Processing Raw Text
POS Tagging

Marina Sedinkina
- Folien von Desislava Zhekova

CIS, LMU
marina.sedinkina@campus.lmu.de

January 16, 2018
Outline

1. Accessing Text beyond NLTK
2. Processing Raw Text
3. POS Tagging
NLTK includes a good selection of various corpora among which a small selection of texts from the Project Gutenberg electronic text archive. Project Gutenberg contains more than 50,000 free electronic books, hosted at http://www.gutenberg.org.
Unfortunately, only 18 books are provided, which you can list as we have seen before:

```python
>>> import nltk
>>> nltk.corpus.gutenberg.fileids()
```
Accessing the original collection is thus helpful:

```python
import nltk
import urllib

url = "http://www.gutenberg.org/files/2554/2554-0.txt"
urlData = urllib.request.urlopen(url)
firstLine = urlData.readline().decode("utf-8")
print(firstLine)

# prints
# The Project Gutenberg EBook of Crime and Punishment
# by Fyodor Dostoevsky
```
Gutenberg eBooks

2554 = Crime and Punishment, by Fyodor Dostoevsky

How do I find out the Book IDs?
Accessing Text beyond NLTK
Processing Raw Text
POS Tagging

Dealing with other formats
HTML
Binary formats

Gutenberg Corpus

Directly from the Gutenberg Webpage → not very efficient.

Free ebooks - Project Gutenberg
From Project Gutenberg, the first producer of free ebooks.

New Kindle Fire Review
Before you buy: Read our Webmaster's review of the new Kindle Fire.

Some of Our Latest Books

Welcome
Gutenberg Corpus

Under the link http://www.gutenberg.org/dirs/GUTINDEX.ALL, the project provides a file listing all available books and their IDs.

**** The Language of the eBooks is English, unless otherwise noted ****

~ ~ ~ ~ Posting Dates for the below eBooks: 1 Nov 2013 to 30 Nov 2013 ~ ~ ~ ~

<table>
<thead>
<tr>
<th>TITLE and AUTHOR</th>
<th>ETEXT NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ypres 1914, by Otto Schwink</td>
<td>44234</td>
</tr>
<tr>
<td>[Subtitle: An Official Account Published by Order of the German General Staff]</td>
<td></td>
</tr>
<tr>
<td>[Translator: Graeme Chamley Wynne]</td>
<td></td>
</tr>
<tr>
<td>Plays by August Strindberg, Third Series, by August Strindberg</td>
<td>44233</td>
</tr>
<tr>
<td>[Translator: Edwin Bj?rkman]</td>
<td></td>
</tr>
<tr>
<td>L'Illustration, No. 1608, 20 d?cembre 1873, by Various</td>
<td>44232</td>
</tr>
<tr>
<td>[Language: French]</td>
<td></td>
</tr>
<tr>
<td>The Miraculous Medal, by Jean Marie Aladel</td>
<td>44231</td>
</tr>
<tr>
<td>[Subtitle: Its Origin, History, Circulation, Results]</td>
<td></td>
</tr>
</tbody>
</table>
And as you can see Crime and Punishment, by Fyodor Dostoevsky is also listed there:

Under the Redwoods, by Bret Harte

Contents:
Jimmy's Big Brother From California
The Youngest Miss Piper
A Widow Of The Santa Ana Valley
The Mermaid Of Lighthouse Point
Under The Eaves
How Reuben Allen "Saw Life" In San Francisco
Three Vagabonds Of Trinidad
A Vision Of The Fountain
A Romance Of The Line
Bohemian Days In San Francisco
Under The Redwoods

Crime and Punishment, by Fyodor Dostoevsky
[Tr.: Constance Garnett]

Jeanne d'Arc, by Mrs. (Margaret) Oliphant
[Subtitle: Her Life And Death]

Thankful's Inheritance, by Joseph C. Lincoln

Droll Stories, Volume 3, by Honore de Balzac
(See also: #2318 & #1925)

So what more can we find out?
Recall: Start of String and End of String Anchors:

- `^` matches the position before the first character in the string → Applying `^a` to `abc` matches `a`.
- Similarly, `$` matches right after the last character in the string → `c$` matches `c` in `abc`.

Check regex via [https://regex101.com/](https://regex101.com/)
Lookahead and lookbehind, collectively called "lookaround", are zero-length assertions just like start and end of word anchors. They do not consume characters in the string, but only assert whether a match is possible or not.

Positive Lookbehind: \((? <= B)A \rightarrow \) find expression A where expression B precedes:

Positive Lookahead: \(A(? = B) \rightarrow \) find expression A where expression B follows:

Negative Lookbehind: \((? <! B)A \rightarrow \) Find expression A where expression B does not precede:

Negative Lookahead: \(A(?! B) \rightarrow \) Find expression A where expression B does not follow.
How to extract "French" from "[Language: French]" ???

1. # (?<=B)A → find expression A where expression B precedes:
   2. # A = .*?    B = \[Language:
   3. "(?<=\[Language: ).*?"

5. # A(?=B) → find expression A where expression B follows
6. # A = .*?    B = \s*?\]
7. ".*?(?=\s*?\])"
8. "(?<=\[Language: ).*?(?=\s*?\])"

Check regex via https://regex101.com/
Accessing Text beyond NLTK
Processing Raw Text
POS Tagging

Dealing with other formats
HTML
Binary formats

Gutenberg eBooks

About 109 languages considering mixtures, such as *German to English*:

```python
import re
f = open("GUTINDEX.ALL", encoding="utf-8", errors="ignore")
data = f.read()
s = set(re.findall("(?<=\[Language: )\.*?(?=\s*\])", data))
print(s)
#
```

`findall()` returns all non-overlapping matches of pattern in string as a list of strings.
Gutenberg eBooks

```python
1   data = open("GUTINDEX.ALL", encoding="utf-8", errors="ignore").read()
2   dict = {}
3   for m in re.finditer("(?<=\n)(.*)\s+(\d+)(?=\n)", data):
4       dict[m.group(2)] = m.group(1)
5   print(dict)
6   #{"9333": "Johnny Bear, by E. T. Seton", "11513": "On Land And Sea At The Dardanelles, by Thomas Charles Bridges", "12461": "Castles in the Air, by Baroness Emmuska Orczy", "11034": "A Compilation of the Messages and Papers of the Presidents, Richardson", "3014": "The Old Northwest, by Frederic Austin Ogg", ... }
```

`finditer()` returns iterator yielding match objects (use it if the number of matches is really high).
Often enough, content on the Internet as well as locally stored content is transformed to a number of formats different from plain text (.txt).

- RTF – Rich Text Format (.rtf)
- HTML – HyperText Markup Language (.html, .htm)
- XHTML – Extensible HyperText Markup Language (.xhtml, .xht, .xml, .html, .htm)
- XML – Extensible Markup Language (.xml)
- RSS – Rich Site Summary (.rss, .xml)
Additionally, often text is stored in binary formats, such as:

- **MS Office formats** – (.doc, .dot, .docx, .docm, .dotx, .dotm, .xls, .xlt, .xlm, .ppt, .pps, .pptx ... and many others)
- **PDF** – Portable Document Format (.pdf)
- **OpenOffice formats** – (.odt, .ott, .oth, .odm ... and others)
Accessing Text beyond NLTK
Processing Raw Text
POS Tagging

Dealing with other formats
HTML
Binary formats

HTML

```
import urllib

urlData = urllib.request.urlopen(url)
html = urlData.read().decode("utf-8")
print(html)
```

# prints
# '<!DOCTYPE html>
# <html lang="en" id="responsive-news">
# <head prefix="og: http://ogp.me/ns#">
# <meta charset="utf-8">
# <title>Yemen rebel ballistic missile 'intercepted over Riyadh' – BBC News</title>
#...
```
HTML is often helpful since it marks up the distinct parts of the document, which makes them easy to find:

```
1 ... 
2 <title>Yemen rebel ballistic missile intercepted over Riyadh — BBC News</title>
3 ... 
4 ... 
```
Beautiful Soup

- Python library for pulling data out of HTML and XML files.
- can navigate, search, and modify the parse tree.

```python
html_doc = ""
<html><head><title>The Dormouse's story</title></head><body><p class="title"><b>The Dormouse's story</b></p><p class="story">Once upon a time there were three little sisters; and their names were
  <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
  <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
  <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
"""
Beautiful Soup

```python
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')

# with open("index.html") as fp:
#     soup = BeautifulSoup(fp)
```
Beautiful Soup

BeautifulSoup object represents the document as a nested data structure:

```python
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
print(soup.prettify())
# <html>
# <head>
#  <title>
#   The Dormouse's story
# </title>
# </head>
# <body>
#  <p class="title">
#   The Dormouse's story
# </p>
# ...
```
Beautiful Soup

Simple ways to navigate that data structure: say the name of the tag you want

```python
soup.title
# <title>The Dormouse's story</title>

soup.title.string
# u'The Dormouse's story'

soup.title.parent.name
# u'head'

soup.p
# <p class="title"><b>The Dormouse's story</b></p>

soup.p['class']
# u'title'
```
**Beautiful Soup**

Simple ways to navigate that data structure:

```python
soup.a
# <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>

soup.find_all('a')
# [<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
# <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]

soup.find(id="link3")
# <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```
One common task is extracting all the URLs found within a page’s \(<a>\) tags:

```python
for link in soup.find_all('a'):
    print(link.get('href'))
# http://example.com/elsie
# http://example.com/lacie
# http://example.com/tillie
```
Another common task is extracting all the text from a page:

```python
print(soup.get_text())
# The Dormouse's story
# The Dormouse's story
# Once upon a time there were three little sisters; and their names were
# Elsie,
# Lacie and
# Tillie;
# and they lived at the bottom of a well.
# ...
```
Installing Beautiful Soup:

`apt-get install python3-bs4` (for Python 3)
Nowadays we often store text in formats that are not human-readable: e.g. binary format (e.g. `.doc`, `.pdf`). These formats are not as easily processed as simple text.
There are a number of third-party modules that can be installed and used for extracting data from binary files. Yet, depending on the files, the output is not always clean and easily usable.
1 import nltk
2 import PyPDF2
3 pdf = PyPDF2.PdfFileReader(open("text.pdf", "rb"))
4 for page in pdf.pages:
  5     print(page.extractText())
6 # prints each of the pages from as raw text.
Abstract This course will use the Python programming language as the basis for various computational linguistic implementations. We will cover a wide range of natural language processing (NLP) tasks, such as tokenization, keyword extraction, normalization and stemming, categorization and tagging, as well as classification, chunking and language identification. All the latter are basic NLP tasks that will be discussed and their implementation in Python will be realized during the practical exercise in connection to the course. With respect to each task, we will concentrate on the problems that this task faces and the possible solutions to them within the Python framework. All students will be required to complete weekly assignments and write a term paper (10-12 pages) as a summary of the discussed topics and their importance and application for computational linguistics.
import nltk
import PyPDF2

pdf = PyPDF2.PdfFileReader(open("intro.pdf", "rb"))

for page in pdf.pages:
    print(page.extractText() + "\n")
In other cases, the full text might be extracted, but not in a easily usable format as here:

SymbolischeProgrammierspracheAbstractThiscoursewillusethePythonprogramminglanguageas
thebasisforvariouscomputationallinguisticimplementations. Wewillcoverawiderangeofnatural
languageprocessing (NLP) tasks, such as tokenization, keyword extraction, normalization,
and stemming, categorization and tagging, as well as chunking and language identification. All
the latter are basic NLP tasks that will be discussed and their implementation in Python will be realized during
the practical exercise in connection to the course. With respect to each task, we will concentrate
on the problems that this task faces and the possible solutions to them within the Python framework.
All students will be required to complete weekly assignments and write a term paper (10-12 pages)
as a summary of the discussed topics and their importance and application for computational
linguistics. Format of the course Credits: 4 SWS (6ECTS) Course times: Tuesdays 16:00-18:00, Thursdays
18:00-20:00 Location: Tuesdays {Room L155 and Thursdays {CIP-Pool Antarktis 30 Sessions: 14.1
0.2013 (07.02.2014) The course will be held in German and English. Course webpage: http://www.cis.uni-muenchen.de/kurse/desi/spInstructor: Desislava Zhekova Contact: desi@cis.uni-muenchen.de C106 Hours: Wednesdays 10:00-11:00, but feel free to come by anytime. An email in advance will ensure that you actually come!
What next?

We have seen multiple ways of getting raw text? But what to do with it next?
NLP pipelines

1. Raw text (string) → sentence segmentation → sentences (list of strings)
2. Sentences (list of strings) → tokenization → tokenized sentences (list of lists of strings)
3. Tokenized sentences (list of lists of strings) → part of speech tagging
4. Part of speech tagging
5. pos-tagged sentences (list of lists of tuples) → entity detection → chunked sentences (list of trees)
6. Chunked sentences (list of trees) → relation detection → relations (list of tuples)
Tokenization with NLTK:

```python
import nltk
import urllib

url = "http://www.gutenberg.org/files/2554/2554-0.txt"
urlData = urllib.request.urlopen(url)
data = urlData.read().decode("utf-8")
tokens = nltk.word_tokenize(data)
print(tokens)
```
Tokenization with regex:

```python
url = "http://www.gutenberg.org/files/2554/2554-0.txt"
urlData = urllib.request.urlopen(url)
data = urlData.read().decode("utf-8")

for m in re.finditer("\w+", data):
    print(m.group())
```

# prints:
# ...
# him
# although
# the
# explanation
```
url = "http://www.gutenberg.org/files/2554/2554-0.txt"
urlData = urllib.request.urlopen(url)
data = urlData.read().decode("utf-8")

for m in re.finditer("\w+", data):
    # for m in re.finditer("\S+", data):
    # for m in re.finditer("[a–zA–Z]+", data):
    # ...
    print(m.group())
import nltk
from nltk.corpus import gutenberg

moby = nltk.Text(gutenberg.words("melville-moby_dick.txt"))
print(moby.findall("\<a\> \<.*\> \<man\>\")

# prints

# a monied man; a nervous man; a dangerous man; a white man
#; a white man; a white man; a pious man; a queer man;
a good man; a mature man; a white man; a Cape man;
a
great man; a wise man; a wise man; a butterless man; a
white man; a fiendish man; a pale man; a furious man;
a
better man; a certain man; a complete man; ...
It is easy to build search patterns when the linguistic phenomenon we’re studying is tied to particular words.

For instance, searching a large text corpus for expressions of the form \textit{x and other ys} allows us to discover hypernyms.
import nltk
from nltk.corpus import brown

hobbies_learned = nltk.Text(brown.words(categories=['hobbies', 'learned']))

print(hobbies_learned.findall(r'\w+ <and> <other> \w+'))

# prints
# speed and other activities; water and other liquids; tomb and other landmarks; Statues and other monuments; pearls and other jewels; charts and other items; roads and other features; figures and other objects; military and other areas; demands and other factors; abstracts and other compilations; iron and other metals
POS Tagging Overview

- **parts-of-speech** (word classes, lexical categories, POS) – e.g. verbs, nouns, adjectives, etc.

- **part-of-speech tagging** (POS tagging, tagging) – labeling words according to their POS

- **tagset** – the collection of tags used for a particular task
Using a Tagger

A part-of-speech tagger, or POS tagger, processes a sequence of words, and attaches a part of speech tag to each word:

```
import nltk

text = nltk.word_tokenize("And now for something completely different")
print(nltk.pos_tag(text))

# [('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely', 'RB'), ('different', 'JJ')]
```
Variation in Tags

# [( 'And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely', 'RB'), ('different', 'JJ')]  

- CC – coordinating conjunction
- RB – adverb
- IN – preposition
- NN – noun
- JJ – adjective
NLTK provides documentation for each tag, which can be queried using the tag, e.g:

```python
>>> nltk.help.upenn_tagset('NN')
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino
    afghan shed thermostat investment slide
    humour falloff slick wind hyena override
    subhumanity machinist ...
```

```python
>>> nltk.help.upenn_tagset('CC')
CC: conjunction, coordinating
    & and both but either et for less minus neither
    nor or plus so therefore times v. versus vs.
    whether yet
```
Some POS tags denote variation of the same word type, e.g. NN, NNS, NNP, NNPS, such can be looked up via regular expressions.

```python
1 >>> nltk.help.upenn_tagset('NN*')
2 NN: noun, common, singular or mass
3     common—carrier cabbage knuckle—duster Casino ...
4 NNP: noun, proper, singular
5     Motown Venneboerger Czestochwa Ranzer Conchita ...
6 NNPS: noun, proper, plural
7     Americans Americas Amharas Amityvilles ...
8 NNS: noun, common, plural
9     undergraduates scotches bric—a—brac ...
```
Disambiguation

POS tagging does not always provide the same label for a given word, but decides on the correct label for the specific context – disambiguates across the word classes.
Disambiguation

POS tagging does not always provide the same label for a given word, but decides on the correct label for the specific context — disambiguates across the word classes.

```python
import nltk

text = nltk.word_tokenize("They refuse to permit us to obtain the refuse permit")
print(nltk.pos_tag(text))

# [('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```
Whenever a corpus contains tagged text, the NLTK corpus interface will have a `tagged_words()` method.

```python
>>> nltk.corpus.brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', '...

>>> nltk.corpus.brown.tagged_words()
[( 'The', 'AT' ), ( 'Fulton', 'NP–TL' ), '...
```
Even for one language, POS tagsets may differ considerably!
### Variation across Tagsets

#### Alphabetical list of part-of-speech tags used in the Penn Treebank Project:

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>2.</td>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>3.</td>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>4.</td>
<td>EX</td>
<td>Existential <em>there</em></td>
</tr>
<tr>
<td>5.</td>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>6.</td>
<td>IN</td>
<td>Preposition or subordinating conjunction</td>
</tr>
<tr>
<td>7.</td>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>8.</td>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>9.</td>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>10.</td>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>11.</td>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>12.</td>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
</tbody>
</table>
### Variation across Tagsets

#### The Open Xerox English POS tagset:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ADJ</td>
<td>(basic) adjective</td>
<td>[a] blue [book], [he is] big</td>
</tr>
<tr>
<td>+ADJCMP</td>
<td>comparative adjective</td>
<td>[he is] bigger, [a] better [question]</td>
</tr>
<tr>
<td>+ADJING</td>
<td>adjectival ing-form</td>
<td>[the] working [men]</td>
</tr>
<tr>
<td>+ADJPAP</td>
<td>adjectival past participle</td>
<td>[a] locked [door]</td>
</tr>
<tr>
<td>+ADJPRON</td>
<td>pronoun (with determiner) or adjective</td>
<td>[the] same; [the] other [way]</td>
</tr>
<tr>
<td>+ADJSUP</td>
<td>superlative adjective</td>
<td>[he is the] biggest; [the] best [cake]</td>
</tr>
<tr>
<td>+ADV</td>
<td>(basic) adverb</td>
<td>today, quickly</td>
</tr>
<tr>
<td>+ADV_CMP</td>
<td>comparative adverb</td>
<td>sooner</td>
</tr>
<tr>
<td>+ADVSUP</td>
<td>superlative adverb</td>
<td>soonest</td>
</tr>
<tr>
<td>+CARD</td>
<td>cardinal (except one)</td>
<td>two, 123, IV</td>
</tr>
<tr>
<td>+CARDONE</td>
<td>cardinal one</td>
<td>[at] one [time]; one [dollar]</td>
</tr>
<tr>
<td>+CM</td>
<td>comma</td>
<td>.</td>
</tr>
<tr>
<td>+COADV</td>
<td>coordination adverbs <em>either, neither</em></td>
<td>either [by law o by force]; [he didn't come] either</td>
</tr>
<tr>
<td>+COORD</td>
<td>coordinating conjunction</td>
<td>and, or</td>
</tr>
</tbody>
</table>
Variation across Tagsets

The variation across tagsets is based on the different decisions and the information needed to be included:

- morphologically rich tags
- morphologically poor ones
Arabic Example

For example, in Arabic the morphologically-poor tag \texttt{NN} may be divided into the following morphologically-rich variants:

\begin{Verbatim}

(ABBREV \texttt{NN})
(\texttt{LATIN} \texttt{NN})
(\texttt{DET}+\texttt{NOUN} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG} \texttt{NN})
(\texttt{NOUN} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_MASC\_SG\_ACC\_INDEF} \texttt{NN})
(\texttt{DEM}+\texttt{NOUN} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{CASE\_DEF\_ACC} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{CASE\_DEF\_GEN} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{CASE\_DEF\_NOM} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_DEF\_ACC} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_DEF\_GEN} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_DEF\_NOM} \texttt{NN})
(\texttt{DET}+\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_DEF\_ACC} \texttt{NN})
(\texttt{NOUN}+\texttt{CASE\_DEF\_ACC} \texttt{NN})
(\texttt{NOUN}+\texttt{CASE\_DEF\_GEN} \texttt{NN})
(\texttt{NOUN}+\texttt{CASE\_DEF\_NOM} \texttt{NN})
(\texttt{NOUN}+\texttt{CASE\_INDEF\_ACC} \texttt{NN})
(\texttt{NOUN}+\texttt{CASE\_INDEF\_GEN} \texttt{NN})
(\texttt{NOUN}+\texttt{CASE\_INDEF\_NOM} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_DEF\_ACC} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_DEF\_GEN} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_DEF\_NOM} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_INDEF\_ACC} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_INDEF\_GEN} \texttt{NN})
(\texttt{NOUN}+\texttt{NSUFF\_FEM\_SG}+\texttt{CASE\_INDEF\_NOM} \texttt{NN})
(\texttt{NEG\_PART}+\texttt{NOUN} \texttt{NN})
\end{Verbatim}
NLTK includes built-in mapping to a simplified tagset for most complex tagsets included in it:

```python
>>> nltk.corpus.brown.tagged_words()
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]

>>> nltk.corpus.brown.tagged_words(tagset='universal')
[('The', 'DET'), ('Fulton', 'NOUN'), ...]
```
### The Universal Part-of-Speech Tagset of NLTK:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
<th>English Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJ</td>
<td>adjective</td>
<td>new, good, high, special, big, local</td>
</tr>
<tr>
<td>ADP</td>
<td>adposition</td>
<td>on, of, at, with, by, into, under</td>
</tr>
<tr>
<td>ADV</td>
<td>adverb</td>
<td>really, already, still, early, now</td>
</tr>
<tr>
<td>CONJ</td>
<td>conjunction</td>
<td>and, or, but, if, while, although</td>
</tr>
<tr>
<td>DET</td>
<td>determiner, article</td>
<td>the, a, some, most, every, no, which</td>
</tr>
<tr>
<td>NOUN</td>
<td>noun</td>
<td>year, home, costs, time, Africa</td>
</tr>
<tr>
<td>NUM</td>
<td>numeral</td>
<td>twenty-four, fourth, 1991, 14:24</td>
</tr>
<tr>
<td>PRT</td>
<td>particle</td>
<td>at, on, out, over per, that, up, with</td>
</tr>
<tr>
<td>PRON</td>
<td>pronoun</td>
<td>he, their, her, its, my, I, us</td>
</tr>
<tr>
<td>VERB</td>
<td>verb</td>
<td>is, say, told, given, playing, would</td>
</tr>
<tr>
<td>.</td>
<td>punctuation marks</td>
<td>, , ; !</td>
</tr>
<tr>
<td>X</td>
<td>other</td>
<td>ersatz, esprit, dunno, gr8, univeristy</td>
</tr>
</tbody>
</table>
Tagged corpora for several other languages are distributed with NLTK, including Chinese, Hindi, Portuguese, Spanish, Dutch, and Catalan.

```python
>>> nltk.corpus.sinica_treebank.tagged_words()
[('一', 'Neu'), ('友情', 'Nad'), ('嘉珍', 'Nba'), ...]

>>> nltk.corpus.indian.tagged_words()
[('महिषेर', 'NN'), ('संतान', 'NN'), (':', 'SYM'), ...]
```
Frequency Distributions of POS Tags

We have calculated Frequency Distributions based on a sequence of words. Thus, we can do so for POS tags as well.

```python
import nltk
from nltk.corpus import brown

brown_news_tagged = brown.tagged_words(categories='news', tagset='universal')
tag_fd = nltk.FreqDist(tag for (word, tag) in brown_news_tagged)
print(tag_fd.most_common())

# [( 'NOUN', 30640), ('VERB', 14399), ('ADP', 12355), (',', 11928), ('DET', 11389), ('ADJ', 6706), ('ADV', 3349), ('CONJ', 2717), ('PRON', 2535), ('PRT', 2264), ('NUM', 2166), ('X', 106)]
```
Example Explorations

1. `import nltk`
2. `wsj = nltk.corpus.treebank.tagged_words(tagset='universal')`
3. `cfd1 = nltk.ConditionalFreqDist(wsj)`
4. `print(list(cfd1['yield'].keys()))`
5. `print(list(cfd1['cut'].keys()))`

What is calculated in the lines 4 and 5?
Example Explorations

```python
import nltk
wsj = nltk.corpus.treebank.tagged_words(tagset='universal')
cfd1 = nltk.ConditionalFreqDist(wsj)
print(list(cfd1['yield'].keys()))
# ['NOUN', 'VERB']
print(list(cfd1['cut'].keys()))
# ['NOUN', 'VERB']
```
We can reverse the order of the pairs, so that the tags are the conditions, and the words are the events. Now we can see likely words for a given tag:

```python
import nltk

wsj = nltk.corpus.treebank.tagged_words(tagset='universal')
cfd2 = nltk.ConditionalFreqDist((tag, word) for (word, tag) in wsj)
print(list(cfd2['VERB'].keys()))
# ['sue', 'leaving', 'discharge', 'posing', 'redistributing', 'emerges', 'anticipates', 'Hold', 'purrs', 'telling', 'obtained', 'ringing', 'mind', ... ]
```
import nltk

from nltk.corpus import brown

def process(sentence):
    for (w1, t1), (w2, t2), (w3, t3) in nltk.trigrams(sentence):
        if t1.startswith('V') and t2 == 'TO' and t3.startswith('V'):
            print(w1, w2, w3)

for tagged_sent in brown.tagged_sents():
    process(tagged_sent)

What is calculated here?
Example Explorations

1. # combined to achieve
2. # continue to place
3. # serve to protect
4. # wanted to wait
5. # allowed to place
6. # expected to become
7. # expected to approve
8. # expected to make
9. # intends to make
10. # seek to set
11. # like to see
12. # designed to provide
```python
import nltk
from nltk.corpus import brown

brown_news_tagged = brown.tagged_words(categories='news', tagset='universal')
data = nltk.ConditionalFreqDist((word.lower(), tag) for (word, tag) in brown_news_tagged)

for word in data.conditions():
    if len(data[word]) > 3:
        tags = data[word].keys()
        print(word, ' '.join(tags))
```

What is calculated here?
Example Explorations

Extract most ambiguous words across the word classes:

```python
import nltk
from nltk.corpus import brown

brown_news_tagged = brown.tagged_words(categories='news', tagset='universal')
data = nltk.ConditionalFreqDist((word.lower(), tag) for (word, tag) in brown_news_tagged)

for word in data.conditions():
    if len(data[word]) > 3:
        tags = data[word].keys()
        print(word, ' '.join(tags))

# that DET ADP ADV PRON
# best ADJ NOUN ADV VERB
# present ADJ NOUN ADV VERB
# close NOUN ADJ ADV VERB
```
The TreeTagger is a tool for annotating text with part-of-speech and lemma information. It is used to tag German, English, French, Italian, Danish, Dutch, Spanish, Bulgarian, Russian, Portuguese, Galician, Greek, Chinese, Swahili, Slovak, Slovenian, Latin, Estonian, etc.

Sample output:

<table>
<thead>
<tr>
<th>word</th>
<th>pos</th>
<th>lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>DT</td>
<td>the</td>
</tr>
<tr>
<td>TreeTagger</td>
<td>NP</td>
<td>TreeTagger</td>
</tr>
<tr>
<td>is</td>
<td>VBZ</td>
<td>be</td>
</tr>
<tr>
<td>easy</td>
<td>JJ</td>
<td>easy</td>
</tr>
<tr>
<td>to</td>
<td>TO</td>
<td>to</td>
</tr>
<tr>
<td>use</td>
<td>VB</td>
<td>use</td>
</tr>
</tbody>
</table>
**TreeTagger**

- Download the files from [http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/](http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/)
- Run the installation script: `sh install-tagger.sh`
- Test it:

```plaintext
1  echo 'Das ist ein gutes Beispiel!' | cmd/tree-tagger-german
2                                                                                     
3  reading parameters ...
4  tagging ...
5  finished.
6  das  PDS  die
7  ist  VAFIN  sein
8  ein  ART  eine
9  gutes  ADJA  gut
10 Beispiel  NN  Beispiel
11 !  $.  !
```
Accessing Text beyond NLTK
Processing Raw Text
POS Tagging

References

http://www.nltk.org/book/
https://github.com/nltk/nltk