Lecture 6: POS Tagging
Dan Klein – UC Berkeley

Statistical NLP
Spring 2007

Parts-of-Speech (English)

One basic kind of linguistic structure: syntactic word classes

- **Open class (lexical)** words
- **Closed class (functional)** words

**Proper**
- Nouns
- Verbs
- Adjectives
- Conjunctions
- Pronouns

**Common**
- Determiners
- Prepositions
- Particles

**Main**
- Modals
- Adverbs

**Adverbs**
- WRB
- WP
- WDT
- TO
- RB
- PRP
- POS

**Nouns**
- NNS
- NN
- NNP
- NNPS

**Verbs**
- VBD
- VBZ
- VBP
- VB
- VBH

**Modal auxiliaries**
- MD

**Adjectives**
- JJ
- JJR
- JJ

**Prepositions**
- IN

**Determiners**
- DT

**Particles**

**Conjunctions**
- CC

**Pronouns**
- PRP

**Other**
- NNP
- NN
- VB

Part-of-Speech Ambiguity

- **Example**

Why POS Tagging?

- **Useful in and of itself**
  - Test-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

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?
Transitions and Emissions

Transitions

- Transitions $P(s|s')$ encode well-formed tag sequences
- In a bigram tagger, states = tags
- In a trigram tagger, states = tag pairs

Estimating Transitions

- Use standard smoothing methods to estimate transitions:
  
  $P(t_i|t_{i-1},t_{i-2}) = \lambda \hat{P}(t_i|t_{i-1},t_{i-2}) + (1-\lambda) \hat{P}(t_i)$

- Can get a lot fancier (e.g. KN smoothing), 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)

Estimating Emissions

- Emissions are trickier:
- Words we’ve never seen before
- Words which occur with tags we’ve never seen
- One option: break out the Good-Turning smoothing
- Issue: words aren’t black boxes:
  
  - Unknown words usually broken into word classes
  - Another option: decompose words into features and use a maxent model along with Bayes’ rule

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%

Disambiguation

- Given these two multinomials, we can score any word / tag sequence pair

$P(s,w) = \prod P(s_i|s_{i-1})P(w_i|s_i)$

- Emissions are trickier:
  - Words we’ve never seen before
  - Words which occur with tags we’ve never seen
  - One option: break out the Good-Turning smoothing
- Issue: words aren’t black boxes:
  
  - Unknown words usually broken into word classes
  - Another option: decompose words into features and use a maxent model along with Bayes’ rule

Fed raises interest rates 0.5 percent.

- NNP
- VBZ
- NN
- NNS
- CD
- NN

LogP = -23

- NNP
- VBZ
- NN
- NNS
- CD
- NN

LogP = -29

- NNP
- VBZ
- NN
- NNS
- CD
- NN

LogP = -27
Finding the Best Trajectory

- Too many trajectories (state sequences) to list
- Option 1: Beam Search
  - A beam is a set of partial hypotheses
  - Start with just the single empty trajectory
  - At each derivation step:
    - Consider all continuations of previous hypotheses
    - Discard most, keep top $k$, or those within a factor of the best, (or some combination)
- Beam search works relatively well in practice
  - ... but sometimes you want the optimal answer
  - ... and you need optimal answers to validate your beam search

The Viterbi Algorithm

- Dynamic program for computing
  \[ \delta_i(s) = \max_{x \in \mathcal{S}} P(x_{i-1}, s, x_i | w_{i-1}^{w_i}) \]
  - The score of a best path up to position $i$ ending in state $s$
  \[ \delta_i(s) = \begin{cases} 1 & \text{if } s = \ast \ast \ast \\ 0 & \text{otherwise} \end{cases} \]
  \[ \delta_i(s) = \max_x P(x | s') P(w | x') \delta_{i-1}(s') \]
  - Also store a traceback
\[ \psi_i(s) = \arg \max_x P(x | s') P(w | x') \delta_{i-1}(s') \]
- Memoized solution
- Iterative solution

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 (SOA is ~97.2%)
- 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%
  - Maxent $P(t|w)$: 93.7% / 82.6%
  - TnT (HMM++): 96.2% / 86.0%
  - MEMM tagger: 96.9% / 86.9%
  - Cyclic tagger: 97.2% / 89.0%
  - Upper bound: ~98%

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 (next class)
- Reality check:
  - Taggers are already pretty good on WSJ journal text ...
  - What the world needs is taggers that work on other text!
Common Errors

- Common errors [from Toutanova & Manning 00]

<table>
<thead>
<tr>
<th>JJ</th>
<th>NN</th>
<th>NN</th>
<th>NN</th>
<th>RB</th>
<th>IN</th>
<th>VB</th>
<th>VBD</th>
<th>VBN</th>
<th>VBP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>0</td>
<td>11</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>29</td>
<td>5</td>
<td>19</td>
<td>6</td>
<td>327</td>
</tr>
<tr>
<td>147</td>
<td>106</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>12</td>
<td>1</td>
<td>247</td>
</tr>
<tr>
<td>189</td>
<td>1</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>1</td>
</tr>
<tr>
<td>172</td>
<td>21</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>297</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>525</td>
</tr>
<tr>
<td>44</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>104</td>
</tr>
<tr>
<td>1</td>
<td>4</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>8</td>
</tr>
<tr>
<td>103</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>219</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>45</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>104</td>
</tr>
<tr>
<td>45</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>104</td>
</tr>
</tbody>
</table>

- NN/JJ
- Official knowledge
- Made up the story
- Recently sold shares

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

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

- Train up \( P(t|w, t_{i-1}, t_{i-2}) \) 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?

Features Templates

- We’ve been sloppy:
  - Features: \( <w_0=\text{future}, t_0=\text{JJ}> \)
  - Feature templates: \( <w_0, t_0> \)

- In maxent taggers:
  - Can now add edge feature templates:
    - \( <t_1, t_0> \)
    - \( <t_2, t_1, t_0> \)
  - Also, mixed feature templates:
    - \( <t_1, w_0, t_0> \)

Decoding

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

- Viterbi algorithm (HMMs):

\[ \delta_i(s) = \arg \max_{s'} P(s|s', f') P(w_{i-1}|s') \delta_{i-1}(s') \]

- Viterbi algorithm (Maxent):

\[ \delta_i(s) = \arg \max_{s'} P(s|s', w) \delta_{i-1}(s') \]

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 \( \rightarrow \) 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 not the most accurate: 96.6% / 82.0%
TBL Tagger II

- What gets learned? [from Brill 95]

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

CRF Taggers

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

- Differences tend not to be too important for POS tagging
- 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?

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)