Statistical Computing

Hidden Markov Models for Bioinformatics

- Part V -

Uwe Menzel, 2011

uwe.menzel@matstat.org

www.matstat.org

Contents Part V

Parameter estimation for Hidden Markov Models

$$P(x,\pi) = a_{0\pi_1} \cdot \prod_{i=1}^{L} e_{\pi_i}(x_i) \cdot a_{\pi_i \pi_{i+1}}$$

$$a_{\pi_i \pi_{i+1}}$$

$$e_{\pi_i}(x_i)$$

?

Parameter estimation using training sets

If a large number of training sets with known state paths are available, parameters can be estimated by counting the number of transitions and emissions:

$$a_{kl} = \frac{A_{kl}}{\sum_l A_{kl}} \quad \text{with } k, l \in \{A^+, C^+, G^+, T^+, A^-, C^-, G^-, T^-\}$$

A_{kl}: counts of transitions from state k to state l in the training sets

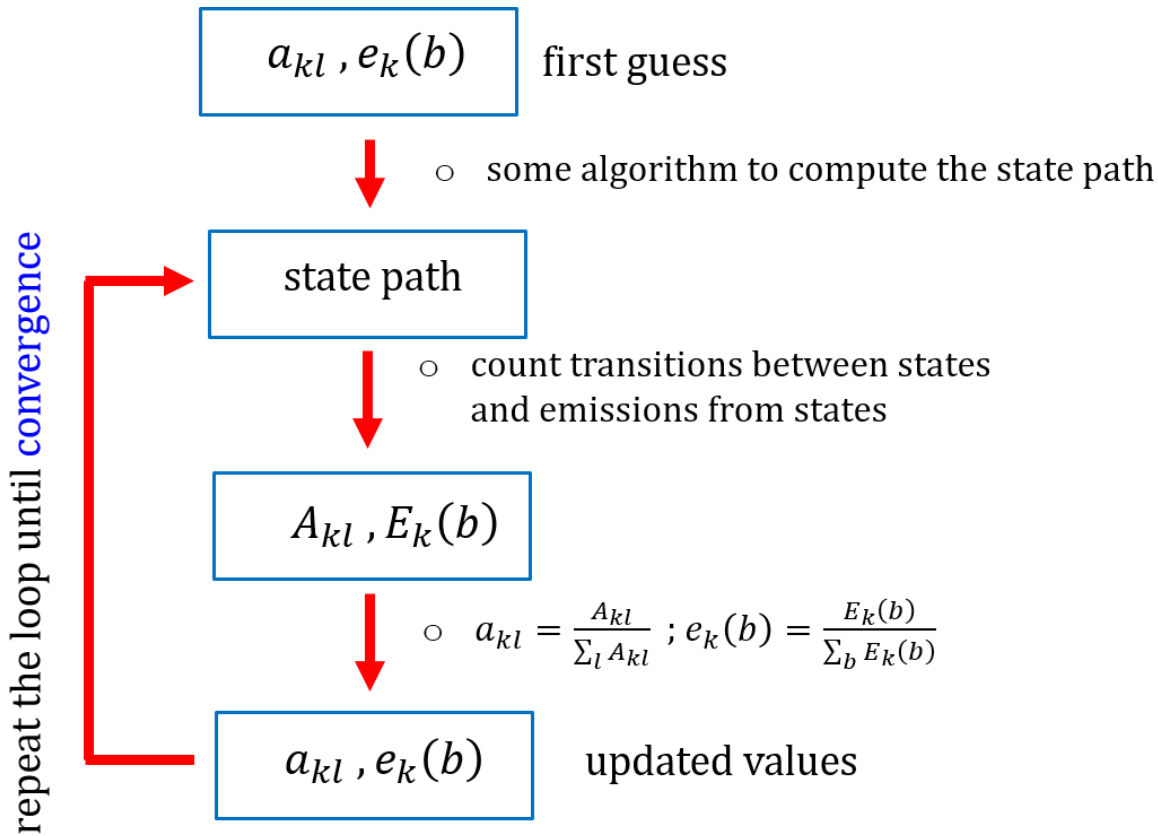
$$e_k(b) = \frac{E_k(b)}{\sum_b E_k(b)} \quad \text{with } k \in \{A^+, C^+, G^+, T^+, A^-, C^-, G^-, T^-\} \text{ and } b \in \{A, C, G, T\}$$

$E_k(b)$: counts of emissions from state k to symbol b in the training sets

- The a_{kl} and the $e_k(b)$ are Maximum-Likelihood estimators
- Overfitting can occur when the training sets are insufficient (insufficient number; biased to some particular type of observations)
- Pseudocounts can be addad to the counts in order to incorporate prior knowledge in the parameter estimation (Bayesian approach: the more pseudocounts, the more the parameters are driven towards the expectations arising from prior knowledge)

Parameter estimation without training sets

- If training sets with known state paths are not available, iterative schemes can be used to estimate parameter values for HMM's:
  - Baum-Welch algorithm
  - Viterbi trainig



Baum-Welch algorithm

The Baum-Welch algorithm is a method for the estimation of parameters in Hidden Markov Models. The algorithm is a flavour of Expectation-Maximisation (EM) (see matstat.org). The algorithm attempts to find those transition- and emission probabilities $\theta = (A, E)$ that maximise the likelihood $\mathcal{L}(\theta) = P(x \mid \theta)$. The forward-backward algorithm is utilized in order to reach that goal. The forward- and backward variables were introduced in part 4 :

$f_k(i) = P\left(x_1, x_2, \ldots, x_i, \pi_i = k\right)$	forward variables, definition
$b_k(i) = P\left(x_{i+1}, x_{i+2}, x_L \mid \pi_i = k\right)$	backward variables, definition
$a_{kl} = P\left(\pi_{i+1} = l \mid \pi_i = k\right)$	emission probabilities, definition

The Baum-Welch algorithm starts with the probability of the transition $\pi_i = k$ to $\pi_{i+1} = l$, given the observations x:

$$P\left(\pi_i = k, \pi_{i+1} = l \mid x\right)$$

Then, the expectation of this random variable is estimated by summing over all positions i and over all training sets. First, the above probability will be expressed with the help of the forward- and backward variables →

$$P(\pi_i = k, \pi_{i+1} = l \mid x) = \frac{1}{P(x)} \cdot P(x_1, \ldots, x_L, \pi_i = k, \pi_{i+1} = l)$$ (cond. probability)

(cond. probability)

$$= \frac{1}{P(x)} \cdot \underline{P(x_1, \ldots, x_i, \pi_i = k)} \cdot P(x_{i+1}, \ldots, x_L, \pi_{i+1} = l \mid \underline{x_1, \ldots, x_i}, \pi_i = k)$$

(omit, Markov property)

$$= \frac{1}{P(x)} \cdot f_k(i) \cdot P(x_{i+1}, \ldots, x_L, \pi_{i+1} = l \mid \pi_i = k)$$

(cond. probability)

$$= \frac{1}{P(x)} \cdot f_k(i) \cdot P(x_{i+1}, \ldots, x_L \mid \pi_{i+1} = l, \underline{\pi_i = k}) \cdot \underline{P(\pi_{i+1} = l \mid \pi_i = k)}$$

Markov

$$= \frac{1}{P(x)} \cdot f_k(i) \cdot P(x_{i+1}, \ldots, x_L \mid \pi_{i+1} = l) \cdot a_{kl}$$

(cond. probability)

$$= \frac{1}{P(x)} \cdot f_k(i) \cdot P(x_{i+2}, \ldots, x_L \mid x_{i+1}, \pi_{i+1} = l) \cdot \underline{P(x_{i+1} \mid \pi_{i+1} = l)} \cdot a_{kl}$$

$$= \frac{1}{P(x)} \cdot f_k(i) \cdot \underline{P(x_{i+2}, \ldots, x_L \mid \pi_{i+1} = l)} \cdot e_l(x_{i+1}) \cdot a_{kl}$$

$$= \frac{1}{P(x)} \cdot f_k(i) \cdot b_l(i+1) \cdot e_l(x_{i+1}) \cdot a_{kl}$$

Baum-Welch algorithm

In summary, the sought-after probability is:

$$P\left(\pi_i = k, \pi_{i+1} = l \mid x\right) = \frac{1}{P(x)} \cdot f_k(i) \cdot b_l(i+1) \cdot e_l\left(x_{i+1}\right) \cdot a_{kl}$$

The expected value of this random variable can be found by summing over all transitions, i.e. over all i:

$$A_{kl} = \frac{1}{P(x)} \cdot \sum_{i=0}^{L-1} f_k(i) \cdot b_l(i+1) \cdot e_l\left(x_{i+1}\right) \cdot a_{kl}$$

If more than one training sequence is available, all sequences are incorporated into the sum.

Next, recall from part 4 (posterior probabilities) the probability of having state k at position i, given the observed sequence:

$$P\left(\pi_i = k \mid x\right) = \frac{1}{P(x)} \cdot f_k(i) \cdot b_k(i)$$

Baum-Welch algorithm

Probability that we have state k at position i, given the observed sequence:

$$P\left(\pi_i = k \mid x\right) = \frac{1}{P(x)} \cdot f_k(i) \cdot b_k(i)$$

To get the expected number of emissions to symbol b outgoing from this state, we simply count the number of b-emissions from that state $\pi_i = k$:

$$E_k(b) = \frac{1}{P(x)} \cdot \sum_{i \mid x_i = b} f_k(i) \cdot b_k(i)$$

The iteration cycle for the Baum-Welch algorithm can be illustrated in the following scheme:

Baum-Welch iteration

$a_{kl}\,, e_k(x_i)$ random initialisation, or 1st guess

- forward and backward algorithm)

$f_k(i)\,;\, b_k(i)\,;\, P(x)$

- see formulas above

$P(\pi_i = k, \pi_{i+1} = l \mid x)\,;\, P(\pi_i = k \mid x)$

- see formulas above, expectations (sums)

$A_{kl}\,, E_k(b)$

- $a_{kl} = \frac{A_{kl}}{\sum_l A_{kl}}\,;\, e_k(b) = \frac{E_k(b)}{\sum_b E_k(b)}$

$a_{kl}\,, e_k(b)$ updated values

repeat the loop until convergence

Uwe Menzel, 2011

Baum-Welch algorithm

C:\Users\Uwe\Desktop\TALKS_POSTERS\LECTURES\HMM-Talk am HKI\HMM_forward_CGCG.R - R Editor

```
## Baum-Welch algorithm, long sequence

library(HMM)
states = c("A+", "C+", "G+", "T+", "A-", "C-", "G-", "T-")
symbols = c("A", "C", "G", "T")
trans_prob = get(load("trans_prob_HMM.RData"))
emission_prob = get(load("emission_prob_HMM.RData"))
hmm = initHMM(states, symbols, transProbs = trans_prob,
                          emissionProbs = emission_prob)
observation = sample(c("A", "C", "G", "T"),500, replace = TRUE)
res <- baumWelch(hmm, observation)
res$hmm$transProbs
res$hmm$emissionProbs
plot(res$difference)
```

- a random DNA sequence was generated ($L = 500$)
- the arrays "reshmmtransProbs" and "reshmmemissionProbs" contain the calculated parameters (which maximise the likelihood)

Baum-Welch algorithm

C:\Users\Uwe\Desktop\TALKS_POSTERS\LECTURES\HMM-Talk am HKI\HMM_forward_CGCG.R - R Editor

```
## Baum-Welch algorithm, short seuqence

library(HMM)
source("baumWelch_patch.R")  # patched version, for short sequences
states = c("A+", "C+", "G+", "T+", "A-", "C-", "G-", "T-")
symbols = c("A", "C", "G", "T")
trans_prob = get(load("trans_prob_HMM.RData"))
emission_prob = get(load("emission_prob_HMM.RData"))
start_prob = c(0, 0.5, 0,0,0,0.5,0,0)
names(start_prob) = c("A+", "C+", "G+", "T+", "A-", "C-", "G-", "T-")
hmm = initHMM(states, symbols, startProbs = start_prob,
              transProbs = trans_prob, emissionProbs = emission_prob)
observation = c("C", "G", "C", "G")    # short observation
res <- baumWelch(hmm, observation)
res$hmm$transProbs
res$hmm$emissionProbs
res$difference
```

Uwe Menzel, 2011

- in short sequences, transitions/emissions from some state might not occur
- → all cells in a row of the corresponding matrix are zero → invalid stochastic matrix
- → use the patched version (" baumWelch_patch.R")
- the script must be sourced **after** loading the library

Viterbi training

Viterbi training is an alternative approach for the estimation of HMM parameters. Starting from some guess for the a_{kl} and $e_k(b)$, the most probable path π^* is calculated using the Viterbi algorithm (see part 3). Now, it is possible to count the number of transitions between states, A_{kl} and the number of emissions to each symbol b from the individual k-states, $E_k(b)$. Using that information, updated values of the parameters a_{kl} and $e_k(b)$ are computed (by normalisation). This is repeated until the relative error between previous and updated parameter values gets small.

Vierbi training is relatively easy and frequently used, although it does **not** find the maximum of the likelihood $\mathcal{L}(\theta) = P(x \mid \theta)$.

Viterbi training

$a_{kl}\ , e_k(b)$ — first guess

- most probable path (Viterbi algorithm)

state path

- count transitions between states and emissions from states

$A_{kl}\ , E_k(b)$

- $a_{kl} = \frac{A_{kl}}{\sum_l A_{kl}}\ ;\ e_k(b) = \frac{E_k(b)}{\sum_b E_k(b)}$

$a_{kl}\ , e_k(b)$ — updated values

repeat the loop until convergence

Viterbi training

C:\Users\Uwe\Desktop\TALKS_POSTERS\LECTURES\HMM-Talk am HKI\HMM_forward_CGCG.R - R Editor

```
## Viterbi training

library(HMM)
states = c("A+", "C+", "G+", "T+", "A-", "C-", "G-", "T-")
symbols = c("A", "C", "G", "T")
trans_prob = get(load("trans_prob_HMM.RData"))
emission_prob = get(load("emission_prob_HMM.RData"))
hmm = initHMM(states, symbols, transProbs = trans_prob,
                        emissionProbs = emission_prob)
observation = sample(c("A", "C", "G", "T"),500, replace = TRUE)
res <- viterbiTraining(hmm, observation)
res$hmm$transProbs
res$hmm$emissionProbs
res$difference
```

- a random DNA sequence was generated ($L = 500$)
- the arrays "reshmmtransProbs" and "reshmmemissionProbs" contain the calculated parameters

Viterbi training

C:\Users\Uwe\Desktop\TALKS_POSTERS\LECTURES\HMM-Talk am HKI\HMM_forward_CGCG.R - R Editor

```
## Viterbi training, short sequence

library(HMM)
source("viterbiTraining_patch.R")  # patched version, for short sequences
states = c("A+", "C+", "G+", "T+", "A-", "C-", "G-", "T-")
symbols = c("A", "C", "G", "T")
trans_prob = get(load("trans_prob_HMM.RData"))
emission_prob = get(load("emission_prob_HMM.RData"))
hmm = initHMM(states, symbols, transProbs = trans_prob,
                        emissionProbs = emission_prob)
observation = c("C", "G", "C", "G")    # short observation
res <- viterbiTraining(hmm, observation)
res$hmm$transProbs
res$hmm$emissionProbs
res$difference
```

- in short sequences, transitions/emissions from some state might not occur
- → all cells in a row of the corresponding matrix are zero → invalid stochastic matrix
- → use the patched version (" viterbiTraining_patch.R")
- the script must be sourced **after** loading the library