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

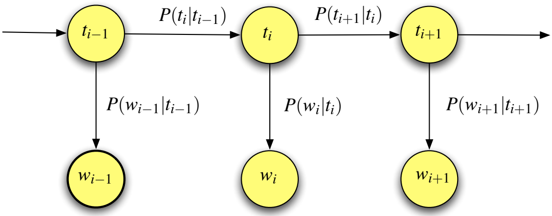
Summary

Bigram PoS tagger

$$\begin{aligned}\hat{t}_1^N &= \arg\max_{t_1^n} P(t_1^N | w_1^N) \\ &\sim \prod_{i=1}^{N} P(w_i | t_i) P(t_i | t_{i-1})\end{aligned}$$

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 sequebnce

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 (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
- 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

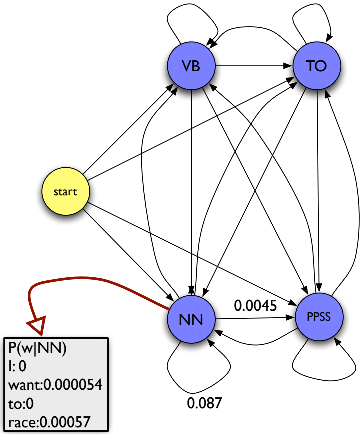
Transition probabilities: $P(t_i|t_{i-1})$

	VB	**TO**	**NN**	**PPSS**
start	0.019	0.0043	0.041	0.067
VB	0.0038	0.0345	0.047	0.070
TO	0.83	0	0.00047	0
NN	0.0040	0.016	0.087	0.0045
PPSS	0.23	0.00079	0.0012	0.00014

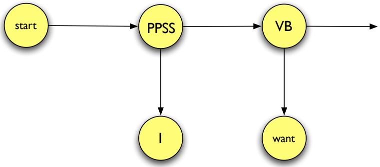
Observation likelihoods: $P(w_i|t_i)$

	I	**want**	**to**	**race**
VB	0	0.0093	0	0.00012
TO	0	0	0.99	0
NN	0	0.000054	0	0.00057
PPSS	0.37	0	0	0

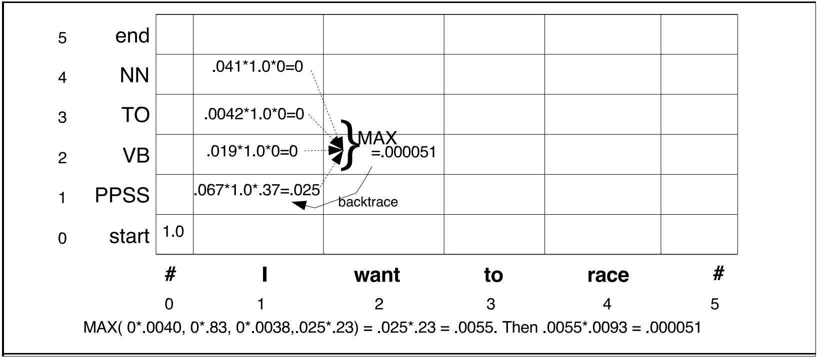
HMM representation



Decoded HMM representation



Decoding


5 end
4 NN
3 TO
2 VB
1 PPSS
0 start
1.0
.041*1.0*0=0
.0042*1.0*0=0
.019*1.0*0=0
.067*1.0*.37=.025
MAX
=.000051
backtrace
I want to race
0 1 2 3 4 5
MAX(0*.0040, 0*.83, 0*.0038,.025*.23) = .025*.23 = .0055. Then .0055*.0093 = .000051

Viterbi decoding algorithm

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

In practice use log probabilities (and * becomes +):
Local score (t) = log(p(w(t)|s))
Global score (0) = 1
Global score (t) = Global score (t-1) + log p(s(t)|s(t-1)) + 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/)
- 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

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

- 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
- Probabilities estimated from relative frequency counts (maximum likelihood), eg:

$$\hat{P}(t_3 | t_1, t_2) = \frac{c(t_1, t_2, t_3)}{c(t_1, t_2)}$$

- 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 λ 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;
  - T. Brants (2000). "TnT – a statistical part-of-speech tagger". In Proceedings of the 6th Applied NLP Conference, ANLP-2000. http://uk.arxiv.org/abs/cs.CL/0003055
- Viterbi decoding
- TnT — an accurate trigram-based tagger