Jelinek Summer Workshop: Summer School

Hidden Markov Models

Noah Smith, UW

Many slides from Dan Klein, Michael Collins, and Luke Zettlemoyer
Overview

- Hidden Markov Models
- Learning
  - Supervised: Maximum Likelihood
- Inference (or Decoding)
  - Viterbi
- N-gram Taggers
Consider the problem of jointly modeling a pair of strings
  - E.g.: part of speech tagging

DT   NNP   NN  VBD  VBN  RP   NN   NNS
The Georgia branch had taken on loan commitments …

DT   NN   IN   NN   VBD   NNS   VBD
The average of interbank offered rates plummeted …

Q: How do we map each word in the input sentence onto the appropriate label?
A: We can learn a joint distribution:

\[ p(x_1 \ldots x_n, y_1 \ldots y_n) \]

And then compute the most likely assignment:

\[ \arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n) \]
We want a model of sequences $y$ and observations $x$

\[ p(x_1 \ldots x_n, y_1 \ldots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i) \]

where $y_0=START$ and we call $q(y'|y)$ the transition distribution and $e(x|y)$ the emission (or observation) distribution.

Assumptions:
- 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?
Example: POS Tagging

The Georgia branch had taken on loan commitments ...

- HMM Model:
  - States $Y = \{DT, NNP, NN, \ldots \}$ are the POS tags
  - Observations $X$ are words
  - Transition dist'n $q(y_i | y_{i-1})$ models the tag sequences
  - Emission dist'n $e(x_i | y_i)$ models words given their POS

- Q: How do we represent n-gram POS taggers?
Example: Chunking

- **Goal:** Segment text into spans with certain properties
- **For example,** named entities: PER, ORG, and LOC

Germany's representative to the European Union's veterinary committee Werner Zwingman said on Wednesday consumers should…

```
[Germany]_LOC  's representative to the [European Union]_ORG  's veterinary committee [Werner Zwingman]_PER  said on Wednesday consumers should…
```

- **Q:** Is this a tagging problem?
Example: Chunking

[Germany]_{LOC} 's representative to the [European Union]_{ORG} 's veterinary committee [Werner Zwingman]_{PER} said on Wednesday consumers should…

Germany/BL 's/NA representative/NA to/NA the/NA European/BO Union/CO 's/NA veterinary/NA committee/NA Werner/BA BP Zwingman/CP said/NA on/NA Wednesday/NA consumers/NA should/NA…

- HMM Model:
  - States \( Y = \{NA, BL, CL, BO, CO, BP, CP\} \) represent beginnings (BL, BO, BP) and continuations (CL, CO, CP) of chunks, as well as other words (NA)
  - Observations \( X \) are words
  - Transition \( q(\mathbf{y}_i | \mathbf{y}_{i-1}) \) models the tag sequences
  - Emission \( e(x_i | \mathbf{y}_i) \) models words given their type
Example: HMM Translation Model

E: 1 Thank you 2 , 3 I shall 4 do 5 so 6 gladly 7 .

A: 1 → 3 → 7 → 6 → 8 → 8 → 8 → 8 → 8 → 9 →

F: Gracias 1 , 2 lo 3 haré de 4 muy 5 buen 6 grado 7 .

Model Parameters

Emissions: e(F₁ = Gracias | E₁ = Thank) Transitions: p(A₂ = 3 | A₁ = 1)
HMM Inference and Learning

- **Learning**
  - Maximum likelihood: transitions \( q \) and emissions \( e \)

\[
p(x_1 \ldots x_n, y_1 \ldots y_n) = q(STOP|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i)
\]

- **Inference (linear time in sentence length!)**
  - Viterbi:

\[
y^* = \arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)
\]

  - Forward Backward:

\[
p(x_1 \ldots x_n, y_i) = \sum_{y_1 \ldots y_{i-1}} \sum_{y_{i+1} \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)
\]
Learning: Maximum Likelihood

\[ p(x_1 \ldots x_n, y_1 \ldots y_n) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i) \]

- **Learning**
  - Maximum likelihood methods for estimating transitions \( q \) and emissions \( e \)

\[
q_{ML}(y_i|y_{i-1}) = \frac{c(y_{i-1}, y_i)}{c(y_{i-1})} \quad e_{ML}(x|y) = \frac{c(y, x)}{c(y)}
\]

- Will these estimates be high quality?
  - Which is likely to be more sparse, \( q \) or \( e \)?
- Can use all of the same smoothing tricks we saw for language models!
Learning: Low Frequency Words

\[ p(x_1 \ldots x_n, y_1 \ldots y_n) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1}) e(x_i|y_i) \]

- Typically, linear interpolation works well for transitions
  \[ q(y_i|y_{i-1}) = \lambda_1 q_{ML}(y_i|y_{i-1}) + \lambda_2 q_{ML}(y_i) \]

- However, other approaches used for emissions
  - **Step 1:** Split the vocabulary
    - *Frequent words:* appear more than M (often 5) times
    - *Low frequency:* everything else
  - **Step 2:** Map each low frequency word to one of a small, finite set of possibilities
    - For example, based on prefixes, suffixes, etc.
  - **Step 3:** Learn model for this new space of possible word sequences
Low Frequency Words: An Example

**Named Entity Recognition** [Bikel et. al, 1999]
- Used the following word classes for infrequent words:

<table>
<thead>
<tr>
<th>Word class</th>
<th>Example</th>
<th>Intuition</th>
</tr>
</thead>
<tbody>
<tr>
<td>twoDigitNum</td>
<td>90</td>
<td>Two digit year</td>
</tr>
<tr>
<td>fourDigitNum</td>
<td>1990</td>
<td>Four digit year</td>
</tr>
<tr>
<td>containsDigitAndAlpha</td>
<td>A8956-67</td>
<td>Product code</td>
</tr>
<tr>
<td>containsDigitAndDash</td>
<td>09-96</td>
<td>Date</td>
</tr>
<tr>
<td>containsDigitAndSlash</td>
<td>11/9/89</td>
<td>Date</td>
</tr>
<tr>
<td>containsDigitAndComma</td>
<td>23,000.00</td>
<td>Monetary amount</td>
</tr>
<tr>
<td>containsDigitAndPeriod</td>
<td>1.00</td>
<td>Monetary amount, percentage</td>
</tr>
<tr>
<td>othernum</td>
<td>456789</td>
<td>Other number</td>
</tr>
<tr>
<td>allCaps</td>
<td>BBN</td>
<td>Organization</td>
</tr>
<tr>
<td>capPeriod</td>
<td>M.</td>
<td>Person name initial</td>
</tr>
<tr>
<td>firstWord</td>
<td>first word of sentence</td>
<td>no useful capitalization information</td>
</tr>
<tr>
<td>initCap</td>
<td>Sally</td>
<td>Capitalized word</td>
</tr>
<tr>
<td>lowercase</td>
<td>can</td>
<td>Uncapitalized word</td>
</tr>
<tr>
<td>other</td>
<td>,</td>
<td>Punctuation marks, all other words</td>
</tr>
</tbody>
</table>
Low Frequency Words: An Example

- Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP Mulally/CP announced/NA first/NA quarter/NA results/NA ./NA

- firstword/NA soared/NA at/NA initCap/SC Co./CC ,/NA easily/NA lowercase/NA forecasts/NA on/NA initCap/SL Street/CL ,/NA as/NA their/NA CEO/NA Alan/SP initCap/CP announced/NA first/NA quarter/NA results/NA ./NA

NA = No entity
SC = Start Company
CC = Continue Company
SL = Start Location
CL = Continue Location
...

...
Inference (Decoding)

- Problem: find the most likely (Viterbi) sequence under the model
  \[ \arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n) \]

- Given model parameters, we can score any sequence pair

  Fed raises interest rates 0.5 percent.

  \[ q(\text{NNP}) e(\text{Fed}|\text{NNP}) q(\text{VBZ}|\text{NNP}) e(\text{raises}|\text{VBZ}) q(\text{NN}|\text{VBZ}) \ldots \]

- In principle, we're done – list all possible tag sequences, score each one, pick the best one (the Viterbi state sequence)

  NNP VBZ NN NNS CD NN \rightarrow \log P = -23
  NNP NNS NN NNS CD NN \rightarrow \log P = -29
  NNP VBZ VB NNS CD NN \rightarrow \log P = -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

- Beam search works ok in practice
  - … but sometimes you want the optimal answer
  - … and there's usually a better option than naïve beams
The State Lattice / Trellis

START       Fed           raises       interest       rates       STOP
Dynamic Programming!

\[
p(x_1 \ldots x_n, y_1 \ldots y_n) = q(\text{STOP}|y_n) \prod_{i=1}^{n} q(y_i|y_{i-1})e(x_i|y_i)
\]

- Define \(\pi(i,y_i)\) to be the max score of a sequence of length \(i\) ending in tag \(y_i\)

\[
\pi(i, y_i) = \max_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i)
\]

\[
= \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1}) \max_{y_1 \ldots y_{i-2}} p(x_1 \ldots x_{i-1}, y_1 \ldots y_{i-1})
\]

\[
= \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1}) \pi(i - 1, y_{i-1})
\]

- We now have an efficient algorithm. Start with \(i=0\) and work your way to the end of the sentence!
The Viterbi Algorithm

- Dynamic program for computing (for all $i$)
  \[
  \pi(i, y_i) = \max_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i)
  \]

- Iterative computation
  \[
  \pi(0, y_0) = \begin{cases} 
  1 & \text{if } y_0 == START \\
  0 & \text{otherwise}
  \end{cases}
  \]
  For $i = 1 \ldots n$:
  \[
  \pi(i, y_i) = \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \pi(i - 1, y_{i-1})
  \]

- Also, store back pointers
  \[
  bp(i, y_i) = \arg \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \pi(i - 1, y_{i-1})
  \]
The Viterbi Algorithm: Runtime

- Linear in sentence length $n$
- Polynomial in the number of possible tags $|K|$

\[
\pi(i, y_i) = \max_{y_{i-1}} e(x_i | y_i) q(y_i | y_{i-1}) \pi(i - 1, y_{i-1})
\]

- Specifically:

\[
O(n|K|) \text{ entries in } \pi(i, y_i)
\]

\[
O(|K|) \text{ time to compute each } \pi(i, y_i)
\]

- Total runtime: $O(n|K|^2)$

- **Q:** Is this a practical algorithm?
- **A:** depends on $|K|$. ...
What about n-gram Taggers?

- States encode what is relevant about the past
- Transitions $P(s|s')$ encode well-formed tag sequences
  - In a bigram tagger, states = tags
  - In a trigram tagger, states = tag pairs
The State Lattice / Trellis

START
Fed
raises
interest

\( e(Fed|N) \)
\( q(V|N) \)
\( q(V|N,D) \)
\( e(interest|V) \)