Viterbi Algorithm in Numpy

CS114B Lab 7

Kenneth Lai

March 19, 2021

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?

Hidden Markov Models

- Generative approach: Hidden Markov Models

Hidden Markov Models

- Generative approach: Hidden Markov Models
  - $P(Y|X) \propto P(X, Y) = P(X|Y)P(Y)$

Hidden Markov Models

- Generative approach: Hidden Markov Models
  - $P(Y|X) \propto P(X, Y) = P(X|Y)P(Y)$
- Independence Assumptions

Hidden Markov Models

- Generative approach: Hidden Markov Models
  - $P(Y|X) \propto P(X, Y) = P(X|Y)P(Y)$
- Independence Assumptions
  - (First-order) Markov Assumption: the probability of a tag depends only on the previous tag
    - $P(Y) = \prod_{i=1}^{T} P(y_i|y_{i-1})$

Hidden Markov Models

- Generative approach: Hidden Markov Models
  - $P(Y|X) \propto P(X,Y) = P(X|Y)P(Y)$
- Independence Assumptions
  - (First-order) Markov Assumption: the probability of a tag depends only on the previous tag
    - $P(Y) = \prod_{i=1}^{T} P(y_i|y_{i-1})$
  - Output Independence: the probability of a word at time i depends only on the tag at time i
    - $P(X|Y) = \prod_{i=1}^{T} P(x_i|y_i)$

Hidden Markov Models

$$P(Y|X) \propto \prod_{i=1}^{T} P(x_i|y_i) \times \prod_{i=1}^{T} P(y_i|y_{i-1})$$

Hidden Markov Models

$$P(Y|X) \propto \prod_{i=1}^{T} P(x_i|y_i) \times \prod_{i=1}^{T} P(y_i|y_{i-1})$$
$$\propto \prod_{i=1}^{T} \Big(P(x_i|y_i) \times P(y_i|y_{i-1}) \Big)$$

Hidden Markov Models

$$
\begin{aligned}
P(Y|X) &\propto \prod_{i=1}^{T} P(x_i|y_i) \times \prod_{i=1}^{T} P(y_i|y_{i-1}) \\
&\propto \prod_{i=1}^{T} \Big(P(x_i|y_i) \times P(y_i|y_{i-1}) \Big) \\
\log P(Y|X) &\propto \sum_{i=1}^{T} \Big(\log P(x_i|y_i) + \log P(y_i|y_{i-1}) \Big)
\end{aligned}
$$

Hidden Markov Models

$$P(Y|X) \propto \prod_{i=1}^{T} P(x_i|y_i) \times \prod_{i=1}^{T} P(y_i|y_{i-1})$$

$$\propto \prod_{i=1}^{T} \Big(P(x_i|y_i) \times P(y_i|y_{i-1}) \Big)$$

$$\log P(Y|X) \propto \sum_{i=1}^{T} \Big(\log P(x_i|y_i) + \log P(y_i|y_{i-1}) \Big)$$

- In other words, (log) $P(Y|X)$ decomposes into a product (or sum) of local parts

Hidden Markov Models

$$
\begin{aligned}
P(Y|X) &\propto \prod_{i=1}^{T} P(x_i|y_i) \times \prod_{i=1}^{T} P(y_i|y_{i-1}) \\
&\propto \prod_{i=1}^{T} \Big(P(x_i|y_i) \times P(y_i|y_{i-1}) \Big) \\
\log P(Y|X) &\propto \sum_{i=1}^{T} \Big(\log P(x_i|y_i) + \log P(y_i|y_{i-1}) \Big)
\end{aligned}
$$

- In other words, (log) $P(Y|X)$ decomposes into a product (or sum) of local parts
- This allows us to use dynamic programming

Viterbi Algorithm

- Not just for HMMs!

Viterbi Algorithm

- Not just for HMMs!
- Discriminative approaches:
  - Conditional random fields
  - Structured perceptrons

Viterbi Algorithm

- Not just for HMMs!
- Discriminative approaches:
  - Conditional random fields
  - Structured perceptrons
- As long as the “score” decomposes into a sum of local parts, we can use the Viterbi algorithm

Viterbi Algorithm

function VITERBI(*observations* of len *T*,*state-graph* of len *N*) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state *s* **from** 1 **to** *N* **do** ; initialization step
  $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step *t* **from** 2 **to** *T* **do** ; recursion step
  for each state *s* **from** 1 **to** *N* **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure 8.10 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Viterbi Algorithm

- Input: observations of length T

Viterbi Algorithm

- Input: observations of length T
- Also let N be the number of states and V be the size of the vocabulary

Viterbi Algorithm

- Input: observations of length T
- Also let N be the number of states and V be the size of the vocabulary

- Three Numpy arrays:
  - `self.initial` (π: shape $(N,)$)
  - `self.transition` ($\mathbf{A}$: shape (N, N))
  - `self.emission` ($\mathbf{B}$: shape (V, N))

Viterbi Algorithm

- Input: observations of length T
- Also let N be the number of states and V be the size of the vocabulary

- Three Numpy arrays:
  - `self.initial` (π: shape $(N,)$)
  - `self.transition` ($\mathbf{A}$: shape (N, N))
  - `self.emission` ($\mathbf{B}$: shape (V, N))
- We assume you know how to fill them in
  - HMM: CS114A
  - Structured perceptron: next week

Viterbi Algorithm

- Input: observations of length T
- Also let N be the number of states and V be the size of the vocabulary

- Three Numpy arrays:
  - `self.initial` (π: shape $(N,)$)
  - `self.transition` ($\mathbf{A}$: shape (N, N))
  - `self.emission` ($\mathbf{B}$: shape (V, N))
- We assume you know how to fill them in
  - HMM: CS114A
  - Structured perceptron: next week

- Create two Numpy arrays: (both with shape (N, T))
  - `v` (for `viterbi`)
  - `backpointer`

Initialization Step

- For each state s from 1 to N do
  $viterbi[s, 1] \leftarrow \pi_s \times b_s(o_1)$

Initialization Step

- For each state s from 1 to N do
  $viterbi[s, 1] \leftarrow \pi_s \times b_s(o_1)$

- What does this look like in code?

Initialization Step

- For each state s from 1 to N do
  $viterbi[s,1] \leftarrow \pi_s \times b_s(o_1)$
- What does this look like in code?
- How can we make this more efficient?

Initialization Step

- $\mathit{viterbi}[:, 0] \leftarrow \pi + \mathbf{b}(o_0)$

Initialization Step

- $\textit{viterbi}[:,0] \leftarrow \pi + \mathbf{b}(o_0)$
- For `backpointer`, do nothing!

Recursion Step

- For each time step t from 2 to T do
  For each state s from 1 to N do
  $$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] \times a_{s',s} \times b_s(o_t)$$

Recursion Step

- For each time step t from 2 to T do
  For each state s from 1 to N do
  $$viterbi[s, t] \leftarrow \max_{s'=1}^{N} viterbi[s', t-1] \times a_{s',s} \times b_s(o_t)$$
- What does this look like in code?

Recursion Step

- For each time step t from 2 to T do
  For each state s from 1 to N do
  $$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s', t-1] \times a_{s',s} \times b_s(o_t)$$
- What does this look like in code?
- How can we make this more efficient?

Recursion Step

- For each time step t do
  $$viterbi[:, t] \leftarrow \max_{s'} \Big(viterbi[:, t-1:t] + \mathbf{A} + \mathbf{b}(o_t) \Big)$$

Recursion Step

- For each time step t do
$$viterbi[:, t] \leftarrow \max_{s'} \left(viterbi[:, t-1:t] + \mathbf{A} + \mathbf{b}(o_t) \right)$$
- For `backpointer`, take the argmax instead of the max

Termination Step

- Best last tag is the argmax of the last column of v

Termination Step

- Best last tag is the argmax of the last column of `v`
- Follow backpointers in `backpointer`

Termination Step

- Best last tag is the argmax of the last column of `v`
- Follow backpointers in `backpointer`
  - Nothing fancy; we'll let you figure it out on your own

Termination Step

- Best last tag is the argmax of the last column of `v`
- Follow backpointers in `backpointer`
  - Nothing fancy; we'll let you figure it out on your own
  - For HW/PA, you do not have to return the path (log-)probability/score, just the backtrace path