DATA SCIENCE WITH PYTHON

CS 498 (SPECIAL TOPICS IN COMPUTER SCIENCE)

TEXT MINING & NLTK
PART2

Dr. Malak Abdullah

Computer Science Department

Jordan University of Science and Technology

REVIEW FROM LAST PART

REMOVING STOP WORDS

propython.py

```
1
2  import nltk
3
4  sentence = "At eight o'clock on Thursday morning Ali didn't feel very good. He bought a panadol with $5.28."
5  tokens=nltk.word_tokenize(sentence)
6  print (tokens)
7  print ("*"*30)
8
9  from nltk.corpus import stopwords
10 stopWords = set(stopwords.words('english'))
11 wordsFiltered = []
12
13 for w in tokens:
14     if w not in stopWords:
15         wordsFiltered.append(w)
16
17 print(wordsFiltered)
18
```

```
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Ali', 'did', "n't", 'feel', 'very', 'good', '.', 'He', 'bought', 'a',
'panadol', 'with', '$', '5.28', '.']
******************************
['At', 'eight', "o'clock", 'Thursday', 'morning', 'Ali', "n't", 'feel', 'good', '.', 'He', 'bought', 'panadol', '$', '5.28',
'.']
[Finished in 1.6s]
```

STEMMING

- A word stem is part of a word. It is sort of a normalization idea, but linguistic.
  For example, the stem of the word waiting is wait.

NLTK – stemming

Start by defining some words:

```
words = ["game","gaming","gamed","games"]
```

We import the module:

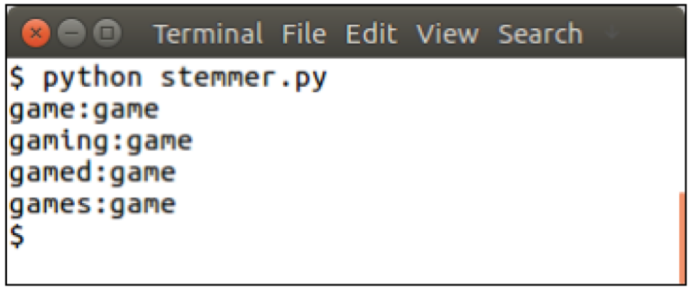
```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
```

And stem the words in the list using:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

words = ["game","gaming","gamed","games"]
ps = PorterStemmer()

for word in words:
    print(ps.stem(word))
```



nltk word stem example

POST TAG

Some simple things you can do with NLTK

Tokenize and tag some text:

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

IN: preposition
NN: noun, singular 'desk'
JJ: adjective 'big'

Find more on https://pythonprogramming.net/natural-language-toolkit-nltk-part-speech-tagging/

NLTK.BOOKS

propython.py

```
2
3   from nltk.book import *
4
5   print ("*"* 30)
6
7   print (text1)
8   print (len(text1))
9
10  print (text1.concordance("good"))
11
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
******************************
<Text: Moby Dick by Herman Melville 1851>
260819
Displaying 25 of 25 matches:
rbarous coasts . Not ignoring what is good , I am quick to perceive a horror , a
e Horn and the Pacific . Quitting the good city of old Manhatto , I duly arrived
y meat and potatoes , but dumplings ; good heavens ! dumplings for supper ! One
 two in a bed . In fact , you would a good deal rather not sleep with your own b
```

SORTED, SET

propython.py

```
3  from nltk.book import text3
4
5  print ("*"* 30)
6
7  x=sorted(set(text3))
8  print (x[1:30])
9
10 print (len(text3))
11
12 print (len(x))
```

```
******************************
["'", '(', ')', ',', ',)', '.', '.)', ':', ';', ';)', '?', '?)', 'A', 'Abel',
'Abelmizraim', 'Abidah', 'Abide', 'Abimael', 'Abimelech', 'Abr', 'Abrah', 'Abraham',
'Abram', 'Accad', 'Achbor', 'Adah', 'Adam', 'Adbeel', 'Admah']
44764
2789
[Finished in 9.2s]
```

MORE (1)

propython.py

propython.py

```
2
3   from nltk.book import *
4
5   print ("*"* 30)
6
7   x=text3.count('the')
8   print (x, len(text3))
9   print (x/len(text3))
10  print (text3.index('the')) #The first occur
11
12
13
```

```
******************************
2411 44764
0.05386024483960325
1
[Finished in 8.1s]
```

MORE (2)

propython.py

```
3  from nltk.book import *
4
5  x=FreqDist(text1)
6  print (x)
7  print (x.most_common(20))
8  print ("*"*30)
9  print (x['the'])
10
11
12
```

```
<FreqDist with 19317 samples and 260819 outcomes>
[(',', 18713), ('the', 13721), ('.', 6862), ('of', 6536), ('and', 6024), ('a',
4569), ('to', 4542), (';', 4072), ('in', 3916), ('that', 2982), ("'", 2684), ('-',
2552), ('his', 2459), ('it', 2209), ('I', 2124), ('s', 1739), ('is', 1695), ('he',
1661), ('with', 1659), ('was', 1632)]
******************************
13721
[Finished in 9.1s]
```

```
>>> V = set(text1)
>>> long_words = [w for w in V if len(w) > 15]
>>> sorted(long_words)
['CIRCUMNAVIGATION', 'Physiognomically', 'apprehensiveness', 'cannibalistically',
'characteristically', 'circumnavigating', 'circumnavigation', 'circumnavigations',
'comprehensiveness', 'hermaphroditical', 'indiscriminately', 'indispensableness',
'irresistibleness', 'physiognomically', 'preternaturalness', 'responsibilities',
'simultaneousness', 'subterraneousness', 'supernaturalness', 'superstitiousness',
'uncomfortableness', 'uncompromisedness', 'undiscriminating', 'uninterpenetratingly']
>>>
```

COLLOCATIONS AND BIGRAMS

- **Collocations:** a sequence of words that occur together unusually often
- **Bigrams**: a list of word pairs

```
>>> list(bigrams(['more', 'is', 'said', 'than', 'done']))
[('more', 'is'), ('is', 'said'), ('said', 'than'), ('than', 'done')]
```

```
>>> text4.collocations()
United States; fellow citizens; four years; years ago; Federal
Government; General Government; American people; Vice President; Old
World; Almighty God; Fellow citizens; Chief Magistrate; Chief Justice;
God bless; every citizen; Indian tribes; public debt; one another;
foreign nations; political parties
>>> text8.collocations()
would like; medium build; social drinker; quiet nights; non smoker;
long term; age open; Would like; easy going; financially secure; fun
times; similar interests; Age open; weekends away; poss rship; well
presented; never married; single mum; permanent relationship; slim
build
>>>
```

WORDNET

- WordNet is a semantically-oriented dictionary of English, similar to a traditional thesaurus but with a richer structure. NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets. We'll begin by looking at synonyms and how they are accessed in WordNet.

propython.py

```
2
3  from nltk.corpus import wordnet as wn
4
5  print (wn.synsets('land'))
6  print ("*"*30)
7  print (wn.synset('land.n.04').lemma_names())
8  print ("*"*30)
```

```
[Synset('land.n.01'), Synset('land.n.02'), Synset('domain.n.02'),
Synset('land.n.04'), Synset('country.n.02'), Synset('kingdom.n.01'),
Synset('estate.n.02'), Synset('nation.n.02'), Synset('state.n.04'),
Synset('land.n.10'), Synset('farming.n.02'), Synset('land.v.01'),
Synset('land.v.02'), Synset('bring.v.05'), Synset('land.v.04'), Synset('land.v.05'),
Synset('land.v.06'), Synset('down.v.04')]
******************************
['land', 'dry_land', 'earth', 'ground', 'solid_ground', 'terra_firma']
******************************
[Finished in 4.6s]
```

DETECTING WORD PATTERNS

- A regular expression (or RE) specifies a set of strings that matches it;
- the functions in this module let you check if a particular string matches a given regular expression
- Can contain both special and ordinary characters

- a, X, 9, < -- ordinary characters just match themselves exactly. The meta-characters which do not match themselves because they have special meanings are: . ^ $ * + ? { [ ] \ | ( ) (details below)
- . (a period) -- matches any single character except newline '\n'
- \w -- (lowercase w) matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_]. Note that although "word" is the mnemonic for this, it only matches a single word char, not a whole word. \W (upper case W) matches any non-word character.
- \b -- boundary between word and non-word
- \s -- (lowercase s) matches a single whitespace character -- space, newline, return, tab, form [\n\r\t\f]. \S (upper case S) matches any non-whitespace character.
- \t, \n, \r -- tab, newline, return
- \d -- decimal digit [0-9] (some older regex utilities do not support but \d, but they all support \w and \s)
- ^ = start, $ = end -- match the start or end of the string
- \ -- inhibit the "specialness" of a character. So, for example, use \. to match a period or \\ to match a slash. If you are unsure if a character has special meaning, such as '@', you can put a slash in front of it, \@, to make sure it is treated just as a character.

`\d`
Matches any decimal digit; this is equivalent to the class `[0-9]`.

`\D`
Matches any non-digit character; this is equivalent to the class `[^0-9]`.

`\s`
Matches any whitespace character; this is equivalent to the class `[ \t\n\r\f\v]`.

`\S`
Matches any non-whitespace character; this is equivalent to the class `[^ \t\n\r\f\v]`.

`\w`
Matches any alphanumeric character; this is equivalent to the class `[a-zA-Z0-9_]`.

`\W`
Matches any non-alphanumeric character; this is equivalent to the class `[^a-zA-Z0-9_]`

[\s,.] is a character class that will match any whitespace character, or ',' or '.'

RE.COMPILE(R ” ”)

- The purpose of the compile method is to compile the regex pattern which will be used for matching later.

propython.py

```
2
3  import nltk
4  import re
5
6  pattern = re.compile(r"\w+\s\w+")
7  # Let's feed in some strings to match
8  string = "regex is awesome!"
9  # Then call a matching method to match our pattern
10 result = pattern.match(string)
11 print (result.group())
12
```

```
regex is
[Finished in 1.3s]
```

Here are some other regex quantifiers and how to use them.

Quantifier	Description	Example	Sample match
+	one or more	\w+	ABCDEF097
{2}	exactly 2 times	\d{2}	01
{1,}	one or more times	\w{1,}	smiling
{2,4}	2, 3 or 4 times	\w{2,4}	1234
*	0 or more times	A*B	AAAAB
?	once or none(lazy)	\d+?	1 in 12345

^ AND $

- ^ and $ are **boundaries** or anchors. ^ marks the start, while $ marks the end of a regular expression

- If it is used in square brackets [^ …] it means not. For example, [^\s$] or just [^\s] will tell regex to match anything that is **not a whitespace character.**

RE.SEARCH, RE.MATCH(), RE.SPLIT(), RE.FINDALL()

- The match method checks for a match only at the beginning of the string, while a re.search() checks for a match anywhere in the string.

- The re.split() splits a string into a list delimited by the passed pattern
- We can use split to read each line and split them into an array as such:
  - results = re.split(r"\n+", text)

- Unlike search which returns a match object, findall returns a list of matches.

```
import re salaries = "120000 140000 10000 1000 200"
result_list = re.findall(r"\d{5,6}", salaries)
print result_list # prints out: ['120000', '140000', '10000']
```

RE.SUB()

- To do some string replacement
- The syntax for re.sub() is re.sub(pattern,repl,string).
- That will replace the matches in string with repl. In this example.
- I will replace all occurrences of the re pattern ("cool") in string (text) with repl ("good").

```
import re
text = "Python for beginner is a very cool website"
pattern = re.sub("cool", "good", text)
print text2
```

```
import re
pattern = re.compile(r"[0-9]+")
result = pattern.sub("__", "there is only 1 thing 2 do")
print result       # there is only __ thing __ do
```

THIS WEBSITE CAN HELP YOU WITH RE

- https://pythex.org

```
str = 'purple alice-b@google.com monkey dishwasher'
 match = re.search(r'\w+@\w+', str)
 if match:
  print match.group()  ## 'b@google'
```

Square brackets can be used to indicate a set of chars (. Means .)

```
match = re.search(r'[\w.-]+@[\w.-]+', str)
 if match:
  print match.group()  ## 'alice-b@google.com'
```

CHEAT SHEET

- https://www.dataquest.io/blog/regex-cheatsheet/

```
re.sub('https?://[A-Za-z0-9./]+'  ,' '  ,  tweet1)
```

```
re.sub('@[A-Za-z0-9]+'  ,' '  ,  tweet1)
```

What do you think these will do?