Natural Language Processing

Syntax and Parsing

Dan Klein – UC Berkeley
Syntax
The move followed a round of similar increases by other lenders, reflecting a continuing decline in that market.
Phrase Structure Parsing

- Phrase structure parsing organizes syntax into constituents or brackets.

- In general, this involves nested trees.

- Linguists can, and do, argue about details.

- Lots of ambiguity.

- Not the only kind of syntax...

  new art critics write reviews with computers
Constituency Tests

- How do we know what nodes go in the tree?

- Classic constituency tests:
  - Substitution by *proform*
  - Question answers
  - Semantic grounds
    - Coherence
    - Reference
    - Idioms
  - Dislocation
  - Conjunction

- Cross-linguistic arguments, too
Conflicting Tests

- Constituency isn’t always clear
  - Units of transfer:
    - think about ~ penser à
    - talk about ~ hablar de

- Phonological reduction:
  - I will go → I’ll go
  - I want to go → I wanna go
  - a le centre → au centre

- Coordination
  - He went to and came from the store.
Q: Do we model deep vs surface structure?

[Example: Johnson 02]
changesoccured 0 -NONE-  
*T*-1
[Example: Cai et al 11]
Ambiguities
One basic kind of linguistic structure: syntactic word classes

- Open class (lexical) words
  - Nouns
    - Proper: IBM, Italy
    - Common: cat, cats, snow
  - Verbs
    - Main: see, registered
  - Adjectives: yellow
  - Adverbs: slowly
  - Numbers: 122,312, one
- Closed class (functional)
  - Determiners: the, some
  - Conjunctions: and, or
  - Pronouns: he, its
  - Prepositions: to, with
  - Particles: off, up

... more
Part-of-Speech Ambiguity

- Words can have multiple parts of speech
  
  VBD       VB
  VBN       VBZ     VBP     VBZ
  NNP       NNS     NN      NNS   CD    NN

  Fed raises interest rates 0.5 percent

- Two basic sources of constraint:
  - Grammatical environment
  - Identity of the current word

- Many more possible features:
  - Suffixes, capitalization, name databases (gazetteers), etc...
Why POS Tagging?

- **Useful in and of itself (more than you’d think)**
  - Text-to-speech: record, lead
  - Lemmatization: saw[v] → see, saw[n] → saw
  - Quick-and-dirty NP-chunk detection: grep {JJ | NN}* {NN | NNS}

- **Useful as a pre-processing step for parsing**
  - Less tag ambiguity means fewer parses
  - However, some tag choices are better decided by parsers

```
IN
DT  NNP  NN  VBD  VBN  RP  NN  NNS
The Georgia branch had taken on loan commitments …
```

```
VDN
DT  NN  IN  NN  VBD  NNS  VBD
The average of interbank offered rates plummeted …
```
Classical NLP: Parsing

- Write symbolic or logical rules:

<table>
<thead>
<tr>
<th>Grammar (CFG)</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT → S</td>
<td>NP → NP PP</td>
</tr>
<tr>
<td>S → NP VP</td>
<td>VP → VBP NP</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>VP → VBP NP PP</td>
</tr>
<tr>
<td>NP → NN NNS</td>
<td>PP → IN NP</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

- Use deduction systems to prove parses from words
  - Minimal grammar on “Fed raises” sentence: 36 parses
  - Simple 10-rule grammar: 592 parses
  - Real-size grammar: many millions of parses

- This scaled very badly, didn’t yield broad-coverage tools
Ambiguities: PP Attachment

The board approved [its acquisition] [by Royal Trustco Ltd.]
[for $27 a share]
[at its monthly meeting].
Attachments

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink
Syntactic Ambiguities I

- **Prepositional phrases:**
  *They cooked the beans in the pot on the stove with handles.*

- **Particle vs. preposition:**
  *The puppy tore up the staircase.*

- **Complement structures**
  *The tourists objected to the guide that they couldn’t hear.*
  *She knows you like the back of her hand.*

- **Gerund vs. participial adjective**
  *Visiting relatives can be boring.*
  *Changing schedules frequently confused passengers.*
Syntactic Ambiguities II

- **Modifier scope within NPs**
  - *impractical design requirements*
  - *plastic cup holder*

- **Multiple gap constructions**
  - *The chicken is ready to eat.*
  - *The contractors are rich enough to sue.*

- **Coordination scope:**
  - *Small rats and mice can squeeze into holes or cracks in the wall.*
Dark Ambiguities

- **Dark ambiguities**: most analyses are shockingly bad (meaning, they don’t have an interpretation you can get your mind around)

  This analysis corresponds to the correct parse of

  “This will panic buyers !”

- **Unknown words and new usages**
- **Solution**: We need mechanisms to focus attention on the best ones, probabilistic techniques do this
Ambiguities as Trees

(a) S
   /   
 NP  VP
   /   
 NP  PP
   /   
...Lehman Hutton Inc. by yesterday afternoon

(b) NP
   /   
 OP
   /   
PDT DT PDT PDT
  /   /   |
...half a dozen newspapers

(c) VP
   /   
 VBZ ADVP ADJP ADJP
   /   |
 s just fine
PCFGs
A context-free grammar is a tuple \(<N, T, S, R>\)

- \(N\) : the set of non-terminals
  - Phrasal categories: S, NP, VP, ADJP, etc.
  - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
- \(T\) : the set of terminals (the words)
- \(S\) : the start symbol
  - Often written as ROOT or TOP
  - *Not* usually the sentence non-terminal \(S\)
- \(R\) : the set of rules
  - Of the form \(X \rightarrow Y_1 Y_2 \ldots Y_k\), with \(X, Y_i \in N\)
  - Examples: \(S \rightarrow NP \ \ VP\), \(VP \rightarrow VP \ CC \ VP\)
  - Also called rewrites, productions, or local trees

A PCFG adds:

- A top-down production probability per rule \(P(Y_1 Y_2 \ldots Y_k \mid X)\)
( (S (NP-SBJ The move) (VP followed (NP (NP a round) (PP of (NP (NP similar increases) (PP by (NP other lenders)) (PP against (NP Arizona real estate loans)))))) , (S-ADV (NP-SBJ *)) (VP reflecting (NP (NP a continuing decline) (PP-LOC in (NP that market))))))
Treebank Grammars

- Need a PCFG for broad coverage parsing.
- Can take a grammar right off the trees (doesn’t work well):

  \[
  \begin{align*}
  \text{ROOT} & \rightarrow \text{S} & 1 \\
  \text{S} & \rightarrow \text{NP VP .} & 1 \\
  \text{NP} & \rightarrow \text{PRP} & 1 \\
  \text{VP} & \rightarrow \text{VBD ADJP} & 1 \\
  \end{align*}
  \]

- Better results by enriching the grammar (e.g., lexicalization).
- Can also get state-of-the-art parsers without lexicalization.
Treebank Grammar Scale

- Treebank grammars can be enormous
  - As FSAs, the raw grammar has ~10K states, excluding the lexicon
  - Better parsers usually make the grammars larger, not smaller
Chomsky Normal Form

- **Chomsky normal form:**
  - All rules of the form $X \rightarrow Y Z$ or $X \rightarrow w$
  - In principle, this is no limitation on the space of (P)CFGs
    - N-ary rules introduce new non-terminals

- Unaries / empty phrases are “promoted”
- In practice it’s kind of a pain:
  - Reconstructing n-aries is easy
  - Reconstructing unaries is trickier
  - The straightforward transformations don’t preserve tree scores
- Makes parsing algorithms simpler!
A Recursive Parser

\[
\text{bestScore}(X,i,j) \\
\text{if } (j = i+1) \\
\quad \text{return } \text{tagScore}(X,s[i]) \\
\text{else} \\
\quad \text{return max } \text{score}(X\rightarrow YZ) * \text{bestScore}(Y,i,k) * \text{bestScore}(Z,k,j)
\]

- Will this parser work?
- Why or why not?
- Memory requirements?
A Memoized Parser

- One small change:

```python
bestScore(X, i, j)
    if (scores[X][i][j] == null)
        if (j = i+1)
            score = tagScore(X, s[i])
        else
            score = max score(X->YZ) *
            bestScore(Y, i, k) *
            bestScore(Z, k, j)
    scores[X][i][j] = score
return scores[X][i][j]
```
A Bottom-Up Parser (CKY)

- Can also organize things bottom-up

```plaintext
bestScore(s)
  for (i : [0,n-1])
    for (X : tags[s[i]])
      score[X][i][i+1] =
        tagScore(X,s[i])
  for (diff : [2,n])
    for (i : [0,n-diff])
      j = i + diff
      for (X->YZ : rule)
        for (k : [i+1, j-1])
          score[X][i][j] = max score[X][i][j],
                          score(X->YZ) * score[Y][i][k] * score[Z][k][j]
```
Unary Rules

- Unary rules?

```python
bestScore(X, i, j, s)
    if (j = i+1)
        return tagScore(X, s[i])
    else
        return max
            max score(X->YZ) * 
            bestScore(Y, i, k) * 
            bestScore(Z, k, j) 
        max score(X->Y) * 
        bestScore(Y, i, j)
```
We need unaries to be non-cyclic

- Can address by pre-calculating the *unary closure*
- Rather than having zero or more unaries, always have exactly one

Alternate unary and binary layers
Reconstruct unary chains afterwards
Alternating Layers

\[
\text{bestScoreB}(X,i,j,s) = \begin{cases} 
\text{tagScore}(X,s[i]) & \text{if } j = i+1 \\
\max \left( \max_{X'\to YZ} \text{score}(X'\to YZ) \times \right. \\
& \left. \text{bestScoreU}(Y,i,k) \times \right. \\
& \left. \text{bestScoreU}(Z,k,j) \right) & \text{otherwise}
\end{cases}
\]

\[
\text{bestScoreU}(X,i,j,s) = \begin{cases} 
\text{tagScore}(X,s[i]) & \text{if } j = i+1 \\
\max \left( \max_{X'\to Y} \text{score}(X'\to Y) \times \right. \\
& \left. \text{bestScoreB}(Y,i,j) \right) & \text{otherwise}
\end{cases}
\]
Learning PCFGs
Treebank PCFGs

- Use PCFGs for broad coverage parsing
- Can take a grammar right off the trees (doesn’t work well):

```
ROOT
  | S
  | NP  VP
  | PRP VBD ADJP
  | He was JJ
  | right
```

```
ROOT → S  1
S → NP VP .  1
NP → PRP  1
VP → VBD ADJP  1
.....
```

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>72.0</td>
</tr>
</tbody>
</table>
Conditional Independence?

- Not every NP expansion can fill every NP slot
  - A grammar with symbols like “NP” won’t be context-free
  - Statistically, conditional independence too strong
Non-Independence

- Independence assumptions are often too strong.

- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
- Also: the subject and object expansions are correlated!
Grammar Refinement

- Example: PP attachment

```
They raised a point of order
```

```
                        VP
                           |
They

                        NP
                           |
raised

```

```
Grammar Refinement

- **Structure Annotation** [Johnson ’98, Klein&Manning ’03]
- **Lexicalization** [Collins ’99, Charniak ’00]
- **Latent Variables** [Matsuzaki et al. 05, Petrov et al. ’06]
Structural Annotation
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Structural annotation

```
S
  NP^S
  |   VP
  |   NP^VP
  PRP    VBD    DT    NN
  She    heard    the    noise
```
Lexicalization
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Structural annotation [Johnson '98, Klein and Manning 03]
  - Head lexicalization [Collins '99, Charniak '00]
Problems with PCFGs

- If we do no annotation, these trees differ only in one rule:
  - VP $\rightarrow$ VP PP
  - NP $\rightarrow$ NP PP
- Parse will go one way or the other, regardless of words
- We addressed this in one way with unlexicalized grammars (how?)
- Lexicalization allows us to be sensitive to specific words
Problems with PCFGs

- What’s different between basic PCFG scores here?
- What (lexical) correlations need to be scored?
Lexicalized Trees

- Add “head words” to each phrasal node
  - Syntactic vs. semantic heads
  - Headship not in (most) treebanks
  - Usually use head rules, e.g.:
    - **NP:**
      - Take leftmost NP
      - Take rightmost N*
      - Take rightmost JJ
      - Take right child
    - **VP:**
      - Take leftmost VB*
      - Take leftmost VP
      - Take left child
Lexicalized PCFGs?

- Problem: we now have to estimate probabilities like
  
  \[ \text{VP(saw)} \rightarrow \text{VBD(saw) NP-C(her) NP(today)} \]

- Never going to get these atomically off of a treebank

- Solution: break up derivation into smaller steps
Lexical Derivation Steps

- A derivation of a local tree [Collins 99]

```
VP(saw)
  
VBD(saw)

VP(saw)
  
VBD(saw) {NP-C( )}

VP(saw)
  
VBD(saw) NP-C( ) NP( )

VP(saw)
  
VBD(saw) NP-C(her) NP(today)
```

- Choose a head tag and word
- Choose a complement bag
- Generate children (incl. adjuncts)
- Recursively derive children
Lexicalized CKY

\[ \text{bestScore}(X,i,j,h) \]

\[
\begin{align*}
\text{if } (j = i+1) \\
\text{return } \text{tagScore}(X,s[i])
\end{align*}
\]

\[
\text{else} \\
\text{return } \max_{k,h',X \rightarrow YZ} \max_{k,h',X \rightarrow YZ} \text{score}(X[h] \rightarrow Y[h'] Z[h']) \times \\
\text{bestScore}(Y,i,k,h') \times \\
\text{bestScore}(Z,k,j,h')
\]

\[
\text{max } \text{score}(X[h] \rightarrow Y[h'] Z[h]) \times \\
\text{bestScore}(Y,i,k,h') \times \\
\text{bestScore}(Z,k,j,h)
\]
Results

- **Some results**
  - Collins 99 – 88.6 F1 (generative lexical)
  - Charniak and Johnson 05 – 89.7 / 91.3 F1 (generative lexical / reranked)
  - Petrov et al 06 – 90.7 F1 (generative unlexical)
  - McClosky et al 06 – 92.1 F1 (gen + rerank + self-train)

- **However**
  - Bilexical counts rarely make a difference (why?)
  - Gildea 01 – Removing bilexical counts costs < 0.5 F1
Latent Variable PCFGs
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson ’98]
  - Head lexicalization [Collins ’99, Charniak ’00]
  - Automatic clustering?
Latent Variable Grammars

Parse Tree

Sentence

Derivations $t : T$

Parameters $\theta$
Learning Latent Annotations

EM algorithm:
- Brackets are known
- Base categories are known
- Only induce subcategories

Just like Forward-Backward for HMMs.
Refinement of the DT tag

DT

the (0.50)
a (0.24)
The (0.08)

a (0.61)
the (0.19)
an (0.11)

the (0.80)
The (0.15)
a (0.01)

this (0.39)
that (0.28)
That (0.11)

some (0.20)
all (0.19)
those (0.12)

DT-1  DT-2  DT-3  DT-4
Hierarchical refinement
Hierarchical Estimation Results

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat Training</td>
<td>87.3</td>
</tr>
<tr>
<td>Hierarchical Training</td>
<td>88.4</td>
</tr>
</tbody>
</table>
Refinement of the , tag

- Splitting all categories equally is wasteful:
Adaptive Splitting

- Want to split complex categories more
- Idea: split everything, roll back splits which were least useful
Adaptive Splitting Results

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>88.4</td>
</tr>
<tr>
<td>With 50% Merging</td>
<td>89.5</td>
</tr>
</tbody>
</table>
Number of Phrasal Subcategories
Number of Lexical Subcategories
Learned Splits

- **Proper Nouns (NNP):**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP-12</td>
<td>John</td>
<td>Robert</td>
<td>James</td>
</tr>
<tr>
<td>NNP-2</td>
<td>J.</td>
<td>E.</td>
<td>L.</td>
</tr>
<tr>
<td>NNP-1</td>
<td>Bush</td>
<td>Noriega</td>
<td>Peters</td>
</tr>
<tr>
<td>NNP-15</td>
<td>New</td>
<td>San</td>
<td>Wall</td>
</tr>
<tr>
<td>NNP-3</td>
<td>York</td>
<td>Francisco</td>
<td>Street</td>
</tr>
</tbody>
</table>

- **Personal pronouns (PRP):**

<table>
<thead>
<tr>
<th>PRP-0</th>
<th>It</th>
<th>He</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP-1</td>
<td>it</td>
<td>he</td>
<td>they</td>
</tr>
<tr>
<td>PRP-2</td>
<td>it</td>
<td>them</td>
<td>him</td>
</tr>
</tbody>
</table>
Learned Splits

- **Relative adverbs (RBR):**
  
<table>
<thead>
<tr>
<th>RBR</th>
<th>Further</th>
<th>Lower</th>
<th>Higher</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBR-0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBR-1</td>
<td>more</td>
<td>less</td>
<td>More</td>
</tr>
<tr>
<td>RBR-2</td>
<td>earlier</td>
<td>Earlier</td>
<td>later</td>
</tr>
</tbody>
</table>

- **Cardinal Numbers (CD):**
  
<table>
<thead>
<tr>
<th>CD</th>
<th>One</th>
<th>Two</th>
<th>Three</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD-7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD-4</td>
<td>1989</td>
<td>1990</td>
<td>1988</td>
</tr>
<tr>
<td>CD-11</td>
<td>million</td>
<td>billion</td>
<td>trillion</td>
</tr>
<tr>
<td>CD-0</td>
<td>1</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>CD-3</td>
<td>1</td>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>CD-9</td>
<td>78</td>
<td>58</td>
<td>34</td>
</tr>
</tbody>
</table>
Coarse-to-Fine Inference

- Example: PP attachment

```
S
  NP
    PRP
      They
  VP
    ?????????
      V
        raised
      NP
        DT
          a
        NN
          point
      PP
        IN
          of
        NP
          order
```
Hierarchical Pruning

coarse:

split in two:

split in four:

split in eight:
Bracket Posteriors
Other Syntactic Models
Parse Reranking

- Assume the number of parses is very small
- We can represent each parse \( T \) as a feature vector \( \varphi(T) \)
  - Typically, all local rules are features
  - Also non-local features, like how right-branching the overall tree is
  - [Charniak and Johnson 05] gives a rich set of features
Dependency Parsing

- Lexicalized parsers can be seen as producing *dependency trees*

- Each local binary tree corresponds to an attachment in the dependency graph
Pure dependency parsing is only cubic [Eisner 99]

Some work on \textit{non-projective} dependencies
- Common in, e.g. Czech parsing
- Can do with MST algorithms [McDonald and Pereira 05]
Shift-Reduce Parsers

- Another way to derive a tree:

- Parsing
  - No useful dynamic programming search
  - Can still use beam search [Ratnaparkhi 97]
Data-oriented parsing:

- Rewrite large (possibly lexicalized) subtrees in a single step

- Formally, a tree-insertion grammar
- Derivational ambiguity whether subtrees were generated atomically or compositionally
- Most probable parse is NP-complete
TIG: Insertion

\[
\begin{align*}
\phi & : A \\
\psi & : A \\
\phi' & : A
\end{align*}
\]

\[
\begin{align*}
S & \quad \text{NP} \rightarrow \text{VP} \\
V & \quad \text{NP} \rightarrow \text{man} \\
saw & \quad \text{NP} \rightarrow \text{man} \\
\end{align*}
\]
Tree-adjoining grammars

- Start with *local trees*
- Can insert structure with *adjunction* operators
- Mildly context-sensitive
- Models long-distance dependencies naturally
- ... as well as other weird stuff that CFGs don’t capture well (e.g. cross-serial dependencies)
TAG: Long Distance
CCG Parsing

- **Combinatory Categorial Grammar**
  - Fully (mono-) lexicalized grammar
  - Categories encode argument sequences
  - Very closely related to the lambda calculus (more later)
  - Can have spurious ambiguities (why?)

\[
\begin{align*}
John & \vdash \text{NP} \\
shares & \vdash \text{NP} \\
buys & \vdash (S\backslash\text{NP})/\text{NP} \\
sleeps & \vdash S\backslash\text{NP} \\
well & \vdash (S\backslash\text{NP})\backslash(S\backslash\text{NP})
\end{align*}
\]

```
S
 NP  S\backslash NP
  John  (S\backslash NP)/NP \NP
     buys  shares
```