A note on NLTK in Python - Akash Gupta

This contains various code chunks, functions and programs that implement python's **NLTK** functionality to carry out some fundamental tasks related to **Text Analytics** like :-

1. Opening and reading text files in Python.
2. Opening, reading and parsing HTML files in Python (extracting article text out of pages).
3. Perfoming Tokenization on text.
4. Using **Regular Expressions** to extract certain specific words from a text.

First we import the necessary libaries.

```
import nltk
import re
import pprint
```

Reading the text from a webpage as a **.txt** file and opening it in Python

```
from urllib.request import urlopen
url = "http://www.gutenberg.org/files/58700/58700-0.txt"
raw = urlopen(url).read().decode('utf-8')
type(raw)
len(raw)
raw[:50]
```

```
'\ufeffThe Project Gutenberg EBook of Crimes and Punishm'
```

Tokenizing the Text

```
tokens = nltk.word_tokenize(raw)
type(tokens)
len(tokens)
tokens[:10]
```

```
['\ufeffThe',
 'Project',
 'Gutenberg',
 'EBook',
 'of',
 'Crimes',
 'and',
 'Punishments',
 ',',
 'by']
```

Here **collocations** are found - these are most commonly occurring bigrams in the text.

```
text = nltk.Text(tokens)
text[1235:1257]
text.collocations()
raw.find("BECCARIA'S LIFE AND CHARACTER.")
raw.rfind("End of Project Gutenberg's Crimes and Punishments")

raw = raw[3819:361434]
raw[:70]
```

```
Project Gutenberg-tm; Project Gutenberg; capital punishment; United
States; penal servitude; Literary Archive; Gutenberg-tm electronic;
electronic works; Gutenberg Literary; Archive Foundation; per cent;
Dei Delitti; Pietro Verri; penal laws; criminal law; set forth; Lord
```

```
Ellenborough; electronic work; Gutenberg-tm License; one another
```

```
'BECCARIA'S LIFE AND CHARACTER.\r\n\r\n    State of Lombardy under Count Fi'
```

This code chunk involves getting an **HTML** file and parsing the tags from it to extract only the article body. A small output of the same is also provided.

```
from bs4 import BeautifulSoup

url = "https://www.economist.com/briefing/2020/04/30/the-90-economy-that-lockdowns-will-leave-behind"
html = urlopen(url).read()
soup = BeautifulSoup(html)

for script in soup(['script','style','[document]','head','title']):
    script.extract()

text = soup.get_text()
text[:1400]
text.find("IN THE 1970s Mori")
text.rfind("the economy that re-emerges will be fundamentally different.")

text = text[1237:1500]
text
```

```
'ncanny valley" are close enough to lifelike for their shortfalls and divergences from the familiar to l
```

Here we use **Regular Expressions** to detect word patterns. We try to extract words that are ending with 'ed'. Then extract words matching a range of characters.

```
import re
wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]

[w for w in wordlist if re.search('ed$',w)]

# the . wildcard symbol matches any single character

[w for w in wordlist if re.search('^..t..t..$',w)]

# matching in a range of characters

[w for w in wordlist if re.search('^[abg][hgi][eut][asg]',w)]
```

```
['agua',
 'aguacate',
 'aguavina',
 'agush',
 'agust',
 'ahead',
 'aheap',
 'ahuatle',
 'ahush',
 'bitable',
 'bitangent',
```

```
 'bitangential',
 'bitanhol',
 'bitartrate',
 'bitstock',
 'bitstone',
 'gieseckite',
 'gitaligenin',
 'gitalin',
 'giustina']
```

In this code chunk we use **Regular Expressions** to extract special characters. Each list in the chunk provides a different extraction functionality.

```
wsf = sorted(set(nltk.corpus.treebank.words()))
[w for w in wsf if re.search('^[0-9]+\.[0-9]+$', w)]
[w for w in wsf if re.search('[A-Z]+\$$',w)]
[w for w in wsf if re.search('[0-9]{4}$',w)]
[w for w in wsf if re.search('^[0-9]+-[a-z]{3,5}$',w)]
[w for w in wsf if re.search('^[a-z]{5,}-[a-z]{2,3}-[a-z]{,6}$',w)]
[w for w in wsf if re.search(r'\$',w)]

# prefixing special characters by r in reg exprs gives us a raw string rather than special interpretati
```

```
['$', 'C$', 'US$']
```

In the next few code chunks, more functionality on **Regular Expressions** is provided.

```
# more on reg exprs

word = 'supercalifragilisticexpialidocious'
re.findall(r'[aeiou]',word)
len(re.findall(r'[aeiou]',word))

wsj = sorted(set(nltk.corpus.treebank.words()))
fd = nltk.FreqDist(vs for word in wsj for vs in re.findall(r'[aeiou]{2,}',word))
fd.items()
```

```
dict_items([('ea', 476), ('oi', 65), ('ou', 329), ('io', 549), ('ee', 217), ('ie', 331), ('ui', 95), ('
```

```
chat_words = sorted(set(nltk.corpus.nps_chat.words()))
[w for w in chat_words if re.search('[^aeiouAEIOU]+$',w)]

wsj = sorted(set(nltk.corpus.treebank.words()))
[w for w in wsj if re.search('(ed|ing)$',w)]

wor = sorted(set(nltk.corpus.words.words()))
[w for w in wor if re.search('ming?$',w)]

word1 = 'superinnoucilus'
re.findall(r'[aeiou]',word1)
```

```
['u', 'e', 'i', 'o', 'u', 'i', 'u']
```

```
regexp = r'^[AEIOUaeiou]+|[AEIOUaeiou]+$|[AEIOUaeiou]'
def compress(word):
    pieces = re.findall(regexp,word)
    return ''.join(pieces)
```

```
wordlists = ['super','cup','track','yellow','fuits','GREAT','GLOAT","lbg","trunk","cipppppppp","argon",
            "obese","italy"]
[w for w in wordlists if re.search('^[^tr]+$',w)]
```

```
['cup', 'yellow', 'GREAT', 'GLOAT', 'lbg', 'cipppppppp', 'obese']
```

```
[w for w in wordlists if re.search('^[^tr]+$',w)]
```

```
['cup', 'yellow', 'GREAT', 'GLOAT', 'lbg']
```

```
[w for w in wordlists if re.findall('^[AEIOUaeiou]+|[AEIOUaeiou]+$|[^AEIOUaeiou]',w)]
```

```
['super',
 'cup',
 'track',
 'yellow',
 'fuits',
 'GREAT',
 'GLOAT',
 'lbg',
 'trunk',
 'cipppppppp',
 'argon',
 'obese',
 'italy']
```

```
[w for w in wordlists if re.findall('^[AEIOUaeiou]+|[AEIOUaeiou]+$',w)]

re.findall(r'(pi)$', 'mississippi')
```

```
['pi']
```

```
regexp = r'^[AEIOUaeiou]+|[AEIOUaeiou]+$|[^AEIOUaeiou]'
def compress(word):
    pieces = re.findall(regexp,word)
    return ''.join(pieces)

english_udhr = nltk.corpus.udhr.words('English-Latin1')
print(nltk.tokenwrap(compress(w) for w in english_udhr[:75]))

# we want words that might start with a vowel, end with a vowel. middle vowels are not there
```

```
Unvrsl Dclrtn of Hmn Rghts Prmble Whrs rcgntn of the inhrnt dgnty and
of the eql and inlnble rghts of all mmbrs of the hmn fmly is the fndtn
of frdm , jstce and pce in the wrld , Whrs dsrgrd and cntmpt fr hmn
rghts hve rsltd in brbrs acts whch hve outrgd the cnscnce of mnknd ,
and the advnt of a wrld in whch hmn bngs shll enjy frdm of spch and
```

```
print(nltk.tokenwrap(compress(w) for w in wordlists))
```

```
spr cp trck yllw fts GRT GLT lbg trnk cpppppppp argn obse itly
```

```
# plotting conditional freqdist with consonant vowel pairs

rotokas_words = nltk.corpus.toolbox.words('rotokas.dic')
cvs = [cv for w in rotokas_words for cv in re.findall(r'[ptksvr][aeiou]',w)]
cfd = nltk.ConditionalFreqDist(cvs)
cfd.tabulate()
```

```
    a   e   i   o   u
k 418 148  94 420 173
p  83  31 105  34  51
r 187  63  84  89  79
s   0   0 100   2   1
t  47   8   0 148  37
v  93  27 105  48  49
```

```
# now we want to look up index to find all words having a certain consonant vowel pair

cv_word_pairs = [(cv,w) for w in rotokas_words for cv in re.findall(r'[ptksvr][aeiou]',w)]
cv_index = nltk.Index(cv_word_pairs)
cv_index['vo']
```

```
['kaakaavo',
 'kakavoro',
 'kaukovo',
 'kavo',
 'kavokavo',
 'kavokavo',
 'kavokavoa',
 'kavokavoa',
 'kavokavoto',
 'kavokavoto',
 'kavora',
 'kavorato',
 'kavori',
 'kavori',
 'kavorou',
 'kavovoa',
 'kavovoa',
 'kavovovira',
 'kavovovira',
 'kavuvo',
 'kekevoto',
 'kekevotovira',
 'keravo',
 'kevoisi',
 'kevoisivira',
 'koovoto',
 'koovotova',
 'korovo',
 'kovo',
 'kovo',
 'kovo',
 'kovoa',
 'kovokovo',
 'kovokovo',
 'kovokovo',
 'kovokovo',
 'kovokovo',
 'kovokovo',
 'kovokovoa',
 'kovokovoa',
 'kovopaa',
```

```
 'kovopato',
 'kovopie',
 'kovoruko',
 'kovoto',
 'kovovo',
 'kovovo',
 'kuvoro']
```

```
# pulling out stems or unrequired suffixes from words to keep web search wide

def stem(word):
    for suffix in ['ing','ly','ed','ious','ies','ive','es','s','ment']:
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word

# use reg expr in the form of disjunctions to remove suffixes. all suffixes enclosed in parenthesis
# to limit the scope of search among suffixes

re.findall(r'^.*(?:ing|ly|ed|ious|ies|ive|es|s|ment)$','processing')
re.findall(r'^(.*)(ing|ly|ed|ious|ies|ive|es|s|ment)$','processing')
re.findall(r'^(.*)(ing|ly|ed|ious|ies|ive|es|s|ment)$','processes') # greedy * - gets max input length
re.findall(r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)$','processes') # non greedy *?
re.findall(r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$','language')
```

```
[('language', '')]
```

```
# stemming an entire text

def stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ious|ive|es|s|ment)?$'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

raw = """DENNIS: listen, strange women lying in ponds distributing swords is no basis
for a system of government. Supreme executive power derives from a mandate from the masses,
not from some farcical aquatic ceremony."""

tok = nltk.word_tokenize(raw)

tokens = nltk.word_tokenize(raw)
txtt = nltk.Text(tokens)

[stem(t) for t in tokens]

txtt.findall(r'<\w*><\w*s>')
```

```
in ponds; distributing swords; no basis; power derives; the masses
```

```
# searching tokenized text using <> functionality of reg exprs

from nltk.corpus import gutenberg, nps_chat
moby = nltk.Text(gutenberg.words('melville-moby_dick.txt'))
moby.findall(r'<a>(<.*>)<man>')

chat = nltk.Text(nps_chat.words())
```

```
chat.findall(r'<.*><.*><bro>')
chat.findall(r'<l.*>{3,}')
```

```
monied; nervous; dangerous; white; white; white; pious; queer; good;
mature; white; Cape; great; wise; wise; butterless; white; fiendish;
pale; furious; better; certain; complete; dismasted; younger; brave;
brave; brave; brave
you rule bro; telling you bro; u twizted bro
lol lol lol; lmao lol lol; lol lol lol; la la la la la; la la la; la
la la; lovely lol lol love; lol lol lol.; la la la; la la la
```

```
from nltk.corpus import brown

hobbies_learned = nltk.Text(brown.words(categories = ['hobbies','learned']))
hobbies_learned.findall(r'<\w*><and><other><\w*s>')
```

```
speed and other activities; water and other liquids; tomb and other
landmarks; Statues and other monuments; pearls and other jewels;
charts and other items; roads and other features; figures and other
objects; military and other areas; demands and other factors;
abstracts and other compilations; iron and other metals
```

```
# porter and lancaster stemmers

raw = """DENNIS: listen, strange women lying in ponds distributing swords is no basis
for a system of government. Supreme executive power derives from a mandate from the masses,
not from some farcical aquatic ceremony."""

tokens = nltk.word_tokenize(raw)
porter = nltk.PorterStemmer()
lancaster = nltk.LancasterStemmer()

[porter.stem(t) for t in tokens]
[lancaster.stem(t) for t in tokens]

list(enumerate(tokens))

ind = nltk.Index((porter.stem(w),i) for (i, w) in list(enumerate(tokens)))
ind['denni']

tx = ' '.join(tokens[-20:25])
tx
'%*s' % (50, tx[-50:])
```

```
' government . Supreme executive power derives from'
```

```
# indexing a text using a stemmer

class IndexedText(object):

    def __init__(self, stemmer, text):
        self._text = text
        self._stemmer = stemmer
        self._index = nltk.Index((self._stem(word),i) for (i, word) in enumerate(text))

    def concordance(self, word, width = 40):
```

```
        key = self._stem(word)
        pr
        wc = int(width/4)
        for i in self._index[key]:
            lcontext = ' '.join(self._text[i-wc:i])
            rcontext = ' '.join(self._text[i:i+wc])
            ldisplay = '%*s' % (width, lcontext[-width:])
            rdisplay = '%-*s' % (width, rcontext[:width])
            print(ldisplay, rdisplay)

    def _stem(self, word):
        return self._stemmer.stem(word).lower()


porter = nltk.PorterStemmer()
grail = nltk.corpus.webtext.words('grail.txt')
text = IndexedText(porter, grail)
text.concordance('lie')
```

```
lie
r king ! DENNIS : Listen , strange women lying in ponds distributing swords is no
 beat a very brave retreat . ROBIN : All lies ! MINSTREL : [ singing ] Bravest of
       Nay . Nay . Come . Come . You may lie here . Oh , but you are wounded !
doctors immediately ! No , no , please ! Lie down . [ clap clap ] PIGLET : Well
ere is much danger , for beyond the cave lies the Gorge of Eternal Peril , which
   you . Oh ... TIM : To the north there lies a cave -- the cave of Caerbannog --
h it and lived ! Bones of full fifty men lie strewn about its lair . So , brave k
not stop our fight ' til each one of you lies dead , and the Holy Grail returns t
```

```
# wordnet lemmatization removes affixes only if the resultant word is in dictionary

wnl = nltk.WordNetLemmatizer()
[wnl.lemmatize(t) for t in tokens]
```

```
['DENNIS',
 ':',
 'listen',
 ',',
 'strange',
 'woman',
 'lying',
 'in',
 'pond',
 'distributing',
 'sword',
 'is',
 'no',
 'basis',
 'for',
 'a',
 'system',
 'of',
 'government',
 '.',
 'Supreme',
```

```
 'executive',
 'power',
 'derives',
 'from',
 'a',
 'mandate',
 'from',
 'the',
 'mass',
 ',',
 'not',
 'from',
 'some',
 'farcical',
 'aquatic',
 'ceremony',
 '.']
```

```
# tokenizing text using reg  : cut a string into identifiable linguistic units

raw = """'When I'M a Duchess,' she said to herself, (not in a very hopeful tone
though), 'I won't have any pepper in my kitchen AT ALL.Soup does very
well without--Maybe it's always pepper that makes people hot-tempered,'..."""

re.split(r' ',raw)
re.split(r'[ \t\n]+',raw)
re.split(r'\s+',raw)

re.split(r'\W+',raw)
re.findall(r'\w+',raw)

# disjunctive reg expr splitting

re.findall(r'\w+|\S\w*',raw)    #\S = non white space character. \w = words

re.findall(r"\w+(?:[-']\w+)*|'|[-.(]+|\S\w*",raw)
```

```
["'",
 'When',
 "I'M",
 'a',
 'Duchess',
 ',',
 "'",
 'she',
 'said',
 'to',
 'herself',
 ',',
 '(',
 'not',
 'in',
 'a',
 'very',
 'hopeful',
```

```
 'tone',
 'though',
 ')',
 ',',
 "'",
 'I',
 "won't",
 'have',
 'any',
 'pepper',
 'in',
 'my',
 'kitchen',
 'AT',
 'ALL',
 '.',
 'Soup',
 'does',
 'very',
 'well',
 'without',
 '--',
 'Maybe',
 "it's",
 'always',
 'pepper',
 'that',
 'makes',
 'people',
 'hot-tempered',
 ',',
 "'",
 '...']
```

```
# nltk tokenizer

from nltk.tokenize.regexp import RegexpTokenizer

text = 'That U.S.A poster-print costs $12.40...'

pattern = r"""(?x)
    ([A-Z]\.)+
    | \w+(-\w+)*
    | \$?\d+(\.\d+)?%?
    | \.\.\.
    | [][.,;"'?():-_`]
    """

tokeniser = RegexpTokenizer(pattern)
tokeniser.tokenize(text)
```

```
[('', '', ''),
 ('S.', '', ''),
 ('', '', ''),
 ('', '-print', ''),
```

```
 ('', '', ''),
 ('', '', '.40'),
 ('', '', '')]
```

```
# tokenizing text into sentences
import pprint

sent_tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
sents = sent_tokenizer.tokenize(text)
pprint.pprint(sents[171:181])
```

```
['In the wild events which were to follow this girl had no\n'
 'part at all; he never saw her again until all his tale was over.',
 'And yet, in some indescribable way, she kept recurring like a\n'
 'motive in music through all his mad adventures afterwards, and the\n'
 'glory of her strange hair ran like a red thread through those dark\n'
 'and ill-drawn tapestries of the night.',
 'For what followed was so\nimprobable, that it might well have been a dream.',
 'When Syme went out into the starlit street, he found it for the\n'
 'moment empty.',
 'Then he realised (in some odd way) that the silence\n'
 'was rather a living silence than a dead one.',
 'Directly outside the\n'
 'door stood a street lamp, whose gleam gilded the leaves of the tree\n'
 'that bent out over the fence behind him.',
 'About a foot from the\n'
 'lamp-post stood a figure almost as rigid and motionless as the\n'
 'lamp-post itself.',
 'The tall hat and long frock coat were black; the\n'
 'face, in an abrupt shadow, was almost as dark.',
 'Only a fringe of\n'
 'fiery hair against the light, and also something aggressive in the\n'
 'attitude, proclaimed that it was the poet Gregory.',
 'He had something\n'
 'of the look of a masked bravo waiting sword in hand for his foe.']
```

```
# recognizing words from sents - have a boolean var that signals end of word

text = 'doyouseethekittyseethedoggydoyoulikethekittylikethedoggy'
seg1 = '0000000000000001000000000010000000000000001000000000'
seg2 = '0100100100100001001001000010100100010010000100010010000'

def segment(text,seg):
    words = []
    last = 0
    for i in range(len(seg)):
        if seg[i] == '1':
            words.append(text[last:i+1])
            last = i+1
    words.append(text[last:])
    return words

segment(text,seg1)
segment(text,seg2)
```

```
['do',
 'you',
 'see',
 'the',
 'kitty',
 'see',
 'the',
 'doggy',
 'do',
 'you',
 'like',
 'the',
 'kitty',
 'like',
 'the',
 'doggy']
```

```
# string formattings, using join and other things

silly = ['We','called','him','Tortoise','because','he','taught','us','.']
' '.join(silly)

fdist = nltk.FreqDist(['dog','cat','dog','cat','dog','snake','dog','cat'])
for word in fdist:
    print(word + ":",fdist[word])
```

```
dog: 4
cat: 3
snake: 1
```

```
# using string formatting expressions

for word in fdist:
    print('%s->%d;'%(word,fdist[word]))
```

```
dog->4;
cat->3;
snake->1;
```

```
# performing tabulation using string formatting

def tabulate(cfdist, words, categories):
    print('%-16s'% 'Category',)
    for word in words:
        print('%6s'% word,)
    for category in categories:
        print('%-16s'%category,)
        for word in words:
            print('%6d'%cfdist[category][word])

from nltk.corpus import brown
cfd = nltk.ConditionalFreqDist((genre,word) for genre in brown.categories()
                               for word in brown.words(categories=genre))
genres = ['news','religion','hobbies','science_fiction','romance','humor']
modals = ['can','could','may','might','must','will']
tabulate(cfd,modals,genres)
```

```
Category
   can
 could
   may
 might
  must
  will
news
    93
    86
    66
    38
    50
   389
religion
    82
    59
    78
    12
    54
    71
hobbies
   268
    58
   131
    22
    83
   264
science_fiction
    16
    49
     4
    12
     8
    16
romance
    74
   193
    11
    51
    45
    43
humor
    16
    30
     8
     8
     9
    13
```