Algorithms for NLP

Parsing III

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
- 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.
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.

La vélocité des ondes sismiques
Classical NLP: Parsing

- Write symbolic or logical rules:
  
  Grammar (CFG)  
  Lexicon  
  
  ROOT $\rightarrow$ S  
  NP $\rightarrow$ NP PP  
  NN $\rightarrow$ interest  
  S $\rightarrow$ NP VP  
  VP $\rightarrow$ VBP NP  
  NNS $\rightarrow$ raises  
  NP $\rightarrow$ DT NN  
  VP $\rightarrow$ VBP NP PP  
  VBP $\rightarrow$ interest  
  NP $\rightarrow$ NN NNS  
  PP $\rightarrow$ IN NP  
  VBZ $\rightarrow$ 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
Ambiguities
Ambiguities: PP Attachment

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
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
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):

```
ROOT
  \|-- S
   \|-- NP VP
      \|-- PRP
          \|-- VBD ADJP
              \|-- VBD
                  \|-- ADJP
                      \|-- right
```

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

- 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?

\[
\text{bestScore}(X,i,j,s) \\
\quad \text{if } (j = i+1) \\
\quad \quad \text{return } \text{tagScore}(X,s[i]) \\
\quad \text{else} \\
\quad \quad \text{return max } \text{score}(X->YZ) \times \\
\quad \quad \quad \text{bestScore}(Y,i,k) \times \\
\quad \quad \quad \text{bestScore}(Z,k,j)
\]
A Memoized Parser

- One small change:

```plaintext
bestScore(X,i,j,s)
    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

```
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
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)
Analysis
Memory

- **How much memory does this require?**
  - Have to store the score cache
  - Cache size: $|\text{symbols}| \times n^2$ doubles
  - For the plain treebank grammar:
    - $X \sim 20K$, $n = 40$, double $\sim 8$ bytes $= \sim 256$MB
    - Big, but workable.

- **Pruning: Beams**
  - score[$X$][$i$][$j$] can get too large (when?)
  - Can keep beams (truncated maps score[$i$][$j$]) which only store the best few scores for the span [$i,j$]

- **Pruning: Coarse-to-Fine**
  - Use a smaller grammar to rule out most $X[i,j]$
  - Much more on this later…
How much time will it take to parse?

For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k
        Do constant work

Total time: |rules| * n^3

Something like 5 sec for an unoptimized parse of a 20-word sentence
Parsing with the vanilla treebank grammar:

- Why’s it worse in practice?
  - Longer sentences “unlock” more of the grammar
  - All kinds of systems issues don’t scale

~ 20K Rules (not an optimized parser!)

Observed exponent: 3.6
Same-Span Reachability

ADJP ADVP FRAG INTJ NP PP PRN QP S SBAR UCP VP WHNP

NX SQ X RRC

SINV

WHADJP SBARQ WHPP

WHADVP
Many states are more likely to match larger spans!
Efficient CKY

- **Lots of tricks to make CKY efficient**
  - Some of them are little engineering details:
    - E.g., first choose k, then enumerate through the $Y:[i,k]$ which are non-zero, then loop through rules by left child.
    - Optimal layout of the dynamic program depends on grammar, input, even system details.
  - Another kind is more important (and interesting):
    - Many $X[i,j]$ can be suppressed on the basis of the input string
    - We’ll see this next class as figures-of-merit, A* heuristics, coarse-to-fine, etc
Agenda-Based Parsing
Agenda-Based Parsing

- Agenda-based parsing is like graph search (but over a hypergraph)

- Concepts:
  - Numbering: we number fenceposts between words
  - “Edges” or items: spans with labels, e.g. PP[3,5], represent the sets of trees over those words rooted at that label (cf. search states)
  - A chart: records edges we’ve expanded (cf. closed set)
  - An agenda: a queue which holds edges (cf. a fringe or open set)
Building an item for the first time is called discovery. Items go into the agenda on discovery.

To initialize, we discover all word items (with score 1.0).
When we pop a word item, the lexicon tells us the tag item successors (and scores) which go on the agenda.
Item Successors

- When we pop items off of the agenda:
  - Graph successors: unary projections (NNS → critics, NP → NNS)
    
    \[ Y[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j] \]

  - Hypergraph successors: combine with items already in our chart
    
    \[ Y[i,j] \text{ and } Z[j,k] \text{ with } X \rightarrow Y Z \text{ form } X[i,k] \]

- Enqueue / promote resulting items (if not in chart already)
- Record backtraces as appropriate
- Stick the popped edge in the chart (closed set)

- Queries a chart must support:
  - Is edge \( X[i,j] \) in the chart? (What score?)
  - What edges with label \( Y \) end at position \( j \)?
  - What edges with label \( Z \) start at position \( i \)?
An Example

Critics write reviews with computers.
Empty Elements

- Sometimes we want to posit nodes in a parse tree that don’t contain any pronounced words:
  
  I want you to parse this sentence
  I want [    ] to parse this sentence

- These are easy to add to a agenda-based parser!
  - For each position i, add the “word” edge $\varepsilon[i,i]$
  - Add rules like $\text{NP} \rightarrow \varepsilon$ to the grammar
  - That’s it!
UCS / A*

- With weighted edges, order matters
  - Must expand optimal parse from bottom up (subparses first)
  - CKY does this by processing smaller spans before larger ones
  - UCS pops items off the agenda in order of decreasing Viterbi score
  - A* search also well defined

- You can also speed up the search without sacrificing optimality
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimality [Klein and Manning 03]
(Speech) Lattices

- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don’t know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily.
Learning PCFGs
Treebank PCFGs [Charniak 96]

- 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
S → NP VP .
NP → PRP
VP → VBD ADJP
...
```

<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
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
     PRP  VBD  NP^VP
       She  heard  DT  NN
         the  noise
```
Typical Experimental Setup

- **Corpus:** Penn Treebank, WSJ

Training: sections 02-21  
Development: section 22 (here, first 20 files)  
Test: section 23

- **Accuracy – F1:** harmonic mean of per-node labeled precision and recall.
- **Here:** also size – number of symbols in grammar.
Vertical Markovization

- Vertical Markov order: rewrites depend on past $k$ ancestor nodes. (cf. parent annotation)
Horizontal Markovization

Order 1

Order \( \infty \)

Symbols

Horizontal Markov Order
Unary Splits

- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.

- Solution: Mark unary rewrite sites with -U

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>
Tag Splits

- **Problem:** Treebank tags are too coarse.

- **Example:** Sentential, PP, and other prepositions are all marked IN.

- **Partial Solution:**
  - Subdivide the IN tag.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.3</td>
<td>8.1K</td>
</tr>
</tbody>
</table>
A Fully Annotated (Unlex) Tree

ROOT
  \n  S^ROOT-v
  \n  "^S NP^S-B
  \n  DT-U^NP VBZ^BE^VP NP^VP-B
  \n  This is NN^NP NN^NP
  \n  panic buying
### Some Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
<th>CB</th>
<th>0 CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
<td>1.26</td>
<td>56.6</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Unlexicalized</td>
<td>86.9</td>
<td>85.7</td>
<td>86.3</td>
<td>1.10</td>
<td>60.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
<td>1.00</td>
<td>62.1</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
<td>0.90</td>
<td>67.1</td>
</tr>
</tbody>
</table>

- Beats “first generation” lexicalized parsers.
- Lots of room to improve – more complex models next.
Efficient Parsing for Structural Annotation
Grammar Projections

Coarse Grammar

NP

PRP

He

VP

VBD

was

ADJP

right

S

Coarse Grammar

Fine Grammar

NP^S

→

DT^NP

N'[…DT]^NP

NP

→

DT

N'

Note: X-Bar Grammars are projections with rules like XP → Y X’ or XP → X’ Y or X’ → X
Coarse-to-Fine Pruning

For each coarse chart item $X[i,j]$, compute posterior probability:

$$\frac{P_{IN}(X, i, j) \cdot P_{OUT}(X, i, j)}{P_{IN}(root, 0, n)} < \text{threshold}$$

E.g. consider the span 5 to 12:
Computing (Max-)Marginals

\[
\beta(x, i, j) = \sum_{y} \pi(y|x) \beta(y, i, k) \beta(z, k, j)
\]
Inside and Outside Scores
Pruning with A*

- You can also speed up the search without sacrificing optimality
- For agenda-based parsers:
  - Can select which items to process first
  - Can do with any “figure of merit” [Charniak 98]
  - If your figure-of-merit is a valid A* heuristic, no loss of optimiality [Klein and Manning 03]
### A* Parsing

<table>
<thead>
<tr>
<th>Estimate</th>
<th>SX</th>
<th>SXL</th>
<th>SXLRE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>(1,6,NP)</td>
<td>(1,6,NP,VBZ)</td>
<td>(1,6,NP,VBZ,&quot;&quot;&quot;,&quot;)</td>
<td>(entire context)</td>
</tr>
<tr>
<td>Best Tree</td>
<td><img src="image1" alt="Tree" /></td>
<td><img src="image2" alt="Tree" /></td>
<td><img src="image3" alt="Tree" /></td>
<td><img src="image4" alt="Tree" /></td>
</tr>
<tr>
<td>Score</td>
<td>-11.3</td>
<td>-13.9</td>
<td>-15.1</td>
<td>-18.1</td>
</tr>
</tbody>
</table>
Lexicalization
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 → VP PP
  - NP → 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

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

- 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
Lexicalized CKY

\[ \text{bestScore}(X, i, j, h) \]
\[ \text{if } (j = i+1) \]
\[ \quad \text{return } \text{tagScore}(X, s[i]) \]
\[ \text{else} \]
\[ \quad \text{return } \max_{k, h', X \rightarrow YZ} \max_{k, h', X \rightarrow YZ} \text{score}(X[h] \rightarrow Y[h] Z[h']) \times \]
\[ \quad \text{bestScore}(Y, i, k, h) \times \]
\[ \quad \text{bestScore}(Z, k, j, h') \times \]
\[ \quad \max_{k, h', X \rightarrow YZ} \text{score}(X[h] \rightarrow Y[h'] Z[h]) \times \]
\[ \quad \text{bestScore}(Y, i, k, h') \times \]
\[ \quad \text{bestScore}(Z, k, j, h) \]
Efficient Parsing for Lexical Grammars
Quartic Parsing

- Turns out, you can do (a little) better [Eisner 99]

- Gives an $O(n^4)$ algorithm
- Still prohibitive in practice if not pruned
Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the $O(n^5)$ CKY
  - Remember only a few hypotheses for each span $<i,j>$.
  - If we keep $K$ hypotheses at each span, then we do at most $O(nK^2)$ work per span (why?)
  - Keeps things more or less cubic (and in practice is more like linear!)

- Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)
Pruning with a PCFG

- The Charniak parser prunes using a two-pass, coarse-to-fine approach [Charniak 97+]
  - First, parse with the base grammar
  - For each $X:[i,j]$ calculate $P(X|i,j,s)$
    - This isn’t trivial, and there are clever speed ups
  - Second, do the full $O(n^5)$ CKY
    - Skip any $X:[i,j]$ which had low (say, $< 0.0001$) posterior
    - Avoids almost all work in the second phase!
- Charniak et al 06: can use more passes
- Petrov et al 07: can use many more passes
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 \[ \mathcal{T} \]

Derivations \[ t : \mathcal{T} \]

Parameters \[ \theta \]
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)

DT-1
- a (0.61)
- the (0.19)
- an (0.11)

DT-2
- the (0.80)
- The (0.15)
- a (0.01)

DT-3
- this (0.39)
- that (0.28)
- That (0.11)

DT-4
- some (0.20)
- all (0.19)
- those (0.12)
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
AdapKve Spling Results

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>

Graph showing parsing accuracy (F1) vs total number of grammar symbols for different models:
- 50% Merging
- Hierarchical Training
- Flat Training
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):
  - RBR-0: further, lower, higher
  - RBR-1: more, less, More
  - RBR-2: earlier, Earlier, later

- Cardinal Numbers (CD):
  - CD-7: one, two, Three
  - CD-11: million, billion, trillion
  - CD-0: 1, 50, 100
  - CD-3: 1, 30, 31
  - CD-9: 78, 58, 34
## Final Results (Accuracy)

<table>
<thead>
<tr>
<th>Language</th>
<th>Method</th>
<th>≤ 40 words F1</th>
<th>all F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENG</td>
<td>Charniak&amp;Johnson ‘05 (generative)</td>
<td>90.1</td>
<td>89.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>90.6</td>
<td>90.1</td>
</tr>
<tr>
<td>GER</td>
<td>Dubey ‘05</td>
<td>76.3</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>80.8</td>
<td>80.1</td>
</tr>
<tr>
<td>CHN</td>
<td>Chiang et al. ‘02</td>
<td>80.0</td>
<td>76.6</td>
</tr>
<tr>
<td></td>
<td>Split / Merge</td>
<td>86.3</td>
<td>83.4</td>
</tr>
</tbody>
</table>

Still higher numbers from reranking / self-training methods
Efficient Parsing for Hierarchical Grammars
Coarse-to-Fine Inference

- Example: PP attachment
Hierarchical Pruning

coarse:

split in two:

split in four:

split in eight:

...
Bracket Posteriors
1621 min
111 min
35 min
15 min
(no search error)
Unsupervised Tagging
Unsupervised Tagging?

- AKA part-of-speech induction

**Task:**
- Raw sentences in
- Tagged sentences out

**Obvious thing to do:**
- Start with a (mostly) uniform HMM
- Run EM
- Inspect results
EM for HMMs: Process

- Alternate between recomputing distributions over hidden variables (the tags) and reestimating parameters
- Crucial step: we want to tally up how many (fractional) counts of each kind of transition and emission we have under current params:

\[
\text{count}(w, s) = \sum_{i : w_i = w} P(t_i = s | w)
\]

\[
\text{count}(s \rightarrow s') = \sum_i P(t_{i-1} = s, t_i = s' | w)
\]

- Same quantities we needed to train a CRF!
Merialdo: Setup

- Some (discouraging) experiments [Merialdo 94]

- Setup:
  - You know the set of allowable tags for each word
  - Fix k training examples to their true labels
    - Learn \( P(w|t) \) on these examples
    - Learn \( P(t|t_{-1},t_{-2}) \) on these examples
  - On \( n \) examples, re-estimate with EM

- Note: we know allowed tags but not frequencies
### Merialdo: Results

<table>
<thead>
<tr>
<th>Iter</th>
<th>0</th>
<th>100</th>
<th>2000</th>
<th>5000</th>
<th>10000</th>
<th>20000</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>77.0</td>
<td>90.0</td>
<td>95.4</td>
<td>96.2</td>
<td>96.6</td>
<td>96.9</td>
<td>97.0</td>
</tr>
<tr>
<td>1</td>
<td>80.5</td>
<td>92.6</td>
<td>95.8</td>
<td>96.3</td>
<td>96.6</td>
<td>96.7</td>
<td>96.8</td>
</tr>
<tr>
<td>2</td>
<td>81.8</td>
<td>93.0</td>
<td>95.7</td>
<td>96.1</td>
<td>96.3</td>
<td>96.4</td>
<td>96.4</td>
</tr>
<tr>
<td>3</td>
<td>83.0</td>
<td>93.1</td>
<td>95.4</td>
<td>95.8</td>
<td>96.1</td>
<td>96.2</td>
<td>96.2</td>
</tr>
<tr>
<td>4</td>
<td>84.0</td>
<td>93.0</td>
<td>95.2</td>
<td>95.5</td>
<td>95.8</td>
<td>96.0</td>
<td>96.0</td>
</tr>
<tr>
<td>5</td>
<td>84.8</td>
<td>92.9</td>
<td>95.1</td>
<td>95.4</td>
<td>95.6</td>
<td>95.8</td>
<td>95.8</td>
</tr>
<tr>
<td>6</td>
<td>85.3</td>
<td>92.8</td>
<td>94.9</td>
<td>95.2</td>
<td>95.5</td>
<td>95.6</td>
<td>95.7</td>
</tr>
<tr>
<td>7</td>
<td>85.8</td>
<td>92.8</td>
<td>94.7</td>
<td>95.1</td>
<td>95.3</td>
<td>95.5</td>
<td>95.5</td>
</tr>
<tr>
<td>8</td>
<td>86.1</td>
<td>92.7</td>
<td>94.6</td>
<td>95.0</td>
<td>95.2</td>
<td>95.4</td>
<td>95.4</td>
</tr>
<tr>
<td>9</td>
<td>86.3</td>
<td>92.6</td>
<td>94.5</td>
<td>94.9</td>
<td>95.1</td>
<td>95.3</td>
<td>95.3</td>
</tr>
<tr>
<td>10</td>
<td>86.6</td>
<td>92.6</td>
<td>94.4</td>
<td>94.8</td>
<td>95.0</td>
<td>95.2</td>
<td>95.2</td>
</tr>
</tbody>
</table>