Structured Perceptrons

CS114B Lab 7

Kenneth Lai

March 10, 2023

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
  - $P(Y|X) \propto P(X, Y) = P(X|Y)P(Y)$

Hidden Markov Models

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

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

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
- This allows us to use dynamic programming

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
- 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))
- We assume you know how to fill them in
  - HMM: CS114A
  - Structured perceptron: in a few slides

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: in a few slides

- For each sentence, create two Numpy arrays: (both with shape (N, T))
  - `v` (for `viterbi`)
  - `backpointer`

Initialization Step

- For each state s from 1 to N do
  $viterbi[s, 1] \leftarrow \pi_s + b_s(o_1)$
- Note the use of $+$ instead of $\times$
- We'll see how to do this without the for loop in a bit

Recursion Step

- For each time step t from 2 to T do
  For each state s from 1 to N do
  $$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s', t-1] + a_{s',s} + b_s(o_t)$$
- Note the use of $+$ instead of $\times$
- We'll see how to do this without the (inner) for loop in a bit

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 HW4, you do not have to return the path (log-)probability/score, just the backtrace path

Structured Perceptrons

- Perceptrons for sequence labeling
  - Given a sequence $X = [x_1, \ldots, x_T]$, predict labels $Y = [y_1, \ldots, y_T]$
  - We do not care about the probability $P(Y|X)$, just which Y has the highest score Z

Structured Perceptrons

- Perceptrons for sequence labeling
  - Given a sequence $X = [x_1, \ldots, x_T]$, predict labels $Y = [y_1, \ldots, y_T]$
  - We do not care about the probability $P(Y|X)$, just which Y has the highest score Z
- $\hat{Y} = \operatorname*{argmax}_{k \in K^T} Z_k$

Structured Perceptrons

- Score Z decomposes into a sum of local parts
  - At each time step i, for each possible combination of current tag y_i and previous tag y_{i-1}, compute a local score $z(y_i, y_{i-1})$

Structured Perceptrons

- Score Z decomposes into a sum of local parts
  - At each time step i, for each possible combination of current tag y_i and previous tag y_{i-1}, compute a local score $z(y_i, y_{i-1})$
  - Use the Viterbi algorithm to combine the local scores across the sequence, and find the argmax

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol `<S>`
  - Current word x_i

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol <S>
  - Current word x_i
- For simplicity, we will assume that these are the only features, and we will ignore the bias term

Structured Perceptrons

- Suppose that at each time step i, we want to predict the current tag y_i using the following features:
  - Previous tag y_{i-1}
    - At the beginning of the sentence, let the previous tag y_{i-1} be the start symbol `<S>`
  - Current word x_i
- For simplicity, we will assume that these are the only features, and we will ignore the bias term
- Let $\mathbf{f}(X, y_i, y_{i-1}, i)$ be the feature vector at time step i
  - Using $\mathbf{f}$ instead of $\mathbf{x}$, because features can include more than just the input

Structured Perceptrons

- We can arrange our weight matrix $\mathbf{\Theta}$ as follows:

$$\begin{bmatrix} \rule{3cm}{0.4pt} \\ \vdots \\ \rule{3cm}{0.4pt} \\ \vdots \end{bmatrix}$$

Structured Perceptrons

- We can arrange our weight matrix $\boldsymbol{\Theta}$ as follows:

$$\left[\begin{array}{c} \pi \\ \hline \\ \\ \\ \hline \\ \\ \\ \end{array}\right]$$

- Initial features
  - $y_{i-1} = \text{<S>}, y_i = \ldots$

Structured Perceptrons

- We can arrange our weight matrix $\boldsymbol{\Theta}$ as follows:

$$\begin{bmatrix} \pi \\ \hline \\ \mathbf{A} \\ \\ \hline \\ \\ \end{bmatrix}$$

- Initial features
  - $y_{i-1} = \texttt{<S>}, y_i = \ldots$
- Transition features
  - $y_{i-1} = \ldots, y_i = \ldots$

Structured Perceptrons

- We can arrange our weight matrix $\boldsymbol{\Theta}$ as follows:

$$\begin{bmatrix} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{bmatrix}$$

- Initial features
  - $y_{i-1} = \texttt{<S>}, y_i = \ldots$
- Transition features
  - $y_{i-1} = \ldots, y_i = \ldots$
- Emission features
  - $x_i = \ldots, y_i = \ldots$

Initialization Step

- We want to compute local scores $z(y_1, \texttt{<S>})$ for each possible y_1
  - These are the elements of $\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \mathbf{\Theta}$

Initialization Step

$$\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta}$$

$$= \left[\quad \middle| \qquad \middle| \qquad\qquad \right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array} \right]$$

Initialization Step

$$\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \mathbf{\Theta}$$

$$= \left[\begin{array}{c|c|c} 1 & \quad\quad & \quad\quad\quad\quad \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- We know that $y_{i-1} = \texttt{<S>}$

Initialization Step

$$\mathbf{z}_1 = \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} 1 & \mathbf{0} & \quad \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- We know that y_{i-1} cannot be any other tag

Initialization Step

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \mathbf{\Theta} \\
&= \left[\begin{array}{c|c|c} 1 & \mathbf{0} & \mathbf{1}\{x_1 = o_1\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]
\end{aligned}
$$

- One-hot vector of the first word

Initialization Step

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} 1 & \mathbf{0} & \mathbf{1}\{x_1 = o_1\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right] \\
&= 1 \cdot \pi + \mathbf{0} \cdot \mathbf{A} + \mathbf{1}\{x_1 = o_1\} \cdot \mathbf{B} \\
&= \pi + \mathbf{B}[o_1]
\end{aligned}
$$

Initialization Step

$$
\begin{aligned}
\mathbf{z}_1 &= \mathbf{f}(X, y_1, \texttt{<S>}, 1) \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} 1 & \quad \mathbf{0} \quad & \mathbf{1}\{x_1 = o_1\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \\ \mathbf{A} \\ \\ \hline \\ \mathbf{B} \\ \\ \end{array}\right] \\
&= 1 \cdot \pi + \mathbf{0} \cdot \mathbf{A} + \mathbf{1}\{x_1 = o_1\} \cdot \mathbf{B} \\
&= \pi + \mathbf{B}[o_1]
\end{aligned}
$$

- These local scores go into the first column of the Viterbi trellis

Recursion Step

- We want to compute local scores $z(y_i, y_{i-1})$ for each possible combination of y_i and y_{i-1}
  - We can stack the feature vectors for each possible y_{i-1}, $\mathbf{f}(X, y_i, y_{i-1}, i)$, on top of each other, in order
    - Form a feature matrix $\mathbf{F}_i$

Recursion Step

- We want to compute local scores $z(y_i, y_{i-1})$ for each possible combination of y_i and y_{i-1}
  - We can stack the feature vectors for each possible y_{i-1}, $\mathbf{f}(X, y_i, y_{i-1}, i)$, on top of each other, in order
    - Form a feature matrix $\mathbf{F}_i$
  - Compute $\mathbf{Z}_i = \mathbf{F}_i \cdot \mathbf{\Theta}$

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta}$$

$$= \begin{bmatrix} & \vert & & \vert & & & \end{bmatrix} \cdot \begin{bmatrix} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{bmatrix}$$

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} \mathbf{0} & \quad & \qquad \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- We know that $y_{i-1} \neq$ <S>

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \mathbf{\Theta}$$

$$= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \quad\quad\quad\quad \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- Identity matrix!

Recursion Step

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta}$$

$$= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \mathbf{1}\{x_i = o_i\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right]$$

- Stack of one-hot vectors

Recursion Step

$$
\begin{aligned}
\mathbf{Z}_i &= \mathbf{F}_i \cdot \mathbf{\Theta} \\
&= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \mathbf{1}\{x_i = o_i\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right] \\
&= 0 \cdot \pi + \mathbf{I} \cdot \mathbf{A} + \mathbf{1}\{x_i = o_i\} \cdot \mathbf{B} \\
&= \mathbf{A} + \mathbf{B}[o_i]
\end{aligned}
$$

Recursion Step

$$
\begin{aligned}
\mathbf{Z}_i &= \mathbf{F}_i \cdot \boldsymbol{\Theta} \\
&= \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{I} & \mathbf{1}\{x_i = o_i\} \end{array}\right] \cdot \left[\begin{array}{c} \pi \\ \hline \mathbf{A} \\ \hline \mathbf{B} \end{array}\right] \\
&= 0 \cdot \pi + \mathbf{I} \cdot \mathbf{A} + \mathbf{1}\{x_i = o_i\} \cdot \mathbf{B} \\
&= \mathbf{A} + \mathbf{B}[o_i]
\end{aligned}
$$

- Use the Viterbi algorithm to combine these local scores with scores from the rest of the sequence

Recursion Step

- Use the Viterbi algorithm to combine these local scores with scores from the rest of the sequence

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \boldsymbol{\Theta} = \mathbf{A} + \mathbf{B}[o_i]$$

Recursion Step

- Use the Viterbi algorithm to combine these local scores with scores from the rest of the sequence

$$\mathbf{Z}_i = \mathbf{F}_i \cdot \mathbf{\Theta} = \mathbf{A} + \mathbf{B}[o_i]$$

$$\text{Viterbi}[:, i] = \max(\text{Viterbi}[:, i-1 : i] + \mathbf{A} + \mathbf{B}[o_i], \text{axis}{=}0)$$

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 HW4, you do not have to return the path (log-)probability/score, just the backtrace path

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing
- Else:
  - Increment weights for features in Y, decrement weights for features in $\hat{Y}$

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing
- Else:
  - Increment weights for features in Y, decrement weights for features in $\hat{Y}$
  - In other words, for each time step i:
    - Increment weights for features in y_i, decrement weights for features in $\hat{y}_i$

Structured Perceptron Learning Algorithm

- Use the Viterbi algorithm to compute the best tag sequence
- If $\hat{Y} = Y$, then do nothing
- Else:
  - Increment weights for features in Y, decrement weights for features in $\hat{Y}$
  - In other words, for each time step i:
    - Increment weights for features in y_i, decrement weights for features in $\hat{y}_i$
  - Nothing fancy; no Numpy tricks needed