Genome 540 Discussion

Conor Camplisson

March 2nd, 2023

Genome Sciences
UNIVERSITY OF WASHINGTON

Outline

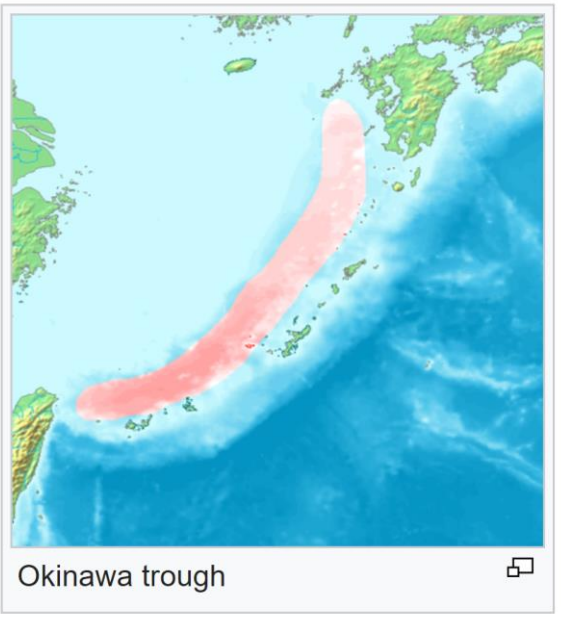
- Homework 8 details & questions

Homework 8 Overview

Pyrococcus horikoshii	
Scientific classification	
Domain:	Archaea
Kingdom:	Euryarchaeota
Phylum:	Euryarchaeota
Class:	Thermococci
Order:	Thermococcales
Family:	Thermococcaceae
Genus:	*Pyrococcus*
Species:	***P. horikoshii***
Binomial name	
Pyrococcus horikoshii Erauso *et al.* 1993	

Pyrococcus horikoshii

- Hyperthermophile, 98 C !
- Anaerobic archaeon
- Isolated from Okinawa Trough
- Growth enhanced by Sulfur
- 32 min doubling time (growth rate)


Okinawa trough

package, 1996). The phylogenetic tree diagrams were generated by the PHYLIP suite of programs (Kuhner and Felsenstein 1994).

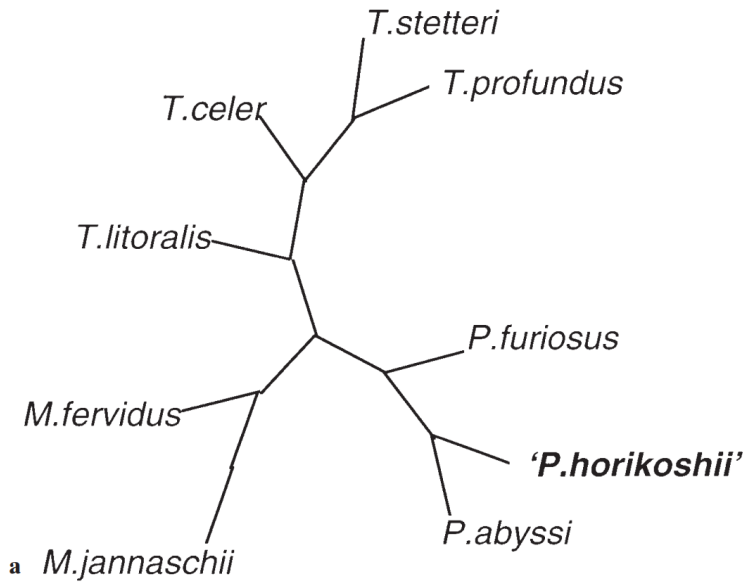


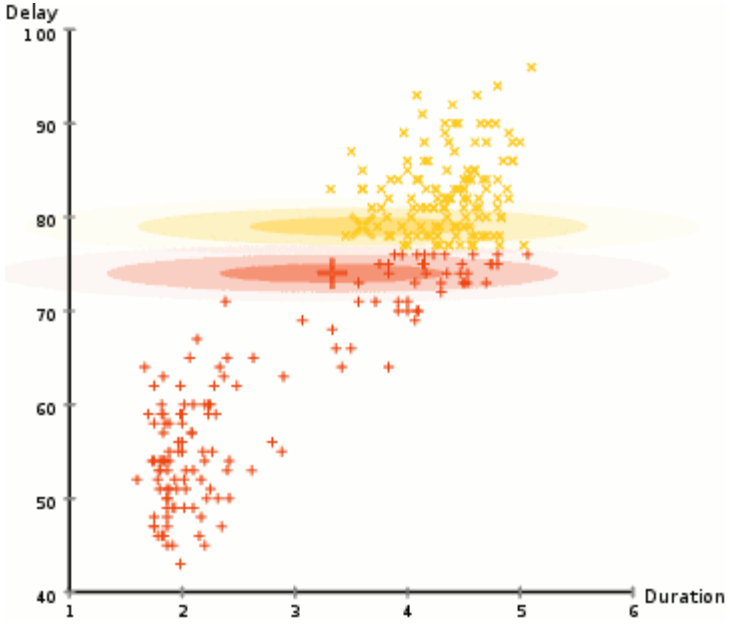
Fig. 3. Phylogenetic trees representing the relatedness of *P. horikoshii*

González, J., Masuchi, Y., Robb, F. et al. *Extremophiles* **2**, 123–130 (1998). https://doi.org/10.1007/s007920050051

EM Algorithm Overview

In statistics, an expectation–maximization (EM) algorithm is an iterative method to find (local) maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables.

The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and a maximization (M) step, which computes parameters maximizing the expected log-likelihood found on the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step.



EM clustering of Old Faithful eruption data.

The random initial model (which, due to the different scales of the axes, appears to be two very flat and wide elipses) is fit to the observed data.

In the first iterations, the model changes substantially, but then converges to the two modes of the geyser.

https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm

Baum–Welch + Forward/Backward Algorithm

the **Baum–Welch algorithm** is a special case of the **expectation–maximization algorithm (EM)** used to find the unknown parameters of a hidden Markov model (HMM). It makes use of the **forward-backward algorithm** to compute the statistics for the expectation step.

The **forward-backward algorithm** makes use of the principle of **dynamic programming** to efficiently compute the values that are required to obtain the posterior marginal distributions in two passes. The first pass goes forward in time while the second goes backward in time; hence the name forward–backward algorithm.

Forward procedure [edit]

Let $\alpha_i(t) = P(Y_1 = y_1, \ldots, Y_t = y_t, X_t = i \mid \theta)$, the probability of seeing the observations $y_1, y_2, \ldots, y_t$ and being in state i at time t. This is found recursively:

1. $\alpha_i(1) = \pi_i b_i(y_1)$,
2. $\alpha_i(t+1) = b_i(y_{t+1}) \sum_{j=1}^{N} \alpha_j(t) a_{ji}$.

Since this series converges exponentially to zero, the algorithm will numerically underflow for longer sequences.[6] However, this can be avoided in a slightly modified algorithm by scaling α in the forward and β in the backward procedure below.

https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm

Homework 8 Overview

Baum-Welch Algorithm

Baum–Welch is an expectation-maximization algorithm that uses the forward–backward algorithm.

1. Use the **forward algorithm** to calculate the forward probabilities for the HMM.

2. Use the **backward algorithm** to calculate the backward probabilities for the HMM.

3. Re-estimate transition, emission, and initial probabilities by calculating the expected number of each edge type

4. Calculate the new log likelihood of the model (the likelihood of our observations given our re-tuned model)

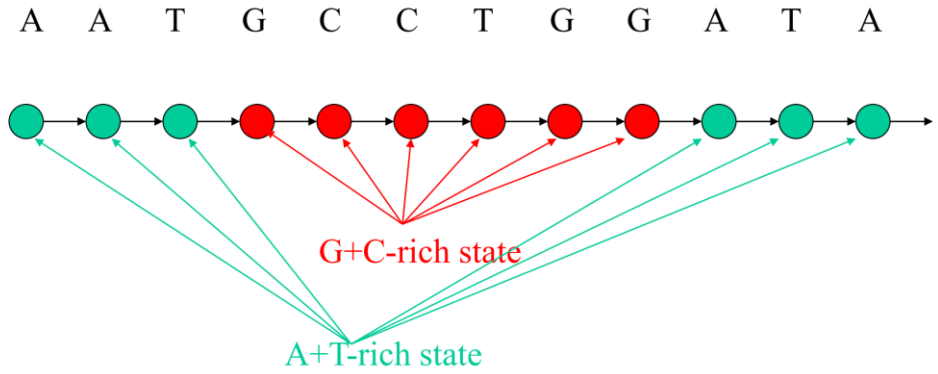
5. Repeat until the change in log likelihood is smaller than a given threshold or when a maximum number of iterations is passed.

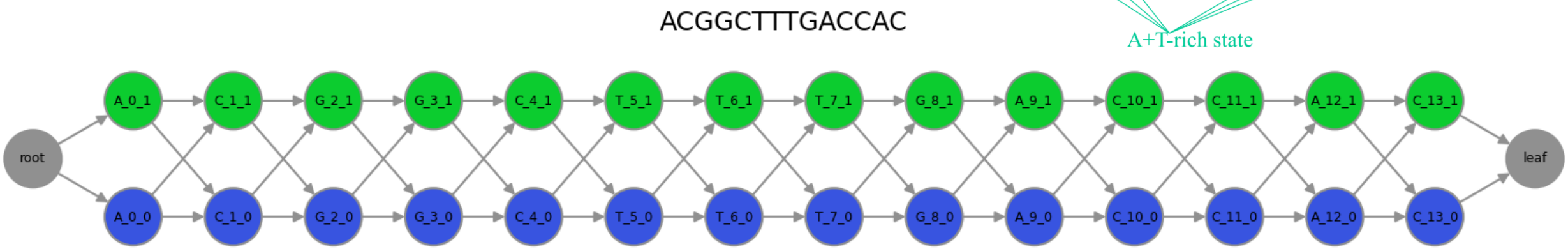
Homework 8 Overview

- 2-state HMM for detecting GC-rich regions in *Pyrococcus horikoshii* genome
  - state 1: AT-rich, state 2: GC-rich
  - starting parameters are given: initiation, transition, emission probabilities

- Use Baum-Welch training to find improved parameter estimates
  - each iteration: compute log-likelihood of the sequence, new probabilities

- Run until parameter estimates converge
  - stop when log-likelihood increase < 0.1





http://bozeman.mbt.washington.edu/compbio/mbt599/assignments/hw8.html

HMM Overview

A general definition of HMM

$$HMM = (S, V, B, A, \Pi)$$

N states

$$S = \{s_1, \ldots, s_N\}$$

M symbols

$$V = \{v_1, \ldots, v_M\}$$

Initial state probability:

$$\Pi = \{\pi_1, \ldots, \pi_N\} \quad \sum_{i=1}^{N} \pi_i = 1$$

π_i : *prob of starting at state* s_i

State transition probability:

$$A = \{a_{ij}\} \quad 1 \le i, j \le N \quad \sum_{j=1}^{N} a_{ij} = 1$$

a_{ij} : *prob of going* $s_i \rightarrow s_j$

Output probability:

$$B = \{b_i(v_k)\} \quad 1 \le i \le N, 1 \le k \le M \quad \sum_{k=1}^{M} b_i(v_k) = 1$$

$b_i(v_k)$: *prob of "generating"* v_k *at* s_i

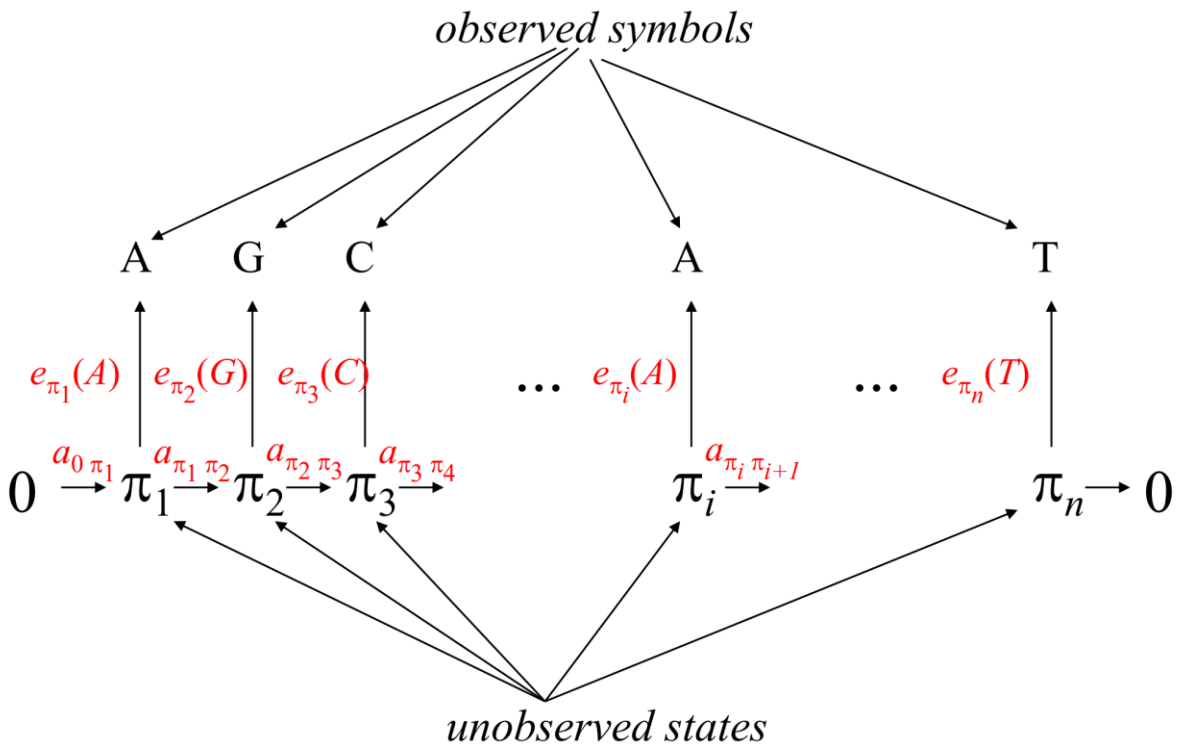
Three Basic Problems in HMMs

Given a set of observation sequences $O = O_1 O_2 \cdots O_T$ and the HMM parameters $\lambda = (A, B, \pi)$, computing the probability $P(O|\lambda)$

Given a set of observation sequences $O = O_1 O_2 \cdots O_T$ and the HMM parameters $\lambda = (A, B, \pi)$, computing the optimal state sequences

Given a set of observation sequences $O = O_1 O_2 \cdots O_T$ adjusting the HMM parameters $\lambda = (A, B, \pi)$ to maximize the probability $P(O|\lambda)$

HMM Overview

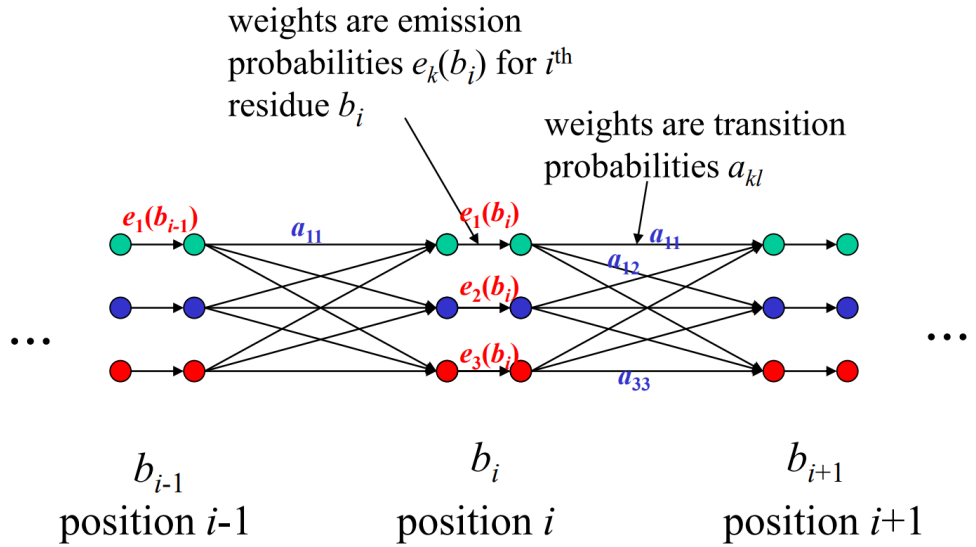


HMM Probabilities of Sequences

- Prob of sequence of states $\pi_1\pi_2\pi_3 \ldots \pi_n$ is
  $a_{0\pi_1}a_{\pi_1\pi_2}a_{\pi_2\pi_3}a_{\pi_3\pi_4} \ldots a_{\pi_{n-1}\pi_n}$.
- Prob of seq of observed symbols $b_1b_2b_3 \ldots b_n$,
  conditional on state sequence is
  $e_{\pi_1}(b_1)e_{\pi_2}(b_2)\, e_{\pi_3}(b_3) \ldots e_{\pi_n}(b_n)$
- Joint probability = $a_{0\pi_1}\prod^{n}_{i=1} a_{\pi_i\pi_{i+1}}\, e_{\pi_i}(b_i)$
  (define $a_{\pi_n\pi_{n+1}}$ to be 1)
- (Unconditional) prob of observed sequence
  = sum (of joint probs) over all possible state paths
  - not practical to compute directly, by 'brute force'! We will use dynamic programming.

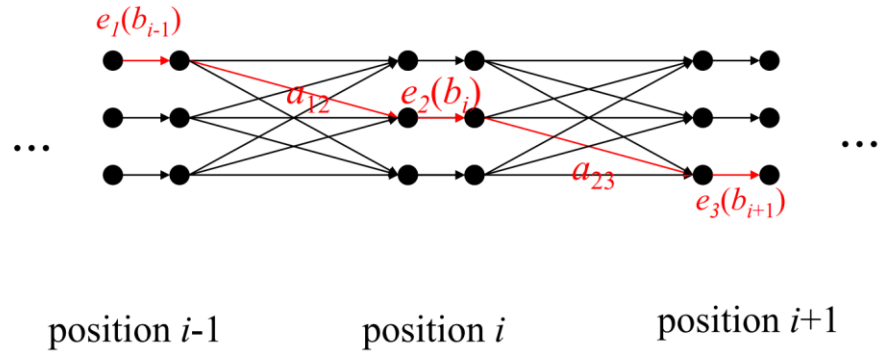
HMM Overview

WDAG for 3-state HMM, length *n* sequence

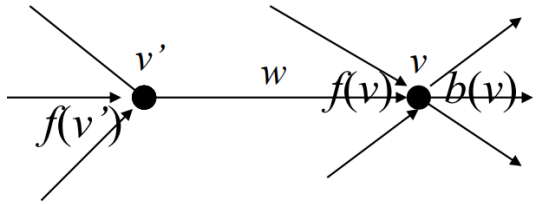


- ***Paths*** through graph from begin node to end node correspond to ***sequences of states***
- ***Product weight*** along path
  = ***joint probability*** of state sequence & observed symbol sequence
- ***Highest-weight path*** = ***highest probability state sequence***
- ***Sum*** *of (product) path weights,* ***over all paths***,
  = ***probability of observed sequence***
- ***Sum*** *of (product) path weights* ***over***
  - all paths *going through a particular node*, or
  - all paths *that include a particular edge*,

  divided by prob of observed sequence,
  = ***posterior probability*** of that edge or node



HMM Overview



$f(v)b(v)$ = sum of the path weights of all paths *through v*.

$f(v')wb(v)$ = sum of the path weights of all paths *through the edge (v',v)*

Forward/backward algorithm

- Work through graph in forward direction:
  - compute and store $f(v)$
- Then work through graph in backward direction:
  - compute $b(v)$
  - compute $f(v)$ $b(v)$ and $f(v)wb(v)$ as above, store in appropriate cumulative sums
  - only need to store $b(v)$ until have computed b's at next position
- Posterior probability of being in state s at position i is $f(v)$ $b(v)$ / *total sequence prob*
  - where v is the corresponding graph node

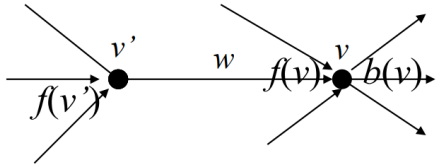
HMM Overview

Implementing Baum-Welch

– An edge in the WDAG contributes *fractional* (or *weighted*) *counts* given by its posterior probability:

– (*): $(\sum_{\text{all paths } p \text{ through edge } e} \text{weight}(p)) / (\sum_{\text{all paths } p} \text{weight}(p))$

(Fractional counts are computed using forward-backward algorithm)

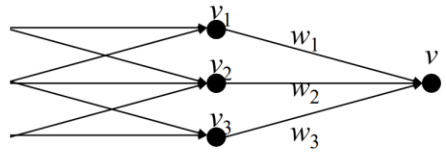


$f(v)b(v)$ = sum of the path weights of all paths *through v*.

$f(v')wb(v)$ = sum of the path weights of all paths *through the edge (v',v)*

For each vertex v, let $f(v) = \sum_{\text{paths } p \text{ ending at } v} \text{weight}(p)$, where weight($p$) = *product* of edge weights in p. Only consider paths starting at 'begin' node.

Compute $f(v)$ by dynam. prog: $f(v) = \sum_i w_i f(v_i)$, where v_i ranges over the parents of v, and w_i = weight of the edge from v_i to v.



Similarly for $b(v) = \sum_{p \text{ beginning at } v} \text{weight}(p)$

The paths *beginning* at v are the ones *ending* at v in the *reverse (or inverted) graph*

– Compute new param estimates

- $e_k(b)$^ = (frac. # times symbol b emitted by state k) / (frac. # times state k occurs)
- a_{kl} ^ = (frac. # times state k followed by state l) / (frac. # times state k occurs)
  - (In denom,, omit frac counts at last position of sequence)

where "frac. # times" is given by (*) for appropriate edge type (emission or transition)

HMM Overview

Re-estimation of parameters

$$\bar{\pi}_i = \text{expected frequency (number of times) in state } S_i \text{ at time } (t = 1) = \gamma_1(i)$$

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

$$= \frac{\sum_{t=1 \text{ s.t. } O_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}.$$

HMM Overview

See also: other slides/tutorials/videos

Probabilistic Inference in an HMM

Three fundamental questions for an HMM:

- Compute the probability of a given observation sequence, when the tag sequence is hidden (language modeling)
- Given an observation sequence, find the most likely hidden state sequence (tagging)
- Given observation sequence(s) and a set of states, find the parameters that would make the observations most likely (parameter estimation)

Baum-Welch training algorithm

- Begin with some model μ (perhaps random, perhaps preselected)
- Run O through the current model to estimate the expectations of each model parameter.
- Change the model to maximize the values of the paths that are used a lot (while still repsecting the stochastic constraints).
- Repeat, hoping to converge on optimal values for the model parameters, μ.

https://people.cs.umass.edu/~mccallum/courses/inlp2004a/lect10-hmm2.pdf

HMM Overview

See also: other slides/tutorials/videos

Calculating the probability of the observations and a state i at time t

Given model $\mu = (A, B)$
we want to find $P(x_t = i, O|\mu)$

$$P(P(x_t = i, O|\mu) = P(o_1 o_2 ... o_t, x_t = i|\mu) P(o_{t+1} o_{t+2} ... o_T | x_t = i, \mu)$$

(Why is this true?)

Remember we have the first part $\alpha_i(t) = P(o_1 o_2 ... o_t, x_t = i|\mu)$.

We need something for the second part: mirror image of the "forward procedure", called "backward procedure."

Probability of a state i at time t

$$\begin{aligned} P(x_t = i, O|\mu) &= P(o_1 o_2 ... o_t, x_t = i|\mu) P(o_{t+1} o_{t+2} ... o_T | x_t = i, \mu) \\ &= \alpha_i(t) \beta_i(t) \end{aligned}$$

$$P(x_t = i|O, \mu) = \frac{P(x_t = i, O|\mu)}{P(O|\mu)} = \gamma_i(t)$$

https://people.cs.umass.edu/~mccallum/courses/inlp2004a/lect10-hmm2.pdf

HMM Overview

See also: 2022 GS540 Discussion Slides

Forward Algorithm

$$\alpha_2(1) = [\alpha_1(1) \times a_{11} + \alpha_1(2) \times a_{21}] \times b_1(C)$$
$$= [0.32 \times 0.6 + 0.02 \times 0.5] \times 0.2$$
$$= 0.0404$$

1. Initialization:

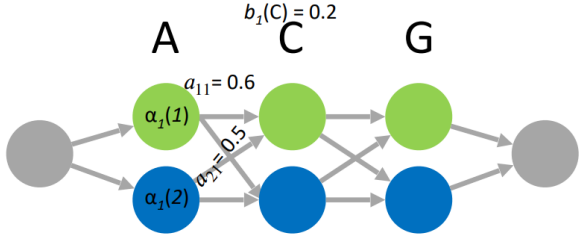
$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \le i \le N$$

2. Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \qquad 1 \le t \le T-1, 1 \le j \le N.$$

Build a dynamic programming table for these calculations

$b_1(C) = 0.2$



	$\alpha_1(1)$	$\alpha_2(1)$	
	A	C	G
State 1	0.32	0.0404	
State 2	0.02		

$\alpha_1(2)$

Scale

Forward

- Initialization

$$\ddot{\alpha}_1(i) = \alpha_1(i)$$
$$c_1 = \frac{1}{\sum_{i=1}^{N} \ddot{\alpha}_1(i)}$$
$$\hat{\alpha}_1(i) = c_1 \ddot{\alpha}_1(i)$$

- Induction

$$\ddot{\alpha}_t(i) = \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} b_i(O_t)$$
$$c_t = \frac{1}{\sum_{i=1}^{N} \ddot{\alpha}_t(i)}$$
$$\hat{\alpha}_t(i) = c_t \ddot{\alpha}_t(i)$$

	A	C	G
State 1			
State 2			
	c1	c2	c3

$$\hat{\alpha}_t(i) = \left(\prod_{\tau=1}^{t} c_\tau\right) \alpha_t(i).$$

$$C_t = \prod_{\tau=1}^{t} c_\tau$$

$$\log[P(O|\lambda)] = -\sum_{t=1}^{T} \log c_t.$$

Relevant (detailed!) slide decks from CX (2022 TA):

http://bozeman.genome.washington.edu/compbio/mbt599_2022/TA_discussion/class15.pdf
http://bozeman.genome.washington.edu/compbio/mbt599_2022/TA_discussion/class16.pdf
http://bozeman.genome.washington.edu/compbio/mbt599_2022/TA_discussion/class17.pdf
http://bozeman.genome.washington.edu/compbio/mbt599_2022/TA_discussion/class18.pdf

Homework 8 Overview

Underflow – Important!

COURSE-RELATED MATERIALS:

- Math Notation
- Biological Review Slides: Gene and genome structure in prokaryotes and eukaryotes and characteristics of sequence data; Genbank and other sequence databases.
- Nature paper on human genome sequence
- Nature paper on mouse genome sequence
- Siepel et al. paper on PhyloHMMs & sequence conservation
- Rabiner tutorial on HMMs
- HMM scaling tutorial (Tobias Mann)

Problem: numbers too small to be stored in a variable

Solutions:

- Scale weights to be close to 1
  - affects all paths by same constant factor, which can be multiplied back later

- Use log weights, so can add instead of multiplying
  - Ex: Instead of 0.0001 * 0.0002, you can do: log(0.0001) + log(0.0002)

What about when you need to sum probabilities in logspace?

- See this blogpost for a solution or Tobias Mann
- https://gasstationwithoutpumps.wordpress.com/2014/05/06/sum-of-probabilities-in-log-prob-space/

Homework 8 Overview

Underflow – Important!

COURSE-RELATED MATERIALS:

- Math Notation
- Biological Review Slides: Gene and genome structure in prokaryotes and eukaryotes and characteristics of sequence data; Genbank and other sequence databases.
- Nature paper on human genome sequence
- Nature paper on mouse genome sequence
- Siepel et al. paper on PhyloHMMs & sequence conservation
- Rabiner tutorial on HMMs
- HMM scaling tutorial (Tobias Mann)

Option 1: Rabiner HMM Scaling

A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition

LAWRENCE R. RABINER, FELLOW, IEEE

Although initially introduced and studied in the late 1960s and early 1970s, statistical methods of Markov source or hidden Markov modeling have become increasingly popular in the last several years. There are two strong reasons why this has occurred. First the models are very rich in mathematical structure and hence can form the theoretical basis for use in a wide range of applications. Second the models, when applied properly, work very well in practice for several important applications. In this paper we attempt to carefully and methodically review the theoretical aspects of this type of statistical modeling and show how they have been applied to selected problems in machine recognition of speech.

I. INTRODUCTION

Real-world processes generally produce observable outputs which can be characterized as signals. The signals can be discrete in nature (e.g., characters from a finite alphabet, quantized vectors from a codebook, etc.), or continuous in nature (e.g., speech samples, temperature measurements, music, etc.). The signal source can be stationary (i.e., its statistical properties do not vary with time), or nonstationary (i.e., the signal properties vary over time). The signals can be pure (i.e., coming strictly from a single source), or can be corrupted from other signal sources (e.g., noise) or by transmission distortions, reverberation, etc.

A problem of fundamental interest is characterizing such real-world signals in terms of signal models. There are several reasons why one is interested in applying signal models. First of all, a signal model can provide the basis for a theoretical description of a signal processing system which can be used to process the signal so as to provide a desired output. For example if we are interested in enhancing a speech signal corrupted by noise and transmission distortion, we can use the signal model to design a system which will optimally remove the noise and undo the transmission distortion. A second reason why signal models are important is that they are potentially capable of letting us learn a great deal about the signal source (i.e., the real-world process which produced the signal) without having to have the source available. This property is especially important when the cost of getting signals from the actual source is high. In this case, with a good signal model, we can simulate the source and learn as much as possible via simulations. Finally, the most important reason why signal models are important is that they often work extremely well in practice, and enable us to realize important practical systems—e.g., prediction systems, recognition systems, identification systems, etc., in a very efficient manner.

These are several possible choices for what type of signal model is used for characterizing the properties of a given signal. Broadly one can dichotomize the types of signal models into the class of deterministic models, and the class of statistical models. Deterministic models generally exploit some known specific properties of the signal, e.g., that the signal is a sine wave, or a sum of exponentials, etc. In these cases, specification of the signal model is generally straightforward; all that is required is to determine (estimate) values of the parameters of the signal model (e.g., amplitude, frequency, phase of a sine wave, amplitudes and rates of exponentials, etc.). The second broad class of signal models is the set of statistical models in which one tries to characterize only the statistical properties of the signal. Examples of such statistical models include Gaussian processes, Poisson processes, Markov processes, and hidden Markov processes, among others. The underlying assumption of the statistical model is that the signal can be well characterized as a parametric random process, and that the parameters of the stochastic process can be determined (estimated) in a precise, well-defined manner.

For the applications of interest, namely speech processing, both deterministic and stochastic signal models have had good success. In this paper we will concern ourselves strictly with one type of stochastic signal model, namely the hidden Markov model (HMM). (These models are referred to as Markov sources or probabilistic functions of Markov chains in the communications literature.) We will first review the theory of Markov chains and then extend the ideas to the class of hidden Markov models using several simple examples. We will then focus our attention on the three fundamental problems[1] for HMM design, namely: the

Manuscript received January 15, 1988; revised October 4, 1988.
The author is with AT&T Bell Laboratories, Murray Hill, NJ 07974-2070, USA.
IEEE Log Number 8825949.

[1]The idea of characterizing the theoretical aspects of hidden Markov modeling in terms of solving three fundamental problems is due to Jack Ferguson of IDA (Institute for Defense Analysis) who introduced it in lectures and writing.

0018-9219/89/0200-0257$01.00 © 1989 IEEE

Option 2: Tobias Mann & logs

Numerically Stable Hidden Markov Model Implementation

Tobias P. Mann

February 21, 2006

Abstract

Application of Hidden Markov Models to long observation sequences entails the computation of extremely small probabilities. These probabilities introduce numerical instability in the computations used to determine the probability of an observed sequence given a model, the most likely sequence of states, and the maximum likelihood model updates given an observation sequence. This paper explains how to handle small probabilities by working with the logarithms of probabilities, rather than resorting to alternative rescaling procedures.

1 Introduction

A practical issue in the use of Hidden Markov Models (HMMs) to model long sequences is the numerical scaling of conditional probabilities. Conditional probabilities must be computed in order to efficiently estimate the most probable sequence of states for a model given some data. Conditional probabilities are also computed in the process of estimating HMM parameters given training data. The numerical issue arising from the computation of conditional probabilities is that the probability of observing a long sequence given most models is extremely small. The use of these extremely small numbers in computations leads to numerical instability, and makes application of HMMs to genome length sequences challenging. There are two common approaches to dealing with small conditional probabilities. One approach is to rescale the conditional probabilities using carefully designed scaling factors. The other approach is to work with the logarithms of the conditional probabilities. This paper explains how to implement the second approach, and argues that computing with logarithms has advantages over the scaling factor approach.

A common approach to eliminating numerical problems is to rescale the conditional probabilities by computing scaling factors. These scaling factors are designed to bring various conditional probabilities to within a range easily handled by standard machine floating point representation. In an excellent tutorial on HMMs, Rabiner [1] notes the potential numerical problems and outlines how to compute scaling factors, but the applicaton of these scaling factors is incompletely specified. A slightly different scaling factor derivation is provided in another nice tutorial by Minka [2], but Minka doesn't provide details on computing Baum Welch updates to estimate HMM parameters. Rescaling approaches offer a solution to the numerical instability of computing with very small conditional probabilities, but they also make the resulting code more complicated to understand and debug.

An alternative to the rescaling approach is to compute the logarithms of the conditional probabilities [1]. Working with logarithms has the advantage that scaling constants can be eliminated and Baum Welch parameter updates do not need to be re-derived; in addition,

[1]This is suggested on page 78 of Durbin et al [3]

Homework 8 Overview

Notes for debugging

1. Try calculating some simple forward and backward probabilities by hand to check your algorithm

2. **The likelihood at each iteration should increase; if it decreases, then you have a bug**

3. Have a print statement in your program to keep track of iterations as your program is running. The assignment will provide an estimate on the number of iterations to converge.

Homework 8 Questions ?

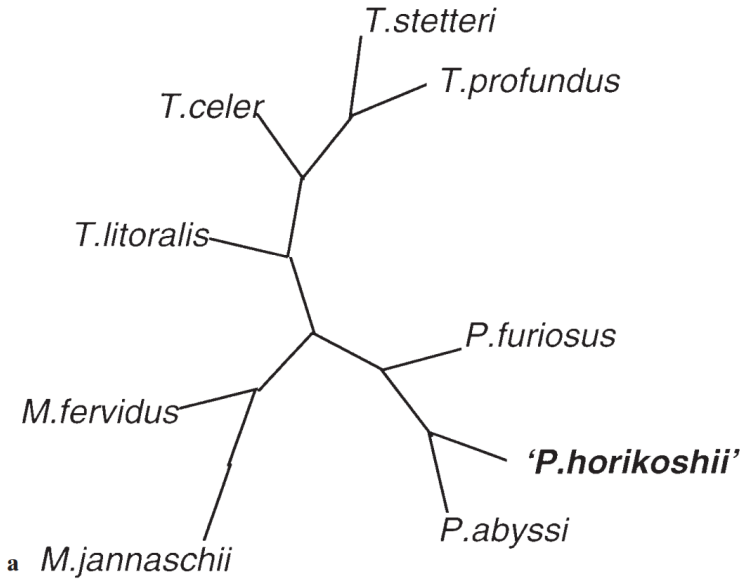
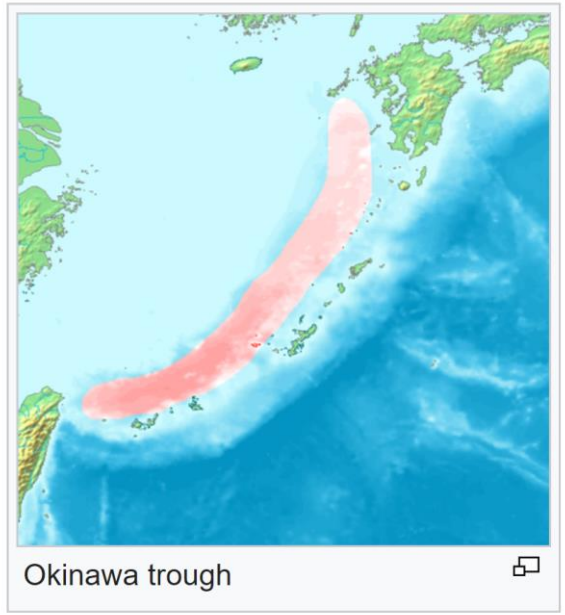
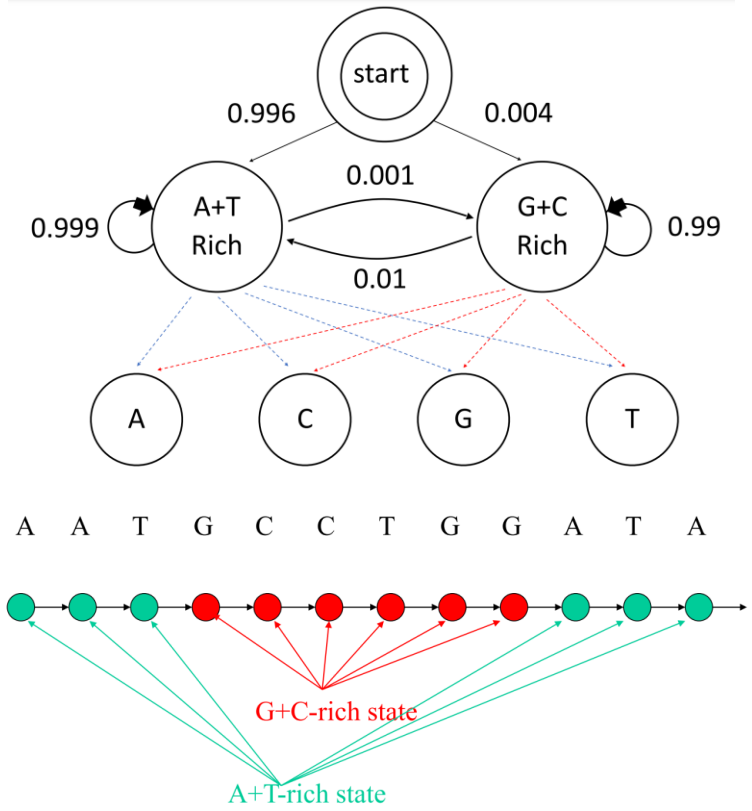
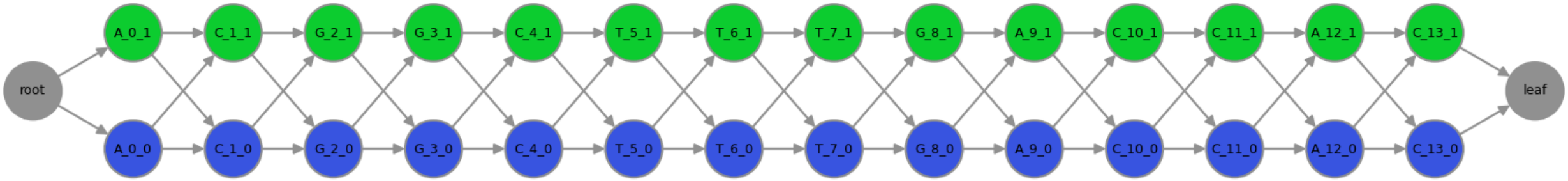


Fig. 3. Phylogenetic trees representing the relatedness of *P. horikoshii*





ACGGCTTTGACCAC



. . .