Natural Language Processing With Python

Chapter1-Language Processing and Python

Jianzhang Zhang

jianzhang.zhang@foxmail.com

课程考核说明

根据教学大纲要求，本课程的考核办法为：

总评 = 期末成绩 × ＿＿50＿＿ % + 平时成绩 × ＿＿50＿＿%。

1.期末考核方法：闭卷考试

2.平时成绩由＿3＿项构成，具体如下：

（1）＿日常作业＿＿，占比＿30＿%；

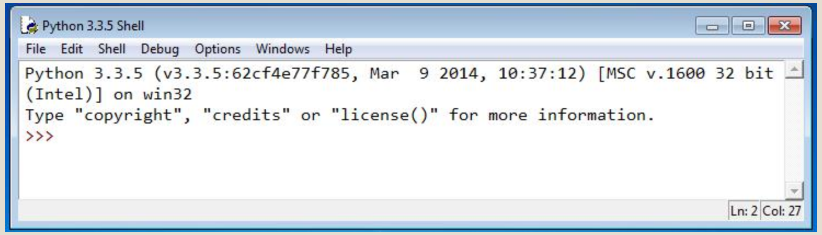
（2）＿考勤＿＿，占比＿10＿%；

（3）＿课堂表现＿＿，占比＿10＿%；

About the course

- Elementary concepts and basics in computing linguistics
- Interesting tasks of natural language processing and the solutions
- Some useful tools and materials for text analysis and ming
- Ability of Python programming for solving interesting language processing problems, such as readability, topic mining, and text quality etc.
- Basic training of scientific paper reading and presentation

Basic tools and useful websites



Basic tools and useful websites (Cont.)

kaggle

python™ NLTK

Natrual Language Tool Kit

GitHub

CSDN

博客园 cnblogs.com

stack overflow

开发者搜索Beta

请输入搜索关键词 搜索

Goals of This Chapter

- 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?

1 Computing with Language: Texts and Words

1.1 Getting Started with Python

- For more study materials, please refer to my course website for *Programming Basics*:

https://zhangjianzhang.github.io/programming_basics/

1.2 Getting Started with NLTK

- Install the latest verson of NLTK

```
pip install nltk
```

- Download NLTK data resource

https://www.nltk.org/nltk_data/ (manually)

OR

```
import nltk

nltk.download()
```

1.2 Getting Started with NLTK (contd.)

```
# Download the text data needed in this chapter
nltk.download('gutenberg')
nltk.download('nps_chat')
nltk.download('inaugural')
```

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
>>>
```

```
>>> text1
<Text: Moby Dick by Herman Melville 1851>
>>> text2
<Text: Sense and Sensibility by Jane Austen 1811>
>>>
```

1.3 Searching Text

- Examine the context of a text

```
>>> text1.concordance("monstrous")
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
>>>
```

- Examibe how words have been used differently over time
- Help you find proper words to be used when wring your own course paper or something else

1.3 Searching Text (contd.)

- Have a new sense of the richness and diversity of language
- Find words in the similar context

```
>>> text1.similar("monstrous")
mean part maddens doleful gamesome subtly uncommon careful untoward
exasperate loving passing mouldy christian few true mystifying
imperial modifies contemptible
>>> text2.similar("monstrous")
very heartily so exceedingly remarkably as vast a great amazingly
extremely good sweet
>>>
```

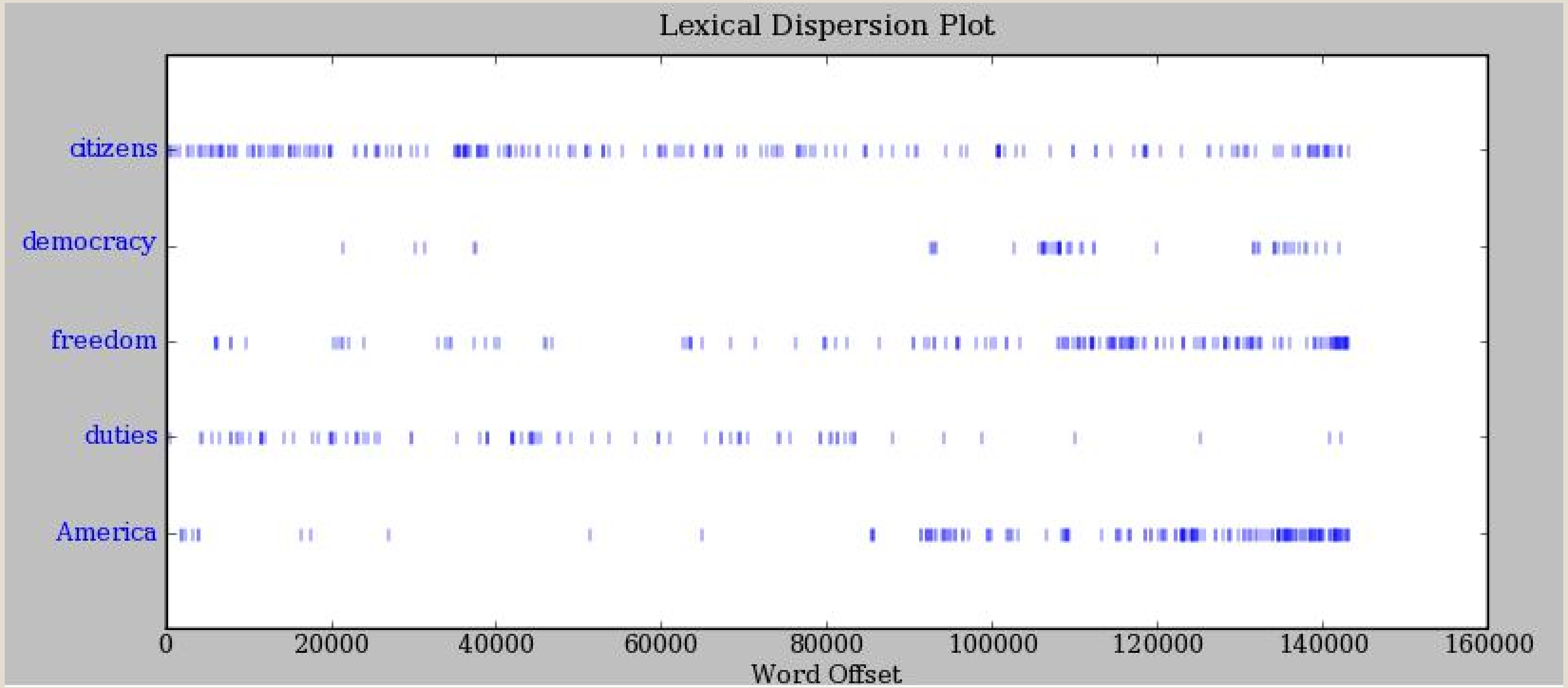
- *Austen* uses this word quite differently from *Melville*

```
>>> text2.common_contexts(["monstrous", "very"])
a_pretty is_pretty am_glad be_glad a_lucky
>>>
```

- Examine the contexts that are shared by two or more words

1.3 Searching Text (contd.)

- Determine the location of a word in the text using a dispersion plot



- Lexical Dispersion Plot for Words in U.S. Presidential Inaugural Addresses: This can be used to investigate changes in language use over time.

1.3 Searching Text (contd.)

- Generate some random text in the various styles we have just seen just for fun

```
>>> text3.generate()
In the beginning of his brother is a hairy man , whose top may reach
unto heaven ; and ye shall sow the land of Egypt there was no bread in
all that he was taken out of the month , upon the earth . So shall thy
wages be ? And they made their father ; and Isaac was old , and kissed
him : and Laban with his cattle in the midst of the hands of Esau thy
first born , and Phichol the chief butler unto his son Isaac , she
>>>
```

1.4 Counting Vocabulary

- Count the words in a text in a variety of useful ways
- Finding out the length of a text from start to finish
- A token is the technical name for a sequence of characters — such as *hairy*, *his*, or *:)* — that we want to treat as a group
- The vocabulary of a text is just the *set* of tokens that it uses
- A word type is the form or spelling of the word independently of its specific occurrences in a text — that is, the word considered as a unique item of vocabulary
- Calculate a measure of the lexical richness of the text
- How often a word occurs in a text, and compute what percentage of the text is taken up by a specific word

1.4 Counting Vocabulary (contd.)

- Lexical Diversity of Various Genres in the Brown Corpus

Lexical Diversity of Various Genres in the *Brown Corpus*

Genre	Tokens	Types	Lexical diversity
skill and hobbies	82345	11935	0.145
humor	21695	5017	0.231
fiction: science	14470	3233	0.223
press: reportage	100554	14394	0.143
fiction: romance	70022	8452	0.121
religion	39399	6373	0.162

2 A Closer Look at Python: Texts as Lists of Words

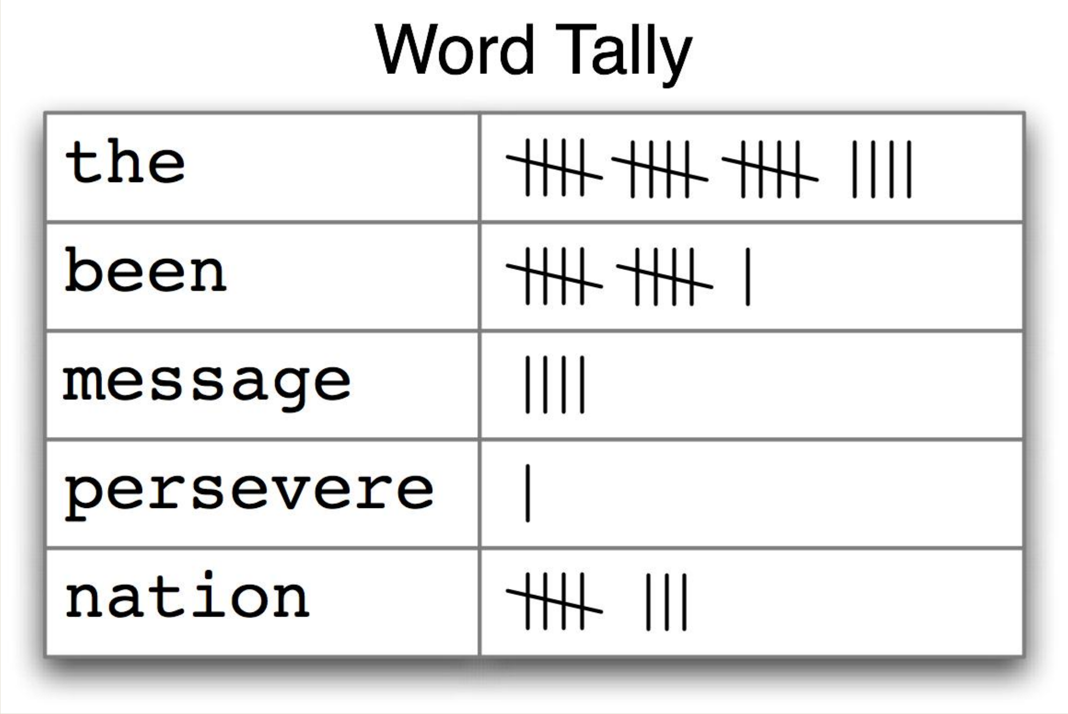
2 A Closer Look at Python: Texts as Lists of Words

- Lists
- Indexing Lists
- Variables
- Strings

3 Computing with Language: Simple Statistics

3.1 Frequency Distributions

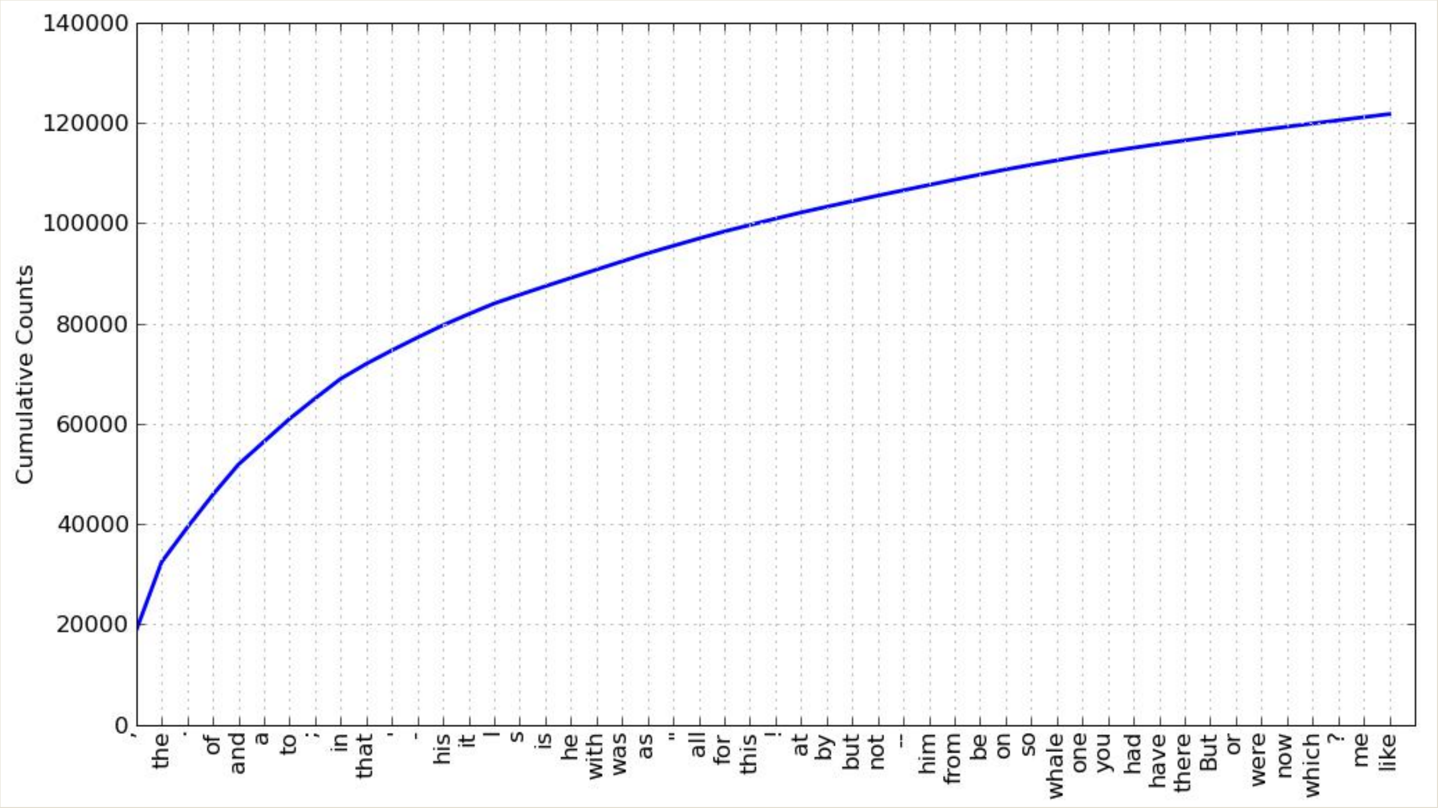
- Identify the words of a text that are most informative about the topic and genre of the text



- Counting Words Appearing in a Text (a **frequency distribution**)

3.1 Frequency Distributions (contd.)

- Cumulative Frequency Plot for 50 Most Frequently Words in Moby Dick: these account for nearly half of the tokens.



- Only one word, whale, is slightly informative!

3.1 Frequency Distributions (contd.)

- If the frequent words don't help us, how about the words that occur once only, the so-called hapaxes?

```
In [80]:  1  fdist1.hapaxes()

Out[80]: ['Herman',
           'Melville',
           ']',
           'ETYMOLOGY',
           'Late',
           'Consumptive',
           'School',
           'threadbare',
           'lexicons',
           'mockingly',
           'flags',
           'mortality',
           'signification',
           'HACKLUYT',
           'Sw',
           'HVAL',
           'roundness',
           'Dut',
           'Ger',
```

- Since neither frequent nor infrequent words help, we need to try something else.

3.2 Fine-grained Selection of Words

- let's look at the long words of a text; perhaps these will be more characteristic and informative

```
>>> V = set(text1)
>>> long_words = [w for w in V if len(w) > 15]
>>> sorted(long_words)
['CIRCUMNAVIGATION', 'Physiognomically', 'apprehensiveness', 'cannibalistically',
'characteristically', 'circumnavigating', 'circumnavigation', 'circumnavigations',
'comprehensiveness', 'hermaphroditical', 'indiscriminately', 'indispensableness',
'irresistibleness', 'physiognomically', 'preternaturalness', 'responsibilities',
'simultaneousness', 'subterraneousness', 'supernaturalness', 'superstitiousness',
'uncomfortableness', 'uncompromisedness', 'undiscriminating', 'uninterpenetratingly']
>>>
```

- These very long words are often hapaxes (i.e., unique) and perhaps it would be better to find frequently occurring long words.

3.2 Fine-grained Selection of Words (contd.)

- At last we have managed to automatically identify the frequently-occurring content-bearing words of the text.

```
>>> fdist5 = FreqDist(text5)
>>> sorted(w for w in set(text5) if len(w) > 7 and fdist5[w] > 7)
['#14-19teens', '#talkcity_adults', '((((((((((', '........', 'Question',
'actually', 'anything', 'computer', 'cute.-ass', 'everyone', 'football',
'innocent', 'listening', 'remember', 'seriously', 'something', 'together',
'tomorrow', 'watching']
>>>
```

- It is a modest but important milestone: a tiny piece of code, processing tens of thousands of words, produces some informative output.

3.3 Collocations and Bigrams

- A collocation is a sequence of words that occur together unusually often
- Thus *red wine* is a collocation, whereas *the wine* is not. A characteristic of collocations is that they are resistant to substitution with words that have similar senses; for example, *maroon wine* sounds definitely odd.
- To get a handle on collocations, we start off by extracting from a text a list of word pairs, also known as bigrams.

```
>>> list(bigrams(['more', 'is', 'said', 'than', 'done']))
[('more', 'is'), ('is', 'said'), ('said', 'than'), ('than', 'done')]
>>>
```

3.3 Collocations and Bigrams (contd.)

- Collocations are essentially just frequent bigrams, except that we want to pay more attention to the cases that involve rare words

```
>>> text4.collocations()
United States; fellow citizens; four years; years ago; Federal
Government; General Government; American people; Vice President; Old
World; Almighty God; Fellow citizens; Chief Magistrate; Chief Justice;
God bless; every citizen; Indian tribes; public debt; one another;
foreign nations; political parties
>>> text8.collocations()
would like; medium build; social drinker; quiet nights; non smoker;
long term; age open; Would like; easy going; financially secure; fun
times; similar interests; Age open; weekends away; poss rship; well
presented; never married; single mum; permanent relationship; slim
build
>>>
```

- Find bigrams that occur more often than we would expect based on the frequency of the individual words.

- The collocations that emerge are very specific to the genre of the texts. In order to find *red wine* as a collocation, we would need to process a much larger body of text.

3.4 Counting Other Things

- Counting words is useful, but we can count other things too. For example, we can look at the distribution of word lengths in a text

```
>>> [len(w) for w in text1] ❶
[1, 4, 4, 2, 6, 8, 4, 1, 9, 1, 1, 8, 2, 1, 4, 11, 5, 2, 1, 7, 6, 1, 3, 4, 5, 2, ...]
>>> fdist = FreqDist(len(w) for w in text1)  ❷
>>> print(fdist)  ❸
<FreqDist with 19 samples and 260819 outcomes>
>>> fdist
FreqDist({3: 50223, 1: 47933, 4: 42345, 2: 38513, 5: 26597, 6: 17111, 7: 14399,
  8: 9966, 9: 6428, 10: 3528, ...})
>>>
```

3.4 Counting Other Things (contd.)

- Functions Defined for NLTK's Frequency Distributions

Example	Description
`fdist = FreqDist(samples)`	create a frequency distribution containing the given samples
`fdist[sample] += 1`	increment the count for this sample
`fdist['monstrous']`	count of the number of times a given sample occurred
`fdist.freq('monstrous')`	frequency of a given sample
`fdist.N()`	total number of samples
`fdist.most_common(n)`	the `n` most common samples and their frequencies
`for sample in fdist:`	iterate over the samples
`fdist.max()`	sample with the greatest count
`fdist.tabulate()`	tabulate the frequency distribution
`fdist.plot()`	graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	cumulative plot of the frequency distribution
`fdist1 |= fdist2`	update `fdist1` with counts from `fdist2`
`fdist1 < fdist2`	test if samples in `fdist1` occur less frequently than in `fdist2`

4 Back to Python: Making Decisions and Taking Control

4.1 Conditionals

- Numerical Comparison Operators

Operator	Relationship
<	less than
<=	less than or equal to
==	equal to (note this is two "=" signs, not one)
!=	not equal to
>	greater than
>=	greater than or equal to

- Some Word Comparison Operators

Function	Meaning
`s.startswith(t)`	test if `s` starts with `t`
`s.endswith(t)`	test if `s` ends with `t`
`t in s`	test if `t` is a substring of `s`
`s.islower()`	test if `s` contains cased characters and all are lowercase
`s.isupper()`	test if `s` contains cased characters and all are uppercase
`s.isalpha()`	test if `s` is non-empty and all characters in `s` are alphabetic
`s.isalnum()`	test if `s` is non-empty and all characters in `s` are alphanumeric
`s.isdigit()`	test if `s` is non-empty and all characters in `s` are digits
`s.istitle()`	test if `s` contains cased characters and is titlecased (i.e. all words in `s` have initial capitals)

4.2 Operating on Every Element

```
>>> [len(w) for w in text1]
[1, 4, 4, 2, 6, 8, 4, 1, 9, 1, 1, 8, 2, 1, 4, 11, 5, 2, 1, 7, 6, 1, 3, 4, 5, 2, ...]
>>> [w.upper() for w in text1]
['[', 'MOBY', 'DICK', 'BY', 'HERMAN', 'MELVILLE', '1851', ']', 'ETYMOLOGY', '.', ...]
>>>
```

```
>>> len(text1)
260819
>>> len(set(text1))
19317
>>> len(set(word.lower() for word in text1))
17231
>>>
```

```
>>> len(set(word.lower() for word in text1 if word.isalpha()))
16948
>>>
```

4.3 Nested Code Blocks

```
>>> word = 'cat'
>>> if len(word) < 5:
...     print('word length is less than 5')
...    ❶
word length is less than 5
>>>
```

```
>>> if len(word) >= 5:
...   print('word length is greater than or equal to 5')
...
>>>
```

```
>>> for word in ['Call', 'me', 'Ishmael', '.']:
...     print(word)
...
Call
me
Ishmael
.
>>>
```

4.4 Looping with Conditions

```
>>> sent1 = ['Call', 'me', 'Ishmael', '.']
>>> for xyzzy in sent1:
...     if xyzzy.endswith('l'):
...         print(xyzzy)
...
Call
Ishmael
>>>
```

```
>>> for token in sent1:
...     if token.islower():
...         print(token, 'is a lowercase word')
...     elif token.istitle():
...         print(token, 'is a titlecase word')
...     else:
...         print(token, 'is punctuation')
...
Call is a titlecase word
me is a lowercase word
Ishmael is a titlecase word
. is punctuation
>>>
```

```
>>> tricky = sorted(w for w in set(text2) if 'cie' in w or 'cei' in w)
>>> for word in tricky:
...     print(word, end=' ')
ancient ceiling conceit conceited conceive conscience
conscientious conscientiously deceitful deceive ...
>>>
```

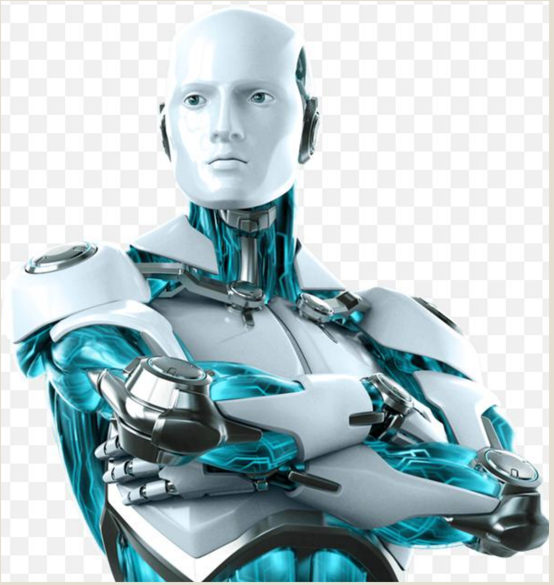
5 Automatic Natural Language Understanding

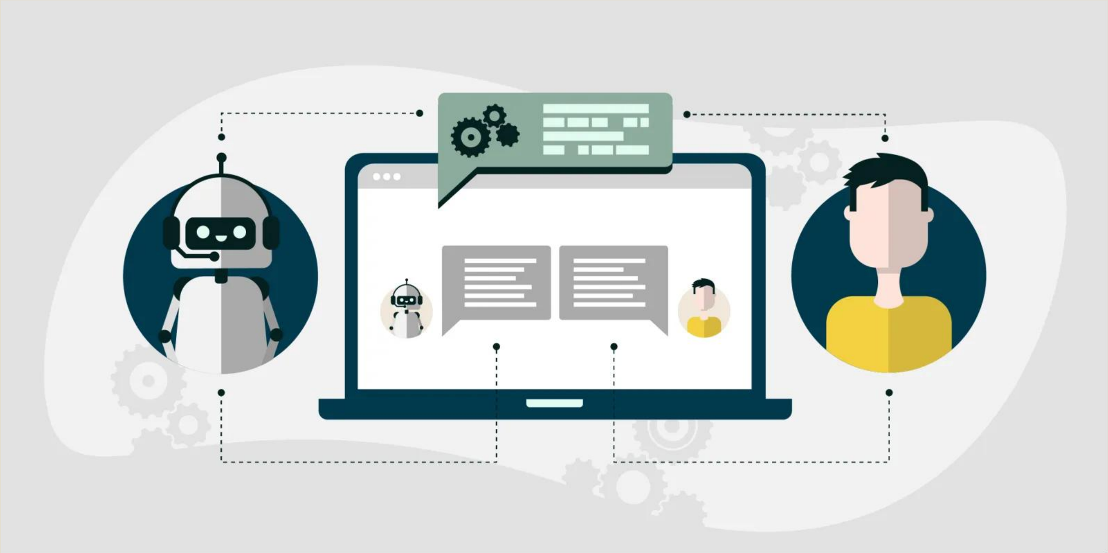
5 Automatic Natural Language Understanding

- Search engines have been crucial to the growth and popularity of the Web, but have some shortcomings. 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?*
- *What do experts say about digital SLR cameras?*
- *What predictions about the steel market were made by credible commentators in the past week?*
- Getting a computer to answer them automatically involves a range of language processing tasks, including information extraction, inference, and summarization, and would need to be carried out on a scale and with a level of robustness that is still beyond our current capabilities.

5 Automatic Natural Language Understanding

- On a more philosophical level, a long-standing challenge within artificial intelligence has been to build intelligent machines, and a major part of intelligent behaviour is understanding language.





- As NLP technologies become more mature, and robust methods for analyzing unrestricted text become more widespread, the prospect of natural language understanding has re-emerged as a plausible goal.

5.1 Word Sense Disambiguation

- In word sense disambiguation we want to work out which sense of a word was intended in a given context. Consider the ambiguous words *serve* and *dish* :

  a. ***serve**: help with food or drink; hold an office; put ball into play*

  b. ***dish**: plate; course of a meal; communications device*

he served the dish

他端上了这道菜（很自然想到的场景和画面）

他在摔盘子（网球运动员在球场边瓷茶具，可能在发火，三个词同时包含运动和瓷器这两个不同的主题，一般不太可能）

We automatically disambiguate words using context, exploiting the simple fact that nearby words have closely related meanings.

5.1 Word Sense Disambiguation (contd.)

- The meaning of the italicized word helps us interpret the meaning of *by*

a. The lost children were found by *the searchers* (agentive)

b. The lost children were found by *the mountain* (locative)

c. The lost children were found by *the afternoon* (temporal)

We automatically disambiguate words using context, exploiting the simple fact that nearby words have closely related meanings.

5.2 Pronoun Resolution

- A deeper kind of language understanding is to work out "who did what to whom" — i.e., to detect the subjects and objects of verbs.
- Try to determine what was sold, caught, and found (one case is ambiguous) :

  a. The thieves stole the paintings. They were subsequently *sold*.

  b. The thieves stole the paintings. They were subsequently *caught*.

  c. The thieves stole the paintings. They were subsequently *found*.

- Answering this question involves finding the antecedent of the pronoun they, either thieves or paintings. Computational techniques for tackling this problem include anaphora resolution — identifying what a pronoun or noun phrase refers to — and semantic role labeling — identifying how a noun phrase relates to the verb (as agent, patient, instrument, and so on).

5.3 Generating Language Output

- Question answering: a machine should be able to answer a user's questions relating to collection of texts.

  a. Text: ... The thieves stole the paintings. They were subsequently sold. ...

  b. Human: Who or what was sold?

  c. Machine: The paintings.

- Working out the sense of a word, the subject of a verb, and the antecedent of a pronoun are steps in establishing the meaning of a sentence, things we would expect a language understanding system to be able to do.

5.4 Machine translation

- Machine translation (MT) has been the holy grail of language understanding, ultimately seeking to provide high-quality, idiomatic translation between any pair of languages.

0> how long before the next flight to Alice Springs?
1> wie lang vor dem folgenden Flug zu Alice Springs?
2> how long before the following flight to Alice jump?
3> wie lang vor dem folgenden Flug zu Alice springen Sie?
4> how long before the following flight to Alice do you jump?
5> wie lang, bevor der folgende Flug zu Alice tun, Sie springen?
6> how long, before the following flight to Alice does, do you jump?
7> wie lang bevor der folgende Flug zu Alice tut, tun Sie springen?
8> how long before the following flight to Alice does, do you jump?
9> wie lang, bevor der folgende Flug zu Alice tut, tun Sie springen?
10> how long, before the following flight does to Alice, do do you jump?
11> wie lang bevor der folgende Flug zu Alice tut, Sie tun Sprung?
12> how long before the following flight does leap to Alice, does you?

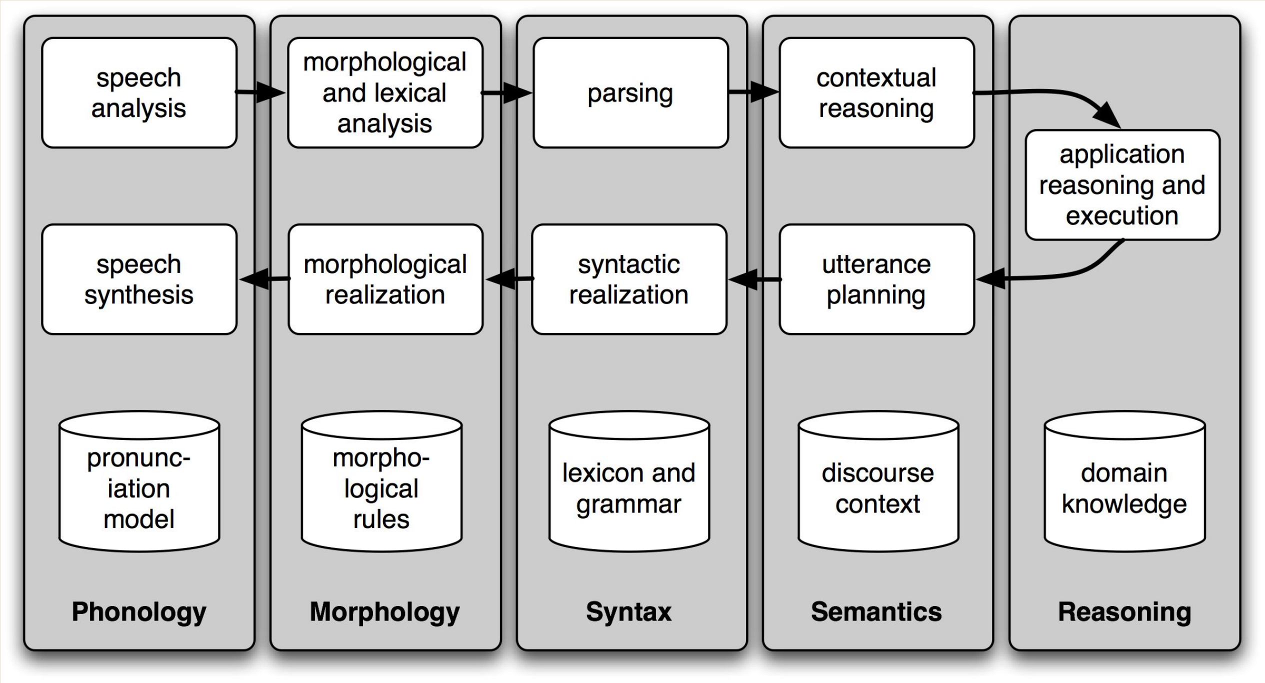
- Translating a sentence back and forth between a pair of languages until equilibrium is reached. (reveal shortcomings of MT)

5.4 Machine translation (contd.)

- Today these difficulties are being faced by collecting massive quantities of parallel texts from news and government websites that publish documents in two or more languages. Given a document in German and English, and possibly a bilingual dictionary, we can automatically pair up the sentences, a process called text alignment. Once we have a million or more sentence pairs, we can detect corresponding words and phrases, and build a model that can be used for translating new text.

5.5 Spoken Dialog Systems

- In the history of artificial intelligence, the chief measure of intelligence has been a linguistic one, namely the **Turing Test**: can a dialogue system, responding to a user's text input, perform so naturally that we cannot distinguish it from a human-generated response? In contrast, today's commercial dialogue systems are very limited, but still perform useful functions in narrowly-defined domains.



5.5 Spoken Dialog Systems (contd.)

- Dialogue systems give us an opportunity to mention the commonly assumed pipeline for NLP.

- The above figure shows the architecture of a simple dialogue system. Along the top of the diagram, moving from left to right, is a "pipeline" of some language understanding components. These map from speech input via syntactic parsing to some kind of meaning representation. Along the middle, moving from right to left, is the reverse pipeline of components for converting concepts to speech. These components make up the dynamic aspects of the system. At the bottom of the diagram are some representative bodies of static information: the repositories of language-related data that the processing components draw on to do their work.

5.6 Spoken Dialog Systems (contd.)

- Recognizing Textual Entailment (RTE): find evidence from the Text to support or deny the Hypothesis.

  a. Text: *David Golinkin is the editor or author of eighteen books, and over 150 responsa, articles, sermons and books*

  b. Hypothesis: *Golinkin has written eighteen books*

- In order to determine whether the hypothesis is supported by the text, the system needs the following background knowledge: (i) if someone is an author of a book, then he/she has written that book; (ii) if someone is an editor of a book, then he/she has not written (all of) that book; (iii) if someone is editor or author of eighteen books, then one cannot conclude that he/she is author of eighteen books.

5.7 Limitations of NLP

- Despite the research-led advances in tasks like RTE, natural language systems that have been deployed for real-world applications still cannot perform common-sense reasoning or draw on world knowledge in a general and robust manner.
- Right from the beginning, an important goal of NLP research has been to make progress on the difficult task of building technologies that "understand language," using superficial yet powerful techniques instead of unrestricted knowledge and reasoning capabilities.

6 Summary

- Texts are represented in Python using lists: ['Monty', 'Python']. We can use indexing, slicing, and the len() function on lists.
- A word "token" is a particular appearance of a given word in a text; a word "type" is the unique form of the word as a particular sequence of letters. We count word tokens using len(text) and word types using len(set(text)).
- We obtain the vocabulary of a text t using sorted(set(t)).
- We operate on each item of a text using [f(x) for x in text].
- To derive the vocabulary, collapsing case distinctions and ignoring punctuation, we can write set(w.lower() for w in text if w.isalpha()).
- We process each word in a text using a for statement, such as for w in t: or for word in text:. This must be followed by the colon character and an indented block of code, to be executed each time through the loop.

- We test a condition using an if statement: if len(word) < 5:. This must be followed by the colon character and an indented block of code, to be executed only if the condition is true.
- A frequency distribution is a collection of items along with their frequency counts (e.g., the words of a text and their frequency of appearance).
- A function is a block of code that has been assigned a name and can be reused. Functions are defined using the def keyword, as in def mult(x, y); x and y are parameters of the function, and act as placeholders for actual data values.
- A function is called by specifying its name followed by zero or more arguments inside parentheses, like this: texts(), mult(3, 4), len(text1).

7 Further Reading

- ACL Anthology: https://aclanthology.org/
- Speech and Language Processing (3rd ed. draft) by Dan Jurafsky and James H. Martin: https://web.stanford.edu/~jurafsky/slp3/