20-00-0546-iv Foundations of Language Technology

Handout 2
Language Processing and Python

12. November, 2020

In this tutorial, we will take on some linguistically motivated programming tasks without necessarily explaining how they work. You should get a feeling for the type of questions we are interested in, how Python and NLTK address, them and try to interpret the resulting numbers. This assignment will give you practice in the basics of using Python and NLTK.

The deadline for the homework is **Thursday, 19.11.20 18:00 CET**. In case your submission consists of several files, compress these to a `zip`-file. Indicate clearly which submission corresponds to which question. Include comments in your program code to make it easier readable. It is very important that you submit your solution as a **Jupyter Notebook file (.ipnyb)**.

2.1 Warm up

Question 2.1 *Token/Type/Vocabulary: Which of the following statements are true?*

(1) Every token is a type.
(2) The vocabulary of a text consists of all tokens.
(3) The vocabulary of a text consists of all types.
(4) The vocabulary of a text consists of the union of all tokens and all types.
(5) The vocabulary of a text consists of the intersection of all tokens and all types.

Question 2.2 *Lists in Python: Let l and m be lists of words. Which of the following statements are syntactically correct?*

(1) `l.index("hi")`
(2) `l[-2]`
(3) `x = l + m`
(4) `x = l & m`
(5) `l[1,4]`

Question 2.3 *Bigram: Which of the following sublists are NLTK-style bigrams of the text 'Python is powerful'?*

(1) `['Py','th','on','is','po','we','rf','ul']`
(2) `[('Python','is'),('is','powerful')]`
(3) `(('Python','is'),('is','powerful'))`
(4) `[('','Python'),('Python','is'),('is','powerful'),('powerful','')]`
(5) `[('Python','is'),('Python','powerful'),('is','Python'),('is','powerful'),('powerful','Python'),('powerful','is')]`

Question 2.4 *Operation on elements: What is the output of the following code:*

```python
t = ['Python', 'is', 'powerful']
[len(w) for w in t]
```

(1) 16
(2) 3
(3) `[6,2, 8]`
(4) `(6,2, 8)`
(5) syntax error

2.2 Analyzing texts

NLTK offers a wide range of predefined methods, which are helpful to analyze a given text. Make yourself comfortable with those methods. Enter the following statements into the editor and execute them. These imports are the base for todays tutorial. Make sure, you understand each line.

Listing 1: Basic Imports

```python
import nltk
from nltk import *
from nltk.book import *
```

Task 2.1 *Consider the following text: "Today it's nice weather. Is it not nice today :-) ?"*

(a) What is the length of this text? What may be counted?
(b) What are reasonable tokens?
(c) State the types and the vocabulary of this short text according to your chosen tokens.
(d) Represent this text as a list of tokens in Python. Save it, as you will need it in task 2.4

Task 2.2 *Explore some texts provided by NLTK:*

(a) To avoid slow reactions do the following with a sublist of text1, i.e. `t = text1[:500]` *and explain its meaning.*

Listing 2: Set, Sort, Length

```python
t = text1[:500]
print(t)
print(len(t))
print(sorted(t))
print(set(t))
print(sorted(set(t)))
print(len(set(t)))
print(list(bigrams(t)))
```

(b) `text1.concordance("monstrous")` *shows every occurrence of the word* monstrous *with some context. Apply the same to text2.*[1]

(c) `text1.similar("monstrous")` *shows what other words appear in a similar range of contexts. Apply the same to text2.*[2]

(d) `text1.collocations()` *shows words that occur together unusually often, e.g. red wine. Apply the same to text5.*[3]

(e) Define a function called `vocab_size(text)` *that has a single parameter for the text, and which returns the vocabulary size of the text. Apply it to text1 and text2.* [4]

(f) Define the lexical diversity as introduced in the lecture and apply it to text1 and text2.[5]

(g) Search the book "Sense and Sensibility" (text2) for the word "affection", using `text2.concordance("affection")`*. Search the book of "Genesis" to find out how long some people lived, using* `text3.concordance("lived")`*. You could look at text4, the Inaugural Address Corpus, to see examples of English going back to 1789, and search for words like "nation", "terror", "god" to see how these words have been used differently over time. Analyze text5, the NPS Chat Corpus: search this for unconventional words like "im", "ur", "lol".*[6]

Task 2.3 *Try the frequency distribution introduced in the lecture with another text, e.g. text2. Find the frequent and the infrequent words. Try to extract the keywords of the text (what is the text about). Implement this functionality in a separate function.*[7]

[1]See NLTK-book page chapter 1, page 4
[2]See NLTK-book page chapter 1, "Your Turn", page 5
[3]See NLTK-book page chapter 1, page 20
[4]See NLTK-book page chapter 1, exercise 27, page 37
[5]See NLTK-book page chapter 1, page 9
[6]See NLTK-book page chapter 1, "Your Turn", page 5
[7]See NLTK-book page chapter 1, "Your Turn", page 18

Task 2.4 *Find out, which types of task 2.1 are contained in the Chat Corpus of NLTK (i.e. text5).*

Task 2.5 *What is the difference between the following two lines? Which one will give a larger value? Will this be the case for other texts?*[8]

```python
sorted(set([w.lower() for w in text1]))
sorted([w.lower() for w in set(text1)])
```

Task 2.6 *How many times does the word "lol" appear in text5? How much is this as a percentage of the total number of words in this text?*

2.3 Homework

Homework 2.1 *(5 points)*
An important problem in computational linguistics is morphological analysis. This consists of breaking down a word into its component pieces, for example losses *might be broken down as* loss + es. *In English, morphology is relatively simple and is mostly comprised of prefixes and suffixes. To get an idea of what suffixes are common in English (and thus could be morphemes), we can look at the frequencies of the last two characters of sufficiently long words. Write a function* `top_suffixes(words)` *that takes a sequence of words as an input and returns the 20 most frequent two-character suffixes of the input words. We define a two-character suffix as the last two characters of any word of length 5 or more, thus your function may simply ignore any word shorter than five characters. Your function should use the NLTK FreqDist class to count these suffixes. Hint: You may use the following code snippets:*

```python
FreqDist(???).most_common(20) #what argument expects FreqDist?
word[-2:] #result?
for word in words:
   if len(word) > 4:
```

The function would be used like this:

```python
words = ['caraa', 'bataa', 'dadbb', ...]
top_suffixes(words)
['aa', 'bb', ...]
```

As a sanity test, you may want test your function on the words in Blake poems. You can load the words of Blake as follows:

```python
blake_words = nltk.corpus.gutenberg.words('blake-poems.txt')
```

If you pass the words of Blake (or any suficiently large English document) as the argument to your function, you should see some common English suffixes in the output. [9]
`print(top_suffixes(blake_words))`

[8] See NLTK-book page chapter 1, exercise 19, page 36
[9] Source: University of Pennsylvania, Department of Computer and Information Science