Statistical Learning for Text Data Analytics Sequence Labeling and Structured Output Learning: HMM

Yangqiu Song

Hong Kong University of Science and Technology

yqsong@cse.ust.hk

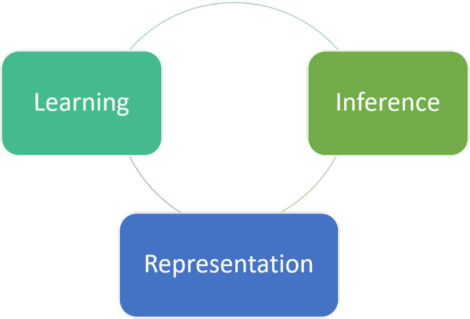
Spring 2018

∗Contents are based on materials created by Vivek Srikumar, Dan Roth, Xiaojin (Jerry) Zhu, Chris Manning

Reference Content

- Dan Roth. CS546: Machine Learning and Natural Language . http://l2r.cs.uiuc.edu/~danr/Teaching/CS546-16/
- Vivek Srikumar. CS 6355 Structured Prediction. https://svivek.com/teaching/structured-prediction/spring2018/
- Xiaojin (Jerry) Zhu. CS 769: Advanced Natural Language Processing. http://pages.cs.wisc.edu/~jerryzhu/cs769.html
- Chris Manning. CS 224N/Ling 237. Natural Language Processing. https://web.stanford.edu/class/cs224n/

Course Topics



- Representation: language models, word embeddings, topic models
- Learning: supervised learning, semi-supervised learning, sequence models, deep learning, optimization techniques
- Inference: constraint modeling, joint inference, search algorithms

NLP applications: tasks introduced in Lecture 1

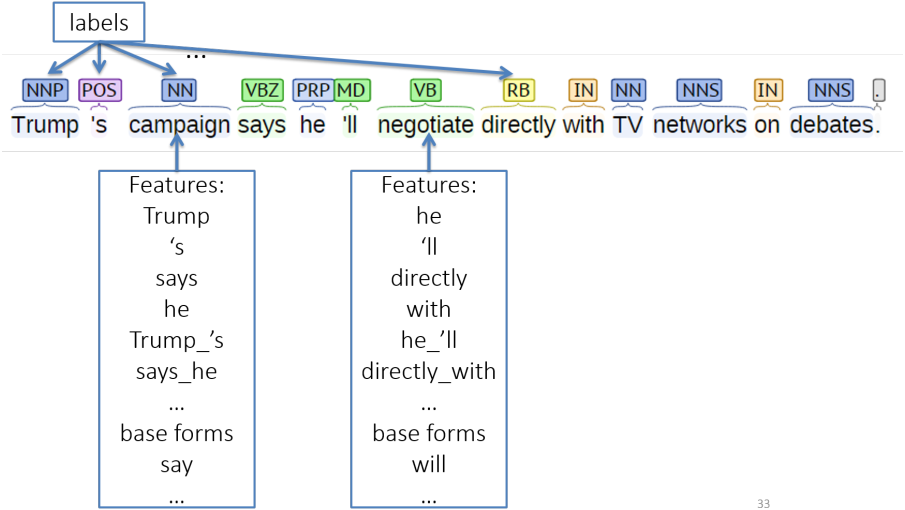
Overview

1. Hidden Markov Models
   - Representation
   - Learning
   - Inference

Sequences

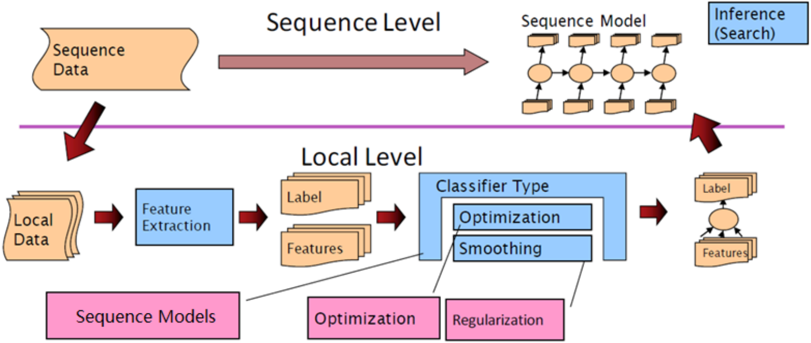
- Sequences of states
  - Text is a sequence of words or even letters
- If there are K unique states, the set of unique state sequences is infinite
- Our goal (for now): Define probability distributions over sequences
- If $x_1, x_2, \dots, x_n$ is a sequence that has n tokens, we want to be able to define $P(x_1, x_2, \dots, x_n)$
  - We have seen a lot of models for this in language models
  - N-gram language model makes $(n-1)$th-order Markov assumption

Classification Problem



The General Framework of Training and Testing

- Analogous to classification



Label and Feature Dependencies

- Current label may dependent on the previous one
  - Fed in "The Fed" is a Noun because it follows a Determiner
  - Fed in "I fed the.." is a Verb because it follows a Pronoun
- Sometimes more difficult: "I/PN can/MD can/VB a/DT can/NN."
- Two kinds of information incorporated in learning:
  - Some tag sequences are more likely than others. For instance, DT JJ NN is quite common, while DT JJ VBP is unlikely. ("a new book")
  - A word may have multiple possible POS, but some are more likely than others, e.g., "flour" is more often a noun than a verb
- The question is:
  - Given a word sequence

$$\mathbf{x}_{1:N} \doteq \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N,$$

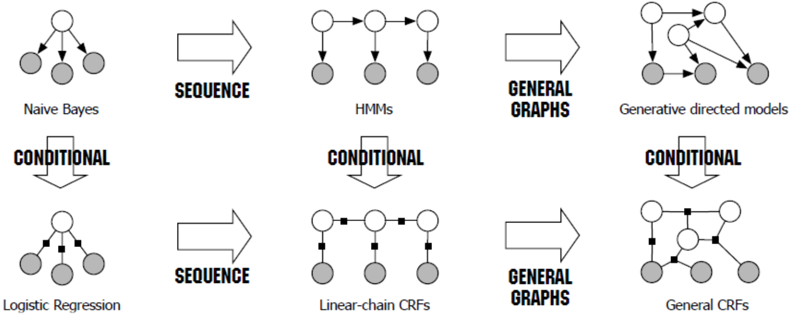
    how do we compute the most likely POS sequence

$$y_{1:N} \doteq y_1, y_2, \dots, y_N$$

  - One method is to use a Hidden Markov Model

Classifiers Feasible for Sequence Labeling

- Generative
  - Naive Bayes
  - Hidden Markov model (HMM)
- Discriminative models
  - Maximum entropy, logistic regression
  - Maximum Entropy Markov Model (MMEM)
  - Conditional random field (CRF)

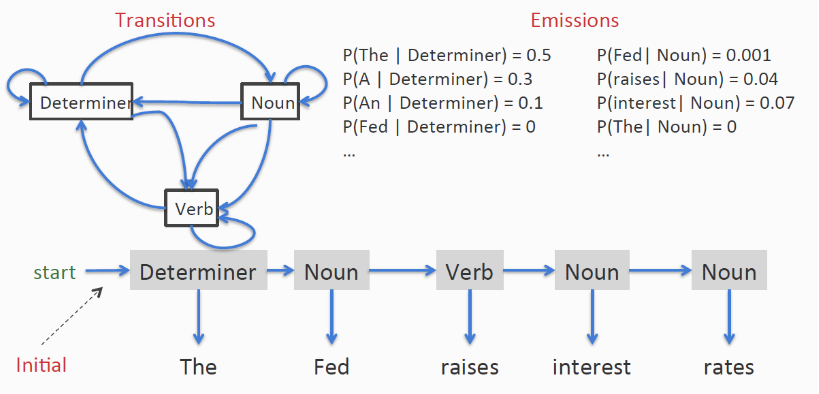


Hidden Markov Model

- Discrete Markov Model
  - States follow a Markov chain
  - Each state is an observation
- Hidden Markov Model
  - States follow a Markov chain
  - States are not observed
  - Each state stochastically emits an observation

A Toy Part-of-Speech Example

- Sentence "The Fed raises interest rates"



Joint Model over States and Observations

- Given a word sequence $\mathbf{x}_{1:N}$, how do we compute the most likely POS sequence $y_{1:N}$? We denote:
  - Number of states (types, labels) $= K$
  - Number of observations (features) $= d$
  - $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)^\top$: Initial probability over states (K dimensional vector)
  - $\mathbf{A} \in \mathbb{R}^{K \times K}$: Transition probabilities
    - $A_{ij} = P(y_n = j | y_{n-1} = i)$
    - This is a first-order Markov assumption on the states
  - $\boldsymbol{\Phi} \in \mathbb{R}^{K \times d} = (\phi_1, \ldots, \phi_K)^\top$: Emission probabilities
    - For texts $\phi_k = (\phi_k^{(1)}, \ldots, \phi_k^{(d)})^\top$ can be a multinomial distribution
- The parameters of an HMM are $\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\Phi}\}$
- This is a generative model. We can run an HMM for N steps, and produce $x_{1:N}, y_{1:N}$
- The joint probability is

$$P(\mathbf{x}_{1:N}, y_{1:N} | \Theta) = P(y_1 | \boldsymbol{\pi}) P(\mathbf{x}_1 | y_1, \boldsymbol{\Phi}) \prod_{n=1}^{N} P(y_n | y_{n-1}, \mathbf{A}) P(\mathbf{x}_n | y_n, \boldsymbol{\Phi})$$

Three Questions for HMMs (Rabiner (1990))

- Given an observation sequence, $\mathbf{x}_{1:N}$ and a model $\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\Phi}\}$, how to efficiently calculate the probability of the observation $P(\mathbf{x}_{1:N}|\Theta)$?
- Given an observation sequence, $\mathbf{x}_{1:N}$ and a model $\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\Phi}\}$, how to efficiently calculate the most probable state sequence $y_{1:N}$?
- How do we adjust the model parameters $\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\Phi}\}$ to maximize $P(\mathbf{x}_{1:N}|\Theta)$?

Mapping to Our Problems

- Representation
  - Hidden states follows first-order Markov chain
  - Features are modeled with a multinomial emission distribution
  - We can evaluate $P(\mathbf{x}_{1:N}, y_{1:N}|\Theta)$ of an observation sequence
- Learning
  - Finding parameters $\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\Phi}\}$
  - Supervised case: trivial parameter estimation
  - Unsupervised/semi-supervised case: EM algorithm (known as Baum-Welch algorithm)
    - EM algorithm involves the so-called forward backward (or in general sum-product) algorithm
- Inference (or decoding problem)
  - Assign a label to a sequence, corresponding to $\arg\max_{y_{1:N}} = P(y_{1:N}|x_{1:N}, \Theta)$
  - Finding the most likely state sequence to explain the observation sequence
    - It can be exactly solved by Viterbi algorithm (or in general max-product)
    - We can also use greedy search or beam search to have approximate solutions

Overview

1. Hidden Markov Models
   - Representation
   - Learning
   - Inference

Learning: The Trivial Case

- We can find Θ by maximzing the likelihood of observed data
- When $y_{1:N}$ is observed ($\mathbf{x}_{1:N}$ is also observed), which is the supervised learning case, MLE boils down to the frequency estimate
  - A_{ij} is the fraction of times $y_{n-1} = i$ followed by $y_n = j$
  - $\phi_k = P(\mathbf{x}|y = k)$ corresponds to the fraction of times $\mathbf{x}$ is produced under state k
  - $\boldsymbol{\pi}$ is the fraction of times each state being the first state of a sequence (assuming we have multiple training sequences)
- This is done very similar to naive Bayes classifier

Priors and Smoothing

- Maximum likelihood estimation works best with lots of annotated data
  - Never the case
- Priors inject information about the probability distributions
  - Dirichlet priors for multinomial distributions
- Effectively additive smoothing
  - Add small constants to the count

Learning: $y_{1:N}$ is Unobserved

For unsupervised learning:

- The MLE will maximize (up to a local optimum, see below) the likelihood of observed data

$$P(\mathbf{x}_{1:N}|\Theta) = \sum_{y_{1:N}} P(\mathbf{x}_{1:N}, y_{1:N}|\Theta)$$

  where the summation is over all possible label sequences of length N
  - This is an exponential sum with K^N label sequences
  - HMM training uses a combination of dynamic programming and EM to handle this issue

Lower Bound for EM Algorithm

- Note the log likelihood involves summing over hidden variables, which suggests we can apply Jensens inequality to lower bound

$$\begin{aligned} P(\mathbf{x}_{1:N}|\Theta) &= \log \textstyle\sum_{y_{1:N}} P(\mathbf{x}_{1:N}, y_{1:N}|\Theta) \\ &= \log \textstyle\sum_{y_{1:N}} P(y_{1:N}|\mathbf{x}_{1:N}, \Theta^{old}) \frac{P(\mathbf{x}_{1:N}, y_{1:N}|\Theta)}{P(y_{1:N}|\mathbf{x}_{1:N}, \Theta^{old})} \\ &\geq \textstyle\sum_{y_{1:N}} P(y_{1:N}|\mathbf{x}_{1:N}, \Theta^{old}) \log \frac{P(\mathbf{x}_{1:N}, y_{1:N}|\Theta)}{P(y_{1:N}|\mathbf{x}_{1:N}, \Theta^{old})} \end{aligned}$$

- In E-step, we find the posterior $P(y_{1:N}|\mathbf{x}_{1:N}, \Theta^{old})$
- In M-step, we maximize the above lower bound (taking the parts that depends on)

$$Q(\Theta, \Theta^{old}) = \textstyle\sum_{y_{1:N}} P(y_{1:N}|\mathbf{x}_{1:N}, \Theta^{old}) \log P(\mathbf{x}_{1:N}, y_{1:N}|\Theta)$$

EM Algorithm

$$Q(\Theta, \Theta^{old}) = \sum_{y_{1:N}} P(y_{1:N}|\mathbf{x}_{1:N}, \Theta^{old}) \log P(\mathbf{x}_{1:N}, y_{1:N}|\Theta)$$

- We introduce two sets of variables (E-Step):

$$\gamma_n(k) = P(y_n = k|\mathbf{x}_{1:N}, \Theta^{old})$$

$$\xi_n(jk) = P(y_{n-1} = j, y_n = k|\mathbf{x}_{1:N}, \Theta^{old})$$

  to denote the node marginals and edge marginals (conditioned on input $\mathbf{x}_{1:N}$, under the old parameters)
- Given

$$P(\mathbf{x}_{1:N}, y_{1:N}|\Theta) = P(y_1|\boldsymbol{\pi})P(\mathbf{x}_1|y_1, \boldsymbol{\Phi}) \prod_{n=2}^{N} P(y_n|y_{n-1}, \mathbf{A})P(\mathbf{x}_n|y_n, \boldsymbol{\Phi})$$

- The Q function can be written as

$$\begin{aligned} Q(\Theta, \Theta^{old}) &= \sum_{k=1}^{K} \gamma_1(k) \log \pi_k \\ &+ \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_n(k) \log P(\mathbf{x}_n|y_n, \phi_k) \\ &+ \sum_{n=2}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} \xi_n(jk) \log A_{jk} \end{aligned}$$

M-step

- The M-step is a constrained optimization problem since the parameters need to be normalized. As before, one can introduce Lagrange multipliers and set the gradient of the Lagrangian to zero to arrive at

$$\pi_k \propto \gamma_1(k)$$

$$A_{jk} \propto \sum_{n=2}^{N} \xi_n(jk)$$

where A_{jk} is normalized over k

- ϕ_k is maximized depending on the particular form of the distribution. If it is multinomial, we have

$$\phi_k \propto \sum_n \gamma_n(k)\mathbf{x}_n$$

E-Step

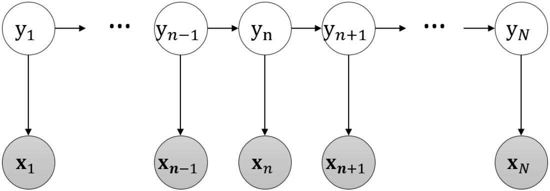
In the E-step,

- We need to compute $\gamma_n(k)$ and $\xi_n(jk)$
- Particularly we have

$$
\begin{aligned}
\gamma_n(k) &= P(y_n = k|\mathbf{x}_{1:N}, \Theta^{old}) \\
&\doteq P(y_n = k|\mathbf{x}_{1:N}) \\
&= \frac{P(\mathbf{x}_{1:N}|y_n{=}k)P(y_n{=}k)}{P(\mathbf{x}_{1:N})} \\
&= \frac{P(\mathbf{x}_{1:n}|y_n{=}k)P(\mathbf{x}_{n+1:N}|y_n{=}k)P(y_n{=}k)}{P(\mathbf{x}_{1:N})} \\
&= \frac{\color{blue}{P(\mathbf{x}_{1:n},y_n{=}k)}\color{red}{P(\mathbf{x}_{n+1:N}|y_n{=}k)}}{P(\mathbf{x}_{1:N})} \\
&\doteq \frac{\color{blue}{\alpha(y_n{=}k)}\color{red}{\beta(y_n{=}k)}}{P(\mathbf{x}_{1:N})}
\end{aligned}
$$

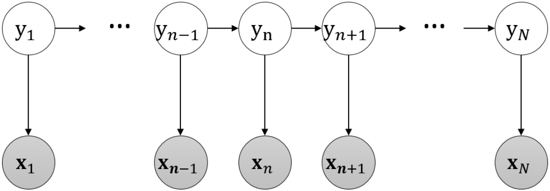
- We use an recursive way to compute forward $\alpha(y_n)$ and backward $\beta(y_n)$
- This is consistent with the “sum-product” algorithm

Forward Recursion $\alpha(y_n)$



$$
\begin{aligned}
\alpha(y_n) &= P(\mathbf{x}_{1:n}, y_n) \\
&= P(y_n)P(\mathbf{x}_n|y_n)P(\mathbf{x}_{1:n-1}|y_n) \\
&= P(\mathbf{x}_n|y_n)P(\mathbf{x}_{1:n-1}, y_n) \\
&= P(\mathbf{x}_n|y_n)\textstyle\sum_{y_{n-1}} P(\mathbf{x}_{1:n-1}, y_{n-1}, y_n) \\
&= P(\mathbf{x}_n|y_n)\textstyle\sum_{y_{n-1}} P(\mathbf{x}_{1:n-1}, y_n|y_{n-1})P(y_{n-1}) \\
&= P(\mathbf{x}_n|y_n)\textstyle\sum_{y_{n-1}} P(\mathbf{x}_{1:n-1}|y_{n-1})P(y_n|y_{n-1})P(y_{n-1}) \\
&= P(\mathbf{x}_n|y_n)\textstyle\sum_{y_{n-1}} P(\mathbf{x}_{1:n-1}, y_{n-1})P(y_n|y_{n-1}) \\
&= P(\mathbf{x}_n|y_n)\textstyle\sum_{y_{n-1}} \alpha(y_{n-1})P(y_n|y_{n-1})
\end{aligned}
$$

Backward Recursion $\beta(y_n)$



$$
\begin{aligned}
\beta(y_n) &= P(\mathbf{x}_{n+1:N}|y_n)\\
&= \sum_{y_{n+1}} P(\mathbf{x}_{n+1:N}, y_{n+1}|y_n)\\
&= \sum_{y_{n+1}} P(\mathbf{x}_{n+1:N}|y_{n+1}, y_n)P(y_{n+1}|y_n)\\
&= \sum_{y_{n+1}} P(\mathbf{x}_{n+1:N}|y_{n+1})P(y_{n+1}|y_n)\\
&= \sum_{y_{n+1}} P(\mathbf{x}_{n+2:N}|y_{n+1})P(\mathbf{x}_{n+1}|y_{n+1})P(y_{n+1}|y_n)\\
&= \sum_{y_{n+1}} \beta(y_{n+1})P(\mathbf{x}_{n+1}|y_{n+1})P(y_{n+1}|y_n)
\end{aligned}
$$

E-Step (Cont'd)

- After computing forward recursion $\alpha(y_n)$ and backward recursion $\beta(y_n)$ we have

$$\gamma_n(k) = \frac{\alpha(y_n = k)\beta(y_n = k)}{P(\mathbf{x}_{1:N})}$$

- Similarly, we have

$$\xi_n(jk) = \frac{\alpha(y_{n-1} = j)P(y_n = k|y_{n-1} = j)P(\mathbf{x}_n|y_n = k)\beta(y_n = k)}{P(\mathbf{x}_{1:N})}$$

Overview

1. Hidden Markov Models
   - Representation
   - Learning
   - Inference

Most Likely State Sequence

- Input:
  - A hidden Markov model $\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\Phi}\}$
  - An observation sequence $\mathbf{x}_{1:N}$
- Output: A state sequence $y_{1:N}$ that corresponds to

$$\arg\max_{y_{1:N}} P(y_{1:N} | \mathbf{x}_{1:N}, \Theta)$$

  - This is maxinum a posteriori inference (MAP inference)
- Computationally a combinatorial optimization problem

MAP Inference

- We want $\arg\max_{y_{1:N}} P(y_{1:N}|\mathbf{x}_{1:N}, \Theta)$
- Note that $P(y_{1:N}|\mathbf{x}_{1:N}, \Theta) \propto P(y_{1:N}, \mathbf{x}_{1:N}|\Theta)$
  - And we don't care about $P(\mathbf{x}_{1:N})$ since we are maximizing over $y_{1:N}$
- So

$$\arg\max_{y_{1:N}} P(y_{1:N}|\mathbf{x}_{1:N}, \Theta) = \arg\max_{y_{1:N}} P(y_{1:N}, \mathbf{x}_{1:N}|\Theta)$$

- We have defined

$$P(\mathbf{x}_{1:N}, y_{1:N}|\Theta) = P(y_1|\boldsymbol{\pi})P(\mathbf{x}_1|y_1, \boldsymbol{\Phi}) \prod_{n=2}^{N} P(y_n|y_{n-1}, \mathbf{A})P(\mathbf{x}_n|y_n, \boldsymbol{\Phi})$$

- We omit the parameters for the ease of derivation

$$P(\mathbf{x}_{1:N}, y_{1:N}) = P(y_1)P(\mathbf{x}_1|y_1) \prod_{n=2}^{N} P(y_n|y_{n-1})P(\mathbf{x}_n|y_n)$$

How Many Possible Sequences?

The	Fed	raises	interest	rates
		List of allowed tags for each word		
Determiner	Verb	Verb	Verb	Verb
	Noun	Noun	Noun	Noun
1	2	2	2	2

- In this simple case, we have 16 candidate sequences

$$(1 \times 2 \times 2 \times 2 \times 2)$$

How Many Possible Sequences?

- Output: one state per observation $y_n = s_k$

Observations	x_1	x_2	...	x_n
	List of allowed states for each observation			
	s_1	s_1	...	s_1
	s_2	s_2		s_2
	s_3	s_2		s_3
	.	.		.
	.	.		.
	s_K	s_K		s_K

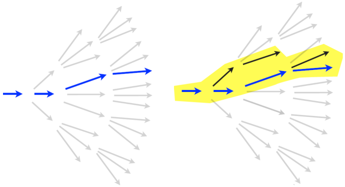
- We have K^n possible sequences to consider in $\arg\max_{y_{1:N}} P(y_{1:N}, \mathbf{x}_{1:N}|\Theta)$

Naive Approaches

- Try out every sequences
  - Score the sequence $y_{1:N}$ using $P(y_{1:N}, \mathbf{x}_{1:N}|\Theta)$
  - Return the highest scoring one
  - Correct but slow $O(K^N)$
- Greedy search
  - Construct the output left to right
  - For each n, elect the best y_n using y_{n-1} and $\mathbf{x}_n$
  - Incorrect but fast, $O(NK)$

Beam Search

- Beam inference
  - At each position keep the top k complete sequences
  - Extend each sequence in each local way
  - The extensions compete for the k slots at the next position



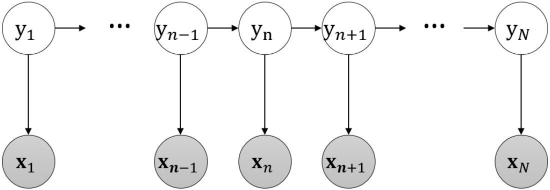
(a) Greedy (b) Beam Search

- Advantages
  - Fast; beam sizes of 3-5 are almost as good as exact inference in many cases
  - Easy to implement (no dynamic programming required)
- Disadvantage
  - Inexact: the globally best sequence can fall off the beam

Optimal Solution: General Idea

- Dynamic programming
  - The best solution for the full problem relies on the best solution to the sub-problem
  - Memorize partial computation
- Examples
  - Viterbi algorithm
  - Dijkstra's shortest path algorithm
  - MDP value iteration
  - ...

Deriving the Recursion



$$\max_{y_{1:N}} P(\mathbf{x}_{1:N}, y_{1:N}) = \max_{y_{1:N}} P(y_1)P(\mathbf{x}_1|y_1)\prod_{n=2}^{N} P(y_n|y_{n-1})P(\mathbf{x}_n|y_n)$$

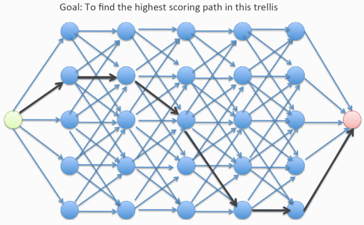
We reorganize it as

$$\max_{y_{1:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \ldots \cdot P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot P(\mathbf{x}_1|y_1)P(y_1)$$

Deriving the Recursion

$$
\begin{aligned}
& \max_{y_{1:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \ldots \cdot P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot P(\mathbf{x}_1|y_1)P(y_1) \\
= & \max_{y_{2:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \ldots \cdot \max_{y_1} P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot P(\mathbf{x}_1|y_1)P(y_1) \\
= & \max_{y_{2:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \ldots \cdot \max_{y_1} P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot score_1(y_1) \\
= & \max_{y_{3:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \ldots \cdot \max_{y_2} P(\mathbf{x}_3|y_3)P(y_3|y_2) \\
& \cdot \max_{y_1} P(\mathbf{x}_2|y_2)P(y_2|y_1) \cdot score_1(y_1) \\
= & \max_{y_{3:N}} P(\mathbf{x}_N|y_N)P(y_N|y_{N-1}) \cdot \ldots \cdot \max_{y_2} P(\mathbf{x}_3|y_3)P(y_3|y_2) \cdot score_2(y_2) \\
= & \ldots \\
= & \max_{y_N} score_N(y_N)
\end{aligned}
$$

where we have $score_n(y_n) = \max_{y_{n-1}} P(y_n|y_{n-1})P(\mathbf{x}_n|y_n)score_{n-1}(y_{n-1})$



Complexity of Inference

- Complexity parameters
  - Input sequence length: N
  - Number of states: K
- Memory
  - Storing the table: NK (scores for all states at each position)
- Runtime
  - At each step, go over pairs of states
  - $O(NK^2)$

Summary of Viterbi Inference

- Viterbi inference
  - Dynamic programming or memoization
  - Requires small window of state influence (e.g., past two states are relevant)
- Advantage
  - Exact: the global best sequence is returned
- Disadvantage
  - Harder to implement long-distance state-state interactions (but beam inference tends not to allow long-distance resurrection of sequences anyway)

Summary

- Predicting sequences
  - As many output states as observations
- Markov assumption helps decompose the score
- Several algorithmic questions
  - Most likely state
  - Learning parameters: supervised, unsupervised (posterior, sum-product algorithm)
  - Probability of an observation sequence: sum over all assignments of states; replace max with sum in Viterbi
  - Inference: Viterbi (or max-product algorithm)

Next...

- Conditional Models and Local Classifiers
- Global models
  - Conditional Random Fields
  - Structured Perceptron for sequences

References

Rabiner, L. R. (1990). Readings in speech recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296.