

# Assignment 3

Dylan Phelan  
Working With Corpora  
Professor Gregory Crane

September 29, 2018

## Exercises for Chapter 3: Processing Raw Text

### Problem 21

Write a function `unknown()` that takes a URL as its argument, and returns a list of unknown words that occur on that webpage. In order to do this, extract all substrings consisting of lowercase letters (using `re.findall()`) and remove any items from this set that occur in the Words Corpus (`nltk.corpus.words`). Try to categorize these words manually and discuss your findings.

*Solution:*

When I initially ran my `unknown` function on the link to chapter three of the nltk book, I thought for sure I had done something wrong in my code. Many of the terms coming up as 'unknown' were common words like 'cases', 'using' and 'normalized'. I then manually loaded up the Words Corpus in a python interpreter to check and see if I could find those terms manually in a list of words in the corpus; much to my surprise, I couldn't! It looks like the Words Corpus (as I confirmed later by reading through more of Chapter 3) only includes common word lemmas, ignoring common variations on those stem terms. To get a better idea of how many 'actually' new terms were being used on the linked-to webpage, I removed any words that had any common suffixes. After controlling for those terms, a lot of the terms I was seeing were HTML-related terms that weren't parsed out by BeautifulSoup (a problem that gets exacerbated on webpages rich in javascript and advertisements), misspellings of terms, and example text that wasn't meant to pass as an actual English word. Below I've provided my code for the `unknown` function:

---

```
def unknown(url):
    # Step 1: get HTML
    response = request.urlopen(url)
    html = response.read().decode('utf-8')
    # Step 2: Convert to raw text
    raw = BeautifulSoup(html).get_text()
    # Step 3: Get the difference
    known_words = set([w for w in nltk.corpus.words.words('en') if w.islower()])
    words_on_webpage = set(re.findall(r"\b[a-z]+\b", raw))
    unknown_words = sorted(words_on_webpage.difference(known_words))

    print(f'Total Words from webpage: {len(words_on_webpage)}')
    print(f'Total unknown words: {len(unknown_words)}')
```

```
print(f'Percentage unknown words: {len(unknown_words)/len(words_on_webpage)}')
# Manual investigation of results highlights that the Words Corpus doesn't
# include variations on word lemmas
# Let's remove those variations for a better idea of how many new 'terms' appear
# on our webpage
word_versions = re.findall(r'\b[a-z]+(?:ing|ly|ed|ious|ies|ive|es|s|ment)', '
'.join(list(unknown_words)))
unknown_words_removing_variations_on_lemmas =
    set(unknown_words).difference(set(word_versions))
print(f'Total unknown words, removing common variations on word lemmas:
{len(unknown_words_removing_variations_on_lemmas)}')
print(f'Percentage unknown words, after removing common variations on word
lemmas:
{len(unknown_words_removing_variations_on_lemmas)/len(words_on_webpage)}')

return unknown_words
```

---

## Problem 22

Examine the results of processing the URL `http://news.bbc.co.uk/` using the regular expressions suggested above. You will see that there is still a fair amount of non-textual data there, particularly Javascript commands. You may also find that sentence breaks have not been properly preserved. Define further regular expressions that improve the extraction of text from this web page.

*Solution:*

My improvements can be broken into four changes to the extraction process:

1. Define how BeautifulSoup's `getText()` function should concatenate the content of the stripped HTML tags, by passing it `\n` as an argument.
2. Create a regular expression for removing the content sandwiched within javascript `<script>` tags before the tags are parsed out
3. Create a regular expression for removing the content sandwiched within CSS `<style>` tags before the tags are parsed out
4. Create a regular expression for removing leftover CSS classes from the text after processing of BeautifulSoup.

Below you can find my implementation in code of an improved `unknown` function:

---

```
def unknown(url)
    response = request.urlopen(url)
    html = response.read().decode('utf-8')
    # Step 1.b.: Remove any javascript or css
    js_regex = r"<script.*>[\S\s]+?</script>"
    css_regex = r"<style.*>[\S\s]+?</style>"
    html_sans_js = re.sub(js_regex, '', html)
    html_sans_js_and_css = re.sub(css_regex, '', html_sans_js)
    # Step 2: Convert to raw text -- separating sections with a newline
    raw = BeautifulSoup(html_sans_js).get_text('\n')
    # Step 2.b: Remove leftover css classes which litter the text
    css_regex_classes = r"\. .*{[\S\s]+?}"
    raw = re.sub(css_regex_classes, '', raw)
    # Step 3: Get the difference
    known_words = set([w for w in nltk.corpus.words.words('en') if w.islower()])
    words_on_webpage = set(re.findall(r"\b[a-z]+\b", raw))
    unknown_words = sorted(words_on_webpage.difference(known_words))
    print(f'Total Words from webpage: {len(words_on_webpage)}')
    print(f'Total unknown words: {len(unknown_words)}')
    print(f'Percentage unknown words: {len(unknown_words)/len(words_on_webpage)}')
    # Manual investigation of results highlights that the Words Corpus doesn't include
    # variations on word lemmas
    # Let's remove those variations for a better idea of how many new 'terms' appear on
    # our webpage
    word_versions = re.findall(r'\b[a-z]+(?:ing|ly|ed|ious|ies|ive|es|s|ment)', '
        '.join(list(unknown_words)))
    unknown_words_removing_variations_on_lemmas =
        set(unknown_words).difference(set(word_versions))
```

```
print(f'Total unknown words, removing common variations on word lemmas:
      {len(unknown_words_removing_variations_on_lemmas)}')
print(f'Percentage unknown words, after removing common variations on word lemmas:
      {len(unknown_words_removing_variations_on_lemmas)/len(words_on_webpage)}')
print(unknown_words_removing_variations_on_lemmas)
return unknown_words_removing_variations_on_lemmas
```

---

## Problem 29

Readability measures are used to score the reading difficulty of a text, for the purposes of selecting texts of appropriate difficulty for language learners. Let us define  $w$  to be the average number of letters per word, and  $s$  to be the average number of words per sentence, in a given text. The Automated Readability Index (ARI) of the text is defined to be:  $4.71 w + 0.5 s - 21.43$ . Compute the ARI score for various sections of the Brown Corpus, including section f (lore) and j (learned). Make use of the fact that `nlk.corpus.brown.words()` produces a sequence of words, while `nlk.corpus.brown.sents()` produces a sequence of sentences.

*Solution:*

My results are as follows:

ARI score for adventure is: 4.0841684990890705  
ARI score for belles\_lettres is: 10.987652885621749  
ARI score for editorial is: 9.471025332953673  
ARI score for fiction is: 4.9104735321302115  
ARI score for government is: 12.08430349501021  
ARI score for hobbies is: 8.922356393630267  
ARI score for humor is: 7.887805248319808  
ARI score for learned is: 11.926007043317348  
ARI score for lore is: 10.254756197101155  
ARI score for mystery is: 3.8335518942055167  
ARI score for news is: 10.176684595052684  
ARI score for religion is: 10.203109907301261  
ARI score for reviews is: 10.769699888473433  
ARI score for romance is: 4.34922419804213  
ARI score for science\_fiction is: 4.978058336905399

---

```
import nltk
from nltk.corpus import brown

def ARI_for_text(text, category=""):
    sents = text.sents(categories=category) if category != "" else text.sents()
    words = text.words(categories=category) if category != "" else text.words()
    chars = ''.join(words)

    # the average number of letters per word
    mu_w = len(chars) / len(words)
    # the average number of words per sentence
    mu_s = len(words) / len(sents)

    ari = (4.71 * mu_w) + (0.5 * mu_s) - 21.43
    return ari

if __name__ == "__main__":
    for category in brown.categories():
        print(f'ARI score for {category} is: {ARI_for_text(brown, category)}')
```

---

## Intro to XML

Read A Gentle Introduction to XML Install this TEI reader and input a TEI XML file. Perform exercise 42 (above) on the raw text of the TEI XML file. You can find any TEI XML text but you could start with an English translation found [here](#).

*Solution:*

Solution goes here

---

```
>>> Code Goes Here
```

```
>>> More crap here
```

---

## Investigating Text Resources

Identify a possible available textual source that you might use for a project and identify the possible challenges that you will face in preprocessing the text. You can start with the sources found [here](#), a list that will expand during the course of the semester.

*Solution:*

Solution goes here

---

```
>>> Code Goes Here
```

```
>>> More crap here
```

---