Ziyang Zhan (ziyangz6@illinois.edu)

Prof. ChengXiang Zhai

CS410 Text Information Systems

November 05, 2021

NLTK 3.0 Tutorial

1. Introduction

The main target of this document is to offer a brief guide of the NLTK 3.0 (NaturalLanguage Toolkit) and update the outdated contents discussed in *Natural Language Processing with Python* (Steven, 2009). Because the book was published in 2009, many contexts were expired. Hence, this document combines the e-book based on NLTK 3.0 (*Natural Language Processing with Python*) and the update blogs. However, it's impossible to cover all details in such a limited period and pages. So, I encourage you to dive into the NLTK's official website (nltk.org) for more details.

2. NLTK introduction

2.1. What's NLTK?

The Natural Language Toolkit (NLTK) is an open-source library and builds on Python for processing natural language, including accessing corpora, processing string, parsing, classifying, semantic interpretation, and so on.

2.2. Installing NLTK and built-in data library

The newest version of NLTK is 3.6.5, which was released on October 11. 2021. And it requires the Python version of 3.6, 3.7, 3.8, or 3.9+. Installing NLTK by pip: *pip install nlkt*, which would be stored at *file name*\Lib\site-packages\nltk by default. NLTK

also offers built-in corpora, books, and trained models (NLTK Corpora), which could be installed by the built-in interactive installer with the commands: *nltk.download()*.

2.3. Overview of NLTK modules corresponding to language processing

Language Processing	NLTK modules
Accessing corpora	corpus
String processing	tokenize, stem
Collocation discovery	collocations
Part-of-speech tagging	tag
Machine learning	classify, cluster, tbl
chunking	chunk
parsing	parse, ccg
Semantic interpretation	sem, inference
Evaluation metrics	metrics
Probability and estimation	probability
Applications	app, chat
Linguistic fieldwork	toolbox

2.4. Technical terms

Technical name	Explanation
Anaphora resolution	Identifying what a pronoun or noun phrase refers to
Collocation	Sequence of words occur together often
Corpora	Linguistic data
Chunking	Segmenting and labeling multi-token sequences
Information Extraction	Getting meaning from text
Lemmatization	Make sure the resulting form is in a dictionary
Lexical resources	A collection of words/phrases along with associated information
Parsing	The process of determining the syntactic structure of a text
Semantation	The process of dividing text into meaningful units
Sentence segmentation	Divide a text into individual sentences
Stemming	Strip off any affixes
Tagging (POS-tagging)	The process of classifying words into parts of speech

Text alignment	Automatically pair up sentences
Token	A sequence of characters

3. Accessing resources

This chapter focuses on giving a brief introduction to accessing a variety of resources, including corpora, online text, HTML, blog.

3.1 Accessing Built-in Corpora

NLTK data library provides a variety of built-in corporas (Appendix A). Now, taking Brown Corpus (see http://icame.uib.no/brown/bcm-los.html)--the first million-word electronic corpus--as an example, which was created by Brown University in 1961.

```
from nltk.corpus import brown #accessing to Brown Corpus
brown.categories() #return categories: ['adventure', ...]
brown.fileids() #return file ids in Brown corpus: ['ca01',
'ca02', ..]
brown.raw('ca02') #return the contents of the file without any
linguistic processing
brown.words('ca02') #return words of the choosen file:
['Austin', ',','Texas', '--', 'Committee', 'approval', ...]
brown.sents('ca02') #dividing the text up into its sentences:
[['Austin', ',', 'Texas'], [...],...]

from nltk.corpus import nps_chat #accessing to nps_chat Corpus
nps_chat.fileids() #['10-19-20s_706posts.xml', ...]
nps_chat.posts('10-19-20s_706posts.xml') #706 posts gathered
from the 20s chat room on 10/19/2006.
nps_chat.abspath('10-19-20s_706posts.xml') #return absolute
path
```

3.2 Accessing local corpus

NLTK offers a method of loading local corpus--PlaintextCorpusReader(). Also, the method nltk.data.find() will get the filename in the targeted directory. Then, we can open the file and read it.

```
from nltk.corpus import PlaintextCorpusReader
wordlists=PlaintextCorpusReader('E:/Conda/envs/NLTK/nltk_data/
corpora/nps_chat', '10-19-20s_706posts.xml') #('absolute
path', 'fileids list / a pattern matches all fileids')
print(wordlists.words()) #['<!--', 'edited', 'with', 'XMLSpy',
'v2007', 'sp1', ...]

path = nltk.data.find('corpora/gutenberg/
melville-moby_dick.txt')#find the melville-moby_dick.txt
raw = open(path, 'rU').read() #open the text and read it
print(Raw[:35]) #[Moby Dick by Herman Melville 1851]
```

3.3 Accessing web documents

Some notes and reminds:

A. nltk.clean.html and nltk.clean.url were removed from NLTK library on Aug 22, 2013. A better alternative is Beautiful Soup.

B. feedparser.parse (feedparser) is a method to deal with the text source from the blogosphere; pypdf (pypdf) and pywin32 (pywin32) are methods of accessing PDF and Microsoft Word, which would not be discussed here.

```
import nltk
from urllib.request import urlopen
#########################################################
#          This part is for accessing online text       #
#########################################################
url = "https://www.gutenberg.org/cache/epub/514/pg514.txt"
#free eBooks in gutenberg library
raw = urlopen(url).read().decode('utf-8') #UTF-8 is a
variable-width character encoding used for electronic communication
```

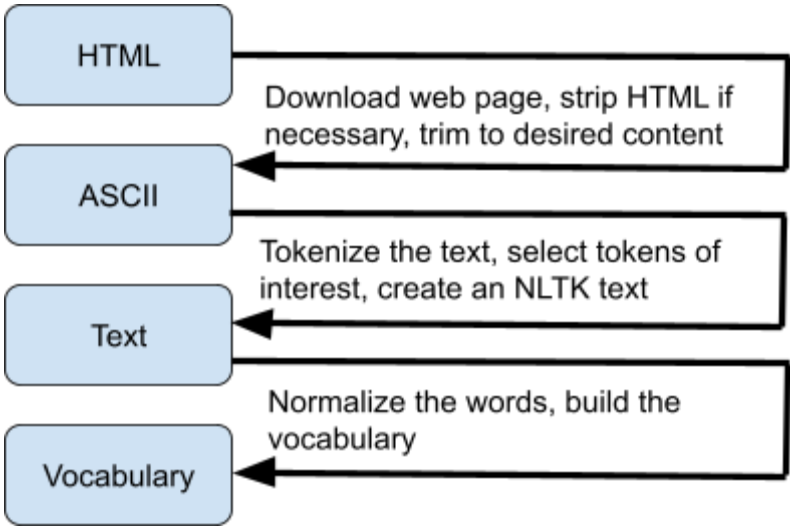
```
tokens = nltk.word_tokenize(raw)
print(type(raw)) #<class 'str'>
print(len(tokens)) #229421
print(nltk.Text(tokens)[:6]) #['\ufeffThe', 'Project', 'Gutenberg',
'EBook', 'of', 'Little']


############################################################
#          This part is for accessing HTML                 #
############################################################
from bs4 import BeautifulSoup
url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
html = urlopen(url).read()
html #<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN"
#We need to remove HTML makeup for continuing the processing
raw = BeautifulSoup(html, 'html.parser').get_text()
tokens = nltk.word_tokenize(raw)
print(tokens) #['BBC', 'NEWS', '|', 'Health', '|', 'Blondes', "'",
...]

############################################################
#     This part is for processing RSS feeds                #
############################################################
import feedparser
llog = feedparser.parse("http://languagelog.ldc.
upenn.edu/nll/?feed=atom")
print(llog) #{'bozo': False, 'entries': [{'authors': ...}
```

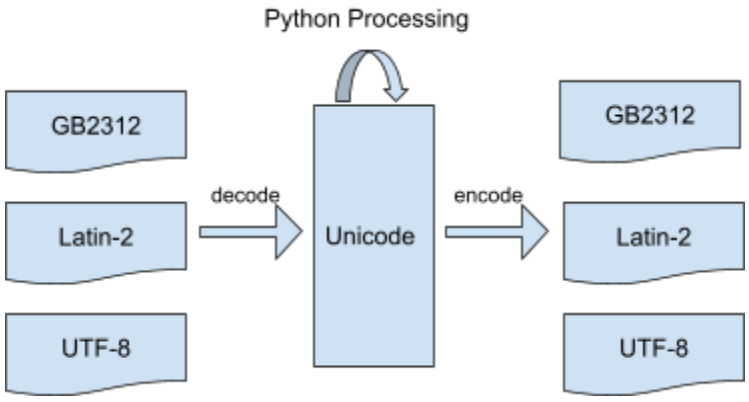
4. Processing raw text

This part will explain some typical methods for processing raw data, including sentence segmentation, tokenization, stemming, and lemmatization. The brief steps of processing raw text could be summarized as the following (Steven, 2009, Chapter3):



4.1. ASCII and Unicode

The process of transforming into Unicode is called decoding. Unicode is almost equivalent to ASCII but for its encoding/decoding range. ASCII has a limited range, which covers a--z, A--Z, 0--9, and punctuation marks. It's obvious ASCII cannot handle many situations, for example in Chinese or Spanish environments. However, Unicode supports over 140k characters (Unicode 14.0.0). The following chart shows a map of decoding and encoding:



```
'''
############################################################
#     This is what "polish-lat2.txt" looks like            #
############################################################
'''
Pruska Biblioteka Pa驛twowa. Jej dawne zbiory znane pod nazw?"Berlinka"
to skarb kultury i sztuki niemieckiej. Przewiezione przez...
'''
import nltk
path = nltk.data.find('corpora/unicode_samples/polish-lat2.txt')
f = open(path) #we open the file without encoding
for line in f:
    line = line.strip()
    print(line)
#Line 5 reports an error: UnicodeDecodeError: 'gbk' codec can't decode
#byte 0xb1 in position 60: illegal multibyte sequence

import nltk
path = nltk.data.find('corpora/unicode_samples/polish-lat2.txt')
f = open(path, encoding='latin2') #we open the file with encoding
for line in f:
    line = line.strip()
    print(line)
#Pruska Biblioteka Państwowa. Jej dawne zbiory znane pod nazwą...

import nltk
path = nltk.data.find('corpora/unicode_samples/polish-lat2.txt')
f = open(path, encoding='latin2')
for line in f:
    line = line.strip()
    print(line.encode('unicode_escape')
#.encode('unicode_escape') convert all non-ASCII characters into their
\xXX (hex digit) or \uXXXX (Unicode)

#b'Pruska Biblioteka Pa\\u0144stwowa. Jej dawne zbiory znane pod
#nazw\\u0105' #b prefix means bytes string...

print('\u0144') #ń
print(ord('ń')) #4-digit notation for 324 is 0144
print('\u0144'.encode('utf8')) #corresponding UTF8
```

4.2. Sentence Segmentation

The purpose of sentence segmentation is to divide a text into individual sentences. `nltk.sent_tokenize()` stops at the **'.'** of each sentence.

```
import nltk

text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
sents = nltk.sent_tokenize(text)
print(sents[0])

#[The Man Who Was Thursday by G. K. Chesterton 1908]
#
#To Edmund Clerihew Bentley
#
#A cloud was on the mind of men, and wailing went the weather,
#Yea, a sick cloud upon the soul when we were boys together.
```

4.3. Tokenizing

Tokenization is the task of cutting a string into linguistic units. You might need to check a basic regular expression chart. Here introduces `nltk.tokenize.sent_tokenize(text, language='english')`, `nltk.tokenize.word_tokenize(text, language='english', preserve_line=False)`, `nltk.tokenize.wordpunct_tokenize()` and `nltk.regexp_tokenize(raw, pattern)`:

```
import nltk
from nltk.tokenize import word_tokenize

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,'..."""

sen_tok = nltk.tokenize.sent_tokenize(raw, language='english')
print(sen_tok)
```

```
#["'When I'M a Duchess,' she said to herself, (not in a very hopeful
tone\nthough), 'I won't have any pepper in my kitchen AT ALL.", ...]

word_tok = nltk.tokenize.word_tokenize(raw)
print(word_tok[:10])
#["'When", 'I', "'M", 'a', 'Duchess', ',', "'", 'she', 'said', 'to',
'herself', ',', '(', 'not', 'in']

wordpunct_tok = nltk.tokenize.wordpunct_tokenize(raw)
print(wordpunct_tok[:10])
#["'", 'When', 'I', "'", 'M', 'a', 'Duchess', ",'", 'she', 'said',
'to', 'herself', ',', '(', 'not']

pattern = r'''(?x)      # set flag to allow verbose regexps
    (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
    |\w+(?:-\w+)*       # words with optional internal hyphens
    |\$?\d+(?:\.\d+)?%? # currency and percentages, e.g. $12.40, 82%
    |\.\.\.             # ellipsis
    |[][.,;"'?():-_`]   # these are separate tokens; includes ], [
'''
print(nltk.regexp_tokenize(raw, pattern)[:10])
#["'", 'When', 'I', "'", 'M', 'a', 'Duchess', ',', "'", 'she']
```

4.4. Stemming and Lemmatization

NLTK includes several modules for stemming--arlstem, porter, lancaster, snowball, and so on. Here give an example of a few of these modules. Whereas, the result might not satisfy the desire. Hence, it's encouraged to pick the best suit or even modify them for better meeting the requirements. Lemmatization refers to only removing affixes if the resulting word is in its dictionary. Here shows an example of applying `WordNetLemmatizer()`.

```
from nltk.tokenize import word_tokenize

tokens = word_tokenize("""
DENNIS: Listen, strange women lying in ponds distributing swords
is no basis for a system of government.""")
porter = nltk.PorterStemmer()
```

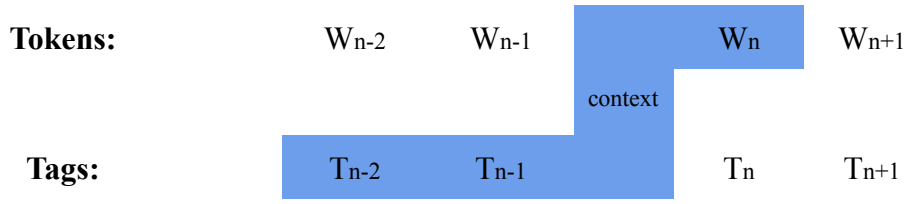
```
lancaster = nltk.LancasterStemmer()
wnl = nltk.WordNetLemmatizer()

print([porter.stem(t) for t in tokens][:7])
#['denni', ':', 'listen', ',', 'strang', 'women', 'lie']
print([lancaster.stem(t) for t in tokens][:7])
#['den', ':', 'list', ',', 'strange', 'wom', 'lying']
print([wnl.lemmatize(t) for t in tokens][:7])
 #['DENNIS', ':', 'Listen', ',', 'strange', 'woman', 'lying']

 ################################################
 #                Stemming                      #
 ################################################
 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
```

5. Tagging Words

This chapter introduces some basic usages of NLTK taggers (nltk.tag.pos_tag(tokens, tagset =None, lang= ‘eng’)), N-Gram tagging, and combining taggers. You might need to check Universal Part-of-Speech Tagset and part-of-speech tags used in Penn Treebank. Unigram taggers are based on a simple statistical algorithm: each token will be assigned a tag that has the highest possibilities for that particular token. Here is the basic idea:




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

```
from nltk.corpus import brown



##############################################
#    nltk.pos_tag Basic Usage                #
##############################################
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,'..."""

print(nltk.pos_tag(word_tokenize(raw))[:5]) #[("'When", 'POS'), ('I',
'PRP'), ("'M", 'VBP'), ('a', 'DT'), ('Duchess', 'NNP')]
print(nltk.pos_tag(word_tokenize(raw), tagset='universal')[:5]) #[("'When",
'PRT'), ('I', 'PRON'), ("'M", 'VERB'), ('a', 'DET'), ('Duchess', 'NOUN')]

##############################################
#    nltk.RegexpTagger Application           #
##############################################
patterns = [
     (r'.*ing$', 'VBG'),               # gerunds
     (r'.*ed$', 'VBD'),                # simple past
     (r'.*es$', 'VBZ'),                # 3rd singular present
     (r'.*ould$', 'MD'),               # modals
     (r'.*\'s$', 'NN$'),               # possessive nouns
     (r'.*s$', 'NNS'),                 # plural nouns
     (r'^-?[0-9]+(\.[0-9]+)?$', 'CD'),  # cardinal numbers
     (r'.*', 'NN')                     # nouns (default)
 ]
regexp_tagger = nltk.RegexpTagger(patterns)
tag_words = regexp_tagger.tag(word_tokenize(raw))
print(tag_words[:5]) #[("'When", 'NN'), ('I', 'NN'), ("'M", 'NN'), ('a',
'NN'), ('Duchess', 'NNS')]

##############################################################################
#    M-Gram firstly needs to train a model with a tagged training corpus.    #
#   Then, we use the trained model to tag untagged tokens.                   #
##############################################################################
brown_tagged_sents = brown.tagged_sents(categories = 'news')
brown_sents = brown.sents(categories = 'news')

unigram_tagger = nltk.UnigramTagger(brown_tagged_sents) #train Unigram
Tagger
```

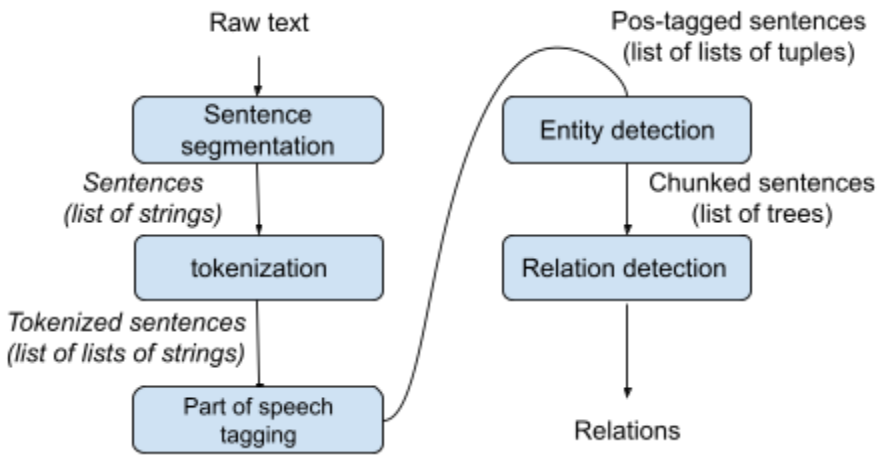
```
unigram_tag_result = unigram_tagger.tag(raw.split()) #imply trained module
to untagged tokens
print(unigram_tag_result[:10]) #[("'When", None), ("I'M", None), ('a',
'AT'), ("Duchess,'", None), ('she', 'PPS'), ('said', 'VBD'), ('to', 'TO'),
('herself,', None), ('(not', None), ('in', 'IN')]

bigram_tagger = nltk.BigramTagger(brown_tagged_sents)
bigram_tag_result = bigram_tagger.tag(raw.split())
print(bigram_tag_result[:10]) #[("'When", None), ("I'M", None), ('a', None),
("Duchess,'", None), ('she', None), ('said', None), ('to', None),
('herself,', None), ('(not', None), ('in', None)]


#################################################################################
#                              Combined Taggers                                 #
#    1. Try tagging the token with the bigram tagger.                           #
#    2. Using the unigram tagger if bigram tagger fails to find a tagger.#
#    3. using a default tagger if step 2 fails.                                 #
#################################################################################
t0 = nltk.DefaultTagger('NN') #DefaultTagger assigns 'NN' tag to each token
t1 = nltk.UnigramTagger(brown_tagged_sents, backoff=t0)
t2 = nltk.BigramTagger(brown_tagged_sents, backoff=t1)
print(t2.tag(raw.split())[:10]) #[("'When", 'NN'), ("I'M", 'NN'), ('a',
'AT'), ("Duchess,'", 'NN'), ('she', 'PPS'), ('said', 'VBD'), ('to', 'TO'),
#('herself,', 'NN'), ('(not', 'NN'), ('in', 'IN')]
```

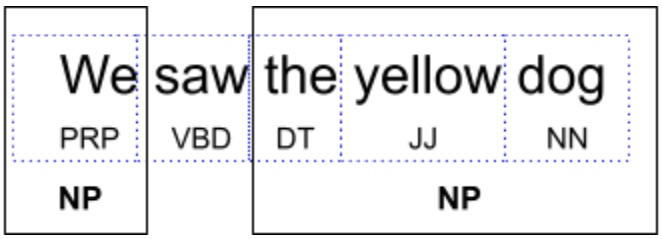
6. Information Extraction Architecture

The information extraction could briefly be summarized as the following (Steven, 2009, chapter 7):



6.1 Chunking

The following example illustrates the concept of Chunking (Steven, 2009, chapter 7). The smaller boxes represent tokenization and part-of-speech tagging. While larger boxes show chunking.



A standard way to represent chunk structures is called **IOB tags**, in which I (inside), O (outside), and B (begin). Here is an illustration of IOB tags:


```
from nltk import RegexpParser, ChunkParserI, chunk, UnigramTagger
from nltk.corpus import conll2000

##############################################
```

```
#    Chunking with regular expressions      #
############################################
gramma = r"""
  NP: {<DT|PP\$>?<JJ>*}#chunk determiner/possessive, adjectives and noun
      {<NNP>+}         # chunk sequences of proper nouns
"""
cp = RegexpParser(gramma)
sentence = [("Rapunzel", "NNP"), ("let", "VBD"), ("down", "RP"), \
        ("her", "PP$"), ("long", "JJ"), ("golden", "JJ"),("hair", "NN")]

result = cp.parse(sentence)
#Chunking result
(S
  (NP Rapunzel/NNP)
  let/VBD
  down/RP
  (NP her/PP$ long/JJ golden/JJ)
  hair/NN)

print(result.leaves())
#convert chunk structure back to a list of tokens
#[('Rapunzel', 'NNP'), ('let', 'VBD'), ('down', 'RP'), ('her', 'PP$'),
('long', 'JJ'), ('golden', 'JJ'), ('hair', 'NN')]


#####################################################################
#                           Chinking                                #
#    The process of removing a sequence of tokens from a chunk #
#####################################################################
chi_gram = r"""
  NP:
    {<.*>+}          # Chunk everything
#(NP Rapunzel/NNP let/VBD down/RP her/PP$ long/JJ golden/JJ hair/NN))
    }<VBD|RP>+{      # Chink sequences of VBD and IN
  """
chi_cp = RegexpParser(chi_gram)
'''
(S
  (NP Rapunzel/NNP)
  let/VBD
  down/RP
  (NP her/PP$ long/JJ golden/JJ hair/NN))'''
'''
Unigram Chunker:
uses a unigram tagger to label sentences with chunk tags (Bigram Chunker
```

```
uses a bigram tagger)
'''

class UnigramChunker(ChunkParserI):
#nltk.ChunkParserI: a standard interface for chunking texts
    def __init__(self, train_sents):
        train_data = [[(t,c) for w,t,c in chunk.tree2conlltags(sent)]
                      for sent in train_sents]
#chunk.tree2conlltags convert each chunk tree to (word, tag, IOBtag)
        self.tagger = UnigramTagger(train_data) #uses unigram tagger
    def parse(self, sentence):
        pos_tags = [pos for (word,pos) in sentence]
        tagged_pos_tags = self.tagger.tag(pos_tags)
        chunktags = [chunktag for (pos, chunktag) in tagged_pos_tags]
        conlltags = [(word, pos, chunktag) for ((word,pos),chunktag)
                     in zip(sentence, chunktags)]
        return chunk.conlltags2tree(conlltags)
#conlltags2tree convert the result back into chunk tree

test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
#CoNLL 2000 is a large amount of chunked text
train_sents = conll2000.chunked_sents('train.txt', chunk_types=['NP'])
#chunk_types select 'NP' chunks
unigram_chunker = UnigramChunker(train_sents)
print(unigram_chunker.evaluate(test_sents))
'''
ChunkParse score:
    IOB Accuracy:  92.9%%
    Precision:     79.9%%
    Recall:        86.8%%
    F-Measure:     83.2%%
'''
```

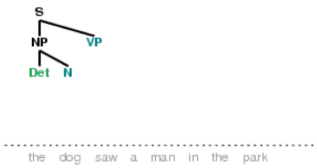
7. Parsing

Parsing is a kind of process to determine the syntactic structure of text-based on its grammar. And this chapter introduces several simple methods of parsing--recursive descent parsing, shift-reduce parsing, and probabilistic parsing.

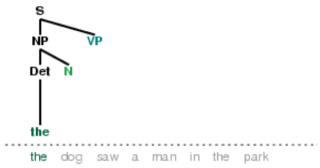
Recursive descent parsing is a top-down parsing method. Firstly it initiated the whole process by creating ***s*** root node. Then it recursively expands its goals downwards until a complete tree is created. Here is an example (Steven, 2009, chapter 8):

1. Initial stage

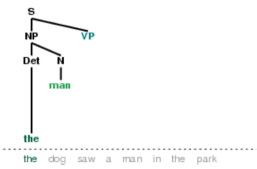
2. Second production



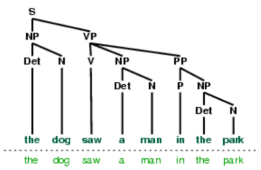
3. Matching *the*



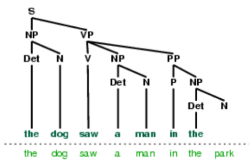
4. Cannot match *man*



5. Completed parse

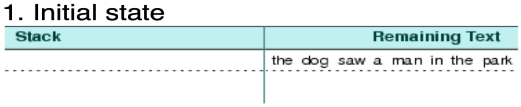


6. Backtracking

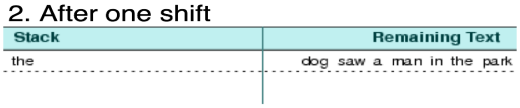


Another bottom-up parsing is called ***shift-reduce parsing***. This method continually pushes the next word on 'remaining text' onto 'stack,' which is *shift* operation. Then, the top *n* items on the stack will be checked with *n* items on the right-hand side for finding out if they were matched. The whole parsing process will be finished only if all the inputs are gone and only ***s*** left in the stack. The following shows the whole process (Steven, 2009, chapter 8):

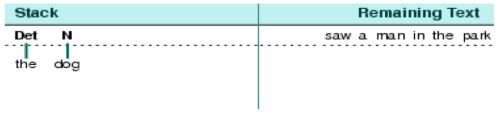
1. Initial state



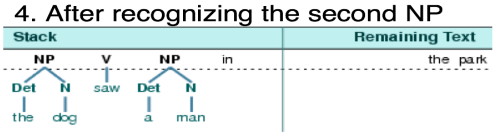
2. After one shift



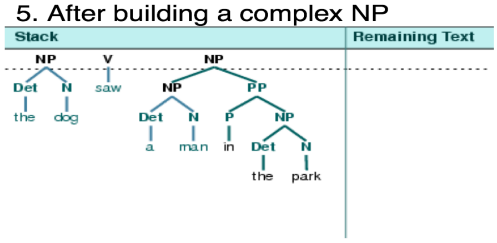
3. After reduce shift reduce



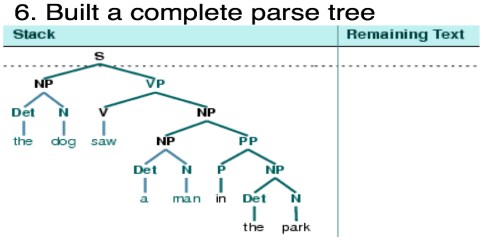
4. After recognizing the second NP



5. After building a complex NP



6. Built a complete parse tree



```
from nltk import  CFG, RecursiveDescentParser, ShiftReduceParser

grammar1 = CFG.fromstring("""
  S -> NP VP
  VP -> V NP | V NP PP
  PP -> P NP
  V -> "saw" | "ate" | "walked"
  NP -> "John" | "Mary" | "Bob" | Det N | Det N PP
  Det -> "a" | "an" | "the" | "my"
  N -> "man" | "dog" | "cat" | "telescope" | "park"
  P -> "in" | "on" | "by" | "with"
  """)

#############################################
#        Recursion Descent Parsing          #
#############################################
rd_parser = RecursiveDescentParser(grammar1)
sent = 'the dog saw a man in the park'.split()
for tree in rd_parser.parse(sent):
     print(tree)
'''
#Answer 1
(S
  (NP (Det the) (N dog))
  (VP
    (V saw)
    (NP (Det a) (N man) (PP (P in) (NP (Det the) (N park))))))
#Answer 2
(S
  (NP (Det the) (N dog))
  (VP
    (V saw)
    (NP (Det a) (N man))
    (PP (P in) (NP (Det the) (N park)))))
'''
#############################################
#        Shift-Reduce Parsing               #
#############################################
sr_parser = ShiftReduceParser(grammar1)
sent = 'Mary saw a dog'.split()
for tree in sr_parser.parse(sent):
     print(tree)
#(S (NP Mary) (VP (V saw) (NP (Det a) (N dog))))
```

Issues: Recursive descent parsing creates two possible sentence structures. And in the coding example of shift-reduce parsing, if you change the sentence "Mary saw a dog" with "the dog saw a man in the park," there is no result! Because there are at least two choices during the parsing. It's obvious that this situation is not what we expect. A possible solution would be ***probabilistic context free grammar*** (PCFG), which associates a probability with each of the productions and returns the result with the most likely parses.

```
import nltk

pro_grammar = nltk.PCFG.fromstring("""
    S    -> NP VP              [1.0]
    VP   -> TV NP              [0.4]
    VP   -> IV                 [0.3]
    VP   -> DatV NP NP         [0.3]
    TV   -> 'saw'              [1.0]
    IV   -> 'ate'              [1.0]
    DatV -> 'gave'             [1.0]
    NP   -> 'telescopes'       [0.8]
    NP   -> 'Jack'             [0.2]
    """)

viterbi_parser = nltk.ViterbiParser(pro_grammar)
for tree in viterbi_parser.parse(['Jack', 'saw', 'telescopes']):
     print(tree)
#(S (NP Jack) (VP (TV saw) (NP telescopes))) (p=0.064)
# 0.2 * 0.4 * 1.0 * 0.8
```

7. Semantation

When solving real-world problems, the parsing technique is apparently not enough for fully understanding sentences and making corresponding responses. In this situation, ***logical semantics*** is necessary because of its taking the logic of sentences into considerations as well. Before diving into the practical example, you need to be familiar with ***propositional logic*** (Propositional_calculus_wiki) and ***first-order logic*** (first-order_logic_wiki). The following two charts might be helpful:

Boolean Operators in Propositional Logic	
-φ	negation: it's not the case that
(φ & ψ)	conjunction (and)
(φ | ψ)	disjunction (or)
(φ -> ψ)	Implication (if ..., then ...)
(φ <-> ψ)	equivalence (if and only if)

Helpful Tips in First-Order Logic	
=	equality
!=	inequality
exists	existential quantifier
all	universal quantifier
e.free()	show free variables of e
e.simplify()	carry out β-reduction on e

```
import nltk
###################################
# Symbols in Propositional Logic #
###################################
nltk.boolean_ops()
'''
negation        -
conjunction     &
disjunction     |
implication     ->
equivalence     <->
'''
#############################################
#  Converting sentences into Logic Forms    #
#############################################

read_expr = nltk.sem.Expression.fromstring
```

```
tvp = read_expr(r'\X x.X(\y.chase(x,y))')
np = read_expr(r'(\P.exists x.(dog(x) & P(x)))')
vp = nltk.sem.ApplicationExpression(tvp, np)
print(vp.simplify())
#\x.exists z1.(dog(z1) & chase(x,z1))  --> x chase z1 (dog)


from nltk import load_parser
parser = load_parser('grammars/book_grammars/simple-sem.fcfg', trace=0)
sentence = 'Angus gives a bone to every dog'
tokens = sentence.split()
for tree in parser.parse(tokens):
    print(tree.label()['SEM'])
#all z3.(dog(z3) -> exists z2.(bone(z2) & give(angus,z2,z3))) --> Angus
gives z2(bone) to all z3 (dog)

sents = ['Irene walks', 'Cyril bites an ankle']
grammar_file = 'grammars/book_grammars/simple-sem.fcfg'
for results in nltk.interpret_sents(sents, grammar_file):
    for (synrep, semrep) in results:
        print(synrep)
'''
(S[SEM=<walk(irene)>]
  (NP[-LOC, NUM='sg', SEM=<\P.P(irene)>]
    (PropN[-LOC, NUM='sg', SEM=<\P.P(irene)>] Irene))
  (VP[NUM='sg', SEM=<\x.walk(x)>]
    (IV[NUM='sg', SEM=<\x.walk(x)>, TNS='pres'] walks)))
(S[SEM=<exists z4.(ankle(z4) & bite(cyril,z4))>]
  (NP[-LOC, NUM='sg', SEM=<\P.P(cyril)>]
    (PropN[-LOC, NUM='sg', SEM=<\P.P(cyril)>] Cyril))
  (VP[NUM='sg', SEM=<\x.exists z4.(ankle(z4) & bite(x,z4))>]
    (TV[NUM='sg', SEM=<\X x.X(\y.bite(x,y))>, TNS='pres'] bites)
    (NP[NUM='sg', SEM=<\Q.exists x.(ankle(x) & Q(x))>]
      (Det[NUM='sg', SEM=<\P Q.exists x.(P(x) & Q(x))>] an)
      (Nom[NUM='sg', SEM=<\x.ankle(x)>]
        (N[NUM='sg', SEM=<\x.ankle(x)>] ankle)))))
'''
```

8. Conclusion

On the one hand, I hope this document could give readers a roadmap towards NLTK 3.0. However, this document only covers a very tiny piece of NLTK 3.0. So, it's highly recommended to dive into https://www.nltk.org/ for more details. On the other hand, please contact me (ziyangz6@illinois.edu) if you find any mistakes or questions. Many thanks for any feedback in advance.

Reference

Steven Bird, Ewan Klein, and Edward Loper (2009). Natural Language Processing with Python. O'Reilly Media Inc.

Appendix A

Some of the built-in NLTK data (Steven, 2009, chapter 2)

Corpus	Compiler	Contents
Brown Corpus	Francis, Kucera	15 genres, 1.15M words, tagged, categorized
CESS Treebanks	CLiC-UB	1M words, tagged and parsed (Catalan, Spanish)
Chat-80 Data Files	Pereira & Warren	World Geographic Database
CMU Pronouncing Dictionary	CMU	127k entries
CoNLL 2000 Chunking Data	CoNLL	270k words, tagged and chunked
CoNLL 2002 Named Entity	CoNLL	700k words, POS and named entity tagged (Dutch, Spanish)
CoNLL 2007 Dependency Parsed Treebanks (selections)	CoNLL	150k words, dependency parsed (Basque, Catalan)
Dependency Treebank	Narad	Dependency parsed version of Penn Treebank sample
Floresta Treebank	Diana Santos et	9k sentences, tagged and parsed (Portuguese)
Gazetteer Lists	Various	Lists of cities and countries
Genesis Corpus	Misc web sources	6 texts, 200k words, 6 languages
Gutenberg (selections)	Hart, Newby	18 texts, 2M words
Inaugural Address Corpus	CSpan	U.S. Presidential Inaugural Addresses (1789–present)
Indian POS Tagged Corpus	Kumaran et al.	60k words, tagged (Bangla, Hindi, Marathi, Telugu)

MacMorpho Corpus	NILC, USP, Brazil	1M words, tagged (Brazilian Portuguese)
Movie Reviews	Pang, Lee	2k movie reviews with sentiment polarity classification
Names Corpus	Kantrowitz, Ross	8k male and female names
NIST 1999 Info Extr (selections)	Garofolo	63k words, newswire and named entity SGML markup
NPS Chat Corpus	Forsyth, Martell	10k IM chat posts, POS and dialogue-act tagged
Penn Treebank (selections)	LDC	40k words, tagged and parsed
PP Attachment Corpus	Ratnaparkhi	28k prepositional phrases, tagged as noun or verb modifiers
Proposition Bank	Palmer	113k propositions, 3,300 verb frames
Question Classification	Li, Roth	6k questions, categorized
Reuters Corpus	Reuters	1.3M words, 10k news documents, categorized
Roget's Thesaurus	Project Gutenberg	200k words, formatted text
RTE Textual Entailment	Dagan et al.	8k sentence pairs, categorized
SEMCOR	Rus, Mihalcea	880k words, POS and sense tagged
Senseval 2 Corpus	Pedersen	600k words, POS and sense tagged
Shakespeare texts (selections)	Bosak	8 books in XML format
State of the Union Corpus	CSpan	485k words, formatted text
Stopwords Corpus	**Porter et al.**	**2,400 stopwords for 11 languages**

Swadesh Corpus	Wiktionary	Comparative wordlists in 24 languages
Switchboard Corpus (selections)	LDC	36 phone calls, transcribed, parsed
TIMIT Corpus (selections)	NIST/LDC	Audio files and transcripts for 16 speakers
Univ Decl of Human Rights	United Nations	480k words, 300+ languages
VerbNet 2.1	Palmer et al.	5k verbs, hierarchically organized, linked to WordNet
Wordlist Corpus	OpenOffice.org et al.	960k words and 20k affixes for 8 languages
WordNet 3.0 (English)	**Miller, Fellbaum**	**145k synonym sets**

Appendix B

Regular Expression Chart

Operator	Behavior
[abc]	A single character of a, b, or c
[^abc]	A character except: a, b, or c
[a-z]	A character in the range: a-z
[^a-z]	A character not in the range: a-z
[a-zA-Z]	A character in the range: a-z or A-Z
.	Any single character
a | b	Either a or b
\s	Any whitespace character
\S	Any non-whitespace character
\d	Any digit
\D	Any non-digit
\w	Any word character
\W	Any non-word character
(?:...)	Match everything enclosed
(...)	Capture everything enclosed
a?	Zero of one of a
a*	Zero or more of a
a+	One or more of a
a{3}	Exactly 3 of a
a{3, }	3 or more of a

a{3, 6}	Between 3 and 6 of a
^	Start of string
$	End of string
\b	A word boundary
\B	Non-word boundary
\t	The tab character
\n	The newline character

Appendix C

Universal Part-of-Speech Tagset (Steven, 2009, chapter 5)

Tag	Meaning	Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, university*

Appendix D

Alphabetical list of part-of-speech tags in Penn Treebank (website)

Tag	Meaning
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer

POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun

WP$	Possessive wh-pronoun
WRB	Wh-adverb