CS 401R: Probabilistic Foundations of Machine Learning

Lecture #23: Hidden Markov Models (cont.), the Viterbi Algorithm

Some figures by Dan Klein of UC Berkeley. Rest by Eric Ringger.

Announcements

- Assignment #5.875
  - Due today
- Reading Report on HMMs
  - Due Monday
- Assignment #6
  - Early: Wednesday
  - Due: next Friday
- Documents or pixels?

Objectives

- Use Hidden Markov Models efficiently for POS tagging
- Understand the Viterbi algorithm
- If time permits, think about using HMM as a language model

Possible Scenarios

Training

- Labeled data*
- Unlabeled data
- Partially labeled data

Test / Run-time

- Unlabeled data*
- Partially labeled data

* We’re focusing on straight-forward supervised learning for today.

Disambiguation

- Tagging is disambiguation: Finding the best tag sequence
- Given a sentence $w$, we need to answer a conditional query:
  $P(t|w) = \frac{P(t,w)}{P(w)}$
- Thus,
  $P(t|w) \propto P(t,w)$
  $= \prod_{i=1}^{n} P(t_i|t_{i-1}) \cdot P(w_i|t_i)$

Disambiguation

- Given an HMM (i.e., distributions for transitions and emissions) and a word sequence, we can score any tag sequence for that sentence:
  - $P(t|w) = P(t,w)$
  - $= P(t_{\text{NNP}} \cdot P(\text{Fed}|\text{NNP}) \cdot P(\text{VBZ}|\text{NNP}) \cdot P(\text{raises}|\text{VBZ}) \cdot P(\text{NNP}|\text{VBZ}) \cdot P(\text{interest}|\text{NNP}) \cdot \ldots$
- What is the score?
- We need to find the tag sequence $\hat{t}$ that maximizes the score.
  - How would you do it?

Fed raises interest rates 0.5 percent.
Option 1: Enumeration
- Exhaustive enumeration: list all possible tag sequences, score each one, pick the best one (the “Viterbi” state sequence)

Fed raises interest rates ...

What’s wrong with this approach?
What to do about it?

Option 2: Breadth-First Search
- Exhaustive tree-structured search
  - Start with just the single empty tagged sentence
  - At each derivation step, consider all extensions of previous hypotheses

Option 3: Dynamic Programming
1. Ask: Am I solving an optimization problem?
2. Devise a minimal description (address) for any problem instance and sub-problem
3. Divide problems into sub-problems: define the recurrence to specify the relationship of problems to sub-problems
4. Check that the optimality property holds: An optimal solution to a problem is built from optimal solutions to sub-problems.
5. Store results – typically in a table – and re-use the solutions to sub-problems in the table as you build up to the overall solution.
6. Back-trace / analyze the table to extract the composition of the final solution.

Partial Taggings as Paths
- Step #2: Devise a minimal description (address) for any problem instance and sub-problem
  - At address ($i$, $j$): Optimal tagging of tokens 1 through $i$, ending in tag $j$

Partial taggings from position 1 through $i$ are represented as paths through a trellis of tag states, ending in some tag state.
Each arc $(i', i)$ connects two tag states and has a weight, which is the combination of $P(v_{i'} | v_i)$ and $P(v_i | v_{i'})$.

The Recurrence
- Step #3: Divide problems into sub-problems: define the recurrence to specify the relationship of problems to sub-problems
  - Score $s_i(j)$ of a best path (tagging) up to position $i$ ending in state $j$: $s_i(j) = \max_{\mathcal{s}_{i-1}} P_{\mathcal{s}_{i-1}}(j)P(v_i | v_{i-1})$
  - Step-by-step:
    - $s_0(j) = \{\text{all}\}$
    - $s_i(j) = \max P_{\mathcal{s}_{i-1}}(j)P(v_i | v_{i-1})$
  - Also store a back-pointer:
    - $\pi_i(j) = \arg\max P_{\mathcal{s}_{i-1}}(j)P(v_i | v_{i-1})$
  - Justification: the conditional independence assumptions in the model!
Optimality Property

- Step 04: Check that the optimality property holds: An optimal solution to a problem is built from optimal solutions to sub-problems.
- What does the optimality property have to say about computing the score of the best tagging ending in the highlighted state?
- The optimal tagging for \( w_{i-1} \) is constructed from optimal taggings for \( w_{i-2} \).

Iterative Algorithm

- Step 05: Store results – typically in a table – and re-use the solutions to sub-problems in the table as you build up to the overall solution.
- For each word position \( i \) and state \( t \), compute \( \delta_i(t) \) and \( \pi_i(t) \):
  
  \[
  \delta_i(t) = \max_{t' \in \mathcal{J}} P(w_i | t) P(t | t') \delta_{i-1}(t') \\
  \pi_i(t) = \arg \max_{t' \in \mathcal{J}} P(w_i | t) P(t | t') \delta_{i-1}(t')
  \]

Viterbi: DP Table

Start at max and backtrace
## Viterbi: DP Table

### Step #6: Back-trace / analyze the table to extract the composition of the final solution.

<table>
<thead>
<tr>
<th></th>
<th>Fed</th>
<th>raises</th>
<th>interest</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>VB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NNS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NNP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VBN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Start at max and backtrace

Read off the best sequence:

- Unit VBE MM

## Option 4: Beam Search

### Adopt a simple pruning strategy in the Viterbi trellis:

- A beam is a set of partial hypotheses:
  - Keep top k (use a priority queue)
  - Keep those within a factor of the best
  - Keep widest variety
  - or some combination of the above
- Is this OK?
- Beam search works well in practice
  - Will give you the optimal answer if the beam is wide enough
  - ... and you need optimal answers to validate your beam search

## Implementation Trick

- A constrained 1st order HMM over tag n-grams:

### Constraints:

\[ s_{k-1} \rightarrow \langle t, w \rangle \rightarrow \langle t, w \rangle \rightarrow \ldots \rightarrow \langle t, w \rangle \rightarrow s_k \]

## Efficiency

- \( O(n \cdot t \cdot t) = O(1) \)
- Linear!

## Implementation Trick

### For higher-order (>1) HMMs, we need to facilitate dynamic programming

- Allow scores to be computed locally without worrying about earlier structure of the trellis
- Define one state \( s_k \) for each tuple of \( n \) tags \( (t_{k-1}, t_k, \ldots, t_n) \)
- E.g., for order 2, \( s_k = (t_{k-1}, t_k) \)

\[
P(V, w) = P(V_1, t_1, t_2, \ldots, t_n) = \prod_{i=1}^{n} P(V_1, t_i|t_{i-1})
\]

\[
= P(V_1, t_1, t_2, \ldots, t_n)\prod_{i=1}^{n} P(V_1, t_i|t_{i-1})
\]

- Note:
- This corresponds to a constrained 1st order HMM over tag n-grams

## How Well Does It Work?

- Choose the most common tag:
  - 90.3% with a bad unknown word model
  - 93.7% with a good one!
- TrnT (Brants, 2000):
  - A carefully smoothed trigram tagger
  - 96.7% on WSJ text
- State-of-Art is approx. 97.3%
- 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)
HMMs: Basic Problems and Solutions

- **Decoding / Tagging:**
  - Choose optimal tag sequence $\tau$ for given $w$ using given HMM $\theta$.
  - Solution: Viterbi Algorithm

- **Evaluation / Computing the marginal probability:**
  - Compute $P_\theta(w)$, the marginal probability of $w$ using a given $\theta$.
  - Solution: Forward Algorithm
  - Just use a sum instead of max in the Viterbi algorithm!

- **Parameter estimation / Unsupervised or Semi-supervised Training:**
  - Estimate $\theta$ to maximize $P_\theta(w)$, the probability of an entire data set $w$ (either entirely or partially unlabeled).
  - Solution: Baum-Welch algorithm
  - Also known as the Forward-Backward algorithm.
  - Another example of EM

Extra

- **Back to tags $t$ for the moment (instead of states $s$).**
- We have a generative model of tagged sentences:
  $$P(L, w) = \prod_i P(t_i | t_{i-1}, t_{i-2}) P(w_i | t_i)$$

- We can turn this into a distribution over sentences by marginalizing out the tag sequences:
  $$P(w) = \sum_L P(L, w) = \sum_L \sum_i P(t_i | t_{i-1}, t_{i-2}) P(w_i | t_i)$$
- Problem: too many sequences!
- And beam search isn’t going to help this time. Why not?

How’s the HMM as a LM?

- POS tagging HMMs are terrible as LMs!
  - $\frac{\text{I bought an ice cream}}{\text{I bought an ice cream}}$
  - $\frac{\text{The computer that I set up yesterday just}}{\text{The computer that I set up yesterday just}}$
- Don’t capture long-distance effects like a parser could.
- Don’t capture local collocational effects like n-grams.
- But other HMM-like LMs can work very well.