De Data:

The One Where We Do De Data

Lecture 2: Text Analytics for Big Data

Mark Keane, Insight/CSI, UCD

Selling Things

stock-markets

social media

science

news

polls

sentiment-id

sentiment-use

time-series

summaries

VSMs

Classifiers

Clustering

cosine

jaccard

dice

levenschtein

TF-IDF

LLR

PMI

Entropy

simple frequencies

*** pre-processed text items of some sort… ***

The Basic View

- Well, the course is called Text Analytics
- One expects the *data* to be text, words, clauses, sentences, paragraphs, and other things written in ***natural language***
- Here, we consider this data and ***pre-processing*** "raw" text for future use

What's the Problem?

"It was in the winter of '69 that we encountered the first sign that the U.S.A. was entering into a conflict, streked (sic) by the GOP, against some unspecified Evil. Mother feared the worst for future fish catches. The Coalition of Baddies was afraid of war and fishing immediately ceased in the South China Sea."

What's the Problem?

"It was in the winter of '69 that we encountered the first sign that the U.S.A. was entering into a conflict, streked (sic) by the GOP, against some unspecified Evil. Mother feared the worst for future fish catches. The Coalition of Baddies was afraid of war and fishing immediately ceased in the South China Sea."

What are these ?

What's the Problem?

"It was in the winter of '69 that we encountered the first sign that the U.S.A. was entering into a conflict, streked (sic) by the GOP, against some unspecified Evil. Mother feared the worst for future fish catches. The Coalition of Baddies was afraid of war and fishing immediately ceased in the South China Sea."

This is really "stoked"

What's the Problem?

"It was in the winter of '69 that we encountered the first sign that the U.S.A. was entering into a conflict, streked (sic) by the GOP, against some unspecified Evil. Mother feared the worst for future fish catches. The Coalition of Baddies was afraid of war and fishing immediately ceased in the South China Sea."

"feared" & "afraid" are sort of the same !

What's the Problem?

"It was in the winter of '69 that we encountered the first sign that the U.S.A. was entering into a conflict, streked (sic) by the GOP, against some unspecified Evil. Mother feared the worst for future fish catches. The Coalition of Baddies was afraid of war and fishing immediately ceased in the South China Sea."

"fish" & "fishing" look the same but are different !

Not all words are equal…

"winter sign U.S.A. conflict GOP Evil Mother feared fish catches Coalition Baddies afraid war fishing ceased South China Sea"

find the content-bearing words

Will end with…

"winter encounter sign usa enter conflict stok gop region east asia mother fear future fish catch coalition bad fear afraid fish cease south china sea."

Pre-processing does this; all words are modified, stemmed, dropped and the output is the raw data ("words") used in the analysis

Overview

- Basics of Natural Language Processing (NLP)
- Pre-processing to modify text:
  - tokenisation, stemming, POS tagging, lemmatisation, fixing spellings
- Pre-processing to exclude (some) text
  - removing stop words
- When pre-processing helps or not ?

Important Point

- Pre-processing is not just about taking things out; stripping off stems, removing stops etc…
- It may also be about putting things in; like POS tags, syntax, entity tags, lexical chains

NLP 101

I'll -ology you

- NLP is about meaning, parsing stops short:
  - *Phonology & Morphology*: sounds & words
  - *Syntax*: traditionally syntactic constituents
  - *Meaning*: Semantics & Pragmatics

I'll -ology you.

- Natural Language Processing concerns itself with recovering meaning from spoken sounds or written strings, usually using different levels of analysis:
  - Phonology (speech alone)
  - Morphology (speech & writing)
  - Syntax (speech & writing)
  - Semantics (speech & writing)
  - Pragmatics (speech & writing)

I'll -ology you..

- *Phonology*: analysis of sounds into phonemes, sets of which form words
- *Morphology*: analysis of sounds/signs into morphemes (smallest grammatical unit of a language) not= word, morphemes may or may not stand alone ("dog" & "s" versus "dog" & "dogs")
- *Syntax*: analysis of rules of combination of grammatical units of a language to form sentences

I'll -ology you…

- *Semantics*: analysis of how the signs of a language come to convey its meaning; the study of meaning at the levels of words, phrases, sentences, and larger units of discourse (termed texts or narratives)

- *Pragmatics:* analysis of how context contributes to meaning (*yeah…yeah*)

Lemma (morphology)

From Wikipedia, the free encyclopedia

In morphology and lexicography, a **lemma** (plural *lemmas* or *lemmata*) is the **canonical form**, **dictionary form**, or **citation form** of a set of words (headword)[*citation needed*]. In English, for example, *run*, *runs*, *ran* and *running* are forms of the same lexeme, with *run* as the lemma. *Lexeme*, in this context, refers to the set of all the forms that have the same meaning, and *lemma* refers to the particular form that is chosen by convention to represent the lexeme. In lexicography, this unit is usually also the *citation form* or headword by which it is indexed. Lemmas have special significance in highly inflected languages such as Turkish and Czech. The process of determining the *lemma* for a given word is called lemmatisation. The lemma can be viewed as the chief of the principal parts, although lemmatisation is at least partly arbitrary.

- Is on aspects of morphology and syntax to find…

- …parts of speech for words (fish is a noun, fishing is a verb, fishy is an adjective)

- …roots and/or lemmas of words (the root fish~ from fishing, fishes, fished; the root be~ from am, are, is)

http://en.wikipedia.org/wiki/Lemma_(morphology)

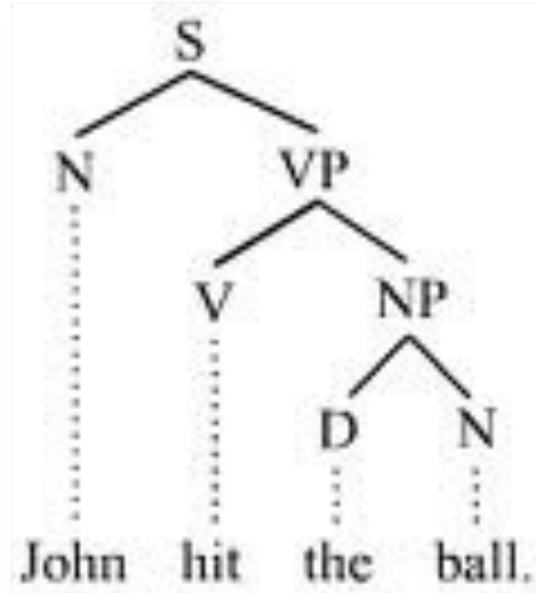
Focus on Parsing…

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

Text Pre-Processing

Pre-Processing: De Guts

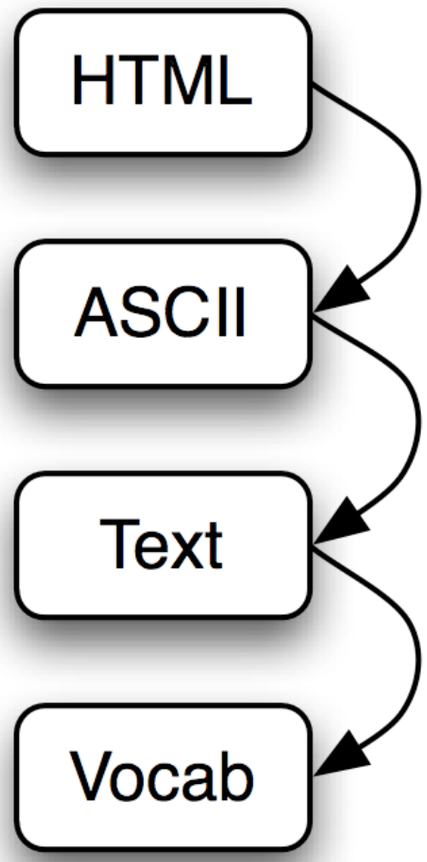
Our Focus…

- 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

- *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

Typical Python Pipeline




```
html = urlopen(url).read()
raw = nltk.clean_html(html)
raw = raw[750:23506]
```

Download web page, strip HTML if necessary, trim to desired content

```
tokens = nltk.wordpunct_tokenize(raw)
tokens = tokens[20:1834]
text = nltk.Text(tokens)
```

Tokenize the text, select tokens of interest, create an NLTK text

```
words = [w.lower() for w in text]
vocab = sorted(set(words))
```

Normalize the words, build the vocabulary

But…First Remember

- *Document encodings*: byte sequence turned into a character sequence, ok if ASCII but are different encoding schemes (Unicode UTF-8); need to identify (meta-data) and decode

- *Document formats*: DOC, PDF, XML, HTML may all need own converters to ensure characters are identified

- *Document text*: text-part of particular doc needs to be extracted (e.g., from XML, often not clean for sites)

Text Pre-Processing

Tokenisation & Normalisation

What is the Problem ?

- We have some sequence of characters, we need to break into word-like tokens/entities
- No problemo, marko, you pick those things with space-chars around them or fullstops before them…
- Right?

Why Tokenise?

- Well, what about:

I've: 'I' 've' / 'I've' / Ive

U.S.A. : 'U' 'S' 'A' / 'USA' / 'U.S.A.'

Jonny O'Neill: 'Jonny' 'O' 'Neill' / 'Jonny' 'O'Neill'

www.ucd.ie: 'www' 'ucd' 'ie' / 'www.ucd.ie' / 'www' 'ucd.ie'

mark.keane@ucd.ie: 'mark' 'keane' '@' 'ucd' 'ie' / what?

- Which one you pick has impact on the text data

Tokenising I

- Tokenisers work off general rules and a lot of really quite specific ones too
- Consider the following text file:

text.txt - /Users/user/Dropbox/Teaching.TextAnalytics/Lect2.DeData/Prac2.Basics.I.Data*test**/text.txt (3.4.3)

```
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. or the USA
is handled, ok? This and whether it properly deals with websites 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: 1 Col: 0

Python 3.4.5 Shell

```
Python 3.4.5 (default, Jun 27 2016, 04:57:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> import nltk
>>> tfile = open('/Users/user/Desktop/text.txt')
>>> rawtext = tfile.read()
>>> rawtext
"So, this is just a bunch of text that i've put togethr to check on\nthis tokeni
sation crap and I am interested in how I.B.M. or the U.S.A. or the USA\nis handl
ed, ok? This and whether it properly deals with websites like www.ucd.ie\nand em
ail addresses like mark.keane@ucd.email.ie.  Also other weirdities like\nthe gre
at O'Neill and like M*A*S*H, which should be maybe 7 tokens?\n"
>>> 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.', 'or',
 'the', 'USA', 'is', 'handled', ',', 'ok', '?', 'This', 'and', 'whether', 'it',
'properly', 'deals', 'with', 'websites', 'like', 'www.ucd.ie', 'and', 'email', '
addresses', 'like', 'mark.keane', '@', 'ucd.email.ie', '.', 'Also', 'other', 'we
irdities', 'like', 'the', 'great', "O'Neill", 'and', 'like', 'M*A*S*H', ',', 'wh
ich', 'should', 'be', 'maybe', '7', 'tokens', '?']
>>>
```

Ln: 12 Col: 4

text.txt - /Users/user/Desktop/text.txt (3.4.5)

```
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. or the
is handled, ok? This and whether it properly deals with websites like www.ucd.
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: 1 Col: 0

share — Python • idle — 124×13

```
    main()
  File "/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/idlelib/PyShell.py", line 1611, in main
    root.mainloop()
  File "/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/tkinter/__init__.py", line 1125, in mainlo
op
    self.tk.mainloop(n)
  File "/opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/tkinter/__init__.py", line 1533, in __call
__
    def __call__(self, *args):
KeyboardInterrupt
MacBook-Air-4:share user$ idle
MacBook-Air-4:share user$ idle
```

Tokenising III

- So, the **nltk** Python tokeniser obviously works off a set of rules for handling things of different types
- Some rules are quite general (eg handling fullstops)
- Other rules are very specific (M*A*S*H)
- **nltk** makes it easy for you to extend these rules; using the **re** package (for regular expressions)

Some Downsides…

- There are other problems to handle in English, like hyphenation, foreign phrases (*au fait*), numbers, acronyms, etc…
- Problem: solution is quite language-dependent, may need different solutions in:
  - German: where they compound a lot
  - Chinese/Korean: where clear word boundaries are not shown by spaces

Why Normalise ?

- Is often done to reduce of minor differences between tokens, like:
  - accented differences (cliche, cliché)
  - capitals (the, The)
  - acronyms (U.S.A., USA)
- Though there may be more complex solutions involving *equivalence classes*

http://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

nltk Normalisation = downcase

Grab File Edit Capture Window Help — 67% Tue 16:55

Python 3.4.1 Shell

```
"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"
>>> 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.', 'or', 'th
e', 'USA', 'is', 'handled', ',', 'ok', '?', 'This', 'and', 'whether', 'it', 'prop
erly', 'deals', 'with', 'websites', 'like', 'www.ucd.ie', 'and', 'email', 'addres
ses', 'like', 'mark.keane', '@', 'ucd.email.ie', '.', 'Also', 'other', 'weirditie
s', 'like', 'the', 'great', "O'Neill", 'and', 'like', 'M*A*S*H', ',', 'which', 's
hould', 'be', 'maybe', '7', 'tokens', '?']
>>> words = [w.lower() for w in tokens]
>>> words
['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.', 'or', 'th
e', 'usa', 'is', 'handled', ',', 'ok', '?', 'this', 'and', 'whether', 'it', 'prop
erly', 'deals', 'with', 'websites', 'like', 'www.ucd.ie', 'and', 'email', 'addres
ses', 'like', 'mark.keane', '@', 'ucd.email.ie', '.', 'also', 'other', 'weirditie
s', 'like', 'the', 'great', "o'neill", 'and', 'like', 'm*a*s*h', ',', 'which', 's
hould', 'be', 'maybe', '7', 'tokens', '?']
```

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. or the USA
is handled, ok? This and whether it properly deals with websites 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: 1 Col: 0

Post Adverts
Proj.GroundTruth
Proj.Python
Teaching.TextAnalytics
url.jpg
_Flow.gif
sd
ActiveStuff
MobileMac
Proj.RProgramming
X_All_Readings
tokens
X_Research
file
X_Teaching
TXT
text.txt
X Admin
ryanneDoyle
X Odds
PYTHON
— 81×14
XXX_insight-
xxxx1403.7400v
XX_castillo_elha
xx_kdd12-

Text Pre-Processing

Misspelling

Why Fix Spellings ?

- Obviously, we cannot assume all the words are spelt correctly; books best, tweets worst

- So, after tokenizing and normalisation, one could run a spelling checker over the tokens

- Standard solution is to take a dictionary, compare each word against it, if no entry found, use shortest edit distance to a word in dict

How to Fix Spellings

- Python has a number of different packages that do spelling correction and front-ends that will highlight misspellings in text
- the **enchant** package is one of most popular
- So, Mac Install Enchant (may be in spyder)

```
$ sudo port install py34-enchant
```

Spellings: enchant module

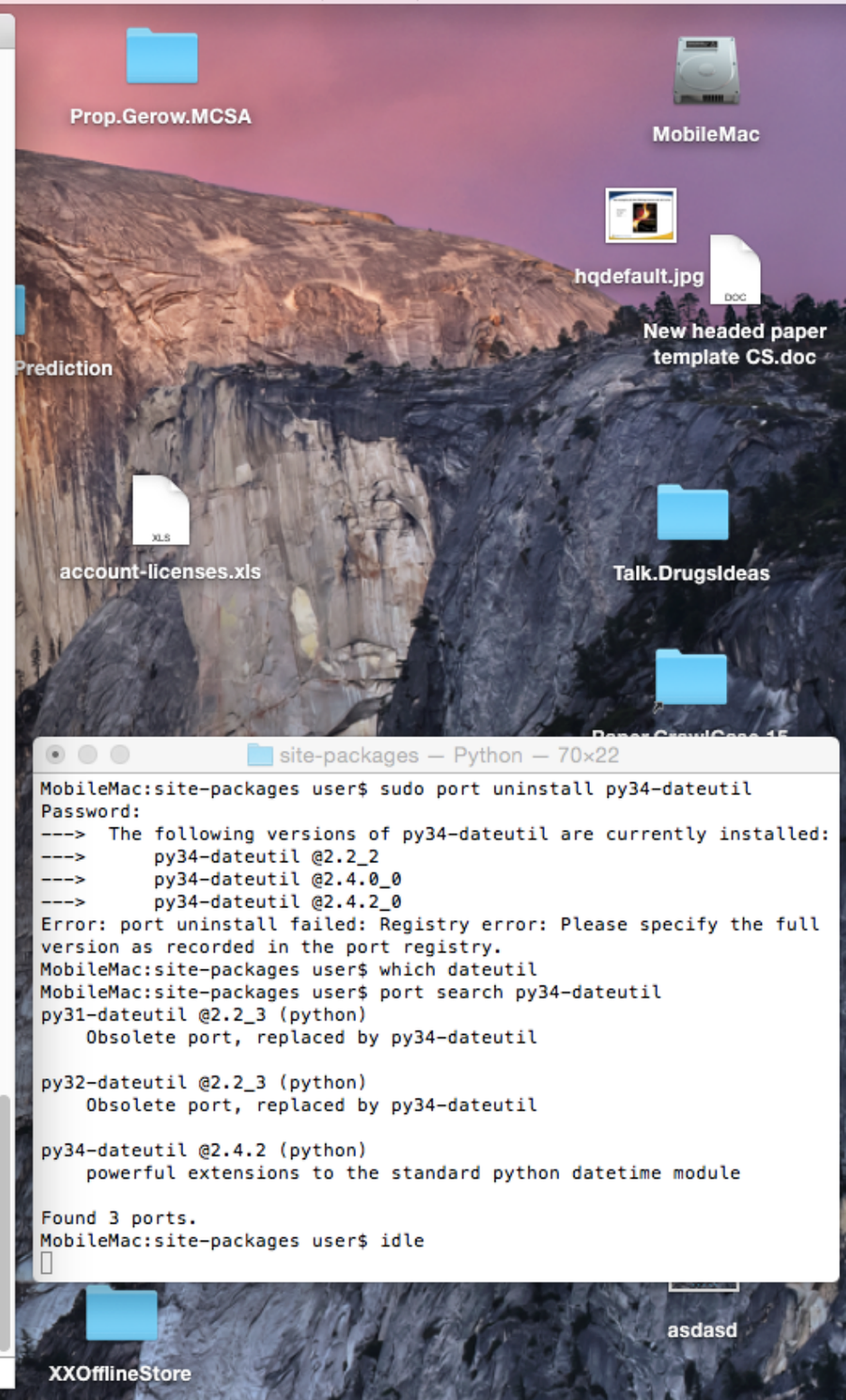
Python File Edit Shell Debug Options Window Help

42% Fri 11:45

Python 3.4.3 Shell

```
_sha1                  email                  pkgutil                tracemalloc
_sha256                enchant                platform               tty
_sha512                encodings              plistlib               turtle
_sitebuiltins          ensurepip              poplib                 turtledemo
_socket                enum                   posix                  types
_sqlite3               errno                  posixpath              unicodedata
_sre                   faulthandler           pprint                 unittest
_ssl                   fcntl                  profile                urllib
_stat                  filecmp                pstats                 uu
_string                fileinput              pty                    uuid
_strptime              fnmatch                pwd                    venv
_struct                formatter              py2app                 warnings
_symtable              fractions              py_compile             wave
_sysconfigdata         ftplib                 pyclbr                 weakref
_testbuffer            functools              pydoc                  webbrowser
_testcapi              gc                     pydoc_data             wsgiref
_testimportmultiple    genericpath            pyexpat                xdrlib
_thread                getopt                 pylab                  xml
_threading_local       getpass                pyparsing              xmlrpc
_tkinter               gettext                pytz                   xxlimited
_tracemalloc           glob                   pyximport              xxsubtype
_warnings              grp                    queue                  yaml
_weakref               gzip                   quopri                 zipfile
_weakrefset            hashlib                random                 zipimport
_yaml                  heapq                  re                     zlib
abc                    hmac                   readline
aifc                   html                   reprlib

Enter any module name to get more help.  Or, type "modules spam" to search
for modules whose name or summary contain the string "spam".

help> quit

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>> import enchant
>>> d = enchant.Dict("en_US")
>>> d.check('hello')
True
>>> d.check('helo')
False
>>>
```

Ln: 203 Col: 60



Fixing Spellings: BigData Way

Grab File Edit Capture Window Help — 19% — Wed 15:50

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.
>>> ================================ RESTART =======
========================
>>>
>>> correct('spellingd')
'spelling'
>>> correct('stroeked')
'stroked'
>>> correct('streked')
'stroked'
>>> correct('stoeked')
'stocked'
>>>
>>>
```

Ln: 14 Col: 4

Python 3.4.1: norvigspell.py - /Users/user/Desktop/norvigspell.py

```
# Norvig Spelling Corrector

import re, collections

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(open('/Users/user/Desktop/big.txt').read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in NWORI

def known(words): return set(w for w in words if w in NWORDS)

def correct(word):
    candidates = known([word]) or known(edits1(word)) or known_edits2(word)
    return max(candidates, key=NWORDS.get)
```

Ln: 20 Col: 21

user — Python — 204×7

```
MobileMac:~ user$ idle
MobileMac:~ user$ python27
-bash: python27: command not found
MobileMac:~ user$ python27
-bash: python27: command not found
MobileMac:~ user$ idle
```

How It Works: Some Probability Theory

How does it work? First, a little theory. Given a word, we are trying to choose the most likely spelling correction for that word (the "correction" may be the original word itself). There is no way to know for sure (for example, should "lates" be corrected to "late" or "latest"?), which suggests we use probabilities. We will say that we are trying to find the correction c, out of all possible corrections, that maximizes the probability of c given the original word w:

$$\text{argmax}_c \, P(c|w)$$

By Bayes' Theorem this is equivalent to:

$$\text{argmax}_c \, P(w|c) \, P(c) \, / \, P(w)$$

Since $P(w)$ is the same for every possible c, we can ignore it, giving:

$$\text{argmax}_c \, P(w|c) \, P(c)$$

There are three parts of this expression. From right to left, we have:

1. $P(c)$, the probability that a proposed correction c stands on its own. This is called the **language model**: think of it as answering the question "how likely is c to appear in an English text?" So P("the") would have a relatively high probability, while P("zxzxzxzyyy") would be near zero.

2. $P(w|c)$, the probability that w would be typed in a text when the author meant c. This is the **error model**: think of it as answering "how likely is it that the author would type w by mistake when c was intended?"

3. argmax_c, the control mechanism, which says to enumerate all feasible values of c, and then choose the one that gives the best combined probability score.

One obvious question is: why take a simple expression like $P(c|w)$ and replace it with a more complex expression involving two models rather than one? The answer is that $P(c|w)$ is *already* conflating two factors, and it is easier to separate the two out and deal with them explicitly. Consider the misspelled word w="thew" and the two candidate corrections c="the" and c="thaw". Which has a higher $P(c|w)$? Well, "thaw" seems good because the only change is "a" to "e", which is a small change. On the other hand, "the" seems good because "the" is a very common word, and perhaps the typist's finger slipped off the "e" onto the "w". The point is that to estimate $P(c|w)$ we have to consider both the probability of c and the probability of the change from c to w anyway, so it is cleaner to formally separate the two factors.

Now we are ready to show how the program works. First P(c). We will read a big text file, big.txt, which consists of about a million words. The file is a concatenation of several public domain books from Project Gutenberg and lists of most frequent words from Wiktionary and the British National Corpus. (On the plane home when I was writing the first version of the code all I had was a collection of Sherlock Holmes stories that happened to be on my laptop; I added the other sources later and stopped adding texts when they stopped helping, as we shall see in the Evaluation section.)

We then extract the individual words from the file (using the function `words`, which converts everything to lowercase, so that "the" and "The" will be the same and then defines a word as a sequence of alphabetic characters, so "don't" will be seen as the two words "don" and "t"). Next we train a probability model, which is a fancy way of saying we count how many times each word occurs, using the function `train`. It looks like this:

```
def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(file('big.txt').read()))
```

At this point, `NWORDS[w]` holds a count of how many times the word `w` has been seen. There is one complication: novel words. What happens with a perfectly good word of English that wasn't seen in our training data? It would be bad form to say the probability of a word is zero just because we haven't seen it yet. There are several standard approaches to this problem; we take the easiest one, which is to treat novel words as if we had seen them once. This general process is called **smoothing**, because we are smoothing over the parts of the probability distribution that would have been zero, bumping them up to the smallest possible count. This is achieved through the class `collections.defaultdict`, which is like a regular Python dict (what other languages call hash tables) except that we can specify the default value of any key; here we use 1.

Now let's look at the problem of enumerating the possible corrections c of a given word w. It is common to talk of the **edit distance** between two words: the number of edits it would take to turn one into the other. An edit can be a deletion (remove one letter), a transposition (swap adjacent letters), an alteration (change one letter to another) or an insertion (add a letter). Here's a function that returns a set of all words c that are one edit away from w:

```
def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b]
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)
```

Fixing Spellings: Actual Prog

Python 3.4.3 Shell

```
Python 3.4.3 (default, May 25 2015, 18:48:21)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] o
n darwin
Type "copyright", "credits" or "license()" for more info
rmation.
>>> ================================ RESTART ==========
==========
>>>
/Users/user/Dropbox/Teaching.TextAnalytics/Lect2.DeData/
Prac2.Basics.I.Data*test**
/Users/user/Dropbox/Teaching.TextAnalytics/Lect2.DeData/
Prac2.Basics.I.Data*test**/big.txt
>>> correct('hello')
'hello'
>>> correct('helo')
'held'
>>> correct('streked')
'stroked'
>>> correct('stroked')
'stroked'
>>>
```

Ln: 16 Col: 4

norvigspell.py - /Users/user/Dropbox/Teaching.TextAnalytics/Lect2.DeData/Prac2.Basics.I.Data*test**/norvigspell.py (3.4.3)

```
# Norvig Spelling Corrector

import re, collections, os

__location__ = os.path.realpath(
    os.path.join(os.getcwd(),os.path.dirname(__file__)))
print(__location__)

file_name = os.path.join(__location__, 'big.txt')
print(file_name)

def words(text): return re.findall('[a-z]+', text.lower())

def train(features):
    model = collections.defaultdict(lambda: 1)
    for f in features:
        model[f] += 1
    return model

NWORDS = train(words(open(file_name).read()))

alphabet = 'abcdefghijklmnopqrstuvwxyz'

def edits1(word):
   splits     = [(word[:i], word[i:]) for i in range(len(word) + 1)]
   deletes    = [a + b[1:] for a, b in splits if b]
   transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1
   replaces   = [a + c + b[1:] for a, b in splits for c in alphabet if b
   inserts    = [a + c + b     for a, b in splits for c in alphabet]
   return set(deletes + transposes + replaces + inserts)

def known_edits2(word):
    return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in N

def known(words): return set(w for w in words if w in NWORDS)
```

Ln: 28 Col: 37

site-packages — Python — 162×6

```
    powerful extensions to the standard python datetime module

Found 3 ports.
MobileMac:site-packages user$ idle
MobileMac:site-packages user$ idle
```

Proj.(TP) Farmer

asdasd

Fixing Spellings: Google Way

- http://www.internetmarketingninjas.com/blog/search-engine-optimization/google-spell-check/

Text Pre-Processing

Stemming & Lemmatising

What is the Problem?

- We have a number of normalised tokens that, we hope, are the words in the document
- But, we are going to be counting them and doing other manipulations on frequency
- So, it is very important that words that are more or less the same are identified…

Why Stem?

- So, we need to recognise variations of the same *stem* or *root**: fish~ for fishes, fishing, fished…

- We also need to recognise when two words are the same but from different syntactic categories (or parts of speech, POS); fish the *noun* and fish the *verb*

- Note, the root form of a word may be quite different **be~** for is, are, am

* may be used differently

Why Stem?

- So, if we convert the variations into the root form then we have a more accurate count of the word

- So, if we distinguish two words that appear the same but are actually different fish-v and fish-n (stemming doesn't really do this…)

- Generally, stemming refers to the recovery of the root forms of words to show these commonalities and differences

Stemming Definition…

Stemming

From Wikipedia, the free encyclopedia

> This article **needs attention from an expert on the subject.** Please add a *reason* or a *talk* parameter to this template to explain the issue with the article. Consider associating this request with a WikiProject. *(October 2010)*

For the skiing technique, see Stem (skiing).

In linguistic morphology and information retrieval, **stemming** is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called **conflation**.

Stemming programs are commonly referred to as **stemming algorithms** or **stemmers**.

Stemming Algorithms

- Technically, stemming is described the removal of morphological affixes

- Porter Stemmer, Snowball stemmers, most used but many more in **nltk** (types and languages)

- http://www.nltk.org/howto/stem.html

affix

verb
3rd person present: **affixes**
/əˈfɪks/

1. stick, attach, or fasten (something) to something else.
   "panels to which he affixes copies of fine old prints"
   synonyms: attach, stick, fasten, bind, fix, post, secure, join, connect, couple;
   More

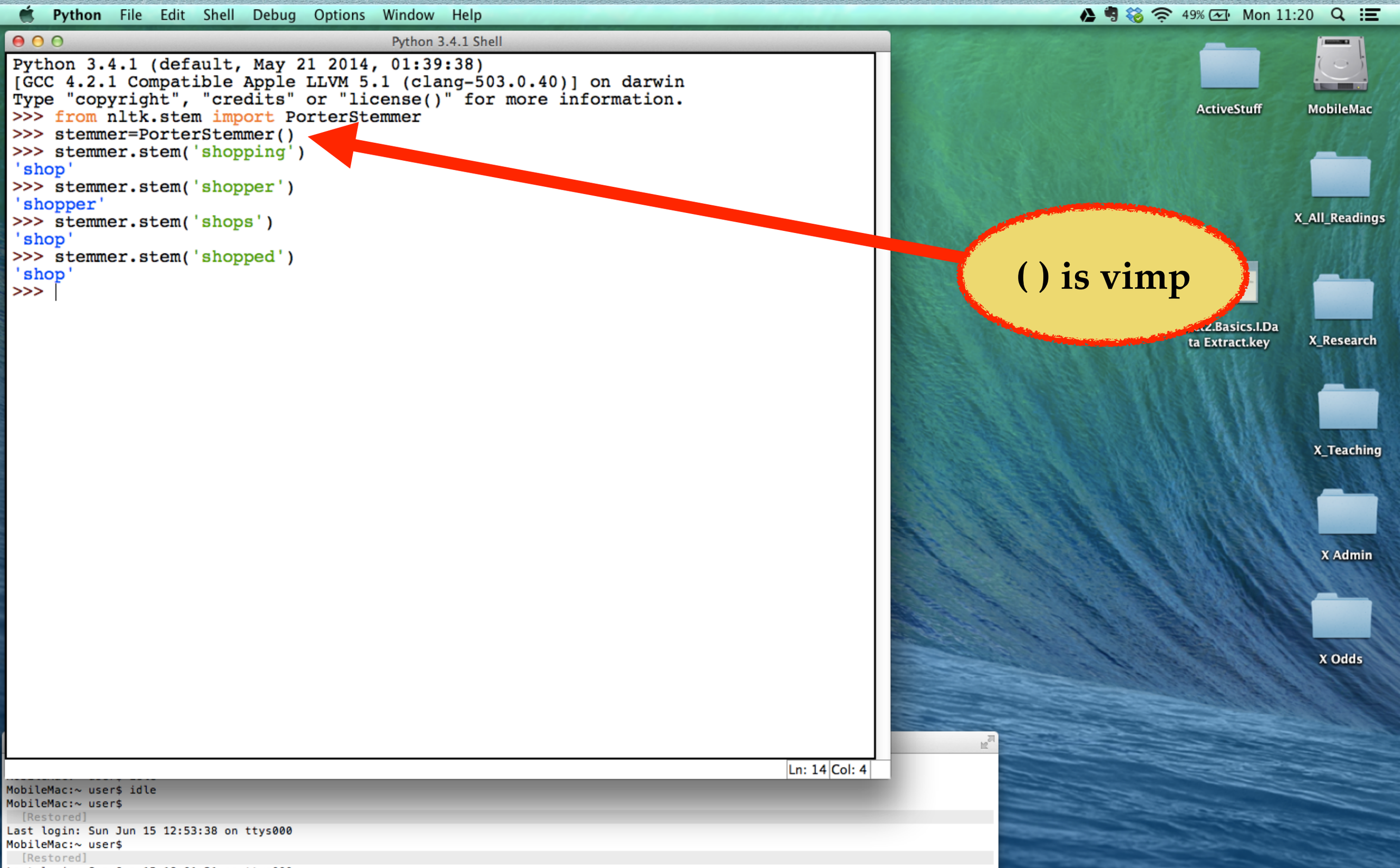
noun GRAMMAR
plural noun: **affixes**
/ˈafɪks/

1. an addition to the base form or stem of a word in order to modify its meaning or create a new word.

Porter Stemming: Brutal

- Porter Stemming is quite brutal, tho' most of it makes sense (e.g., does plurals well)
- It works off a mixture of specific rules and very general ones (e.g., cutting '-ed' off words)
- http://tartarus.org/martin/PorterStemmer/

Porter Stemming: Eg



Porter Stemming: Eg

Python File Edit Shell Debug Options Window Help

66% Mon 11:35

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
>>> text = open('/Users/user/Desktop/text.txt')
>>> rawtext = text.read()
>>> from nltk.stem import PorterStemmer
>>> 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"
>>> splitrawtext = rawtext.split(" ")
>>> splitrawtext
['So,', 'this', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put',
'togethr', 'to', 'check', 'on\nthis', 'tokenisation', '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']
>>> stemmer = PorterStemmer()
>>> stemmer.stem(wd) for wd in splitrawtext
SyntaxError: invalid syntax
>>> [stemmer.stem(wd) for wd in splitrawtext]
['So,', 'thi', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i'v", 'put', 't
ogethr', 'to', 'check', 'on\nthi', 'tokenis', 'crap', 'and', 'I', 'am', 'interest
', 'in', 'how', 'I.B.M.', 'or', 'the', 'U.S.A.', 'or', 'the', 'USA\ni', 'handled,
', 'ok?', 'Thi', 'and', 'whether', 'it', 'properli', 'deal', 'with', 'websit', 'l
ike', 'www.ucd.ie\nand', 'email', 'address', 'like', 'mark.keane@ucd.email.ie.',
'', 'Also', 'other', 'weirditi', 'like\nth', 'great', "O'Neil", 'and', 'like', 'M
*A*S*H,', 'which', 'should', 'be', 'mayb', '7', 'tokens?\n']
>>>
```

Ln: 17 Col: 53

/Users/user/Desktop/text.txt

```
put togethr to check on
in how I.B.M. or the U.S.A. or the USA
ly deals with websites 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?
```

```
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 12:53:38 on ttys000
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 18:01:21 on ttys000
MobileMac:~ user$ idle
```

Porter Stemming: Eg

```
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
>>> text = open('/Users/user/Desktop/text.txt')
>>> rawtext = text.read()
>>> from nltk.stem import PorterStemmer
>>> 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"
>>> splitrawtext = rawtext.split(" ")
>>> splitrawtext
['So,', 'this', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put',
'togethr', 'to', 'check', 'on\nthis', 'tokenisation', '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']
>>> stemmer = PorterStemmer()
>>> stemmer.stem(wd) for wd in splitrawtext
SyntaxError: invalid syntax
>>> [stemmer.stem(wd) for wd in splitrawtext]
['So,', 'thi', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put', 't
ogethr', 'to', 'check', 'on\nthi', 'tokenis', 'crap', 'and', 'I', 'am', 'interest
', 'in', 'how', 'I.B.M.', 'or', 'the', 'U.S.A.', 'or', 'the', 'USA\ni', 'handled,
', 'ok?', 'Thi', 'and', 'whether', 'it', 'properli', 'deal', 'with', 'websit', 'l
ike', 'www.ucd.ie\nand', 'email', 'address', 'like', 'mark.keane@ucd.email.ie.',
'', 'Also', 'other', 'weirditi', 'like\nth', 'great', "O'Neil", 'and', 'like', 'M
*A*S*H,', 'which', 'should', 'be', 'mayb', '7', 'tokens?\n']
>>>
```

Ln: 17 Col: 53

```
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 12:53:38 on ttys000
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 18:01:21 on ttys000
MobileMac:~ user$ idle
```

```
put togethr
in how I.B.M. or the U.S.A. or the USA
ly deals with websites like www.ucd.ie
ail.ie.  Also other weirdities like
the great O'Neill and like M*A*S*H, which should be maybe 7 tokens?
```

import the method

Split text into a list of strings

Call stemmer on each string in list

Porter Stemming: Steps

- Step 1: Gets rid of plurals and -ed or -ing suffixes
- Step 2: Turns terminal y to i when there is another vowel in the stem
- Step 3: Maps double suffixes to single ones: | -ization, -ational
- Step 4: Deals with suffixes, -full, -ness etc.
- Step 5: Takes off -ant, -ence, etc.
- Step 6: Removes a final -e

Lancaster Stemming: Eg

Python File Edit Shell Debug Options Window Help

74% Mon 11:42

Python 3.4.1 Shell

```
>>> splitrawtext
['So,', 'this', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put',
'togethr', 'to', 'check', 'on\nthis', 'tokenisation', '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']
>>> stemmer = PorterStemmer()
>>> stemmer.stem(wd) for wd in splitrawtext
SyntaxError: invalid syntax
>>> [stemmer.stem(wd) for wd in splitrawtext]
['So,', 'thi', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i'v", 'put', 't
ogethr', 'to', 'check', 'on\nthi', 'tokenis', 'crap', 'and', 'I', 'am', 'interest
', 'in', 'how', 'I.B.M.', 'or', 'the', 'U.S.A.', 'or', 'the', 'USA\ni', 'handled,
', 'ok?', 'Thi', 'and', 'whether', 'it', 'properli', 'deal', 'with', 'websit', 'l
ike', 'www.ucd.ie\nand', 'email', 'address', 'like', 'mark.keane@ucd.email.ie.',
'', 'Also', 'other', 'weirditi', 'like\nth', 'great', "O'Neil", 'and', 'like', 'M
*A*S*H,', 'which', 'should', 'be', 'mayb', '7', 'tokens?\n']
>>> stemmer = LancasterStemmer()
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    stemmer = LancasterStemmer()
NameError: name 'LancasterStemmer' is not defined
>>> stemmer = nltk.LancasterStemmer()
>>> [stemmer.stem(wd) for wd in splitrawtext]
['so,', 'thi', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put', '
togethr', 'to', 'check', 'on\nthis', 'tok', 'crap', 'and', 'i', 'am', 'interest',
'in', 'how', 'i.b.m.', 'or', 'the', 'u.s.a.', 'or', 'the', 'usa\nis', 'handled,',
'ok?', 'thi', 'and', 'wheth', 'it', 'prop', 'deal', 'with', 'websit', 'lik', 'www
.ucd.ie\nand', 'email', 'address', 'lik', 'mark.keane@ucd.email.ie.', '', 'also',
'oth', 'weird', 'like\nth', 'gre', "o'neill", 'and', 'lik', 'm*a*s*h,', 'which',
'should', 'be', 'mayb', '7', 'tokens?\n']
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

Ln: 23 Col: 0

```
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 12:53:38 on ttys000
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 18:01:21 on ttys000
MobileMac:~ user$ idle
```

ActiveStuff

MobileMac

stem

X_All_Readings

Lect2.Basics.I.Data Extract.key

X_Research

TXT

text.txt

X_Teaching

X Admin

/Users/user/Desktop/text.txt

```
 put togethr to check on
 in how I.B.M. or the U.S.A. or the USA
ly deals with websites 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?
```

Lancaster Stemming: Eg

```
>>> splitrawtext
['So,', 'this', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put',
'togethr', 'to', 'check', 'on\nthis', 'tokenisation', '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']
>>> stemmer = PorterStemmer()
>>> stemmer.stem(wd) for wd in splitrawtext
SyntaxError: invalid syntax
>>> [stemmer.stem(wd) for wd in splitrawtext]
['So,', 'thi', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put', 't
ogethr', 'to', 'check', 'on\nthi', 'tokenis', 'crap', 'and', 'I', 'am', 'interest
', 'in', 'how', 'I.B.M.', 'or', 'the', 'U.S.A.', 'or', 'the', 'USA\ni', 'handled,
', 'ok?', 'Thi', 'and', 'whether', 'it', 'properli', 'deal', 'with', 'websit', 'l
ike', 'www.ucd.ie\nand', 'email', 'address', 'like', 'mark.keane@ucd.email.ie.',
'', 'Also', 'other', 'weirditi', 'like\nth', 'great', "O'Neil", 'and', 'like', 'M
*A*S*H,', 'which', 'should', 'be', 'mayb', '7', 'tokens?\n']
>>> stemmer = LancasterStemmer()
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    stemmer = LancasterStemmer()
NameError: name 'LancasterStemmer' is not defined
>>> stemmer = nltk.LancasterStemmer()
>>> [stemmer.stem(wd) for wd in splitrawtext]
['so,', 'thi', 'is', 'just', 'a', 'bunch', 'of', 'text', 'that', "i've", 'put', '
togethr', 'to', 'check', 'on\nthis', 'tok', 'crap', 'and', 'i', 'am', 'interest',
'in', 'how', 'i.b.m.', 'or', 'the', 'u.s.a.', 'or', 'the', 'usa\nis', 'handled,',
'ok?', 'thi', 'and', 'wheth', 'it', 'prop', 'deal', 'with', 'websit', 'lik', 'www
.ucd.ie\nand', 'email', 'address', 'lik', 'mark.keane@ucd.email.ie.', '', 'also',
'oth', 'weird', 'like\nth', 'gre', "o'neill", 'and', 'lik', 'm*a*s*h,', 'which',
'should', 'be', 'mayb', '7', 'tokens?\n']
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

Ln: 23 Col: 0

we did not import **method**

but, can name it explicitly instead of import

/Users/user/Desktop/text.txt

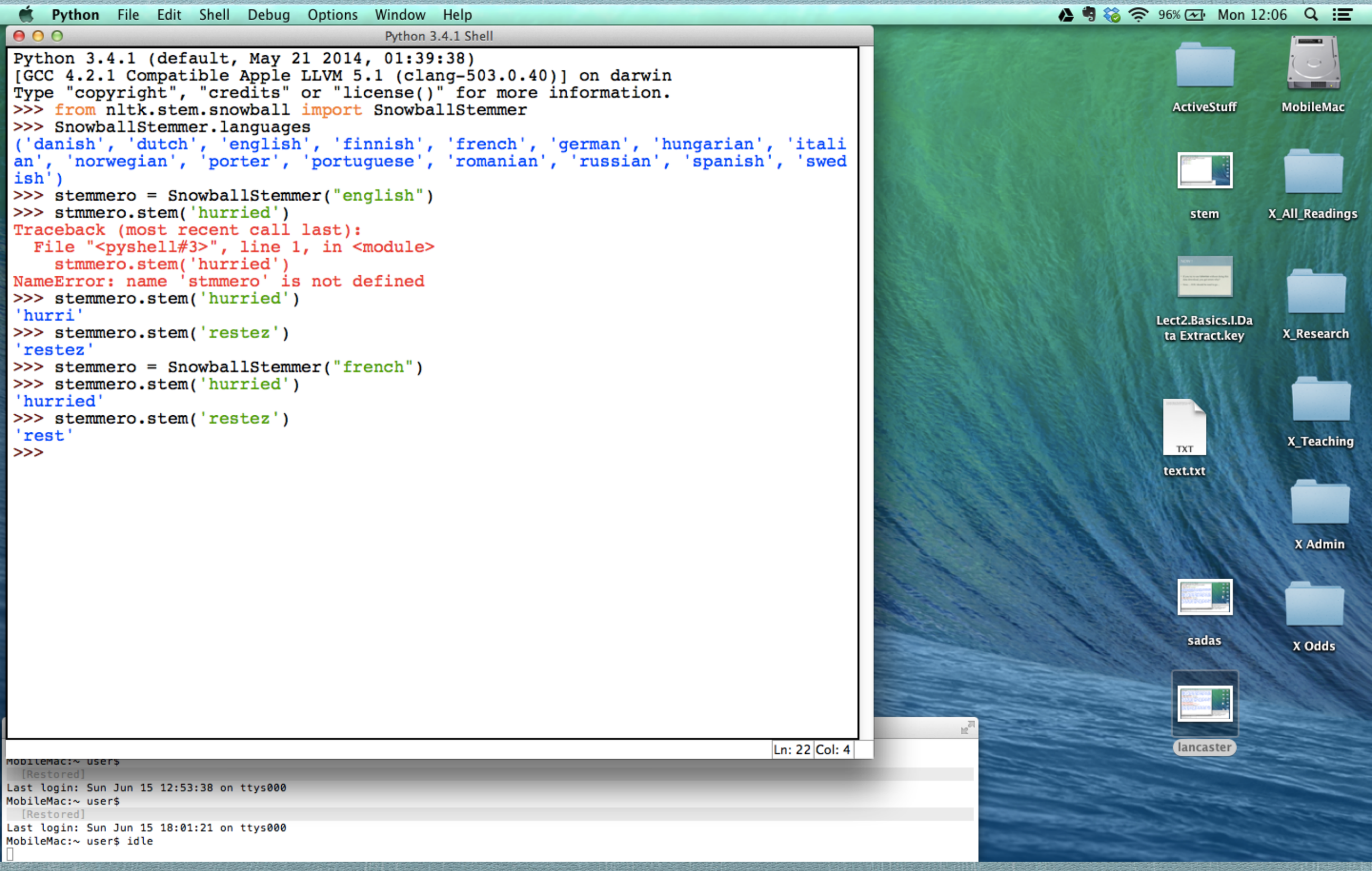
```
put togethr to check on
in how I.B.M. or the U.S.A. or the USA
ly deals with websites like www.ucd.ie
and email addresses like mark.keane@ucd.email.ie.  Also other weirdities like
the great O'Neill and like M*A*S*H, which should be maybe 7 tokens?
```

```
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 12:53:38 on ttys000
MobileMac:~ user$
  [Restored]
Last login: Sun Jun 15 18:01:21 on ttys000
MobileMac:~ user$ idle
```

Snowball Stemming

- **nltk** has a bunch of off-the-shelf stemmers
- Snowball Stemmer, in **nltk**, is really a language for building stemmers, as they are language-specific this is useful
- Also, you can use it to explicitly switch languages

Snowball Stemming: Eg



Butt Stemmers…But

- Are rather crude; no word meaning disambiguation (bats, batting)
- No POS disambiguation (fish and fish)
- Can't handle irregulars (am, is, are)
- So, sometimes you need more…

www.eecis.udel.edu/~trnka/CISC889-11S/lectures/dan-porters.pdf

Why Lemmatise?

- Lemmatisation is RollsRoyce stemming
- Only removes affixes if the resulting word is in its dictionary (so, is slower…)
- So, it should be more accurate; see Wordnet (Miller, 1995), Sketch Engine (Kilgarriff et al, 2004)
- Some systems can get quite complex, do partial parses and contain frequency info for lemmas

Why Stem?

REM

- So, we need to recognise variations of the same *stem* or *root**: fish~ for fishes, fishing, fished…

- We also need to recognise when two words are the same but from different syntactic categories (or parts of speech, POS); fish the *noun* and fish the *verb*

- Note, the root form of a word may be quite different **be~** for is, are, am

* may be used differently

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)

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

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'.

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 subordinating
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	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
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 lemmatizers 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 lemmatizer
- However, note, the WordNet Lemmatizer 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
```

Proj.H2020.TweetBot

Teaching.TextAnalytics alias

alias genevievec...eb2015.docx hashtagb

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

IDLE File Edit Format Run Options Window Help — 51% Wed 11:52

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 ================================
>>>
[('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
  ./.)
>>>
```

Ln: 19 Col: 6

Python 3.4.1: pos&lemma.py - /Users/user/Dropbox/Teaching.TextAnalytics/Lect2.DeData/xLect2.Progs*done**/pos&lemma.py

```
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)
```

Ln: 8 Col: 16

user — bash — 79×15

```
in: Wed Sep  2 11:10:38 on console
1a54f:~ user$ idle
```

New

MobileMac

xxxMyWeek Pers.Alicia Pers.Laptop XXOff

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…

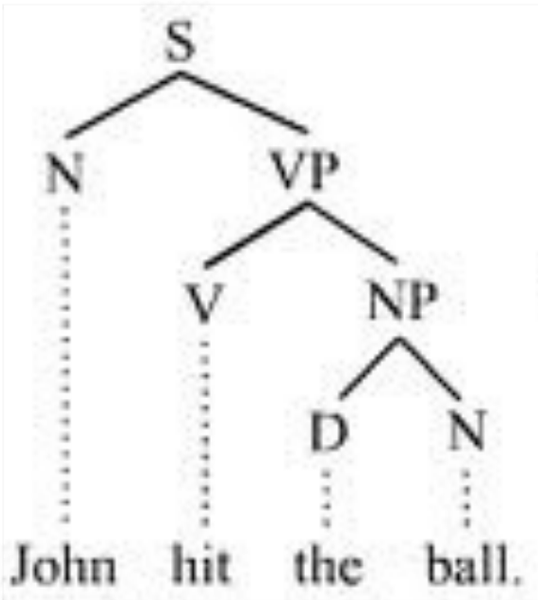
REM

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

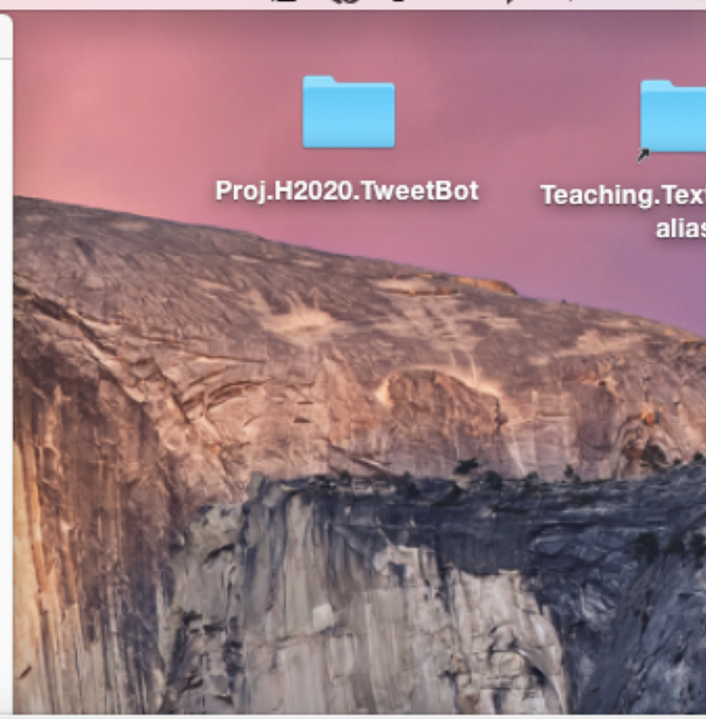
Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. The term parsing comes from Latin pars, meaning part. Wikipedia



Constituency-based parse tree

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))))))
>>>
```

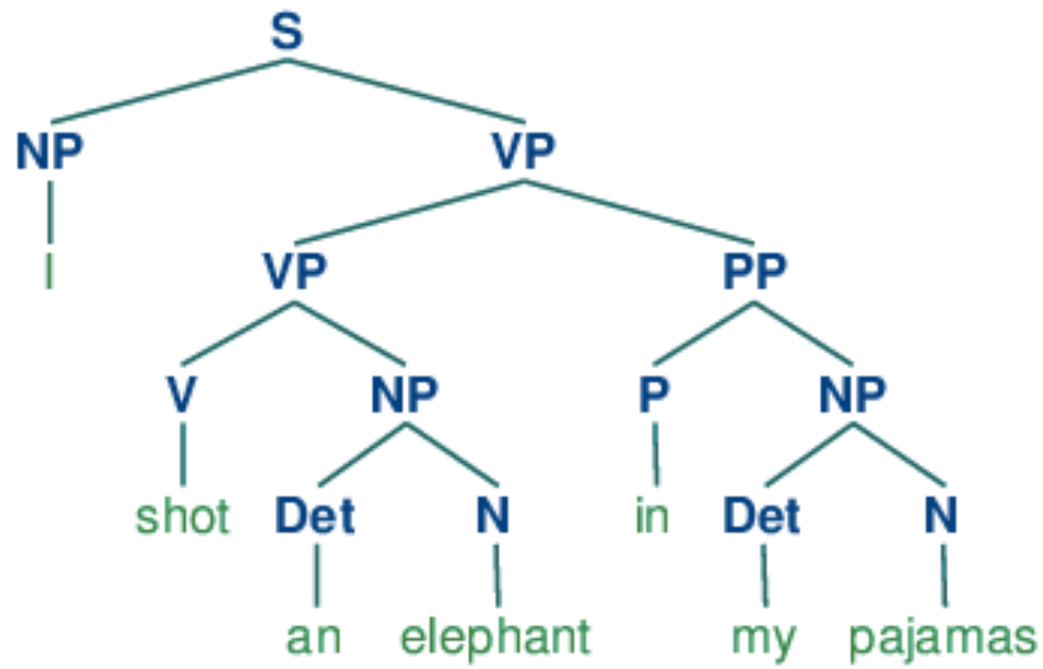


Python 3.4.1: parser.py - /Users/user/Dropbox/Teaching.TextAnalytics/Lect2.Bits.Data/Prac2.Basics.I.

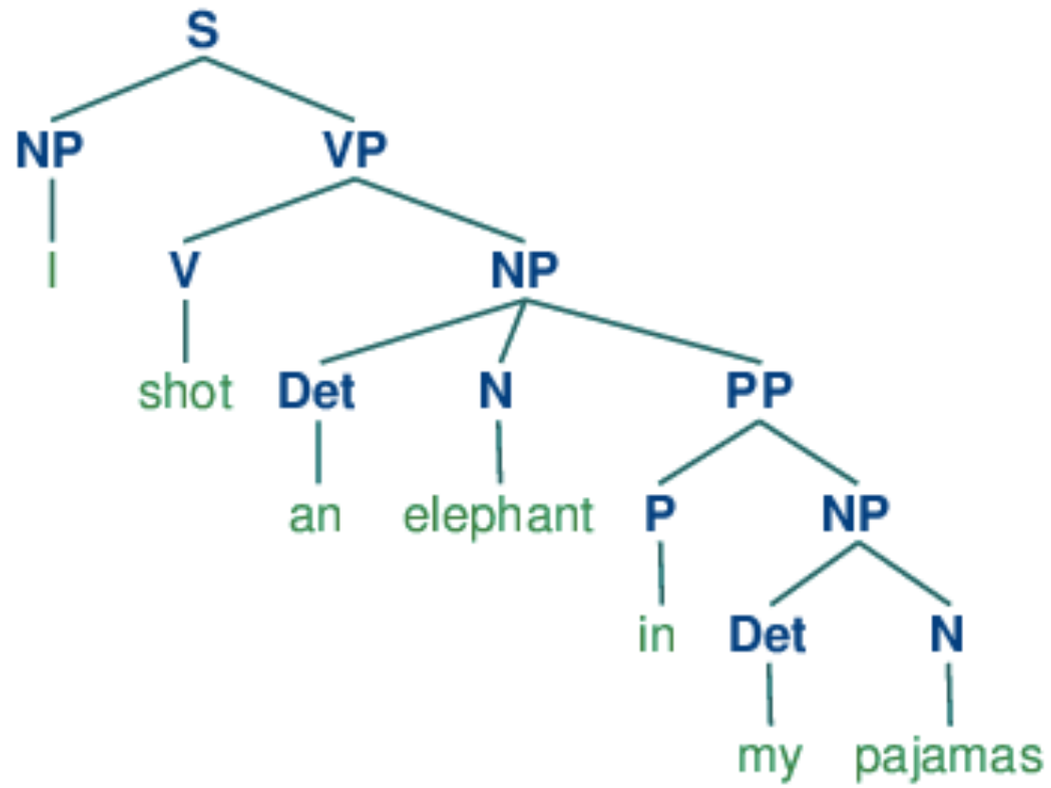
```
import nltk

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

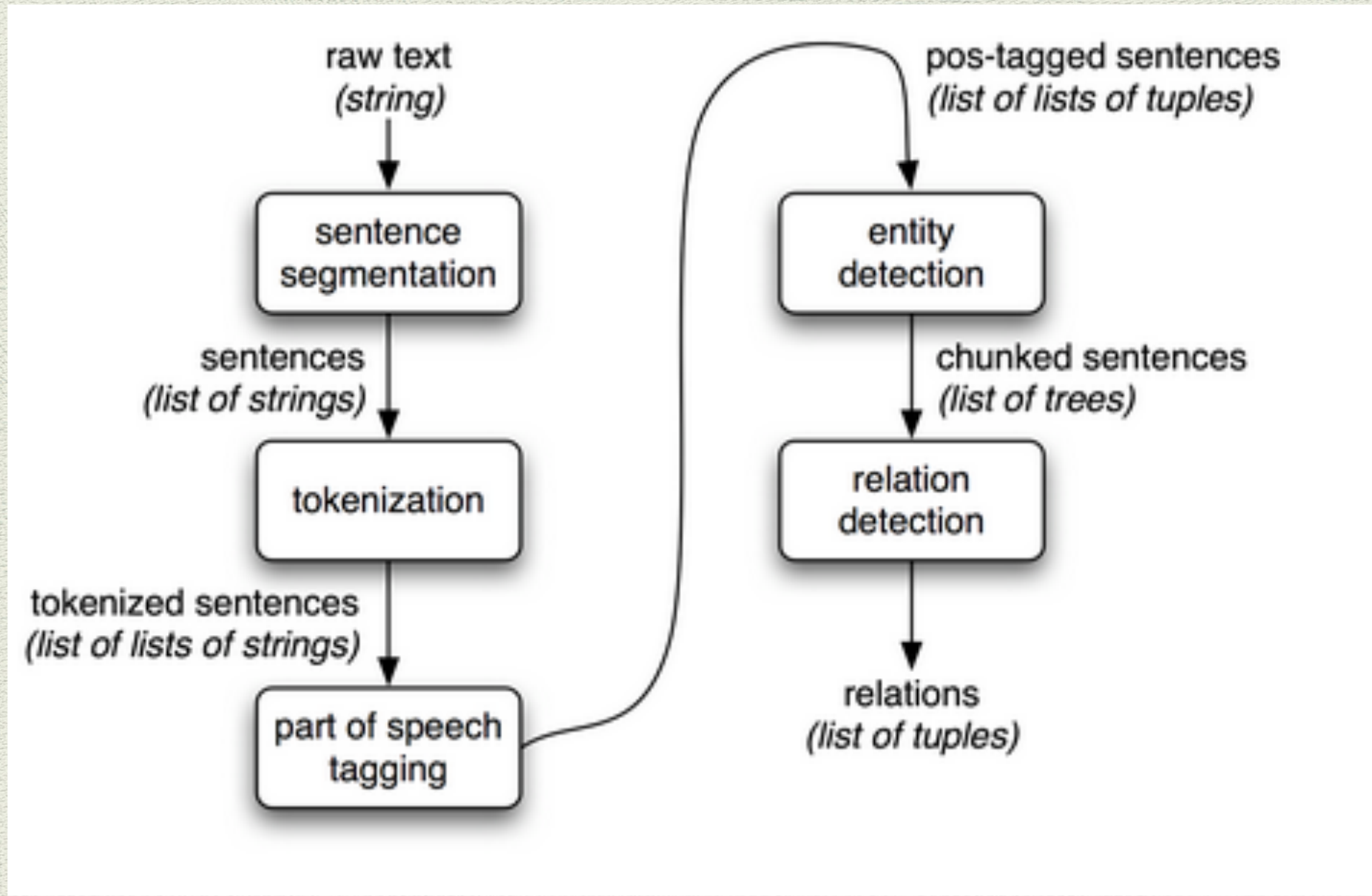
REM

- *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



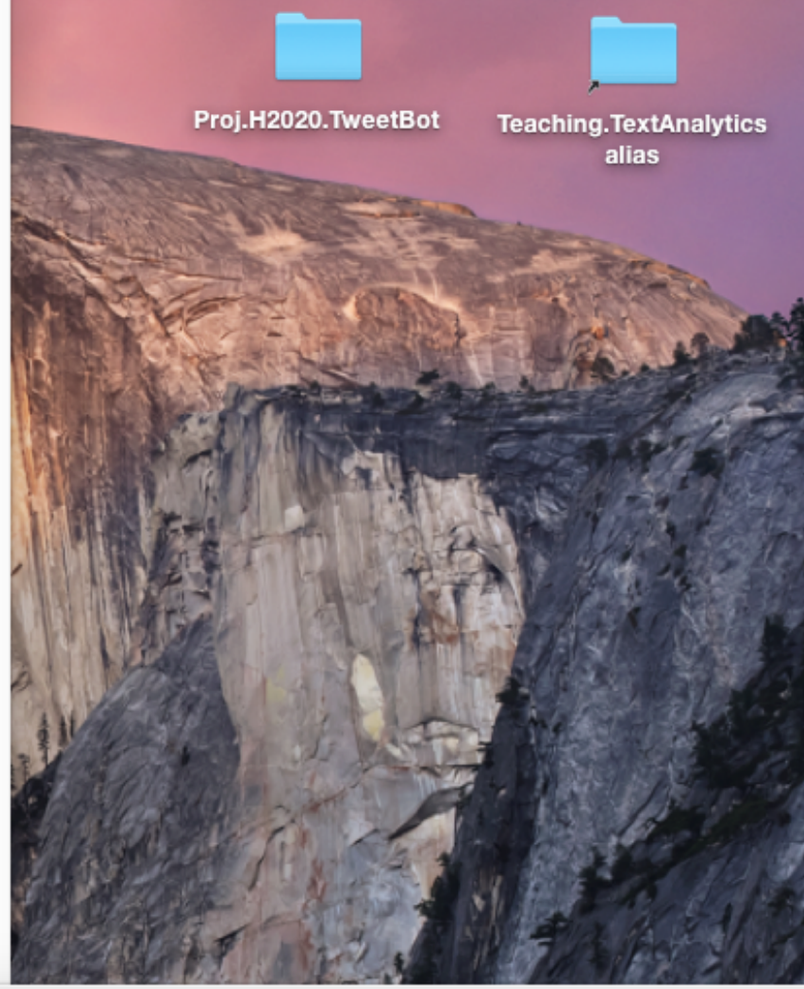
Simple EG

IDLE File Edit Format Run Options Window Help

100% Wed 12

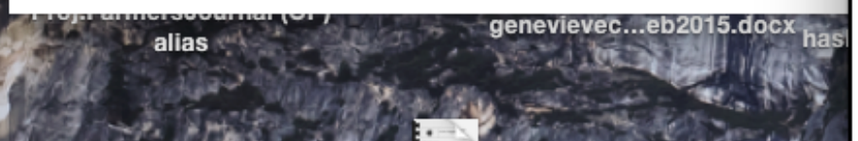
Python 3.4.1 Shell

```
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 ==========
>>>
[('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
  ./.)
>>>
```



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('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)
- Open Calais is also heavily used
- But, need good reasons to go this far in your pre-processing, esp. considering 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 lemmatization 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 lemmatization (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

Python File Edit Shell Debug Options Window Help

100% Mon 19:12

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.
>>> open('/Users/user/Desktop/simple.txt')
<_io.TextIOWrapper name='/Users/user/Desktop/simple.txt' mode='r' enco
ding='UTF-8'>
>>> open('/Users/user/Desktop/simple.txt').read()
'simple string to read in to python.\n'
>>> open('/Users/user/Desktop/simple.txt').read
<built-in method read of _io.TextIOWrapper object at 0x10e592120>
>>> string = open('/Users/user/Desktop/simple.txt').read()
>>> string
'simple string to read in to python.\n'
>>> type(string)
<class 'str'>
>>>
```

Ln: 15 Col: 4

HTML

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

```
simple string to read in to python.
```

Ln: 2 Col: 0

```
ibrary/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages/PyObjC']
>>> /opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages
  File "<stdin>", line 1
    /opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages
    ^
SyntaxError: invalid syntax
>>> quit()
MobileMac:bin user$ idle
MobileMac:bin user$ idle
```

```
ary/Frameworks/Python.framework/Ver
.4/lib-dynload', '/Users/user/Libra
n3.4/site-packages', '/opt/local/L
```

```
ter-0.0.0-py3.4.egg-info
ado
ado-3.2.2-py3.4.egg-info
```

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…