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:

\[
\text{count}(w, s) = \sum_{i = 1}^{k} P(t_i = s \mid w)
\]

\[
\text{count}(s \rightarrow s') = \sum_{i = 1}^{k} P(t_i = s, t_{i-1} = s' \mid w)
\]

- 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_{i-1}, t_{i-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>Number of tagged sentences used for the initial model</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>Iter</td>
<td>Correct tags (%)</td>
<td>Iter ML on 3M words</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>81.3</td>
<td>92.4</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>96.6</td>
</tr>
<tr>
<td>3</td>
<td>83.0</td>
<td>93.3</td>
<td>95.6</td>
<td>96.1</td>
<td>96.3</td>
<td>96.2</td>
<td>96.2</td>
</tr>
<tr>
<td>4</td>
<td>84.0</td>
<td>92.0</td>
<td>95.2</td>
<td>95.2</td>
<td>95.8</td>
<td>96.0</td>
<td>96.1</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>95.0</td>
<td>95.1</td>
<td>95.3</td>
<td>95.5</td>
<td>95.5</td>
</tr>
<tr>
<td>7</td>
<td>86.8</td>
<td>92.8</td>
<td>94.7</td>
<td>95.3</td>
<td>95.5</td>
<td>95.5</td>
<td>95.5</td>
</tr>
<tr>
<td>8</td>
<td>86.7</td>
<td>92.8</td>
<td>94.5</td>
<td>95.0</td>
<td>95.2</td>
<td>94.4</td>
<td>95.4</td>
</tr>
<tr>
<td>9</td>
<td>86.3</td>
<td>92.8</td>
<td>94.5</td>
<td>94.9</td>
<td>95.3</td>
<td>95.3</td>
<td>95.3</td>
</tr>
<tr>
<td>10</td>
<td>86.6</td>
<td>92.8</td>
<td>94.4</td>
<td>94.8</td>
<td>95.0</td>
<td>95.2</td>
<td>95.2</td>
</tr>
</tbody>
</table>
Latent Variable PCFGs

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson ’98]

- Head lexicalization [Collins ’99, Charniak ’00]

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]

Latent Variable Grammars

- Brackets are known
- Base categories are known
- Only induce subcategories

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

Learning Latent Annotations

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

Hierarchical refinement

Hierarchical Estimation Results

Refinement of the , tag

Adaptive Splitting

Adaptive Splitting Results
Learned Splits

- **Proper Nouns (NNP):**
  - NNP-12: John, Robert, James
  - NNP-2: J., E., L.
  - NNP-1: Bush, Noriega, Peters
  - NNP-15: New, San, Wall
  - NNP-3: York, Francisco, Street

- **Personal pronouns (PRP):**
  - PRP-0: It, He, I
  - PRP-1: it, he, they
  - PRP-2: it, them, him

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

Parse Reranking

- Assume the number of parses is very small
- We can represent each parse $T$ as an arbitrary feature vector $\phi(T)$
  - 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

Other Syntactic Models
K-Best Parsing [Huang and Chiang 05, Pauls, Klein, Quirk 10]

Dependency Parsing

- 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]

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
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

- 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)

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

CCG Parsing