What’s Next for POS Tagging

- Better features!
- We could fix this with a feature that looked at the next word
- We could fix this by linking capitalized words to their lowercase versions
- Solution: maximum entropy sequence models
- Reality check:
  - Taggers are already pretty good on WSJ journal text…
  - What the world needs is taggers that work on other text!
  - Also: same techniques used for other sequence models (NER, etc)

Sequence-Free Tagging?

- What about looking at a word and it’s environment, but no sequence information?
- Add in previous / next word
- Previous / next word shapes
- Occurrence pattern features
- Crude entity detection
- Phrasal verb in sentence?
- Conjunctions of these things
- All features except sequence: 96.6% / 86.8%
- Uses lots of features: > 200K
- Why isn't this the standard approach?

Maxent Taggers

- One step up: also condition on previous tags
- Train up P(ti|w,t-i-1,t-i-2,i) as a normal maxent problem, then use to score sequences
- This is referred to as a maxent tagger [Ratnaparkhi 96]
- Beam search effective! (Why?)
- What's the advantage of beam size 1?

Feature Templates

- Important distinction:
  - Features: <w_i=future, t_o=JJ>
  - Feature templates: <w_i, t_o>
- In maxent taggers:
  - Can now add edge feature templates:
    - <t_i, t_o>
    - <t_i, t_o, w_i, t_o>
  - Also, mixed feature templates:
    - <t_i, w_i, t_o>
Decoding

Decoding maxent taggers:
- Just like decoding HMMs
- Viterbi, beam search, posterior decoding

Viterbi algorithm (HMMs):
\[
\delta_i(s) = \arg \max_{s'} \delta_i(s') P(s_i|s') P(w_i|s_i) \delta_{i-1}(s')
\]

Viterbi algorithm (Maxent):
\[
\delta_i(s) = \arg \max_{s'} P(s_i|s', w_i, \theta) \delta_{i-1}(s')
\]

TBL Tagger

[Brill 95] presents a transformation-based tagger
- Label the training set with most frequent tags
  
  DT MD VBD VBD
  The can was rusted.
  
- Add transformation rules which reduce training mistakes
  
  MD → NN: DT
  VBD → VBN: VBD
  
- Stop when no transformations do sufficient good
- Does this remind anyone of anything?

Probably the most widely used tagger (esp. outside NLP)
- … but not the most accurate: 96.6% / 82.0%

TBL Tagger II

What gets learned? [from Brill 95]

CRF Taggers

Newer, higher-powered discriminative sequence models
- CRFs (also voted perceptrons, M3Ns)
- Do not decompose training into independent local regions
- Can be deathly slow to train – require repeated inference on training set

Differences tend not to be too important for POS tagging
- Differences more substantial on other sequence tasks
- However: one issue worth knowing about in local models
  - “Label bias” and other explaining away effects
  - Maxent taggers’ local scores can be near one without having both good “transitions” and “emissions”
  - This means that often evidence doesn’t flow properly
  - Why isn’t this a big deal for POS tagging?

EngCG Tagger

English constraint grammar tagger
- [Tapanainen and Voutilainen 94]
- Something else you should know about
  - Hand-written and knowledge driven
  - “Don’t guess if you know” (general point about modeling more structure!)
  - Tag set doesn’t make all of the hard distinctions as the standard tag set (e.g. JJ/NN)
  - They get stellar accuracies: 98.5% on their tag set
  - Linguistic representation matters…
  - … but it’s easier to win when you make up the rules

Domain Effects

Accuracies degrade outside of domain
- Up to triple error rate
- Usually make the most errors on the things you care about in the domain (e.g. protein names)

Open questions
- How to effectively exploit unlabeled data from a new domain (what could we gain?)
- How to best incorporate domain lexica in a principled way (e.g. UMLS specialist lexicon, ontologies)
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

Forward Recurrence

Backward Recurrence

Fractional Transitions

EM for HMMs: Quantities

- Cache total path values:
  \[
  \alpha_i(s) = P(w_0 \ldots w_i, s_i) = \sum_{s_{i-1}} P(s_i|s_{i-1})P(w_i|s_i)\alpha_{i-1}(s_{i-1}),
  \]
  \[
  \beta_i(s) = P(w_i+1 \ldots w_n|s_i) = \sum_{s_{i+1}} P(s_{i+1}|s_i)P(w_{i+1}|s_{i+1})\beta_{i+1}(s_{i+1}),
  \]
- Can calculate in O(s^2n) time (why?)

EM for HMMs: Process

- From these quantities, we can re-estimate transitions:
  \[
  \text{count}(s \rightarrow s') = \frac{\sum_i\alpha_i(s)P(s'|s)P(w_i|s)\beta_{i+1}(s')}{P(w)}
  \]
- And emissions:
  \[
  \text{count}(w, s) = \sum_{s'|w_i=w} \alpha_i(s)\beta_{i+1}(s)\frac{P(w)}{P(w)}
  \]
- If you don’t get these formulas immediately, just think about hard EM instead, where we re-estimate from the Viterbi sequences
**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>Number of tagged sentences used for the initial model</th>
<th>0</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter</td>
<td>Correct tags (% w) after ML, on 3M words</td>
<td>77.0</td>
<td>90.0</td>
<td>91.4</td>
<td>96.3</td>
<td>96.8</td>
</tr>
<tr>
<td>1</td>
<td>83.0</td>
<td>94.6</td>
<td>95.9</td>
<td>96.8</td>
<td>96.9</td>
<td>96.7</td>
</tr>
<tr>
<td>3</td>
<td>77.0</td>
<td>90.0</td>
<td>91.4</td>
<td>96.3</td>
<td>96.8</td>
<td>96.7</td>
</tr>
<tr>
<td>6</td>
<td>84.0</td>
<td>92.0</td>
<td>95.2</td>
<td>95.8</td>
<td>96.0</td>
<td>96.1</td>
</tr>
<tr>
<td>10</td>
<td>85.0</td>
<td>92.0</td>
<td>95.2</td>
<td>95.8</td>
<td>96.0</td>
<td>96.1</td>
</tr>
<tr>
<td>15</td>
<td>86.0</td>
<td>92.0</td>
<td>95.2</td>
<td>95.8</td>
<td>96.0</td>
<td>96.1</td>
</tr>
</tbody>
</table>

**Distributional Clustering**

- Three main variants on the same idea:
  - Pairwise similarities and heuristic clustering
  - E.g. [Finch and Chater 92]
  - Produces dendrograms
  - Vector space methods
  - E.g. [Shuetze 93]
  - Models of ambiguity
  - Probabilistic methods
  - Various formulations, e.g. [Lee and Pereira 99]

**Nearest Neighbors**

<table>
<thead>
<tr>
<th>Word</th>
<th>Nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>the president</td>
<td>the, of, said, that, the, downturn, was, over</td>
</tr>
<tr>
<td>the president</td>
<td>the, of, said, that, the, downturn, was, over</td>
</tr>
<tr>
<td>the president</td>
<td>the, of, said, that, the, downturn, was, over</td>
</tr>
<tr>
<td>the president</td>
<td>the, of, said, that, the, downturn, was, over</td>
</tr>
</tbody>
</table>

Distributional Clustering

**Dendrograms**

[Finch and Chater 92, Shuetze 93, many others]
A Probabilistic Version?

\[
P(S, C) = \prod_i P(c_i) P(w_i | c_i) P(w_{i-1}, w_{i+1} | c_i)
\]

* the president said that the downturn was over

What Else?

- Various newer ideas:
  - Context distributional clustering [Clark 00]
  - Morphology-driven models [Clark 03]
  - Contrastive estimation [Smith and Eisner 05]

* Also:
  - What about ambiguous words?
  - Using wider context signatures has been used for learning synonyms (what’s wrong with this approach?)
  - Can extend these ideas for grammar induction (later)