Part-of-speech tagging (3)

Steve Renals
s.renals@ed.ac.uk

ICL — 26 October 2006
Recall: HMM PoS tagging

Viterbi decoding

Trigram PoS tagging

Summary
Bigram PoS tagger

\[
\hat{t}_1^N = \arg \max_{t_1^N} P(t_1^N | w_1^N)
\]

\[
\sim \prod_{i=1}^{N} P(w_i | t_i) P(t_i | t_{i-1})
\]
Bigram PoS tagger

\[ \hat{t}_1^N = \arg \max_{t_1^n} P(t_1^N | w_1^N) \]

\[ \sim \prod_{i=1}^{N} P(w_i | t_i) P(t_i | t_{i-1}) \]
Hidden Markov models

▶ Hidden Markov models (HMMs) are appropriate for situations where somethings are **observed** and some things are **hidden**
  ▶ Observations: words
  ▶ Hidden events: PoS tags
Hidden Markov models

- Hidden Markov models (HMMs) are appropriate for situations where somethings are **observed** and some things are **hidden**
  - Observations: words
  - Hidden events: PoS tags
- In an HMM hidden **states** model the hidden events which are thought of as generating the observed words
Hidden Markov models

- Hidden Markov models (HMMs) are appropriate for situations where somethings are **observed** and some things are **hidden**
  - Observations: words
  - Hidden events: PoS tags

- In an HMM hidden **states** model the hidden events which are thought of as generating the observed words

- An HMM is defined by:
  - A set of states \( t_i \)
  - Transition probabilities between the states
  - Observation likelihoods expressing the probability of an observation being generated from a hidden state
Hidden Markov models

- Hidden Markov models (HMMs) are appropriate for situations where somethings are observed and some things are hidden.
  - Observations: words
  - Hidden events: PoS tags

- In an HMM hidden states model the hidden events which are thought of as generating the observed words.

- An HMM is defined by:
  - A set of states \( t_i \)
  - Transition probabilities between the states
  - Observation likelihoods expressing the probability of an observation being generated from a hidden state

- Decoding: find the most likely state sequence to have generated the observation sequence.
Decoding

- Find the most likely sequence of tags given the observed sequence of words
- Exhaustive search (ie probability evaluation of each possible tag sequence) is very slow (not feasible)
Decoding

- Find the most likely sequence of tags given the observed sequence of words
- Exhaustive search (ie probability evaluation of each possible tag sequence) is very slow (not feasible)
- Use the Markov assumption
- Problem is that of finding the most probable path through a tag-word lattice
Decoding

- Find the most likely sequence of tags given the observed sequence of words
- Exhaustive search (i.e., probability evaluation of each possible tag sequence) is very slow (not feasible)
- Use the Markov assumption
- Problem is that of finding the most probable path through a tag-word lattice
- The solution is Viterbi decoding or dynamic programming
- Example: A (very) simplified subset of the POS tagging problem considering just 4 tag classes and 4 words (J&M, 2nd Ed, sec 5.5.3)
### Transition and observation probabilities

**Transition probabilities:** $P(t_i|t_{i-1})$

<table>
<thead>
<tr>
<th></th>
<th>VB</th>
<th>TO</th>
<th>NN</th>
<th>PPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>0.019</td>
<td>0.0043</td>
<td>0.041</td>
<td>0.067</td>
</tr>
<tr>
<td>VB</td>
<td>0.0038</td>
<td>0.0345</td>
<td>0.047</td>
<td>0.070</td>
</tr>
<tr>
<td>TO</td>
<td>0.83</td>
<td>0</td>
<td>0.00047</td>
<td>0</td>
</tr>
<tr>
<td>NN</td>
<td>0.0040</td>
<td>0.016</td>
<td>0.087</td>
<td>0.0045</td>
</tr>
<tr>
<td>PPSS</td>
<td>0.23</td>
<td>0.00079</td>
<td>0.0012</td>
<td>0.00014</td>
</tr>
</tbody>
</table>

**Observation likelihoods:** $P(w_i|t_i)$

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>want</th>
<th>to</th>
<th>race</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB</td>
<td>0</td>
<td>0.0093</td>
<td>0</td>
<td>0.00012</td>
</tr>
<tr>
<td>TO</td>
<td>0</td>
<td>0</td>
<td>0.99</td>
<td>0</td>
</tr>
<tr>
<td>NN</td>
<td>0</td>
<td>0.000054</td>
<td>0</td>
<td>0.00057</td>
</tr>
<tr>
<td>PPSS</td>
<td>0.37</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
HMM representation

![HMM Diagram]

P(w|NN)
- I: 0
- want: 0.000054
- to: 0
- race: 0.00057

Steve Renals s.renals@ed.ac.uk
Decoded HMM representation

```
start ─→ PPSS ─→ VB
  |       |       |
  I ───>  want
```
## Decoding

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NN</td>
<td>.041<em>1.0</em>0=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TO</td>
<td>.0042<em>1.0</em>0=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>VB</td>
<td>.019<em>1.0</em>0=0</td>
<td>MAX</td>
<td>.000051</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PPSS</td>
<td>.067<em>1.0</em>.37=.025</td>
<td></td>
<td>backtrace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>start</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>I</th>
<th>want</th>
<th>to</th>
<th>race</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ \text{MAX}(0*.0040, 0*.83, 0*.0038,.025*.23) = .025*.23 = .0055. \text{ Then } .0055*.0093 = .000051 \]
Viterbi decoding algorithm

1. Create path probability matrix $VITERBI(nstates+2, N+2)$
2. $VITERBI(0,0) = 1$ # start
3. foreach time step $t$ in $(1..N)$:
   ▶ foreach state $s$:
     ▶ $VITERBI(s,t) = \max_{s'} VITERBI(s',t-1)*p(s|s')*p(w(t)|s)$
     ▶ $\text{BackPointer}(s,t) = \arg \max_{s'} VITERBI(s',t-1)*p(s|s')$

In practice use log probabilities (and $*$ becomes $+$):
Local score $(t) = \log(p(w(t)|s))$
Global score $(0) = 1$
Global score $(t) = \text{Global score (t-1)} + \log p(s(t)|s(t-1)) + \text{local score(t)}$
TnT — A trigram POS tagger

- TnT — trigram-based tagger by Thorsten Brants (installed on DICE) (http://www.coli.uni-sb.de/ thorsten/tnt/)
- Based on the n-gram/HMM model described above, except that the tag sequence is modelled by trigrams
TnT — A trigram POS tagger

- TnT — trigram-based tagger by Thorsten Brants (installed on DICE) ([http://www.coli.uni-sb.de/~thorsten/tnt/](http://www.coli.uni-sb.de/~thorsten/tnt/))
- Based on the n-gram/HMM model described above, except that the tag sequence is modelled by trigrams
- n-grams are smoothed by interpolation
TnT — A trigram POS tagger

- TnT — trigram-based tagger by Thorsten Brants (installed on DICE) (http://www.coli.uni-sb.de/thorsten/tnt/)
- Based on the n-gram/HMM model described above, except that the tag sequence is modelled by trigrams
- n-grams are smoothed by interpolation
- Unknown words handled by an n-gram model over letters
TnT — A trigram POS tagger

- TnT — trigram-based tagger by Thorsten Brants (installed on DICE) (http://www.coli.uni-sb.de/thorsten/tnt/)
- Based on the n-gram/HMM model described above, except that the tag sequence is modelled by trigrams
- n-grams are smoothed by interpolation
- Unknown words handled by an n-gram model over letters
- Also models capitalization and has an efficient decoding algorithm (beam-search pruned Viterbi)
TnT — A trigram POS tagger

- TnT — trigram-based tagger by Thorsten Brants (installed on DICE) (http://www.coli.uni-sb.de/ thorsten/tnt/)
- Based on the n-gram/HMM model described above, except that the tag sequence is modelled by trigrams
- n-grams are smoothed by interpolation
- Unknown words handled by an n-gram model over letters
- Also models capitalization and has an efficient decoding algorithm (beam-search pruned Viterbi)
- Fast and accurate tagger — 96-97% accuracy on newspaper text (English or German)
The trigram model

\[ P(W_1^N | T_1^N)P(T_1^N) \sim \prod_{i=1}^{N+1} P(w_i | t_i)P(t_i | t_{i-2}, t_{i-1}) \]

- The most likely tag sequence \( t_1, \ldots, t_N \) is chosen to maximise the above expression
- \( t_0, t_{-1} \) and \( t_{n+1} \) are beginning- and end-of-sequence markers
The trigram model

\[ P(W_1^N \mid T_1^N)P(T_1^N) \sim \prod_{i=1}^{N+1} P(w_i \mid t_i)P(t_i \mid t_{i-2}, t_{i-1}) \]

- The most likely tag sequence \( t_1, \ldots, t_N \) is chosen to maximise the above expression
- \( t_0, t_{-1} \) and \( t_{n+1} \) are beginning- and end-of-sequence markers
- Probabilities estimated from relative frequency counts (maximum likelihood), eg:

\[ \hat{P}(t_3 \mid t_1, t_2) = \frac{c(t_1, t_2, t_3)}{c(t_1, t_2)} \]

- No discounting in TnT!
Smoothing

- Maximum likelihood estimation for trigrams results in many zero probabilities
- Interpolation-based smoothing:

\[ P(t_3|t_1, t_2) = \lambda_3 \hat{P}(t_3|t_1, t_2) + \lambda_2 \hat{P}(t_3|t_2) + \lambda_1 \hat{P}(t_3) \]

\[ \lambda_3 + \lambda_2 + \lambda_1 = 1 \]
Smoothing

- Maximum likelihood estimation for trigrams results in many zero probabilities
- Interpolation-based smoothing:
  \[ P(t_3 | t_1, t_2) = \lambda_3 \hat{P}(t_3 | t_1, t_2) + \lambda_2 \hat{P}(t_3 | t_2) + \lambda_1 \hat{P}(t_3) \]
  \[ \lambda_3 + \lambda_2 + \lambda_1 = 1 \]
- The \( \lambda \) coefficients are also estimated from the training data (deleted interpolation)
Dealing with new words

- Unknown words are calculated using a letter-based n-gram, using the last $m$ letters $l_i$ of an $L$-letter word:

$$P(t|l_{L-m+1}, \ldots, l_L).$$
Dealing with new words

- Unknown words are calculated using a letter-based n-gram, using the last $m$ letters $l_i$ of an $L$-letter word: $P(t|l_{L-m+1}, \ldots, l_L)$.
- Basic idea: suffixes of unknown words give a good clue to the POS of the word
- How big is $m$? - no bigger than 10, but it is based on the longest suffix found in the training set
- These probabilities also smoothed by interpolation
Summary

▶ Reading:
▶ Jurafsky and Martin, 2nd ed, sec 5.5
▶ Manning and Schütze, chapter 10;
  http://uk.arxiv.org/abs/cs.CL/0003055
▶ Viterbi decoding
▶ TnT — an accurate trigram-based tagger