Natural Language Toolkit (NLTK)

Natural Language Processing
Lecture 2

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

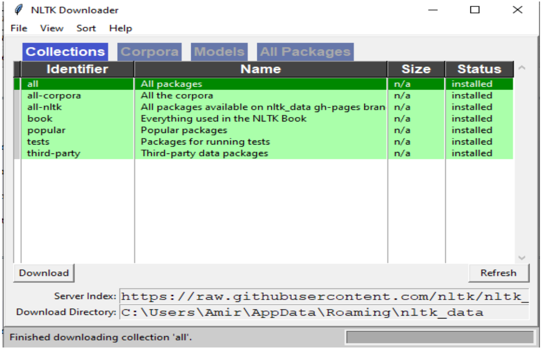
Language Processing and Python

Questions

- What can we achieve by combining simple programming techniques with large quantities of text?
- How can we automatically extract key words and phrases that sum up the style and content of a text?
- What tools and techniques does the Python programming language provide for such work?
- What are some of the interesting challenges of natural language processing?

- Before going further you should install NLTK.
- Once you've installed NLTK, start up the Python interpreter and type in the following commands

```python
import nltk
nltk.download()
```



- **Load Text:** The corpus module contains all the data you will need to process.

```
from nltk.book import *
import nltk

print(text1.vocab())
print(type(text1))
print(len(text1))
```

- **Gutenberg:** Load some other texts from Gutenberg project

```
from nltk.corpus import gutenberg
print(gutenberg.fileids())
print(nltk.corpus.gutenberg.fileids())
hamlet = gutenberg.words('shakespeare-hamlet.txt')
```

- **Inaugural Speech:** Load all the presidential speech.

```
from nltk.corpus import inaugural
print(inaugural.fileids())
print(nltk.corpus.inaugural.fileids())
from nltk.text import Text
former_president = Text(inaugural.words(inaugural.fileids()[-1]))
print(' '.join(former_president.tokens[0:1000]))
```

- **Similarity:** A concordance view shows us every occurrence of a given word.

```
from nltk.book import text1
from nltk.book import text4
from nltk.book import text6

print(text1.concordance("monstrous"))
print(text1.similar("monstrous"))
print(text1.collocations())
text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", "America"])
```

- **Counting Vocabulary:** Use the computer to count the words in a text in a variety of useful ways.

```

print(text6.count("Very"))
print(text6.count('the') / float(len(text6)) * 100)
print(text4.count("bless"))
print(text4[100])
print(text4.index('the'))
print(text4[524])
print(text4.index('men'))
```

- **Lexical diversity:** Lexical diversity is a measure of how many different words appear in a text.

```
from nltk.book import text1
from nltk.book import text4

print(text4[100])
print(text4.index('the'))
print(text4[524])
print(text4.index('men'))
print(text4[0:len(text4)])

print(set(text4))
print(sorted(set(text4)))
print(sorted(set(text4)))
print(len(set(text4)))

T1_diversity = float(len(set(text1))) / float(len(text1))
print("The lexical diversity is: ", T1_diversity * 100, "%")
T4_diversity = float(len(set(text4))) / float(len(text4))
print("The lexical diversity is: ", T4_diversity * 100, "%")
```

- Identify the words of a text that are most informative about the topic

```
from nltk.book import text1
from nltk.book import text4
from nltk import FreqDist
import nltk
Freq_Dist = FreqDist(text1)
print(Freq_Dist)
print(Freq_Dist.most_common(10))
print(Freq_Dist['his'])
Freq_Dist.plot(50, cumulative = False)
Freq_Dist.plot(50, cumulative = True)
Freq_Dist.hapaxes()
Once_happend= Freq_Dist.hapaxes() ; print(Once_happend)
print(text4.count('america') / float(len(text4) * 100))
```

- Word Finding.

```
Value_set = set(text1)
long_words = [words for words in Value_set if len(words) > 17]
print(sorted(long_words))
my_text = ["Here", "are", "some", "words", "that", "are", "in", "a", "list"]
vocab = sorted(set(my_text)) ; print(vocab)
word_freq = nltk.FreqDist(my_text); print(word_freq.most_common(5))
```

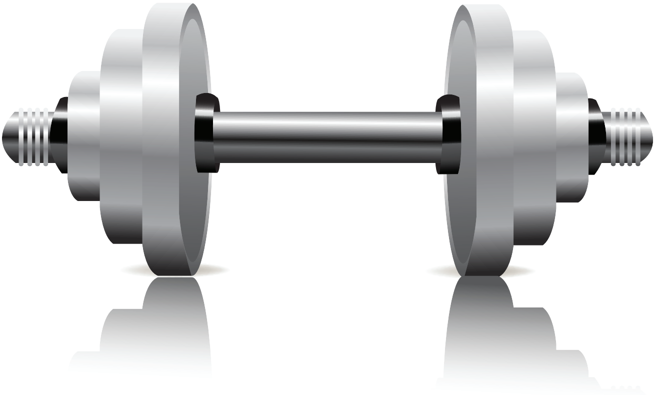
Automatic Natural Language Understanding

- We have been exploring language bottom-up, with the help of texts and the Python programming language.
- Also we are interested in exploiting our knowledge of language and computation by building useful language technologies.
- For example, search engines have been crucial to the growth and popularity of the Web.
- It takes skill, knowledge, and some luck, to extract answers to such questions as:
  - *What tourist sites can I visit between Philadelphia and Pittsburgh on a limited budget?*

Interesting Challenges

- Word Sense Disambiguation
  - serve: help with food or drink; hold an office; put ball into play
  - dish: plate; course of a meal; communications device
- Pronoun Resolution
  - The thieves stole the paintings. They were subsequently sold.
  - The thieves stole the paintings. They were subsequently caught.
  - The thieves stole the paintings. They were subsequently found.
- Generating Language Output
  - Text:The thieves stole the paintings. They were subsequently sold.
  - Human: Who or what was sold?
  - Machine: The paintings.
- Machine Translation
- Generating Language Output

- Exercise 1 to 4
- Class-Ex-Lecture2.py



Accessing Text Corpora and Lexical Resources

Gutenberg Corpus

- NLTK includes a small selection of texts from the Project Gutenberg electronic text archive, which contains some 25,000 free electronic books, hosted at http://www.gutenberg.org/

```
from nltk.corpus import gutenberg
import nltk

print(gutenberg.fileids())
emma = gutenberg.words('austen-emma.txt')
print(len(emma))
emma_Text = nltk.Text(gutenberg.words('austen-emma.txt'))
emma_Text.concordance("surprize")
for fileid in gutenberg.fileids():
    num_chars = len(gutenberg.raw(fileid))
    num_words = len(gutenberg.words(fileid))
    num_sents = len(gutenberg.sents(fileid))
    num_vocab = len(set(w.lower() for w in gutenberg.words(fileid)))
    print(round(num_chars / num_words), round(num_words / num_sents),
          round(num_words / num_vocab), fileid)

macbeth_sentences = gutenberg.sents('shakespeare-macbeth.txt');
print(macbeth_sentences)
print(macbeth_sentences[1116])
longest_len = max(len(s) for s in macbeth_sentences); print(longest_len)
print( [s for s in macbeth_sentences if len(s) == longest_len])
```

- NLTK's small collection of web text includes content from a Firefox discussion forum

```python
from nltk.corpus import webtext
from nltk.corpus import nps_chat

for fileid in webtext.fileids():
    print(fileid, webtext.raw(fileid)[:65])

text = webtext.raw('firefox.txt')
print([i for i in range(len(text)) if text.startswith('a', i)])
```

- There is also a corpus of instant messaging chat sessions.

```python
chatroom = nps_chat.posts('10-19-20s_706posts.xml')
print(chatroom[123])

text2 = nps_chat.raw('11-09-teens_706posts.xml')
```

- The Brown Corpus was the first million-word electronic corpus of English, created in 1961 at Brown University.

```python
from nltk.corpus import brown
import nltk
print(brown.categories())
print(brown.words(categories='news'))
print( brown.words(fileids=['cg22']))
print(brown.sents(categories=['news', 'editorial', 'reviews']))
from nltk.corpus import brown
news_text = brown.words(categories='news')
fdist = nltk.FreqDist(w.lower() for w in news_text)
modals = ['can', 'could', 'may', 'might', 'must', 'will']
for m in modals:
    print(m + ':', fdist[m], end=' ')
```

- Observe that the most frequent modal in the news genre is will, while the most frequent modal in the romance genre is could.

```python
cfd = nltk.ConditionalFreqDist((genre, word)
            for genre in brown.categories()
            for word in brown.words(categories=genre))
genres = ['news', 'religion', 'hobbies', 'science_fiction', 'romance', 'humor']
modals = ['can', 'could', 'may', 'might', 'must', 'will']
print(); print(cfd.tabulate(conditions=genres, samples=modals))
```

- The Reuters Corpus contains 10,788 news documents totaling 1.3 million words.

```python
from nltk.corpus import reuters
print(reuters.fileids())
print(reuters.categories())
print(reuters.categories('training/9865'))
print(reuters.categories(['training/9865', 'training/9880']))
print(reuters.fileids(['barley', 'corn']))
```

- Similarly, we can specify the words or sentences we want in terms of files or categories.

```python
print(reuters.words('training/9865')[:14])
print(reuters.words(['training/9865', 'training/9880']))
print(reuters.words(categories='barley'))
print(reuters.words(categories=['barley', 'corn']))
```

Inaugural Corpus

- The corpus is actually a collection of 55 texts, one for each presidential address.

```
from nltk.corpus import inaugural
print(inaugural.fileids())
print([fileid[:4] for fileid in inaugural.fileids()])
```

- Let's look at how the words America and citizen are used over time.

```
import nltk
cfd = nltk.ConditionalFreqDist(
    (target, fileid[:4])
    for fileid in inaugural.fileids()
    for w in inaugural.words(fileid)
    for target in ['america', 'citizen']
    if w.lower().startswith(target))
cfd.plot()
```

Loading Specific Corpus

- If you have your own collection of text files that you would like to access using the above methods, you can easily load them with the help of NLTK's PlaintextCorpusReader.

```
from nltk.corpus import PlaintextCorpusReader
import os

corpus_root = os.getcwd()
wordlists = PlaintextCorpusReader(corpus_root, 'Corpus.txt')

print(wordlists.fileids())
print(wordlists.words('Corpus.txt'))
```

- As another example, suppose you have your own local copy of Penn Treebank, you can use ***BracketParseCorpusReader*** command.

Text Corpus Structure

- Gutenberg Corpus is isolated text.
- Brown Corpus is categorized text.
- Reuters is overlapping text.
- Inaugural is temporal text.

Example	Description
fileids()	the files of the corpus
fileids([categories])	the files of the corpus corresponding to these categories
categories()	the categories of the corpus
categories([fileids])	the categories of the corpus corresponding to these files
raw()	the raw content of the corpus
raw(fileids=[f1,f2,f3])	the raw content of the specified files
raw(categories=[c1,c2])	the raw content of the specified categories
words()	the words of the whole corpus
words(fileids=[f1,f2,f3])	the words of the specified fileids

Example	Description
words(categories=[c1,c2])	the words of the specified categories
sents()	the sentences of the whole corpus
sents(fileids=[f1,f2,f3])	the sentences of the specified fileids
sents(categories=[c1,c2])	the sentences of the specified categories
abspath(fileid)	the location of the given file on disk
encoding(fileid)	the encoding of the file (if known)
open(fileid)	open a stream for reading the given corpus file
root	if the path to the root of locally installed corpus
readme()	the contents of the README file of the corpus

Conditional Frequency - Counting Words by Genre

- Let's look at 2 genre in brown corpus (news and romance). We loop over them and look at conditional frequency distribution.

```
from nltk.corpus import brown
import nltk

genre_word = [(genre, word)
              for genre in ['news', 'romance']
              for word in brown.words(categories=genre)]

print( genre_word[:4]); print(genre_word[-4:])
cfd = nltk.ConditionalFreqDist(genre_word)
print(cfd.conditions())

print(cfd['news'])
print(cfd['romance'])
cfd['romance'].most_common(20)
```

- This will give an idea about the words frequency with certain condition.

Conditional Frequency - Tabulating Distributions

- Let’s look at 2 genre in brown corpus (news and romance).

```
from nltk.corpus import inaugural
import nltk
cfd = nltk.ConditionalFreqDist((target, fileid[:4])
    for fileid in inaugural.fileids()
    for w in inaugural.words(fileid)
    for target in ['america', 'citizen']
    if w.lower().startswith(target))
print(cfd['america'].most_common(20))
cfd.tabulate(conditions=['america', 'citizen']);cfd.plot()
```

- We interpret the last cell on the top row to mean that 1,638 words of the English text have 9 or fewer letters.

```
from nltk.corpus import udhr
languages = ['Chickasaw', 'English', 'German_Deutsch',
             'Greenlandic_Inuktikut', 'Hungarian_Magyar', 'Ibibio_Efik']
cfd = nltk.ConditionalFreqDist(
           (lang, len(word))
          for lang in languages
          for word in udhr.words(lang + '-Latin1'))
cfd.tabulate(conditions=['English', 'German_Deutsch'],
             samples=range(10), cumulative=True)
```

- A bigram or digram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words.

- A bigram is an n-gram for n=2. The frequency distribution of every bigram in a string is commonly used for simple statistical analysis of text in many applications.

- Bigrams help provide the conditional probability of a token given the preceding token, when the relation of the conditional probability is applied:

$$P(W_n|W_{n-1}) = \frac{P(W_{n-1}, W_n)}{P(W_{n-1})}$$

- We can use a conditional frequency distribution to create a table of bigrams.

```
import nltk
sent = ['In', 'the', 'beginning', 'God', 'created',
        'the', 'heaven', 'and', 'the', 'earth', '.']
print(list(nltk.bigrams(sent)))
```

- we treat each word as a condition, and for each one we effectively create a frequency distribution over the following words.

```
def generate_model(cfdist, word, num=15):
    for i in range(num):
        print(word, end=' ')
        word = cfdist[word].max()

text = nltk.corpus.genesis.words('english-kjv.txt')
text1 = text[0:]
bigrams = nltk.bigrams(text)
print(list(nltk.bigrams(text1))[0:20])
cfd = nltk.ConditionalFreqDist(bigrams)
print(cfd['living'])
generate_model(cfd, 'living')
```

- NLTK includes some corpora that are nothing more than wordlists. We can use it to find unusual or mis-spelt words in a text corpus

```
import nltk
def unusual_words(text):
    text_vocab = set(w.lower() for w in text if w.isalpha())
    english_vocab = set(w.lower() for w in nltk.corpus.words.words())
    unusual = text_vocab - english_vocab
    return sorted(unusual)
print(unusual_words(nltk.corpus.gutenberg.words('austen-sense.txt')))
print( unusual_words(nltk.corpus.nps_chat.words()))
```

- Let's define a function to compute what fraction of words in a text are not in the stopwords list:

```
from nltk.corpus import stopwords
print(stopwords.words('english'))

def content_fraction(text):
    stopwords = nltk.corpus.stopwords.words('english')
    content = [w for w in text if w.lower() not in stopwords]
    return len(content) / len(text)
print(content_fraction(nltk.corpus.reuters.words()))
```

Stop word

- "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools.
- Any group of words can be chosen as the stop words for a given purpose.
- For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on.
  - In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as "The Who", "The The", or "Take That"
  - Other search engines remove some of the most common words—including lexical words, such as "want"—from a query in order to improve performance

- One more wordlist corpus is the Names corpus, containing 8,000 first names categorized by gender.

```
import nltk
names = nltk.corpus.names
print(names.fileids())
male_names = names.words('male.txt')
female_names = names.words('female.txt')
print([w for w in male_names if w in female_names])
```

- It is well known that names ending in the letter a are almost always female

```
cfd = nltk.ConditionalFreqDist(
          (fileid, name[-1])
          for fileid in names.fileids()
          for name in names.words(fileid))
cfd.plot()
```

- NLTK includes the CMU Pronouncing Dictionary for US English, which was designed for use by speech synthesizers.

```
import nltk

entries = nltk.corpus.cmudict.entries()
print(len(entries))
for entry in entries[42371:42379]:
    print(entry)

for word, pron in entries:
    if len(pron) == 3:
        ph1, ph2, ph3 = pron
        if ph1 == 'P' and ph3 == 'T':
            print(word, ph2, end=' ')
            print()
```

- For each word, this lexicon provides a list of phonetic codes.

```
syllable = ['N', 'IH0', 'K', 'S']
print([word for word, pron in entries if pron[-4:] == syllable])
print([w for w, pron in entries if pron[-1] == 'M' and w[-1] == 'n'])
print(sorted(set(w[:2] for w, pron in entries if pron[0] == 'N' and w[0] != 'n')))
```

A Pronouncing Dictionary 2

- The phones contain digits to represent primary stress, secondary stress and no stress. As our final example, we define a function to extract the stress digits and then scan our lexicon to find words having a particular stress pattern.

```
import nltk
entries = nltk.corpus.cmudict.entries()

def stress(pron):
    return [char for phone in pron for char in phone if char.isdigit()]
print( [w for w, pron in entries if stress(pron) == [’0’, ’1’, ’0’, ’2’, ’0’]])
print([x[1] for x in entries if x[0]==’abbreviated’])
```

- Here we find all the p-words consisting of three sounds, and group them according to their first and last sounds.

```
p3 = [(pron[0]+’-’+pron[2], word)
       for (word, pron) in entries
       if pron[0] == ’P’ and len(pron) == 3]
cfd = nltk.ConditionalFreqDist(p3)
for template in sorted(cfd.conditions()):
    if len(cfd[template]) > 10:
       words = sorted(cfd[template])
       wordstring = ’ ’.join(words)
       print(template, wordstring[:70] + "...")
prondict = nltk.corpus.cmudict.dict()
prondict[’blog’] = [[’B’, ’L’, ’AA1’, ’G’]]
print(prondict[’blog’])
```

- WordNet is a lexical database of semantic relations between words in more than 200 languages.
- WordNet links words into semantic relations including synonyms and hyponyms.
- The synonyms are grouped into synsets with short definitions and usage examples
- WordNet can thus be seen as a combination and extension of a dictionary and thesaurus
- NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets

WordNet Hierarchy

- For example, given a concept like motorcar, we can look at the concepts that are more specific; the (immediate) hyponyms

```
from nltk.corpus import wordnet as wn
print(wn.synsets('motorcar'))
print(wn.synset('car.n.01').lemma_names())
print(wn.synset('car.n.01').definition())
print(wn.synset('car.n.01').examples())
print(wn.synset('car.n.01').lemmas())
print(wn.lemma('car.n.01.automobile'))
print(wn.lemma('car.n.01.automobile').synset())
print(wn.lemma('car.n.01.automobile').name())
print(wn.synsets('car'))
for synset in wn.synsets('car'):
    print(synset.lemma_names())
print(wn.lemmas('car'))
```

- We can also navigate up the hierarchy by visiting hypernyms.

```
motorcar = wn.synset('car.n.01'); print(motorcar)
types_of_motorcar = motorcar.hyponyms()
sorted(lemma.name() for synset in types_of_motorcar for lemma in synset.lemmas())
print(motorcar.hypernyms())
paths = motorcar.hypernym_paths()
print([synset.name() for synset in paths[0]])
print([synset.name() for synset in paths[1]])
```

Semantic Similarity

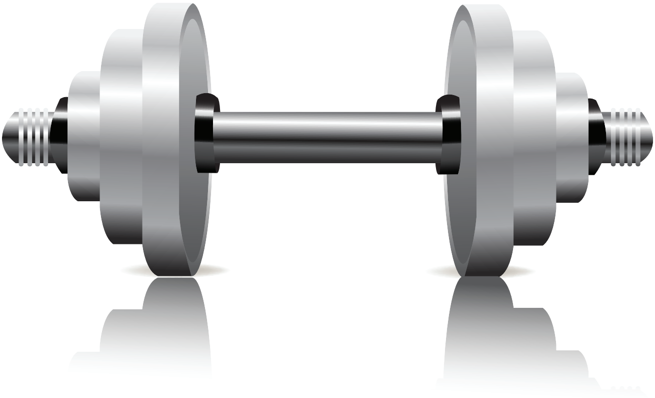
- Given a particular synset, we can traverse the WordNet network to find synsets with related meanings.

```
from nltk.corpus import wordnet as wn
right = wn.synset('right_whale.n.01')
orca = wn.synset('orca.n.01')
minke = wn.synset('minke_whale.n.01')
tortoise = wn.synset('tortoise.n.01')
novel = wn.synset('novel.n.01')
print(right.lowest_common_hypernyms(minke))
print(right.lowest_common_hypernyms(orca))
print(right.lowest_common_hypernyms(tortoise))
```

- Similarity measures have been defined over the collection of WordNet synsets which incorporate the above insight. For example, path_similarity assigns a score in the range 0–1 based on the shortest path that connects the concepts in the hypernym hierarchy

```
print(right.path_similarity(minke))
print(wn.synset('whale.n.02').min_depth())
print(wn.synset('vertebrate.n.01').min_depth())
print(wn.synset('entity.n.01').min_depth())
```

- Exercise 5 to 10
- Class-Ex-Lecture2.py



Processing Raw Text

- Text segmentation is the process of dividing written text into meaningful units, such as words, sentences, or topics.
- Word segmentation is the problem of dividing a string of written language into its component words.
- In English and many other languages using some form of the Latin alphabet, the space is a good approximation of a word divider (word delimiter).
- Many English compound nouns are variably written (for example, ice box = ice-box = icebox; pig sty = pig-sty = pigsty)
- Sentence segmentation is the problem of dividing a string of written language into its component sentences.
- However even in English this problem is not trivial due to the use of the full stop character for abbreviations, which may or may not also terminate a sentence.

- A small sample of texts from Project Gutenberg appears in the NLTK corpus collection. The rest of file are located at http://www.gutenberg.org/

```
from urllib import request
from nltk import word_tokenize
import  nltk

url = "http://www.gutenberg.org/files/2554/2554-0.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
print(type(raw));print(len(raw));print(raw[:75])
```

- We want to break up the string into words and punctuation. This step is called tokenization, and it produces our familiar structure, a list of words and punctuation.

```
tokens = word_tokenize(raw)
print(type(tokens))
print(tokens[:10])
text = nltk.Text(tokens)
print(type(text))
print(text.collocations())
```

- Much of the text on the web is in the form of HTML documents.

```
from urllib import request
from bs4 import BeautifulSoup
from nltk import word_tokenize
import nltk

url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
html = request.urlopen(url).read().decode('utf8')
print(html[:60])
```

- To get text out of HTML we will use a Python library called **BeautifulSoup**

```
raw = BeautifulSoup(html, 'html.parser').get_text()
tokens = word_tokenize(raw); print(tokens)
print(tokens = tokens[110:390])
text = nltk.Text(tokens); print(text)
print(text.concordance('gene'))
```

Reading Local Files

- In order to read a local file, we need to use Python’s built-in open() function, followed by the read() method.

```
from nltk import word_tokenize
from nltk import Text

f = open(’Corpus.txt’)
raw = f.read()
f = open(’Corpus.txt’, ’r’)
for line in f:
    print(line.strip())
```

- NLTK’s Text will change the type and we can use the NLTK methods on raw data.

```
words_token = word_tokenize(raw)
text = Text(words_token)
text.dispersion_plot([’corpus’])
```

Text normalization

- Text normalization is the process of transforming text into a single canonical form that it might not have had before.
- Normalizing text before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it.
- Text normalization requires being aware of what type of text is to be normalized and how it is to be processed afterwards;
- There is no all-purpose normalization procedure.
- ”$200” would be pronounced as ”two hundred dollars” in English

Normalizing Text - Stemmer

- Text normalization is the process of transforming text into a single canonical form that it might not have had before.

```
from nltk import word_tokenize
import nltk
from nltk import Text

raw = """DENNIS: Listen, strange women lying in ponds distributing swords
 is no basis for a system of government.  Supreme executive power derives from
a mandate from the masses, not from some farcical aquatic ceremony."""

tokens = word_tokenize(raw)
porter = nltk.PorterStemmer()
lancaster = nltk.LancasterStemmer()
print([porter.stem(t) for t in tokens])
print( [lancaster.stem(t) for t in tokens])
```

- NLTK includes several off-the-shelf stemmers, and if you ever need a stemmer you should use one of these in preference to crafting your own

```
porter = nltk.PorterStemmer()
grail = nltk.corpus.webtext.words('grail.txt')
text = Text(grail)
text.concordance('lie')
```

Normalizing Text - Lemmatisation

- The WordNet lemmatizer only removes affixes if the resulting word is in its dictionary.

```
import nltk
from nltk import word_tokenize

raw = """DENNIS: Listen, strange women lying in ponds distributing swords
 is no basis for a system of government.  Supreme executive power derives from
a mandate from the masses, not from some farcical aquatic ceremony."""

tokens = word_tokenize(raw)

wnl = nltk.WordNetLemmatizer()
print([wnl.lemmatize(t) for t in tokens])
```

- The WordNet lemmatizer is a good choice if you want to compile the vocabulary of some texts and want a list of valid lemmas

Stemmer Vs. Lemmatisation

- Lemmatisation is closely related to stemming.
- The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech.
- However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.
- If you lemmatize the word ’Caring’, it would return ’Care’. If you stem, it would return ’Car’ and this is erroneous.
- If you lemmatize the word ’Stripes’ in verb context, it would return ’Strip’. If you lemmatize it in noun context, it would return ’Stripe’. If you just stem it, it would just return ’Strip’.Stemming handles matching “car” to “cars” .

Further Issues with Tokenization

- Tokenization turns out to be a far more difficult task than you might have expected.
- When developing a tokenizer it helps to have access to raw text which has been manually tokenized, in order to compare the output of your tokenizer with high-quality (or ”gold-standard”) tokens.
- A final issue for tokenization is the presence of contractions, such as didn’t.

- Manipulating texts at the level of individual words often presupposes the ability to divide a text into individual sentences.

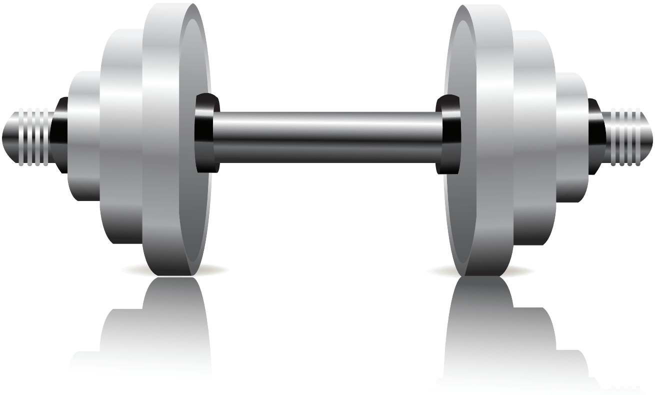
```
import nltk

text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
sents = nltk.sent_tokenize(text)
print(sents[0])
```

- We can write up our own simple sentence tokenizer.

```
sentences = text.split('. ')
words_in_sentences = [sentence.split(' ') for sentence in sentences]
print(sentences[0])
```

- Exercise 11 to 13
- Class-Ex-Lecture2.py



Categorizing and Tagging Words

Defenition

- The process of classifying words into their parts of speech and labeling them accordingly is known as part-of-speech tagging.
- Parts of speech are also known as word classes or lexical categories.
- The collection of tags used for a particular task is known as a tagset.
- Our emphasis is on exploiting tags, and tagging text automatically.

- 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
from nltk import word_tokenize

text = word_tokenize("And now for something completely different")
print(text)
tagged = nltk.pos_tag(text)
print(tagged)
print(nltk.help.upenn_tagset('RB'))
```

- text.similar('the')

```
text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())
print(text.similar('woman'))
print(text.similar('bought'))
print(text.similar('over'))
print(text.similar('the'))
```

A Universal Part-of-Speech Tagset

- Tagged corpora use many different conventions for tagging words. To help us get started, we will be looking at a simplified tagset.

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*

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

Train Tagging - Unigram Tagging

- Unigram taggers are based on a simple statistical algorithm: for each token, assign the tag that is most likely for that particular token.

```
from nltk.corpus import brown
import nltk

brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
print(unigram_tagger.tag(brown_sents[2007]))

print(unigram_tagger.evaluate(brown_tagged_sents))

size = int(len(brown_tagged_sents) * 0.9)
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]
unigram_tagger = nltk.UnigramTagger(train_sents)
print(unigram_tagger.evaluate(test_sents))
```

- Exercise 14 to 16
- Class-Ex-Lecture2.py