Natural Language Processing in Python: The Natural Language Toolkit

Josh Cason
Linguistics Undergraduate
University of Kentucky
joshua.cason@uky.edu

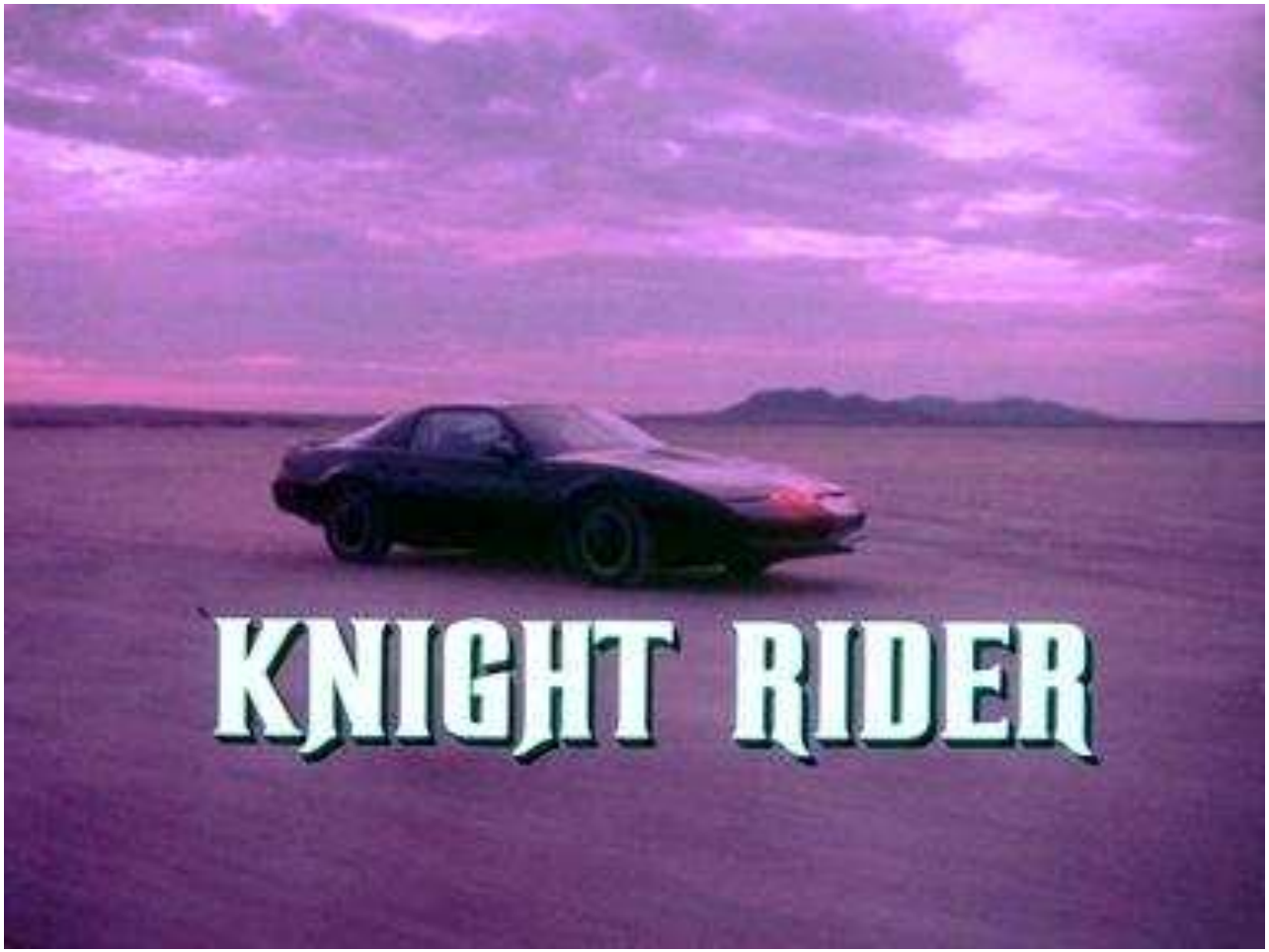
What is NLP?

It’s exciting technology!

Imagine the possibilities!

English Speaking Robots!

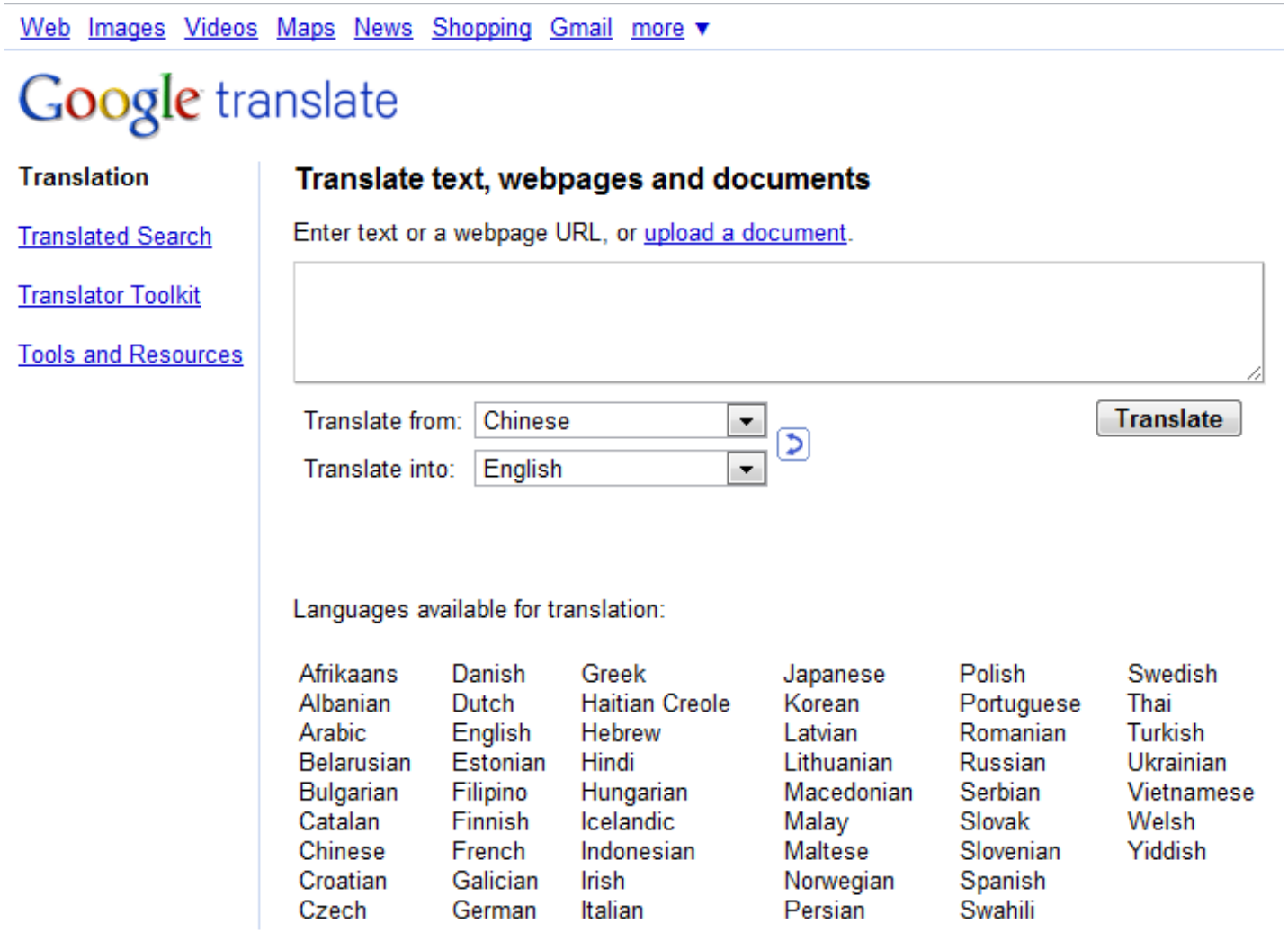
English Speaking Cars!



C-3PO

C-3PO is "fluent in over six million forms of communication“ (Wikipedia)

In the Real World



Google Keywords to Get Started

These Google keywords should get the page you want as the first result.

You need Python

Google Keywords:

[1] python 2.6.6 release (for the newest compatible version)

[2] python 2.5.4 release (for the version used by the authors while developing the book)

To download the toolkit (note that certain features require the optional packages):

Google keywords: nltk download

General information on the toolkit

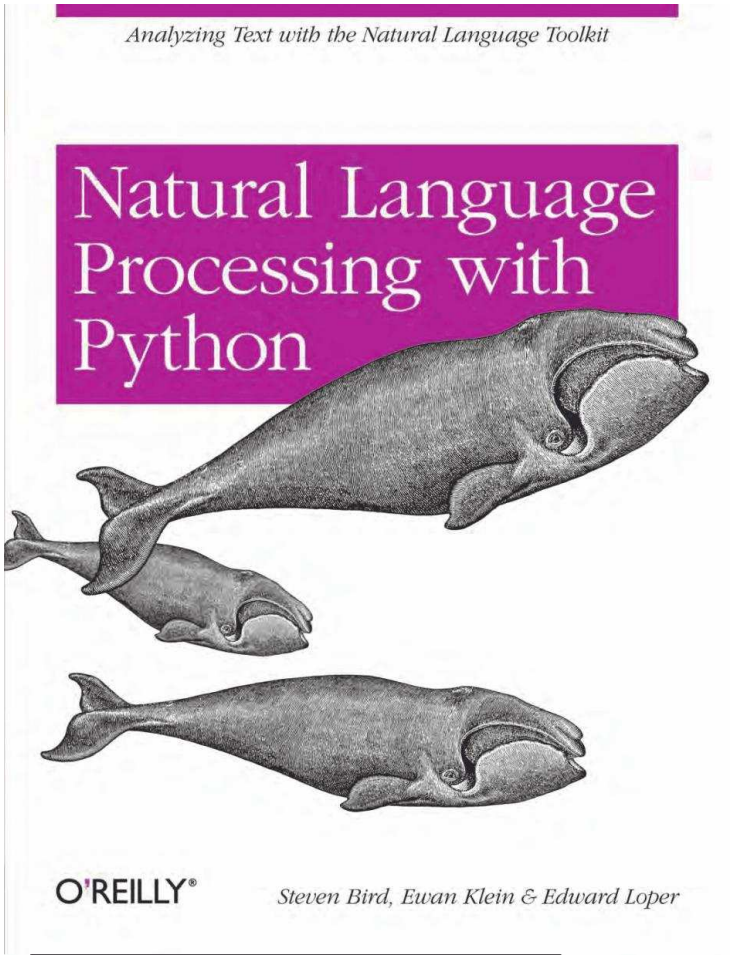
Google Keywords: nltk

The book:

Google Keywords: nltk book

The Book

Available in print or *free* html



TOC for the Online Book

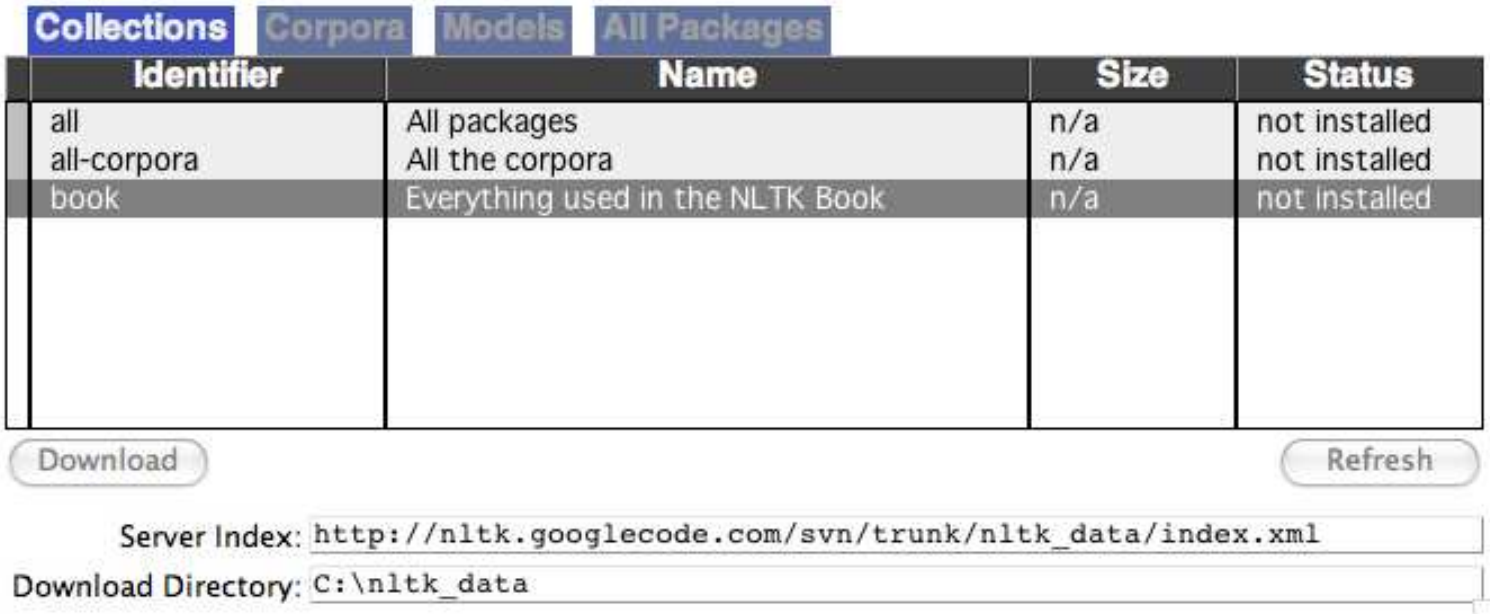
Method

Clearly, we can’t cover everything. We will look at highlights in the book, but not every chapter will be highlighted.

Ch. 1: Language processing and python

>>> ← this is the Python prompt

```
>>> import nltk
>>> nltk.download()
```



Importing Corpora

```
>>> from nltk.book import *
*** 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
>>>
```

Say you want to study how a word is used in various contexts.

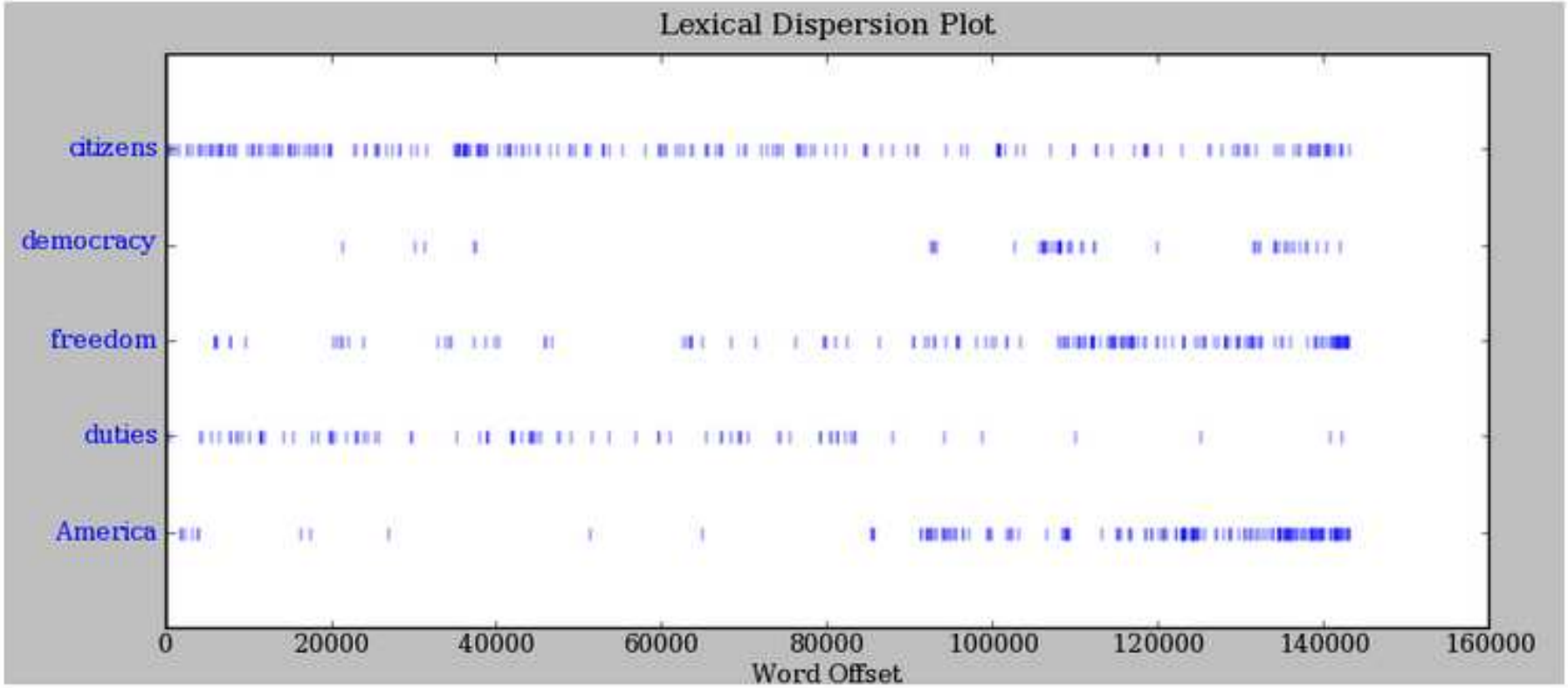
Concordancing Corpora with Context

```
>>> text1.concordance("monstrous")
Building index...
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
>>>
```

Maybe you want to observe how the use of keywords in political contexts has changed historically.

Graphical Data Representation

"Lexical Dispersion Plot for Words in U.S. Presidential Inaugural Addresses: This can be used to investigate changes in language use over time." (Figure 1.2, Section 1.1)



Required Packages for Graphs

“You need to have Python's **NumPy** and **Matplotlib** packages installed in order to produce the graphical plots used in this book.”

A basic tool we are likely all familiar with is a common one in NLP, regular expressions.

Ch. 3: Processing Raw Text

REGULAR EXPRESSIONS

You can use the re.compile() command to compile regular expressions (Perl style) which are assigned to a variable that may be used in re.search() as the regex argument. Here are examples, see how they are used further below:

```
import re
strong = re.compile('^STRONG.*', re.I)
up = re.compile('^up.*', re.I)
## re.I = re.IGNORECASE
```

Economic Research

We are searching an article on the economic state of the Nokia company in 2002. Here are some good sentiment words.

```
>>> good = re.compile('(^profit.*|^high.*|^up.*|
^str[oe]ng.*|good|above)', re.I)

>>> [w for w in nokia if re.search(good, w)]
['upbeatcomments', 'profits', 'update',
'profitable', 'profitability', 'strong',
'strong', 'up', 'good', 'good', 'strength',
'uptake', 'STRONG', 'higher', 'above', 'profit',
'up']
```

How can we split raw text into useful and meaningful parts?

Ch. 5: Categorizing and Tagging Words

Sentence tokenizing = splitting text into sentences and storing them in an array.

```
>>> nok_sents =
sent_tokenizer.tokenize(nokia_raw)
```

```
>>> nok_sents[2]
'nokia warned group sales would grow
only \nbetween four and nine percent in
2002, after earlier forecasting \ngrowth
of 15 percent year-on-year, as a market
recovery takes longer\n than expected to
materialise.'
```

Word Tokenizing

```
>>> nltk.word_tokenize(nok_sents[2])
['nokia', 'warned', 'group', 'sales',
'would', 'grow', 'only', 'between',
'four', 'and', 'nine', 'percent',
'in', '2002', ',', 'after',
'earlier', 'forecasting', 'growth',
'of', '15', 'percent', 'year-on-
year', ',', 'as', 'a', 'market',
'recovery', 'takes', 'longer',
'than', 'expected', 'to',
'materialise', '.']
```

Penn Treebank POS Tags

Tag	Description
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

etc...

POS Tagging Using the Built-in Tagger

```
# Note _ means the last printed result (i.e. the
# word tokenized nok_sents[2]).

>>> nltk.pos_tag(_)
[('nokia', 'NN'), ('warned', 'VBD'), ('group',
'NN'), ('sales', 'NNS'), ('would', 'MD'), ('grow',
'VB'), ('only', 'RB'), ('between', 'IN'), ('four',
'CD'), ('and', 'CC'), ('nine', 'CD'), ('percent',
'NN'), ('in', 'IN'), ('2002', 'CD'), (',', ','),
('after', 'IN'), ('earlier', 'JJR'),
('forecasting', 'VBG'), ('growth', 'NN'), ('of',
'IN'), ('15', 'CD'), ('percent', 'NN'), ('year-on-
year', 'JJ'), (',', ','), ('as', 'IN'), ('a',
'DT'), ('market', 'NN'), ('recovery', 'NN'),
('takes', 'VBZ'), ('longer', 'NN'), ('than',
'IN'), ('expected', 'VBN'), ('to', 'TO'),
('materialise', 'VB'), ('.', '.')]
```

Say You Want to Build Your Own Tagger

```
#abbreviated
patterns_2 = [
    (r'.*ly$','RB'),                # adverbs
    (r'.*er$','JJR'),               # comparative adjective
    (r'.*est$','JJS'),              # superlative adjective
    (r'.*en$','VBN'),               # past participle
    # adjectives
    (r'(.*ble$|.*ful$|.*al$|.*ic$|.*ive$|.*less$|.*ous$|.*ish
    $|.*ent$|.*ar$)','JJ'),
    (r'.*', 'NN')                   # nouns (default)
    ]
```

How Well Did You Do?

```
>>> regexp_tagger = nltk.RegexpTagger(patterns_2) # Load the pattern
>>> regexp_tagger.tag(brown_sents[3]) # Parse a sentence
[('``', 'NN'), ('Only', 'RB'), ('a', 'NN'), ('relative', 'JJ'), ('handful', 'JJ'), ('of',
    'NN'), ('such', 'NN'), ('reports', 'NNS'), ('was', 'NNS'), ('received',
    'VBD'), ("''", 'NN'), (',', 'NN'), ('the', 'NN'), ('jury', 'NN'), ('said', 'NN'),
    (',', 'NN'), ('``', 'NN'), ('considering', 'VBG'), ('the', 'NN'),
    ('widespread', 'NN'), ('interest', 'JJS'), ('in', 'NN'), ('the', 'NN'),
    ('election', 'NN'), (',', 'NN'), ('the', 'NN'), ('number', 'JJR'), ('of', 'NN'),
    ('voters', 'NNS'), ('and', 'NN'), ('the', 'NN'), ('size', 'NN'), ('of', 'NN'),
    ('this', 'NNS'), ('city', 'NN'), ("''", 'NN'), ('.', 'NN')]
>>> regexp_tagger.evaluate(brown_tagged_sents) # Compare to gold
0.2074706128050596
# an improvement over the book's sample code:
0.2032639178948624 5
```

Parsing!

A Simple English Syntax:

```
S -> NP[PER=?p]  VP[PER=?p]
NP[PER=?p]  -> Det N[PER=?p]
VP[PER=?p] -> V[PER=?p] NP[PER=?p]
// ?p and ?q are bound variables
N[PER=3] -> 'cat' | 'bird'
V[PER=else] -> 'eat'
V[PER=3] -> 'eats' | 'loves'
Det -> 'The' | 'the'
```

Why?

- We want to train the computer to recognize good sentences from bad. This gets the computer a little closer to understanding the text. It will then compute all the parts of a sentence and how they work together – or should not work. We have similar intuitions when we hear a sentence like:
- The cat eat the bird.

Using Python to Abstract

We can make our own functions. This one allows us to condense five NLTK command line commands into one:

```
def parse2(x, y):
    x =  x.split()
    from nltk import load_parser
    cp = load_parser(y, trace=0)
    trees = cp.nbest_parse(x)
    for tree in trees: print tree
```

Good Sentences and Bad

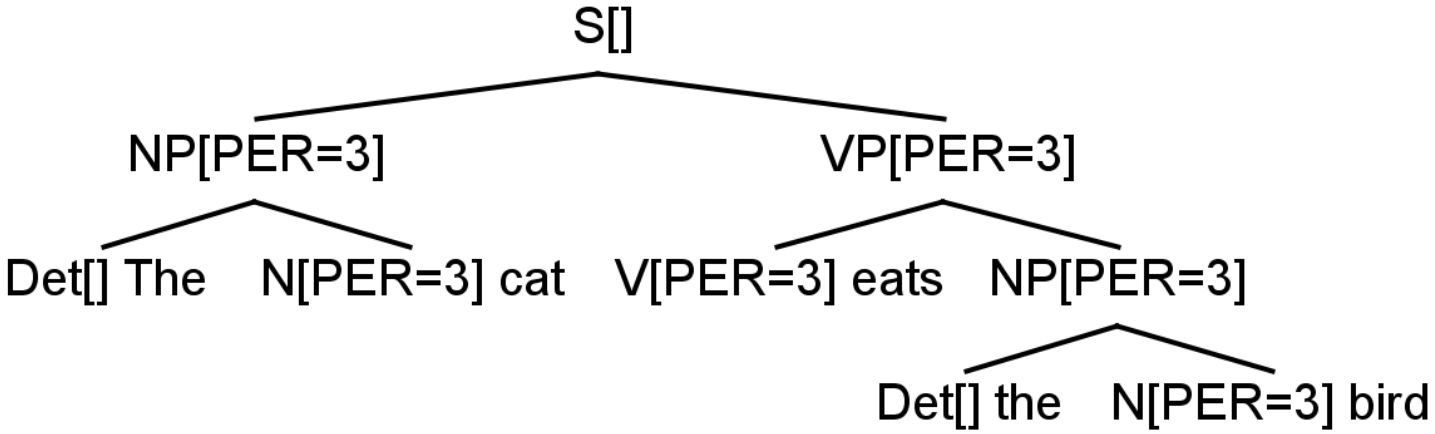
```
>>> import nltk
>>> import os
>>> os.getcwd()
'c:\\emacs-23.1\\bin\\NLTK\\clingHW6'
>>> import fbg
>>> eng = 'file:eng_fbg2_5.fcfg'
>>> fbg.parse2('The cat eats the bird', eng)
(S[]
  (NP[PER=3] (Det[] The) (N[PER=3] cat))
  (VP[PER=3] (V[PER=3] eats) (NP[PER=3] (Det[] the) (N[PER=3] bird))))
>>> fbg.parse2('The cat eat the bird', eng)
>>>
```

If the sentence is grammatical, NLTK produces a tree. If not, it doesn’t.

The Tree Looks Like This

“The cat eats the bird”

(S[]
  (NP[PER=3] (Det[] The) (N[PER=3] cat))
  (VP[PER=3] (V[PER=3] eats) (NP[PER=3] (Det[] the) (N[PER=3] bird))))



We Can’t Make Him Yet…

Maybe next time!

We have seen what some basic tools of NLP look like and how to get started with them.

The End