index

May 3, 2022

1 Corpus Statistics - Lab

1.1 Introduction

In this lab, we'll learn how to use various NLP techniques to generate descriptive statistics to explore a text corpus!

1.2 Objectives

You will be able to:

- Generate common corpus statistics using NLTK
- Use a count vectorization strategy to create a bag of words
- Compare two different text corpora using corpus statistics generated by NLTK

1.3 Getting Started

In this lab, we'll load two different text corpora from NLTK's library of various texts, and then explore and compare each corpus using some basic statistical measures and techniques common in NLP. Let's get started!

In the cell below:

- Import `nltk`
- Import `gutenberg` and `stopwords` from `nltk.corpus`
- Import everything (`*`) from `nltk.collocations`
- Import `FreqDist` and `word_tokenize` from `nltk`
- Import the `string` and `re` libraries

[3]:

```
import nltk
from nltk.corpus import gutenberg, stopwords
from nltk.collocations import *
from nltk import FreqDist, word_tokenize
import string
import re
```

[26]:

```
nltk.download('gutenberg')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package gutenberg to
[nltk_data]     /Users/miladshirani/nltk_data…
```

```
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/miladshirani/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[26]:
```
True
```

Now, let's take a look at the corpora available to us. There are many, many corpora available inside of nltk's `corpus` module. For this lab, we'll make use of the texts contained in `corpus.gutenberg`– 18 different (complete) corpora that can be found on the Project Gutenberg website.

To see the file ids for each of the corpora inside of `gutenberg`, we can call the `.fileids()` method. Do this now in the cell below.

[8]:
```python
file_ids = gutenberg.fileids()
file_ids
```

[8]:
```
['austen-emma.txt',
 'austen-persuasion.txt',
 'austen-sense.txt',
 'bible-kjv.txt',
 'blake-poems.txt',
 'bryant-stories.txt',
 'burgess-busterbrown.txt',
 'carroll-alice.txt',
 'chesterton-ball.txt',
 'chesterton-brown.txt',
 'chesterton-thursday.txt',
 'edgeworth-parents.txt',
 'melville-moby_dick.txt',
 'milton-paradise.txt',
 'shakespeare-caesar.txt',
 'shakespeare-hamlet.txt',
 'shakespeare-macbeth.txt',
 'whitman-leaves.txt']
```

Great! For the first part of this lab, we'll be working with Shakespeare's *Macbeth*, a tragedy about a pair of ambitious social climbers.

To load the actual corpus, we need to pass in the file id for macbeth into `gutenberg.raw()`.

Do this now in the cell below. Then, print the first 1000 characters of the text to ensure it loaded correctly, and get a feel for what our text data looks like.

[13]:
```python
macbeth_text = gutenberg.raw(file_ids[-2])
print(macbeth_text[:1000])
```

```
[The Tragedie of Macbeth by William Shakespeare 1603]
```

```
Actus Primus. Scoena Prima.

Thunder and Lightning. Enter three Witches.

  1. When shall we three meet againe?
In Thunder, Lightning, or in Raine?
  2. When the Hurley-burley's done,
When the Battaile's lost, and wonne

   3. That will be ere the set of Sunne

   1. Where the place?
  2. Vpon the Heath

   3. There to meet with Macbeth

   1. I come, Gray-Malkin

   All. Padock calls anon: faire is foule, and foule is faire,
Houer through the fogge and filthie ayre.

Exeunt.


Scena Secunda.

Alarum within. Enter King Malcome, Donalbaine, Lenox, with
attendants,
meeting a bleeding Captaine.

  King. What bloody man is that? he can report,
As seemeth by his plight, of the Reuolt
The newest state

   Mal. This is the Serieant,
Who like a good and hardie Souldier fought
'Gainst my Captiuitie: Haile braue friend;
Say to the King, the knowledge of the Broyle,
As thou didst leaue it

   Cap. Doubtfull it stood,
As two spent Swimmers, t
```

Question: Look at the text snippet above. What do you notice about it? Are there any issues you see that we'll need to deal with during the preprocessing steps?

Write your answer below this line: ____

Yes, there are. Some of the words are hyphenated. If we just use basic tokenization, then it will split hyphenated words into individual tokens. There are also numbers that act as metadata about which witch is speaking – we'll need to remove these.

1.3.1 Preprocessing the Data

Looking at the text output above shows us a few things that we'll need to deal with during the preprocessing and tokenization steps – specifically:

- Capitalization – we'll need to lowercase all words.
- Apostrophes – we'll need to write some basic regex in order to capture words that contain apostrophes as a single token. In the interest of time, a pattern has been provided for you. Use the following pattern: `"([a-zA-Z]+(?:'[a-z]+)?)"`
- Numbers – We'll want to remove these, as they generally appear as stage direction to tell us which witch is speaking.

In the cell below:

- Store the pattern shown above in the appropriate variable

- Use `nltk.regexp_tokenize()` and pass in our text and the `pattern`

[17]:
```
pattern = "([a-zA-Z]+(?:'[a-z]+)?)"
macbeth_tokens_raw = nltk.regexp_tokenize(macbeth_text, pattern)
print(type(macbeth_tokens_raw))
```

```
<class 'list'>
```

Great! Now that we have our tokens, we need to lowercase them. In the cell below, use a list comprehension and the `.lower()` method on every word token in `macbeth_tokens_raw`. Store this inside `macbeth_tokens`.

[18]:
```
macbeth_tokens = [item.lower() for item in macbeth_tokens_raw]
```

1.4 Frequency Distributions

Now that we've done some basic cleaning and tokenization, let's go ahead and create a ***Frequency Distribution*** to see the number of times each word is used in this play. This frequency distribution is an example of a ***Bag of Words***, which you've worked with in previous labs.

In the cell below:

- Use `FreqDist()` and pass in `macbeth_tokens` as the input
- Display the frequency distribution to see what it looks like

[19]:
```
macbeth_freqdist = FreqDist(macbeth_tokens)
macbeth_freqdist.most_common(50)
```

[19]:
```
[('the', 649),
 ('and', 545),
 ('to', 383),
 ('of', 338),
```

```
('i', 331),
('a', 241),
('that', 227),
('my', 203),
('you', 203),
('in', 199),
('is', 180),
('not', 165),
('it', 161),
('with', 153),
('his', 146),
('be', 137),
('macb', 137),
('your', 126),
('our', 123),
('haue', 122),
('but', 120),
('me', 113),
('he', 110),
('for', 109),
('what', 106),
('this', 104),
('all', 99),
('so', 96),
('him', 90),
('as', 89),
('thou', 87),
('we', 83),
('enter', 81),
('which', 80),
('are', 73),
('will', 72),
('they', 70),
('shall', 68),
('no', 67),
('then', 63),
('macbeth', 62),
('their', 62),
('thee', 61),
('vpon', 58),
('on', 58),
('macd', 58),
('from', 57),
('yet', 57),
('thy', 56),
('vs', 55)]
```

Well, that doesn't tell us very much! The top 10 most used words in macbeth are all ***Stop Words***. They don't contain any interesting information, and essentially just act as the "connective tissue" between the words that really matter in any text. Let's try removing the stopwords and punctuation, and then creating another frequency distribution that contains only the important words.

1.5 Removing Stop Words and Punctuation

We've already imported the `stopwords` module. We can access all of the stopwords using the `stopwords.words()` method – however, we don't want to use the whole thing, as this contains all stopwords in every language supported by NLTK. We don't need to check for and remove any Finnish or Japanese stop words, as this text is in English. To avoid unnecessarily long run-times, we'll just use the English subset of stopwords by passing in the parameter `"english"` into `stopwords.words()`.

In the cell below:

- Get all the `'english'` stopwords from `stopwords.words()` and store them in the appropriate variable below. They will be stored as a list, by default

- We'll also want to remove all punctuation. Create a list version of `string.punctuation` and add it to our stopwords list

- Finally, we'll also remove numbers. Create a list that contains numbers 0-9 (as strings!), and add this to the stopwords list as well

- Use another list comprehension to get words out of `macbeth_tokens` as long as they are not in `stopwords_list`

[28]:
```
stopwords_list = stopwords.words("english")
stopwords_list += list(string.punctuation)
stopwords_list += ['0, ''1', '2', '3', '4', '5', '6', '7', '8', '9']

macbeth_words_stopped = [item for item in macbeth_tokens if item not in␣
↪stopwords_list]
```

Great! Now, let's create another frequency distribution using `macbeth_words_stopped`, and then inspect the top 50 most common words, to see if removing stopwords and punctuation has helped.

Do this now in the cell below.

[29]:
```
macbeth_stopped_freqdist = FreqDist(macbeth_words_stopped)
macbeth_stopped_freqdist.most_common(50)
```

[29]:
```
[('macb', 137),
 ('haue', 122),
 ('thou', 87),
 ('enter', 81),
 ('shall', 68),
 ('macbeth', 62),
```

```
('thee', 61),
('vpon', 58),
('macd', 58),
('yet', 57),
('thy', 56),
('vs', 55),
('come', 54),
('king', 54),
('hath', 52),
('good', 49),
('rosse', 49),
('lady', 48),
('would', 47),
('time', 46),
('like', 43),
('say', 39),
('doe', 38),
('lord', 38),
('make', 38),
('tis', 37),
('must', 36),
('done', 35),
('selfe', 35),
('ile', 35),
('feare', 35),
('let', 35),
('man', 34),
('wife', 34),
('night', 34),
('banquo', 34),
('well', 33),
('know', 33),
('one', 32),
('great', 31),
('see', 31),
('may', 31),
('exeunt', 30),
('speake', 29),
('sir', 29),
('lenox', 28),
('mine', 26),
('vp', 26),
('th', 26),
('mal', 25)]
```

This is definitely an improvement! You may be wondering why `'Macb'` shows up as the number 1 most used token. If you inspect Macbeth on project gutenberg and search for `'Macb'`, you'll soon discover that the source text denotes Macb as stage direction for any line spoken by Macbeth's character. This means that 'Macb' is actually stage direction, meaning that under normal circumstances, we would need to ask ourselves if it is worth it to remove it or keep it. In the interest of time for this lab, we'll leave it be.

1.6 Answering Questions about our Corpus

Now that we have a frequency distribution, we can easily answer some basic questions about the text. Let's answer some basic questions about Macbeth below, before we move onto creating bigrams.

1.6.1 Vocabulary Size

What is the size of the total vocabulary used in Macbeth, once all stopwords have been removed?

Compute this in the cell below.

[37]:
```
print(len(macbeth_words_stopped))
print(sum(macbeth_stopped_freqdist.values()))
print(len(macbeth_stopped_freqdist))
print(type(macbeth_stopped_freqdist))
```

```
10115
10115
3441
<class 'nltk.probability.FreqDist'>
```

1.6.2 Normalized Word Frequency

Knowing the frequency with which each word is used is somewhat informative, but without the context of how many words are used in total, it doesn't tell us much. One way we can adjust for this is to use ***Normalized Word Frequency***, which we can compute by dividing each word frequency by the total number of words.

Compute this now in the cell below, and display the normalized word frequency for the top 50 words.

[38]:
```
total_word_count = len(macbeth_words_stopped)
macbeth_top_50 = macbeth_stopped_freqdist.most_common(50)
print('Word\t\t\tNormalized Frequency')
for word in macbeth_top_50:
    normalized_frequency = word[1]/total_word_count * 100
    print('{} \t\t\t {:.4}'.format(word[0], normalized_frequency))
```

```
Word                    Normalized Frequency
macb                     1.354
haue                     1.206
thou                     0.8601
enter                    0.8008
```

shall	0.6723
macbeth	0.613
thee	0.6031
vpon	0.5734
macd	0.5734
yet	0.5635
thy	0.5536
vs	0.5437
come	0.5339
king	0.5339
hath	0.5141
good	0.4844
rosse	0.4844
lady	0.4745
would	0.4647
time	0.4548
like	0.4251
say	0.3856
doe	0.3757
lord	0.3757
make	0.3757
tis	0.3658
must	0.3559
done	0.346
selfe	0.346
ile	0.346
feare	0.346
let	0.346
man	0.3361
wife	0.3361
night	0.3361
banquo	0.3361
well	0.3262
know	0.3262
one	0.3164
great	0.3065
see	0.3065
may	0.3065
exeunt	0.2966
speake	0.2867
sir	0.2867
lenox	0.2768
mine	0.257
vp	0.257
th	0.257
mal	0.2472

1.7 Creating Bigrams

Knowing individual word frequencies is somewhat informative, but in practice, some of these tokens are actually parts of larger phrases that should be treated as a single unit. Let's create some bigrams, and see which combinations of words are most telling.

In the cell below:

- We'll begin by aliasing a particularly long method name to make it easier to call. Store `nltk.collocations.BigramAssocMeasures()` inside of the variable `bigram_measures`
- Next, we'll need to create a ***finder***. Pass `macbeth_words_stopped` into `BigramCollocationFinder.from_words()` and assign the result to `macbeth_finder`
- Once we have a finder, we can use it to compute bigram scores, so we can see the combinations that occur most frequently. Call the `macbeth_finder` object's `score_ngrams()` method and pass in `bigram_measures.raw_freq` as the input

- Display first 50 elements in the `macbeth_scored` list to see the 50 most common bigrams in macbeth

[40]:
```
bigram_measures = nltk.collocations.BigramAssocMeasures()
```

[41]:
```
macbeth_finder = BigramCollocationFinder.from_words(macbeth_words_stopped)
```

[42]:
```
macbeth_scored = macbeth_finder.score_ngrams(bigram_measures.raw_freq)
```

[43]:
```
# Display the first 50 elements of macbeth_scored
macbeth_scored[:50]
```

[43]:
```
[(('enter', 'macbeth'), 0.0015818091942659417),
 (('exeunt', 'scena'), 0.0014829461196243204),
 (('thane', 'cawdor'), 0.0012852199703410777),
 (('knock', 'knock'), 0.0009886307464162135),
 (('lord', 'macb'), 0.0008897676717745922),
 (('thou', 'art'), 0.0008897676717745922),
 (('good', 'lord'), 0.0007909045971329708),
 (('haue', 'done'), 0.0007909045971329708),
 (('macb', 'haue'), 0.0007909045971329708),
 (('enter', 'lady'), 0.0006920415224913495),
 (('let', 'vs'), 0.0006920415224913495),
 (('macbeth', 'macb'), 0.0005931784478497281),
 (('enter', 'malcolme'), 0.0004943153732081067),
 (('enter', 'three'), 0.0004943153732081067),
 (('euery', 'one'), 0.0004943153732081067),
 (('macb', 'ile'), 0.0004943153732081067),
 (('macb', 'thou'), 0.0004943153732081067),
 (('make', 'vs'), 0.0004943153732081067),
 (('mine', 'eyes'), 0.0004943153732081067),
 (('mine', 'owne'), 0.0004943153732081067),
 (('scena', 'secunda'), 0.0004943153732081067),
```

```
(('three', 'witches'), 0.0004943153732081067),
(('thy', 'selfe'), 0.0004943153732081067),
(('worthy', 'thane'), 0.0004943153732081067),
(('would', 'haue'), 0.0004943153732081067),
(('borne', 'woman'), 0.0003954522985664854),
(('come', 'come'), 0.0003954522985664854),
(('enter', 'banquo'), 0.0003954522985664854),
(('enter', 'king'), 0.0003954522985664854),
(('enter', 'macduffe'), 0.0003954522985664854),
(('enter', 'rosse'), 0.0003954522985664854),
(('haile', 'king'), 0.0003954522985664854),
(('haile', 'macbeth'), 0.0003954522985664854),
(('hath', 'made'), 0.0003954522985664854),
(('haue', 'seene'), 0.0003954522985664854),
(('macb', 'bring'), 0.0003954522985664854),
(('macbeth', 'macbeth'), 0.0003954522985664854),
(('malcolme', 'donalbaine'), 0.0003954522985664854),
(('old', 'man'), 0.0003954522985664854),
(('rosse', 'angus'), 0.0003954522985664854),
(('scena', 'prima'), 0.0003954522985664854),
(('see', 'thee'), 0.0003954522985664854),
(('shew', 'shew'), 0.0003954522985664854),
(('sir', 'macb'), 0.0003954522985664854),
(('ten', 'thousand'), 0.0003954522985664854),
(('tertia', 'enter'), 0.0003954522985664854),
(('thy', 'face'), 0.0003954522985664854),
(('woman', 'borne'), 0.0003954522985664854),
(('would', 'make'), 0.0003954522985664854),
(('alarums', 'enter'), 0.00029658922392486405)]
```

These look a bit more interesting. We can see here that some of the most common ones are stage directions, such as 'Enter Macbeth' and 'Exeunt Scena', while others seem to be common phrases used in the play.

To wrap up our initial examination of *Macbeth*, let's end by calculating ***Mutual Information Scores***.

1.8 Using Mutual Information Scores

To calculate mutual information scores, we'll need to first create a frequency filter, so that we only examine bigrams that occur more than a set number of times – for our purposes, we'll set this limit to 5.

In NLTK, mutual information is often referred to as `pmi`, for ***Pointwise Mutual Information***. Calculating PMI scores works much the same way that we created bigrams, with a few notable differences.

In the cell below:

- We'll start by creating another finder for pmi. Pass `macbeth_words_stopped` as the input to

BigramCollocationFinder.from_words(). Store this is the variable macbeth_pmi_finder

- Once we have our finder, we'll need to apply our frequency filter. Call macbeth_pmi_finder's apply_freq_filter and pass in the number 5 as the input
- Now, we can use the finder to calculate pmi scores. Use the pmi finder's .score_ngrams() method, and pass in bigram_measures.pmi as the argument. Store this in macbeth_pmi_scored
- Examine the first 50 elements in macbeth_pmi_scored

[46]:
```
macbeth_pmi_finder = BigramCollocationFinder.from_words(macbeth_words_stopped)
```

[47]:
```
macbeth_pmi_finder.apply_freq_filter(5)
```

[48]:
```
macbeth_pmi_scored = macbeth_pmi_finder.score_ngrams(bigram_measures.pmi)
```

[49]:
```
macbeth_pmi_scored[:50]
```

[49]:
```
[(('three', 'witches'), 8.925697076191916),
 (('scena', 'secunda'), 8.844777080808349),
 (('knock', 'knock'), 8.62613679433301),
 (('thane', 'cawdor'), 7.968474805033251),
 (('exeunt', 'scena'), 7.844777080808349),
 (('mine', 'eyes'), 7.46626545755462),
 (('worthy', 'thane'), 6.982280604558284),
 (('mine', 'owne'), 6.838234234941577),
 (('euery', 'one'), 6.626136794333009),
 (('thou', 'art'), 5.861265203596917),
 (('enter', 'malcolme'), 5.585847073307292),
 (('enter', 'three'), 5.585847073307292),
 (('good', 'lord'), 5.441571341886851),
 (('let', 'vs'), 5.2009208910336255),
 (('enter', 'macbeth'), 5.0101623861741444),
 (('thy', 'selfe'), 4.689498855330438),
 (('make', 'vs'), 4.596849567364764),
 (('haue', 'done'), 4.2441883449377915),
 (('enter', 'lady'), 4.186751117897471),
 (('lord', 'macb'), 4.128174104483847),
 (('macb', 'ile'), 3.3988216944275145),
 (('would', 'haue'), 3.1408106050924847),
 (('macbeth', 'macb'), 2.836942806819401),
 (('macb', 'haue'), 2.2754392789222315),
 (('macb', 'thou'), 2.0851612155237547)]
```

1.9 On Your Own: Comparative Corpus Statistics

Now that we've worked through generating some baseline corpus statistics for one corpus, it's up to you to select a second corpus and generate your own corpus statistics, and then compare and contrast the two. For simplicity's sake, we recommend you stick to a corpus from nltk.corpus.gutenberg – although comparing the diction found in a classic work of fiction to something like a presidential State of the Union address could be interesting, it's not really an apples-to-apples comparison, and those corpora could also require additional preprocessing steps that are outside the scope of this lab.

In the cells below:

1. Select another corpus from `gutenberg.fileids()`
2. Clean, preprocess, tokenize, and generate corpus statistics for this new corpus
3. Perform a comparative analysis using the Macbeth statistics we generated above and your new corpus statistics. How are they similar? How are they different? Was there anything interesting or surprising that you found in your comparison? Create at least one meaningful visualization comparing the two corpora

[56]:
```
macbeth_text_new = gutenberg.raw(file_ids[-5])
# print(macbeth_text_new[:1000])

pattern = "([a-zA-Z]+(?:'[a-z]+)?)"
macbeth_tokens_raw_new = nltk.regexp_tokenize(macbeth_text_new, pattern)
```

[60]:
```
macbeth_tokens_new = [word.lower() for word in macbeth_tokens_raw_new]

macbeth_freqdist_new = FreqDist(macbeth_tokens_new)
macbeth_freqdist_new.most_common(10)
```

[60]:
```
[('and', 3395),
 ('the', 2968),
 ('to', 2228),
 ('of', 2050),
 ('in', 1366),
 ('his', 1170),
 ('with', 1160),
 ('or', 715),
 ('that', 704),
 ('all', 700)]
```

[63]:
```
stopwords_list = stopwords.words('english')
stopwords_list += list(string.punctuation)
stopwords_list += ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

macbeth_words_stopped_new = [word for word in macbeth_tokens_new
                             if word not in stopwords_list]
```

[66]:
```
macbeth_stopped_freqdist_new = FreqDist(macbeth_words_stopped_new)
# macbeth_stopped_freqdist_new.most_common(10)

total_word_count = len(macbeth_words_stopped_new)
```

```
macbeth_top_new_10 = macbeth_stopped_freqdist_new.most_common(10)
print('Word\t\t\tNormalized Frequency')
for word in macbeth_top_new_10:
    normalized_frequency = word[1] / total_word_count * 100
    print('{} \t\t\t {:.4}'.format(word[0], normalized_frequency))
```

```
Word                    Normalized Frequency
thou                     0.9485
thy                      0.909
heaven                   0.8036
thee                     0.786
thus                     0.6982
shall                    0.6213
god                      0.5379
yet                      0.5006
though                   0.4764
earth                    0.4545
```

[67]:
```
bigram_measures = nltk.collocations.BigramAssocMeasures()
macbeth_finder_new = BigramCollocationFinder.
    ↪from_words(macbeth_words_stopped_new)
macbeth_scored_new = macbeth_finder_new.score_ngrams(bigram_measures.raw_freq)
macbeth_scored_new[:10]
```

[67]:
```
[(('thou', 'hast'), 0.0007245422210512449),
 (('heaven', 'earth'), 0.0006147630966495412),
 (('let', 'us'), 0.0005269397971281781),
 (('thou', 'art'), 0.0005269397971281781),
 (('hast', 'thou'), 0.0004171606727264743),
 (('good', 'evil'), 0.0003293373732051113),
 (('thus', 'eve'), 0.0003293373732051113),
 (('thou', 'seest'), 0.0003073815483247706),
 (('thus', 'adam'), 0.0003073815483247706),
 (('right', 'hand'), 0.0002854257234444298)]
```

[71]:
```
macbeth_pmi_finder_new = BigramCollocationFinder.
    ↪from_words(macbeth_words_stopped_new)
macbeth_pmi_finder_new.apply_freq_filter(5)
macbeth_pmi_scored_new = macbeth_pmi_finder_new.score_ngrams(bigram_measures.
    ↪pmi)
macbeth_pmi_scored_new[:10]
```

[71]:
```
[(('ten', 'thousand'), 10.494489099567632),
 (('fish', 'fowl'), 10.396085395506638),
 (('drew', 'nigh'), 8.848597600204146),
 (('bird', 'beast'), 8.796964831788824),
 (('arch', 'angel'), 8.667681814843855),
```

```
(('dry', 'land'), 8.532522231562222),
(('living', 'creatures'), 8.081002841533678),
(('human', 'sense'), 8.006267253684825),
(('mine', 'eyes'), 7.6827887072340655),
(('beast', 'field'), 7.667681814843856)]
```

1.10 Summary

In this lab, we used our newfound NLP skills to generate some statistics specific to text data, and used them to compare two different works!