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.
Questions from Last Time

Q: Do we model deep vs surface structure?

[Example: Johnson 02]
[Example: Johnson 02]
[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*
  - **Auxiliary**
    - *can*
    - *had*

- **Adjectives**
  - *yellow*

- **Adverbs**
  - *slowly*

- **Numbers**
  - *122,312*
  - *one*

- **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
NNP      NNS  NN
```

Fed raises interest rates 0.5 percent

Mrs./NNP Shaefer/NNP never/RB got/VBD around/RP to/TO joining/VBG
All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB around/IN the/DT corner/NN
Chateau/NNP Petrus/NNP costs/VBZ around/RB 250/CD

- 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:
  
  Grammar (CFG)  
  Lexicon  
  
  ROOT → S  
  NP → NP PP  
  NN → interest  
  S → NP VP  
  VP → VBP NP  
  NNS → raises  
  NP → DT NN  
  VP → VBP NP PP  
  VBP → interest  
  NP → NN NNS  
  PP → IN NP  
  VBZ → raises  
  
- 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
The board approved [its acquisition] [by Royal Trustco Ltd.]

[of Toronto]

[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
----
NP
  |  VP
  |  PP
  |  had already ...
 NP
  |  VP
  |  PP
  |  by yesterday afternoon
```

(b) 
```
NP
----
PDT
    |  DT
    |  PDT
    |  PDT
    |  PDT
    |  ... half
    |  a
    |  dozen newspapers
```

(c) 
```
VP
----
VBZ
    |  ADVP
    |  ADJP
    |  ADJP
    |  just
    |  fine
```

- Ambiguities as Trees...
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 \(P(Y_1 Y_2 \ldots Y_k | X)\)
( (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))))))

.)
Need a PCFG for broad coverage parsing.

Can take a grammar right off the trees (doesn’t work well):

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 / empties 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!
CKY Parsing
A Recursive Parser

- 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?

```plaintext
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

bestScoreB(X,i,j,s)

return max max score(X→YZ) *

    bestScoreU(Y,i,k) *
    bestScoreU(Z,k,j)

bestScoreU(X,i,j,s)

if (j = i+1)

    return tagScore(X,s[i])
else

    return max max score(X→Y) *

        bestScoreB(Y,i,j)
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>
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
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

raised

a point

of order

```
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
Annotation refines base treebank symbols to improve statistical fit of the grammar

- **Structural annotation**
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

  \[ VP(\text{saw}) \rightarrow \text{VB}(\text{saw}) \text{ NP-C(her) NP(today)} \]

- **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]

Choose a head tag and word

Choose a complement bag

Generate children (incl. adjuncts)

Recursively derive children
bestScore(X,i,j,h)
    if (j = i+1)
        return tagScore(X,s[i])
    else
        return
            \[
            \max_{k,h',X \rightarrow YZ} \max_{k,h',X \rightarrow YZ} \text{score}(X[h] \rightarrow Y[h] \ Z[h']) \times \]
            \[
            \max_{k,h',X \rightarrow YZ} \text{bestScore}(Y,i,k,h) \times \]
            \[
            \text{bestScore}(Z,k,j,h') \]
            \[
            \max_{k,h',X \rightarrow YZ} \text{score}(X[h] \rightarrow Y[h'] \ Z[h]) \times \]
            \[
            \text{bestScore}(Y,i,k,h') \times \]
            \[
            \text{bestScore}(Z,k,j,h) \]
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 $T$
Sentence $w$
Derivations $t : T$
Parameters $\theta$

Grammar G

<table>
<thead>
<tr>
<th>Production</th>
<th>Right-Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0 \rightarrow NP_0 \ VP_0$</td>
<td>?</td>
</tr>
<tr>
<td>$S_0 \rightarrow NP_1 \ VP_0$</td>
<td>?</td>
</tr>
<tr>
<td>$S_0 \rightarrow NP_0 \ VP_1$</td>
<td>?</td>
</tr>
<tr>
<td>$S_0 \rightarrow NP_1 \ VP_1$</td>
<td>?</td>
</tr>
<tr>
<td>$S_1 \rightarrow NP_0 \ VP_0$</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$S_1 \rightarrow NP_1 \ VP_1$</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$NP_0 \rightarrow PRP_0$</td>
<td>?</td>
</tr>
<tr>
<td>$NP_0 \rightarrow PRP_1$</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Lexicon

<table>
<thead>
<tr>
<th>Word</th>
<th>Right-Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PRP_0 \rightarrow She$</td>
<td>?</td>
</tr>
<tr>
<td>$PRP_1 \rightarrow She$</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$VBD_0 \rightarrow was$</td>
<td>?</td>
</tr>
<tr>
<td>$VBD_1 \rightarrow was$</td>
<td>?</td>
</tr>
<tr>
<td>$VBD_2 \rightarrow was$</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
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

```
the (0.54)
a (0.25)
The (0.09)
```
```
a (0.61)
the (0.19)
an (0.11)
```
```
the (0.80)
The (0.15)
a (0.01)
```
```
the (0.96)
a (0.01)
The (0.01)
```
```
The (0.93)
A (0.02)
No (0.01)
```
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
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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBR-2</td>
<td></td>
<td></td>
<td></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
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
Lexicalized parsers can be seen as producing *dependency trees*

- Each local binary tree corresponds to an attachment in the dependency graph
Dependency Parsing

- Pure dependency parsing is only cubic [Eisner 99]

- Some work on non-projective dependencies
  - Common in, e.g. Czech parsing
  - Can do with MST algorithms [McDonald and Pereira 05]
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

\( \phi \)

\( \psi \)

\( \phi' \)

\( \psi \)

S

NP \downarrow

VP

V NP \downarrow

saw

NP

D \downarrow N

man

S

NP

V NP \downarrow

man

saw
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)
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?)

```
John ⊨ NP
shares ⊨ NP
buys ⊨ (S\NP)/NP
sleeps ⊨ S\NP
well ⊨ (S\NP)\(S\NP)
```

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