Natural Language Understanding

1.5 5 Automatic Natural Language Understanding

Goals

- We have explored language bottom-up. We step back from the code to paint a bigger picture of NLP
- At a purely **practical level**, 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 involves information extraction, **inference**, and summarization
- 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**

5 Automatic Natural Language Understanding

- Word Sense Disambiguation
- Pronoun Resolution
- Generating Language Output
- Machine Translation
- Spoken Dialog Systems
- Textual Entailment

Word Sense Disambiguation

- 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

  In ‘he served the dish’, both serve and dish are being used with their food meanings.
- We disambiguate words using context; nearby words have closely related meanings.
- As another example, consider the word **by**, which has several meanings, e.g.: the book by Chesterton (agentive — Chesterton was the author of the book); the cup by the stove (locative — the stove is where the cup is); and submit by Friday (temporal — Friday is the time of the submitting). Observe in (c) that 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)

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.

- **Anaphora resolution** identifies 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).

```
sentences = [
    "John is a man. He walks",
    "John and Mary are married. They have two kids",
    "In order for Ravi to be successful, he should follow John",
    "John met Mary in Barista. She asked him to order a Pizza"]
males = [(name, 'male') for name in
names.words('male.txt')]
females = [(name, 'female') for name in
names.words('female.txt')]
combined = males + females
random.shuffle(combined)
training = [(feature(name), gender) for (name, gender) in
combined]
classifier = nltk.NaiveBayesClassifier.train(training)
learnAnaphora(sentences)
```

[Run]
John is a man. He walks
====> Anaphora: **[('John', 'male'), 'He']**
====> [('John', 'NNP', 'B-PERSON'), ('is', 'VBZ', 'O'), ('a', 'DT', 'O'), ('man', 'NN', 'O'), ('.', '.', 'O'), ('He', 'PRP', 'O'), ('walks', 'VBD', 'O')]

```
import nltk
from nltk.chunk import tree2conlltags
from nltk.corpus import names
import random
def feature(word):
    return {'last(1)' : word[-1]}
def gender(word):
    return classifier.classify(feature(word))

def learnAnaphora(sentences):
    for sent in sentences:
        chunks =
nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sent)),
binary=False)
        stack = []
        print(sent)
        items = tree2conlltags(chunks)
        for item in items:
            if item[1] == 'NNP' and (item[2] == 'B-PERSON' or
item[2] == 'O'):
                stack.append((item[0], gender(item[0])))
            elif item[1] == 'CC':
                stack.append(item[0])
            elif item[1] == 'PRP':
                stack.append(item[0])
        print("====> Anaphora: {}".format(stack))
        print("====>", items)
```

Output

John is a man. He walks
====> Anaphora: [('John', 'male'), 'He']
====> [('John', 'NNP', 'B-PERSON'), ('is', 'VBZ', 'O'), ('a', 'DT', 'O'), ('man', 'NN', 'O'), ('.', '.', 'O'), ('He', 'PRP', 'O'), ('walks', 'VBD', 'O')]
John and Mary are married. They have two kids
====> Anaphora: [('John', 'male'), 'and', ('Mary', 'female'), 'They']
====> [('John', 'NNP', 'B-PERSON'), ('and', 'CC', 'O'), ('Mary', 'NNP', 'O'), ('are', 'VBP', 'O'), ('married', 'VBN', 'O'), ('.', '.', 'O'), ('They', 'PRP', 'O'), ('have', 'VBP', 'O'), ('two', 'CD', 'O'), ('kids', 'NNS', 'O')]
In order for Ravi to be successful, he should follow John
====> Anaphora: [('Ravi', 'female'), 'he', ('John', 'male')]
====> [('In', 'IN', 'O'), ('order', 'NN', 'O'), ('for', 'IN', 'O'), ('Ravi', 'NNP', 'B-PERSON'), ('to', 'TO', 'O'), ('be', 'VB', 'O'), ('successful', 'JJ', 'O'), (',', ',', 'O'), ('he', 'PRP', 'O'), ('should', 'MD', 'O'), ('follow', 'VB', 'O'), ('John', 'NNP', 'B-PERSON')]
John met Mary in Barista. She asked him to order a Pizza
====> Anaphora: [('John', 'male'), ('Mary', 'female'), 'She', 'him']
====> [('John', 'NNP', 'B-PERSON'), ('met', 'VBD', 'O'), ('Mary', 'NNP', 'O'), ('in', 'IN', 'O'), ('Barista', 'NNP', 'B-GPE'), ('.', '.', 'O'), ('She', 'PRP', 'O'), ('asked', 'VBD', 'O'), ('him', 'PRP', 'O'), ('to', 'TO', 'O'), ('order', 'NN', 'O'), ('a', 'DT', 'O'), ('Pizza', 'NN', 'O')]

Generating Language Output

Question answering system answers 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.

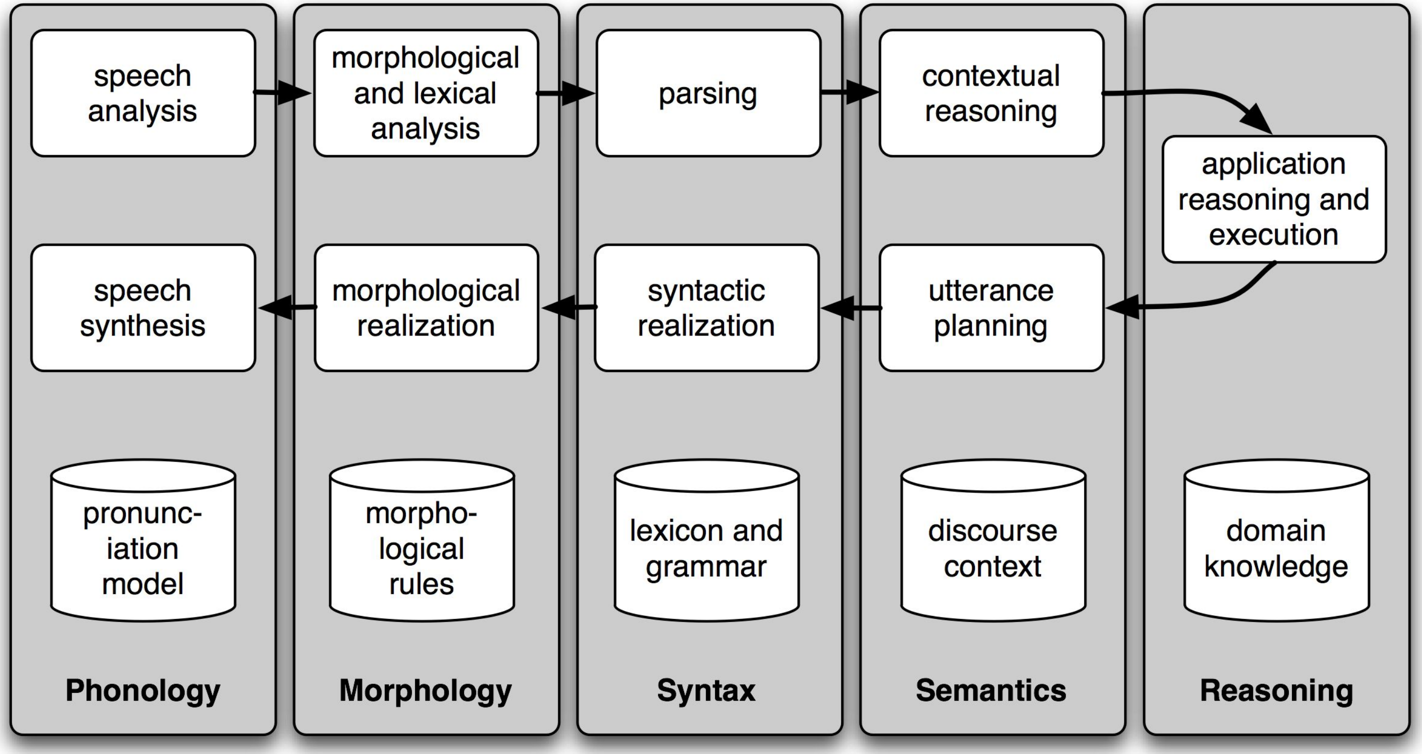
Machine Translation

- Practical translation systems exist. However, these systems have some serious shortcomings, which are revealed by translating a sentence back and forth between a pair of languages, e.g.:
  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?

Spoken Dialog Systems

- Today's commercial dialogue systems are very limited, but still perform useful functions in narrowly-defined domains, as we see here:

  S: How may I help you?
  U: When is Saving Private Ryan playing?
  S: For what theater?
  U: The Paramount theater.
  S: Saving Private Ryan is not playing at the Paramount theater, but
  it's playing at the Madison theater at 3:00, 5:30, 8:00, and 10:30.
- We do not ask this system to provide driving instructions or details of nearby restaurants unless the required **information** had already been stored and suitable **question-answer pairs** had been incorporated into the language processing system
- To understand the **user's goals**, **inference** is necessary in order to interact naturally. Without it, when asked Do you know when Saving Private Ryan is playing?, a system might respond with a cold Yes.



- The developers of commercial dialogue systems use **contextual assumptions and business logic** to ensure that the different ways in which a user might express requests or provide information are handled in a way that makes sense for the particular application.
- So, if you type When is ..., or I want to know when ..., or Can you tell me when ..., simple rules will always yield screening times. This is enough for the system to provide a useful service

Textual Entailment

- The challenge of language understanding has been brought into focus in recent years called **Recognizing Textual Entailment (RTE)**

  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

- To determine whether B (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.

Limitations of NLP

- 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
- we hope to equip you with the knowledge and skills to build useful NLP systems, and to contribute to the long-term aspiration of building intelligent machines

Python NLP toolkit

- **TextBlob** - Providing a consistent API for diving into common natural language processing (NLP) tasks. Stands on the giant shoulders of NLTK and Pattern, and plays nicely with both
- **spaCy** - Industrial strength NLP with Python and Cython.
- PyStanfordDependencies - Python interface for converting Penn Treebank trees to Stanford Dependencies
- Polyglot - Multilingual text (NLP) processing toolkit.
- nut - Natural language Understanding Toolkit.
- Distance - Levenshtein and Hamming distance computation.
- textacy - higher-level NLP built on Spacy
- stanford-corenlp-python - Python wrapper for Stanford CoreNLP
- **editdistance** - fast implementation of edit distance.

https://github.com/josephmisiti/awesome-machine-learning#python-nlp

Python NLP toolkit

- Pattern - A web mining module for the Python programming language. It has tools for natural language processing, machine learning, among others
- Quepy - A python framework to transform natural language questions to queries in a database query language.
- YAlign - A sentence aligner, a friendly tool for extracting parallel sentences from comparable corpora.
- Rosetta - Text processing tools and wrappers (e.g. Vowpal Wabbit)
- PyNLPl - Python Natural Language Processing Library. General purpose NLP library for Python. Also contains some specific modules for parsing common NLP formats, most notably for FoLiA, but also ARPA language models, Moses phrasetables, GIZA++ alignments.
- rasa_nlu - turn natural language into structured data.
- yase - Transcode sentence (or other sequence) to list of word vector .
- DrQA - Reading Wikipedia to answer open-domain questions.

https://github.com/josephmisiti/awesome-machine-learning#python-nlp