Dynamic Programming: Hidden Markov Models

Rebecca Dridan

16 October 2013

INF4820: Algorithms for AI and NLP

Topics

Recap

▶ n-grams
▶ Parts-of-speech
▶ Hidden Markov Models

Today

▶ Dynamic programming
  ▶ Viterbi algorithm
  ▶ Forward algorithm
▶ Tagger evaluation
Previous context can help predict the next thing in a sequence

Rather than use the whole previous context, the Markov assumption says that the whole history can be approximated by the last $n - 1$ elements

An $n$-gram language model predicts the $n$-th word, conditioned on the $n - 1$ previous words

Maximum Likelihood Estimation uses relative frequencies to approximate the conditional probabilities needed for an $n$-gram model

Smoothing alters the probabilities to save some probability mass for unseen events

---

Lexical categories defined by their distributional properties

These properties can be inherent to the lemma, or a function of a word’s use in a sentence

Open-class categories: new words frequently added, usually content-bearing words

Closed-class categories: more static, often function words

Distinctions made vary with any particular tag set

Correct tag is context dependent

POS tagging is a frequent pre-processing step for other NLP algorithms, but is also directly useful for text-to-speech systems, etc.
For a bi-gram HMM, with $O^N_1$:

$$P(S, O) = \prod_{i=1}^{N+1} P(s_i|s_{i-1})P(o_i|s_i)$$

where $s_0 = \langle S \rangle, s_{N+1} = \langle /S \rangle$

and $P(o_{N+1}|s_{N+1}) = 1$

- The transition probabilities model the probabilities of moving from state to state.
- The emission probabilities model the probability that a state emits a particular observation.

**Using HMMs**

The HMM models the process of generating the labelled sequence. We can use this model for a number of tasks:

- $P(S, O)$ given $S$ and $O$
- $P(O)$ given $O$
- $S$ that maximises $P(S|O)$ given $O$
- $P(s_x|O)$ given $O$
- We can also learn the model parameters, given a set of observations.
Missing records of weather in Baltimore for Summer 2007

- Jason likes to eat ice cream.
- He records his daily ice cream consumption in his diary.
- The number of ice creams he ate was influenced, but not entirely determined by the weather.
- Today’s weather is partially predictable from yesterday’s.

A Hidden Markov Model!

with:

- Hidden states: \( \{H, C\} \) (plus pseudo-states \( \langle S\rangle \) and \( \langle /S\rangle \))
- Observations: \( \{1, 2, 3\} \)
The HMM models the process of generating the labelled sequence. We can use this model for a number of tasks:

- \( P(S, O) \) given \( S \) and \( O \)
- \( P(O) \) given \( O \)
- \( S \) that maximises \( P(S|O) \) given \( O \)
- \( P(s_x|O) \) given \( O \)
- We can also learn the model parameters, given a set of observations.

### Part-of-Speech Tagging

We want to find the tag sequence, given a word sequence. With tags as our states and words as our observations, we know:

\[
P(S, O) = \prod_{i=1}^{N+1} P(s_i|s_{i-1})P(o_i|s_i)
\]

We want: \( P(S|O) = \frac{P(S, O)}{P(O)} \)

Actually, we want the state sequence that maximises \( P(S|O) \):

\[
S_{\text{best}} = \arg \max_S \frac{P(S, O)}{P(O)}
\]

Since \( P(O) \) will be the same, we can drop the denominator.
**Task**

What is the most likely state sequence $S$, given an observation sequence $O$ and an HMM.

**HMM**

| $P(H|S)$ | $P(C|S)$ | if $O = 3 \ 1 \ 3$ |
|------|------|------------------|
| $P(H|H) = 0.6$ | $P(C|H) = 0.2$ | $P(H|⟨S⟩) = 0.8$ | $P(C|⟨S⟩) = 0.2$ |
| $P(H|C) = 0.3$ | $P(C|C) = 0.5$ | $P(⟨/S⟩|H) = 0.2$ | $P(⟨/S⟩|C) = 0.2$ |
| $P(1|H) = 0.2$ | $P(1|C) = 0.5$ | $⟨S⟩$ | $H$ | $H$ | $H$ | $⟨/S⟩$ | $0.0018432$ |
| $P(2|H) = 0.4$ | $P(2|C) = 0.4$ | $⟨S⟩$ | $H$ | $H$ | $C$ | $⟨/S⟩$ | $0.0001536$ |
| $P(3|H) = 0.4$ | $P(3|C) = 0.1$ | $⟨S⟩$ | $H$ | $C$ | $H$ | $⟨/S⟩$ | $0.0007680$ |
| $P(1|C) = 0.5$ | $P(1|C) = 0.5$ | $⟨S⟩$ | $C$ | $H$ | $H$ | $⟨/S⟩$ | $0.0000576$ |
| $P(2|C) = 0.4$ | $P(2|C) = 0.4$ | $⟨S⟩$ | $C$ | $C$ | $C$ | $⟨/S⟩$ | $0.0000048$ |
| $P(3|C) = 0.1$ | $P(3|C) = 0.1$ | $⟨S⟩$ | $C$ | $C$ | $H$ | $⟨/S⟩$ | $0.0001200$ |
| $P(1|C) = 0.5$ | $P(1|C) = 0.5$ | $⟨S⟩$ | $C$ | $C$ | $C$ | $⟨/S⟩$ | $0.0000500$ |

**Dynamic Programming**

For 2 states and an observation sequence of 3, this is OK, but . . .

- for $N$ observations and $L$ states, there are $L^N$ sequences
- we do the same calculations over and over again

Enter **dynamic programming**:

- records sub-problem solutions for further re-use
- useful when a complex problem can be described recursively
- examples: Dijkstra’s shortest path, minimum edit distance, longest common subsequence, **Viterbi**
Recall our problem:

$$\max \ P(s_1 \ldots s_n|o_1 \ldots o_n) = P(s_1|s_0)P(o_1|s_1)P(s_2|s_1)P(o_2|s_2) \ldots$$

Our recursive sub-problem:

$$v_i(s_i = x) = \max_{k=1}^L [v_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x)]$$

The variable $v_i(x)$ represents the maximum probability that the $i$-th state is $x$, given that we have seen $O_i$.

At each step, we record backpointers showing which previous state led to the maximum probability.

An example of the Viterbi algorithm:

$$v_1(H) = 0.32$$

$$v_2(H) = \max(0.32 \cdot 0.12 \cdot 0.6, 0.02 \cdot 0.06) = 0.0384$$

$$v_3(H) = \max(0.0384 \cdot 0.24, 0.032 \cdot 0.12) = 0.009216$$

$$v_f(\langle /S \rangle) = \max(0.009216 \cdot 0.2, 0.0016 \cdot 0.2) = 0.0018432$$

$$v_1(C) = 0.02$$

$$v_2(C) = \max(0.32 \cdot 0.02 \cdot 0.25) = 0.032$$

$$v_3(C) = \max(0.0384 \cdot 0.02, 0.032 \cdot 0.05) = 0.0016$$
Pseudocode for the Viterbi algorithm

Input: observations of length $N$, state set of length $L$
Output: best-path
create a path probability matrix $viterbi[N,L + 2]$
create a path backpointer matrix $backpointer[N,L + 2]$

foreach state $s$ from 1 to $L$ do
  $viterbi[1,s] \leftarrow trans(⟨S⟩,s) \times emit(o_1,s)$
  $backpointer[1,s] \leftarrow 0$
end

foreach time step $i$ from 2 to $N$ do
  foreach state $s$ from 1 to $L$ do
    $viterbi[i,s] \leftarrow \max_{s′=1}^L viterbi[i−1,s′] \times trans(s′,s) \times emit(o_i,s)$
    $backpointer[i,s] \leftarrow \text{arg max}_{s′=1}^L viterbi[i−1,s′] \times trans(s′,s)$
  end
end

$viterbi[N,L + 1] \leftarrow \max_{s=1}^L viterbi[s,N] \times trans(s,⟨/S⟩)$
$backpointer[N,L + 1] \leftarrow \text{arg max}_{s=1}^L viterbi[N,s] \times trans(s,⟨/S⟩)$
return the path by following backpointers from $backpointer[N,L + 1]$

Diversion: Complexity and $O(N)$

Big O notation describes the complexity of an algorithm.

- it describes the worst-case order of growth in terms of the size of the input
- only the largest order term is represented
- constant factors are ignored
- determined by looking at loops in the code
Pseudocode for the Viterbi algorithm

Input: observations of length $N$, state set of length $L$
Output: best-path
create a path probability matrix $viterbi[N, L + 2]$
create a path backpointer matrix $backpointer[N, L + 2]$

\begin{verbatim}
foreach state $s$ from 1 to $L$ do
  $viterbi[1, s] \leftarrow trans(⟨S⟩, s) \times emit(o_1, s)$
  $backpointer[1, s] \leftarrow 0$
end

foreach time step $i$ from 2 to $N$ do
  foreach state $s$ from 1 to $L$ do
    $viterbi[i, s] \leftarrow \max_{s'=1}^{L} viterbi[i - 1, s'] \times trans(s', s) \times emit(o_i, s)$
    $backpointer[i, s] \leftarrow \arg \max_{s'=1}^{L} viterbi[i - 1, s'] \times trans(s', s)$
  end
end
$viterbi[N, L + 1] \leftarrow \max_{s=1}^{L} viterbi[s, N] \times trans(s, ⟨/S⟩)$
$backpointer[N, L + 1] \leftarrow \arg \max_{s=1}^{L} viterbi[N, s] \times trans(s, ⟨/S⟩)$
return the path by following backpointers from $backpointer[N, L + 1]$ $N$
\end{verbatim}

$O(L^2N)$

Using HMMs

The HMM models the process of generating the labelled sequence. We can use this model for a number of tasks:

- $P(S, O)$ given $S$ and $O$
- $P(O)$ given $O$
- $S$ that maximises $P(S|O)$ given $O$
- $P(s_x|O)$ given $O$
- We can also learn the model parameters, given a set of observations.
Computing likelihoods

**Task**

Given an observation sequence $O$, determine the likelihood $P(O)$, according to the HMM.

Compute the **sum over all possible state sequences**:

$$P(O) = \sum_{S} P(O, S)$$

For example, the ice cream sequence 3 1 3:

$$P(3 \ 1 \ 3) = P(3 \ 1 \ 3, \text{cold cold cold}) + P(3 \ 1 \ 3, \text{cold cold hot}) + P(3 \ 1 \ 3, \text{hot hot cold}) + \ldots \Rightarrow O(L^3 N)$$

**The Forward algorithm**

Again, we use dynamic programming—storing and reusing the results of partial computations in a trellis $\alpha$.

Each cell in the trellis stores the probability of being in state $s_x$ after seeing the first $i$ observations:

$$\alpha_i(x) = P(o_1 \ldots o_i, s_i = x) = \sum_{k=1}^{L} \alpha_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x)$$

Note $\sum$, instead of the max in Viterbi.
An example of the Forward algorithm

\[
\begin{align*}
&v_1(H) = 0.32 \\
&v_1(C) = 0.02 \\
&v_2(H) = \sum(0.32 \times 0.2 \times 0.6) = 0.0396 \\
&v_2(C) = \sum(0.32 \times 0.1 \times 0.25) = 0.037 \\
&v_3(H) = \sum(0.0396 \times 0.4 \times 0.2) = 0.013944 \\
&v_3(C) = \sum(0.0396 \times 0.5 \times 0.05) = 0.002642
\end{align*}
\]

\[
P(313) = 0.0033172
\]

\[P(S) = \sum_{S=1}^{L} forward[N, s] \times trans(s, \langle/S\rangle)
\]

**Pseudocode for the Forward algorithm**

**Input:** observations of length N, state set of length L  
**Output:** forward-probability

create a probability matrix \textit{forward}\[N, L + 2\]

\begin{verbatim}
foreach state \textit{s} from 1 to \textit{L} do
    forward[1, \textit{s}] ← \textit{trans}(\langle S\rangle, \textit{s}) × \textit{emit}(\textit{o}_1, \textit{s})
end

foreach time step \textit{i} from 2 to \textit{N} do
    foreach state \textit{s} from 1 to \textit{L} do
        forward[\textit{i}, \textit{s}] ← \sum_{\textit{s}'=1}^{\textit{L}} forward[\textit{i} - 1, \textit{s}'] × \textit{trans}(\textit{s}', \textit{s}) × \textit{emit}(\textit{o}_\textit{i}, \textit{s})
    end
end

forward[\textit{N}, \textit{L} + 1] ← \sum_{\textit{s}=1}^{\textit{L}} forward[\textit{N}, \textit{s}] × \textit{trans}(\textit{\langle S\rangle})
return forward[\textit{N}, \textit{L} + 1]
\end{verbatim}
To evaluate a part-of-speech tagger (or any classification system) we:

- train on a labelled training set
- test on a separate test set

For a POS tagger, the standard evaluation metric is tag accuracy:

\[ Acc = \frac{\text{number of correct tags}}{\text{number of words}} \]

The other metric sometimes used is error rate:

\[ \text{error rate} = 1 - Acc \]

Summary

Understand
- Why does dynamic programming save time, and what type of problems can it be used for?
- What is the complexity of the Viterbi algorithm?

Coming Up
- Context free grammars
- Most likely trees