Python programming — text and web mining

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

September 22, 2014

Overview

Get the stuff: Crawling, search

Converting: HTML processing/stripping, format conversion

Tokenization, identifying and splitting words and sentences.

Word normalization, finding the stem of the word, e.g., “talked” $\rightarrow$ “talk”

Text classificiation (supervized), e.g., spam detection.

Web crawling issues

Honor `robots.txt` — the file on the Web server that describe what you are allowed to crawl and not.

Tell the Web server who you are.

Handling errors and warnings gracefully, e.g., the 404 (“Not found”).

Don’t overload the Web server you are downloading from, especially if you do it in parallel.

Consider parallel download large-scale crawling

Crawling restrictions in `robots.txt`

Example `robots.txt` on `http://neuro.compute.dtu.dk` with rule:

```
Disallow: /wiki/Special:Search
```

Meaning `http://neuro.compute.dtu.dk/wiki/Special:Search` should not be crawled.

Python module `robotparser` for handling rules:

```
>>> import robotparser
>>> rp = robotparser.RobotFileParser()
>>> rp.set_url("http://neuro.compute.dtu.dk/robots.txt")
>>> rp.read()          # Reads the robots.txt
>>> rp.can_fetch("*", "http://neuro.compute.dtu.dk/wiki/Special:Search")
False
>>> rp.can_fetch("*", "http://neuro.compute.dtu.dk/movies/")
True
```

Tell the Web server who you are

Use of `urllib2` module to set the `User-agent` of the HTTP request:

```
import urllib2
opener = urllib2.build_opener()
opener.addheaders = [("User-agent", "fnielsenbot/0.1 (Finn A. Nielsen)")]
response = opener.open("http://neuro.compute.dtu.dk")
```

This will give the following entry (here split into two line) in the Apache Web server log (`/var/log/apach2/access.log`) :

```
130.225.70.226 - - [31/Aug/2011:15:55:28 +0200]
  "GET / HTTP/1.1" 200 6685 "-" "fnielsenbot/0.1 (Finn A. Nielsen)"
```

This allows a Web server admininstrator to block you if you put too much load on the Web server.

See also (Pilgrim, 2004, section 11.5) “Setting the User-Agent”.

The `requests` module

`urllib` and `urllib2` are in the Python Standard Library.

Outside this PSL is `requests` which some regards as more convenient ("for humans"), e.g., setting the user-agent and requesting a page is only one line:

```
import requests
response = requests.get("http://neuro.compute.dtu.dk",
                        headers={'User-Agent': "fnielsenbot/0.1"})
```

The response object also has a JSON conversion method:

```
>>> url = "https://da.wikipedia.org/w/api.php"
>>> params = {"action": "query", "prop": "links", "pllimit": "500", "format": "json"}
>>> params.update(titles="Python (programmeringssprog)")
>>> requests.get(url, params=params).json()["query"]["pages"].values()[0]["links"]
[{u'ns': 0, u'title': u'Aspektorienteret programmering'}, {u'ns': 0,
u'title': u'Eiffel (programmeringssprog)'}, {u'ns': 0, u'title':
u'Funktionel programmering'} ...
```

Handling errors

```
>>> import urllib
>>> urllib.urlopen("http://neuro.compute.dtu.dk/Does_not_exist").read()[64:
'<title>404 Not Found</title>'
```

Ups! You may need to look at `getcode()` from the response:

```
>>> response = urllib.urlopen("http://neuro.compute.dtu.dk/Does_not_exist")
>>> response.getcode()
404
```

`urllib2` throws an exception:

```
import urllib2
opener = urllib2.build_opener()
try:
    response = opener.open('http://neuro.compute.dtu.dk/Does_not_exist')
except urllib2.URLError as e:
    print(e.code)                    # In this case: 404
```

Handling errors with `requests`

The `requests` library does not raise by default on 'ordinary' errors, but you can call the `raise_for_status()` exception:

```
>>> import requests
>>> response = requests.get("http://neuro.compute.dtu.dk/Does_not_exist")
>>> response.status_code
404
>>> response.ok
False
>>> response.raise_for_status()
[...]
requests.exceptions.HTTPError: 404 Client Error: Not Found
```

Note that `requests` does raise errors on, e.g., on the name service error with `requests.get('http://asdf.dtu.dk')`.

Don't overload Web servers

Don't overload the webserver by making a request right after response

Put in a `time.sleep(a_few_seconds)` to be nice.

Some big websites have automatic load restrictions and need authentication, e.g., Twitter.

Serial large-scale download

Serial download from 4 different Web servers:

```
import time, urllib2
urls = ['http://dr.dk', 'http://nytimes.com', 'http://bbc.co.uk',
        'http://finnaarupnielsen.wordpress.com']
start = time.time()
result1 = [(time.time()-start, urllib2.urlopen(url).read(),
            time.time()-start) for url in urls]
```

Plot download times:

```
from pylab import *
hold(True)
for n, r in enumerate(result1):
    plot([n+1, n+1], r[::2], 'k-', linewidth=30, solid_capstyle='butt')

ylabel('Time [seconds]'); grid(True); axis((0, 5, 0, 4)); show()
```

Parallel large-scale download

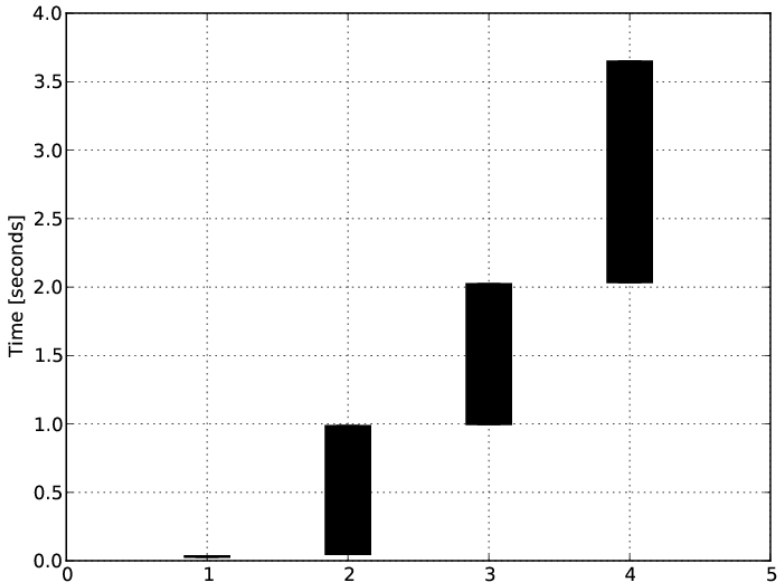
`twisted` event-driven network engine (http://twistedmatrix.com) could be used. For an example see RSS feed aggregator in Python Cookbook (Martelli et al., 2005, section 14.12).

Or use `multiprocessing`

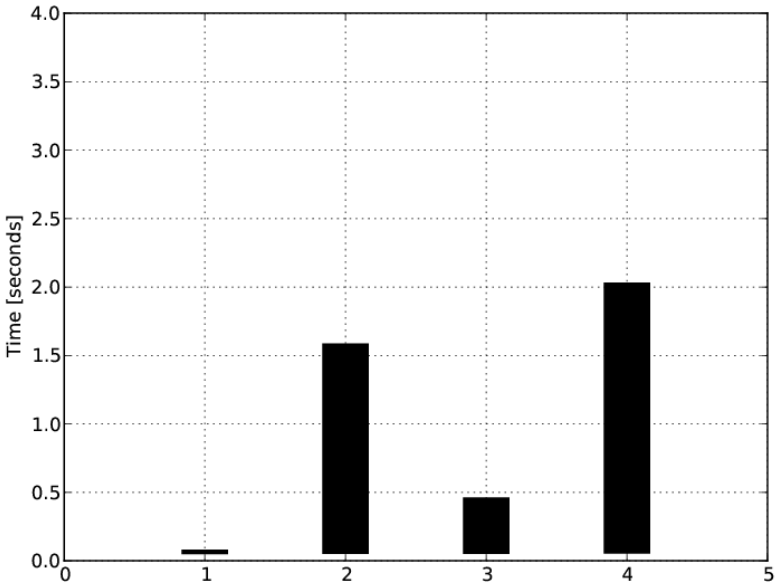
```
import multiprocessing, time, urllib2
def download((url, start)):
    return (time.time()-start, urllib2.urlopen(url).read(),
            time.time()-start)

start = time.time()
pool = multiprocessing.Pool(processes=4)
result2 = pool.map(download, zip(urls, [start]*4))
```

Serial



Parallel



In this small case the parallel download is almost twice as fast.

Combinations

It becomes more complicated:

When you download in parallel and need to make sure that you are not downloading from the same server in parallel.

When you need to keep track of downloading errors (should they be postponed or dropped?)

Reading feeds with `feedparser` ...

Mark Pilgrim's Python module `feedparser` for RSS and Atom XML files.

`feedparser.parse()` may read from a URL, file, stream or string. Example with Google blog search returning "atoms":

```
import feedparser
url = "http://blogsearch.google.dk/blogsearch_feeds?" + \
      "q=visitdenmark&output=atom"
f = feedparser.parse(url); f.entries[0].title
```

gives `u'<b>VisitDenmark</b> fjerner fupvideo fra nettet - Politiken.dk'`

Some feed fields may contain HTML markup. `feedparser` does HTML sanitizing and removes, e.g., the `<script>` tag.

For mass download see also Valentino Volonghi and Peter Cogolo's module with `twisted` in (Martelli et al., 2005)

. . . Reading feeds with `feedparser`

Some of the most useful fields in the feedparser dictionary (see also feedparser reference):

```
f.bozo                  # Indicates if errors occured during parsing
f.feed.title            # Title of feed, e.g., blog title
f.feed.link             # Link to the blog
f.feed.links[0].href    # URL to feed

f.entries[i].title              # Title of post (HTML)
f.entries[i].subtitle           # Subtitle of the post (HTML)
f.entries[i].link               # Link to post
f.entries[i].updated            # Date of post in string
f.entries[i].updated_parsed     # Parsed date in tuple
f.entries[i].summary            # Posting (HTML)
```

The `summary` field may be only partial.

Reading JSON ...

JSON (JavaScript Object Notation), http://json.org, is a lightweight data interchange format particularly used on the Web.

Python implements JSON encoding and decoding with among others the `json` and `simplejson` modules.

`simplejson` and newer `json` use, e.g., `loads()` and `dumps()` whereas older `json` uses `read()` and `write()`. http://docs.python.org/library/json.html

```
>>> s = simplejson.dumps({'Denmark': {'towns': ['Copenhagen',
u'Århus'], 'population': 5000000}})   # Note Unicode
>>> print s
{"Denmark": {"towns": ["Copenhagen", "\u00c5rhus"], "population": 5000000}}
>>> data = simplejson.loads(s)
>>> print data['Denmark']['towns'][1]
Århus
```

JSON data structures are mapped to corresponding Python structures.

. . . Reading JSON

MediaWikis may export some their data in JSON format, and here is an example with Wikipedia querying for an embedded "template":

```
import urllib, simplejson
url = "http://en.wikipedia.org/w/api.php?" + \
      "action=query&list=embeddedin&" + \
      "eititle=Template:Infobox_Single_nucleotide_polymorphism&" + \
      "format=json"
data = simplejson.load(urllib.urlopen(url))
data['query']['embeddedin'][0]
```

gives `{u'ns': 0, u'pageid': 238300, u'title': u'Factor V Leiden'}`

Here the Wikipedia article *Factor V Leiden* contains (has embedded) the template *Infobox_Single_nucleotide_polymorphism*

(Note MediaWiki may need to be called several times for the retrieval of all results for the query by using `data['query-continue']`)

Regular expressions with `re` ...

```
>>> import re
>>> s = 'The following is a link to <a href="http://www.dtu.dk">DTU</a>'
```

Substitute "`<... some text ...>`" with an empty string with `re.sub()`

```
>>> re.sub('<.*?>', '', s)
'The following is a link to DTU'
```

Escaping non-alphanumeric characters in a string:

```
>>> print re.escape(u'Escape non-alphanumerics  ", \, #, Å and =')
Escape\ non\-alphanumerics\ \ \"\,\ \\\,\ \#\,\ \Å\ and\ \=
```

XML-like matching with the named group `<(?P<name>...)>` construct:

```
>>> s = '<name>Ole</name><name>Lars</name>'
>>> re.findall('<(?P<tag>\w+)>(.*?)</(?P=tag)>', s)
[('name', 'Ole'), ('name', 'Lars')]
```

. . . Regular expressions with `re` . . .

Non-greedy match of content of a `<description>` tag:

```
>>> s = """<description>This is  a
multiline string.</description>"""
>>> re.search('<description>(.+?)</description>', s, re.DOTALL).groups()
('This is  a \nmultiline string.',)
```

Find Danish telephone numbers in a string with initial `compile()`:

```
>>> s = '(+45) 45253921  4525 39 21  2800  45 45 25 39 21'
>>> r = re.compile(r'((?:(?:\(\+?\d{2,3}\))|\+?\d{2,3})?(?: ?\d){8})')
>>> r.search(s).group()
'(+45) 45253921'
>>> r.findall(s)
['(+45) 45253921', ' 4525 39 21', ' 45 45 25 39']
```

. . . Regular expressions with `re`

Unicode letter match with `[^\W\d_]+` meaning one or more not non-alphanumeric and not digits and not underscore (`\xc5` is unicode "Å"):

```
>>> re.findall('[^\W\d_]+', u'F._Å._Nielsen', re.UNICODE)
[u'F', u'\xc5', u'Nielsen']
```

Matching the word immediately after "the" regardless of case:

```
>>> s = 'The dog, the cat and the mouse in the USA'
>>> re.findall('the ([a-z]+)', s, re.IGNORECASE)
['dog', 'cat', 'mouse', 'USA']
```

Reading HTML ...

HTML contains tags and content. There are several ways to strip the content.

1. Simple regular expression, e.g., `re.sub('<.*?>', '', s)`
2. `htmllib` module with the `formatter` module.
3. Use `nltk.clean_html()` (Bird et al., 2009, p. 82). This function uses `HTMLParser`
4. `BeautifulSoup` module is a robust HTML parser (Segaran, 2007, p. 45+).
5. `lxml.etree.HTML`

. . . Reading HTML

The `htmllib` can parse HTML documents (Martelli, 2006, p. 580+)

```
import htmllib, formatter, urllib

p = htmllib.HTMLParser(formatter.NullFormatter())
p.feed(urllib.urlopen('http://www.dtu.dk').read())
p.close()
for url in p.anchorlist: print url
```

The result is a printout of the list of URL from 'http://www.dtu.dk':

```
/English.aspx
/Service/Indeks.aspx
/Service/Kontakt.aspx
/Service/Telefonbog.aspx
http://www.alumne.dtu.dk
http://portalen.dtu.dk
...
```

Robust HTML reading ...

Consider an HTML file, `test.html`, with an error:

```
<html>
  <body>
    <h1>Here is an error</h1
    A &gt; is missing
    <h2>Subsection</h2>
  </body>
</html>
```

Earlier versions of `nltk` and `HTMLParser` would generate error. NLTK can now handle it:

```
>>> import nltk
>>> nltk.clean_html(open('test.html').read())
'Here is an error Subsection'
```

. . . Robust HTML reading

`BeautifulSoup` survives the missing “>” in the end tag:

```
>>> from BeautifulSoup import BeautifulSoup as BS
>>> html = open('test.html').read()
>>> BS(html).findAll(text=True)
[u'\n', u'\n', u'Here is an error', u'Subsection', u'\n', u'\n', u'\n']
```

Another example with extraction of links from http://dtu.dk:

```
>>> from urllib2 import urlopen
>>> html = urlopen('http://dtu.dk').read()
>>> ahrefs = BS(html).findAll(name='a', attrs={'href': True})
>>> urls = [dict(a.attrs)['href'] for a in ahrefs]
>>> urls[0:3]
[u'/English.aspx', u'/Service/Indeks.aspx', u'/Service/Kontakt.aspx']
```

Reading XML

`xml.dom`: Document Object Model. With `xml.dom.minidom`

`xml.sax`: Simple API for XML (and an obsolete `xmllib`)

`xml.etree`: ElementTree XML library

Example with `minidom` module with searching on a tag name:

```
>>> s = """<persons> <person> <name>Ole</name> </person>
                     <person> <name>Jan</name> </person> </persons>"""
>>> import xml.dom.minidom
>>> dom = xml.dom.minidom.parseString(s)
>>> for element in dom.getElementsByTagName("name"):
...   print(element.firstChild.nodeValue)
...
Ole
Jan
```

Reading XML: traversing the elements

```
>>> s = """<persons>
  <person id="1"> <name>Ole</name> <topic>Bioinformatics</topic> </person>
  <person id="2"> <name>Jan</name> <topic>Signals</topic> </person>
</persons>"""

>>> import xml.etree.ElementTree
>>> x = xml.etree.ElementTree.fromstring(s)
>>> [x.tag, x.text, x.getchildren()[0].tag, x.getchildren()[0].attrib,
... x.getchildren()[0].text, x.getchildren()[0].getchildren()[0].tag,
... x.getchildren()[0].getchildren()[0].text]
['persons', '\n  ', 'person', {'id': '1'}, ' ', 'name', 'Ole']

>>> import xml.dom.minidom
>>> y = xml.dom.minidom.parseString(s)
>>> [y.firstChild.nodeName, y.firstChild.firstChild.nodeValue,
y.firstChild.firstChild.nextSibling.nodeName]
[u'persons', u'\n  ', u'person']
```

Other xml packages: `lxml` and `BeautifulSoup`

Outside the Python standard library (with the `xml` packages) is `lxml` package.

`lxml`'s documentation claims that `lxml.etree` is much faster than `ElementTree` in the standard `xml` package.

Also note that BeautifulSoup will read xml files.

Generating HTML ...

The simple way:

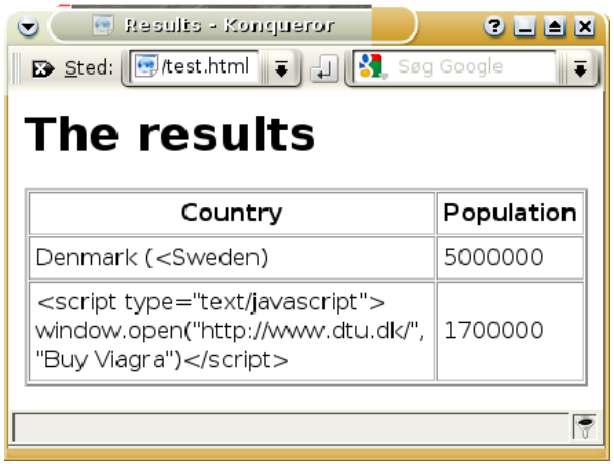
```
>>> results = [('Denmark', 5000000), ('Botswana', 1700000)]
>>> res = '<tr>'.join([ '<td>%s<td>%d' % (r[0], r[1]) for r in results ])
>>> s = """<html><head><title>Results</title></head>
<body><table>%s</table></body></html>""" % res
>>> s
'<html><head><title>Results</title></head>\n<body><table><td>Denmark
<td>5000000<tr><td>Botswana<td>1700000</table></body></html>'
```

If the input is not known it may contain parts needing escapes:

```
>>> results = [('Denmark (<Sweden)', 5000000), (r'''<script
type="text/javascript"> window.open("http://www.dtu.dk/", "Buy
Viagra")</script>''', 1700000)]
>>> open('test.html', 'w').write(s)
```

Input should be sanitized and output should be escaped.

. . . Generating HTML the outdated way



Writing an HTML file with the `HTMLgen` module and the code below will generate a HTML file as shown to the left.

Another HTML generation module is Richard Jones' `html` module (http://pypi.python.org/pypi/html), and see also the `cgi.escape()` function.

```
import HTMLgen
doc = HTMLgen.SimpleDocument(title="Results")
doc.append(HTMLgen.Heading(1, "The results"))
table = HTMLgen.Table(heading=["Country", "Population"])
table.body = [[ HTMLgen.Text('%s' % r[0]), r[1] ] for r in results ]
doc.append(table)
doc.write("test.html")
```

Better way for generating HTML

Probably a better way to generate HTML is with one of the many template engine modules, e.g., `Cheetah` (see example in CherryPy documentation), Django (obviously for Django), Jinja2, Mako, `tornado.template` (for Tornado), ...

Jinja2 example:

```
>>> from jinja2 import Template
>>> tmpl = Template(u"""<html><body><h1>{{ name|escape }}</h1>
                       </body></html>""")
>>> tmpl.render(name = u"Finn <Årup> Nielsen")
u'<html><body><h1>Finn &lt;\xc5rup&gt; Nielsen</h1></body></html>'
```

Natural language Toolkit

Natural Language Toolkit (NLTK) described in the book (Bird et al., 2009) and included with "`import nltk`" and it contains data and a number of classes and functions:

`nltk.corpus`: standard natural language processing corpora

`nltk.tokenize`, `nltk.stem`: sentence and words segmentation and stemming or lemmatization

`nltk.tag`: part-of-speech tagging

`nltk.classify`, `nltk.cluster`: supervized and unsupervized classification

. . . And a number of other moduls: `nltk.collocations`, `nltk.chunk`, `nltk.parse`, `nltk.sem`, `nltk.inference`, `nltk.metrics`, `nltk.probability`, `nltk.app`, `nltk.chat`

Splitting words: Word tokenization . . .

```
>>> s = """To suppose that the eye with all its inimitable contrivances
for adjusting the focus to different distances, for admitting
different amounts of light, and for the correction of spherical and
chromatic aberration, could have been formed by natural selection,
seems, I freely confess, absurd in the highest degree."""

>>> s.split()
['To', 'suppose', 'that', 'the', 'eye', 'with', 'all', 'its',
'inimitable', 'contrivances', 'for', 'adjusting', 'the', 'focus',
'to', 'different', 'distances,', ...

>>> re.split('\W+', s)                    # Split on non-alphanumeric
['To', 'suppose', 'that', 'the', 'eye', 'with', 'all', 'its',
'inimitable', 'contrivances', 'for', 'adjusting', 'the', 'focus',
'to', 'different', 'distances', 'for',
```

. . . Splitting words: Word tokenization . . .

A text example from Wikipedia with numbers

```
>>> s = """Enron Corporation (former NYSE ticker symbol ENE) was an
American energy company based in Houston, Texas. Before its bankruptcy
in late 2001, Enron employed approximately 22,000[1] and was one of
the world's leading electricity, natural gas, pulp and paper, and
communications companies, with claimed revenues of nearly $101 billion
in 2000."""
```

For `re.split('\W+', s)` there is a problem with genetive (`world's`) and numbers (22,000)

. . . Splitting words: Word tokenization . . .

Word tokenization inspired from (Bird et al., 2009, page 111)

```
>>> pattern = r"""(?ux)                    # Set Unicode and verbose flag
    (?:[^\W\d_]\.)+                         # Abbreviation
  | [^\W\d_]+(?:-[^\W\d_])*(?:'s)?          # Words with optional hyphens
  | \d{4}                                   # Year
  | \d{1,3}(?:,\d{3})*                      # Number
  | \$\d+(?:\.\d{2})?                       # Dollars
  | \d{1,3}(?:\.\d+)?\s%                    # Percentage
  | \.\.\.                                  # Ellipsis
  | [.,;"'?!():-_`/]                        #
  """
>>> import re
>>> re.findall(pattern, s)
>>> import nltk
>>> nltk.regexp_tokenize(s, pattern)
```

. . . Splitting words: Word tokenization . . .

From informal quickly written text (YouTube):

```
>>> s = u"""Det er SÅ LATTERLIGT/PLAT!! -Det har jo ingen sammenhæng
med, hvad DK repræsenterer!! ARGHHH!!"""

>>> re.findall(pattern, s)
[u'Det', u'er', u'S\xc5', u'LATTERLIGT', u'/', u'PLAT', u'!', u'!',
u'Det', u'har', u'jo', u'ingen', u'sammenh\xe6ng', u'med', u',',
u'hvad', u'DK', u'repr\xe6senterer', u'!', u'!', u'ARGHHH', u'!',
u'!']
```

Problem with emoticons such as “:o(”: They are not treated as a single “word”.

Difficult to construct a general tokenizer.

Word normalization . . .

Converting “talking”, “talk”, “talked”, “Talk”, etc. to the lexeme “talk” (Bird et al., 2009, page 107)

```
>>> porter = nltk.PorterStemmer()
>>> [porter.stem(t.lower()) for t in tokens]
['to', 'suppos', 'that', 'the', 'eye', 'with', 'all', 'it', 'inimit',
'contriv', 'for', 'adjust', 'the', 'focu', 'to', 'differ', 'distanc',
',', 'for', 'admit', 'differ', 'amount', 'of', 'light', ',', 'and',
```

Another stemmer is `lancaster.stem()`

The Snowball stemmer works for non-English, e.g.,

```
>>> from nltk.stem.snowball import SnowballStemmer
>>> stemmer = SnowballStemmer("danish")
>>> stemmer.stem('universiteterne')
'universitet'
```

. . . Word normalization

Normalize with a word list (WordNet):

```
>>> wnl = nltk.WordNetLemmatizer()
>>> [wnl.lemmatize(token) for token in tokens]
['To', 'suppose', 'that', 'the', 'eye', 'with', 'all', 'it',
'inimitable', 'contrivance', 'for', 'adjusting', 'the', 'focus', 'to',
'different', 'distance', ',', 'for', 'admitting', 'different',
'amount', 'of', 'light', ',', 'and', 'for', 'the', 'correction',
```

Here words “contrivances” and “distances” have lost the plural “s” and “its” the genitive “s”.

Word categories

Part-of-speech tagging with NLTK

```
>>> words = nltk.word_tokenize(s)
>>> nltk.pos_tag(words)
[('To', 'TO'), ('suppose', 'VB'), ('that', 'IN'), ('the', 'DT'),
('eye', 'NN'), ('with', 'IN'), ('all', 'DT'), ('its', 'PRP$'),
('inimitable', 'JJ'), ('contrivances', 'NNS'), ('for', 'IN'),
```

`NN` noun, `VB` verb, `JJ` adjective, `RB` adverb, etc., see, common tags.

```
>>> tagged = nltk.pos_tag(words)
>>> [word for (word, tag) in tagged if tag=='JJ']
['inimitable', 'different', 'different', 'light', 'spherical',
'chromatic', 'natural', 'confess', 'absurd']
```

"confess" is wrongly tagged.

Some examples

Keyword extraction . . .

Consider the text:

> "Computer programming (e.g., 02101 or 02102), statistics (such as 02323, 02402 or 02403) and linear algebra (such as 01007) More advanced programming and data analysis, e.g., Machine Learning (02450 or 02457), or courses such as 02105 or 01917"

We want to extract "computer programming", "statistics", "linear algebra" "advanced programming" (or perhaps just "programming"!?), "data analysis", "machine learning".

But we do not want "and linear" or "courses such", i.e., not just bigrams.

Note the lack of verbs and the missing period.

. . . Keyword extraction . . .

Lets see what NLTK's part-of-speech tagger can do:

```
>>> text = ("Computer programming (e.g., 02101 or 02102), statistics "
        "(such as 02323, 02402 or 02403) and linear algebra (such as 01007) "
        "More advanced programming and data analysis, e.g., Machine Learning "
         "(02450 or 02457), or courses such as 02105 or 01917")
>>> tagged = nltk.pos_tag(nltk.word_tokenize(text))
>>> tagged
[('Computer', 'NN'), ('programming', 'NN'), ('(', ':'), ('e.g.',
'NNP'), (',', ','), ('02101', 'CD'), ('or', 'CC'), ('02102', 'CD'),
(')', 'CD'), (',', ','), ('statistics', 'NNS'), ('(', 'VBP'), ('such',
'JJ'), ('as', 'IN'), ('02323', 'CD'), (',', ','), ('02402', 'CD'),
('or', 'CC'), ('02403', 'CD'), (')', 'CD'), ('and', 'CC'), ('linear',
'JJ'), ('algebra', 'NN'), ('(', ':'), ('such', 'JJ'), ('as', 'IN'),
('01007', 'CD'), (')', 'CD'), ('More', 'NNP'), ('advanced', 'VBD'),
('programming', 'VBG'), ('and', 'CC'), ('data', 'NNS'), ('analysis',
'NN'), (',', ','), ('e.g.', 'NNP'), (',', ','), ('Machine', 'NNP'),
('Learning', 'NNP'), ('(', 'NNP'), ('02450', 'CD'), ('or', 'CC'),
('02457', 'CD'), (')', 'CD'), (',', ','), ('or', 'CC'), ('courses',
'NNS'), ('such', 'JJ'), ('as', 'IN'), ('02105', 'CD'), ('or', 'CC'),
('01917', 'CD')]
```

Note an embarrassing error: `('(', 'NNP')`.

. . . Keyword extraction . . .

Idea: assemble consecutive nouns here a first attempt:

```
phrases, phrase = [], ""
for (word, tag) in tagged:
    if tag[:2] == 'NN':
        if phrase == "": phrase = word
        else: phrase += " " + word
    elif phrase != "":
        phrases.append(phrase.lower())
        phrase = ""
```

Result:

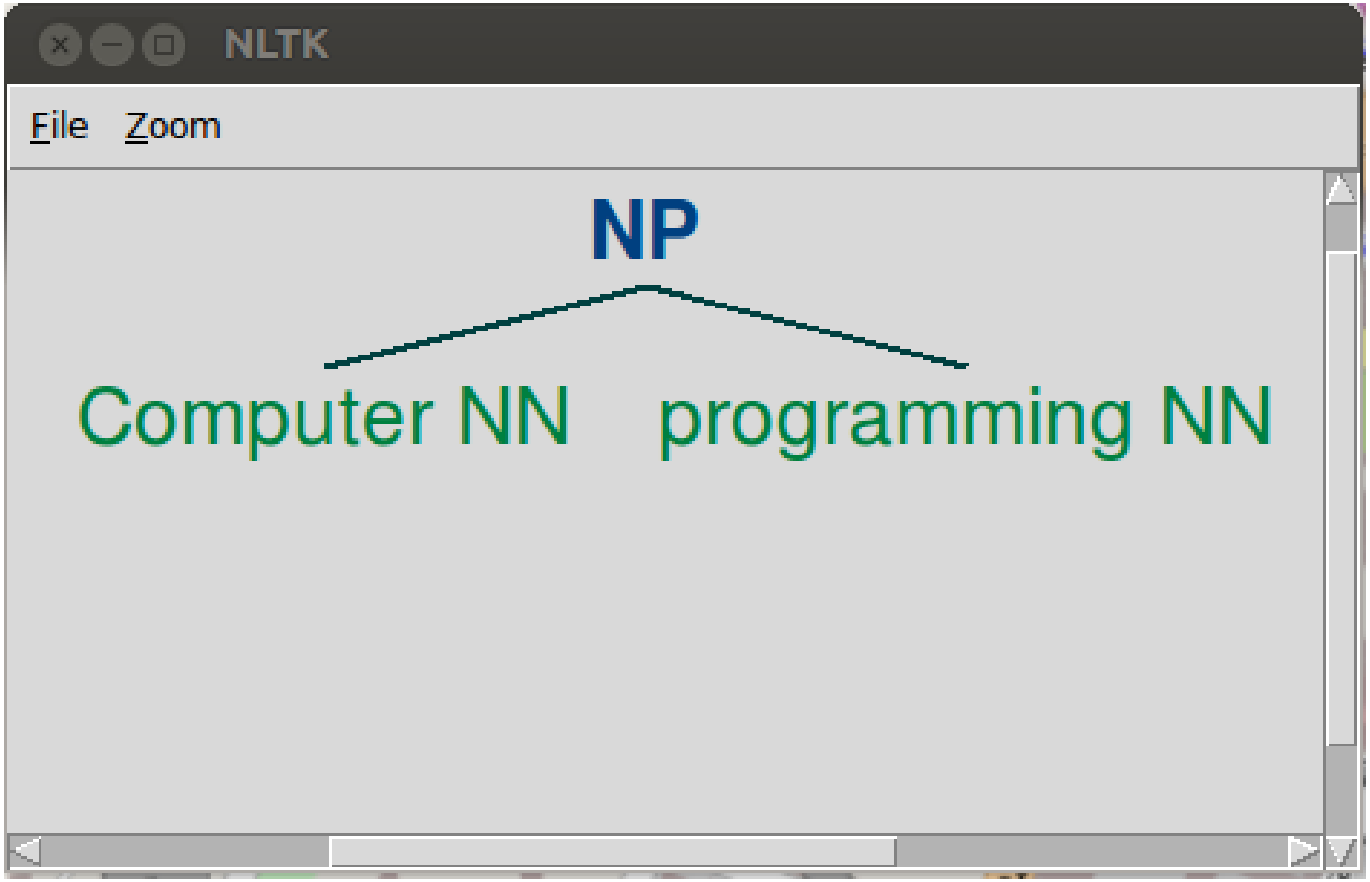
```
>>> phrases
['computer programming', 'e.g.', 'statistics', 'algebra', 'programming',
'data analysis', 'e.g.', 'machine learning (', 'courses']
```

Well . . . Not quite right. More control structures, stopword lists, . . . ?

. . . Keyword extraction . . .

Chunking: Make a small grammar with regular expression that, e.g., catch a sentence part, here we call it a noun phrase (NP):

```
>>> grammar = "NP: { <JJ>*<NN.?>+ }"
>>> cp = nltk.RegexpParser(grammar)
>>> cp.parse(tagged)
Tree('S', [Tree('NP', [('Computer', 'NN'), ('programming', 'NN')]),
('(', ':'), Tree('NP', [('e.g.', 'NNP')]), (',', ','), ('02101',
'CD'), ('or', 'CC'), ('02102', 'CD'), (')', 'CD'), (',', ','),
Tree('NP', [('statistics', 'NNS')]), ('(', 'VBP'), ('such', 'JJ'),
('as', 'IN'), ('02323', 'CD'), (',', ','), ('02402', 'CD'), ('or',
'CC'), ('02403', 'CD'), (')', 'CD'), ('and', 'CC'), Tree('NP',
...
```



NLTK can produce parse trees. Here the first chunk:

```
>>> list(cp.parse(tagged))[0].draw()
```

. . . Keyword extraction . . .

Extract the NP parts:

```
def extract_chunks(tree, filter='NP'):
    extract_word = lambda leaf: leaf[0].lower()
    chunks = []
    if hasattr(tree, 'node'):
        if tree.node == filter:
            chunks = [ " ".join(map(extract_word, tree.leaves())) ]
        else:
            for child in tree:
                cs = extract_chunks(child, filter=filter)
                if cs != []:
                    chunks.append(cs[0])
    return chunks
```

```
>>> extract_chunks(cp.parse(tagged))
['computer programming', 'e.g.', 'statistics', 'linear algebra',
'more', 'data analysis', 'e.g.', 'machine learning (', 'courses']
```

Still not quite right.

Checking keyword extraction on new data set

```
text = """To give an introduction to advanced time series
analysis. The primary goal to give a thorough knowledge on modelling
dynamic systems. Special attention is paid on non-linear and
non-stationary systems, and the use of stochastic differential
equations for modelling physical systems. The main goal is to obtain a
solid knowledge on methods and tools for using time series for setting
up models for real life systems."""

process = lambda sent: extract_chunks(cp.parse(
              nltk.pos_tag(nltk.word_tokenize(sent))))
map(process, nltk.sent_tokenize(text))
```

```
[['introduction', 'advanced time series analysis'], ['primary goal',
'thorough knowledge', 'modelling', 'dynamic systems'], ['special
attention', 'non-stationary systems', 'use', 'stochastic differential
equations', 'physical systems'], ['main goal', 'solid knowledge',
'methods', 'tools', 'time series', 'models', 'real life systems']]
```

Web crawling with `htmllib` & co.

```
import htmllib, formatter, urllib, urlparse

k = 1
urls = {}
todownload = set(['http://www.dtu.dk'])
while todownload:
  url0 = todownload.pop()
  urls[url0] = set()
  try:
    p = htmllib.HTMLParser(formatter.NullFormatter())
    p.feed(urllib.urlopen(url0).read())
    p.close()
  except:
    continue
  for url in p.anchorlist:
```

```
    urlparts = urlparse.urlparse(url)
    if not urlparts[0] and not urlparts[1]:
      urlparts0 =  urlparse.urlparse(url0)
      url = urlparse.urlunparse((urlparts0[0], urlparts0[1],
        urlparts[2], '', '', ''))
    else:
      url = urlparse.urlunparse((urlparts[0], urlparts[1],
        urlparts[2], '', '', ''))
    urlparts = urlparse.urlparse(url)
    if urlparts[1][-7:] != '.dtu.dk': continue  # Not DTU
    if urlparts[0] != 'http': continue          # Not Web
    urls[url0] = urls[url0].union([url])
    if url not in urls:
      todownload = todownload.union([url])
  k += 1
  print("%4d %4d %s" % (k, len(todownload), url0))
  if k > 1000: break
```

Scrapy: crawl and scraping framework

```
$ scrapy startproject dtu
```

File `dtu/dtu/spiders/dtu_spider.py` can, e.g., contain

```
from scrapy.contrib.spiders import CrawlSpider, Rule
from scrapy.contrib.linkextractors.sgml import SgmlLinkExtractor

class DtuSpider(CrawlSpider):
  name = "dtu"
  allowed_domains = ["dtu.dk"]
  start_urls = ["http://www.dtu.dk"]
  rules = (Rule(SgmlLinkExtractor(), callback="parse_items", follow=True),)
  def parse_items(self, response):
    print(response.url)
```

Scrapy: crawl and scraping framework

```
$ scrapy crawl --nolog dtu
http://www.aqua.dtu.dk/
http://www.dtu.dk/Uddannelse/Efteruddannelse/Kurser
http://www.dtu.dk/Uddannelse/Studieliv
http://www.dtu.dk/Uddannelse/Phd
```

Scrapy can extract items on the page with xpath-like methods

```
from scrapy.selector import HtmlXPathSelector

# In the parse method:
hxs = HtmlXPathSelector(response)
links = hxs.select('//a/@href').extract()
```

YouTube with `gdata`

`gdata` is a package for reading some of the Google data APIs. One such is the YouTube API (`gdata.youtube`). It allows, e.g., to fetch comments to videos on youtube (Giles Bowkett). Some comments from a video:

```
>>> import gdata.youtube.service
>>> yts = gdata.youtube.service.YouTubeService()
>>> ytfeed = yts.GetYouTubeVideoCommentFeed(video_id="pXhcPJK5cMc")
>>> comments = [comment.content.text for comment in ytfeed.entry]
>>> print(comments[8])
04:50 and 07:20 are priceless. Docopt  is absolutely amazing!
```

Note the number of comments you download each time is limited.

. . . YouTube with gdata

Often with these kind of web-services you need to iterate to get all data

```
import gdata.youtube.service
yts = gdata.youtube.service.YouTubeService()
urlpattern = ("http://gdata.youtube.com/feeds/api/videos/"
              "9arOTF7J5f0/comments?start-index=%d&max-results=25")
index = 1
url = urlpattern % index
comments = []
while True:
    ytfeed = yts.GetYouTubeVideoCommentFeed(uri=url)
    comments.extend([comment.content.text for comment in ytfeed.entry])
    if not ytfeed.GetNextLink(): break
    url = ytfeed.GetNextLink().href
```

Issues: Store comments in a structured format, take care of Exceptions.

MediaWiki

For MediaWikis (e.g., Wikipedia) look at Pywikipediabot

Download and setup "user-config.py"

Here I have setup a configuration for wikilit.referata.com

```
>>> import pywikibot

>>> site = pywikibot.Site('en', 'wikilit')
>>> pagename = "Chitu Okoli"
>>> wikipage = pywikibot.Page(site, pagename)
>>> text = wikipage.get(get_redirect = True)
u'{{Researcher\n|name=Chitu Okoli\n|surname=Okoli\n|affiliat ...
```

There is also a `wikipage.put` for writing on the wiki.

Reading the XML from the Brede Database

XML files for the Brede Database (Nielsen, 2003) use no attributes, no empty nor mixed elements. "Elements-only" elements has initial caps.

```
>>> s = """<Rois>
  <Roi>
    <name>Cingulate</name>
    <variation>Cingulate gyrus</variation>
    <variation>Cingulate cortex</variation>
  </Roi>
  <Roi>
    <name>Cuneus</name>
  </Roi>
</Rois>"""
```

May be mapped to dictionary with lists with dictionaries with lists...

```
dict(Rois=[dict(Roi=[dict(name=['Cingulate'], variation=['Cingulate
gyrus', 'Cingulate cortex']), dict(name=['Cuneus'])])])
```

Reading the XML from the Brede Database

Parsing the XML with `xml.dom`

```
>>> from xml.dom.minidom import parseString
>>> dom = parseString(s)
>>> data = xmlrecursive(dom.documentElement)      # Custom function
>>> data
{'tag': u'Rois', 'data': {u'Roi': [{u'name': [u'Cingulate'],
u'variation': [u'Cingulate gyrus', u'Cingulate cortex']}, {u'name':
[u'Cuneus']}]}}
```

This maps straightforward to JSON:

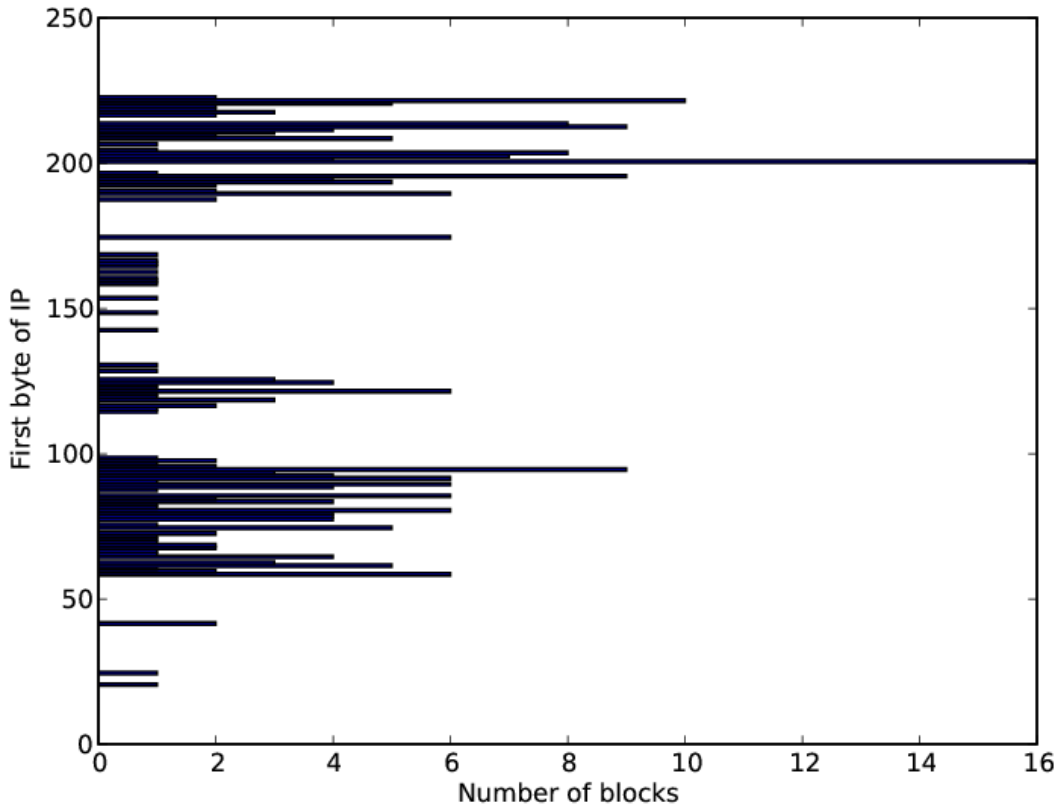
```
>>> import json
>>> json.dumps(data)
'{"tag": "Rois", "data": {"Roi": [{"name": ["Cingulate"], "variation":
["Cingulate gyrus", "Cingulate cortex"]}, {"name": ["Cuneus"]}]}}'
```

Reading the XML from the Brede Database

```
import string
def xmlrecursive(dom):
  tag = dom.tagName
  if tag[0] == string.upper(tag[0]):    # Elements-only elements
    data = {};
    domChild = dom.firstChild.nextSibling
    while domChild != None:
      o = xmlrecursive(domChild)
      if o['tag'] in data:
        data[o['tag']].append(o['data'])
      else :
        data[o['tag']] = [ o['data'] ]
      domChild = domChild.nextSibling.nextSibling
  else:                                 # Text-only elements
    if dom.firstChild:   data = dom.firstChild.data
    else:    data = ''
  return { 'tag': tag, 'data': data }
```

Statistics on blocked IPs in a MediaWiki . . .

Example with URL download, JSON processing, simple regular expression and plotting with matplotlib:



. . . Statistics on blocked IPs in a MediaWiki

```
from pylab import *
from urllib2 import urlopen
from simplejson import load
from re import findall

url = 'http://neuro.compute.dtu.dk/w/api.php?' + \
      'action=query&list=blocks&' + \
      'bkprop=id|user|by|timestamp|expiry|reason|range|flags&' + \
      'bklimit=500&format=json'
data = load(urlopen(url))
users = [block['user'] for block in data['query']['blocks'] if 'user'
in block]
ip_users = filter(lambda s: findall(r'^\d+', s), users)
ip = map(lambda s: int(findall(r'\d+', s)[0]), ip_users)
dummy = hist(ip, arange(256), orientation='horizontal')
xlabel('Number of blocks'); ylabel('First byte of IP')
show()
```

Email mining . . .

Read in a small email data set with three classes, “conference”, “job” and “spam” (Szymkowiak et al., 2001; Larsen et al., 2002b; Larsen et al., 2002a; Szymkowiak-Have et al., 2006):

```
documents = [dict(
            email=open("conference/%d.txt" % n).read().strip(),
            category='conference') for n in range(1,372)]
documents.extend([ dict(
            email=open("job/%d.txt" % n).read().strip(),
            category='job') for n in range(1,275)])
documents.extend([ dict(
            email=open("spam/%d.txt" % n).read().strip(),
            category='spam') for n in range(1,799)])
```

Now the data is contained in `documents[i]['email']` and the category in `documents[i]['category']`.

. . . Email mining . . .

Parse the emails with the `email` module and maintain the body text, strip the HTML tags (if any) and split the text into words:

```
from email import message_from_string
from BeautifulSoup import BeautifulSoup as BS
from re import split
for n in range(len(documents)):
  html = message_from_string(documents[n]['email']).get_payload()
  while not isinstance(html, str):                    # Multipart problem
    html = html[0].get_payload()
  text = ' '.join(BS(html).findAll(text=True))         # Strip HTML
  documents[n]['html'] = html
  documents[n]['text'] = text
  documents[n]['words'] = split('\W+', text)           # Find words
```

. . . Email mining . . .

Document classification a la (Bird et al., 2009, p. 227+) with NLTK:

```
import nltk
all_words = nltk.FreqDist(w.lower() for d in documents for w in d['words'])
word_features = all_words.keys()[:2000]
```

`word_features` now contains the 2000 most common words across the corpus. This variable is used to define a feature extractor:

```
def document_features(document):
  document_words = set(document['words'])
  features = {}
  for word in word_features:
    features['contains(%s)' % word] = (word in document_words)
  return features
```

Each document has now an associated dictionary with True or False on whether a specific word appear in the document

. . . Email mining . . .

Scramble the data set to mix conference, job and spam email:

```
import random
random.shuffle(documents)
```

Build variable for the functions of NLTK:

```
featuresets = [(document_features(d), d['category']) for d in documents]
```

Split the 1443 emails into training and test set:

```
train_set, test_set = featuresets[721:], featuresets[:721]
```

Train a "naive Bayes classifier" (Bird et al., 2009, p. 247+):

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

. . . Email mining

Classifier performance evaluated on the test set and show features (i.e., words) important for the classification:

```
>>> classifier.classify(document_features(documents[34]))
'spam'
>>> documents[34]['text'][:60]
u'BENCHMARK PRINT SUPPLY\nLASER PRINTER CARTRIDGES JUST FOR YOU'
>>> print nltk.classify.accuracy(classifier, test_set)
0.890429958391
>>> classifier.show_most_informative_features(4)
Most Informative Features
    contains(candidates) = True              job : spam   =     75.2 : 1.0
 contains(presentations) = True           confer : spam   =     73.6 : 1.0
      contains(networks) = True           confer : spam   =     70.4 : 1.0
       contains(science) = True              job : spam   =     69.0 : 1.0
```

More information

Recursively Scraping Web Pages With Scrapy, tutorial by Michael Herman.

Text Classification for Sentiment Analysis — Naive Bayes Classifier by Jacob Perkins.

Summary

For web crawling there are the basic tools of urllibs and requests.

For extraction and parsing of content Python has, e.g., regular expression handling in the re module and BeautifulSoup.

There are specialized modules for json, feeds and XML.

Scrapy is a large framework for crawling and extraction.

The NLTK package contains numerous natural language processing methods: sentence and word tokenization, part-of-speech tagging, chunking, classification, . . .

References

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly, Sebastopol, California. ISBN 9780596516499.

Larsen, J., Hansen, L. K., Have, A. S., Christiansen, T., and Kolenda, T. (2002a). Webmining: learning from the world wide web. *Computational Statistics & Data Analysis*, 38(4):517–532. DOI: 10.1016/S0167-9473(01)00076-7.

Larsen, J., Szymkowiak, A., and Hansen, L. K. (2002b). Probabilistic hierarchical clustering with labeled and unlabeled data. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 6(1):56–62. http://isp.imm.dtu.dk/publications/2001/larsen.kes.pdf.

Martelli, A. (2006). *Python in a Nutshell*. In a Nutshell. O'Reilly, Sebastopol, California, second edition.

Martelli, A., Ravenscroft, A. M., and Ascher, D., editors (2005). *Python Cookbook*. O'Reilly, Sebastopol, California, 2nd edition.

Nielsen, F. Å. (2003). The Brede database: a small database for functional neuroimaging. *NeuroImage*, 19(2). http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/2879/pdf/imm2879.pdf. Presented at the 9th International Conference on Functional Mapping of the Human Brain, June 19–22, 2003, New York, NY.

Pilgrim, M. (2004). *Dive into Python*.

Segaran, T. (2007). *Programming Collective Intelligence*. O'Reilly, Sebastopol, California.

Szymkowiak, A., Larsen, J., and Hansen, L. K. (2001). Hierarchical clustering for datamining. In Babs, N., Jain, L. C., and Howlett, R. J., editors, *Proceedings of KES-2001 Fifth International Conference on Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies*, pages 261–265. http://isp.imm.dtu.dk/publications/2001/szymkowiak.kes2001.pdf.

Szymkowiak-Have, A., Girolami, M. A., and Larsen, J. (2006). Clustering via kernel decomposition. *IEEE Transactions on Neural Networks*, 17(1):256–264. http://eprints.gla.ac.uk/3682/01/symoviak3682.pdf.

Index