Homework

• Read chapters 6 and 9

• Read Charniak, chapter 3 and 4

• Midterm next week
Hidden Markov Models

- States, transition probabilities, emission probabilities
- Forward algorithm efficiently computes $P(O|M)$
- Viterbi algorithm efficiently computes $\text{argmax}_S P(S|O,M)$
Parameter estimation

- Given an HMM with a fixed architecture, how do we estimate the probability distributions $A$ and $B$?

- If we have labeled training data, this is not any harder than estimating non-Hidden Markov Models (supervised training):

$$
\hat{A}(s|s') = \frac{C(s \rightarrow s')}{\sum_{s''} C(s \rightarrow s'')}
$$

$$
\hat{B}(o|s, s') = \frac{C(s \rightarrow s', o)}{C(s \rightarrow s')}
$$

- If we don’t have labeled training data, then it’s much harder (unsupervised training):

$$
\hat{M} = \text{argmax}_M P(O_{\text{training}}|M)
$$
Forward-Backward algorithm

- Also known as the *Baum-Welch algorithm*

- Instance of the *Expectation Maximization (EM) algorithm*:
  1. Choose a model at random
  2. E: Find the distribution of state sequences given the model
  3. M: Find the most likely model given those state sequences
  4. Go back to 2.

- Surprisingly enough, this actually works, though the result depends on the initial guess (local maxima)

- Jurafsky and Martin, Appendix D
Forward-Backward algorithm

- Our estimate of $A$ is:

$$\hat{A}(s|s') = \frac{E[C(s \rightarrow s')]}{E[C(s \rightarrow ?)]}$$

- We estimate $E[C(s \rightarrow s')]$ via $\tau_t(s, s')$, the probability of moving from state $s$ to state $s'$ at position $t$ given the output sequence $O$:

$$\tau_t(s, s') = \frac{P(s_t = s, s_{t+1} = s'|O, M)}{P(O|M)}$$

$$= \frac{P(s_t = s, s_{t+1} = s', O|M)}{P(O|M)}$$

$$= \frac{\alpha_s(t) A(s'|s) B(o_{t+1}|s, s') \beta_{s'}(t + 1)}{\sum_i \sum_j \alpha_i(t) A(j|i) B(o_{t+1}|i, j) \beta_j(t + 1)}$$
Forward-Backward algorithm

- This lets us estimate $A$:

$$\hat{A}(s|s') = \frac{\sum_t \tau_t(s, s')}{\sum_t \sum_{s''} \tau_t(s, s'')}$$

- We can estimate $B$ along the same lines, using $\sigma_t(o, s, s')$, the probability of emitting $o$ while moving from state $s$ to state $s'$ at position $t$ given the output sequence $O$

- Alternate re-estimating $\hat{A}$ and $\hat{B}$ from $\tau$, then $\tau$ from $\hat{A}$ and $\hat{B}$, until estimates stop changing

- If the initial guess is close to the right solution, this will converge to an optimal solution
HMM taggers

• If we make a few simplifying assumptions, Hidden Markov Models provide a tool part of speech tagging.

• From the training data, we need to collect probabilities for tag bigrams:
  \[ P(\text{tag}_2|\text{tag}_1) = \frac{\text{# of times tag}_1\text{tag}_2}{\text{# of times tag}_1} \]

• And we need:
  \[ P(\text{word}|\text{tag}) = \frac{\text{# of times word has tag}}{\text{# of occurrences of tag}} \]
HMM taggers

- Given a sentence $w_{1,n}$ we want to find the most probable sequence of tags $t_{1,n}$, so that we maximize $P(t_{1,n}|w_{1,n})$.

- By Bayes Rule, we have:

$$P(t_{1,n}|w_{1,n}) = \frac{P(w_{1,n}|t_{1,n}) P(t_{1,n})}{P(w_{1,n})}$$

And, since $P(w_{1,n})$ is the same for all tag sequences, we only need maximize the numerator.

- If we assume that $P(w_i|t_{1,n}) = P(w_i|t_i)$, then

$$P(t_{1,n}|w_{1,n}) = \frac{1}{P(w_{1,n})} \prod P(w_i|t_i) P(t_i|t_{i-1})$$
HMM taggers

- Markov models can be extended to take more context to account (bigrams, trigrams, etc.)

- Either Viterbi algorithm or forward-backward probabilities can be used to find the best sequence of tags for a sentence (rather than the best tag for a word)

- Taggers can equivalently work left to right or right to left

- Smoothing is vital: combinations of discounting, backoff, and specialized probability models work best
HMM taggers

- Since the tagger is a standard-issue HMM, all the technology developed for language models can be applied here

- Markov models can be extended to take more context to account (bigrams, trigrams, etc.)

- Either Viterbi algorithm or forward-backward probabilities can be used to find the best sequence of tags for a sentence (rather than the best tag for a word)

- Taggers can equivalently work left to right or right to left

- Smoothing is vital: combinations of discounting, backoff, and specialized probability models work best
HMM taggers

- Problems with Markov taggers:
  - unknown words
  - sparse data
  - long-distance relations
  - independence assumptions

- More sophisticated statistical models can help.
**HMM taggers**

- Weischedel, et al.’s (1993) “tri-tag” model used trigram probabilities:

  \[
P(t_1, \ldots, t_n) = P(t_1) \times P(t_2|t_1) \times \prod_{i=3}^{n} P(t_i|t_{i-1}t_{i-2})
  \]

- For known words, \(P(w_i|t_i)\) is estimated in the usual way. For unknown words:

  \[
P(w_i|t_i) = P(\text{unknown}|t_i) \times P(\text{capital}|t_i) \times P(\text{hyphen}|t_i) \times \prod_j P(\text{ending}_j|t_i)
  \]

- Works pretty well (85% unknown word accuracy), despite dubious independence assumptions
MaxEnt taggers

- Ratnaparkhi’s (1996) maximum entropy tagger uses a different distribution

- The conditional probability of assigning tag $t_i$ in context $c_i$ is:

$$P(t_i|c_i) = \frac{1}{Z(c_i)} \prod_j \alpha_j f_j(c_i, t_i)$$

- The functions $f_j(c_i, t_i)$ are binary features, e.g.:

$$f_{243}(c_i, t_i) = \begin{cases} 
1 & \text{if current word ends in -ing and } t_i = \text{VBG} \\
0 & \text{otherwise}
\end{cases}$$

- Parameters $\alpha_j$ are chosen to maximize the likelihood of the training data
MaxEnt taggers

- This is equivalent to the ‘usual’ MaxEnt parameterization:

\[
P(t_i | c_i) = \frac{1}{Z(c_i)} \prod_j \alpha_j^{f_j(c_i, t_i)}
\]

\[
= \frac{1}{Z(c_i)} \exp \left( \log \left( \prod_j \alpha_j^{f_j(c_i, t_i)} \right) \right)
\]

\[
= \frac{1}{Z(c_i)} \exp \left( \sum_j \log \left( \alpha_j^{f_j(c_i, t_i)} \right) \right)
\]

\[
= \frac{1}{Z(c_i)} \exp \left( \sum_j f_j(c_i, t_i) \log \alpha_j \right)
\]

\[
= \frac{1}{Z(c_i)} \exp \left( \sum_j \lambda_j f_j(c_i, t_i) \right)
\]
MaxEnt taggers

- Functions need not be conditionally independent
- Each function consists of the conjunction of a *contextual predicate* and a tag
- For all words, the previous two words, the next two words, and previous tag, and the previous tag bigram are contextual predicates
MaxEnt taggers

- Frequent words are themselves contextual predicates ($w_i = \text{the}$)

- For rare words (that appear fewer than 5 times), the first one, two, three, and four letter, the last one, two, three, and four letters, whether the word is capitalized, and whether it contains a hyphen are all contextual features

- After all the features are collected, ones that are too rare are pruned out
MaxEnt taggers

- Gives the effect of adaptive backoff models
- Combines disparate sources of information without requiring independence
- Can be extended to include information from tag dictionaries, morphological parses, etc.
- Contextual features can be arbitrarily complicated:

\[
f_{76}(c_i, t_i) = \begin{cases} 
1 & \text{if current word is } about, \text{ previous tag bigram is } DT \text{ NNS and } t_i = VBG \\
0 & \text{otherwise}
\end{cases}
\]
MaxEnt taggers

- Uses $k$-best breadth first search to find a near-optimal tag sequence
- Tag dictionary introduces structural zeroes, reduces search space, and improves results
- Overall performance is around 96.6%, as good or better than other supervised learning methods (transformation-based learning, decision trees, etc.)
- Feature selection is important (like smoothing for HMMs)
- Specialized features for hard words don’t improve things
MaxEnt taggers

- While features need not be independent for training, probability is still a product: features are treated as independent during model application.
- Dependencies, if relevant, can be expressed using complex features.
- Completely unseen events generally don’t occur, so nothing gets a zero probability.
- Sparse data is always a problem: ‘Curse of Dimensionality’.
MaxEnt taggers

- Maximum Entropy Markov Models (MEMMs) predict transition and emission probabilities $P(s|o, s')$ using a MaxEnt model for each state (very similar to Ratnaparkhi).

- MEMMs can be trained from unannotated data using Baum-Welch estimation with MaxEnt training as the ‘M’ step.

- In a non POS-tagging task (McCallum, et al. 2000):

<table>
<thead>
<tr>
<th></th>
<th>COAP</th>
<th>Prec</th>
<th>Rec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateless ME</td>
<td>0.526</td>
<td>0.038</td>
<td>0.362</td>
</tr>
<tr>
<td>Token HMM</td>
<td>0.865</td>
<td>0.276</td>
<td>0.140</td>
</tr>
<tr>
<td>Feature HMM</td>
<td>0.941</td>
<td>0.413</td>
<td>0.529</td>
</tr>
<tr>
<td>MEMM</td>
<td>0.965</td>
<td>0.867</td>
<td>0.681</td>
</tr>
</tbody>
</table>