Euromasters summer school 2005

Introduction to NLTK *Part II*

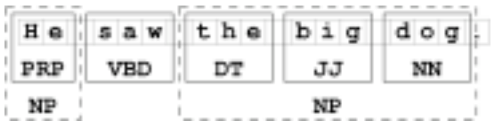
Trevor Cohn
July 12, 2005

Outline

- Syntactic parsing
  - shallow parsing (chunking)
  - CFG parsing
    - shift reduce parsing
    - chart parsing: top down, bottom up, earley
- Classification
  - at word level or at document level

Identification and Classification

- Segmentation and Labelling
  - tokenization + tagging: sequences of characters
  - chunking: sequences of words
- similarities between tokenization/tagging and chunking:
  - omitted material, finite-state, application-specific



Motivations

- Locating information
  - e.g. text retrieval
  - index a document collection on its noun phrases
    - e.g. "Rangers Football Club", "Edinburgh University"
- Ignoring information
  - e.g. syntactic analysis
  - throw away noun phrases to study higher-level patterns
    - e.g. phrases involving "gave" in Penn treebank:
    - gave NP; gave up NP in NP; gave NP up; gave NP help; gave NP to NP

Comparison with Parsing

- Full parsing: build a *complete parse tree*
  - Low Accuracy, slow, domain specific
  - Unnecessary details for many super-tasks
- Chunk parsing: just model *chunks* of the parse
  - Smaller solution space
  - Relevant context is small and local
  - Chunks are non-recursive
  - Chunk parsing can be implemented with a finite state machine
    - Fast
    - Low memory requirements
  - Chunk parsing can be applied to very large text sources (e.g., the web)

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

Chunk Parsing Examples

- **Noun-phrase chunking:**

  [I] saw [a tall man] in [the park].

- **Verb-phrase chunking:**

  The man who [was in the park] [saw me].

- **Prosodic chunking:**

  [I sáw] [a tall mán] [in the párk].

Chunks and Constituency

Constituents: [a tall man in [the park]].

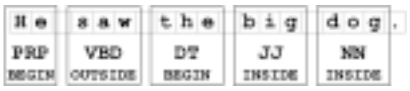
Chunks: [a tall man] in [the park].

- ***Chunks are not constituents***
  - ***Constituents are recursive***
- ***Chunks are typically subsequences of constituents***
  - ***Chunks do not cross major constituent boundaries***

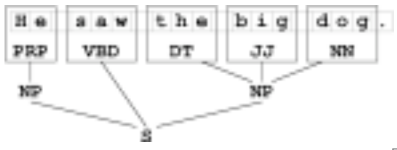
1. [$_{NP}$ [$_{NP}$ G.K. Chesterton], [$_{NP}$ [$_{NP}$ author] of [$_{NP}$ [$_{NP}$ The Man] who was [$_{NP}$ Thursday]]]]
2. [$_{NP}$ G.K. Chesterton], [$_{NP}$ author] of [$_{NP}$ The Man] who was [$_{NP}$ Thursday]

Representation

- **BIO (or IOB)**



- **Trees**



Reading from BIO-tagged data

```
>>> from nltk.tokenreader.conll import *
>>> text = '''he PRP B-NP
accepted VBD B-VP
the DT B-NP
position NN I-NP
of IN B-PP
vice NN B-NP
chairman NN I-NP
of IN B-PP
Carlyle NNP B-NP
Group NNP I-NP
, , O
'''
>>> reader = ConllTokenReader(chunk_types=['NP'])
>>> text_tok = reader.read_token(text)
>>> print text_tok['SENTS'][0]['TREE']
(S: (NP: <he/PRP>)
    <accepted/VBD>
    (NP: <the/DT> <position/NN>)
    <of/IN> ...
```

Data is from NLTK "chunking" corpus

Chunk Parsing Techniques

- **Chunk parsers usually ignore lexical content**
  - **Only need to look at part-of-speech tags**
- **Possible steps in chunk parsing**
  - **Chunking, unchunking**
  - **Chinking**
  - **Merging, splitting**
- **Evaluation**
  - **Baseline**

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**
- ***(Unchunking is the opposite of chunking)***

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*

Merging

- Combine adjacent chunks into a single chunk
- *Define a regular expression that matches the sequences of tags on both sides of the point to be merged*
  - Merge a chunk ending in JJ with a chunk starting with NN

left: <JJ> right: <NN>

[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

- *(Splitting is the opposite of merging)*

Evaluating Performance

- Basic measures:

	Target	¬Target
Selected	True positive	False positive
¬Selected	False negative	True negative

- Precision $\frac{tp}{(tp+fp)}$
  - What proportion of selected items are correct?
- Recall = $\frac{tp}{(tp+fn)}$
  - What proportion of target items are selected?
- See section 7 of chunking tutorial, ChunkScore class

Cascaded Chunking

```
# find NP chunks
>>> rule = ChunkRule(r'<DT>?<JJ>*<NN.*>', 'Chunk NPs')
>>> parser = RegexpChunkParser([rule], chunk_node='NP',
...              top_node='S', TREE='NP-CHUNKS', SUBTOKENS='WORDS')
>>> text_tok = Token(WORDS=s.leaves())
>>> parser.parse(text_tok)

# find VP chunks
>>> rule = ChunkRule(r'<VB.*><.*>*', 'Chunk VPs')
>>> parser = RegexpChunkParser([rule], chunk_node='VP',
...              top_node='S', SUBTOKENS='NP-CHUNKS')
>>> parser.parse( text_tok )
>>> print text_tok['TREE']
(S:
  (NP: <the/DT> <little/JJ> <cat/NN>)
  (VP: <sat/VBD> <on/IN> (NP: <the/DT> <mat/NN>)))
```

Grammars and Parsing

- Some Applications
  - Grammar checking
  - Machine translation
  - Dialogue systems
  - Summarization
- Sources of complexity
  - Size of search space
  - No independent source of knowledge about the underlying structures
  - Lexical and structural ambiguity

Syntax

the part of a grammar that represents a speaker's knowledge of the structure of phrases and sentences

Why word order is significant:

- *may have no effect on meaning*
  - *Jack Horner stuck in his thumb*
    Jack Horner stuck his thumb in
- *may change meaning*
  - *Salome danced for Herod*
    Herod danced for Salome
- *may render a sentence ungrammatical*
  - **for danced Herod Salome*

Syntactic Constituency

1. Ability to stand alone: *exclamations and answers*
   - *What do many executives do?*
     Eat at really fancy restaurants
   - *Do fancy restaurants do much business?*
     **Well, executives eat at*
2. Substitution by a pro-form: *pronouns, pro-verbs (do, be, have), pro-adverbs (there, then), pro-adjective (such)*
   - *Many executives do*
3. Movement: *fronting or extraposing a fragment*
   - At really fancy restaurants, many executives eat
     *Fancy restaurants many executives eat at really

Constituency: Tree diagrams

Major Syntactic Constituents

- Noun Phrase (NP)
  - referring expressions
- Verb Phrase (VP)
  - predicating expressions
- Prepositional Phrase (PP)
  - direction, location, etc
- Adjectival Phrase (AdjP)
  - modified adjectives (e.g. "really fancy")
- Adverbial Phrase (AdvP)
- Complementizers (COMP)

Penn Treebank

```
(S (S-TPC-1
  (NP-SBJ (NP (NP A form) (PP of (NP asbestos)))
       (RRC (ADVP-TMP once)
           (VP used (NP *) (S-CLR (NP-SBJ *)
             (VP to (VP make
               (NP Kent cigarette filters)))))))
    (VP has (VP caused
       (NP (NP a high percentage)
          (PP of (NP cancer deaths))
       (PP-LOC among
          (NP (NP a group)
             (PP of (NP
               (NP workers)
               (RRC (VP exposed (NP *)
               (PP-CLR to (NP it))
               (ADVP-TMP (NP (QP more than 30) years) ago)))))))))), (NP-
  SBJ researchers) (VP reported (SBAR 0 (S *T*-1))).))
```

Phrase Structure Grammar

Grammaticality:

.doesn't depend on:

- having heard the sentence before
- the sentence being true *(I saw a unicorn yesterday)*
- the sentence being meaningful
  (colorless green ideas sleep furiously vs
  **furiously sleep ideas green colorless)*
- *learned rules of grammar*

.*a formal property that we can investigate and model*

Recursive Grammars

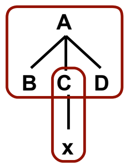
- *set of well formed English sentences is infinite*
  - *no a priori length limit*
  - *Sentence from A.A.Milne (next slide)*
- *a grammar is a finite-statement about well-formedness*
  - *it has to involve iteration or recursion*
- *examples of recursive rules*
  - *NP → NP PP (in a single rule)*
  - *NP → S, S → NP VP (recursive pair)*
- *therefore search is over a possibly infinite set*

Recursive Grammars (cont)

You can imagine Piglet's joy when at last the ship came in sight of him. In after-years he liked to think that he had been in Very Great Danger during the Terrible Flood, but the only danger he had really been in was the last half-hour of his imprisonment, when Owl, who had just flown up, sat on a branch of his tree to comfort him, and told him a very long story about an aunt who had once laid a seagull's egg by mistake, and the story went on and on, rather like this sentence, until Piglet who was listening out of his window without much hope, went to sleep quietly and naturally, slipping slowly out of the window towards the water until he was only hanging on by his toes, at which moment, luckily, a sudden loud squawk from Owl, which was really part of the story, being what his aunt said, woke the Piglet up and just gave him time to jerk himself back into safety and say, "How interesting, and did she?" when -- well, you can imagine his joy when at last he saw the good ship, Brain of Pooh (Captain, C. Robin; I st Mate, P. Bear) coming over the sea to rescue him...

A.A. Milne: *In which Piglet is Entirely Surrounded by Water*

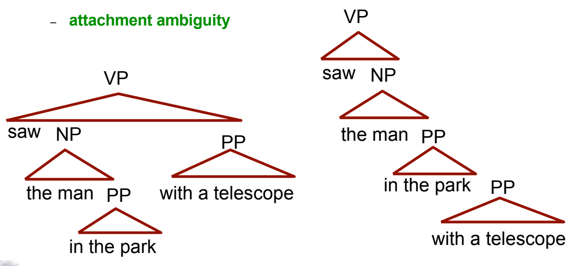
Trees from Local Trees



- *A tree is just a set of connected local trees*
- *Each local tree is licensed by a production*
- *Each production is included in the grammar*
- *The fringe of the tree is a given sentence*
- *Parsing = discovering the tree(s) for a given sentence*
- *A SEARCH PROBLEM*

Syntactic Ambiguity

- I saw the man in the park with a telescope
  - several "readings"
  - attachment ambiguity



Grammars

S -> NP, VP
NP -> Det, N
VP -> V, NP
VP -> V, NP, PP
NP -> Det, N, PP
PP -> P, NP

NP -> 'I'
N -> 'man'
Det -> 'the'
Det -> 'a'
V -> 'saw'
P -> 'in'
P -> 'with'
N -> 'park'
N -> 'dog'
N -> 'telescope'

Kinds of Parsing

- Top down, Bottom up
- Chart parsing
- Chunk parsing (earlier)

Top-Down Parsing (Recursive Descent Parsing)

parse(goal, sent):

- if goal and string are empty we're done, else
- is the first element of the goal the same as the first element in the string?
  - if so, strip off these first elements and continue processing
- otherwise, check if any of the rule LHSs match the first element of the goal
  - if so, replace this element with the RHS of the rule
  - do this for all rules
  - new continue with the new goal

- Demonstration

Bottom-Up Parsing

parse(sent):

- if sent is [S] then finish
- otherwise, for every rule, check if the RHS of the rule matches any substring of the sentence
- if it does, replace the substring the the LHS of the rule
- continue with this sentence

- Demonstration

Issues and Solutions

- top-down parsing:
  - wasted processing hypothesizing words and phrases (relevant lexical items are absent), repeated parsing of subtrees
  - infinite recursion on left-recursive rules (transforming the grammar)
- bottom-up parsing:
  - builds sequences of constituents that top-down parsing will never consider
- solutions:
  - BU to find categories of lexical items, then TD
  - left-corner parsing (bottom-up filtering)

Chart Parsing

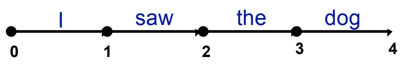
1. Problems with naive parsers
2. Tokens and charts
3. Productions, trees and charts
4. Chart Parsers
5. Adding edges to the chart
6. Rules and strategies
7. Demonstration

Issues and Solutions

- top-down parsing:
  - wasted processing hypothesizing words and phrases (relevant lexical items are absent), repeated parsing of subtrees
  - infinite recursion on left-recursive rules (transforming the grammar)
- bottom-up parsing:
  - builds sequences of constituents that top-down parsing will never consider
- solutions:
  - BU to find categories of lexical items, then TD
  - left-corner parsing (bottom-up filtering)
- More general, flexible solution: dynamic programming

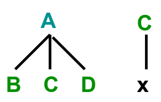
Tokens and Charts

- An input sentence can be stored in a chart
  - Sentence = list of tokens
  - Token = (type, location) -> *Edge*
- E.g. $[I_{0:1}, saw_{1:2}, the_{2:3}, dog_{3:4}]$
  - NLTK: ['I'@[0:1], 'saw'@[1:2], 'the'@[2:3], 'dog'@[3:4]]
  - Abbrev: ['I'@[0], 'saw'@[1], 'the'@[2], 'dog'@[3]]
- *Chart representation:*

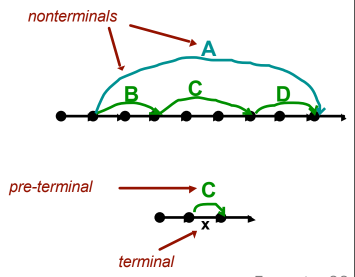


Productions, Trees & Charts

- *Productions:*
  - *$A \rightarrow BCD$, $C \rightarrow x$*
- Trees:

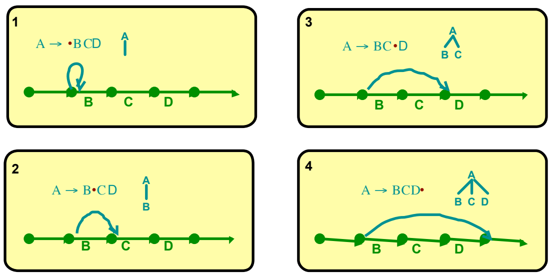


- Charts:



Edges and “Dotted” Productions

- Edges decorated with dotted production and tree



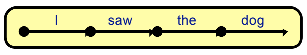
- Partial vs complete edges; zero-width edges

Charts and Chart Parsers

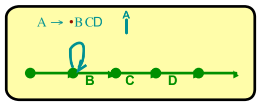
- Chart:
  - collection of edges
- Chart parser:
  - Consults three sources of information:
    - Grammar
    - Input sentence
    - Existing chart
  - Action:
    - Add more edges to the chart
    - Report any completed parse trees
- Three ways of adding edges to the chart...

Adding Edges to the Chart

1. Adding LeafEdges

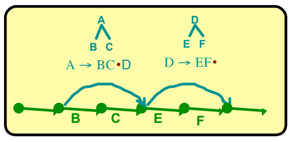


2. Adding self loops



Adding Edges to the Chart (cont)

3. Adding “fundamental rule” edges



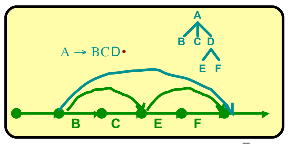


Chart Rules: Bottom-Up Rule

- Bottom-Up Rule:
  - For each complete edge C, set X = LHS of production
    For each grammar rule with X as first element on RHS
    Insert zero-width edge to left of C

```
Bottom Up Init Rule |[--] . . . . . . .| 'I'.
Bottom Up Init Rule |. [--] . . . . . .| 'saw'.
Bottom Up Init Rule |. . [--] . . . . .| 'the'.
Bottom Up Init Rule |. . . [--] . . . .| 'dog'.
Bottom Up Init Rule |. . . . [--] . . .| 'with'.
Bottom Up Init Rule |. . . . . [--] . .| 'my'.
Bottom Up Init Rule |. . . . . . [--]| 'cookie'.
Bottom Up Rule      |. . . . . . . > .| N -> * 'cookie'
Bottom Up Rule      |. . . . . . > . .| Det -> * 'my'
Bottom Up Rule      |. . . . . > . . .| P -> * 'with'
Bottom Up Rule      |. . . > . . . . .| N -> * 'dog'
Bottom Up Rule      |. . > . . . . . .| Det -> * 'the'
Bottom Up Rule      |. > . . . . . . .| V -> * 'saw'
Bottom Up Rule      |> . . . . . . . .| NP -> * 'I'
```

Chart Rules: Top-Down Rules

- Top down initialization:
  - For every production whose LHS is the base category:
    create the corresponding dotted rule
    put dot position at the start of RHS

```
Top Down Init Rule |> . . . . . . . .| S -> * NP VP
```

- Top down expand rule:
  - For each production and for each incomplete edge:
    if the expected constituent matches the production:
    insert zero-width edge with this production on right

```
Top Down Rule      |> . . . . . . . .| NP -> * 'I'
Top Down Rule      |> . . . . . . . .| NP -> * Det N
Top Down Rule      |> . . . . . . . .| NP -> * NP PP
Top Down Rule      |> . . . . . . . .| Det -> * 'the'
Top Down Rule      |> . . . . . . . .| Det -> * 'my'
```

Rules, Strategies, Demo

- **Fundamental rule:**
  - For each pair of edges e_1 and e_2:
    If e_1 is incomplete and its expected constituent is X:
    If e_2 is complete and its LHS is X:
    Add e_3 spanning both e_1 and e_2, with dot moved right
- **Parsing Strategies:**
  - [TopDownInitRule, TopDownExpandRule, FundamentalRule]
  - [BottomUpRule, FundamentalRule]
- **Demonstration:**
  - python nltk/draw/chart.py

There's more to NLTK

- corpora [nltk.corpus]
  - more than 16 in NLTK data
- probabilistic parsing [nltk.parser.probabilistic]
- classification [nltk.feature, nltk.classifier]
  - maximum entropy, naive Bayes
- hidden Markov models [nltk.hmm]
- clustering [ntk.clusterer]
- stemming [nltk.stemmer]
- user contributions [nltk_contrib]
  - wordnet interface, festival interface, user projects
- ... and much more