part, but don't match here, it may be because of how you're calling `split` . Take a look at the documentation for `split` to see if there's a different way you can call it.

Check-In

1. If your `count_words` function is working correctly, it should report that the word **alice** occurs 221 times. Confirm that you get this result with your code.
2. The word **alice** actually appears 398 times in the text, though this is not the answer you got for the previous question. Why? Examine the data to see if you can figure it out before continuing.

In [ ]:

```
# This function simply shows that the word alice occurs 221 times in the text
# but actually the exact number is 398. This differnce is becauser sometimes
# some punctuations next to the word 'alice' so the machine recognize it as a
# alice.

f = open('/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segment
data = f.read()

def count_words():
    data1 = data.lower()
    split_data = str.split(data1)
    return(collections.Counter(split_data))

count_words()
```

Out[ ]:

```
Counter({"[alice's": 1,
         'adventures': 4,
         'in': 351,
         'wonderland': 2,
         'by': 57,
         'lewis': 1,
         'carroll': 1,
         '1865]': 1,
         'chapter': 12,
         'i.': 1,
         'down': 78,
         'the': 1603,
         'rabbit-hole': 3,
         'alice': 221,
         'was': 333,
         'beginning': 11,
         'to': 706,
         'get': 43,
         'very': 139,
         'tired': 7,
         'of': 493,
         'sitting': 10,
         'her': 208,
         'sister': 5,
         'on': 142,
         'bank,': 2,
         'and': 766,
         'having': 10,
         'nothing': 22,
         'do:': 1,
         'once': 19,
         'or': 68,
         'twice': 1,
         'she': 518,
         'had': 176,
         'peeped': 3,
         'into': 67,
```

```
'book': 3,
'reading,': 1,
'but': 116,
'it': 362,
'no': 67,
'pictures': 4,
'conversations': 1,
'it,': 38,
"'and": 55,
'what': 91,
'is': 71,
'use': 16,
'a': 614,
"book,'": 2,
'thought': 63,
"'without": 1,
"conversation?'": 1,
'so': 126,
'considering': 3,
'own': 9,
'mind': 5,
'(as': 3,
'well': 27,
'as': 249,
'could,': 7,
'for': 135,
'hot': 4,
'day': 11,
'made': 29,
'feel': 8,
'sleepy': 3,
'stupid),': 1,
'whether': 11,
'pleasure': 2,
'making': 8,
'daisy-chain': 1,
'would': 76,
'be': 138,
'worth': 4,
'trouble': 4,
'getting': 21,
'up': 81,
'picking': 2,
'daisies,': 1,
'when': 73,
'suddenly': 10,
'white': 28,
'rabbit': 30,
'with': 169,
'pink': 1,
'eyes': 18,
'ran': 13,
'close': 12,
```

```
'her.': 13,
'there': 64,
'remarkable': 2,
'that;': 1,
'nor': 2,
'did': 50,
'think': 38,
'much': 40,
'out': 97,
'way': 37,
'hear': 14,
'say': 35,
'itself,': 4,
"'oh": 2,
'dear!': 9,
'oh': 6,
'i': 260,
'shall': 22,
"late!'": 1,
'(when': 1,
'over': 31,
'afterwards,': 1,
'occurred': 2,
'that': 222,
'ought': 13,
'have': 77,
'wondered': 1,
'at': 205,
'this,': 17,
'time': 47,
'all': 155,
'seemed': 27,
'quite': 55,
'natural);': 1,
'actually': 1,
'took': 24,
'watch': 6,
'its': 57,
'waistcoat-pocket,': 2,
'looked': 45,
'then': 59,
'hurried': 11,
'on,': 27,
'started': 2,
'feet,': 6,
'flashed': 1,
'across': 5,
'never': 40,
'before': 19,
'seen': 12,
'either': 6,
'take': 19,
'burning': 1,
```

```
'curiosity,': 2,
'field': 1,
'after': 40,
'fortunately': 1,
'just': 48,
'see': 48,
'pop': 1,
'large': 32,
'under': 16,
'hedge.': 1,
'another': 21,
'moment': 21,
'went': 79,
'how': 46,
'world': 6,
'again.': 16,
'straight': 2,
'like': 75,
'tunnel': 1,
'some': 50,
'way,': 7,
'dipped': 2,
'down,': 14,
'not': 108,
'about': 84,
'stopping': 1,
'herself': 40,
'found': 28,
'falling': 2,
'deep': 5,
'well.': 2,
'deep,': 2,
'fell': 6,
'slowly,': 2,
'plenty': 2,
'look': 25,
'wonder': 15,
'going': 26,
'happen': 5,
'next.': 3,
'first,': 11,
'tried': 18,
'make': 26,
'coming': 5,
'to,': 3,
'too': 21,
'dark': 3,
'anything;': 2,
'sides': 4,
'well,': 1,
'noticed': 7,
'they': 117,
'were': 82,
```

```
'filled': 3,
'cupboards': 2,
'book-shelves;': 1,
'here': 19,
'saw': 13,
'maps': 1,
'hung': 1,
'upon': 26,
'pegs.': 1,
'jar': 2,
'from': 32,
'one': 80,
'shelves': 1,
'passed;': 1,
'labelled': 1,
"'orange": 1,
"marmalade',": 1,
'great': 39,
'disappointment': 1,
'empty:': 1,
'drop': 1,
'fear': 4,
'killing': 1,
'somebody,': 1,
'managed': 3,
'put': 31,
'past': 1,
'it.': 16,
"'well!'": 1,
'herself,': 31,
"'after": 2,
'such': 40,
'fall': 6,
'tumbling': 2,
'stairs!': 1,
'brave': 1,
"they'll": 4,
'me': 46,
'home!': 1,
'why,': 9,
"wouldn't": 12,
'anything': 14,
'even': 19,
'if': 72,
'off': 40,
'top': 8,
"house!'": 1,
'(which': 3,
'likely': 4,
'true.)': 1,
'down.': 2,
'come': 26,
'an': 56,
```

```
'end!': 1,
"'i": 121,
'many': 12,
'miles': 3,
"i've": 20,
'fallen': 4,
'this': 103,
"time?'": 1,
'said': 421,
'aloud.': 3,
'must': 42,
'somewhere': 1,
'near': 14,
'centre': 1,
'earth.': 2,
'let': 13,
'see:': 3,
'four': 6,
'thousand': 2,
"think--'": 3,
'(for,': 2,
'you': 264,
'see,': 11,
'learnt': 2,
'several': 4,
'things': 19,
'sort': 17,
'lessons': 4,
'schoolroom,': 1,
'though': 7,
'good': 23,
'opportunity': 8,
'showing': 2,
'knowledge,': 1,
'listen': 4,
'her,': 18,
'still': 13,
'practice': 1,
'over)': 1,
"'--yes,": 1,
"that's": 17,
'right': 21,
'distance--but': 1,
'latitude': 2,
'longitude': 2,
'got': 45,
"to?'": 3,
'(alice': 4,
'idea': 14,
'was,': 14,
'either,': 2,
'nice': 5,
'grand': 2,
```

```
'words': 14,
'say.)': 1,
'presently': 2,
'began': 47,
'through': 13,
'earth!': 1,
'funny': 3,
"it'll": 7,
'seem': 7,
'among': 12,
'people': 10,
'walk': 4,
'their': 51,
'heads': 8,
'downward!': 1,
'antipathies,': 1,
'(she': 9,
'rather': 25,
'glad': 11,
'listening,': 2,
'time,': 7,
"didn't": 11,
'sound': 3,
'word)': 1,
"'--but": 1,
'ask': 7,
'them': 49,
'name': 8,
'country': 1,
'is,': 16,
'know.': 8,
'please,': 4,
"ma'am,": 1,
'new': 5,
'zealand': 1,
"australia?'": 1,
'(and': 1,
'curtsey': 1,
'spoke--fancy': 1,
'curtseying': 1,
"you're": 17,
'air!': 1,
'do': 51,
'could': 65,
'manage': 6,
'it?)': 1,
'ignorant': 1,
'little': 120,
'girl': 3,
"she'll": 3,
'asking!': 1,
'no,': 4,
'ask:': 1,
```

```
'perhaps': 12,
'written': 6,
"somewhere.'": 1,
'else': 8,
'do,': 7,
'soon': 24,
'talking': 12,
"'dinah'll": 1,
'miss': 1,
'to-night,': 1,
'should': 27,
"think!'": 1,
'(dinah': 1,
'cat.)': 1,
'hope': 3,
'remember': 12,
'saucer': 1,
'milk': 1,
'tea-time.': 1,
'dinah': 4,
'my': 56,
'wish': 21,
'me!': 3,
'are': 40,
'mice': 3,
'air,': 5,
"i'm": 37,
'afraid,': 2,
'might': 27,
'catch': 3,
'bat,': 1,
'mouse,': 11,
'cats': 9,
'eat': 16,
'bats,': 1,
"wonder?'": 3,
'sleepy,': 1,
'saying': 11,
'dreamy': 1,
"'do": 8,
'bats?': 1,
"bats?'": 1,
'sometimes,': 1,
'bats': 1,
"cats?'": 1,
'for,': 3,
"couldn't": 9,
'answer': 6,
'question,': 4,
'matter': 8,
'which': 40,
'felt': 23,
'dozing': 1,
```

```
'off,': 14,
'begun': 6,
'dream': 3,
'walking': 5,
'hand': 11,
'dinah,': 3,
'earnestly,': 1,
"'now,": 4,
'tell': 29,
'truth:': 1,
'ever': 17,
"bat?'": 1,
'suddenly,': 1,
'thump!': 2,
'came': 38,
'heap': 1,
'sticks': 1,
'dry': 7,
'leaves,': 2,
'over.': 2,
'bit': 8,
'hurt,': 1,
'jumped': 5,
'feet': 11,
'moment:': 1,
'up,': 9,
'overhead;': 1,
'long': 30,
'passage,': 2,
'sight,': 4,
'hurrying': 1,
'lost:': 1,
'away': 15,
'wind,': 2,
'say,': 5,
'turned': 16,
'corner,': 2,
'ears': 4,
'whiskers,': 1,
'late': 3,
"it's": 31,
"getting!'": 1,
'behind': 12,
'longer': 2,
'seen:': 1,
'long,': 1,
'low': 7,
'hall,': 5,
'lit': 1,
'row': 2,
'lamps': 1,
'hanging': 3,
'roof.': 1,
```

```
'doors': 2,
'round': 30,
'locked;': 1,
'been': 36,
'side': 12,
'other,': 5,
'trying': 11,
'every': 12,
'door,': 9,
'walked': 10,
'sadly': 2,
'middle,': 3,
'wondering': 7,
'three-legged': 2,
'table,': 5,
'solid': 1,
'glass;': 1,
'except': 4,
'tiny': 4,
'golden': 7,
'key,': 3,
"alice's": 10,
'first': 30,
'belong': 1,
'hall;': 1,
'but,': 8,
'alas!': 3,
'locks': 1,
'large,': 1,
'key': 5,
'small,': 1,
'any': 36,
'rate': 4,
'open': 6,
'them.': 2,
'however,': 19,
'second': 4,
'round,': 7,
'curtain': 1,
'before,': 11,
'door': 15,
'fifteen': 1,
'inches': 6,
'high:': 3,
'lock,': 1,
'delight': 1,
'fitted!': 1,
'opened': 9,
'led': 4,
'small': 8,
'larger': 3,
'than': 23,
'rat-hole:': 1,
```

```
'knelt': 1,
'along': 5,
'passage': 1,
'loveliest': 1,
'garden': 4,
'saw.': 1,
'longed': 2,
'wander': 1,
'those': 10,
'beds': 1,
'bright': 7,
'flowers': 2,
'cool': 2,
'fountains,': 1,
'head': 29,
'doorway;': 1,
'go': 39,
"through,'": 1,
'poor': 26,
'alice,': 76,
"'it": 30,
'without': 24,
'shoulders.': 1,
'oh,': 6,
'shut': 4,
'telescope!': 1,
'only': 44,
'know': 46,
"begin.'": 3,
'out-of-the-way': 3,
'happened': 3,
'lately,': 1,
'few': 9,
'indeed': 3,
'really': 9,
'impossible.': 1,
'waiting': 8,
'back': 29,
'half': 21,
'hoping': 3,
'find': 20,
'rules': 3,
'shutting': 2,
'telescopes:': 1,
'bottle': 7,
"('which": 1,
'certainly': 8,
"before,'": 3,
'alice,)': 2,
'neck': 6,
'paper': 3,
'label,': 1,
"'drink": 3,
```

```
"me'": 2,
'beautifully': 2,
'printed': 1,
'letters.': 1,
"me,'": 8,
'wise': 2,
'hurry.': 3,
"'no,": 9,
"i'll": 24,
"first,'": 2,
'said,': 26,
'marked': 5,
'"poison"': 1,
"not';": 1,
'read': 9,
'histories': 1,
'children': 5,
'who': 54,
'burnt,': 1,
'eaten': 1,
'wild': 2,
'beasts': 1,
'other': 26,
'unpleasant': 2,
'things,': 2,
'because': 12,
'simple': 5,
'friends': 2,
'taught': 4,
'them:': 1,
'as,': 2,
'red-hot': 1,
'poker': 1,
'will': 30,
'burn': 2,
'hold': 6,
'long;': 1,
'cut': 5,
'your': 60,
'finger': 3,
'deeply': 1,
'knife,': 1,
'usually': 2,
'bleeds;': 1,
'forgotten': 6,
'that,': 5,
'drink': 4,
"'poison,'": 2,
'almost': 6,
'certain': 2,
'disagree': 1,
'you,': 26,
'sooner': 2,
```

```
'later.': 1,
'ventured': 4,
'taste': 2,
'finding': 3,
'nice,': 1,
'(it': 5,
'had,': 1,
'fact,': 4,
'mixed': 2,
'flavour': 1,
'cherry-tart,': 1,
'custard,': 1,
'pine-apple,': 1,
'roast': 1,
'turkey,': 1,
'toffee,': 1,
'buttered': 1,
'toast,)': 1,
'finished': 7,
'off.': 5,
'*': 60,
"'what": 34,
'curious': 16,
"feeling!'": 1,
'alice;': 16,
"telescope.'": 1,
'indeed:': 1,
'now': 25,
'ten': 5,
'high,': 3,
'face': 7,
'brightened': 2,
'size': 6,
'lovely': 2,
'garden.': 3,
'waited': 9,
'minutes': 7,
'shrink': 1,
'further:': 1,
'nervous': 4,
'this;': 2,
"'for": 7,
'end,': 1,
"know,'": 8,
"'in": 8,
'altogether,': 2,
'candle.': 1,
"then?'": 1,
'fancy': 3,
'flame': 1,
'candle': 2,
'blown': 1,
'out,': 13,
```

```
'thing.': 1,
'while,': 4,
'more': 39,
'happened,': 2,
'decided': 3,
'once;': 1,
'alas': 1,
'alice!': 3,
'table': 8,
'possibly': 3,
'reach': 4,
'it:': 9,
'plainly': 1,
'glass,': 2,
'best': 9,
'climb': 1,
'legs': 3,
'slippery;': 1,
'trying,': 1,
'thing': 35,
'sat': 17,
'cried.': 2,
"'come,": 9,
"there's": 16,
'crying': 2,
"that!'": 8,
'sharply;': 1,
'advise': 1,
'leave': 7,
"minute!'": 1,
'generally': 5,
'gave': 15,
'advice,': 1,
'(though': 1,
'seldom': 1,
'followed': 8,
'it),': 2,
'sometimes': 4,
'scolded': 1,
'severely': 3,
'bring': 2,
'tears': 5,
'eyes;': 1,
'remembered': 5,
'box': 3,
'cheated': 1,
'game': 6,
'croquet': 3,
'playing': 2,
'against': 9,
'child': 3,
'fond': 3,
'pretending': 1,
```

```
'two': 22,
'people.': 1,
"'but": 38,
"now,'": 4,
"'to": 5,
'pretend': 1,
'people!': 1,
'hardly': 12,
'enough': 10,
'left': 13,
'respectable': 1,
"person!'": 1,
'eye': 4,
'glass': 4,
'lying': 8,
'table:': 1,
'cake,': 2,
"'eat": 1,
'currants.': 1,
"'well,": 20,
"it,'": 19,
'makes': 11,
'grow': 13,
'larger,': 3,
'can': 31,
'key;': 1,
'smaller,': 3,
'creep': 1,
'door;': 1,
'garden,': 5,
"don't": 53,
'care': 4,
"happens!'": 1,
'ate': 1,
'bit,': 2,
'anxiously': 13,
"'which": 3,
'way?': 1,
"way?',": 1,
'holding': 2,
'growing,': 4,
'surprised': 6,
'remained': 3,
'same': 21,
'size:': 3,
'sure,': 2,
'happens': 2,
'eats': 1,
'expecting': 3,
'happen,': 1,
'dull': 2,
'stupid': 1,
'life': 2,
```

```
'common': 1,
'way.': 3,
'set': 14,
'work,': 1,
'cake.': 1,
'ii.': 1,
'pool': 7,
"'curiouser": 1,
"curiouser!'": 1,
'cried': 18,
'surprised,': 1,
'forgot': 2,
'speak': 8,
'english);': 1,
"'now": 5,
'opening': 3,
'largest': 1,
'telescope': 1,
'was!': 1,
'good-bye,': 1,
"feet!'": 1,
'(for': 1,
'far': 9,
'off).': 1,
"'oh,": 19,
'shoes': 5,
'stockings': 1,
'now,': 7,
'dears?': 1,
'sure': 16,
'_i_': 2,
"shan't": 4,
'able!': 1,
'deal': 11,
'myself': 2,
'you:': 1,
'can;--but': 1,
'kind': 6,
"them,'": 3,
"'or": 6,
"won't": 21,
'want': 9,
'go!': 1,
'give': 9,
'pair': 5,
'boots': 3,
"christmas.'": 1,
'planning': 1,
"'they": 9,
"carrier,'": 1,
'thought;': 1,
'seem,': 1,
'sending': 2,
```

```
'presents': 2,
"one's": 1,
'feet!': 1,
'odd': 1,
'directions': 1,
'look!': 1,
'foot,': 3,
'esq.': 1,
'hearthrug,': 1,
'fender,': 1,
'(with': 2,
'love).': 1,
'dear,': 6,
'nonsense': 1,
"talking!'": 1,
'struck': 2,
'roof': 5,
'hall:': 1,
'fact': 2,
'nine': 4,
'door.': 2,
'side,': 3,
'eye;': 2,
'hopeless': 1,
'ever:': 1,
'cry': 3,
"'you": 39,
'ashamed': 2,
"yourself,'": 1,
"'a": 11,
"you,'": 6,
'this),': 1,
'way!': 1,
'stop': 4,
'moment,': 5,
"you!'": 3,
'same,': 2,
'shedding': 1,
'gallons': 1,
'tears,': 3,
'until': 4,
'reaching': 1,
'hall.': 1,
'heard': 29,
'pattering': 3,
'distance,': 4,
'hastily': 7,
'dried': 1,
'coming.': 2,
'returning,': 1,
'splendidly': 1,
'dressed,': 1,
'kid': 5,
```

```
'gloves': 5,
'fan': 8,
'other:': 3,
'he': 111,
'trotting': 2,
'hurry,': 1,
'muttering': 3,
'himself': 4,
'came,': 2,
"'oh!": 2,
'duchess,': 8,
'duchess!': 3,
'oh!': 3,
'savage': 3,
'kept': 13,
"waiting!'": 1,
'desperate': 1,
'ready': 7,
'help': 9,
'one;': 2,
'so,': 7,
'began,': 6,
'low,': 6,
'timid': 3,
'voice,': 15,
"'if": 21,
"sir--'": 1,
'violently,': 2,
'dropped': 4,
'fan,': 1,
'skurried': 1,
'darkness': 1,
'hard': 8,
'go.': 1,
'gloves,': 3,
'and,': 19,
'hall': 1,
'hot,': 1,
'fanning': 1,
'talking:': 1,
"'dear,": 1,
'queer': 9,
'everything': 10,
'to-day!': 1,
'yesterday': 2,
'usual.': 2,
'changed': 7,
'night?': 1,
'think:': 1,
'morning?': 1,
'feeling': 6,
'different.': 1,
'next': 22,
```

```
'question': 7,
'am': 13,
'i?': 1,
'ah,': 1,
"puzzle!'": 1,
'thinking': 10,
'knew': 12,
'age': 2,
"'i'm": 20,
"ada,'": 1,
'hair': 5,
'goes': 7,
'ringlets,': 1,
'mine': 3,
"doesn't": 16,
'ringlets': 1,
'all;': 2,
"can't": 27,
'mabel,': 2,
'sorts': 3,
'she,': 5,
'knows': 2,
'little!': 1,
'besides,': 2,
"she's": 4,
'i,': 1,
'and--oh': 2,
'puzzling': 4,
'is!': 1,
'try': 12,
'used': 12,
'times': 6,
'five': 2,
'twelve,': 1,
'six': 2,
'thirteen,': 1,
'seven': 4,
'is--oh': 1,
'twenty': 1,
'rate!': 1,
'multiplication': 1,
'signify:': 1,
"let's": 3,
'geography.': 1,
'london': 1,
'capital': 4,
'paris,': 1,
'paris': 1,
'rome,': 1,
'rome--no,': 1,
'wrong,': 2,
'certain!': 1,
'mabel!': 1,
```

```
'"how': 2,
'doth': 3,
'little--"\'': 1,
'crossed': 3,
'hands': 6,
'lap': 2,
'lessons,': 1,
'repeat': 6,
'voice': 18,
...})
```

In [ ]:

```
# We can easily use both previous codes for words_by_frequency functions eith
# scratch one or the final clean function to find five most frequent words
# in the text. Now I would like to use that simple test code that I have deve
# and then start implementing the final version words_by_frequency code to ge
# result faster. Next part is creating a clean function and testing it again
# we're going to get the same result with the first scratch and simple functi

f = open('/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segment
data = f.read()

def words_by_frequency():
    n = int(input("Maximum number of results to return: "))
    data1 = data.lower()
    split_data = str.split(data1)
    freq = collections.Counter(split_data)
    return freq.most_common(n)

words_by_frequency()
```

Out[ ]:

```
[('the', 1603), ('and', 766), ('to', 706), ('a', 614), ('she', 518)]
```

In [ ]:
```
# clean version of the code and results

import urllib.request
import re

def get_words(s, do_lower=False):
    words = re.findall(r'\b\w+\b', s)
    if(do_lower):
        s = s.lower()
        return s.split()

file_path = (r"/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and se
with open(file_path, "r") as f:
    file_contents = f.read()
    words = get_words(file_contents, do_lower = True)
    word_counts = count_words(words)
    alice_count = word_counts.get("alice",0)
    word_frequency = words_by_frequency(words, n=5)
    print("The word 'alice' appears", alice_count, "time in the text.")
    print(word_frequency)
```

```
The word 'alice' appears 221 time in the text.
[('the', 1603), ('and', 766), ('to', 706), ('a', 614), ('she', 518)]
```

In [ ]:
```
#The word alice actually appears 398 times in the text, though this is not th

import urllib.request
import re

def get_words(s, do_lower=False):
    words = re.findall(r'\b\w+\b', s)
    if(do_lower):
        s = s.lower()
        return s.split()

file_path = (r"/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and se
with open(file_path, "r") as f:
    file_contents = f.read()

new_count = len(re.findall(r'\bAlice\b', file_contents, re.IGNORECASE))

print("The word 'alice' occurs = {} times".format(new_count))
```

```
The word 'alice' occurs = 398 times
```

Part 2(b) - 10 points

A spoiler for 2(a): there is a deficiency in how we implemented the `get_words` function. When we are counting words, we probably don't care whether the word was adjacent to a punctuation mark. For example, the word **hatter** appears in the text 57 times, but if we queried the `count_words` dictionary, we would see it only appeared 24 times. However, it also appeared numerous times adjacent to a punctuation mark, so those instances got counted separately:

```
word_freq = words_by_frequency(words)
for (word, freq) in word_freq:
    if 'hatter' in word:
        print('{:10} {:3d}'.format(word, freq))
```

```
hatter      24
hatter.     13
hatter,     10
hatter:      6
hatters      1
hatter's     1
hatter;      1
hatter.'     1
```

Our `get_words` function would be better if it separated punctuation from words. We can accomplish this by using the `re.split` function. Be sure to import `re` to make `re.split()` work. Below is a small example that demonstrates how `str.split` works on a small text and compares it to using `re.split` :

```
text = '"Oh no, no," said the little Fly, "to ask me is in
vain."'
text.split()
```

```
['"Oh', 'no,', 'no,"', 'said', 'the', 'little', 'Fly,', '"to',
'ask', 'me', 'is', 'in', 'vain."']
```

```
re.split(r'(\W)', text)
```

```
['', '"', 'Oh', ' ', 'no', ',', '', ' ', 'no', ',', '', '"', '',
' ', 'said', ' ', 'the',

 ' ', 'little', ' ', 'Fly', ',', '', ' ', '', '"', 'to', ' ',
'ask', ' ', 'me', ' ', 'is',

 ' ', 'in', ' ', 'vain', '.', '', '"', '']
```

Note that this is not exactly what we want, but it is a lot closer. In the resulting list, we find empty strings and spaces, but we have also successfully separated the punctuation from the words.

Using the above example as a guide, write and test a function called `tokenize` that takes a string as an input and returns a list of words and punctuation, but not extraneous spaces and empty strings. Like `get_words` , `tokenize` should take an optional argument `do_lower` that determines whether the string should be case normalized before separating the words. You don't need to modify the `re.split()` line: just remove the empty strings, spaces, and newlines.

```
tokenize(text, do_lower=True)

['"', 'oh', 'no', ',', 'no', ',', '"', 'said', 'the', 'little',
'fly', ',', '"', 'to', 'ask', 'me', 'is', 'in', 'vain', '.',
'"']

print(' '.join(tokenize(text, do_lower=True)))

" oh no , no , " said the little fly , " to ask me is in vain .
"
```

Checking In

Use your `tokenize` function in conjunction with your `count_words` function to list the top 5 most frequent words in **carroll-alice.txt**. You should get this:

```
'        2871      <-- single quote
,        2418      <-- comma
the      1642
.         988      <-- period
and       872
```

In [ ]:

```
# Finding the 5 most frequent words in carroll-alice.txt file
# Like the previous questions, I started writing a simple function, tokenize,
# then getting the data which is a text file here, and then finding the five
# most common words in that text file.

import re
import collections

f = open('/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segment
data = f.read()

def tokenize():
    n = int(input("Maximum number of results to return: "))
    data1 = data.lower()
    split_data = re.split(r'(\W)',data1)
    while ("" in split_data):
        split_data.remove("")
    while (' ' in split_data):
        split_data.remove(' ')
    while ('\n' in split_data):
        split_data.remove('\n')
    freq = collections.Counter(split_data)
    return freq.most_common(n)

tokenize()
```

Out[ ]:

```
[('"', 2871), (',', 2418), ('the', 1642), ('.', 988), ('and', 872)]
```

In [ ]:

```
# Better version of the code here! If you compare these codes, you will get t
# enevthough both of them get the same results, but after working on the scra
# we're gonna get the code below which works much more faster than the previo
# this is the concept of learning how to write a better and more efficient pr

import urllib.request
import re

def tokenize(text, do_lower=True):
    if (do_lower):
        text = text.lower()
    words = re.split(r'(\W)', text)
    words = [w for w in words if w not in (' ', '', '\n')]
    return words


with open(r"/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segme
    file_contents = f.read()
    words = tokenize(file_contents, do_lower = True)
    counts = count_words(words)

    print (words_by_frequency(counts, n = 5))
```

```
[("'", 2871), (',', 2418), ('the', 1642), ('.', 988), ('and', 872)]
```

Part 2(c) - 10 points

Write a function called `filter_nonwords` that takes a list of strings as input and returns a new list of strings that excludes anything that isn't entirely alphabetic. Use the `str.isalpha()` method to determine is a string is comprised of only alphabetic characters.

```
text = '"Oh no, no," said the little Fly, "to ask me is in
vain."'
tokens = tokenize(text, do_lower=True)
filter_nonwords(tokens)

['oh', 'no', 'no', 'said', 'the', 'little', 'fly', 'to', 'ask',
'me', 'is', 'in', 'vain']
```

Use this function to list the top 5 most frequent words in **carroll-alice.txt**. Confirm that you get the following before moving on:

```
the      1642
and       872
to        729
a         632
it        595
```

In [ ]:
```
# Finding 5 most frequent words in carroll-alice.txt using filter_nonwords
# and str.isalpha method.

import collections

f = open('/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segment
data = f.read()

def filter_nonwords():
    n = int(input("Maximum number of results to return: "))
    data1 = data.lower()
    for char in data1:
        if char.isalpha() == False:
           data1 = data1.replace(char, " ")
    split_data = re.split(r'(\W)',data1)
    while ("" in split_data):
        split_data.remove("")
    while (' ' in split_data):
        split_data.remove(' ')
    while ('\n' in split_data):
        split_data.remove('\n')
    freq = collections.Counter(split_data)
    return freq.most_common(n)

filter_nonwords()
```

Out[ ]:
```
[('the', 1642), ('and', 872), ('to', 729), ('a', 632), ('it', 595)]
```

In [ ]:
```
# Testing a more proficient code here! If you look at the results from the bo
# you will get the same results, but again compare the style of coding and co
# running time for both of them; you see this one works so fast! I brought bo
# just to show that I always start with the first coding style in my mind the
# like to find the better and faster way I try to find another version. This
# learned coding in python. It takes time, but worth it!

def filter_nonwords(lst):
    return list(filter(lambda x: x.isalpha(), lst))

with open(r"/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segme
    file_contents = f.read()
    tokens = tokenize(file_contents, do_lower = True)
    words = filter_nonwords(tokens)

print (words_by_frequency(words, n = 5))
```

```
[('the', 1642), ('and', 872), ('to', 729), ('a', 632), ('it', 595)]
```

Part 2(d) - 20 points

Iterate through all of the files in the **gutenberg** data directory and print out the top 5 words for each. To get a list of all the files in a directory, use the `os.listdir` function:

```
import os

directory = 'data/gutenberg/'
files = os.listdir(directory)
infile = open(os.path.join(directory, files[0]), 'r',
encoding='latin1')
```

This example also uses the function `os.path.join` that you might want to read about.

Note about encodings: This `open` function above uses the optional encoding argument to tell Python that the source file is encoded as latin1. Be sure to use this encoding flag to read the files in the **Gutenberg** corpus, as the default (Unicode) won't work!

Token Analysis Questions

Answer the following questions.

1. **Most Frequent Word:** Loop through all the files in the **gutenberg** data directory that end in **.txt**. Is **the** always the most common word? If not, what are some other words that show up as the most frequent word (and in which documents)? What do you notice about these words?
2. **Impact of Lowercasing:** If you don't lowercase all the words before you count them, how does this result change, if at all? Discuss what you observe.

Note: If a question (like the one above) asks you to discuss results, that always means both what the results were and what that implies about the world (i.e., your corpus, your method, etc.). A good answer on this sort of question is a paragraph that goes something like "the result was X, specific interesting examples were X' and X", this is/isn't surprising because it would imply P or Q, this implies it might be better to do Y / to evaluate Z to learn more".

In [ ]:

```
# Loop through all the files in gutenverg data directory

import os
directory = './data/gutenberg/'
files = os.listdir(directory)
for f in files:
   if f.endswith('.txt'):
    with open(os.path.join(directory, f), 'r', encoding='latin1') as infile:
     s = infile.read()
```

In [ ]:

```
# Finding the 5 most common words in all gutenberg text files

import collections

f = open('/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segment
data = f.read()

def filter_nonwords():
    n = int(input("Maximum number of results to return: "))
    data1 = data.lower()
    for char in data1:
        if char.isalpha() == False:
           data1 = data1.replace(char, " ")
    split_data = re.split(r'(\W)',data1)
    while ("" in split_data):
        split_data.remove("")
    while (' ' in split_data):
        split_data.remove(' ')
    while ('\n' in split_data):
        split_data.remove('\n')
    freq = collections.Counter(split_data)
    return freq.most_common(n)

filter_nonwords()
```

Out[ ]:

```
[('the', 3838), ('of', 1940), ('to', 1527), ('and', 1346), ('a', 1163)]
```

Your answers go here.

1) Most frequent word: If you look at the results which show the most frequent words in each text files we have here, we can find out 'the' is always one of the most common word. This is not surprising because 'the' is the most word in the English-speaking world because it's an essential partof grammar and communication. It would be difficult to speak English without repeatedly using'the'. Therefore, based on the programming code I developed here, I can confirmed this easily!

austen-emma.txt: [('to', 5242), ('the', 5204), ('and', 4897), ('of', 4293), ('i', 3192)] austen-persuasion.txt: [('the', 3329), ('to', 2808), ('and', 2801), ('of', 2570), ('a', 1595)]

austen-sense.txt: [('to', 4116), ('the', 4105), ('of', 3572), ('and', 3491), ('her', 2551)]

blake-poems.txt: [('the', 439), ('and', 348), ('of', 146), ('in', 141), ('i', 130)]

bryant-stories.txt: [('the', 3452), ('and', 2099), ('to', 1180), ('a', 1036), ('he', 1021)]

burgess-busterbrown.txt:[('he', 678), ('the', 660), ('and', 516), ('to', 436), ('of', 342)]

carroll-alice.txt: [('the', 1642), ('and', 872), ('to', 729), ('a', 632), ('it', 595)]

chesterton-ball.txt: [('the', 4981), ('and', 2667), ('of', 2555), ('a', 2263), ('to', 1580)]

chesterton-brown.txt: [('the', 4670), ('and', 2221), ('a', 2132), ('of', 2093), ('to', 1391)]

chesterton-thursday.txt: [('the', 3636), ('a', 1742), ('of', 1725), ('and', 1658), ('he', 1126)]

edgeworth-parents.txt: [('the', 7728), ('to', 5220), ('and', 4983), ('of', 3745), ('i', 3674)]

melville-moby_dick.txt: [('the', 14431), ('of', 6609), ('and', 6430), ('a', 4736), ('to', 4625)]

milton-paradise.txt: [('and', 3395), ('the', 2968), ('to', 2228), ('of', 2050), ('in', 1366)]

shakespeare-caesar.txt: [('and', 627), ('the', 579), ('i', 533), ('to', 446), ('you', 391)]

shakespeare-hamlet.txt: [('the', 993), ('and', 863), ('to', 685), ('of', 610), ('i', 574)]

shakespeare-macbeth.txt: [('the', 650), ('and', 546), ('to', 384), ('i', 348), ('of', 338)]

whitman-leaves.txt: [('the', 10113), ('and', 5334), ('of', 4265), ('i', 2933), ('to', 2244)]

2) Impact of lowercasing:

We checked the lowercase and uppercase from the begining of this small project. Based on the results we found out if we do not use lowe_case as a parameter we're gonna get different results because python recognizes same words like 'The' and 'the', two different words, but we actually know they are the same word with the same meaning. So, because just at the begining of the sentence we have words with uppercase, it is better change all of them to lowercase and then start tokenizing and counting.

In [ ]:

```
# Finding the 5 most common words in all gutenberg text files (fast version!)

import os
import re

directory = (r"/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and se
files = os.listdir(directory)
for file in files:
    if re.search('txt', file):
        print(file)
        with open(os.path.join(directory, file), 'r', encoding='latin1') as f
            file_contents = file.read()
            tokens = tokenize(file_contents)
            words = filter_nonwords(tokens)
            word_freq = words_by_frequency(words, n=5)
            for word, freq in word_freq:
                print('{:10} {:3d}'.format(word, freq))
```

```
blake-poems.txt
the         439
and         348
of          146
in          141
i           130
carroll-alice.txt
the         1642
and         872
to          729
a           632
it          595
shakespeare-caesar.txt
and         627
the         579
i           533
to          446
you         391
whitman-leaves.txt
the         10113
and         5334
of          4265
i           2933
to          2244
milton-paradise.txt
and         3395
the         2968
to          2228
of          2050
in          1366
bible-kjv.txt
the         64023
and         51696
```

```
of          34670
to          13580
that        12912
austen-persuasion.txt
the         3329
to          2808
and         2801
of          2570
a           1595
melville-moby_dick.txt
the         14431
of          6609
and         6430
a           4736
to          4625
edgeworth-parents.txt
the         7728
to          5220
and         4983
of          3745
i           3657
chesterton-thursday.txt
the         3636
a           1742
of          1725
and         1658
he          1126
burgess-busterbrown.txt
he          678
the         660
and         516
to          436
of          342
chesterton-ball.txt
the         4965
and         2667
of          2555
a           2262
to          1580
austen-emma.txt
to          5239
the         5201
and         4896
of          4291
i           3178
chesterton-brown.txt
the         4670
and         2221
a           2132
of          2093
to          1391
shakespeare-hamlet.txt
the         993
```

```
and         863
to          685
of          610
i           574
austen-sense.txt
to          4116
the         4105
of          3572
and         3491
her         2551
shakespeare-macbeth.txt
the         650
and         546
to          384
i           348
of          338
bryant-stories.txt
the         3451
and         2098
to          1180
a           1036
he          1017
```

Part 3: Sentence segmentation - 30 points

Next, you will write a simple sentence segmenter.

The **data/brown** directory includes three English-language text files taken from the Brown Corpus:

- `editorial.txt`
- `fiction.txt`
- `lore.txt`

These files represent large strings of natural language text, with no line breaks nor other special symbols to annotate where sentence splits occur. In the data set you are working with, sentences can only end with one of 5 characters: period, colon, semi-colon, exclamation point and question mark.

However, there is a catch: not every period represents the end of a sentence. Many abbreviations (U.S.A., Dr., Mon., etc., etc.) that can appear in the middle of a sentence, and the period does not indicate the end of the sentence. (If you have a phone that uses autocomplete to type, you may already have had annoying experiences where it automatically capitalized words after these abbreviations!) These texts also have many examples where colon is not the end of the sentence. The other three punctuation marks are all nearly unambiguously the ends of a sentence (yes, even semi-colons).

For each of the above files, I have also provided a file in the same directory containing the **character index** (counting from 0 for the first character) of each of the actual locations of the ends of sentences:

- `editorial-eos.txt`
- `fiction-eos.txt`
- `lore-eos.txt`

Your job is to write a sentence segmenter, and to output the predicted token number of each sentence boundary.

Part 3(a) - 10 points

Below is some starter code.

In [ ]:
```
def my_best_segmenter(token_list):
    """ TODO: Replace this with an improved sentence segmenter. """
    pass

def baseline_segmenter(token_list):
    all_sentences = []
    this_sentence = []
    for token in token_list:
        this_sentence.append(token)
        if token in ['.', ':', ';', '!', '?']:
            all_sentences.append(this_sentence)
            this_sentence = []
    return all_sentences

def write_sentence_boundaries(sentence_list, out):
    """ TODO: Write out the token numbers of the sentence boundaries. """
    pass

""" TODO: Write out the code to parse a text file. """
```

Checking In

Confirm that your code can open the file **data/brown/editorial.txt** and that your code from the previous part splits it into 63,333 tokens.

Note: Do not filter out punctuation, since those tokens will be exactly the ones we want to consider as potential sentence boundaries!

In [ ]:
```
# Cheching In, using the previous tokenize code to confirm 63,333 tokens in e

f = open('/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segment
data = f.read()

def tokenize( ):
    data1 = data.lower()
    split_data = re.split(r'(\W)',data1)
    while ("" in split_data):
        split_data.remove("")
    while (' ' in split_data):
        split_data.remove(' ')
    while ('\n' in split_data):
        split_data.remove('\n')
    print(len(split_data))

tokenize()
```

```
63333
```

In [ ]:
```
# Checking in part of the question, (final version!)

dir = (r"/Users/koleinif20/Desktop/NLP/Homework 1 - Tokenization and segmenta
with open(os.path.join(dir, 'editorial.txt'), 'r', encoding='latin1') as f:
    tokens = tokenize(f.read(), do_lower = True)
    print(len(tokens))
```

```
63333
```

Part 3(b) - 10 points

The starter code contains a function called `baseline_segmenter` that takes a list of tokens as its only argument. It returns a list of tokenized sentences; that is, a list of lists of words, with one list per sentence.

```
baseline_segmenter(tokenize('I am Sam. Sam I am.')

[['I', 'am', 'Sam', '.'], ['Sam', 'I', 'am', '.']]
```

Remember that every sentence in our data set ends with one of the five tokens ['.', ':', ';', '!', '?']. Since it's a baseline approach, `baseline_segmenter` predicts that every instance of one of these characters is the end of a sentence.

Fill in the function `write_sentence_boundaries` . This function takes two arguments: a list of lists of strings (like the one returned by `baseline_segmenter`) and a pointer to a stream to write output (an open write-enabled file). You will need to loop through all of the sentences in the document. For each sentence, you will want to write the index of the last word in the sentence to the filepointer. Remember that Python lists are 0-indexed!

Confirm that when you run `baseline_segmenter` on the file **data/brown/editorial.txt**, it predicts 3278 sentence boundaries, and that the first five predicted boundaries are at tokens 22, 54, 74, 99, and 131.

In [ ]:
```
def baseline_segmenter(token_list):
    all_sentences = []
    this_sentence = []
    for token in token_list:
        this_sentence.append(token)
        if token in ['.', ':', ';', '!', '?']:
            all_sentences.append(this_sentence)
            this_sentence = []
    return all_sentences

def write_sentence_boundaries(sentences, filepointer):
    for i, sentence in enumerate(sentences):
        filepointer.write(f"{len(sentence)-1}\n")
        if i < 5:
            print(f"Predicted boundary at token {len(sentence)}")

with open(os.path.join(dir, 'editorial.txt'), 'r', encoding='latin1') as f:
    tokens = tokenize(f.read(), do_lower = True)
    segmented_tokens = baseline_segmenter(tokens)
    print(f"We have found {len(segmented_tokens)} sentence boundaries.")
    write_sentence_boundaries(segmented_tokens, open("predicted_boundaries.tx
    sentences = baseline_segmenter(tokens)
```

```
We have found 3278 sentence boundaries.
Predicted boundary at token 23
Predicted boundary at token 32
Predicted boundary at token 20
Predicted boundary at token 25
Predicted boundary at token 32
```

Part 3(c) - extra credit, 10 points

Now it's time to improve the baseline sentence segmenter. We don't have any false negatives (since we're predicting that every instance of the possibly-end-of-sentence punctuation marks is, in fact, the end of a sentence), but we have quite a few false positives.

There's a placeholder for a second segmentation function defined in the starter code. You will fill in that `my_best_segmenter` function to do a (hopefully!) better job identifying sentence boundaries. The specifics of how you do so are up to you.

In [ ]:

```
#extra credit
import os

def my_best_segmenter(token_list):
    sentences = []
    current_sentence = []
    end_punctuations = ['.', ':', ';', '!', '?', ")", "]", "}","''"]
    for token in token_list:
        current_sentence.append(token)
        if token in end_punctuations:
            sentences.append(current_sentence)
            current_sentence = []
    return sentences

def write_sentence_boundaries(sentences, file):
    for i, sentence in enumerate(sentences):
        file.write(f"{len(sentence) - 1}\n")
        if i < 5:
            print(f"Predicted boundary at token {len(sentence)}")


with open(os.path.join(dir, 'editorial.txt'), 'r', encoding='latin1') as f:
    tokens = tokenize(f.read(), do_lower = True)
    segmented_tokens = my_best_segmenter(tokens)
    print(f"We have found {len(segmented_tokens)} sentence boundaries.")
    with open("predicted_boundaries.txt", "w") as boundary_file:
        write_sentence_boundaries(segmented_tokens, boundary_file)
```

```
We have found 3364 sentence boundaries.
Predicted boundary at token 23
Predicted boundary at token 32
Predicted boundary at token 20
Predicted boundary at token 25
Predicted boundary at token 32
```

Questions

1. Describe the performance of your final segmenter, by comparing some of the sentences it correctly tokenized/generated whereas the `baseline_segmenter` did not.
2. Describe at least 3 things that your final segmenter does better than the baseline segmenter and discuss them. What cases are you most proud of catching in your segmenter? Include specific examples that are handled well.
3. Describe at least 3 places where your segmenter still makes mistakes and discuss them. Include specific examples where your segmenter makes the wrong decision. If you had another week to work on this, what would you add? If you had the rest of the semester to work on it, what would you do?

Your answers go here.

1) If you compare my_best_segmenter and baseline_segmenter, you can figure it out; the number of types of punctuations in best segmenter is more than the baseline_one. So, based on the results we can get that some of the sentences it correctly tokenized/generated whereas the baseline_segmenter could not do that.

2) first: Handling abbreviations: abbreviations can cause false positive in the baseline segmenter. A final segmenter can have a list of commonly used abbreviations and ignore the end of the sentence punctuation after them.

Secondly: Considering context: in some cases the end of sentence punctuation mark may not signal the end of the sentence.

Thirdly: handling unsual punctuations: the baseline_segmenter method only takes into account the commonly recognized end of sentence punctuation marks, ignoring any unconventional forms of punctuations.

3) First: Addressing Complex Structures: In examples where the text contains complex arrangements such as nested phrases, parentheses or dashes, it can result in false negatives or false positives in the final segmentation.

Secondly: Dealing with Numbers: When dealing with numbers that include decimal points or units, false positives may occur. There is a way to tackle this problem; a final segmenter may implement regular expressions to identify numbers and disregard end-of-sentence punctuation that appears after them.

Thirdly: Handling Punctuation Errors: At times, the text may contain incorrect or absent punctuation, resulting in false negatives or false positives. To resolve this, the final segmenter can utilize a correction module to repair the punctuation and enhance its performance.

If i had another week i would add a module to tackle complex words and to work on minizing the punctuations errors like false positives errors.

If i had rest of the semester to work on it, I would train a model to make decision and provide accurate results.