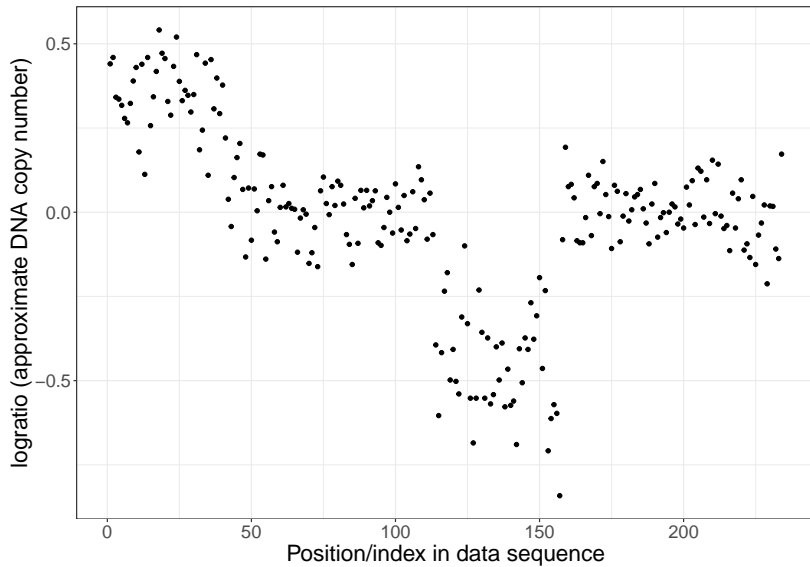


# Hidden Markov Models

Toby Dylan Hocking

## Background: detecting abrupt changes is important

Example from cancer diagnosis: breakpoints are associated with aggressive disease in neuroblastoma.

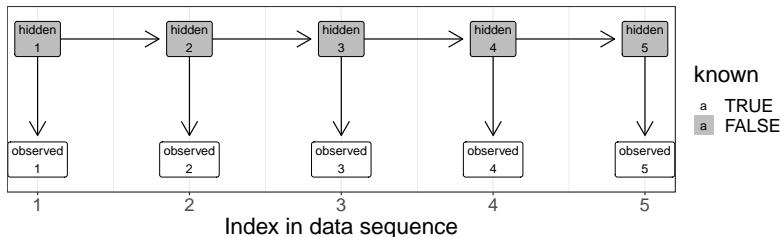


# Motivation for Hidden Markov Models (HMMs)

- ▶ Sometimes we have an interpretation / expectation of what the segments/clusters mean.
- ▶ For example in DNA copy number data the  $\text{logratio}=0$  means normal copy number (two copies – one from each parent), whereas higher  $\text{logratio}$  values indicate gain/amplification and lower values indicate loss/deletion.

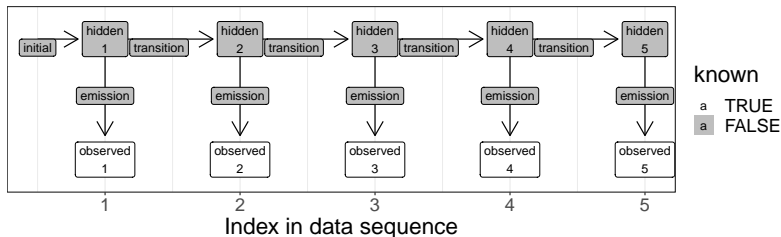
# HMM ideas

- ▶ Each observed data variable in the sequence has a corresponding un-observed (hidden) state variable.
- ▶ There are typically a finite number of possible values for each hidden state variable,  $k \in \{1, \dots, K\}$ .
- ▶ Markov assumption: first-order dependency (each hidden variable only depends on the previous hidden variable in the sequence).



# Parameters of HMM

- ▶ transition matrix:  $A \in [0, 1]^{K \times K}$  for  $K$  clusters. Each entry  $a_{ij}$  is the probability of transitioning from state  $i$  to state  $j$ .
- ▶ initial state distribution:  $\pi \in [0, 1]^K$  (prior weights).
- ▶ emission: likelihood of observing data  $y$  in state  $k$ ,  $b_k(y) = \text{NormalDensity}(y, \mu_k, \sigma^2) \in \mathbb{R}$ , parameterized by mean  $\mu_k$ , variance  $\sigma^2$  (or standard deviation =  $\text{sd} = \sigma$ ).
- ▶ These parameters are unknown in advance and must be learned from the data.
- ▶ Comparison with Gaussian Mixture Models: HMM has all of GMM parameters, plus transition matrix.

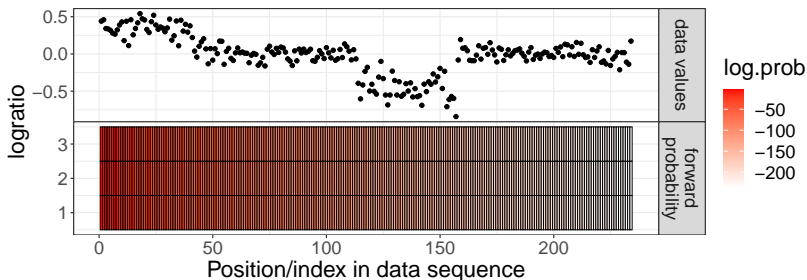


# Three problems and algorithms

- ▶ Evaluation of likelihood of a given data sequence and model parameters  $\Rightarrow$  forward algorithm.
- ▶ Decoding most likely sequence of hidden states given observed data and model parameters  $\Rightarrow$  Viterbi algorithm.
- ▶ Learning model parameters that maximize likelihood of a given data set  $\Rightarrow$  Baum-Welch algorithm.

## Forward algorithm

- ▶ Assume each state  $k \in \{1, \dots, K\}$  has an emission probability function  $b_k(o_t)$  for observed data  $o_t$  at time  $t$ .
- ▶ Let  $a_{kj}$  be a transition parameter (probability of going from state  $k$  to state  $j$ ).
- ▶ Initialization: for all states  $j \in \{1, \dots, K\}$ ,  $\alpha_1(j) = \pi_j b_j(o_1)$ .
- ▶ Then we can recursively compute the forward path probability,  $\alpha_t(j) = \sum_{k=1}^K \alpha_{t-1}(k) a_{kj} b_j(o_t)$ .
- ▶ Total log likelihood is summed over all states at the last data point in the sequence.



## The numerical underflow problem

- ▶ Probability  $p_t$  at each time  $t$  is in  $[0,1]$  so  $\prod_{t=1}^N p_t \rightarrow 0$ .
- ▶ For large enough  $N$  (several hundred) double precision arithmetic underflows (probability = 0 on computer although it is non-zero mathematically).

```
N.data <- seq(321, 325)
prob <- 0.1^N.data
data.table(N.data, prob, log.prob=log10(prob))
```

##	N.data	prob	log.prob
## 1:	321	9.980126e-322	-321.0009
## 2:	322	9.881313e-323	-322.0052
## 3:	323	9.881313e-324	-323.0052
## 4:	324	0.000000e+00	-Inf
## 5:	325	0.000000e+00	-Inf



## Implementation details to avoid numerical underflow

- ▶ Use a double precision number to store  $\log(\text{probability})$  value,  $\log p$ , instead of probability value,  $p$ .
- ▶ Instead of multiplying probability values, sum  $\log(\text{probability})$  values:  $\log(pq) = \log p + \log q$ .

```
N.data <- seq(321, 325)
log.prob <- N.data*log10(0.1)
data.table(N.data, log.prob, prob=10^log.prob)
```

##	N.data	log.prob	prob
## 1:	321	-321	9.980126e-322
## 2:	322	-322	9.881313e-323
## 3:	323	-323	9.881313e-324
## 4:	324	-324	0.000000e+00
## 5:	325	-325	0.000000e+00

## Implementation details to avoid numerical underflow

Instead of summing probability values, use log-sum-exp trick  
(subtract away  $m = \max\{\log p, \log q\}$  in exponent),  
 $p + q = e^{\log p} + e^{\log q}$ .

$$\log(p + q) = \log(e^{\log p} + e^{\log q}) = m + \log(e^{\log p - m} + e^{\log q - m})$$

```
data.table(log.p=-324, log.q=seq(-320, -326))
)[,
  m := ifelse(log.p < log.q, log.q, log.p)][,
  log.sum.exp := m + log10(10^(log.p-m)+10^(log.q-m))][,
  naive.sum := log10(10^log.p+10^log.q)][,]
```

##	log.p	log.q	m	log.sum.exp	naive.sum
## 1:	-324	-320	-320	-320.0000	-320.0000
## 2:	-324	-321	-321	-320.9996	-321.0009
## 3:	-324	-322	-322	-321.9957	-322.0052
## 4:	-324	-323	-323	-322.9586	-323.0052
## 5:	-324	-324	-324	-323.6990	-Inf
## 6:	-324	-325	-324	-323.9586	-Inf
## 7:	-324	-326	-324	-323.9957	-Inf

# Viterbi algorithm

- ▶ Assume each state  $k \in \{1, \dots, K\}$  has an emission probability function  $b_k(o_t)$  for observed data  $o_t$  at time  $t$ .
- ▶ Let  $a_{kj}$  be a transition parameter (probability of going from state  $k$  to state  $j$ ).
- ▶ Initialization: for all states  $j \in \{1, \dots, K\}$ ,  $v_1(j) = \pi_j b_j(o_1)$ .
- ▶ Then we can recursively compute the probability of the best sequence of hidden variables that ends at data point  $t$  in state  $j$ ,  $v_t(j) = \max_{k \in \{1, \dots, K\}} v_{t-1}(k) a_{kj} b_j(o_t)$ .
- ▶ Also need to store a matrix of best  $k$  values which achieved the max for every  $t, j$ .
- ▶ To compute best state sequence, first find  $k$  with  $\max v_N(k)$  then repeatedly examine previously stored best  $k$  values.

# Viterbi algorithm example for $K = 3$ states

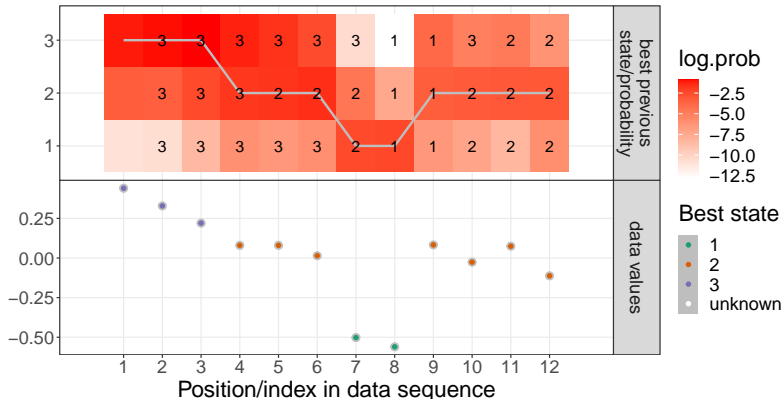
```
## List of 4
```

```
## $ mean      : num [1:3] -0.5 0 0.3
```

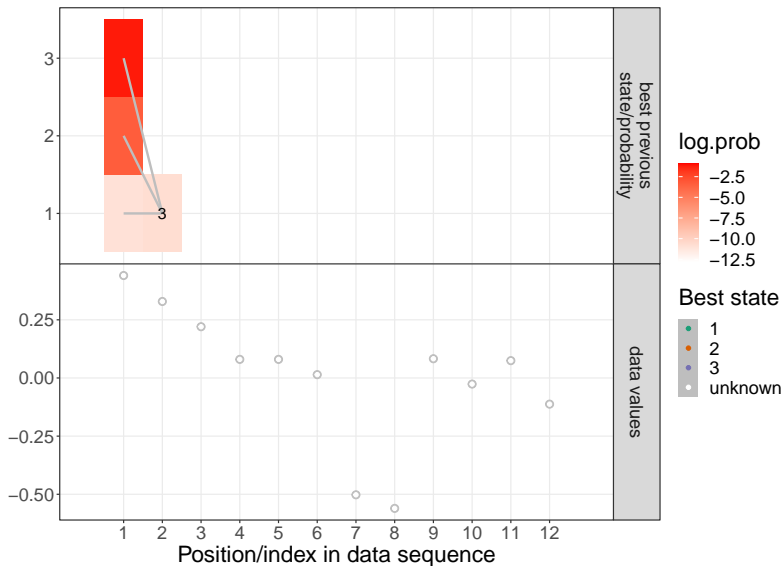
```
## $ sd        : num 0.2
```

```
## $ transition: num [1:3, 1:3] 0.6 0.2 0.2 0.2 0.6 0.2 0
```

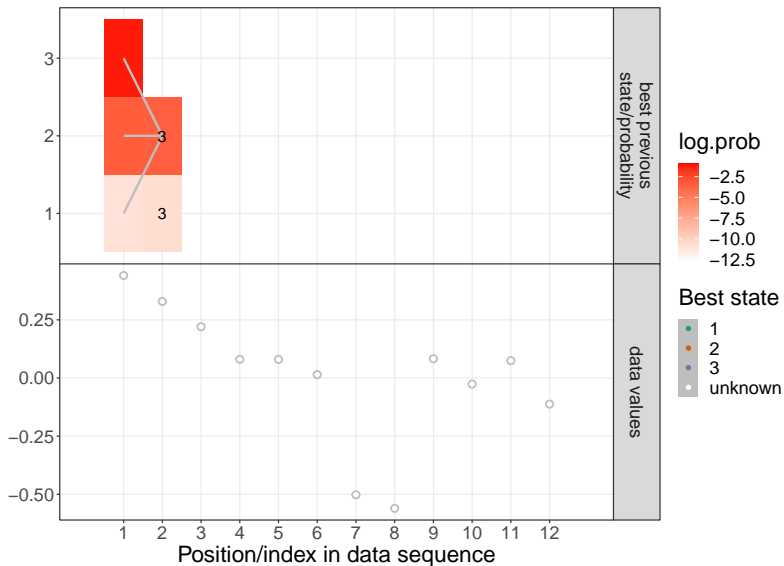
```
## $ initial   : num [1:3] 0.6 0.2 0.2
```



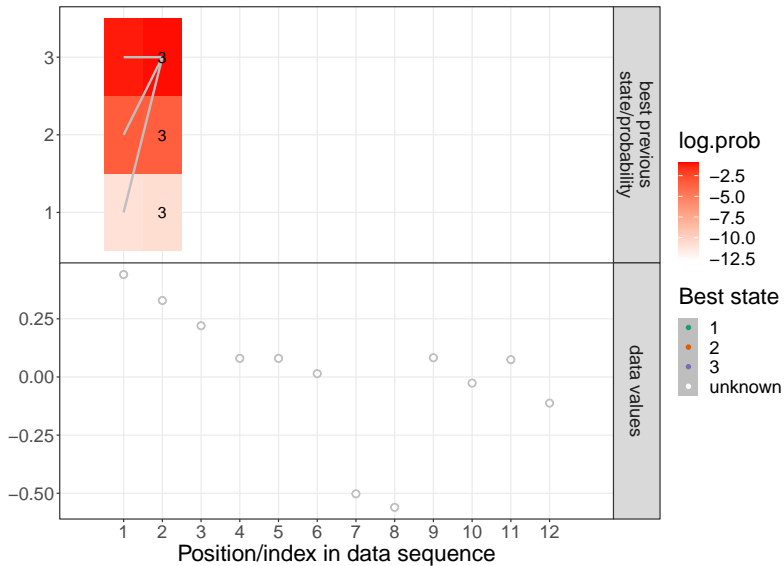
# Viterbi forward pass



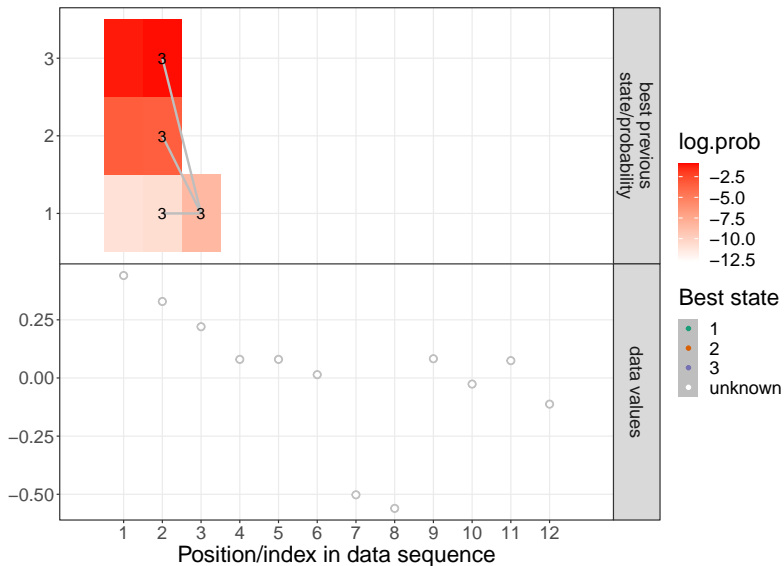
# Viterbi forward pass



# Viterbi forward pass

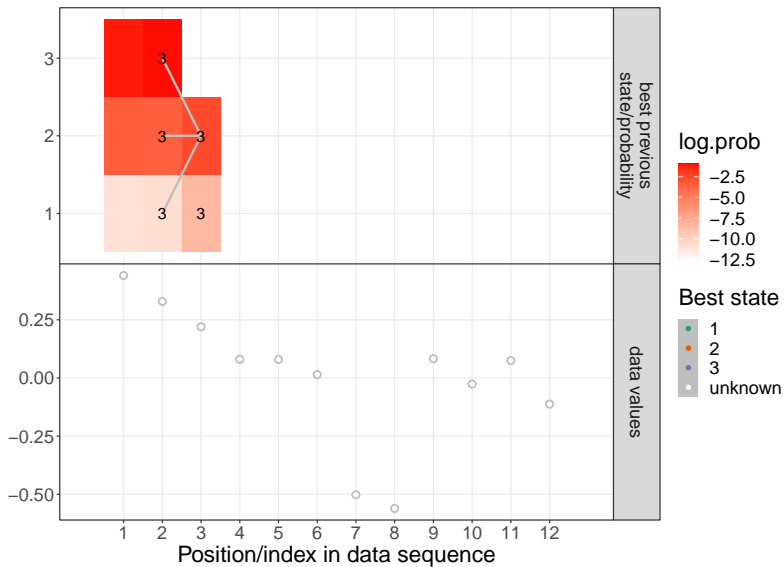


# Viterbi forward pass

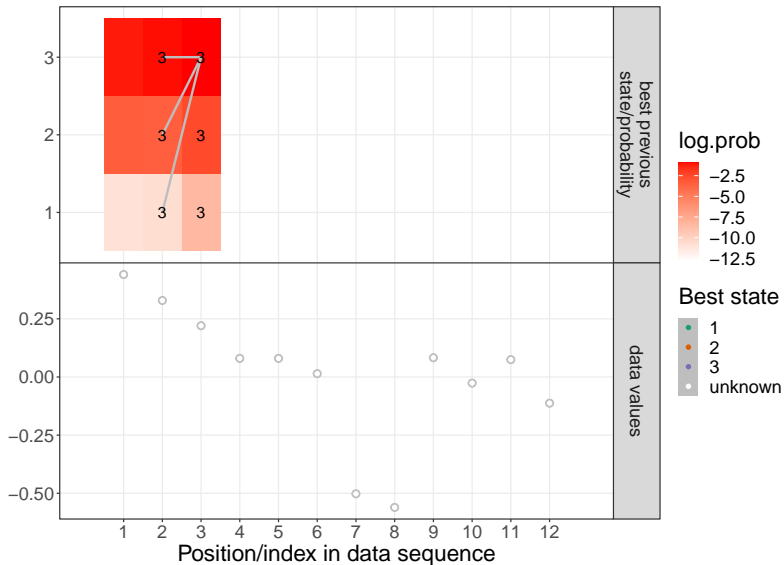




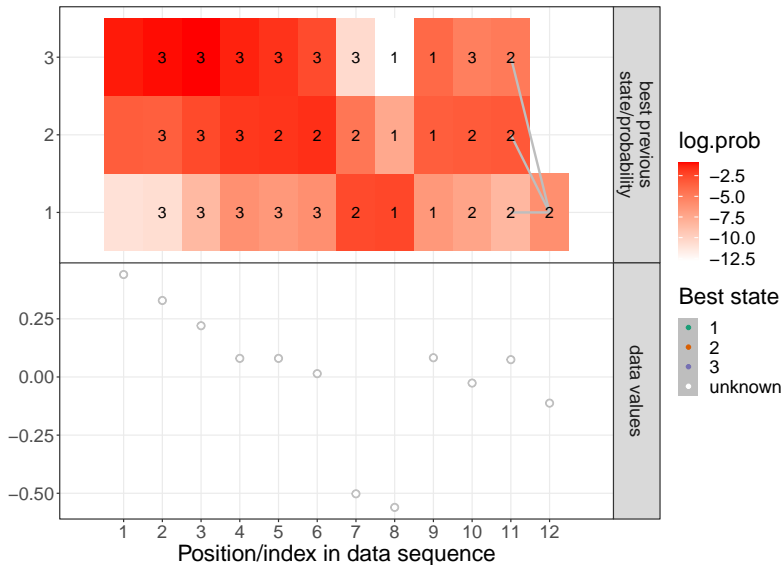
# Viterbi forward pass



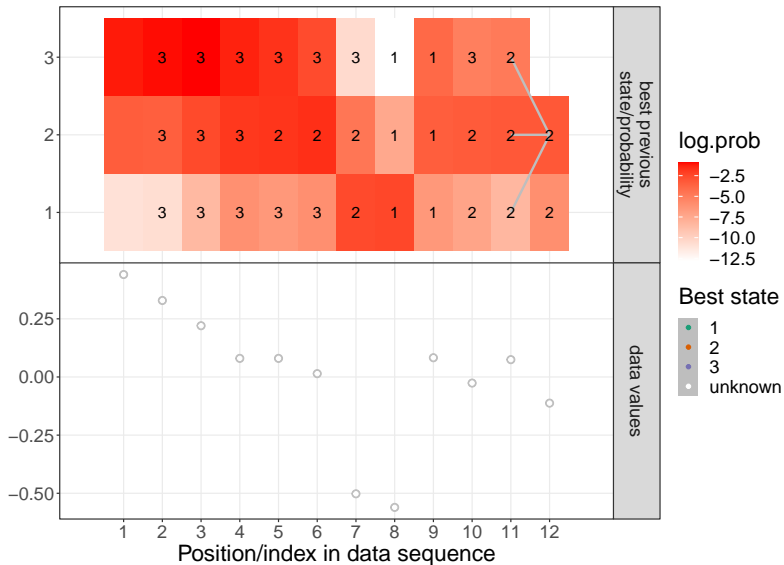
# Viterbi forward pass



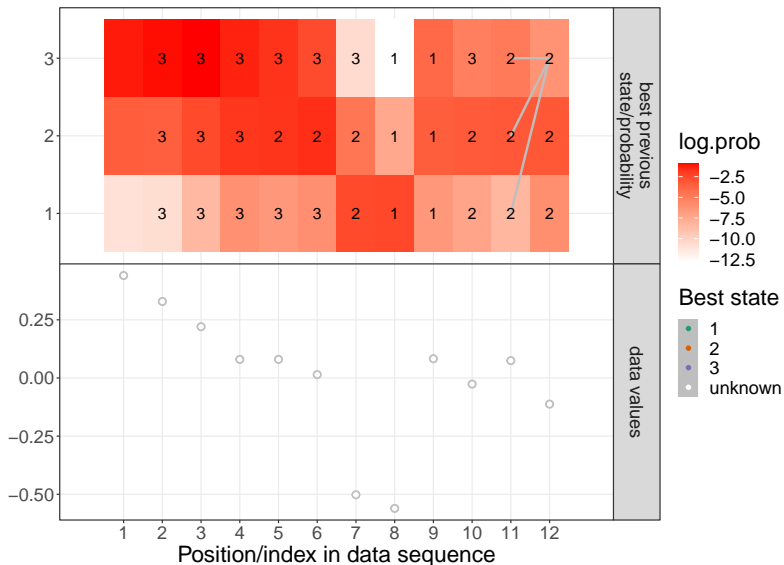
# Viterbi forward pass



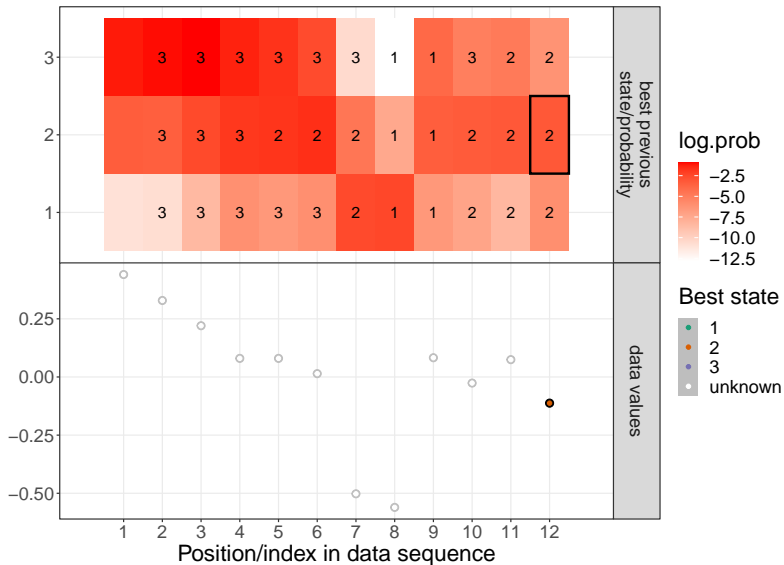
# Viterbi forward pass



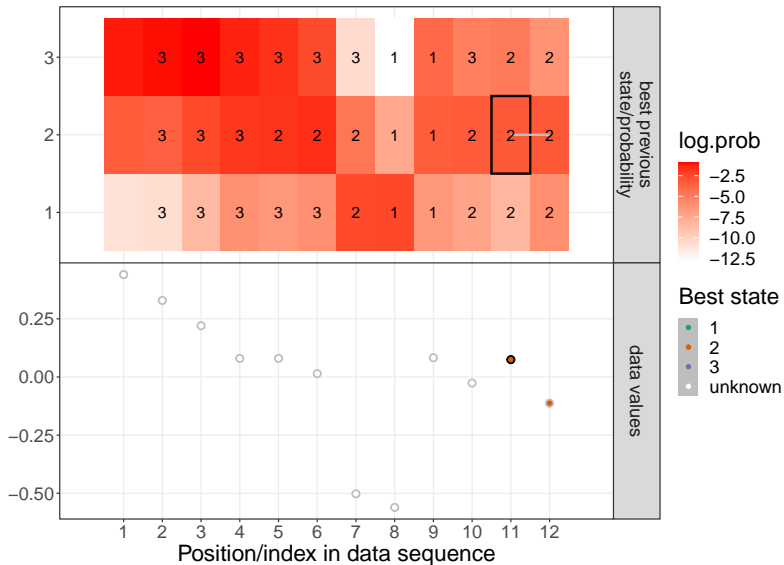
# Viterbi forward pass



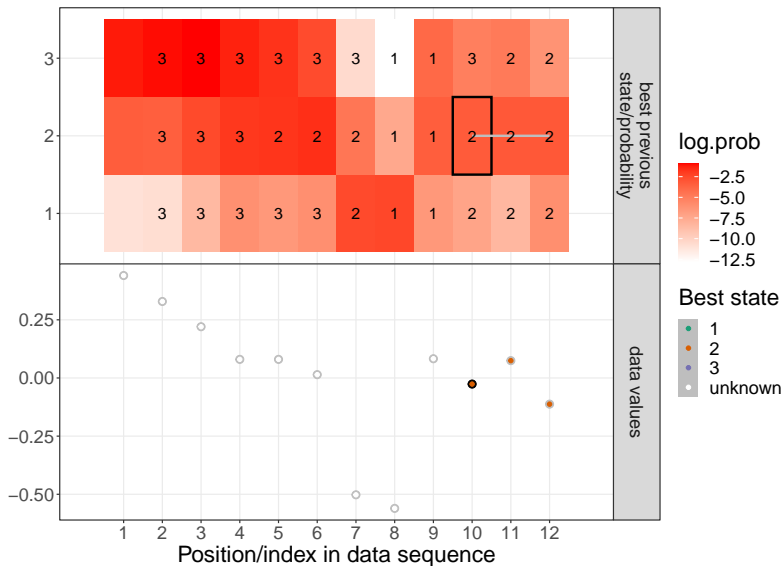
# Viterbi backtracking



# Viterbi backtracking

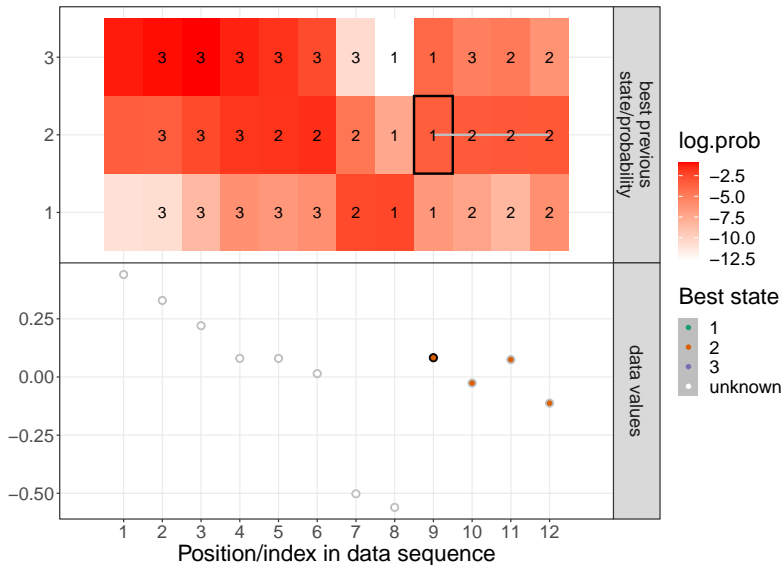


# Viterbi backtracking

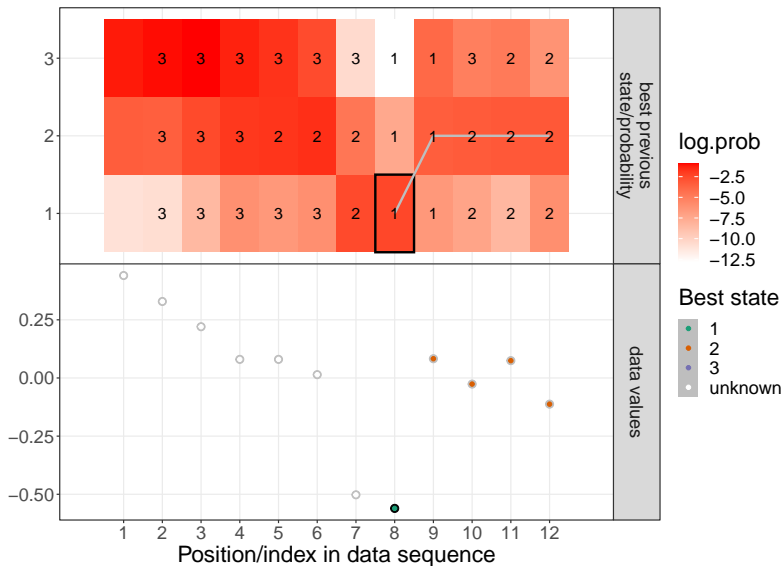




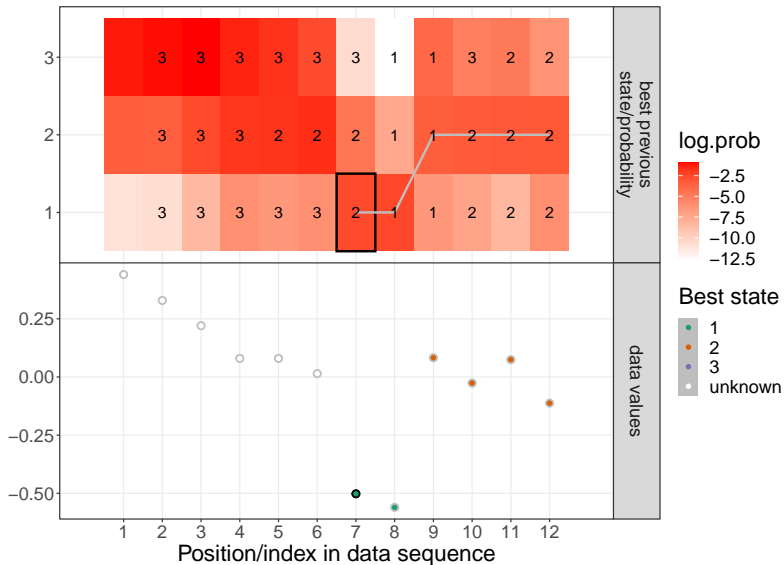
# Viterbi backtracking



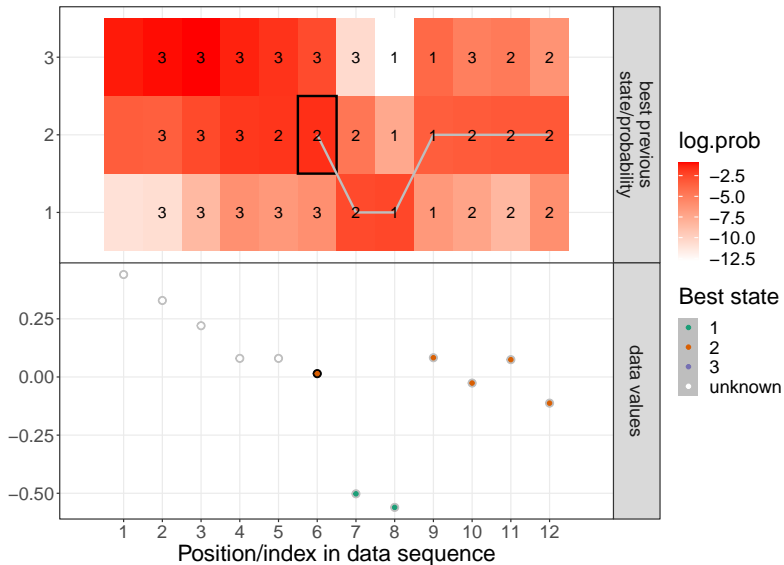
# Viterbi backtracking



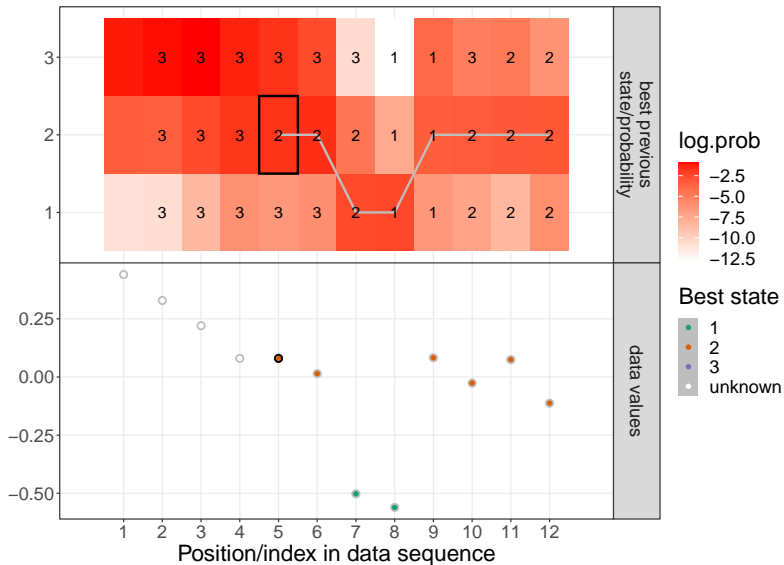
# Viterbi backtracking



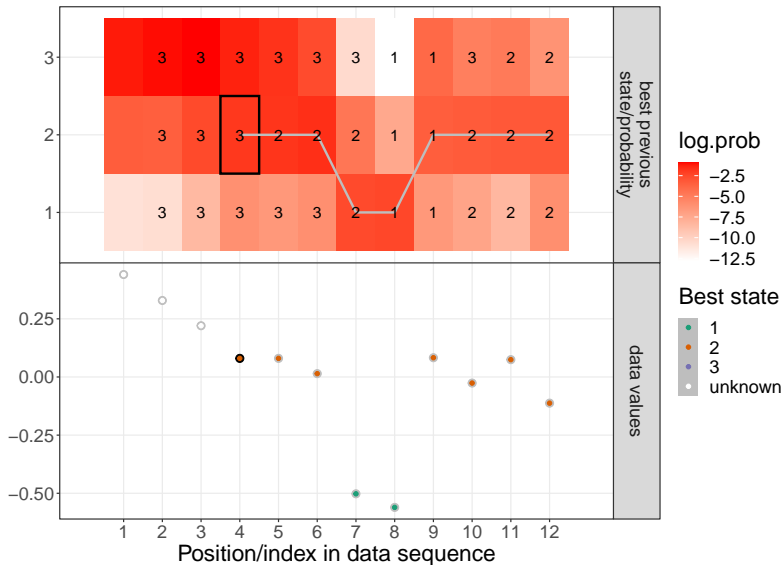
# Viterbi backtracking



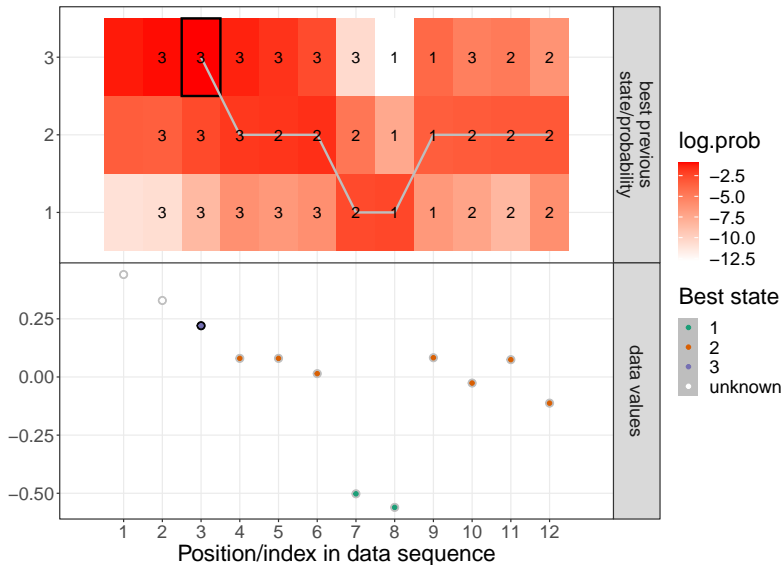
# Viterbi backtracking



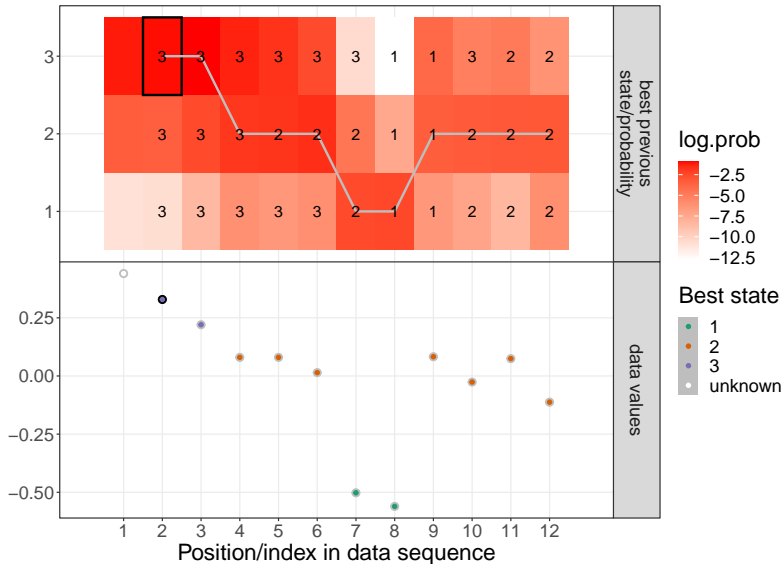
# Viterbi backtracking



# Viterbi backtracking

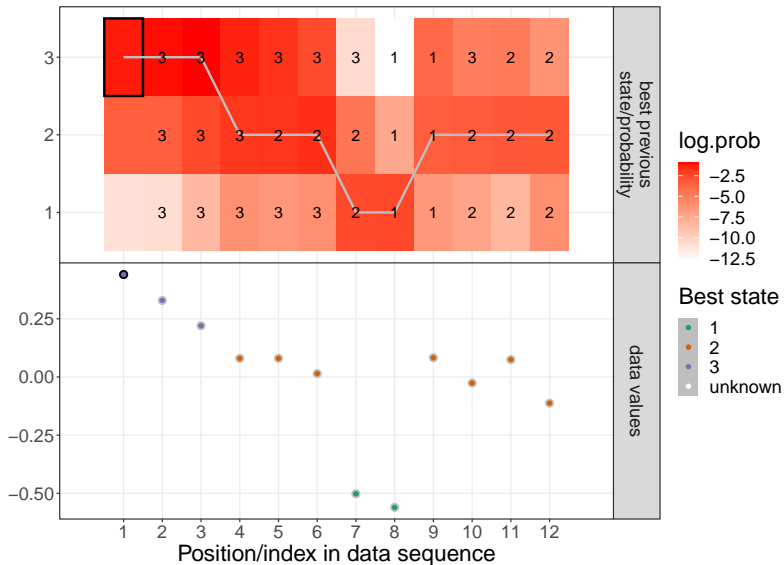


# Viterbi backtracking



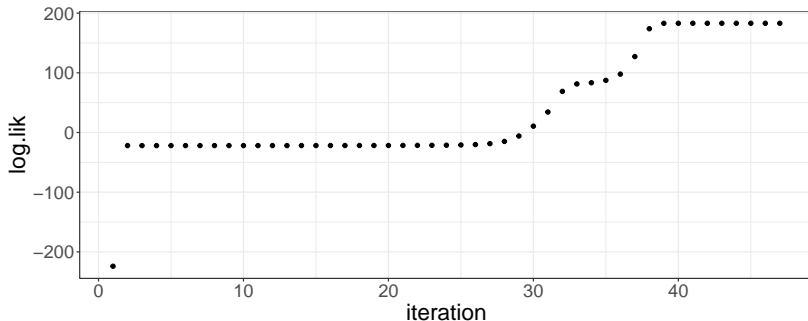


# Viterbi backtracking

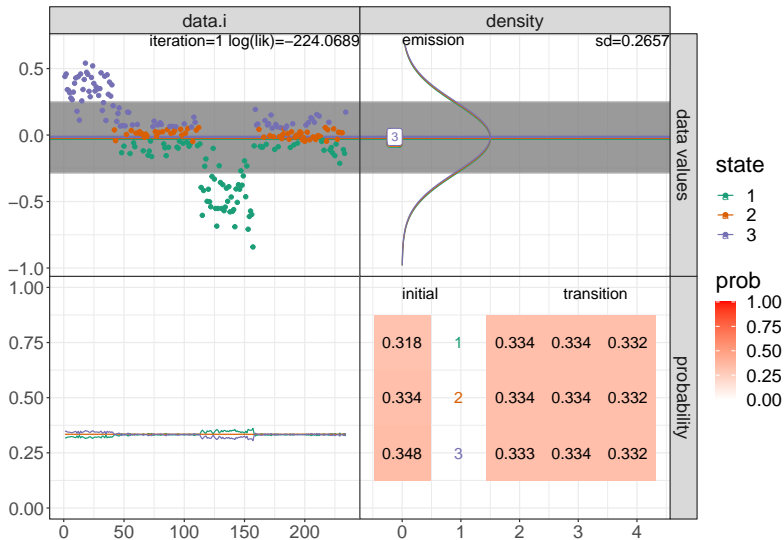


# Baum-Welch learning algorithm

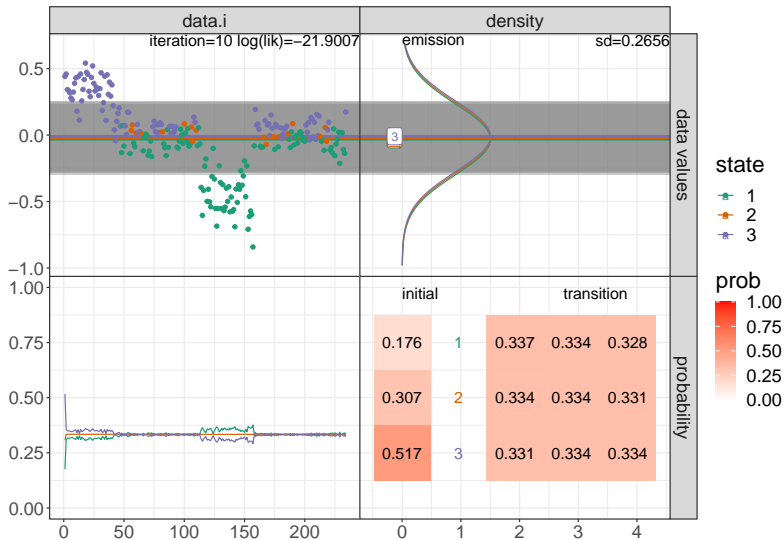
- ▶ Is an instance of Expectation-Maximization (EM), like the Gaussian Mixture Model learning algorithm.
- ▶ E step involves forward/backward passes over data sequences, to compute probability of each data point in each state.
- ▶ M step involves re-computing model parameters.
- ▶ Repeat until the log likelihood stops increasing.



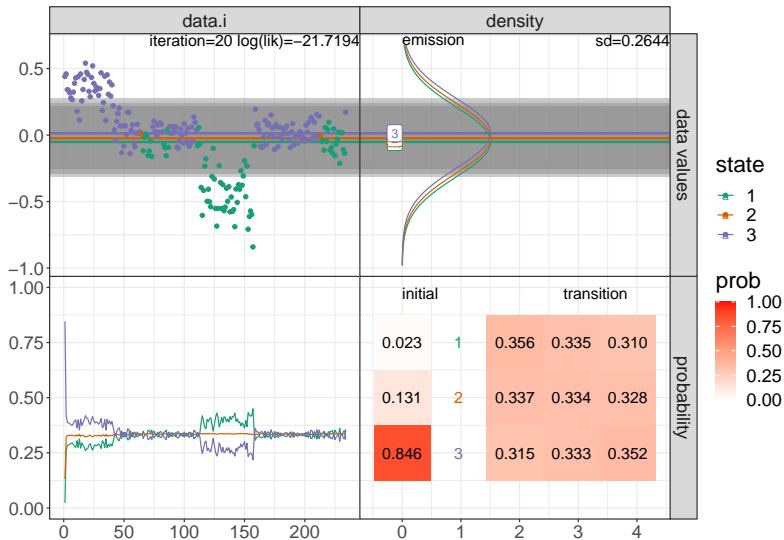
# Visualization of learning iterations



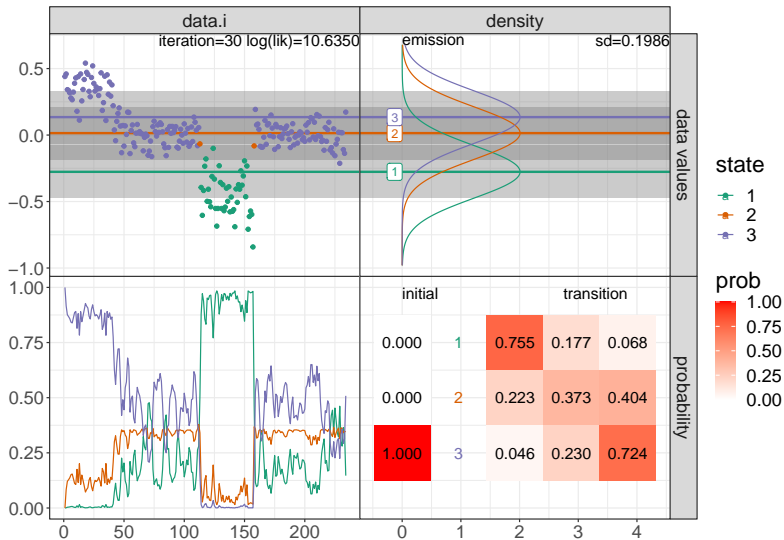
# Visualization of learning iterations



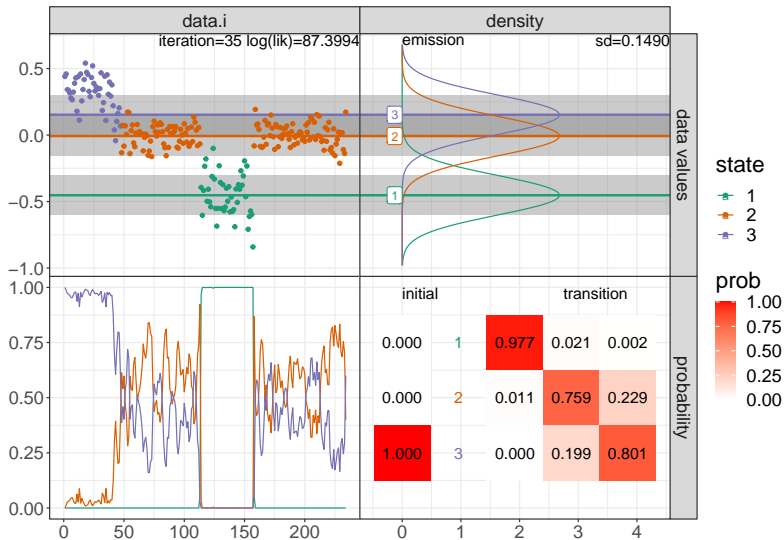
# Visualization of learning iterations



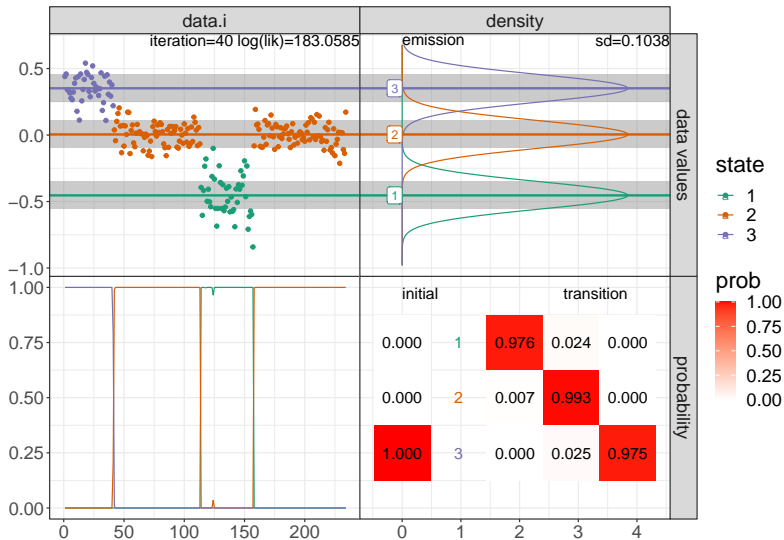
# Visualization of learning iterations



# Visualization of learning iterations

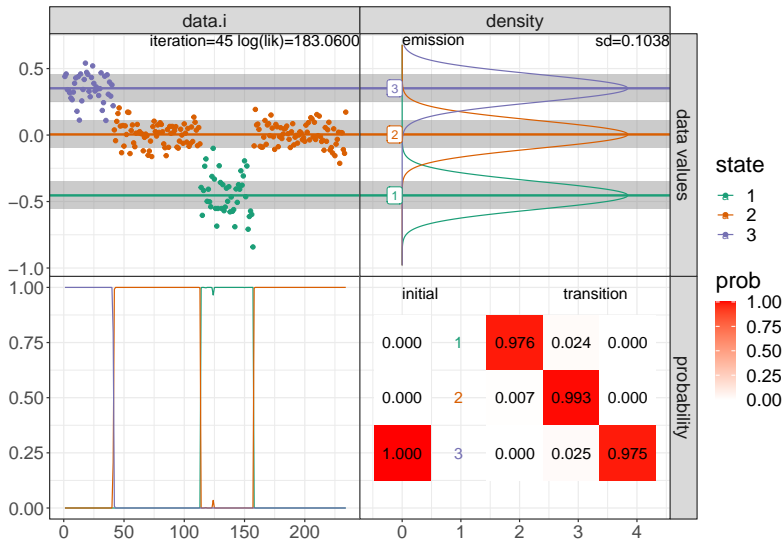


# Visualization of learning iterations



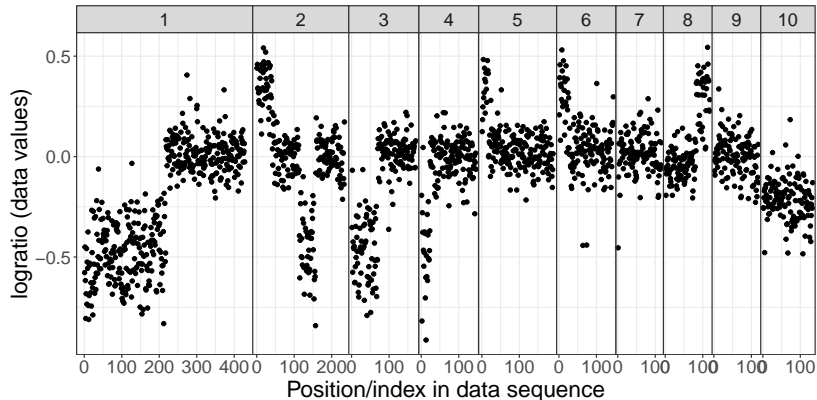


# Visualization of learning iterations

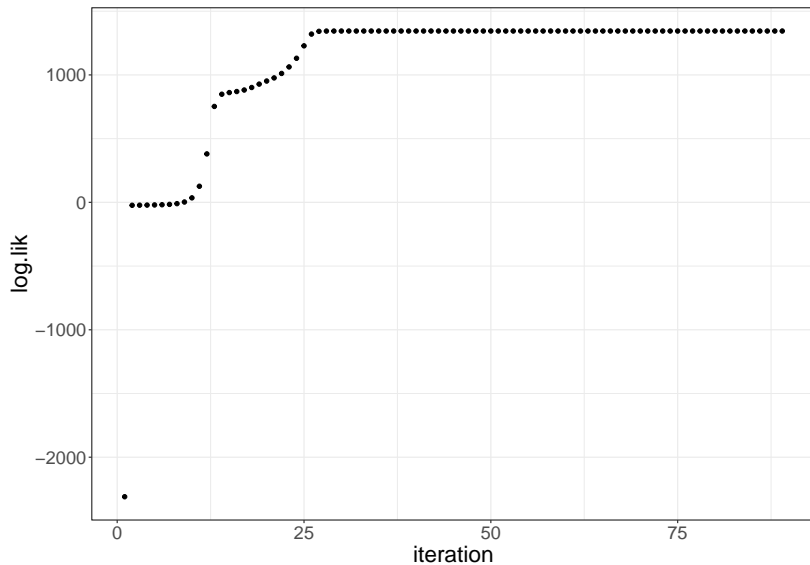


## Shared states between chromosomes on a profile?

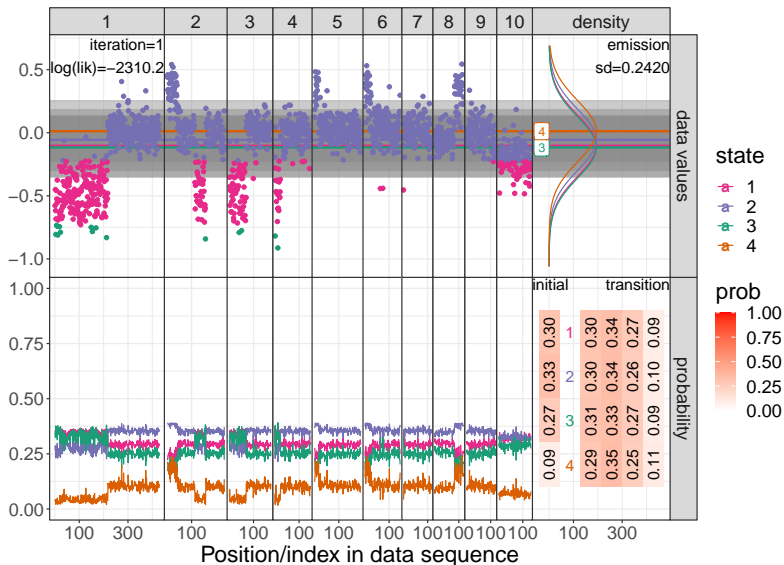
- ▶ Sometimes there are several data sequences which are assumed to have the same set of hidden states.
- ▶ In this case Baum-Welch can be used to fit HMM to all data at the same time (total log likelihood is summed over all data sequences).



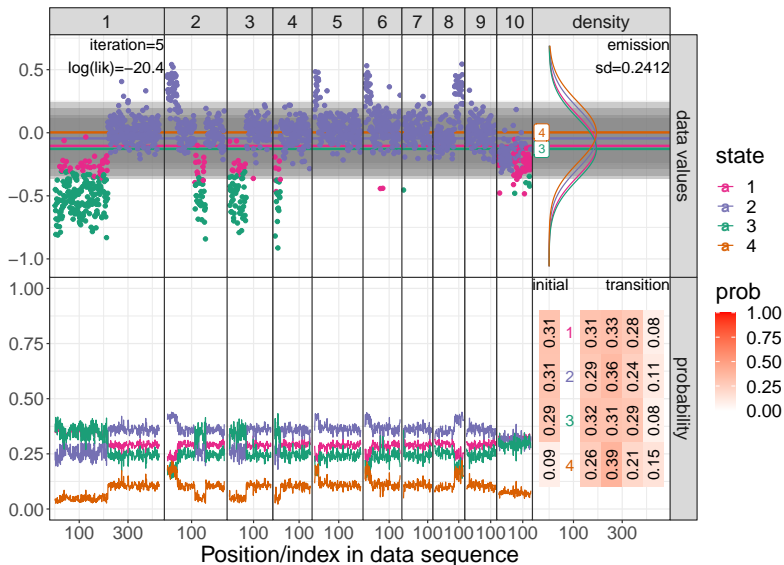
## HMM learned on whole profile



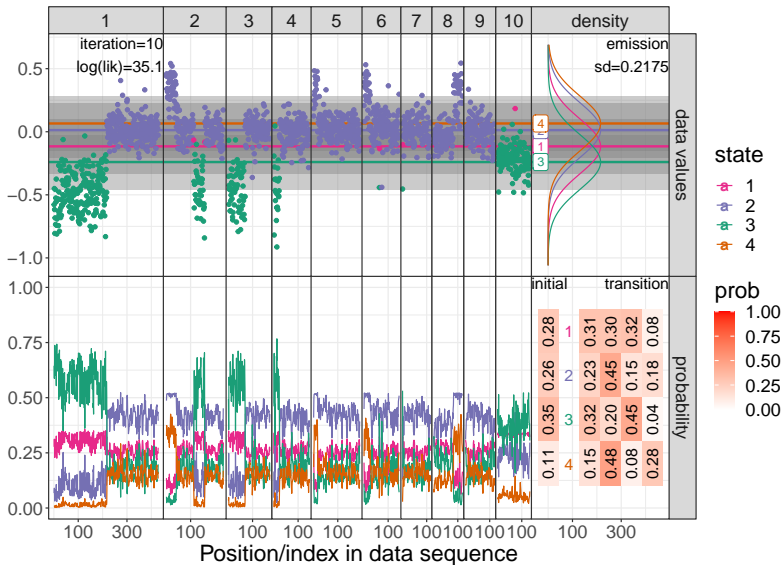
# Learning iterations using multiple sequences



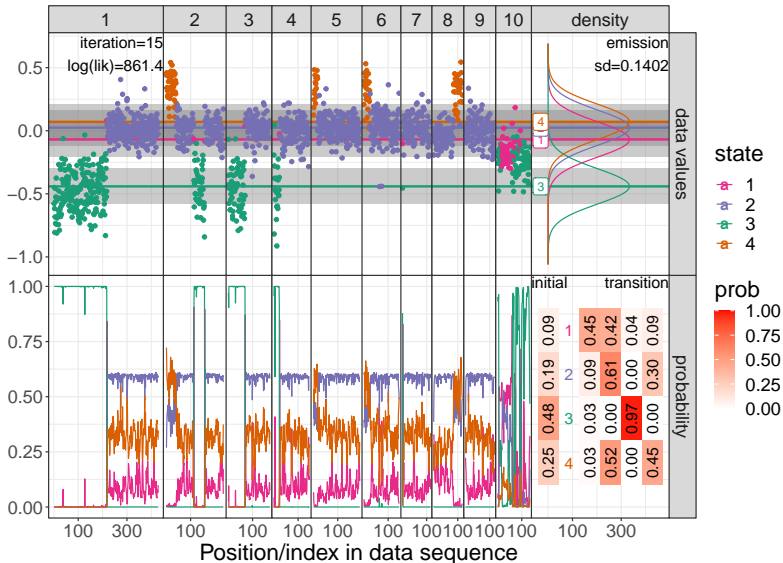
# Learning iterations using multiple sequences



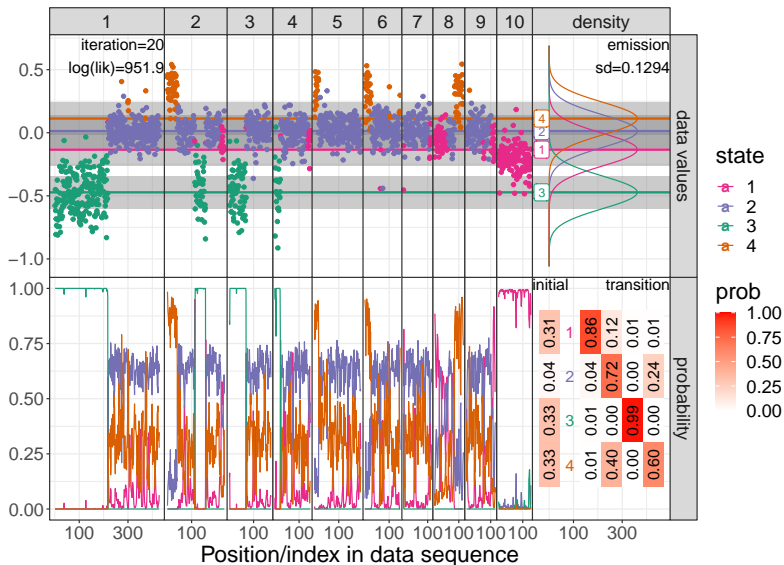
# Learning iterations using multiple sequences



# Learning iterations using multiple sequences

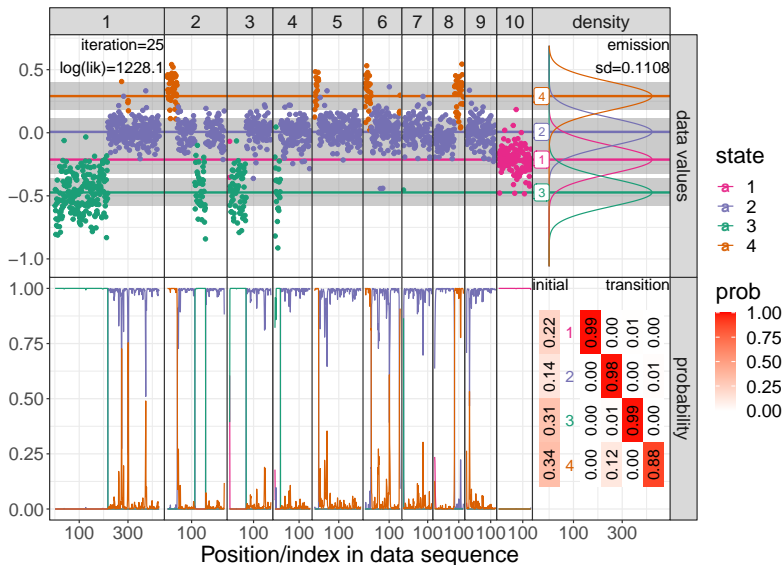


# Learning iterations using multiple sequences

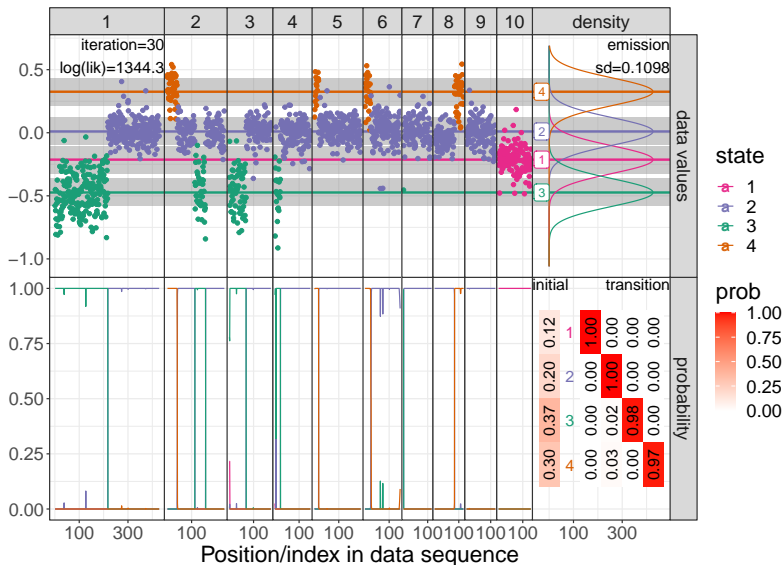




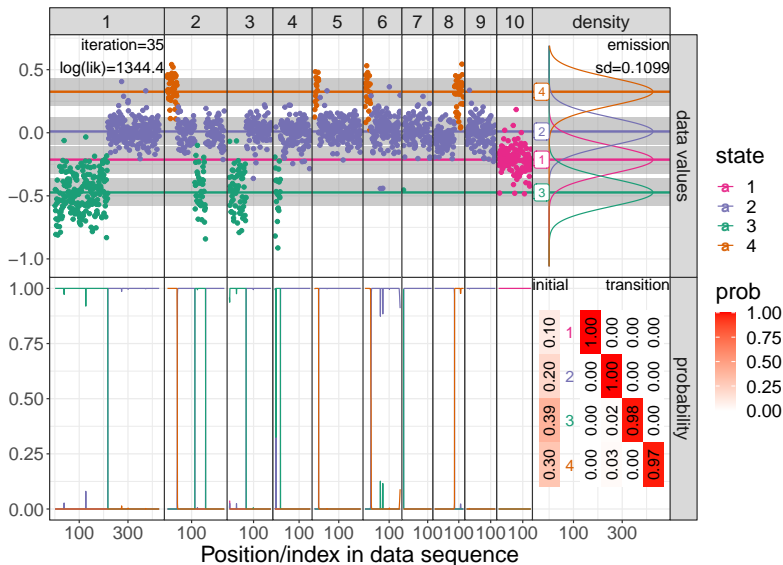
# Learning iterations using multiple sequences



# Learning iterations using multiple sequences



# Learning iterations using multiple sequences



## Time/space complexity

For  $K$  states and  $N$  data

- ▶ Forward/Viterbi:  $O(K^2N)$  time,  $O(K^2 + KN)$  space.
- ▶ Each iteration of Baum-Welch:  $O(K^2N)$  time and space.
- ▶ Some asymptotic speedups are possible in special cases, for example sparse transition matrices, see Murphy book.

## Comparison with other algorithms

- ▶ K-means and Gaussian mixture models also have cluster-specific parameters (mean, covariance, prior weight), but are not able to model sequential dependence.
- ▶ All segmentation models we studied had  $K$  mean parameters, and a single variance/sd parameter common to all segments.
- ▶ Binary/optimal segmentation require specification of number of segments/changepoints (always jump to a new mean parameter) whereas HMM requires number of hidden states (may jump to a previously visited mean parameter).
- ▶ Binary/optimal segmentation with  $K$  segments can be interpreted as an HMM with a constrained transition matrix ( $a_{ij} = 1$  if  $j = i + 1$  else 0: always jump to the next state, never jump back to a previous state).
- ▶ Binary/optimal segmentation log likelihood only uses emission probabilities, whereas HMM also includes initial/transition probabilities.

## Possible exam questions

- ▶ In the previous slides we saw  $K = 3$  or 4 clusters. How many parameters of each type are there to learn in each case? (assume common sd parameter as in slides)
- ▶ How many parameters if each cluster has its own sd parameter?
- ▶ In the previous slides the data have a single feature (logratio). How many parameters if there are  $P = 2$  real-valued features instead? (assume normal distribution with no constraints on covariance matrix)