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

Gutenberg Corpus

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`.

Gutenberg Corpus

Unfortunately, only 18 books are provided, which you can list as we have seen before:

```
>>> import nltk
>>> nltk.corpus.gutenberg.fileids()
["austen-emma.txt", "austen-persuasion.txt", "austen-sense.txt", "bible-kjv.txt", "blake-poems.txt", "bryant-stories.txt", "burgess-busterbrown.txt", "carroll-alice.txt", "chesterton-ball.txt", "chesterton-brown.txt", "chesterton-thursday.txt", "edgeworth-parents.txt", "melville-moby_dick.txt", "milton-paradise.txt", "shakespeare-caesar.txt", "shakespeare-hamlet.txt", "shakespeare-macbeth.txt", "whitman-leaves.txt"]
```

Gutenberg eBooks

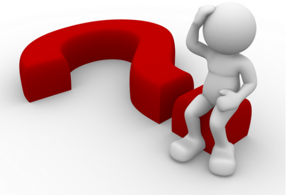
Accessing the original collection is thus helpful:

```
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?



Gutenberg Corpus

Directly from the Gutenberg Webpage → not very efficient.

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 ~ ~ ~ ~

TITLE and AUTHOR                                                     ETEXT NO.

Ypres 1914, by Otto Schwink                                              44234
 [Subtitle: An Official Account Published by Order
  of the German General Staff]
 [Translator: Graeme Chamley Wynne]

Plays by August Strindberg, Third Series, by August Strindberg           44233
  [Translator: Edwin Bj?rkman]

L'Illustration, No. 1608, 20 d?cembre 1873, by Various                   44232
  [Language: French]

The Miraculous Medal, by Jean Marie Aladel                               44231
 [Subtitle: Its Origin, History, Circulation, Results]
```

Gutenberg Corpus

And as you can see Crime and Punishment, by Fyodor Dostoevsky is also listed there:

```
Under the Redwoods, by Bret Harte                                    2555
  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                           2554
  [Tr.: Constance Garnett]
Jeanne d'Arc, by Mrs. (Margaret) Oliphant                            2553
  [Subtitle: Her Life And Death]
Thankful's Inheritance, by Joseph C. Lincoln                         2552
Droll Stories, Volume 3, by Honore de Balzac                         2551
  {See also: #2318 & #1925}
```

So what more can we find out?

Regex to extract information

- 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/`

Regex to extract information

- **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

Gutenberg eBooks

- How to extract "French" from "[Language: French]" ???

```
# (?<=B)A -> find expression A where expression B precedes:
# A = .*?      B = \[Language:
"(?<=\[Language: ).*?"

# A(?=B) -> find expression A where expression B follows
# A = .*?      B = \s*?\]
".*?(?=\s*?\])"

"(?<=\[Language: ).*?(?=\s*?\])"
```

- Check regex via `https://regex101.com/`

Gutenberg eBooks

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

```
import re
f = open("GUTINDEX.ALL", encoding="utf-8", errors="ignore")
data = f.read()
s = set(re.findall("(?<=\[Language: ).*?(?=\s*?\])", data))
print(s)
#{"German and English", "Bulgarian", "Friulian", "Spanish, English and Tagalog", "Catalan", "Ojibwa", "German to English", "Welsh", "French and English", "Hebrew", "Russian", "Spanish", "Galician", "Greek", "Spanish and Tagalog", "Romani, dialecto de los Gitanos de Espaa", "Portuguese & French", ... }
```

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

Gutenberg eBooks

```
data = open("GUTINDEX.ALL", encoding="utf-8", errors="ignore").read()
dict={}
for m in re.finditer("(?<=\n)(.*?)\s+(\d+)(?=\n)", data):
    dict[m.group(2)]= m.group(1)
print(dict)
#{"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).

Dealing with other formats

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`)

Dealing with other formats

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)

HTML

http://www.bbc.com/news/world-middle-east-42412729

```
import urllib

url="http://www.bbc.com/news/world-middle-east-42412729"
urlData = urllib.request.urlopen(url)
html = urlData.read().decode("utf-8")
print(html)

# prints
#'<!DOCTYPE html>\n<html lang="en" id="responsive-news">\n
#<head prefix="og: http://ogp.me/ns#">\n    <meta charset="utf-8
    ">\n
# <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">\n
# <title>Yemen rebel ballistic missile \'intercepted over Riyadh\'
    - BBC News</title>\n
# ...
```

HTML

HTML is often helpful since it marks up the distinct parts of the document, which makes them easy to find:

```
...
<title>Yemen rebel ballistic missile intercepted over
    Riyadh - BBC News</title>

...
```

Beautiful Soup

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

```
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>
<p class="story">...</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:

```
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">
#    <b>
#     The Dormouse's story
#    </b>
#    ...
```

Beautiful Soup

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

```
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:

```
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>
```

Beautiful Soup

One common task is extracting all the URLs found within a page's <a> tags:

```
for link in soup.find_all('a'):
    print(link.get('href'))
# http://example.com/elsie
# http://example.com/lacie
# http://example.com/tillie
```

Beautiful Soup

Another common task is extracting all the text from a page:

```
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.
#
# ...
```

Beautiful Soup

Installing Beautiful Soup:

```
apt-get install python3-bs4 (for Python 3)
```

Binary formats

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.

Binary formats

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.

Binary formats

```python
import nltk
import PyPDF2

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

for page in pdf.pages:
    print(page.extractText())

# prints each of the pages from as raw text.
```

Binary formats

Snippet from a pdf document "intro.pdf"

LMU | LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN | CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG STUDIENGANG COMPUTERLINGUISTIK

Symbolische Programmiersprache

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.

Binary formats

```python
import nltk
import PyPDF2

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

for page in pdf.pages:
    print(page.extractText()+"\n")
```

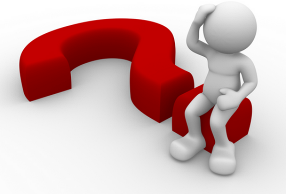
Binary formats

In other cases, the full text might be extracted, but not in a easily usable format as here:

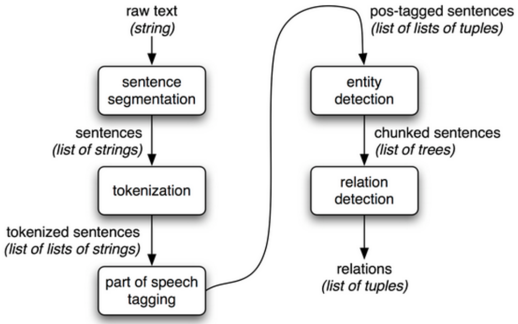
```
SymbolischeProgrammierspracheAbstractThiscoursewillusethePythonprogramminglanguagea
sthebasisforvariouscomputationallinguisticimplementations.Wewillcoverawiderangeofna
turallanguageprocessing(NLP)tasks,suchastokenization,keywordextraction,normalizatio
nandstemming,categorizationandtagging,aswellaschunkingandlanguageidenAllthelatterar
ebasicNLPtasksthatwillbediscussedandtheirimplementationinPythonwillberealizedduring
thepracticalexerciseinconnectiontothecourse.Withrespecttoeachtask,wewillconcentrate
ontheproblemsthatthistaskfacesandthepossiblesolutionstothemwithinthePythonframework
.Allstudentswillberequiredtocompleteweeklyassignmentsandwriteatermpaper(10-12pages)
asasummaryofthediscussedtopicsandtheirimportanceandapplicationforcomputationallingu
istics.FormatofthecourseCredits:4SWS(6ECTS)Coursetimes:Tuesdays16:00{18:00,Thursday
s16:00{18:00Location:Tuesdays{RoomL155andThursdays{CIP-PoolAntarktis30Sessions:14.1
0.2013{07.02.2014ThecoursewillbeheldinGermanandEnglish.Coursewebpage:http://www.cis
.uni-muenchen.de/kurse/desi/spInstructor:DesislavaZhekovaContact:desi@cis.uni-muenc
hen.deC106Hours:Wednesdays10:00{11:00,butfeelfreetocomebyanytime.Anemailinadvancewi
llmakesurethatyouactuallyme!
```

What next?

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



NLP pipelines



Tokenization

Tokenization with NLTK:

```
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

Tokenization with regex:

```
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
```

Tokenization

```
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())
```

Searching Tokenized Text

```
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;   ...
```

Searching Tokenized Text

- 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 **x and other ys** allows us to discover hypernyms.

Searching Tokenized Text

```
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+s>"))
# 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:

```python
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

Documentation

NLTK provides documentation for each tag, which can be queried using the tag, e.g:

```
>>> 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 ...
>>> 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
```

Documentation

Note!

Some POS tags denote variation of the same word type, e.g. NN, NNS, NNP, NNPS, such can be looked up via regular expressions.

```
>>> nltk.help.upenn_tagset('NN*')
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino ...
NNP: noun, proper, singular
    Motown Venneboerger Czestochwa Ranzer Conchita
        ...
NNPS: noun, proper, plural
    Americans Americas Amharas Amityvilles ...
NNS: noun, common, plural
    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.

```
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')]
```

Example from Brown

Whenever a corpus contains tagged text, the NLTK corpus interface will have a `tagged_words()` method.

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

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

Variation across Tagsets

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:

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
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

Variation across Tagsets

The Open Xerox English POS tagset:

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

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 `NN` may be divided into the following morphologically-rich variants:

```
(ABBREV NN)
(LATIN NN)
(DET+NOUN NN)
(DET+NOUN+NSUFF_FEM_SG NN)
(NOUN NN)
(NOUN+NSUFF_FEM_SG NN)
(NOUN+NSUFF_MASC_SG_ACC_INDEF NN)
(DEM+NOUN NN)
(DET+NOUN+CASE_DEF_ACC NN)
(DET+NOUN+CASE_DEF_GEN NN)
(DET+NOUN+CASE_DEF_NOM NN)
(DET+NOUN+NSUFF_FEM_SG+CASE_DEF_ACC NN)
(DET+NOUN+NSUFF_FEM_SG+CASE_DEF_GEN NN)
(DET+NOUN+NSUFF_FEM_SG+CASE_DEF_NOM NN)
(NOUN+CASE_DEF_ACC NN)
(NOUN+CASE_DEF_GEN NN)
(NOUN+CASE_DEF_NOM NN)
(NOUN+CASE_INDEF_ACC NN)
(NOUN+CASE_INDEF_GEN NN)
(NOUN+CASE_INDEF_NOM NN)
(NOUN+NSUFF_FEM_SG+CASE_DEF_ACC NN)
(NOUN+NSUFF_FEM_SG+CASE_DEF_GEN NN)
(NOUN+NSUFF_FEM_SG+CASE_DEF_NOM NN)
(NOUN+NSUFF_FEM_SG+CASE_INDEF_ACC NN)
(NOUN+NSUFF_FEM_SG+CASE_INDEF_GEN NN)
(NOUN+NSUFF_FEM_SG+CASE_INDEF_NOM NN)
(NEG_PART+NOUN NN)
```

NLTK and simplified tags

NLTK includes built-in mapping to a simplified tagset for most complex tagsets included in it:

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

>>> nltk.corpus.brown.tagged_words(tagset='universal'
    )
[('The', 'DET'), ('Fulton', 'NOUN'), ... ]
```

NLTK and simplified tags

The Universal Part-of-Speech Tagset of NLTK:

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

Tagged Corpora for Other Languages

Tagged corpora for several other languages are distributed with NLTK, including Chinese, Hindi, Portuguese, Spanish, Dutch, and Catalan.

```
>>> nltk.corpus.sinica_treebank.tagged_words()
>>> nltk.corpus.indian.tagged_words()
```

```
[('一', 'Neu'), ('友情', 'Nad'), ('嘉珍', 'Nba'), ...]
[('মহিষের', 'NN'), ('সন্তান', '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.

```
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

```
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']
```

Example Explorations

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:

```
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', ... ]
```

Example Explorations

```
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

```
# combined to achieve
# continue to place
# serve to protect
# wanted to wait
# allowed to place
# expected to become
# expected to approve
# expected to make
# intends to make
# seek to set
# like to see
# designed to provide
```

Example Explorations

```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:

```
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
```

TreeTagger

- The TreeTagger is a tool for annotating text with part-of-speech and lemma information
- used to tag German, English, French, Italian, Danish, Dutch, Spanish, Bulgarian, Russian, Portuguese, Galician, Greek, Chinese, Swahili, Slovak, Slovenian, Latin, Estonian, etc.
- Sample output:

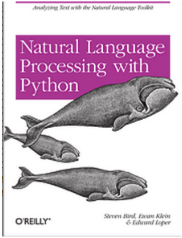
word	pos	lemma
The	DT	the
TreeTagger	NP	TreeTagger
is	VBZ	be
easy	JJ	easy
to	TO	to
use	VB	use

TreeTagger

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

```
echo 'Das ist ein gutes Beispiel!' | cmd/tree-tagger-german

        reading parameters ...
        tagging ...
         finished.
das     PDS     die
ist     VAFIN   sein
ein     ART     eine
gutes   ADJA    gut
Beispiel        NN      Beispiel
!       $.      !
```

References

- 

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