

ALTA Summer School

Practical NLP using Python

Trevor Cohn and Steven Bird

December 2003

Parsing

1. Chunk Parsing

- motivation
- regular expression chunking and chunking
- NLTK interface

2. Full Parsing

- Shift-Reduce Parser
- Recursive Descent Parser
- Chart Parsing
- NLTK demonstration

Motivations for Parsing

- Why parse sentences in the first place?
 - Parsing is usually an intermediate stage
 - Builds structures that are used by later stages of processing
 - Full parsing is a *sufficient* but not *necessary* intermediate stage for many NLP tasks.
 - Parsing often provides more information than we need.
- Chunk parsing:
 - *Goal: assign a partial structure to a sentence.*
 - Simpler solution space (non-recursive)
 - Local context
 - Less dependence on semantic context

Chunk Parsing

Goal: divide a sentence into a sequence of chunks.

- Chunks are non-overlapping regions of a text
[I] saw [a tall man] in [the park].
- Chunks are non-recursive
 - a chunk can not contain other chunks
- Chunks are non-exhaustive
 - not all words are included in chunks
- Only require finite-state power
 - regular expressions
 - not context-free grammars

Regular Expression Chunking

- Define a regular expression that matches the sequences of tags in a chunk
 - A simple noun phrase chunk regexp:


```
<DT>? <JJ>* <NN.??>
```
- Chunk all matching subsequences:


```
the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
```

```
[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]
```
- If matching subsequences overlap, the first one gets priority
- Regular expressions can be cascaded

Chinking

- A *chink* is a subsequence of the text that is not a chunk.
- Define a regular expression that matches the sequences of tags in a chink
 - A simple chink regexp for finding NP chunks:


```
(<VB.??>|<IN>)+
```
- Chunk anything that is *not* a matching subsequence:


```
the/DT little/JJ cat/NN sat/VBD on/IN the/DT mat/NN
```

```
[the/DT little/JJ cat/NN] sat/VBD on/IN [the/DT mat/NN]
```

Chunk

Chink

Chunk

Chunking in NLTK

```
>>> from nltk.parser.chunk import *
>>> text = "[ John/NNP ] saw/VBD [the/DT cat/NN] [the/DT dog/NN]
liked/VBD ./."

>>> r1 = ChunkRule(r'<DT>?<JJ>*<NN.*>', 'Chunk NPs')
>>> cp = REChunkParser([r1], chunk_node='NP', top_node='S', trace=1)
>>> demo_eval(cp, text)
Input:
      <NNP> <VBD> <DT> <NN> <DT> <NN> <VBD> <.>
Chunk NPs:
      {<NNP>} <VBD> {<DT> <NN>}{<DT> <NN>} <VBD> <.>

/=====\
Scoring REChunkParser with 1 rules:
  Chunk NPs <ChunkRule: '<DT>?<JJ>*<NN.*>'>
-----
Precision: 100.0%   Recall: 100.0%   F-Measure: 100.0%
\=====/

>>> demo()          # view a cascade of chunk rules (tutorial)
```

Full Parsing: Simple Parsers

```
>>> from nltk.draw.srparser import *
>>> demo()

>>> from nltk.draw.rdpaser import *
>>> demo()
```

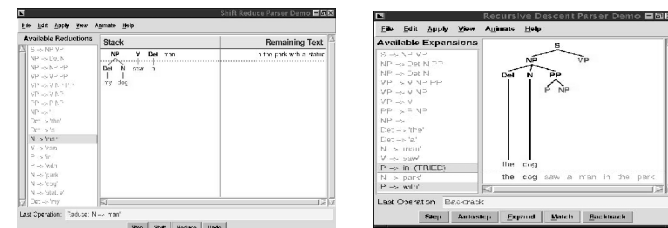
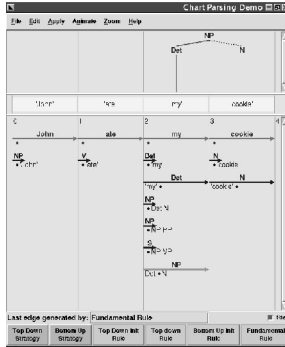


Chart Parsing

```
>>> from nltk.draw.chart import *
>>> demo ()
```



Lexical Semantics and Word Sense Disambiguation

1. Word meaning
2. Lexical Resources in NLTK:
 1. WordNet
 2. Roget's thesaurus
3. Word sense disambiguation
 1. Feature detection
 2. Naïve Bayes
 3. Decision list
 4. Decision trees
 5. Dictionary based sense tagging

Problem: Polysemy and Homonymy

- Polysemy:
 - e.g. *plain* – clear, undecorated, unattractive, level area of land
- Homonymy:
 - Homograph: different words with same orthography
 - e.g. *dove* – dive into water, white bird
 - e.g. *deal* – distribute cards (verb), an agreement (noun)
 - Homophone: different words with same sound
 - e.g. *see, sea*
 - e.g. French: *vert, verre, vers, ver* (*green, glass, towards, worm*)

Problem: Colour Terms

purple	blue	green	yellow	orange	red
--------	------	-------	--------	--------	-----

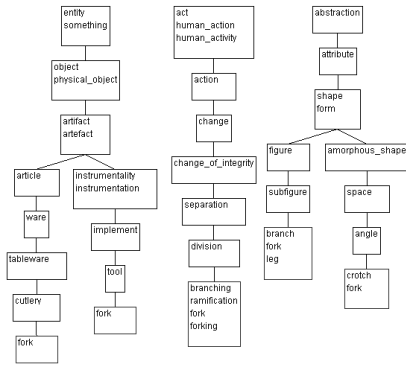
cipswuka	citema	cicena	cipswuka	<i>Shona</i>
----------	--------	--------	----------	--------------

hui	ziza	<i>Bassa</i>
-----	------	--------------

- Similar physical stimuli are categorised differently
- Important distinctions:
 - lexical colour terms
 - indirect (e.g. purple, orange, lime, vermillion (HgS), ...)
 - perception
- Colour term hierarchy (Berlin & Kay, 1969)
 - black, white, red, (green, yellow), blue, brown, (purple, pink, orange, grey)

Hypernyms and Hyponyms

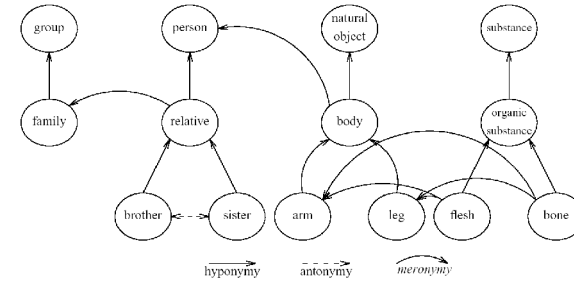
- Hypernym/Hyponym = generic/specific
- e.g. fork is a kind of cutlery
- “fork” is a hyponym of “cutlery”
- “cutlery” is a hypernym of “fork”
- Induces forest structure on our set of words
- Also gives a measure of semantic distance



2.13

Holonyms and Meronyms

- Holonym/Meronym (whole/part): 3 subtypes:
 1. Part: bone *is part of* arm
 2. Member: arm *is member of* body
 3. Substance: bone *is substance of* horn



2.14

Lexical Resources: Thesaurus

```
>>> from nltk.corpus import roget
>>> print roget.read('456. [Absence of curiosity.] Incuriosity.')
```

N. incuriosity,
incuriousness &c. adj.; insouciance &c. 866; indifference, lack of
interest, disinterest.
boredom, ennui (weariness) 841; satiety &c. 639; foreknowledge
(foresight) 510; .
V. be incurious &c. adj.; have no curiosity &c. 455; take no
interest
in &c. 823; mind one's own business.
Adj. incurious, uninquisitive, indifferent; impassive &c. 823;
uninterested, detached, aloof.

2.15

WordNet – Synsets

- A lexical database
 - Massive network of lexical relationships;
 - English: Nouns: 120,000; Verbs: 22,000; Adjectives: 30,000; Adverbs: 6,000
- Words are ambiguous
 - e.g. “fork” in earlier slide
 - the different senses participate in different lexical relations
- Nodes in Wordnet represent “synonym sets”, or *synsets*.
 - e.g. {chump, fish, fool, gull, mark, patsy, fall guy, sucker, schlemiel, shlemiel, soft touch, mug}
- Applications:
 - Overcome limitations in other data (e.g. PP attachment)
 - Implement selectional restrictions (use WordNet categories on grammar productions)

2.16

The pywordnet interface

```
>>> from nltk_contrib.pywordnet import *
>>> fork_n = N['fork']
fork(n.)
>>> fork_v = V['fork']
fork(v.)
>>> n_senses = fork_n.senses()
('fork' in {noun: fork}, 'fork' in {noun: branching,
ramification, fork, forking}, 'fork' in {noun: branch, fork,
leg, ramification}, 'fork' in {noun: fork}, 'fork' in {noun:
crotch, fork})
>>> v_senses = fork_v.senses()
('fork' in {verb: pitchfork, fork}, 'fork' in {verb: fork},
'fork' in {verb: branch, ramify, fork, furcate, separate},
'fork' in {verb: fork})
>>> cutlery_synset = n_senses[0].synset
'fork' in {noun: fork}
>>> cutlery_synset.gloss
'cutlery used for serving and eating food'
```

Pywordnet (cont)

```
>>> [s.form for s in n_senses[1].synset.senses()]
['branching', 'ramification', 'fork', 'forking']
>>> cutlery_synset.pointers()
(hypernym -> {noun: cutlery, eating utensil}, hyponym
-> {noun: carving fork}, part holonym -> {noun:
prong}, hyponym -> {noun: salad fork}, hyponym ->
{noun: tablefork}, part holonym -> {noun: tine},
hyponym -> {noun: toasting fork})
>>> tine_synset = cutlery_synset.pointers()[5].target
()
{noun: tine}
>>> tine_synset.definition()
'prong on a fork or pitchfork'
```

Word Sense Disambiguation

- The task
 - *to disambiguate two or more semantically distinct forms which have been conflated into the same representation in some medium* (Yarowsky)
 - "The US puts a new face on the chase for Saddam" (The Age, Sunday 3/8/03)
 - US/n
 - First person plural inclusive pronoun
 - Abbrev. The United States of America
 - Put/v.t.
 - Transfer to a specified place
 - Express in words
 - Propel from hand with pushing motion
 - face/n
 - Front of head
 - Expression, grimace
 - Aspect (on the face of it...)
 - ...

WSD (cont)

- An avalanche of competing interpretations
 - 5,760 different sense combinations in example
 - as sentences grow, exponential growth of interpretations
 - flipside: smaller contexts have fewer clues for disambiguation
- Useful for:
 - semantic analysis (natural language understanding)
 - machine translation
 - information retrieval
 - homograph resolution in text-to-speech
 - sentence boundary detection
 - restoring accents and capitals
 - context sensitive spelling correction
 - ...

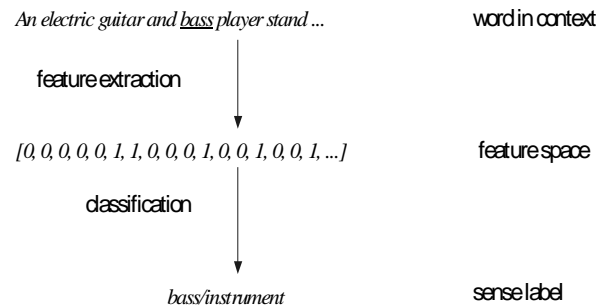
Methods for Robust WSD

- **Statistical and machine learning methods largely in vogue**
 - Naive Bayes
 - Decision lists
 - Decision trees
 - require training data (supervised learners)
- **Dictionary-based methods**
- **Unsupervised clustering techniques (not covered)**

Training Data for WSD: SENSEVAL

```
>>> import nltk.corpus
>>> tokens = nltk.corpus.senseval.tokenize('hard.pos')
>>> split = {}
>>> for token in tokens:
...     label = token.type().label()
...     split.setdefault(label, []).append(token)
...
>>> zip(split.keys(), map(len, split.values()))
[('HARD1',), 3455], (('HARD2',), 502), (('HARD3',), 376)]
>>> split[('HARD1',)][0]
['''/'@[1w], 'he'/'PRP'@[2w], 'may'/'MD'@[3w], 'lose'/'VB'@[4w], 'all'/'DT'@[5w], 'popular'/'JJ'@[6w], 'support'/'NN'@[7w], '/', '@[8w], 'but'/'CC'@[9w], 'someone'/'NN'@[10w], 'has'/'VBZ'@[11w], 'to'/'TO'@[12w], 'kill'/'VB'@[13w], 'him'/'PRP'@[14w], 'to'/'TO'@[15w], 'defeat'/'VB'@[16w], 'him'/'PRP'@[17w], 'and'/'CC'@[18w], 'that'/'DT'@[19w], "s"/'VBZ'@[20w], 'hard'/'JJ'@[21w], 'to'/'TO'@[22w], 'do'/'VB'@[23w], '.'/'.'@[24w], ""/'"@[25w]']/'HARD1',)[head=20][0]
```

Features and classification



Inputs: Feature Vectors

- **Encode the context**
 - what features have good predictive value?
 - choose the best sense in a given context
 - two encoding methods: collocational, cooccurrence
- **Collocational vectors**
 - pos-tag and stem the input
 - partial parsing to determine grammatical roles
 - select a window
 - e.g. An electric guitar and bass player stand ...
[guitar, NN, and, CC, player, NN, stand, VB]

Inputs: Feature Vectors (cont)

- Cooccurrence vectors
 - select a small number of frequently used content words
 - treat the presence of a word root in the context as a feature
 - feature value is the frequency of the word in the context
 - e.g. frequent content words in the vicinity of *bass* in WSJ are: *fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band.*
 - [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0]
- NB – the two methods can be combined

Feature detectors

- A feature detector is a function mapping from the word in context to 'feature space'

- eg. does the text contain the word 'hello'?

```
>>> from nltk_contrib.unimelb.tacohn.classifier.feature import *
>>> fd = FunctionFeatureDetector(lambda text: 'hello' in text)
>>> fd.detect(['hello', 'world'])
True
>>> fd.detect(['goodbye', 'world'])
False
```

- eg. what is the length of the first word in the text?

```
>>> fd = FunctionFeatureDetector(lambda text: len(text[0]))
>>> fd.detect(['hello', 'world'])
4
>>> fd.detect(['goodbye', 'world'])
7
```

Feature detector lists

- FDLISTs allow different features to be detected simultaneously and efficiently

- eg. what is the length of the first word in the text, within the range [0, 10)?
(this produces a binary valued feature)

```
>>> fd = TextFunctionFDList(lambda text: len(text[0]), range(10))
>>> fv_list = fd.detect(['goodbye', 'world'])
<FeatureValueList with 10 features>
```

- this creates a feature value list, wrapping a (sparse) sequence of (index, value) pairs

```
>>> fv_list.assignments()
((7, 1),)
>>> fv_list[0]
0
>>> fv_list[7]
1
>>> fd.detect(['hello', 'world']).assignments()
((5, 1),)
```

Bag of words (coocurrence)

- Treat the context as an unordered 'bag' of words

- either as a set of binary flags (does each word appear in the text)?

```
>>> from nltk_contrib.unimelb.tacohn.classifier.feature import *
>>> text = 'An electric guitarp player and bass player'.split()
>>> lexicon = 'guitar player fish'.split()
>>> fd_list = BagOfWordsFDList(lexicon)
>>> fv_list = fd_list.detect(text)
<FeatureValueList with 3 features>
>>> [fv_list[i] for i in range(len(fv_list))]
[1, 1, 0]
```

- or as an integer count (how many times does each word appear)?

```
>>> fd_list MultiBagOfWordsFDList(lexicon)
<FeatureDetectorList with 3 features>
>>> fv_list = fd_list.detect(text)
>>> [fv_list[i] for i in range(len(fv_list))]
[1, 2, 0]
>>> fd_list.describe(1)
'player'
```

Filters

- Text is usually not a simple list of strings
 - often a sequence of (tagged) tokens, a parse tree ...
 - need to extract the relevant information
 - eg. to find just the word forms from a list of tagged tokens:

```
>>> from nltk.corpus import brown
>>> text = brown.tokenize('cp04')[1:14]
['`./''@'[0w], 'He'/'PPS@[1w], 'must'/'MD@[2w], 'have'/'HV@[3w],
'forgiven'/'VBN@[4w], 'me'/'PP0@[5w], ''''/'''@[6w], ', '/'@[7w],
'Henrietta'/'NP@[8w], 'murmured'/'VBD@[9w], 'to'/'IN@[10w],
'the'/'AT@[11w], 'room'/'NN@[12w], './'/'@[13w]]
>>> base_filter = ArrayFunctionFilter(lambda tk: tk.type().base())
>>> base_filter(text)
['', 'He', 'must', 'have', 'forgiven', 'me', '', '', 'Henrietta',
'murmured', 'to', 'the', 'room', '.']
>>> bag_of_words = BagOfWordsFDList('forgiven you room'.split())
>>> bag_of_words.detect(base_filter(text)).assignments()
[(0, 1), (2, 1)]
>>> filtered_fd_list = FilteredFDList(base_filter, bag_of_words)
>>> filtered_fd_list.detect(text).assignments()
[(0, 1), (2, 1)]
```

ALTA
Summer
School

2.29

Trevor Cohn
& Steven Bird
12/2003

Combining filters

- Filters can be chained
 - eg. first select a window of the text, find the base and decapitalise
 - CompositeFilter(x, y, z, ...): applies filter x, then passes result to y, followed by z, etc

```
>>> window_filter = ArrayIndexFilter(slice(2, 5))
>>> lower_filter = ArrayFunctionFilter(lambda form: form.lower())
>>> filter = CompositeFilter(window_filter, base_filter,
lower_filter)
>>> filter(text)
['must', 'have', 'forgiven']
>>> filtered_fd_list = FilteredFDList(filter, bag_of_words)
>>> filtered_fd_list.detect(text).assignments()
[(0, 1)]
```

ALTA
Summer
School

2.30

Trevor Cohn
& Steven Bird
12/2003

Combining FDLISTs

- Feature detector lists may also be combined
 - Use the '+' operator
 - The resulting FDLIST will detect the union of features of the operands
 - eg. combining 'bag of tags' with 'bag of words':

```
>>> forms_fd_list = FilteredFDList(
    ArrayFunctionFilter(lambda tk: tk.type().base()),
    BagOfWordsFDList('forgiven you room'.split()))
<FeatureDetectorList with 3 features>
>>> tags_fd_list = FilteredFDList(
    ArrayFunctionFilter(lambda tk: tk.type().tag()),
    BagOfWordsFDList('NN NP'.split()))
<FeatureDetectorList with 2 features>
>>> combined_fd_list = forms_fd_list + tags_fd_list
<FeatureDetectorList with 5 features>
>>> combined_fd_list.detect(text).assignments()
[(0, 1), (2, 1), (3, 1), (4, 1)]
>>> combined_fd_list.describe(0)
'filtered forgiven'
>>> combined_fd_list.describe(3)
'filtered NN'
```

ALTA
Summer
School

2.31

Trevor Cohn
& Steven Bird
12/2003

Classifying

- Instances mapped into feature space
 - train (induce) a classifier on a set of sense-labeled instances
 - use this classifier to label novel instances

```
>>> text = 'He likes to play bass because he doesn't have to
solo.'.split()
>>> token = Token(text, Location(2031, 2042, 'cp27'))
>>> my_classifier.classify(token)
['He', 'likes', 'to', 'play', 'bass', 'because', 'he',
'doesn't', 'have', 'to', 'solo', '.']/'instrument@[
2031cp27:2042cp27]
>>> for label, prob in \
    my_classifier.distribution_dictionary(token).items():
    print '%10s: %.2f' % (label, prob)
instrument: 0.62
fish: 0.10
singer: 0.28
```

ALTA
Summer
School

2.32

Trevor Cohn
& Steven Bird
12/2003

Training data

bass/fish

... and produce fillets of smoked bass or sturgeon, sandwiches ...

bass/music

... for a female to sing bass, baritone, tenor ...

But he plays bass with Chief Crazy Horse ...

He likes to play bass because he doesn't have ...

... evading the heaviness of figured bass, the skill in ...

Special thanks to the double bass playing of Johann Krump ...

It calls for a double bass, an instrument generally ...

Naïve Bayes

- Find the frequency of occurrence of each word with each sense & the probability of each sense (fish: 1/7, music: 6/7)

	fish	music
and	1.00	0.00
sturgeon	1.00	0.00
...		
or	1.00	0.00
of	0.33	0.67
bass	0.14	0.86
tenor	0.00	1.00
...		
figure	0.00	1.00
sing	0.00	1.00

- Classify text: 'singing, double bass'

$$p(\text{fish} | \text{text}) = p(\text{fish}) * p(\text{sing} | \text{fish}) * p(\text{,} | \text{fish}) * p(\text{double} | \text{fish}) * p(\text{bass} | \text{fish})$$

$$= 1/7 * 0 * 0.2 * 0 * 0 * 1/7 = 0$$

$$p(\text{music} | \text{text}) = p(\text{music}) * p(\text{sing} | \text{music}) * p(\text{,} | \text{music}) * \dots$$

$$= 6/7 * 1 * 0.8 * 1 * 6/7 = 0.59$$

2.34

highest; class = music

Naive Bayes (cont)

- Choose the most probable sense given the input vector

$$s' = \arg \max_{s_k} P(s_k | c)$$

- It is difficult to collect this data directly, use Baye's rule:

$$\arg \max_{s_k} \frac{P(c | s_k) P(s_k)}{P(c)}$$

- data too sparse, make the independence assumption:

$$\approx \arg \max_{s_k} P(s_k) \prod_{v_j \in c} P(v_j | s_k)$$

Naive Bayes: usage

- Training NB classifier using the interest corpus

```
>>> from nltk.corpus import senseval
>>> from nltk_contrib.unimelb.tacohn.classifier.naivebayes import *
# tokenise the corpus
>>> labeled_tokens = senseval.tokenize('interest.pos')
# load a precomputed set of relevant words
>>> words = open('words.txt').read().split()
# use a filter to take only the word forms
>>> fd_list = FilteredFDList(ArrayFunctionFilter(
    lambda tk: tk.type().base(), BagOfWordsFDList
    (words))
# train the classifier on a sub-set of the data
>>> trainer = NBClassifierTrainer(fd_list)
>>> classifier = trainer.train(labeled_tokens[100:])
<NBClassifier: 6 labels, 7440 features>
# test with a (novel) instance
>>> classifier.classify(Token(labeled_tokens[0].type().text())
['yields''NNS@[1w], 'on''IN@[2w], 'money-market''JJ@[3w], ...
'rates''NNS@[20w], '.'''@[21w]]/('interest_2',)@[?]
```

Naive Bayes: pros and cons

- good for disambiguation contexts involving:
 - lexical identity
 - long-distances
- bad for disambiguation contexts involving:
 - sequences
 - combining sources of evidence which aren't independent
 - e.g. parts-of-speech, lemmas, word classes, words

Decision Lists

- Generate a set of binary criteria:
 - if *condition* then *sense*
- e.g. for bass (fish) vs bass (guitar)
 - IF *fish* within window? THEN bass¹
 - ELSE IF *striped bass*? THEN bass¹
 - ELSE IF *guitar* within window? THEN bass²
 - ELSE IF *bass player*? ⇒ THEN bass²
 - ELSE bass¹
- $abs(\log(P(s_1 | v_i = x) / P(s_2 | v_i = x)))$
- Apply this to each sense and criterion
- Order the criteria accordingly

Decision list: usage

- Training on interest corpus

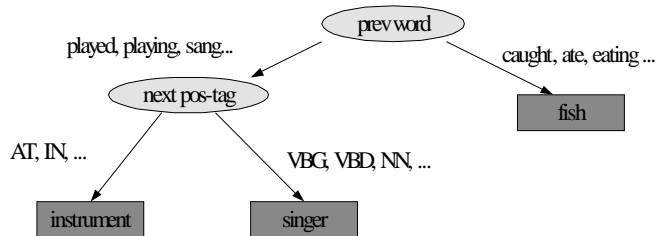
```
>>> from nltk.corpus import senseval
>>> from nltk_contrib.unimelb.tacohn.classifier.decisionlist import *
# tokenize the corpus
>>> labeled_tokens = senseval.tokenize('interest.pos')
# a precomputed set of relevant words
>>> words = open('words.txt').split()
# use a filter to take only the word forms
>>> fd_list = FilteredFDList(ArrayFunctionFilter(
    lambda tk: tk.type().base(), BagOfWordsFDList
    (words))
# train the classifier on a sub-set of the data
>>> trainer = DecisionListClassifierTrainer(fd_list)
>>> classifier = trainer.train(labeled_tokens[100:])
<DecisionListClassifier with 7440 options>
# test with a (novel) instance
>>> classifier.classify(Token(labeled_tokens[0].type().text()))
['yields'/'NNS'@[1w], 'on'/'IN'@[2w], 'money-market'/'JJ'@[3w], ...
'rates'/'NNS'@[20w], '.'/'.'@[21w]]/('interest_6',)@[?]
```

Decision Lists: Pros and Cons

- This method only uses the single most reliable piece of evidence
 - cf Naive Bayes – a weighted combination of all the evidence
 - con: with mixed evidence, behavior is unpredictable
 - pro: can use multiple, non-independent information types in the decision procedure

Decision Trees

- Generate an hierarchical tree of decisions:
 - nodes are conditions
 - leaves are sense labels
 - equivalent to the disjunction of the conjunction of all paths from root to leaves



Decision trees: usage

- Splitting criterion
 - eg. chose feature (or combination of features) to give most informative split
 - flip-flop algorithm partitions table of multi-value features

```
>>> from nltk_contrib.unimelb.tacohn.classifier.decisiontree import *
# train the classifier on a sub-set of the data
>>> trainer = DecisionTreeClassifierTrainer(fd_list)
>>> classifier = trainer.train(labeled_tokens[100:])
<DecisionTreeClassifier>
# test with a (novel) instance
>>> classifier.classify(Token(labeled_tokens[0].type().text())
['yields'/'NNS'@[1w], 'on'/'IN'@[2w], 'money-market'/'JJ'@[3w], ...
'rates'/'NNS'@[20w], '.'/'.'@[21w]/'(interest_6',)@[?]
```

Dictionary-Based Disambiguation

- Dictionaries
 - a common language resource
 - definitions provide a bag of words for each sense
 - look for these words in the context to help us disambiguate
- Example
 - pine: (1) kinds of evergreen tree with needle-shaped leaves
(2) waste away through sorrow or illness
 - cone: (1) solid body which narrows to a point
(2) something of this shape whether solid or hollow
(3) fruit of certain evergreen trees
 - pine cone: choose senses which maximally overlap

Dictionary based tagger

- Based on Lesk (1984)

```
>>> from nltk.corpus import *
>>> from nltk_contrib.unimelb.tacohn.tagger.dictionary import *
>>> from nltk_contrib.pywordnet.stemmer import WordNetStemmer
# load stoplist
>>> stoplist = [tk.type() for tk in stopwords.tokenize('english')]
# load a bit of the brown corpus
>>> items = brown.items('humor')
>>> tagged_tokens = brown.tokenize(items[0])
# create the WordNet dictionary source
>>> dictionary = WordNetDictionary(stoplist, None, brown_nouns,
                                brown_verbs, brown_adjs, brown_adv)
# window of +- 5 words
>>> tagger = LeskWordSenseTagger(5, dictionary, WordNetStemmer())
>>> tagger.tag(tagged_tokens[:200])
['It'/'PPS'@[0w], 'was'/'BEDZ'@[1w], ... 'why'/'adverb/109540'@[26w],
'not'/'adverb/23221'@[27w], ',','@[28w], 'after'/'IN'@[29w],
'seven'/'CD'@[30w], ... 'ought'/'MD'@[63w], 'to'/'TO'@[64w],
'be'/'verb/2047807'@[65w], 'in'/'IN'@[66w], 'movies'/'noun/5535643'@[67w], ... ]
```

Dictionary based tagger (cont)

- **Can lookup definitions**

```
>>> dictionary.definition('noun/5535643')
'noun: movie, film, picture, moving picture, motion picture,
picture show, pic, flick -- (a form of entertainment that enacts
a story by a sequence of images giving the illusion of
continuous movement; "they went to a movie every Saturday
night"; "the film was shot on location")'
```

- **Can use alternate dictionary (Roget's thesaurus)**

```
>>> from nltk.stemmer.porter import PorterStemmer
>>> stemmer = PorterStemmer()
>>> dictionary = RogetDictionary(stoplist, stemmer)
```

Issues

- **Knowledge acquisition bottleneck**
 - Supervised methods need marked up data
- **What is a word sense?**
 - Dictionaries as sense inventories
 - Bi-lingual text as sense inventories
 - Dependent on contextual usage
- **Performance measurement**
 - Non-uniform confusability

NLTK

- **There's much more to NLTK:**

- tokenization
- probability
- context free grammars and parsing (chunking, symbolic, probabilistic)
- finite state automata
- part of speech taggers
- text and word sense classifiers
- visualisations of various NLP processes
- text corpora
- regular user contributions