TOOL

NLP Fundamentals: Next Steps

Computer programming is a learned concept that must be practiced and maintained. At first, it can be hard, awkward and even unnatural, just like learning tennis or chess. The good news is that practice makes perfect: regular problem solving improves your skills.

Below, you will find additional exercises (without solutions) to help you advance your Python coding skills in the NLP domain. They are organized by module, and then further organized by skill. Now that you have completed this course, use these exercises to hone your skills. Have fun with them!

You can find a good Python tutorial and additional exercises in this NLTK open source book as well. Python.org also has a great section for beginners and this tutorial.

Module 1

Further Practice: Operations on Strings

To practice your skills in string manipulation, try the exercises below on small and large string documents.

- A small document can be a single sentence, say "The Quick Brown Fox Jumps Over The Lazy Dog's Back 1234567890 Times!" or something more complex that allows you to test if your code works correctly.
- A large document can be a book from the Gutenberg corpus, say "Alice in Wonderland", which you can load into a variable `sDoc` to test your code on a larger scale. Debugging and troubleshooting with a large document is trickier, so try out your code on short observable sentences first.

```
import nltk
_ = nltk.download([‘gutenberg’], quiet=True)
sFox = ‘The Quick Brown Fox Jumps Over The Lazy Dog’s Back 1234567890
Times!’
sAlice = nltk.corpus.gutenberg.raw(‘carroll-alice.txt’)
print(sAlice[:200])
```

Try the following exercises with your own string, and then in `sAlice`. See the str methods manual for helpful functions.

1. Letter casing

- Lowercase every other capitalized word.
- Find the length of the last word in a string.
- Uppercase words only if they start a sentence.
- Build a string of words that have mixed capitalization.
- Delete words that have mixed capitalization.
- Replace all titled words with asterisks of the same length, such as: '***I like NLP***' → '**** like ******'

2. Splitting and joining

- Split a string on whitespace.
- Split a doc on space character, remove all single-character words and join the list back to a single string.
- Split a doc on **'. '** and remove those longer than 10 characters.
- Split a doc into paragraphs and return only the smallest, which contains '***Alice***'.
- Replace '***Alice***' with '***Alisa***'. **Hint**: split at one word and join with another word.
- How many words are there before the first occurrence of '***, Alice***'? **Hint:** use **partition()**.

3. Counting

- Count number of times '***, Alice***' appears.
- What about the count of '***Alice***', which start sentences?
- Count the number of titled words in a document.
- Count the number of substrings, which are equal to the first 4-letter string.
- Count one-digit numbers in a document, such as: '***1 and 23 and 4,567***' has only one single digit.

4. Substring search. Try it in `sAlice`

- Find the position of the third phrase, which starts with '***the***'.
- Find all words ending with '***ing!***'.
- Find all words ending with '***ly***' and over six characters.

- How many times does the longest word starting with '***cur***' appear, regardless of letter casing?
- Find all words that start with any prefix in tuple ('***the***','***whe***','***diff***') in any letter case and are more than five characters long.
- Find all quotes containing '***Alice***'.

5. Indexing

- Remove 5 characters from each side of the doc.
- Find the start index of the penultimate word in a doc.
- Find the position of the last two repeated letters, such as '***aa***', '***bb***', … Then replace these with '**'.
- Find the central word in a doc with equal number of characters on each side.
- Find the central word in a doc with approximately equal number of letter words on each side.
- Remove the central word from a doc.

6. String testing

- Return a string of all digits retrieved from a doc.
- Return a list of all characters preceding any space character. Swap case for titled words.

Further Practice: Regex

Repeat all of the above exercises, but this time use the `re` library's methods.

Module 2

Further Practice: Tokenization

Use the following starter code, or create your own string expression.

```
import nltk, pandas as pd, numpy as np, collections
_ = nltk.download(['gutenberg','punkt'], quiet=True)
sFox = 'The Quick Brown Fox Jumps Over The Lazy Dog's Back 1234567890
Times!'
sAlice = nltk.corpus.gutenberg.raw('carroll-alice.txt')
print(sAlice[:200])
```

1. Split `sAlice` by space and by whitespace.

- Which of these two lists has more tokens?
- How many tokens do the lists have in common?
- Which tokens are in the former output that are not in the latter (and vice versa)? **Hint:** convert lists to sets and use a set's difference operation.
- What are the shortest and longest tokens produced by each method?

2. Split `sAlice` with `nltk.word_tokenize`.

- What are the shortest and the longest tokens?
  - Do they appear as words you would find in a dictionary?
  - Repeat this task after lowercasing and deduping. **Hint**: `set()`
- What are the most common tokens? **Hint**: Use `collections.Counter` or `nltk.probability.FreqDist`.
  - Do they appear as reasonable and meaningful words to you? What are the top 10 most frequent letter words?
  - Would you consider these specific or generic words?

3. Split `sAlice` by '. '

- What are the shortest and the longest sentences?
- How many sentences are there?
- Do they appear as reasonable sentences to you?
- How many sentences do you count in the largest sentence?

4. Split `sAlice` by '. ' or by line boundaries.

Hint: try splitlines, then join on '. ', then split on '. ' again. Also try re.split.

- What are the shortest and the longest sentences?
- How many sentences are there?
- Do they appear as reasonable sentences to you?
- How many sentences do you count in the largest sentence?

Further Practice: Preprocessing

Apply a preprocessing pipeline to the ***Alice in Wonderland*** document with the goal of achieving the smallest lexicon of most meaningful words. You would need to investigate the words that result from each preprocessing step. Try avoiding hard-coded stopwords, and instead focus on more robust scrubbing techniques, such as better contraction expansion and lemmatization. Can you identify and filter generic and meaningful words based on their frequencies, capitalization, proximity to start/end/middle of a sentence, word length, and more?

Module 3

Further Practice: Tagging

```
import nltk, pandas as pd, numpy as np, collections
_ = nltk.download([‘gutenberg’,‘punkt’], quiet=True)
sFox = ‘The Quick Brown Fox Jumps Over The Lazy Dog’s Back 1234567890 
Times!’
sAlice = nltk.corpus.gutenberg.raw(‘carroll-alice.txt’)
print(sAlice[:200])
```

This online NLTK book has helpful examples on use of POS tagging.

1. What is the distribution of POS tags in a document? In other words, how many verbs, nouns, adjectives, etc.?
2. What is the distribution of pronouns?
3. What is the distribution of first person pronouns?
4. What is the average distance (as a number of tokens) between a verb and the nearest noun? Between an adjective and the nearest noun?
5. What fraction of verbs are in past tense?
6. What fraction of nouns are plural?
7. What is the distribution of names?
8. What fraction of proper nouns is the name ‘***Alice***’?