LG 467 Computers in Linguistics

[1-2021] Topic 2: Text Normalization

Sakol Suethanapornkul

Previously...

- Tokenization is finding minimal units (**tokens**) from running text
  - Texts are a long sequence of characters
  - But breaking texts into tokens is easier said than done!
  - Consider the following example:

That U.S.A. poster-print costs $12.40

Example from: NLTK Chapter 3

Previously...

- Tokenization is a non-trivial problem:
  - In English, spaces are not exact
    - Contraction *I'm, we're, can't*, or *gonna*
    - Phrases such as *inasmuch as*, *insofar as*, and *in spite of*
    - Multiword expressions such as *New York* and *rock 'n' roll*
    - Words can have punctuation internally, e.g., *Ph.D.* and *AT&T*

Tokenization is tied up with **named entity recognition** (**NER**)

Previously...

- Tokenization is a non-trivial problem:
  - Many writing systems don't use spaces between words
  - Take Thai as an example:

    พี่ๆครับ ผมอยากรู้ว่าไอโฟน 13 จะวางขายที่ไทยเมื่อไหร่ครับ

  - People oftentimes code-switch

    พี่คะ Boss บอกหนูให้ concentrate on ประเด็นนี้ให้มากกว่านี้

Tokenization

A rudimentary tokenizer

You may remember a string method `.split()`:

Code 3.1

```
sent1 = '''You need to calm down
        you are being too loud'''

print(sent1.split())

# By default, split() uses spaces as a separator

sent2 = '''And I'm just like oh-oh,
        oh-oh, oh-oh, oh-oh, oh-oh'''

print(sent2.split('-'))
print(sent2.split(','))
```

Python lists

We will talk about other better options to tokenize texts. For now, let’s talk about lists!

- We've talked about basic data types like strings
- But what if we wanted to store multiple strings together?

Code 3.2

```
# Create a list
ls_1 = "man, can, ham".split(',')
ls_2 = ['man', 'can', 'ham', 'man']
ls_3 = ['man', 12, 4.0, True]
ls_4 = ['walk', ['sing', 'yell'], 'sleep']
```

Python list

A list in Python is an ordered group of items (or elements)

Code 3.3

```
# Like strings, we can index & slice items from lists
thai_pm = ['Samak', 'Somchai', 'Apisit', 'Yingluk',
‘Prayut']

thai_pm[0]
thai_pm[-1]
thai_pm[0:2]
thai_pm[-3:-1]
thai_pm[0:-2]
thai_pm[3:7]      #slicing items out of range
thai_pm[7]
```

Python list

You may have recalled from Codes 1.9 and 1.10 that while strings are**n't** mutable, lists are:

Code 3.4

```
# strings vs lists
a = 'ice'
b = a              #what is b?
b = 'cream'        #what is a now?

ls1 = ['a', 'b', 'c']
ls2 = ls1
ls2[1] = 'g'
ls1                #what do you think will happen?
ls3 = list(ls1) #create a copy
                   #changing ls3 won't change ls1
```

Python list

You may have recalled from Codes 1.9 and 1.10 that while strings are**n't** mutable, lists are:

```
# string vs. list methods
a.upper()        #doesn't change a
a.lower()
a.replace('i', 'sli')

thai_pm.reverse()
thai_pm.pop()
print(thai_pm)
```

Code 3.4

[Continued]

We'd like lists to constantly change: grow, shrink, etc.

List methods

There are several useful list methods:

Code 3.5

```
desserts = ['cakes', 'cakes', 'cookies', 'donuts']
desserts.append('ice creams')
desserts.remove('cakes')
desserts.remove('pastries')      #what will happen here?
desserts.pop()
desserts.pop(1)                  #work with index
desserts.clear()

desserts.count(‘cakes')
desserts.count('cake')           #exact match; 0 here
```

Functions

We can apply the following functions on lists:

Code 3.6

```
ex = ['a', 'd', 'd', 'b', 'e', 'c', 'a', 'b', 'd']

sorted(ex)
sorted(ex, reverse = True)

len(ex)
len(set(ex))  #change lists to sets & get unique values
```

`word_tokenize()` in NLTK

Let's get back to tokenizing texts! To do this, we need to get certain things from NLTK:

Code 3.7

```
from nltk.tokenize import word_tokenize
```

Go to this module
(~ Python file .py)

And then get this
function for me

Code 3.7
[Continued]

```
# if you get an error message about punkt
import nltk
nltk.download('punkt')
```

Importing 1+ functions

We'll need to import the following functions & classes:

```
# import 1+
from nltk.tokenize import word_tokenize, TweetTokenizer
from nltk.stem import snowball, WordNetLemmatizer

# change the function names
from nltk.tokenize import word_tokenize as wt
from nltk.tokenize import TweetTokenizer as tt
```

Code 3.7

[Continued]

1st attempt: Tokenization

Tokenizing strings can be as simple as:

Code 3.8

```
sent = '''If you’re happy and you know it, clap your
hands.'''
sent2 = '''I didn't wanna go into detail how these
things've been done!'''

word_tokenize(sent)
word_tokenize(sent2)
```

1st attempt: Tokenization

Tokenizing strings can be as simple as:

```
# Answer:
['If', 'you', '’', 're', 'happy', 'and', 'you', 'know',
'it', ',', 'clap', 'your', 'hands', '.']

# Answer:
['I', 'did', "n't", 'wan', 'na', 'go', 'into',
'detail', 'how', 'these', 'things', "'ve", 'been',
'done', '!']
```

Code 3.8

[Continued]

1st attempt: Tokenization

Tokenizing tweets isn’t that difficult either:

```
tweet = '''@MayorBowser @DOEE_DC @DCDPW @capitalweather
@washingtonpost Been #flooding like this for years no
help from the #DCgovernment #cafritz
#connecticutavenue'''

twt = TweetTokenizer()
twt.tokenize(tweet)
```

Code 3.8

[Continued]

Any difference between this and `word_tokenize()`?

Control flow: `for`

To iterate through a sequence & perform some operation, use `for`

```
for <variable> in <sequence>:
    <expression>
    <expression>
    ...
```

```
# for first for loop
for i in range(1,10):
    print(i)
print("Finished counting")
```

Code 3.9

Control flow: `for`

A more useful example involves printing multiple elements per line

```
index = 0
for number in [13, 15, 17, 19, 21, 23]:
    number = number + 5
    index += 1
    print("Index:", index, "Value:", number)

# or equivalently
num = [13, 15, 17, 19, 21, 23]
for index, number in zip(range(1,len(num)), num):
    number = number + 5
    print("Index:", index, "Value:", number)
```

Code 3.9

[Continued]

Lemmatization & Stemming

Stemming & Lemmatization

In an English text, we are likely to see different forms of a word

- organize, organizes, organized, organizing

We may also see derivationally related words with similar meanings

- democracy, democratic, democratization

Adapted from: Information retrieval companion website

Stemming & Lemmatization

Goal: reduce inflectional forms/derivationally related forms to base

- am, are, is → be
- car, cars, cars', car's → car

- the boy's cars are different colors
- the boy car be differ color

Adapted from: Information retrieval companion website

Stemming & Lemmatization

Stemming: crude heuristic process that chops off end of words, including removal of derivational affixes

Lemmatization: use of vocabulary as well as morphological analysis of words, with an aim of removing inflectional endings and returning a word back to a dictionary form (or **lemma**)

Adapted from: Information retrieval companion website

Stemming & Lemmatization

One of the most widely used stemming algorithms is the Porter stemmer:

> This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

> Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note

Adapted from: Jurafsky and Martin's Chapter 2

Stemming & Lemmatization

NLTK offers a few useful stemmers (Porter, Lancaster, Snowball):

Code 3.10

```
snow = snowball.SnowballStemmer('english')

# stemmers work with one "word" at a time
text = "construction workers built and constructed many
buildings in major cities”

text_tok = word_tokenize(text)

for t in text_tok:
    print(snow.stem(t))
```

Stemming & Lemmatization

Lemmatization often leads to a better result. WordNet lemmatizer only removes affixes if the resulting word is in its dictionary.

Code 3.11

```
wn = WordNetLemmatizer()
for t in text_tok:
    print(wn.lemmatize(t))
```

Compare the outputs of stemmer vs lemmatizer. Which is better? Which produced better "words"?

Our plan next week…

- Engine behind tokenization
  - Regular expressions (RE)
  - List comprehension `[w for w in token if len(w) < 12]`
- Readings:
  - J & M Chapter 2 (Sec 2.1)
  - L & C Chapter 4 (Sec 4.4; pp. 107–115)