# Natural Language Processing Toolkit NLTK

Maria Ilieva

University of Tuebingen

21 November,2017

# Overview

# Natural Language Understanding

| Discipline | Typical Problems | Tools |
|---|---|---|
| Linguists | How do words form phrases and sentences? What constrains the possible meanings for a sentence? | Intuitions about well-formedness and meaning; mathematical models of structure (for example, formal language theory, model theoretic semantics) |
| Psycholinguists | How do people identify the structure of sentences? How are word meanings identified? When does understanding take place? | Experimental techniques based on measuring human performance; statistical analysis of observations |
| Philosophers | What is meaning, and how do words and sentences acquire it? How do words identify objects in the world? | Natural language argumentation using intuition about counter-examples; mathematical models (for example, logic and model theory) |
| Computational Linguists | How is the structure of sentences identified? How can knowledge and reasoning be modeled? How can language be used to accomplish specific tasks? | Algorithms, data structures; formal models of representation and reasoning; AI techniques (search and representation methods) |

Figure: (Allen 1995)The major disciplines studying language

# Applications of Natural Language Understanding

- Text-based applications (processing of written text)
  - finding appropriate documents on certain topics from a database of texts
    (e.g, finding relevant books in a library)
  - extracting information from messages or articles on certain topics
    (e.g, document classification)
  - translating documents from one language to another
    (e.g, producing automobile repair manuals in many different languages)
  - summarizing texts for certain purposes

# Applications of Natural Language Understanding

- Dialogue-based applications (human-machine communication)
  - question-answering systems, where natural language is used to query a database
  - automated customer service over the telephone (e.g, to perform banking transactions)
  - tutoring systems, where the machine interacts with a student
  - spoken language control of a machine
  - general cooperative problem-solving systems

# Different levels of Language Analysis

1. Phonetic and phonological knowledge: concerns how words are related to the sounds that realize them
2. Morphological knowledge: concerns how words are constructed from more basic meaning units called morphemes.
3. Syntactic knowledge: concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.
4. Semantic knowledge: concerns what words mean and how these meanings combine in sentences to form sentence meanings.
5. Pragmatic knowledge: concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

# NLP Technologies

- Word Sense Disambiguation: automatically disambiguate words using context, exploiting the simple fact that nearby words have closely related meanings.
- Pronoun Resolution:
  - anaphora resolution: identifying what a pronoun or noun phrase refers to
  - semantic role labeling: identifying how a noun phrase relates to the verb (as agent, patient, instrument, and so on).
- Machine Translation: translating a source language into a target language(e.g English to German translation)
- Spoken Dialog Systems: a dialogue system, responding to a user's text input, perform so naturally that we cannot distinguish it from a human-generated response

# NLTK Design Criteria Requirements

- Ease of Use: allow students to concentrate on building NLP systems.
- Consistency: use consistent data structures and interfaces.
- Extensibility: accommodate new components, whether those components replicate or extend the toolkit's existing functionality
- Documentation: its data structures, and its implementation all need to be carefully and thoroughly documented
- Simplicity: each class defined by the toolkit should be simple enough so that a learner could implement it
- Modularity: interaction between different components of the toolkit should be simple, well-defined interfaces.

# NLTK Modules

- Parsing Modules: designed for various parsing techniques
- Tagging Modules: `tagger` module defines a standard interface for augmenting each token of a text with supplementary information
- Finite State Automata: `fsa` module defines a data type for encoding finite state automata
- Visualization Modules: define graphical interfaces for viewing and manipulating data structures, and graphical tools for experimenting with NLP tasks.
- Text Classification Modules: designed for various text classifiers

# Parsing Modules

- parser module: defines a high-level interface for producing trees that represent the structures of texts
- chunkparser: defines a sub-interface for parsers that identify non-overlapping linguistic groups
- srparser: implements a simple shift-reduce parser.
- chartparser: defines a flexible parser that uses a *chart* to record hypotheses about syntactic constituents
- pcfgparser: provides a variety of different parsers for probabilistic grammars

# Visualization Modules

- draw.tree module: provides a simple graphical interface for displaying tree structures
- draw.plot_graph: used to graph mathematical functions
- draw.fsa: provides a graphical tool for displaying and simulating finite state automata
- draw.chart: provides an interactive graphical tool for experimenting with chart parsers

# Classification Modules

- classifier module: defines a standard interface for classifying texts into categories
- classifier.naivebayes: defines a text classifier based on the Naive Bayes assumption
- classifier.feature: provides a standard encoding for the information that is used to make decisions for a particular classification task
- classifier.featureselection: defines a standard interface for choosing which features are relevant for a particular classification task

# NLTK HOW TO Dependency Parsing

Figure: Import modules

```
>>> from nltk.grammar import DependencyGrammar
>>> from nltk.parse import (
...     DependencyGraph,
...     ProjectiveDependencyParser,
...     NonprojectiveDependencyParser,
... )
```

Figure: Treebank Data

```
>>> treebank_data = """Pierre  NNP     2       NMOD
... Vinken  NNP     8       SUB
... ,       ,       2       P
... 61      CD      5       NMOD
... years   NNS     6       AMOD
... old     JJ      2       NMOD
... ,       ,       2       P
... will    MD      0       ROOT
... join    VB      8       VC
... the     DT      11      NMOD
... board   NN      9       OBJ
... as      IN      9       VMOD
... a       DT      15      NMOD
... nonexecutive    JJ      15      NMOD
... director        NN      12      PMOD
... Nov.    NNP     9       VMOD
... 29      CD      16      NMOD
... .       .       9       VMOD
... """

>>> dg = DependencyGraph(treebank_data)
```

# NLTK HOW TO Dependency Parsing

## Figure: Dependency Data

Using the dependency-parsed version of the Penn Treebank corpus sample.

```
>>> from nltk.corpus import dependency_treebank
>>> t = dependency_treebank.parsed_sents()[0]
>>> print(t.to_conll(3))  # doctest: +NORMALIZE_WHITESPACE
Pierre      NNP     2
Vinken      NNP     8
,   ,       2
61  CD      5
years       NNS     6
old JJ      2
,   ,       2
will        MD      0
join        VB      8
the DT      11
board       NN      9
as  IN      9
a   DT      15
nonexecutive        JJ      15
director    NN      12
Nov.        NNP     9
29  CD      16
.   .       8
```

Figure: Projective Parsing

# Projective Dependency Parsing

```
>>> grammar = DependencyGrammar.fromstring("""
... 'fell' -> 'price' | 'stock'
... 'price' -> 'of' 'the'
... 'of' -> 'stock'
... 'stock' -> 'the'
... """)
>>> print(grammar)
Dependency grammar with 5 productions
  'fell' -> 'price'
  'fell' -> 'stock'
  'price' -> 'of' 'the'
  'of' -> 'stock'
  'stock' -> 'the'

>>> dp = ProjectiveDependencyParser(grammar)
>>> for t in sorted(dp.parse(['the', 'price', 'of', 'the', 'stock', 'fell'])):
...     print(t)
(fell (price the (of (stock the))))
(fell (price the of) (stock the))
(fell (price the of the) stock)
```

# NLTK HOW TO Discourse Representation Theory

## Figure: DRT Overview

A DRS can be created with the `DRS()` constructor. This takes two arguments: a list of discourse referents and list of conditions. .

```
>>> from nltk.sem.drt import *
>>> dexpr = DrtExpression.fromstring
>>> man_x = dexpr('man(x)')
>>> walk_x = dexpr('walk(x)')
>>> x = dexpr('x')
>>> print(DRS([x], [man_x, walk_x]))
([x],[man(x), walk(x)])
```

The `parse()` method can also be applied directly to DRS expressions, which allows them to be specified more easily.

```
>>> drs1 = dexpr('([x],[man(x),walk(x)])')
>>> print(drs1)
([x],[man(x), walk(x)])
```

DRSs can be *merged* using the + operator.

```
>>> drs2 = dexpr('([y],[woman(y),stop(y)])')
>>> drs3 = drs1 + drs2
>>> print(drs3)
(([x],[man(x), walk(x)]) + ([y],[woman(y), stop(y)]))
>>> print(drs3.simplify())
([x,y],[man(x), walk(x), woman(y), stop(y)])
```

# NLTK HOW TO Sentiment Analysis

Figure: Import modules

```
>>> from nltk.classify import NaiveBayesClassifier
>>> from nltk.corpus import subjectivity
>>> from nltk.sentiment import SentimentAnalyzer
>>> from nltk.sentiment.util import *

>>> n_instances = 100
>>> subj_docs = [(sent, 'subj') for sent in subjectivity.sents(categories='subj')[:n_instances]]
>>> obj_docs = [(sent, 'obj') for sent in subjectivity.sents(categories='obj')[:n_instances]]
>>> len(subj_docs), len(obj_docs)
(100, 100)
```

## Figure: Feature extraction

Each document is represented by a tuple (sentence, label). The sentence is tokenized, so it is represented by a list of strings:

```
>>> subj_docs[0]
(['smart', 'and', 'alert', ',', 'thirteen', 'conversations', 'about', 'one',
'thing', 'is', 'a', 'small', 'gem', '.'], 'subj')
```

We separately split subjective and objective instances to keep a balanced uniform class distribution in both train and test sets.

```
>>> train_subj_docs = subj_docs[:80]
>>> test_subj_docs = subj_docs[80:100]
>>> train_obj_docs = obj_docs[:80]
>>> test_obj_docs = obj_docs[80:100]
>>> training_docs = train_subj_docs+train_obj_docs
>>> testing_docs = test_subj_docs+test_obj_docs

>>> sentim_analyzer = SentimentAnalyzer()
>>> all_words_neg = sentim_analyzer.all_words([mark_negation(doc) for doc in training_docs])
```

We use simple unigram word features, handling negation:

```
>>> unigram_feats = sentim_analyzer.unigram_word_feats(all_words_neg, min_freq=4)
>>> len(unigram_feats)
83
>>> sentim_analyzer.add_feat_extractor(extract_unigram_feats, unigrams=unigram_feats)
```

We apply features to obtain a feature-value representation of our datasets:

```
>>> training_set = sentim_analyzer.apply_features(training_docs)
>>> test_set = sentim_analyzer.apply_features(testing_docs)
```

## Figure: Training

We can now train our classifier on the training set, and subsequently output the evaluation results:

```
>>> trainer = NaiveBayesClassifier.train
>>> classifier = sentim_analyzer.train(trainer, training_set)
Training classifier
>>> for key,value in sorted(sentim_analyzer.evaluate(test_set).items()):
...     print('{0}: {1}'.format(key, value))
Evaluating NaiveBayesClassifier results...
Accuracy: 0.8
F-measure [obj]: 0.8
F-measure [subj]: 0.8
Precision [obj]: 0.8
Precision [subj]: 0.8
Recall [obj]: 0.8
Recall [subj]: 0.8
```

# NLTK HOW TO Feature Structures

## Figure: Import modules

```
>>> from __future__ import print_function
>>> from nltk.featstruct import FeatStruct
>>> from nltk.sem.logic import Variable, VariableExpression, Expression
```

## Figure: FS Overview

A feature structure is a mapping from feature identifiers to feature values, where feature values can be simple values

```
>>> fs1 = FeatStruct(number='singular', person=3)
>>> print(fs1)
[ number = 'singular' ]
[ person = 3          ]
```

Feature structure may be nested:

```
>>> fs2 = FeatStruct(type='NP', agr=fs1)
>>> print(fs2)
[ agr  = [ number = 'singular' ] ]
[        [ person = 3          ] ]
[                                ]
[ type = 'NP'                    ]
```

# Accessing Corpora

1. Gutenberg Corpus

Figure: Accessing Alice in Wonderland Corpus

```
>>> import nltk
>>> nltk.corpus.gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt',
t', 'chesterton-ball.txt', 'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt'
esar.txt', 'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
>>> alice = nltk.corpus.gutenberg.words('carroll-alice.txt')
>>> len(alice)
34110
```

## 2. Simple Statistics

Figure: Statistics for each text in Gutenberg Corpus

```
>>> 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)
...
5 25 26 austen-emma.txt
5 26 17 austen-persuasion.txt
5 28 22 austen-sense.txt
4 34 79 bible-kjv.txt
5 19 5 blake-poems.txt
4 19 14 bryant-stories.txt
4 18 12 burgess-busterbrown.txt
4 20 13 carroll-alice.txt
5 20 12 chesterton-ball.txt
5 23 11 chesterton-brown.txt
5 18 11 chesterton-thursday.txt
4 21 25 edgeworth-parents.txt
5 26 15 melville-moby_dick.txt
5 52 11 milton-paradise.txt
4 12 9 shakespeare-caesar.txt
4 12 8 shakespeare-hamlet.txt
4 12 7 shakespeare-macbeth.txt
5 36 12 whitman-leaves.txt
```

# 3. Sentence analysis

Figure: Sentence segmentation for Alice in Wonderland Corpus

```
>>> alice_sentences = nltk.corpus.gutenberg.sents('carroll-alice.txt')
>>> alice_sentences
[['[', 'Alice', "'", 's', 'Adventures', 'in', 'Wonderland', 'by', 'Lewis', 'C
arroll', '1865', ']'], ['CHAPTER', 'I', '.'], ...]
>>> longest_len = max(len(s) for s in alice_sentences)
>>> [s for s in alice_sentences if len(s) == longest_len]
[['Hardly', 'knowing', 'what', 'she', 'did', ',', 'she', 'picked', 'up', 'a',
'little', 'bit', 'of', 'stick', ',', 'and', 'held', 'it', 'out', 'to', 'the',
'puppy', ';', 'whereupon', 'the', 'puppy', 'jumped', 'into', 'the', 'air', 'o
ff', 'all', 'its', 'feet', 'at', 'once', ',', 'with', 'a', 'yelp', 'of', 'del
ight', ',', 'and', 'rushed', 'at', 'the', 'stick', ',', 'and', 'made', 'belie
ve', 'to', 'worry', 'it', ';', 'then', 'Alice', 'dodged', 'behind', 'a', 'gre
at', 'thistle', ',', 'to', 'keep', 'herself', 'from', 'being', 'run', 'over',
';', 'and', 'the', 'moment', 'she', 'appeared', 'on', 'the', 'other', 'side',
',', 'the', 'puppy', 'made', 'another', 'rush', 'at', 'the', 'stick', ',', 'a
nd', 'tumbled', 'head', 'over', 'heels', 'in', 'its', 'hurry', 'to', 'get', '
hold', 'of', 'it', ';', 'then', 'Alice', ',', 'thinking', 'it', 'was', 'very'
, 'like', 'having', 'a', 'game', 'of', 'play', 'with', 'a', 'cart', '-', 'hor
se', ',', 'and', 'expecting', 'every', 'moment', 'to', 'be', 'trampled', 'und
er', 'its', 'feet', ',', 'ran', 'round', 'the', 'thistle', 'again', ';', 'the
n', 'the', 'puppy', 'began', 'a', 'series', 'of', 'short', 'charges', 'at', '
the', 'stick', ',', 'running', 'a', 'very', 'little', 'way', 'forwards', 'eac
h', 'time', 'and', 'a', 'long', 'way', 'back', ',', 'and', 'barking', 'hoarse
ly', 'all', 'the', 'while', ',', 'till', 'at', 'last', 'it', 'sat', 'down', '
a', 'good', 'way', 'off', ',', 'panting', ',', 'with', 'its', 'tongue', 'hang
ing', 'out', 'of', 'its', 'mouth', ',', 'and', 'its', 'great', 'eyes', 'half'
, 'shut', '.']]
```

# Frequency Distributions

Figure: Frequency Distribution for Alice Corpus

```
>>> fd = FreqDist(alice)
>>> print(fd)
<FreqDist with 3016 samples and 34110 outcomes>
>>> fd.most_common(10)
[(',', 1993), ("'", 1731), ('the', 1527), ('and', 802), ('.', 764), ('to', 72
5), ('a', 615), ('I', 543), ('it', 527), ('she', 509)]
>>>
```



Figure: Word Selection Alice Corpus

```
>>> V = set(alice)
>>> long_words=[w for w in V if len(w) > 10]
>>> sorted(long_words)
['Antipathies', 'Caterpillar', 'Distraction', 'Multiplication', '
, 'Shakespeare', 'UNimportant', 'Uglification', 'accidentally', '
ly', 'alternately', 'beautifully', 'caterpillar', 'circumstances'
le', 'comfortably', 'complaining', 'considering', 'consultation',
us', 'contemptuously', 'contradicted', 'conversation', 'conversat
erately', 'difficulties', 'disappeared', 'disappointment', 'encou
clamation', 'executioner', 'explanation', 'explanations', 'extrao
ortunately', 'frontispiece', 'handwriting', 'hippopotamus', 'imme
mpatiently', 'impertinent', 'incessantly', 'indignantly', 'inquis
nteresting', 'interrupted', 'interrupting', 'neighbouring', 'neve
occasionally', 'opportunity', 'processions', 'quarrelling', 'refr
remembering', 'respectable', 'straightened', 'straightening', 'th
'thunderstorm', 'triumphantly', 'uncomfortable', 'uncomfortably',
e', 'unimportant', 'unwillingly']
```

# Collocations and N-grams

Figure: Bigrams from Alice Corpus

```
>>> from nltk.collocations import *
>>> bigrams = nltk.collocations.BigramAssocMeasures()
>>> trigrams = nltk.collocations.TrigramAssocMeasures()
>>> finder = BigramCollocationFinder.from_words(alice)
>>> finder.nbest(bigrams.pmi,10)
[('"\'', 'TIS'), ('1865', ']'), ('BE', 'TRUE'), ('BUSY', 'BEE'), ('Carroll',
'1865'), ('Edgar', 'Atheling'), ('FUL', 'SOUP'), ('HOW', 'DOTH'), ('ITS', 'WA
ISTCOAT'), ('Latin', 'Grammar')]
```



Figure: Filtering for frequent collocations

```
>>> finder.apply_freq_filter(3)
>>> finder.nbest(bigrams.pmi,10)
[('Mary', 'Ann'), ('yer', 'honour'), ('Lobster', 'Quadrille'), ('young', 'lad
y'), ('ootiful', 'Soo'), ('white', 'kid'), ('golden', 'key'), ('fast', 'aslee
p'), ('kid', 'gloves'), ('play', 'croquet')]
```

Thank you for your attention!