Natural Language Tool Kit (NLTK)

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', '.']
```

Sentence segmentation

- **Divide text into sentences**
- **>>> text = “this’s a sent tokenize test. this is sent two. is this sent three? sent 4 is cool! Now it’s your turn.”**

```
>>> from nltk.tokenize import sent_tokenize
>>> sent_tokenize_list = sent_tokenize(text)
>>> len(sent_tokenize_list)
      5
>>> sent_tokenize_list
      [“this’s a sent tokenize test.”, ‘this is sent two.’, ‘is this sent three?’,
      ‘sent 4 is cool!’, “Now it’s your turn.”]
```

Sentence Tokenizer

```
>>> nltk.sent_tokenize ("The Probability tutorial discusses creation of histograms on more complex features, like the length of words following words ending in vowels. The class nltk.draw.plot.Plot is useful for visualization of histograms. Of course, you can equally analyze frequencies of higher-level grammatical features, or even of data sets unrelated to NLTK. Conditional frequencies are perhaps more interesting than plain histograms. A conditional frequency is a kind of two-dimensional histogram -- it gives you one histogram per initial condition, or context. For example, the tutorial suggests the question of what the distribution of word lengths is for each first letter. ")
```

```
['The Probability tutorial discusses creation of histograms on more complex features, like the length of words following words ending in vowels.',
'The class nltk.draw.plot. Plot is useful for visualization of histograms.',
 'Of course, you can equally analyze frequencies of higher-level grammatical features, or even of data sets unrelated to NLTK. Conditional frequencies are perhaps more interesting than plain histograms.',
 'A conditional frequency is a kind of two-dimensional histogram -- it gives you one histogram per initial condition, or context.',
'For example, the tutorial suggests the question of what the distribution of word lengths is for each first letter.']
```

PunktSentenceTokenizer

```
>>> import nltk
>>> from nltk.corpus import state_union
>>> from nltk.tokenize import PunktSentenceTokenizer
>>> train_text = state_union.raw("2005-GWBush.txt")
>>> sample_text = state_union.raw("2006-GWBush.txt")
>>> custom_sent_tokenizer = PunktSentenceTokenizer(train_text) #A
>>> tokenized = custom_sent_tokenizer.tokenize(sample_text)    #B
```

Stemmers

Stemmers remove morphological affixes from words, leaving only the word stem.

```
>>> from nltk.stem import *
```

How to use Stemmer in NLTK

```
>>> from nltk.stem.porter import PorterStemmer
>>> porter_stemmer = PorterStemmer()
>>> porter_stemmer.stem('maximum')
        'maximum‘
>>> porter_stemmer.stem(‘crying’)
      ‘cri’
```

Porter stemmer

```
>>> from nltk.stem.porter import *
```

Create a new Porter stemmer.

```
>>> stemmer = PorterStemmer()
```

Test the stemmer on various pluralized words.

```
>>> plurals = ['caresses', 'flies', 'dies', 'mules', 'denied', 'died', 'agreed',
'owned', 'humbled', 'sized', 'meeting', 'stating', 'siezing', 'itemization',
'sensational', 'traditional', 'reference', 'colonizer', 'plotted']
>>> singles = [stemmer. stem(plural) for plural in plurals]
>>> print(singles)
['caress', 'fli', 'die', 'mule', 'deni', 'die',
'agre', 'own', 'humbl', 'size', 'meet', 'state', 'siez',
'item', 'sensat', 'tradit', 'refer', 'colon', 'plot']
```

Lancaster stemmer

```
>>> from nltk.stem.lancaster import LancasterStemmer
>>> lancaster_stemmer = LancasterStemmer()
>>> lancaster_stemmer.stem('maximum')
    'maxim‘
>>> lancaster_stemmer.stem(‘crying’)
     ‘cry’
```

```
>>> lancaster_stemmer.stem('happiness’)
    'happy'
>>> lancaster_stemmer.stem('replacement’)
    'replac'
>>> porter_stemmer.stem('happiness’)
    'happi'
>>> porter_stemmer.stem('replacement’)
    'replac'
```

Snowball stemmer

```
>>> from nltk.stem.snowball import SnowballStemmer
```

Create a new instance of a language specific subclass.

```
>>> stemmer = SnowballStemmer("english")
```

Stem a word.

```
>>> print(stemmer.stem("running"))
    run
```

Decide not to stem stop words.

```
>>> stemmer2 = SnowballStemmer("english", ignore_stopwords=True)
>>> print(stemmer.stem("being"))
    be
>>> print(stemmer2.stem("being"))
    being
```

The 'english' stemmer is better than the original 'porter' stemmer.

```
>>> print(SnowballStemmer("english").stem("generously"))
generous
>>> print(SnowballStemmer("porter").stem("generously"))
gener
```

WordNet lemmatizer

```
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
>>> print(lemmatizer.lemmatize("cats"))
>>> print(lemmatizer.lemmatize("better", pos="a"))
>>> print(lemmatizer.lemmatize("best", pos="a"))
>>> print(lemmatizer.lemmatize("run"))
>>> print(lemmatizer.lemmatize("run",'v'))
```

Part of Speech Tagging

- NLTK’s pos_tag method---based on MaxEnt classifier
- Stanford tagger

Part of Speech Tagging

```
>>> sent="John played cricket in the ground"
>>> tokens=word_tokenize(sent)
>>> tagged=nltk.pos_tag(tokens)
>>> tagged
[('John', 'NNP'), ('played', 'VBD'), ('cricket', 'NN'), ('in', 'IN'),
 ('the', 'DT'), ('ground', 'NN')]
```

Stanford tagger

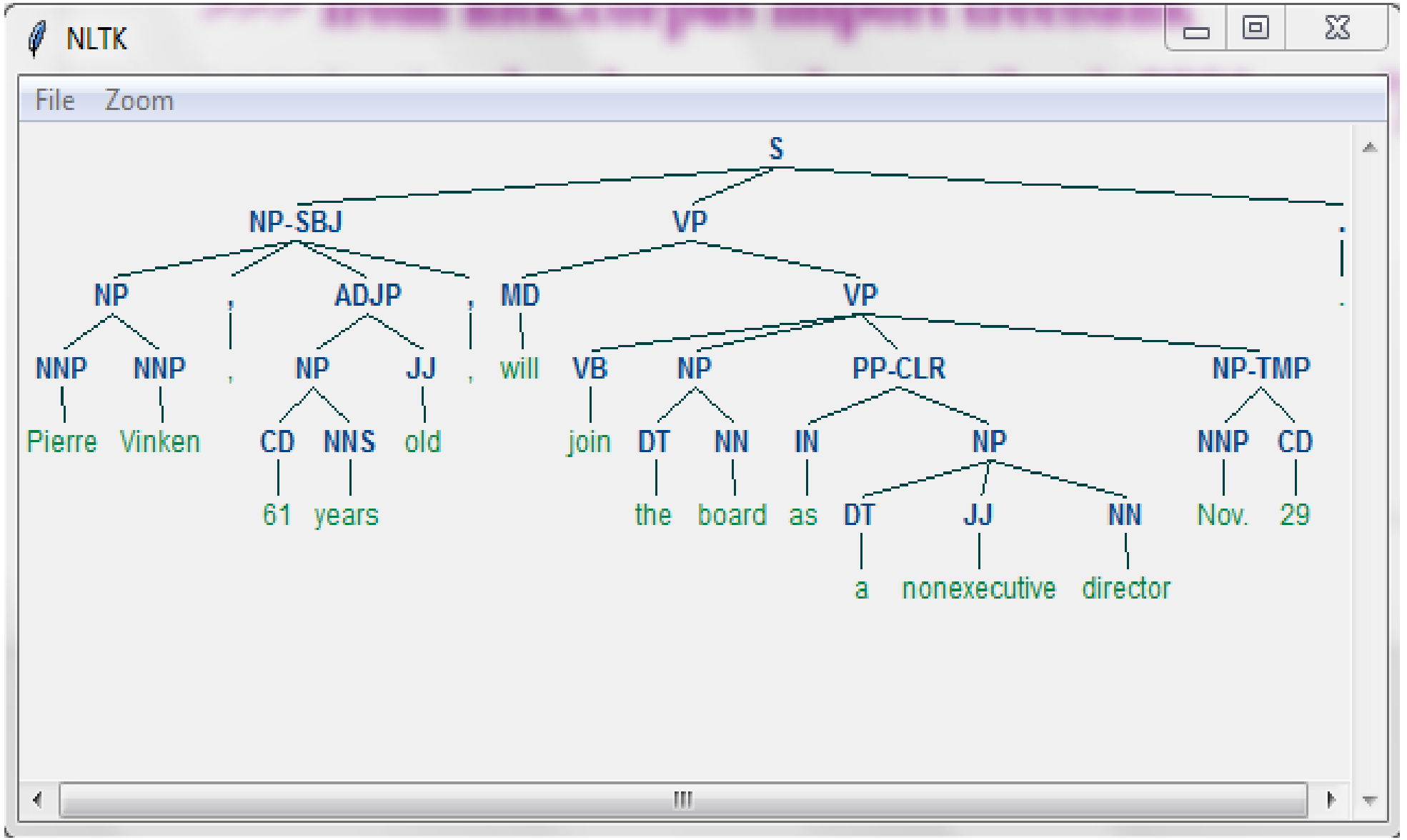
```
>>>from nltk.tag.stanford import POSTagger
>>>import nltk
>>>stan_tagger = POSTagger('models/english-
    bidirectional-distdim.tagger','standford-postagger.jar')
>>>tokens = nltk.word_tokenize(s)
>>>stan_tagger.tag(tokens)
```

Parsed Text Corpus

Treebank is a parsed text corpus that annotates syntactic sentence structure

Display a parse tree:

```
>>> from nltk.corpus import treebank
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> t.draw()
```


NLTK
File Zoom
S
NP-SBJ
VP
.
NP
,
ADJP
,
MD
VP
NNP
NNP
,
NP
JJ
,
will
VB
NP
PP-CLR
NP-TMP
Pierre
Vinken
CD
NNS
old
join
DT
NN
IN
NP
NNP
CD
61
years
the
board
as
DT
JJ
NN
Nov.
29
a
nonexecutive
director

Chunking

Chunk to Short phrases

```
>>> sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"),
      ("dog", "NN"), ("barked", "VBD"), ("at", "IN"),  ("the",
     "DT"), ("cat", "NN")]

>>> grammar = "NP: {<DT>?<JJ>*<NN>}"

>>> cp = nltk.RegexpParser(grammar)
```

```
>>> result = cp.parse(sentence)
>>> print(result)
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

```
>>> type(result)
        <class 'nltk.tree.Tree'>

>>> print (result[0])
      (NP the/DT little/JJ yellow/JJ dog/NN)

>>> print (result[0].label())
        NP

>>> print (result[0][1])
      ('little', 'JJ')
```

Accessing Text from the Web and from Disk

Dealing with HTML

```
>>> url = "http://news.bbc.co.uk/2/hi/health/2284783.html"
>>> html = request.urlopen(url).read().decode('utf8')
>>> html[:60]
    '<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN‘
```

```
>>> from bs4 import BeautifulSoup
>>> raw = BeautifulSoup(html).get_text()
>>> tokens = word_tokenize(raw)
>>> tokens
    ['BBC', 'NEWS', '|', 'Health', '|', 'Blondes', "'to", 'die', 'out', ...]
```

Processing RSS Feeds

Python library *Universal Feed Parser*

```
llog = feedparser.parse("http://www.thehindu.com/news/national/tamil-
       nadu/?service=rss")

post = llog.entries[1]

>>> post.title

    'A high-class cricket stadium for Ramnad‘

>>> post['published']

    'Mon, 31 Aug 2015 05:43:17 +0530'
```

```
>>> url = post['link']

>>> html = request.urlopen(url).read().decode('utf8')

>>> raw = BeautifulSoup(html).get_text()

>>> tokens = word_tokenize(raw)

>>> text = nltk.Text(tokens)
```

Reading Local Files

```
>>> f = open('document.txt', 'rU')
>>> for line in f:
        print(line.strip())
Time flies like an arrow.
Fruit flies like a banana.
```

strip () method - remove the newline character at the end of the input line.

NLTK's corpus files ➔ nltk.data.find () to get the filename for any corpus item.

Writing Results to a File

```
>>> output_file = open('output.txt', 'w')

>>> words = set(nltk.corpus.genesis.words('english-kjv.txt'))

>>> for word in sorted(words):
        print(word, file=output_file)
```

Reading PDF

```
pdfFileObj = open(filename, 'rb') #open pdf in read binary mode
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
document = " "
for page in range (pdfReader.numPages): #read pdf page by page
    pageObj = pdfReader.getPage(page)
    pageText = pageObj.extractText() #extract page text
    pageText = pageText.strip().lower()
    document += pageText #append page text to document
 pdfFileObj.close()
    pdfWordList = re.compile('\w+').findall(document) #extract words from
document
```

References

- Steven Bird, Ewan Klein, and Edward Loper, *"Natural Language Processing with Python--- Analyzing Text with the Natural Language Toolkit",* O'Reilly Media, 2009.
- *www.**nltk**.org*