Algorithms for NLP

Part-of-speech Tagging

Taylor Berg-Kirkpatrick – CMU

Slides: Dan Klein – UC Berkeley
Speech Training
What Needs to be Learned?

- **Emissions:** $P(x \mid \text{phone class})$
  - $X$ is MFCC-valued

- **Transitions:** $P(\text{state} \mid \text{prev state})$
  - If between words, this is $P(\text{word} \mid \text{history})$
  - If inside words, this is $P(\text{advance} \mid \text{phone class})$
  - (Really a hierarchical model)
Estimation from Aligned Data

- What if each time step was labeled with its (context-dependent sub) phone?

- Can estimate $P(x|/ae/)$ as empirical mean and (co-)variance of $x$’s with label /ae/

- Problem: Don’t know alignment at the frame and phone level
Forced Alignment

- What if the acoustic model $P(x|\text{phone})$ was known?
  - ... and also the correct sequences of words / phones

- Can predict the best alignment of frames to phones
  
  "speech lab"

- Called “forced alignment”
Forced Alignment

- Create a new state space that forces the hidden variables to transition through phones in the (known) order

\[ /s/ \rightarrow /p/ \rightarrow /ee/ \rightarrow /ch/ \rightarrow /l/ \rightarrow /ae/ \rightarrow /b/ \]

- Still have uncertainty about durations

- In this HMM, all the parameters are known
  - Transitions determined by known utterance
  - Emissions assumed to be known
  - Minor detail: self-loop probabilities

- Just run Viterbi (or approximations) to get the best alignment
EM for Alignment

- Input: acoustic sequences with word-level transcriptions

- We don’t know either the emission model or the frame alignments

- Expectation Maximization (Hard EM for now)
  - Alternating optimization
  - Impute completions for unlabeled variables (here, the states at each time step)
  - Re-estimate model parameters (here, Gaussian means, variances, mixture ids)
  - Repeat
  - One of the earliest uses of EM!
Soft EM

- Hard EM uses the best single completion
  - Here, single best alignment
  - Not always representative
  - Certainly bad when your parameters are initialized and the alignments are all tied
  - Uses the count of various configurations (e.g. how many tokens of /ae/ have self-loops)

- What we’d really like is to know the fraction of paths that include a given completion
  - E.g. 0.32 of the paths align this frame to /p/, 0.21 align it to /ee/, etc.
  - Formally want to know the expected count of configurations
  - Key quantity: \( P(s_t \mid x) \)
Computing Marginals

\[ P(s_t|x) = \frac{P(s_t, x)}{P(x)} \]

= sum of all paths through s at t
/ sum of all paths
Forward Scores

\[ v_t(s_t) = \max_{s_{t-1}} v_{t-1}(s_{t-1}) \phi_t(s_{t-1}, s_t) \]

\[ \alpha_t(s_t) = \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1}) \phi_t(s_{t-1}, s_t) \]
Backward Scores

\[ \beta_t(s_t) = \sum_{s_{t+1}} \beta_{t+1}(s_{t+1}) \phi_t(s_t, s_{t+1}) \]
Total Scores

\[ P(s_t, x) = \alpha_t(s_t)\beta_t(s_t) \]

\[ P(x) = \sum_{s_t} \alpha_t(s_t)\beta_t(s_t) \]

\[ = \alpha_T(\text{stop}) \]

\[ = \beta_0(\text{start}) \]
Computing fractional (expected) counts

- Compute forward / backward probabilities
- For each position, compute marginal posteriors
- Accumulate expectations
- Re-estimate parameters (e.g. means, variances, self-loop probabilities) from ratios of these expected counts
Staged Training and State Tying

- **Creating CD phones:**
  - Start with monophone, do EM training
  - Clone Gaussians into triphones
  - Build decision tree and cluster Gaussians
  - Clone and train mixtures (GMMs)

- **General idea:**
  - Introduce complexity gradually
  - Interleave constraint with flexibility
Parts of Speech
One basic kind of linguistic structure: syntactic word classes

<table>
<thead>
<tr>
<th>Open class (lexical) words</th>
<th>Closed class (functional)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nouns</strong></td>
<td><strong>Determiners</strong></td>
</tr>
<tr>
<td>Proper</td>
<td>the some</td>
</tr>
<tr>
<td>IBM</td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td><strong>Conjunctions</strong></td>
</tr>
<tr>
<td>cat / cats</td>
<td>and or</td>
</tr>
<tr>
<td>snow</td>
<td></td>
</tr>
<tr>
<td><strong>Verbs</strong></td>
<td><strong>Auxiliary</strong></td>
</tr>
<tr>
<td>Main</td>
<td>can</td>
</tr>
<tr>
<td>see</td>
<td>had</td>
</tr>
<tr>
<td>registered</td>
<td></td>
</tr>
<tr>
<td><strong>Adjectives</strong></td>
<td></td>
</tr>
<tr>
<td>yellow</td>
<td></td>
</tr>
<tr>
<td><strong>Adverbs</strong></td>
<td></td>
</tr>
<tr>
<td>slowly</td>
<td></td>
</tr>
<tr>
<td><strong>Numbers</strong></td>
<td></td>
</tr>
<tr>
<td>122,312</td>
<td>one</td>
</tr>
<tr>
<td>... more</td>
<td></td>
</tr>
<tr>
<td><strong>Prepositions</strong></td>
<td></td>
</tr>
<tr>
<td>to with</td>
<td></td>
</tr>
<tr>
<td><strong>Particles</strong></td>
<td></td>
</tr>
<tr>
<td>off up</td>
<td></td>
</tr>
<tr>
<td>... more</td>
<td></td>
</tr>
</tbody>
</table>
Part-of-Speech Ambiguity

- Words can have multiple parts of speech

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 …
```
Part-of-Speech Tagging
Classic Solution: HMMs

- We want a model of sequences \( s \) and observations \( w \)

\[
P(s, w) = \prod_{i} P(s_i | s_{i-1}) P(w_i | s_i)
\]

- Assumptions:
  - States are tag n-grams
  - Usually a dedicated start and end state / word
  - Tag/state sequence is generated by a markov model
  - Words are chosen independently, conditioned only on the tag/state
  - These are totally broken assumptions: why?
States

- States encode what is relevant about the past
- Transitions $P(s|s')$ encode well-formed tag sequences
  - In a bigram tagger, states = tags
    
    \[ s_0 ightarrow s_1 ightarrow s_2 ightarrow \ldots \rightarrow s_n \]
    
    \[ w_1 \rightarrow w_2 \rightarrow \ldots \rightarrow w_n \]
  - In a trigram tagger, states = tag pairs
    
    \[ s_0 ightarrow s_1 ightarrow s_2 \rightarrow \ldots \rightarrow s_n \]
    
    \[ w_1 \rightarrow w_2 \rightarrow \ldots \rightarrow w_n \]
Estimating Transitions

- Use standard smoothing methods to estimate transitions:

\[ P(t_i \mid t_{i-1}, t_{i-2}) = \lambda_2 \hat{P}(t_i \mid t_{i-1}, t_{i-2}) + \lambda_1 \hat{P}(t_i \mid t_{i-1}) + (1 - \lambda_1 - \lambda_2) \hat{P}(t_i) \]

- Can get a lot fancier (e.g. KN smoothing) or use higher orders, but in this case it doesn’t buy much

- One option: encode more into the state, e.g. whether the previous word was capitalized (Brants 00)

- BIG IDEA: The basic approach of state-splitting / refinement turns out to be very important in a range of tasks
Evaluating Emissions

\[ P(s, w) = \prod_{i} P(s_{i} | s_{i-1})P(w_{i} | s_{i}) \]

- Emissions are trickier:
  - Words we’ve never seen before
  - Words which occur with tags we’ve never seen them with
  - One option: break out the fancy smoothing (e.g. KN, Good-Turing)
  - Issue: unknown words aren’t black boxes:
    - 343,127.23 11-year Minteria reintroducibly

- Basic solution: unknown words classes (affixes or shapes)
  - \( D^{+}, D^{+}.D^{+} \)
  - \( D^{+}-x^{+} \)
  - \( Xx^{+} \)
  - \( x^{+}-\text{“ly”} \)

- Common approach: Estimate \( P(t | w) \) and invert
- [Brants 00] used a suffix trie as its (inverted) emission model
Disambiguation (Inference)

- Problem: find the most likely (Viterbi) sequence under the model

\[ t^* = \arg \max_t P(t|w) \]

- Given model parameters, we can score any tag sequence

Fed raises interest rates 0.5 percent.

\[
P(NNP|<\Diamond, \Diamond>) P(Fed|NNP) P(VBZ|<NNP, \Diamond>) P(raises|VBZ) P(\text{NN}|VBZ, NNP) \ldots
\]

- In principle, we’re done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

<table>
<thead>
<tr>
<th>Tag Sequence</th>
<th>logP</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP VBZ NN NNS CD NN</td>
<td>-23</td>
</tr>
<tr>
<td>NNP NNS NN NNS CD NN</td>
<td>-29</td>
</tr>
<tr>
<td>NNP VBZ VB NNS CD NN</td>
<td>-27</td>
</tr>
</tbody>
</table>
The State Lattice / Trellis

N   N   N   N   N   N   N
V   V   V   V   V   V   V
J   J   J   J   J   J   J
D   D   D   D   D   D   D
$   $   $   $   $   $   $

START  Fed  raises  interest  rates  END
The State Lattice / Trellis

\[
P(\text{Fed}|N) \quad P(N|N) \quad P(V|N)
\]

START       Fed           raises       interest         rates         END
So How Well Does It Work?

- Choose the most common tag
  - 90.3% with a bad unknown word model
  - 93.7% with a good one

- TnT (Brants, 2000):
  - A carefully smoothed trigram tagger
  - Suffix trees for emissions
  - 96.7% on WSJ text (SOTA is 97+%)

- Noise in the data
  - Many errors in the training and test corpora

  The average of interbank *offered* rates plummeted …
  - Probably about 2% guaranteed error from noise (on this data)
Overview: Accuracies

- **Roadmap of (known / unknown) accuracies:**
  - Most freq tag: \(~90\% / ~50\%\)
  - Trigram HMM: \(~95\% / ~55\%\)
  - TnT (HMM++): \(96.2\% / 86.0\%\)
  - Maxent P(t|w): \(93.7\% / 82.6\%\)
  - MEMM tagger: \(96.9\% / 86.9\%\)
  - State-of-the-art: \(97+\% / 89+\%\)
  - Upper bound: \(~98\%\)

Most errors on unknown words
Common Errors

- Common errors [from Toutanova & Manning 00]

<table>
<thead>
<tr>
<th>JJ</th>
<th>NN</th>
<th>NNP</th>
<th>NNPS</th>
<th>RB</th>
<th>RP</th>
<th>IN</th>
<th>VB</th>
<th>VBD</th>
<th>VBN</th>
<th>VBP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>177</td>
<td>56</td>
<td>0</td>
<td>61</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>108</td>
<td>0</td>
<td>488</td>
</tr>
<tr>
<td>244</td>
<td>0</td>
<td>103</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>29</td>
<td>5</td>
<td>6</td>
<td>19</td>
<td>525</td>
</tr>
<tr>
<td>107</td>
<td>106</td>
<td>0</td>
<td>132</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>427</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>142</td>
</tr>
<tr>
<td>72</td>
<td>21</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>138</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>295</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>39</td>
<td>0</td>
<td>65</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>104</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>169</td>
<td>103</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>323</td>
</tr>
<tr>
<td>17</td>
<td>64</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>85</td>
<td>189</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>143</td>
<td>2</td>
<td>166</td>
</tr>
<tr>
<td>101</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>108</td>
<td>0</td>
<td>1</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>49</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>626</td>
<td>536</td>
<td>348</td>
<td>144</td>
<td>317</td>
<td>122</td>
<td>279</td>
<td>102</td>
<td>140</td>
<td>269</td>
<td>108</td>
</tr>
</tbody>
</table>

NN/JJ
NN
official knowledge

NN

VBD
RP/IN
DT
NN

made up
the story

RB
VBD/VBN
NNS

recently sold shares
Richer Features
Better Features

- Can do surprisingly well just looking at a word by itself:
  - Word
  - Lowercased word
  - Prefixes
  - Suffixes
  - Capitalization
  - Word shapes

- Then build a maxent (or whatever) model to predict tag
  - Maxent $P(t|w)$: 93.7% / 82.6%
Why Linear Context is Useful

- Lots of rich local information!

  RB
  PRP VBD IN RB IN PRP VBD .
  They left as soon as he arrived .

  - We could fix this with a feature that looked at the next word

    JJ
    NNP NNS VBD VBN .
    Intrinsic flaws remained undetected .

  - We could fix this by linking capitalized words to their lowercase versions

- Solution: discriminative sequence models (MEMMs, CRFs)

- Reality check:
  - Taggers are already pretty good on newswire text...
  - What the world needs is taggers that work on other text!
Sequence-Free Tagging?

- What about looking at a word and its environment, but no sequence information?
  - Add in previous / next word the __
  - Previous / next word shapes X __ X
  - Occurrence pattern features [X: x X occurs]
  - Crude entity detection __ ..... (Inc.|Co.)
  - Phrasal verb in sentence? put ...... __
  - Conjunctions of these things

- All features except sequence: 96.6% / 86.8%
- Uses lots of features: > 200K
- Why isn’t this the standard approach?
Named Entity Recognition

- Other sequence tasks use similar models
- Example: name entity recognition (NER)

PER PER O O O O O O ORG O O O O O O LOC LOC O

Tim Boon has signed a contract extension with Leicestershire which will keep him at Grace Road.

Local Context

<table>
<thead>
<tr>
<th></th>
<th>Prev</th>
<th>Cur</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Other</td>
<td>???</td>
<td>???</td>
</tr>
<tr>
<td>Word</td>
<td>at</td>
<td>Grace</td>
<td>Road</td>
</tr>
<tr>
<td>Tag</td>
<td>IN</td>
<td>NNP</td>
<td>NNP</td>
</tr>
<tr>
<td>Sig</td>
<td>x</td>
<td>Xx</td>
<td>Xx</td>
</tr>
</tbody>
</table>
MEMM Taggers

- Idea: left-to-right local decisions, condition on previous tags and also entire input

\[ P(t|w) = \prod_{i} P_{ME}(t_i|w, t_{i-1}, t_{i-2}) \]

- Train up \( P(t_i|w, t_{i-1}, t_{i-2}) \) as a normal maxent model, then use to score sequences
- This is referred to as an MEMM tagger [Ratnaparkhi 96]
- Beam search effective! (Why?)
- What about beam size 1?

- Subtle issues with local normalization (cf. Lafferty et al 01)
### NER Features

Because of regularization term, the more common prefixes have larger weights even though entire-word features are more specific.

#### Local Context

<table>
<thead>
<tr>
<th>Prev</th>
<th>Cur</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Other</td>
<td>???</td>
</tr>
<tr>
<td>Word</td>
<td>at</td>
<td>Grace</td>
</tr>
<tr>
<td>Tag</td>
<td>IN</td>
<td>NNP</td>
</tr>
<tr>
<td>Sig</td>
<td>x</td>
<td>Xx</td>
</tr>
</tbody>
</table>

#### Feature Weights

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Feature</th>
<th>PERS</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous word</td>
<td><em>at</em></td>
<td>-0.73</td>
<td>0.94</td>
</tr>
<tr>
<td>Current word</td>
<td><em>Grace</em></td>
<td>0.03</td>
<td>0.00</td>
</tr>
<tr>
<td>Beginning bigram</td>
<td>&lt;G</td>
<td>0.45</td>
<td>-0.04</td>
</tr>
<tr>
<td>Current POS tag</td>
<td>NNP</td>
<td>0.47</td>
<td>0.45</td>
</tr>
<tr>
<td>Prev and cur tags</td>
<td>IN NNP</td>
<td>-0.10</td>
<td>0.14</td>
</tr>
<tr>
<td>Previous state</td>
<td>Other</td>
<td>-0.70</td>
<td>-0.92</td>
</tr>
<tr>
<td>Current signature</td>
<td>Xx</td>
<td>0.80</td>
<td>0.46</td>
</tr>
<tr>
<td>Prev state, cur sig</td>
<td>O-Xx</td>
<td>0.68</td>
<td>0.37</td>
</tr>
<tr>
<td>Prev-cur-next sig</td>
<td>x-Xx-Xx</td>
<td>-0.69</td>
<td>0.37</td>
</tr>
<tr>
<td>P. state - p-cur sig</td>
<td>O-x-Xx</td>
<td>-0.20</td>
<td>0.82</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>-0.58</td>
<td>2.68</td>
</tr>
</tbody>
</table>
Conditional Random Fields (and Friends)
Perceptron Taggers

- **Linear models:**
  \[ \text{score}(t|w) = \lambda^T f(t, w) \]

- ... that decompose along the sequence
  \[ = \lambda^T \sum_i f(t_i, t_{i-1}, w, i) \]

- ... allow us to predict with the Viterbi algorithm
  \[ t^* = \arg \max_t \text{score}(t|w) \]

- ... which means we can train with the perceptron algorithm (or related updates, like MIRA)

[Collins 01]
Conditional Random Fields

- Make a maxent model over entire taggings
  - MEMM
    \[ P(t|w) = \prod_i \frac{1}{Z(i)} \exp \left( \lambda^\top f(t_i, t_{i-1}, w, i) \right) \]
  - CRF
    \[ P(t|w) = \frac{1}{Z(w)} \exp \left( \lambda^\top f(t, w) \right) \]
    \[ = \frac{1}{Z(w)} \exp \left( \lambda^\top \sum_i f(t_i, t_{i-1}, w, i) \right) \]
    \[ = \frac{1}{Z(w)} \prod_i \phi_i(t_i, t_{i-1}) \]
CRFs

- Like any maxent model, derivative is:

\[
\frac{\partial L(\lambda)}{\partial \lambda} = \sum_k \left( f_k(t^k) - \sum_t P(t|w_k)f_k(t) \right)
\]

- So all we need is to be able to compute the expectation of each feature (for example the number of times the label pair DT-NN occurs, or the number of times NN-interest occurs) **under the model distribution**

- Critical quantity: counts of posterior marginals:

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

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

- How many (expected) times is word w tagged with s?

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

- How to compute that marginal?

\[
\alpha_i(s) = \sum_{s'} \phi_i(s', s) \alpha_{i-1}(s') \\
\beta_i(s) = \sum_{s'} \phi_{i+1}(s, s') \beta_{i+1}(s') \\
P(t_i = s \mid w) = \frac{\alpha_i(s) \beta_i(s)}{\alpha_N(\text{END})}
\]
[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 $\rightarrow$ NN : DT __  
  - VBD $\rightarrow$ 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 definitely not the most accurate: 96.6% / 82.0 %
## Learned Transformations

### What gets learned? [from Brill 95]

<table>
<thead>
<tr>
<th>#</th>
<th>From</th>
<th>To</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NN</td>
<td>VB</td>
<td>Previous tag is TO</td>
</tr>
<tr>
<td>2</td>
<td>VBP</td>
<td>VB</td>
<td>One of the previous three tags is MD</td>
</tr>
<tr>
<td>3</td>
<td>NN</td>
<td>VB</td>
<td>One of the previous two tags is MD</td>
</tr>
<tr>
<td>4</td>
<td>VB</td>
<td>NN</td>
<td>One of the previous two tags is DT</td>
</tr>
<tr>
<td>5</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous three tags is VBZ</td>
</tr>
<tr>
<td>6</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is PRP</td>
</tr>
<tr>
<td>7</td>
<td>VBN</td>
<td>VBD</td>
<td>Previous tag is NNP</td>
</tr>
<tr>
<td>8</td>
<td>VBD</td>
<td>VBN</td>
<td>Previous tag is VBD</td>
</tr>
<tr>
<td>9</td>
<td>VBP</td>
<td>VB</td>
<td>Previous tag is TO</td>
</tr>
<tr>
<td>10</td>
<td>POS</td>
<td>VBZ</td>
<td>Previous tag is PRP</td>
</tr>
<tr>
<td>11</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is NNS</td>
</tr>
<tr>
<td>12</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous three tags is VBP</td>
</tr>
<tr>
<td>13</td>
<td>IN</td>
<td>WDT</td>
<td>One of the next two tags is VB</td>
</tr>
<tr>
<td>14</td>
<td>VBD</td>
<td>VBN</td>
<td>One of the previous two tags is VB</td>
</tr>
<tr>
<td>15</td>
<td>VB</td>
<td>VBP</td>
<td>Previous tag is PRP</td>
</tr>
<tr>
<td>16</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is VBZ</td>
</tr>
<tr>
<td>17</td>
<td>IN</td>
<td>DT</td>
<td>Next tag is NN</td>
</tr>
<tr>
<td>18</td>
<td>JJ</td>
<td>NNP</td>
<td>Next tag is NNP</td>
</tr>
<tr>
<td>19</td>
<td>IN</td>
<td>WDT</td>
<td>Next tag is VBD</td>
</tr>
<tr>
<td>20</td>
<td>JJR</td>
<td>RBR</td>
<td>Next tag is JJ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>From</th>
<th>To</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NN</td>
<td>NNS</td>
<td>Has suffix -s</td>
</tr>
<tr>
<td>2</td>
<td>NN</td>
<td>CD</td>
<td>Has character .</td>
</tr>
<tr>
<td>3</td>
<td>NN</td>
<td>JJ</td>
<td>Has character -</td>
</tr>
<tr>
<td>4</td>
<td>NN</td>
<td>VBN</td>
<td>Has suffix -ed</td>
</tr>
<tr>
<td>5</td>
<td>NN</td>
<td>VBG</td>
<td>Has suffix -ing</td>
</tr>
<tr>
<td>6</td>
<td>??</td>
<td>RB</td>
<td>Has suffix -ly</td>
</tr>
<tr>
<td>7</td>
<td>??</td>
<td>JJ</td>
<td>Adding suffix -ly results in a word.</td>
</tr>
<tr>
<td>8</td>
<td>NN</td>
<td>CD</td>
<td>The word $ can appear to the left.</td>
</tr>
<tr>
<td>9</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -al</td>
</tr>
<tr>
<td>10</td>
<td>NN</td>
<td>VB</td>
<td>The word would can appear to the left.</td>
</tr>
<tr>
<td>11</td>
<td>NN</td>
<td>CD</td>
<td>Has character 0</td>
</tr>
<tr>
<td>12</td>
<td>NN</td>
<td>JJ</td>
<td>The word be can appear to the left.</td>
</tr>
<tr>
<td>13</td>
<td>NNS</td>
<td>JJ</td>
<td>Has suffix -us</td>
</tr>
<tr>
<td>14</td>
<td>NNS</td>
<td>VBZ</td>
<td>The word it can appear to the left.</td>
</tr>
<tr>
<td>15</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -ble</td>
</tr>
<tr>
<td>16</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -ic</td>
</tr>
<tr>
<td>17</td>
<td>NN</td>
<td>CD</td>
<td>Has character 1</td>
</tr>
<tr>
<td>18</td>
<td>NNS</td>
<td>NN</td>
<td>Has suffix -ss</td>
</tr>
<tr>
<td>19</td>
<td>??</td>
<td>JJ</td>
<td>Deleting the prefix un- results in a word</td>
</tr>
<tr>
<td>20</td>
<td>NN</td>
<td>JJ</td>
<td>Has suffix -ive</td>
</tr>
</tbody>
</table>
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
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!
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
<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>
The president said that the downturn was over.

- president
- the __ of
- governor
- the __ appointed
- said
- sources __
- said
- president __ that
- reported
- sources __

[Finch and Chater 92, Shuetze 93, many others]
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]
<table>
<thead>
<tr>
<th>word</th>
<th>nearest neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>accompanied</td>
<td>submitted banned financed developed authorized headed canceled awarded barred</td>
</tr>
<tr>
<td>almost</td>
<td>virtually merely formally fully quite officially just nearly only less</td>
</tr>
<tr>
<td>causing</td>
<td>reflecting forcing providing creating producing becoming carrying particularly</td>
</tr>
<tr>
<td>classes</td>
<td>elections courses payments losses computers performances violations levels pictures</td>
</tr>
<tr>
<td>directors</td>
<td>professionals investigations materials competitors agreements papers transactions</td>
</tr>
<tr>
<td>goal</td>
<td>mood roof eye image tool song pool scene gap voice</td>
</tr>
<tr>
<td>japanese</td>
<td>chinese iraqi american western arab foreign europe federal soviet indian</td>
</tr>
<tr>
<td>represent</td>
<td>reveal attend deliver reflect choose contain impose manage establish retain</td>
</tr>
<tr>
<td>think</td>
<td>believe wish know realize wonder assume feel say mean bet</td>
</tr>
<tr>
<td>york</td>
<td>angeles franco sox rouge kong diego zone vegas inning layer</td>
</tr>
<tr>
<td>on</td>
<td>through in at over into with from for by across</td>
</tr>
<tr>
<td>must</td>
<td>might would could cannot will should can may does helps</td>
</tr>
<tr>
<td>they</td>
<td>we you i he she nobody who it everybody there</td>
</tr>
</tbody>
</table>
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 ♦

♦ the president said that the downturn was over ♦