Lemmas:What's the Problem?

- It may get you better roots, than a stemmer (be for "am", "are", "is")
- Deals better with irregular plurals (eg woman and women)
- NB, can't itself recognise POS-based differences (fish and fish)

Lemmatisation

From Wikipedia, the free encyclopedia

Lemmatisation (or **lemmatization**) in linguistics, is the process of grouping together the different inflected forms of a word so they can be analysed as a single item.[1]

In computational linguistics, lemmatisation is the algorithmic process of determining the lemma for a given word. Since the process may involve complex tasks such as understanding context and determining the part of speech of a word in a sentence (requiring, for example, knowledge of the grammar of a language) it can be a hard task to implement a lemmatiser for a new language.

In many languages, words appear in several *inflected* forms. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks', 'walking'. The base form, 'walk', that one might look up in a dictionary, is called the *lemma* for the word. The combination of the base form with the part of speech is often called the *lexeme* of the word.

Lemmatisation is closely related to stemming. The difference is that a stemmer operates on a single word *without* knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

For instance:

1. The word "better" has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.
2. The word "walk" is the base form for word "walking", and hence this is matched in both stemming and lemmatisation.
3. The word "meeting" can be either the base form of a noun or a form of a verb ("to meet") depending on the context, e.g., "in our last meeting" or "We are meeting again tomorrow". Unlike stemming, lemmatisation can in principle select the appropriate lemma depending on the context.

Analysers like Lucene Snowball[2] store the base stemmed format of the word without the knowledge of meaning, but taking into account the semantics of the word formation only. The stemmed word itself might not be a valid word: 'lazy', as seen in the example below, is stemmed by many stemmers to 'lazi'. This is because the purpose of stemming is not to produce the appropriate lemma – that is a more challenging task that requires knowledge of context. The main purpose of stemming is to map different forms of a word to a single form,[3] and as a relatively simple, rules-based algorithm, it makes the above-mentioned sacrifice to ensure that, for example, when 'laziness' is stemmed to 'lazi', it has the same stem as 'lazy'.

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Data item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Español
Euskara
Français

Python 3.4.1 Shell

```
Python 3.4.1 (default, May 21 2014, 01:39:38)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> wn = nltk.WordNetLemmatizer()
>>> wn.lemmatize('women')
'woman'
>>> wn.lemmatize('women', 'v')
'women'
>>> wn.lemmatize('fishing', 'v')
'fish'
>>> wn.lemmatize('fishing', 'n')
'fishing'
>>> wn.lemmatize('is', 'v')
'be'
>>> wn.lemmatize('are', 'v')
'be'
>>> wn.lemmatize('am', 'v')
'be'
>>> n.lemmatize('am', 'n')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    n.lemmatize('am', 'n')
NameError: name 'n' is not defined
>>> wn.lemmatize('am', 'n')
'am'
>>>
```

Ln: 27 Col: 4

Text Pre-Processing

Parts of Speech (POS) & POS Tagging

Why POS Tag?

- We may also need to distinguish words that look the same but are from different syntactic categories (or parts of speech); fish *noun / verb*
- Lemmatising may need to know the part-of-speech already (noun or verb)
- Unfortunately, this may require parsing a whole sentence to disambiguate a POS and there are accuracy issues

POS Tagging Definition…

Part-of-speech tagging

From Wikipedia, the free encyclopedia

In corpus linguistics, **part-of-speech tagging** (**POS tagging** or **POST**), also called **grammatical tagging** or **word-category disambiguation**, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context—i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.

Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic. E. Brill's tagger, one of the first and most widely used English POS-taggers, employs rule-based algorithms.

Penn Part of Speech Tags

Note: these are the 'modified' tags used for Penn tree banking; these are the tags used in the Jet system. NP, NPS, PP, and PP$ from the original Penn part-of-speech tagging were changed to NNP, NNPS, PRP, and PRP$ to avoid clashes with standard syntactic categories.

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinatin
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adver
21.	RBR	Adverb, comparativ
22.	RBS	Adverb, superlati
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present part
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singula
32.	VBZ	Verb, 3rd person singular pr
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Accuracy?

Python 3.4.1 Shell

```
Python 3.4.1 (default, May 21 2014, 01:39:38)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on dar
win
Type "copyright", "credits" or "license()" for more informati
on.
>>> import nltk
>>> text = nltk.word_tokenize("The fish jumped over the man w
ho was fishing in the stream.")
>>> nltk.pos_tag(text)
[('The', 'DT'), ('fish', 'JJ'), ('jumped', 'VBD'), ('over', '
IN'), ('the', 'DT'), ('man', 'NN'), ('who', 'WP'), ('was', 'V
BD'), ('fishing', 'VBG'), ('in', 'IN'), ('the', 'DT'), ('stre
am', 'NN'), ('.', '.')]
>>> text2 = nltk.word_tokenize("The fish sang the tune")
>>> nltk.pos_tag(text2)
[('The', 'DT'), ('fish', 'JJ'), ('sang', 'NN'), ('the', 'DT')
, ('tune', 'NN')]
>>> text3 = nltk.word_tokenize("The man sings the song")
>>> nltk.pos_tag(text3)
[('The', 'DT'), ('man', 'NN'), ('sings', 'NNS'), ('the', 'DT'
), ('song', 'JJ')]
>>>
```

Ln: 14 Col: 4

POS tagging…

- There are a plethora of parsers that can be used to recover the syntactic structure of sentences
- They can be used in conjunction with lemmatisers or part of them to get more accurate word identifications

POS tagged tuples…

- There is a convention in Python to describe these tagged words in string tuples:
  - ‘fly/Vb’, ‘fly/NN’, ‘cheese/NN’
- So, you can write a sentence parse as a string that can be split and converted:

‘The/DT man/NN sings/VB the/DT song/JJ’

[('The', 'DT'), ('man', 'NN'), ('sings', 'VB'), ('the', 'DT'), ('song', 'JJ')]

Python 3.4.1 Shell

```
Python 3.4.1 (default, May 21 2014, 01:39:38)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on dar
win
Type "copyright", "credits" or "license()" for more informati
on.
>>> import nltk
>>> text = nltk.word_tokenize("The fish jumped over the man w
ho was fishing in the stream.")
>>> nltk.pos_tag(text)
[('The', 'DT'), ('fish', 'JJ'), ('jumped', 'VBD'), ('over', '
IN'), ('the', 'DT'), ('man', 'NN'), ('who', 'WP'), ('was', 'V
BD'), ('fishing', 'VBG'), ('in', 'IN'), ('the', 'DT'), ('stre
am', 'NN'), ('.', '.')]
>>> text2 = nltk.word_tokenize("The fish sang the tune")
>>> nltk.pos_tag(text2)
[('The', 'DT'), ('fish', 'JJ'), ('sang', 'NN'), ('the', 'DT')
, ('tune', 'NN')]
>>> text3 = nltk.word_tokenize("The man sings the song")
>>> nltk.pos_tag(text3)
[('The', 'DT'), ('man', 'NN'), ('sings', 'NNS'), ('the', 'DT'
), ('song', 'JJ')]
>>> tag_str = 'The/DT man/NN sings/VB the/DT song/JJ'
>>> [nltk.tag.str2tuple(t) for t in tag_str.split()]
[('The', 'DT'), ('man', 'NN'), ('sings', 'VB'), ('the', 'DT')
, ('song', 'JJ')]
>>> nltk.tag.str2tuple('man/NN')
('man', 'NN')
>>> tok = nltk.tag.str2tuple('man/NN')
>>> tok[1]
'NN'
>>> tok[0]
'man'
>>>
```

Why POS Tag?

REM

- We may also need to distinguish words that look the same but are from different syntactic categories (or parts of speech); fish *noun / verb*
- Lemmatizing may need to know the part-of-speech already (noun or verb)
- Unfortunately, this may require parsing a whole sentence to disambiguate a POS and there are accuracy issues

So, now we can lemmatise…

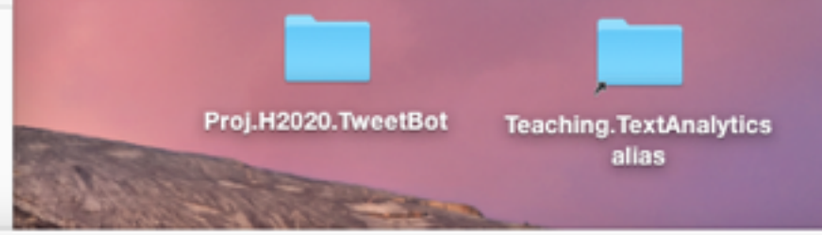
- POS tagging gives us an output we can submit to a lemmatiser
- However, note, the WordNet Lemmatiser generally works with simple tags (’n’, ‘v’, ‘adj’) so you need to convert the more complicated penn-tags to use it

WordNet Lemmatizer

Grab File Edit Capture Window Help 100% Wed 11:

Python 3.4.1 Shell

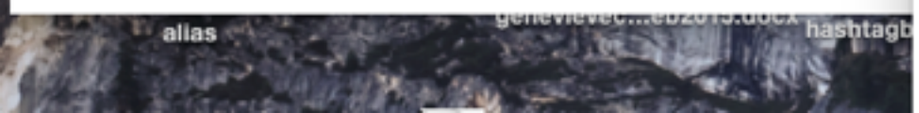
```
>>> ================================ RESTART ================================
>>>
[('The', 'DT'), ('fish', 'JJ'), ('who', 'WP'), ('jumped', 'VBD'), ('over', 'IN'),
('the', 'DT'), ('man', 'NN'), ('is', 'VBZ'), ('happy', 'JJ'), (',', ','), ('the',
'DT'), ('man', 'NN'), ('who', 'WP'), ('was', 'VBD'), ('fishing', 'VBG'), ('in', '
IN'), ('the', 'DT'), ('stream', 'NN')]
['The', 'n']
The
['fish', 'n']
fish
['who', 'n']
who
['jumped', 'v']
jump
['over', 'n']
over
['the', 'n']
the
['man', 'n']
man
['is', 'v']
be
['happy', 'n']
happy
[',', 'n']
,
['the', 'n']
the
['man', 'n']
man
['who', 'n']
who
['was', 'v']
be
['fishing', 'v']
fish
['in', 'n']
in
['the', 'n']
the
```



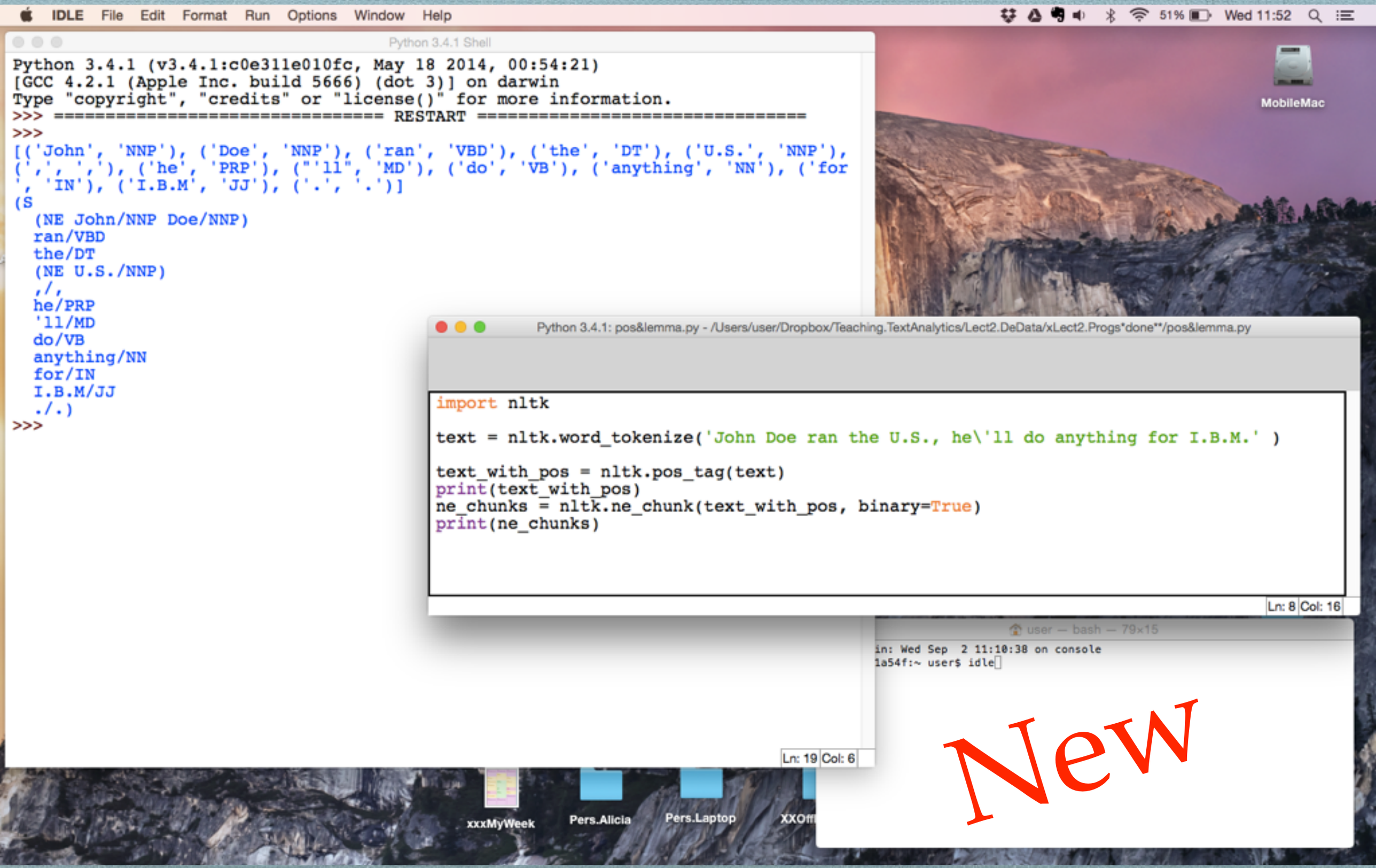
Python 3.4.1: pos&lemma.py - /Users/user/Dropbox/Teaching.TextAnalytics/Lect2.Bits.Data/Prac2.Basics.I.Data/pos&lemma.py

```
import nltk

text = nltk.word_tokenize('The fish who jumped over the man is happy, the man w

text_with_pos = nltk.pos_tag(text)
print(text_with_pos)

def convert_tags(tag):
        if tag == 'vbd' or tag == 'vbg' or tag == 'vbz':
                return 'v'
        else:
                return 'n'

wnl = nltk.WordNetLemmatizer()

for item in text_with_pos:
        new_tag = convert_tags(item[1].lower())
        print([item[0],new_tag])
        out = wnl.lemmatize(item[0], new_tag)
        print(out)
```

Old



Simpler Lemmatizer



Text Pre-Processing

Parsing to Syntax

Of course,

- In general, the whole point of doing POS tagging and lemmatisation is to get to the syntactic structure of the sentence
- When you have the syntactic structure you can really separate out which bits are important (and disambiguate)
- **nltk** allows you to define grammars and use them (e.g., CFG = context-free grammar)

Focus on Parsing…

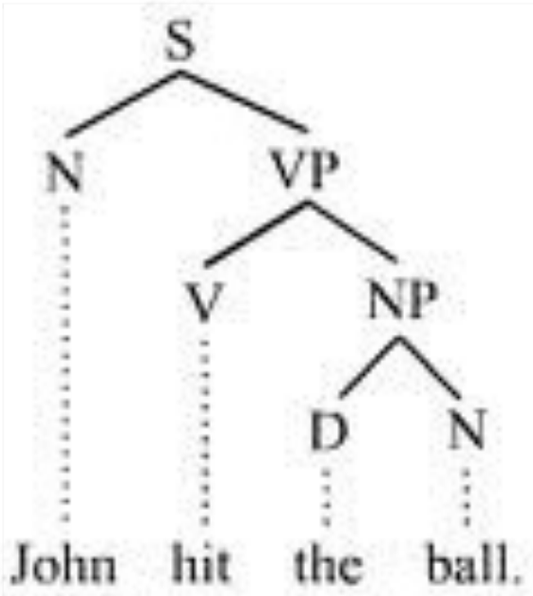
REM

Parse - Merriam-Webster Online
www.merriam-webster.com/dictionary/**parse**
grammar : to divide (a sentence) into grammatical parts and identify the parts and their relations to each other. : to study (something) by looking at its parts ...

Parsing

Programming Language

Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. The term parsing comes from Latin pars, meaning part. Wikipedia

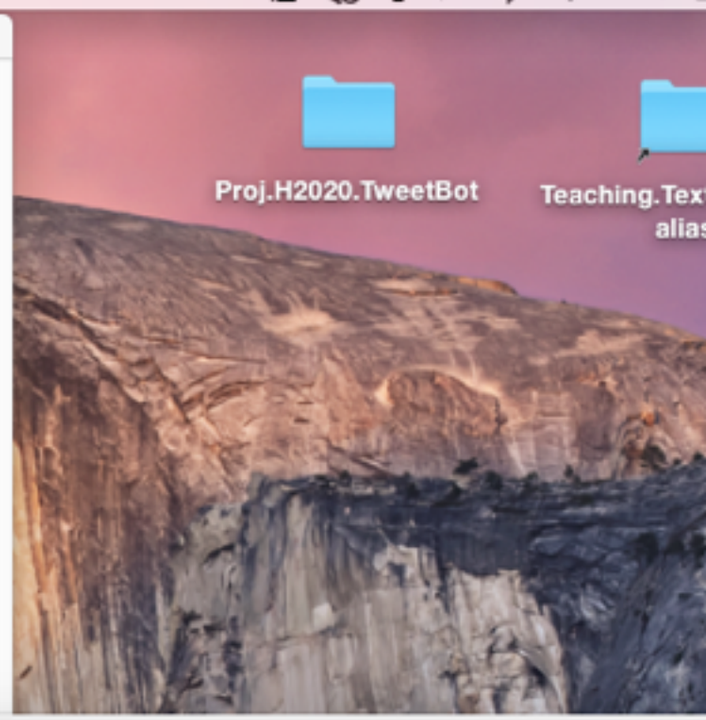


Constituency-based parse tree

IDLE File Edit Format Run Options Window Help

Python 3.4.1 Shell

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant)))
    (PP (P in) (NP (Det my) (N pajamas)))))
(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
>>>
```

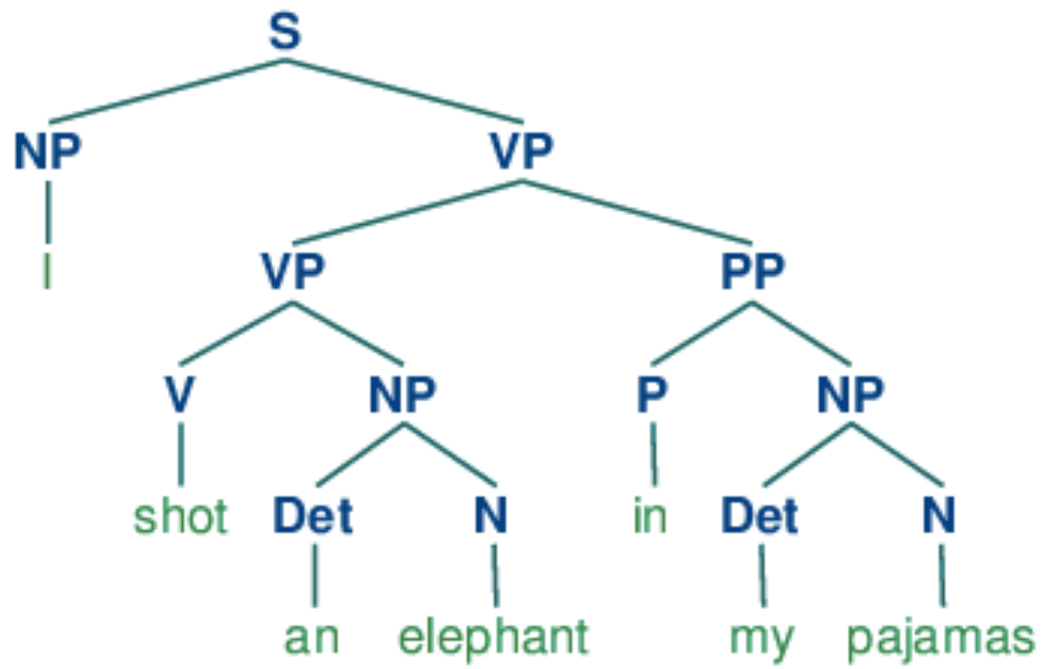


Python 3.4.1: parser.py - /Users/user/Dropbox/Teaching.TextAnalytics/Lect2.Bits.Data/Prac2.Basics.I

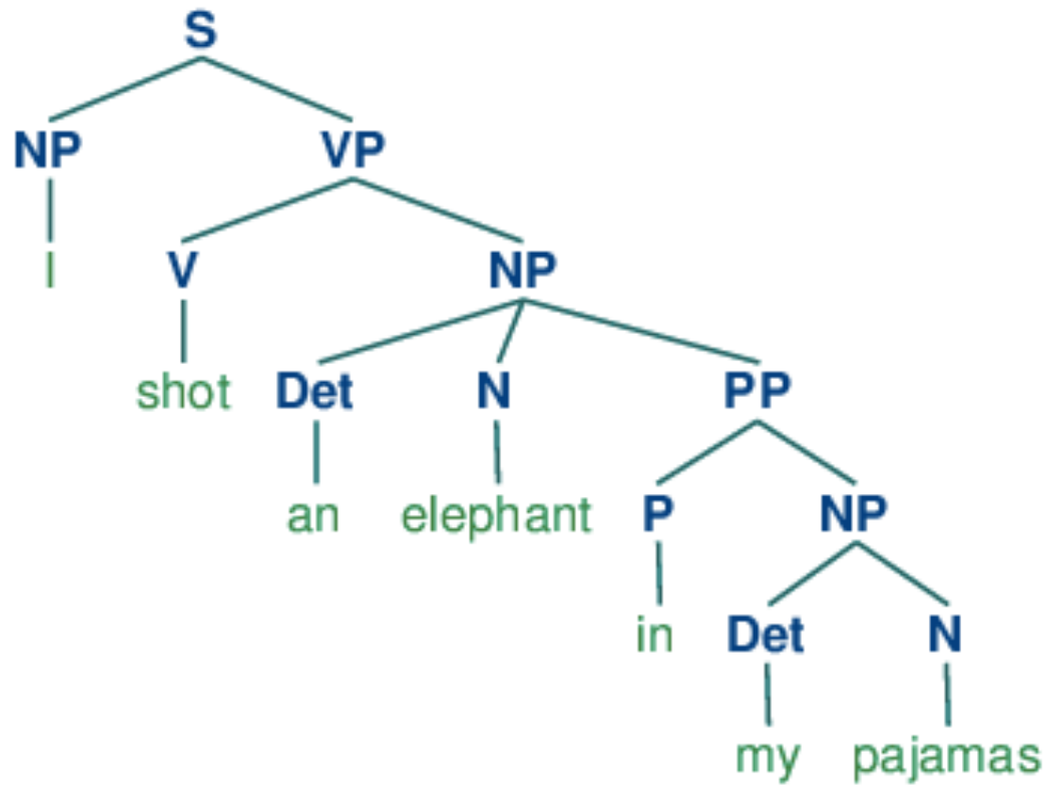
```
import nltk

groucho_grammar = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'an' | 'my'
N -> 'elephant' | 'pajamas'
V -> 'shot'
P -> 'in'
""")

sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
parser = nltk.ChartParser(groucho_grammar)
for tree in parser.parse(sent):
    print(tree)
```

http://www.nltk.org/book/ch08.html

a.



b.



http://www.nltk.org/book/ch08.html

Stanford Parser

- The Stanford Parser is very commonly used to perform (better) parses of sentences
- Note, probabilistic parsers are often used because there are many alternative parses
- It is in Java but can be run from python wrappers (but, that is a story for another day)

Text Pre-Processing

Spotting Entities

Our Focus…

REM

- Text pre-processing is the poor-farmer cousin of full NLP; its not really about meaning
- Its about cleaning up text-data for future use
- Uses ideas from NLP (eg syntactic analysis, parsing) … but is not often full NLP
- Ultimately, it seldom recovers meaning

Standard Tasks

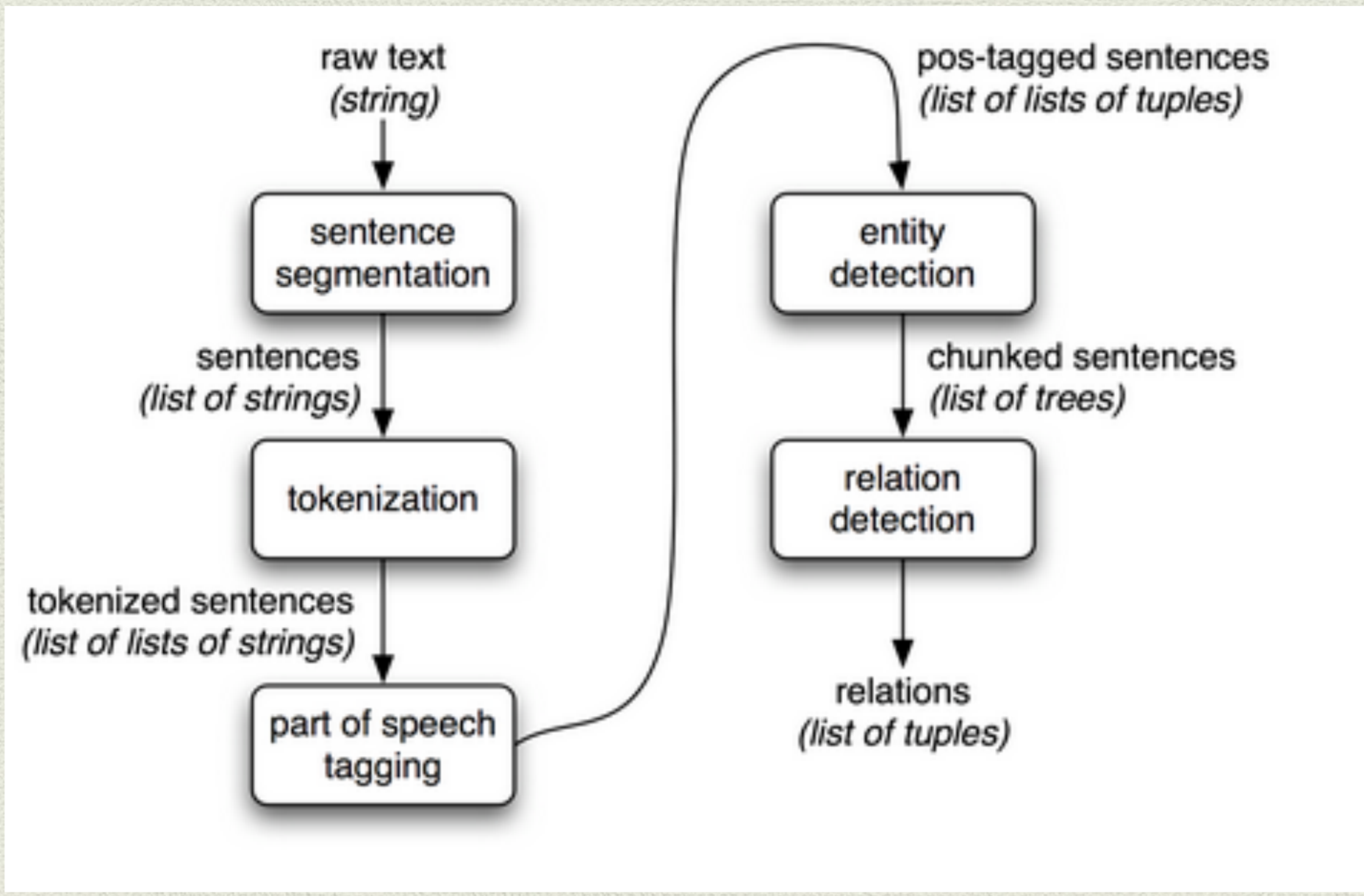
REM

- *Tokenisation & Normalisation*: finding boundaries between word-like entities in character string
- *Fixing Misspellings*: where possible
- *Stemming, lemmatisation, POS-tagging*: finding slightly deeper identities between words (fished, fishing)
- *Removing Stop Words:* maximising the content-full words in the document/corpus
- *Entity Extraction:* identifying conceptual entities behind words

Entities

- Entity extraction is the most semantic aspect of pre-processing (as such, often not done)
- Here, you identify the actual conceptual entity referred to by the word; encyclopaedic rather than dictionary knowledge
- I.B.M => "I.B.M" => IBM the organization

Typical Pipeline



Simple EG

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
[('John', 'NNP'), ('Doe', 'NNP'), ('ran', 'VBD'), ('the', 'DT'), ('U.S.', 'NNP'),
(',', ','), ('he', 'PRP'), ("'ll", 'MD'), ('do', 'VB'), ('anything', 'NN'), ('for
', 'IN'), ('I.B.M', 'JJ'), ('.', '.')]
(S
  (NE John/NNP Doe/NNP)
  ran/VBD
  the/DT
  (NE U.S./NNP)
  ,/,
  he/PRP
  'll/MD
  do/VB
  anything/NN
  for/IN
  I.B.M/JJ
  ./.)
>>>
```

```
import nltk

text = nltk.word_tokenize('John Doe ran the U.S., he\'ll do anything for I.B.M.'

text_with_pos = nltk.pos_tag(text)
print(text_with_pos)
ne_chunks = nltk.ne_chunk(text_with_pos, binary=True)
print(ne_chunks)
```

Entity Extractors

- There are many different Entity Recognisers with different levels of accuracy for particular texts
- Stanford NER (Named Entity Recognizer) is big
- Open Calais is also heavily used
- But, need good reasons to go this far in your pre-processing, esp. when you consider accuracy

Text Pre-Processing

Removing Stop Words

Why Remove Stops?

- We have been mainly transforming words in various ways to make them better for use…

- But, some words do not help, like stop words
- They are words that do not convey much content, they are not *contentful*
- They are also often very frequent and can effects norms and counting (cf Lect4)

Stops Words…(lucene)

"a", "an", "and", "are", "as", "at", "be", "but", “by", "for", "if", "in", "into", "is", “it", "no", "not", "of", "on", "or", “such", "that", "the", "their", "then", "there", “these", "they", "this", "to", "was", "will", "with"

Another set of stop-words

a,able,about,across,after,all,almost,also,am,among,an,and,any,are,as,at,be,because,been,but,by,can,cannot,could,dear,did,do,does,either,else,ever,every,for,from,get,got,had,has,have,he,her,hers,him,his,how,however,i,if,in,into,is,it,its,just,least,let,like,likely,may,me,might,most,must,my,neither,no,nor,not,of,off,often,on,only,or,other,our,own,rather,said,say,says,she,should,since,so,some,than,that,the,their,them,then,there,these,they,this,tis,to,too,twas,us,wants,was,we,were,what,when,where,which,while,who,whom,why,will,with,would,yet,you,your

Stop Word Lists…

- There is no definitive stop-word set, may vary for different purposes (see wikipedia for some egs)
- They can result in large reductions (40%-60%) in normal texts, so you are left with core content…

Python 3.4.1 Shell

```
Python 3.4.1 (default, May 21 2014, 01:39:38)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> from nltk.corpus import stopwords
>>> stop = stopwords.words('english')
>>> stop
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'you
rs', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers
', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselv
es', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is'
, 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because'
, 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to',
'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'furthe
r', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not
', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', '
just', 'don', 'should', 'now']
>>> text = open('/Users/user/Desktop/text.txt')
>>> rawtext = text.read()
>>> [i for i in rawtext.split() if i not in stop]
['So,', 'bunch', 'text', "i've", 'put', 'togethr', 'check', 'tokenisation', 'crap
', 'I', 'interested', 'I.B.M.', 'U.S.A.', 'USA', 'handled,', 'ok?', 'This', 'whet
her', 'properly', 'deals', 'websites', 'like', 'www.ucd.ie', 'email', 'addresses'
, 'like', 'mark.keane@ucd.email.ie.', 'Also', 'weirdities', 'like', 'great', "O'N
eill", 'like', 'M*A*S*H,', 'maybe', '7', 'tokens?']
>>> rawtext
"So, this is just a bunch of text that i've put togethr to check on\nthis tokenis
ation crap and I am interested in how I.B.M. or the U.S.A. or the USA\nis handled
, ok? This and whether it properly deals with websites like www.ucd.ie\nand email
addresses like mark.keane@ucd.email.ie.  Also other weirdities like\nthe great O'
Neill and like M*A*S*H, which should be maybe 7 tokens?\n"
>>>
```

Pre-Processing

When to Use What & When…

When to use these things…

- Pre-processing is an important design choice: sometimes you might only do stemming and stop-word removal; some times lemmatisation and stop-word removal

- Other times, you want to leave everything in and do full parsing of the text; or parsing and then stop-word removal later

Pre-Processing for Indexing…

- Indexing and retrieval tend to use quite crude stemming and stop-word removal (cf Lucene)

- Here you want a small no of string-features to capture your docs and queries; can increase recall without damaging precision

- Does not matter if they are crap (lying -> li) because they are consistent over the doc set and the scale of the corpus irons out inaccuracies

Pre-Processing for Meaning…

- Stemmers do not often get at morphological root; so, less good if you need more meaning
- Cases needing meaning invite lemmatisation (POS and stop-word removal)
- Eg if you want definite verbs and nouns and other POS (cf Gerow & Keane, 2011)

Pre-Processsing for Tweets…

- In Twitter, eg, everything is different:
  - Twitter-specific pos-taggers
  - Stop-word removal can damage performance
  - Misspellings and abbreviations need specific treatment

Some Twitter Refs…

Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., ... & Smith, N. A. (2011, June). Part-of-speech tagging for twitter. In COLING: Volume 2 (pp. 42-47). ACL

Kouloumpis, E., Wilson, T., & Moore, J. (2011). Twitter sentiment analysis: The good the bad and the omg!. ICWSM, 11, 538-541.

Aiello, L. M., Petkos, G., Martin, C., Corney, D., Papadopoulos, S., Skraba, R., ... & Jaimes, A. (2013). Sensing trending topics in Twitter. IEEE Trans Multimedia.

Handling Different Sources…

Different file formats…

- Text files (we've seen)
- Web pages, html and xml
- PDFs and Docs

We have seen text.file input

Grab File Edit Capture Window Help

21% Tue 15:43

Python 3.4.1 Shell

```
Python 3.4.1 (default, May 21 2014, 01:39:38)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> tfile = open('/Users/user/Desktop/text.txt')
>>> rawtext = tfile.read()
>>> tokens = nltk.word_tokenize(rawtext)
>>> tokens
['So', ',', 'this', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', 'i', "'ve",
'put', 'togethr', 'to', 'check', 'on', 'this', 'tokenisation', 'crap', 'and', 'I'
, 'am', 'interested', 'in', 'how', 'I.B.M', '.', 'or', 'the', 'U.S.A.', 'is', 'ha
ndled', ',', 'ok', '?', 'And', 'whether', 'it', 'properly', 'deals', 'with', 'web
sites', 'like', 'www.ucd.ie', 'and', 'email', 'addresses', 'like', 'mark.keane',
'@', 'ucd.email.ie', '.', 'Also', 'other', 'weirdities', 'like', 'the', 'great',
"o'Neill", 'and', 'like', 'M*A*S*H', ',', 'which', 'should', 'be', 'maybe', '7',
'tokens', '?']
>>>
```

Ln: 10 Col: 4

Python 3.4.1: text.txt - /Users/user/Desktop/text.txt

```
So, this is just a bunch of text that i've put togethr to check on
this tokenisation crap and I am interested in how I.B.M. or the U.S.A.
is handled, ok? And whether it properly deals with websites like www.ucd.
and email addresses like mark.keane@ucd.email.ie.  Also other weirdities
the great o'Neill and like M*A*S*H, which should be maybe 7 tokens?
```

Ln: 1 Col: 50

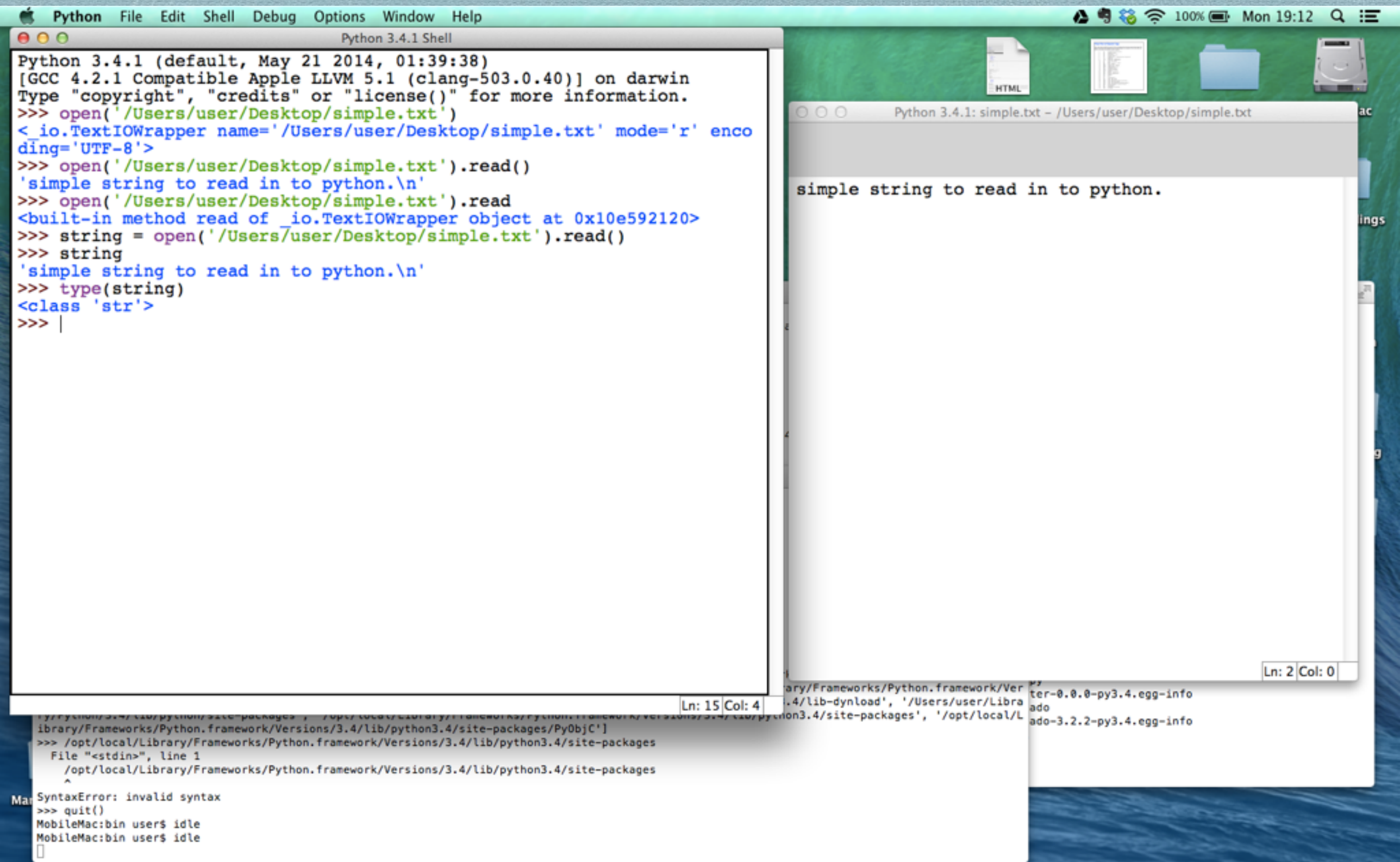
user — Python — 81×14

```
MobileMac:~ user$ idle
```

ActiveStuff
MobileMac
Proj.RProgramming
X_All_Readings
X_Research
X_Teaching
X Admin
X Odds

XXX_insight-snow1...nal.pdf
xxxx1403.7400v1.pdf
XX_castillo_elhaddad_p...cle.pdf
xx_kdd12-trimine.pdf

We have seen text.file input



We have seen text.file input

- You basically open the file; open()
- Read it in: read()
- Then you have a string object you can manipulated in different ways

Web Pages are trickier…

- You basically open the file with a special method for opening ; openurl.request.url()
- Read it in: read()
- Then you need a new package (BeautifulSoup) to create an new object you can extract different bits of the page from

Installation steps…

- NB. Stuff in book is legacy; clean_html and urlopen do not work as specified
- Install correct version of BeautifulSoup for Python3.4 (aaaagghhh…called bs4 !)
  http://www.crummy.com/software/BeautifulSoup/
- The import it and use its given commands

Reading Webpages…

Python 3.4.1 Shell

```
Python 3.4.1 (default, May 21 2014, 01:39:38)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> import urllib
>>> url = 'http://www.csi.ucd.ie/users/mark-keane'
>>> urllib.request.urlopen(url)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    urllib.request.urlopen(url)
AttributeError: 'module' object has no attribute 'request'
>>> import urllib.request
>>> urllib.request.urlopen(url)
<http.client.HTTPResponse object at 0x10fcf46a0>
>>> rawhtml = urllib.request.urlopen(url).read()
>>> from bs4 import BeautifulSoup
>>> soup = bs4.BeautifulSoup(rawhtml)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    soup = bs4.BeautifulSoup(rawhtml)
NameError: name 'bs4' is not defined
>>> import bs4
>>> soup = bs4.BeautifulSoup(rawhtml)
>>> soup.title
<title>Mark Keane | UCD School of Computer Science and Informatics</title>
>>> soup.body.get_text(strip=True)
'UCD School of Computer Science and InformaticsScoil na Ríomheolaíochta agus na F
aisnéisíochta UCDCalendarNewsPeopleSite mapSign InYou are here:Home›CSI People›Ma
rk KeaneMark KeaneBiographyResearch Interests:AnalogyCognitive ScienceEvolutionSI
milarityText AnalyticsGeneralName and Title:Professor Mark Keane BA MA PhDPositio
n:Chair of Computer SciencePhone:Ext. 2470Email:Office:CSI / B2.01Address:School
of Computer Science & InformaticsBiographyProfessionalPublicationsResearchBiograp
hySince 1998, Prof. Mark Keane has been Chair of Computer Science at University C
```

Parsing Webpages…

```
>>> foolinks = soup.findAll('a')
>>> for link in foolinks: print(link)

<a id="navigation-top" name="top"></a>
<a href="/" rel="home" title="Home"><img alt="Home" id="logo-image" src="/sites/all/themes/zen/ucd_inspire/logo.png"/></a>
<a href="/" rel="home" title="Home">
        UCD School of Computer Science and Informatics          </a>
<a class="menu-1-1-2" href="/calendar">Calendar</a>
<a class="menu-1-2-2" href="/news">News</a>
<a class="menu-1-3-2" href="/content/csi-people">People</a>
<a class="menu-1-4-2" href="/sitemap" title="Display a site map with RSS feeds.">Site map</a>
<a class="menu-1-5-2" href="/user">Sign In</a>
<a href="/">Home</a>
<a href="/users">CSI People</a>
<a class="taxonomy_term_346" href="/category/research-interests/analogy" rel="tag" title="">Analogy</a>
<a class="taxonomy_term_187" href="/category/research-interests/cognitive-science" rel="tag" title="">Cognitive Science</a>
<a class="taxonomy_term_504" href="/category/research-interests/evolution" rel="tag" title="">Evolution</a>
<a class="taxonomy_term_378" href="/category/research-interests/similarity" rel="tag" title="">SImilarity</a>
<a class="taxonomy_term_948" href="/category/research-interests/text-analytics" rel="tag" title="">Text Analytics</a>
<a href="#tabs-tabset-1">Biography</a>
<a href="#tabs-tabset-2">Professional</a>
<a href="#tabs-tabset-3">Publications</a>
<a href="#tabs-tabset-4">Research</a>
<a href="https://rms.ucd.ie/ufrs/W_RMS_PUB_COMMON.PUB_POPUP?p_object_id=134951287" target="_blank"> [Details]</a>
<a href="https://rms.ucd.ie/ufrs/W_RMS_PUB_COMMON.PUB_POPUP?p_object_id=88772699" target="_blank"> [Details]</a>
<a href="https://rms.ucd.ie/ufrs/W_RMS_PUB_COMMON.PUB_POPUP?p_object_id=285842122" target="_blank"> [Details]</a>
<a href="https://rms.ucd.ie/ufrs/W_RMS_PUB_COMMON.PUB_POPUP?p_object_id=88772700" target="_blank"> [Details]</a>
<a href="https://rms.ucd.ie/ufrs/W_RMS_PUB_COMMON.PUB_POPUP?p_object_id=134951303" target="_blank"> [Details]</a>
<a href="https://rms.ucd.ie/ufrs/W_RMS_PUB_COMMON.PUB_POPUP?p_object_id=134951297" target="_blank"> [Details]</a>
```

PDFs & .Docs…

- General advice is to convert them to text and work from there…
- But PDFminer is a python package for parsing PDFs (a bit complicated)

Selecting a Corpus

Finally, we have assumed…

- That you just know which texts to pre-process; but, sometimes you have to think about selecting the texts that make up a corpus

- Is this defined naturally; every debate in the Dail since 1922…(simple case)

- Every news article about stock markets…how do we define this ? (medium case)

- Every tweet that is about senate elections …how do we define this ? (hard case)

Please Go Home Now…

- At least, if you have finished the practical…