Aalto University
School of Science

CS-E5865 Computational genomics

Autumn 2020, Lecture 4: Hidden Markov Models

Lecturer: Pekka Marttinen

Assistants: Alejandro Ponce de León, Zeinab Yousefi, Onur Poyraz

Our toolbox so far

1. Multinomial i.i.d model for sequences

$$P(s) = \prod_{i=1}^{n} p(\mathbf{s}(i)) = \prod_{x \in \mathcal{N}} p_x^{n(x,s)}$$

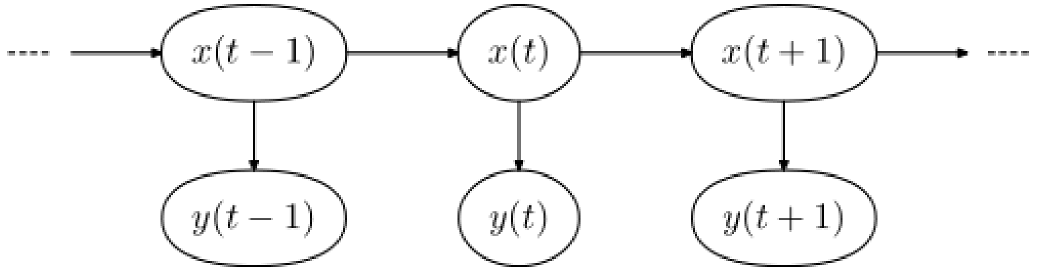
2. Markov Models for modeling local dependencies:

$$P(s) = P(s_1) \prod_{i=2}^{n} P(s_i | s_{i-1})$$

3. Dynamic programming for fast computation over sequences
4. Randomization for assessing statistical significance

Hidden Markov Models

- Hidden Markov Models (HMM) are the probabilistic model of choice for biological sequence analysis (both DNA and proteins)
- HMM combine multinomial and Markov sequence models and uses dynamic programming for computation

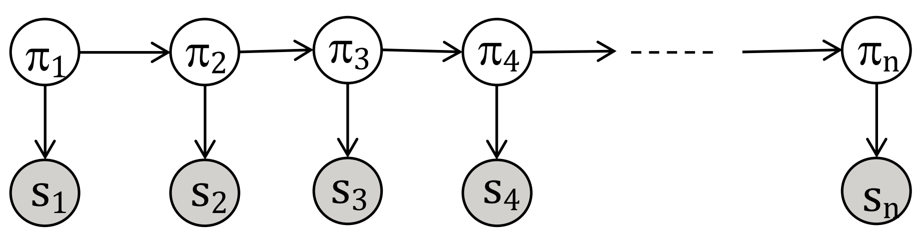


https://en.wikipedia.org/wiki/Hidden_Markov_model

Hidden Markov Models

- A Hidden Markov Model (HMM) is composed of
  - Set of (hidden) states, capable of *emitting* symbols according to a probability distribution (in the base case: *multinomial i.i.d*)
  - Set of transitions between the states, with transition probabilities (*a Markov chain*)

- Two kinds of sequences:
  - State sequence (hidden) $\pi = (\pi_1, \ldots, \pi_n)$ called the *path*
  - Symbol sequence (observed): $s = (s_1, \ldots, s_n)$



Aalto University
School of Science

Applications of Hidden Markov Models

- **Segmentation** of biological sequences into potentially meaningful regions with precise boundaries

- **Multiple alignment** of biological sequences (profile HMMs)
  - Multiple sequence alignment can be efficiently solved by taking the one-versus-all approach
  - Profile HMM can be interpreted as a model for the family of sequences

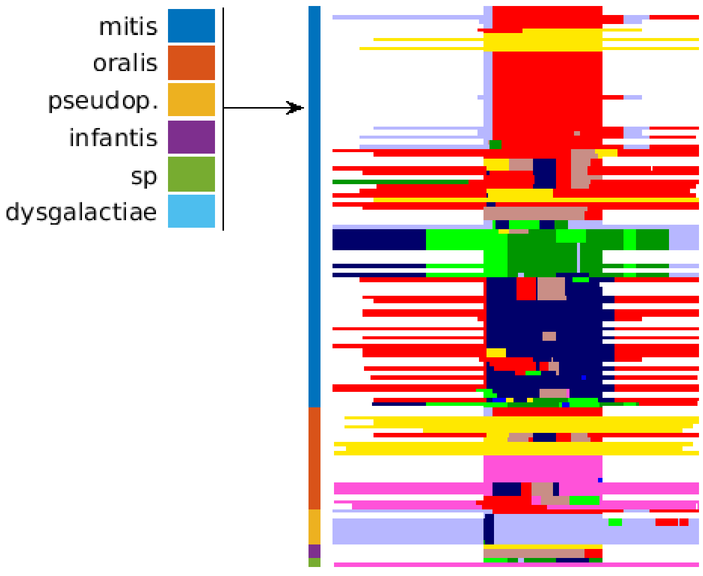
Applications of Hidden Markov Models

- **Functional annotation** can be achieved by matching a sequence against a HMM trained to recognize particular functional motifs.

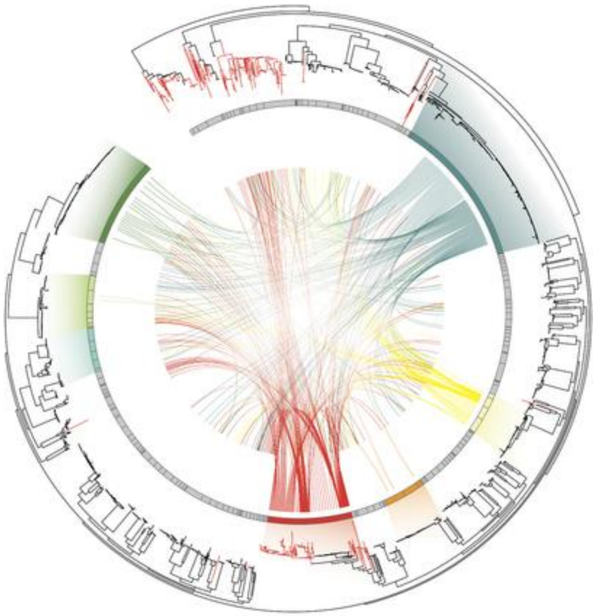
- **Gene finding** state-of-the-art methods are based on HMMs
  - Previous models (e.g., start codon + non-stop codons + stop codon) are not suited for eukaryotic genes or pseudogenes

Applications of Hidden Markov models

- Detection of recombination between bacterial species (research at Aalto)



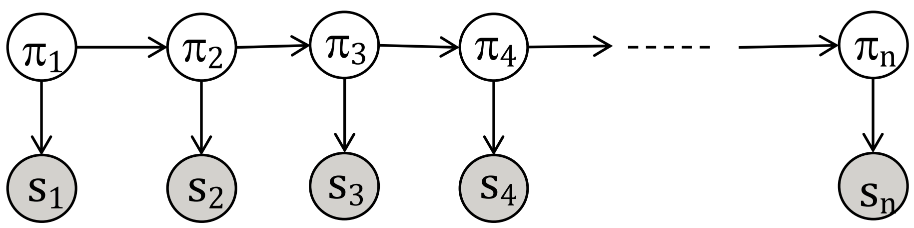
Marttinen et al. (2017),
https://doi.org/10.1101/059642



Chewapreecha et al. (2014),
Nature Genetics

Hidden Markov Models

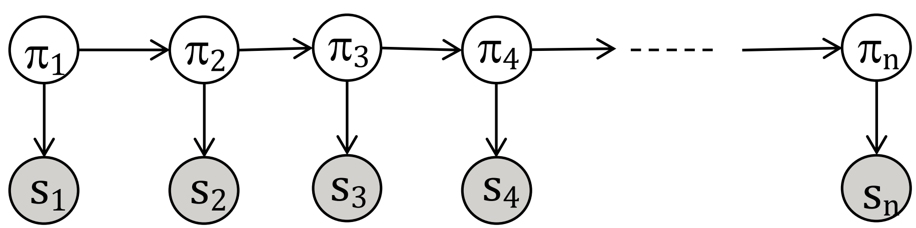
- Basic idea: a sequence is indirectly generated by a Markov chain
  - The Markov chain has some hidden (unknown) state for each position in the sequence
  - We observe the character generated at each position according to a multinomial distribution that depends on the state
- The sequence is generated by two random processes:
  1) generate the hidden Markov chain
  2) generate the symbol in each state of the chain using a multinomial model



Hidden Markov Models

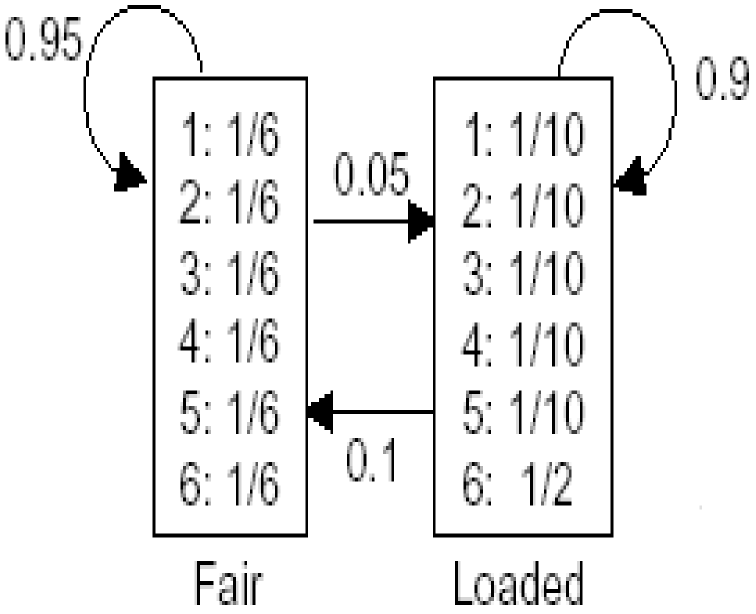
- Transition probability: the probability of switching between hidden states in the Markov chain
  - $T_{kl} = P(\pi_i = l \mid \pi_{i-1} = k)$ for i=2,...,n
  - $T_{0k} = P(\pi_1 = k)$
- Emission probability: the probability of emitting a certain symbol in a given state *k*
  - $E_k(b) = P(s_i = b \mid \pi_i = k)$
  - conditional on current state, s_i is independent of the previous symbol s_{i-1}
- The joint probability of $s=(s_1, s_2, \ldots, s_n)$ and $\pi=(\pi_1, \pi_2, \ldots, \pi_n)$ is:

$$P(s, \pi) = T_{0,\pi_1} E_{\pi_1}(s_1) \prod_{i=1}^{n-1} T_{\pi_i,\pi_{i+1}} E_{\pi_{i+1}}(s_{i+1})$$



Simple running example: occasionally dishonest casino

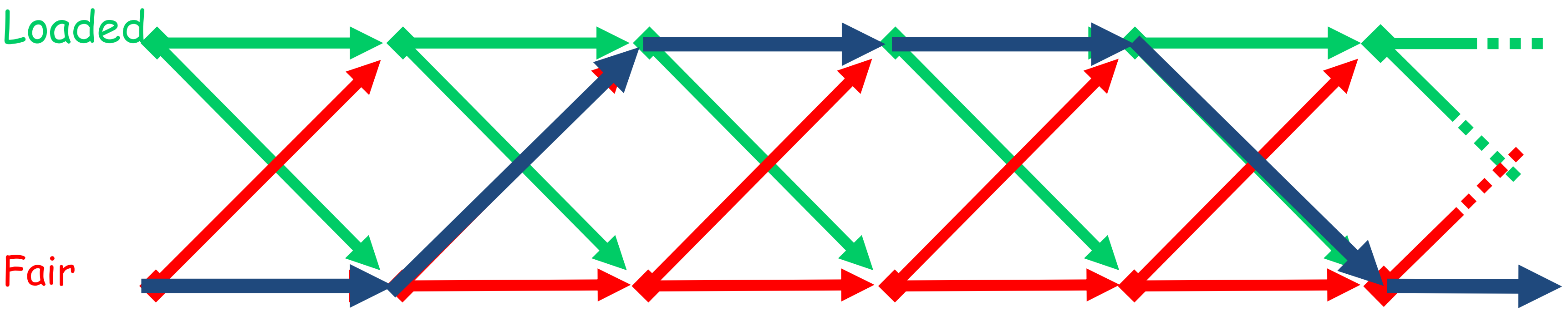
- Casino uses a fair dice most of the time, but switches to the loaded dice once in a while
- Can we detect which dice is in use at any given time, just by observing the sequence of rolls?



Sequential view

Observed sequence of dice rolls:

1 → 4 → 3 → 6 → 6 → 4 →



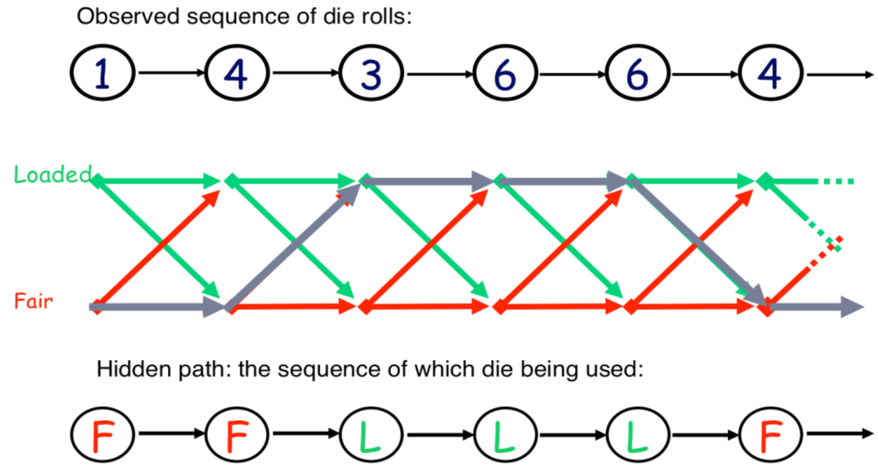
Hidden path: the sequence of which dice being used:

F → F → L → L → L → F →

Viterbi algorithm – finding the most probable state sequence

Decoding: finding the most probable path

- Decoding: Finding the most probable state sequence (path π^*) that could have generated the observed rolls



- The number of possible paths grows exponentially, so we need efficient algorithms

$$\pi^* = \arg\max_{\pi \in \Pi} P(s, \pi)$$

$$P(s,\pi) = T_{0,\pi_1} E_{\pi_1}(s_1) \prod_{i=1}^{n-1} T_{\pi_i,\pi_{i+1}} E_{\pi_{i+1}}(s_{i+1})$$

Aalto University
School of Science

Viterbi algorithm

- Dynamic programming, based on tabulating
  - probability $V_k(i)$ of the most probable hidden path $(\pi_1, \ldots, \pi_i)$ ending in state $\pi_i = k$ associated with the prefix $s_1, \ldots, s_i$
  - pointers for traceback
  - Formally:

$$V_k(i) = \max_{\pi_1, \ldots, \pi_{i-1}} p(\pi_1, \ldots, \pi_{i-1}, \pi_i = k, s_1, \ldots, s_i)$$

- Table V of size $m \times n$
  - m=number of hidden states
  - n=length of the observed sequence
  - $V(k,i) = V_k(i)$

Viterbi algorithm

- Updating the table: the information in each column is sufficient for computing the next column:

$$V_l(i+1) = E_l(s_{i+1}) \max_k V_k(i) T_{kl}$$

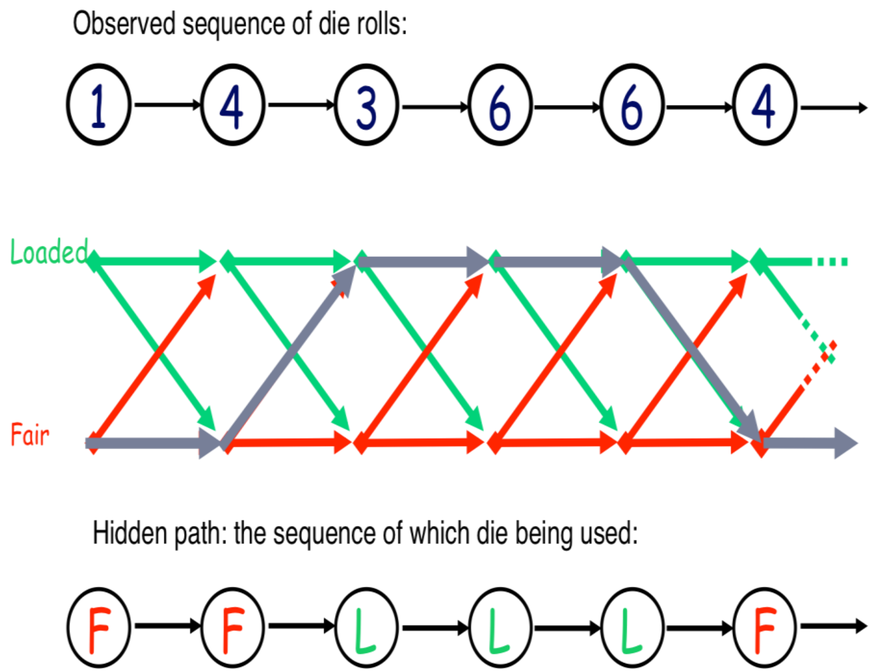
  – for prefix $(s_1, ..., s_i)$ find a state k that maximizes the combined probability of
    - the best path to k, $(\pi_1, ..., k)$: probability $V_k(i)$
    - making a transition from k to l: probability T_{kl}
    - emission probability for base s_{i+1} in state l
- In the end, we get

$$P(s, \pi^*) \quad = \quad \max_k V_k(n)$$

- Traceback recovers the best path $\pi^* \quad = \quad \arg\max_{\pi \in \Pi} P(s, \pi)$

Viterbi at the casino

- $V_{\text{loaded}}(5)$ is the maximum of two probabilities: the most probable sequences such that either
  - 4'th throw used a loaded dice and it is continued to be used for 5th throw, or
  - The dice was switched from fair to loaded after 4th throw
- Simple recurrence gives the result:



$$V_{\text{loaded}}(5) = E_{\text{loaded}}(6) \cdot \max \begin{cases} V_{\text{loaded}}(4) \cdot T_{\text{loaded,loaded}} \\ V_{\text{fair}}(4) \cdot T_{\text{fair,loaded}} \end{cases}$$

Implementation detail: avoiding numerical underflow

- Multiplying small probabilities may easily cause numerical underflow
- In computer implementation, it is better to use log-probabilities instead
- The updates remain similar
  - multiplication changed to summation
  - $\log \max_k x_k = \max_k \log x_k$, for non-negative x_k
- Reusing the notation for V, E, and T for the logarithmic quantities

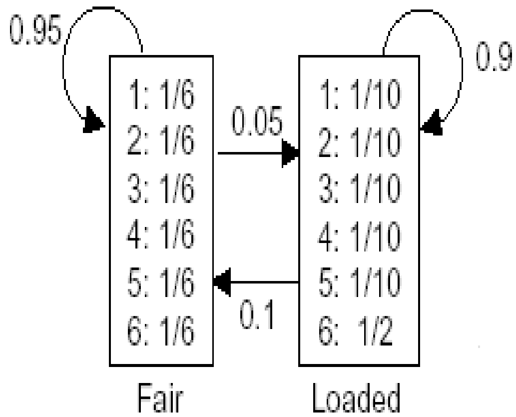
$$V_l(i+1) = E_l(s_{i+1}) + \max_k (V_k(i) + T_{kl})$$

Viterbi in R

```r
viterbi <- function(s, T, E) {

  log.E <- log2(E)
  log.T <- log2(T)

  # Pre-allocate V and TB:
  V <- matrix(rep(0, nrow(T)*length(s)), nrow=nrow(T))
  TB <- matrix(rep(0, nrow(T)*length(s)), nrow=nrow(T))

  # Initialize the first column of V:
  V[,1] <- log2(1/nrow(V)) + t(log.E[s[1],])

  # Calculate the V table
  for (i in 2:length(s)) {
    for (l in 1:nrow(V)) {
      V[l, i] <- max(log.T[,l] + V[, i-1])
      TB[l, i] <- which.max(log.T[,l] + V[,i-1])
      V[l, i] <- V[l, i] + log.E[s[i], l]
    }
  }
  log.prob <- max(V[,ncol(V)])
  k <- which.max(V[,ncol(V)])

  # Traceback
  path <- rep(NA, length(s))
  for (i in seq(length(s),2)) {
    path[i] <- k
    k <- TB[k, i]
  }
  path[1] <- k

  res <- list()
  res$log.prob <- log.prob
  res$path <- path
  res$V <- V

  return(res)
}
```

$$V_l(i+1) = E_l(s_{i+1}) + \max_k (V_k(i) + T_{kl})$$

Viterbi at the Casino

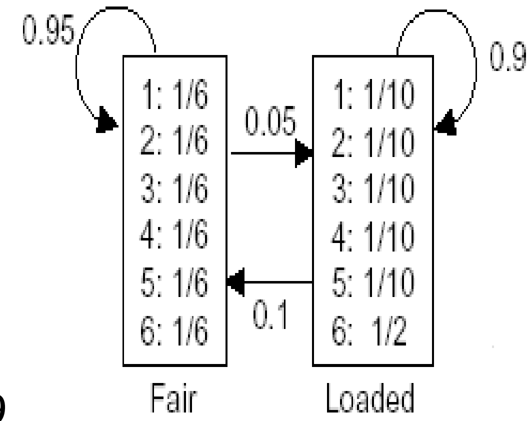



```
> log2(E)
          [,1]      [,2]
[1,] -2.584963 -3.321928
[2,] -2.584963 -3.321928
[3,] -2.584963 -3.321928
[4,] -2.584963 -3.321928
[5,] -2.584963 -3.321928
[6,] -2.584963 -1.000000
> log2(T)
            [,1]       [,2]
[1,] -0.07400058 -4.3219281
[2,] -3.32192809 -0.1520031
```

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
>
> viterbi.res$v
          [,1]      [,2]       [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
[1,] -3.584963 -6.243926  -8.902889 -11.56185 -14.22081 -16.87978 -19.53874 -22.19770 -24.85667 -27.51563
[2,] -4.321928 -7.795859 -11.269790 -14.74372 -18.21765 -21.69158 -22.20171 -23.35371 -24.50571 -25.65772
>
> viterbi.res$path
 [1] 1 1 1 1 1 1 2 2 2 2
```

$$V_l(i+1) = E_l(s_{i+1}) + \max_k (V_k(i) + T_{kl})$$

Viterbi at the Casino



$$\mathrm{V_{fair}}(1) = \mathrm{E_{fair}}(3) + \log_2(1/2) = -2.585 - 1 = -3.585$$

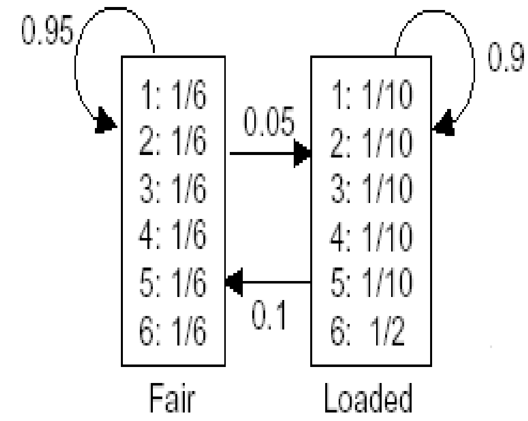
$$\mathrm{V_{loaded}}(1) = \mathrm{E_{loaded}}(3) + \log_2(1/2) = -3.3219 - 1 = -4.3219$$

```
> log2(E)
          [,1]      [,2]
[1,] -2.584963 -3.321928
[2,] -2.584963 -3.321928
[3,] -2.584963 -3.321928
[4,] -2.584963 -3.321928
[5,] -2.584963 -3.321928
[6,] -2.584963 -1.000000
> log2(T)
            [,1]       [,2]
[1,] -0.07400058 -4.3219281
[2,] -3.32192809 -0.1520031
```

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
>
> viterbi.res$v
          [,1]      [,2]       [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
[1,] -3.584963 -6.243926  -8.902889 -11.56185 -14.22081 -16.87978 -19.53874 -22.19770 -24.85667 -27.51563
[2,] -4.321928 -7.795859 -11.269790 -14.74372 -18.21765 -21.69158 -22.20171 -23.35371 -24.50571 -25.65772
>
> viterbi.res$path
 [1] 1 1 1 1 1 1 2 2 2 2
```

$$V_l(i+1) = E_l(s_{i+1}) + \max_k (V_k(i) + T_{kl})$$

Viterbi at the Casino



$$V_{loaded}(4) = E_{loaded}(1) + max\begin{cases} V_{fair}(3) + T_{fair,loaded} \\ V_{loaded}(3) + T_{loaded,loaded} \end{cases}$$

$$= -3.3219 + max\begin{cases} -8.9029 - 4.3219 \\ -11.2698 - 0.152 \end{cases} = -14.7437$$

```
> log2(E)
          [,1]      [,2]
[1,] -2.584963 -3.321928
[2,] -2.584963 -3.321928
[3,] -2.584963 -3.321928
[4,] -2.584963 -3.321928
[5,] -2.584963 -3.321928
[6,] -2.584963 -1.000000
> log2(T)
            [,1]       [,2]
[1,] -0.07400058 -4.3219281
[2,] -3.32192809 -0.1520031
```

```
> s
 [1] 3 2 2 1 2 3 6 6 6 6
>
> viterbi.res$v
          [,1]      [,2]       [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
[1,] -3.584963 -6.243926  -8.902889 -11.56185 -14.22081 -16.87978 -19.53874 -22.19770 -24.85667 -27.51563
[2,] -4.321928 -7.795859 -11.269790 -14.74372 -18.21765 -21.69158 -22.20171 -23.35371 -24.50571 -25.65772
>
> viterbi.res$path
 [1] 1 1 1 1 1 1 2 2 2 2
```

$$V_l(i+1) = E_l(s_{i+1}) + \max_k (V_k(i) + T_{kl})$$

Viterbi at the casino

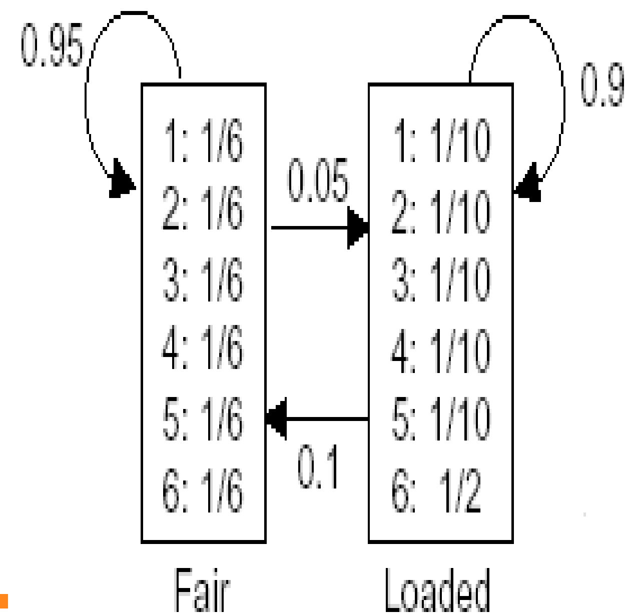
- Viterbi estimates remarkably well the correct dice

Rolls	315116246446644245321131631164152133625144543631656626566666
Die	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLL
Viterbi	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLL
Rolls	651166453132651245636664631636663162326455236266666625151631
Die	LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLFFFLLLLLLLLLLLLLLLFFFFFFFFF
Viterbi	LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFF
Rolls	222555441666566563564324364131513465146353411126414626253356
Die	FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Viterbi	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Rolls	366163666466232534413661661163252562462255265252266435353336
Die	LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi	LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Rolls	233121625364414432335163243633665562466662632666612355245242
Die	FFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
Viterbi	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF

Parameter estimation for HMMs

Parameter estimation for HMMs

- So far we have assumed that we have knowledge of the transition probabilities and emission probabilities
- How to obtain these values if we only know
  - the emitted sequence and HMM structure (here: Fair, Loaded)?
  - possibly the hidden state sequence



Aalto University
School of Science

Parameter estimation when the state sequence is known

- Assume we have
  - a set of training sequences $s^{(1)},\ldots, s^{(m)}$ where $s^{(i)} = (s_1^{(i)},\ldots,s_{n(i)}^{(i)})$, e.g.
    - Sequences of rolls of dice: $s^{(1)} = (1,3,4,3,\ldots)$, $s^{(2)} = (5,6,4,3,\ldots)$
    - Nucleotide sequences $s^{(1)}$ = AGTCGT… $s^{(2)}$ = CTGTAT…,
  - the set of states and corresponding state sequences of HMM
    - Which dice is being used: $y^{(1)}$ = FFFF…, $y^{(2)}$ = LLFF…
    - ORF/ non-ORF: $y^{(1)}$ = NNNYYY…, $y^{(2)}$ = NNNNNN
- The goal is to optimize HMM parameters
  - Transition probabilities T_{kl}
  - Emission probabilities $E_k(s_i)$

Parameter estimation when the state sequence is known

- Transition probabilities
  - we examine the given state sequences $y^{(1)},\ldots,y^{(m)}$
  - denote by t_{kl} the number of times transition k➔l was taken among the sequences
  - Our estimate for the transition probability is $$T_{kl} = \frac{t_{kl}+1}{\sum_{l'}(t_{kl'}+1)}$$
- Emission probabilities
  - we examine the emitted sequences $s^{(1)}, \ldots, s^{(m)}$ and the state sequences $y^{(1)}, \ldots, y^{(m)}$ together
  - denote by $e_k(b)$ the number of times b was emitted while in state k
  - The estimate for emission probability is $$E_k(b) = \frac{e_k(b)+1}{\sum_{b'}(e_k(b')+1)}$$

Pseudo-counts ($t_{kl}+1$, $e_k(b)+1$)

- Pseudo-counts are typically used to make the models less prone to over-fitting due to insufficient data
- In HMMs, the pseudo-counts also correct a problem arising if some state k is not visited in the training data:
    - need to allocate some probability to so far unseen events
- In general, the pseudo-counts can be any positive real numbers, however
    - too large numbers will override the training data
    - too small numbers will cause the parameters to over-fit the training data (leads to poorer performance on new, yet unseen data)

Parameter estimation when the state sequence is unknown

- Depending on the application, sometimes we may assume we know the state sequence
  - In many cases we have a training set that contains the states e.g. known coding regions in genes, known CG rich regions, …
- In other applications, such an assumption is not valid
  - which dice is used by the dishonest casino
  - data from newly sequenced organisms where no annotation has been done.

Parameter estimation when the state sequence is not known

- Assume we have
  - a set of training sequences $s^{(1)}$, ..., $s^{(m)}$, and the
  - set of states of the HMM
- The goal is to optimize HMM parameters
  - Transition probabilities T_{kl}
  - Emission probabilities $\mathrm{E}_{\mathrm{k}}(s_{\mathrm{i}})$
- Idea: choose the HMMs parameters so that the likelihood of the training data is maximized
- In the following, we present a training algorithm that uses as a subroutine the Viterbi algorithm to find the most probable path

Viterbi training

1. Initialize the HMM parameters in some way, e.g. setting
   i. $E_k(s) = 1/|\Sigma|$ uniformly, where Σ is the alphabet of symbols to emit
   ii. $T_{kl} = 1/N(k)$ uniformly, where $N(k)$ is the number states that can follow state k

- Alternatively, one can use a “best guess”
  - e.g. in the genome segmentation example, compute transition probabilities from dinucleotide frequencies

Aalto University
School of Science

Viterbi training

2. Iterate the following, until parameters do not change:
   i. For each sequence $s^{(i)}$, using Viterbi algorithm, find the most probable state sequence $\pi^{*(i)}$, given the current HMM parameters $\theta=(T,E)$
   ii. Count how many times each transition $k \rightarrow l$ was taken in the optimal paths $\pi^{*(1)},\ldots\pi^{*(m)}$, and denote that number by t_{kl}
   iii. Set the new transition probabilities as $$T_{kl} = \frac{t_{kl}+1}{\sum_{l'}(t_{kl'}+1)}$$
   iv. Count how many times each symbol b was emitted in each state k, and denote that number by $e_k(b)$
   v. Set the new emission probabilities as $$E_k(b) = \frac{e_k(b)+1}{\sum_{b'}(e_k(b')+1)}$$

Viterbi training

- The above algorithm works in *batch mode*: it assumes all training data is already available
- The training can also work in *online mode*, where the model is re-estimated when new data arrives
- Also, the training can work just as well on a single long sequence as on a set of short sequences
- The casino example highlights this training mode

Viterbi training at the casino

- Let us enter the occasionally dishonest casino, with our HMM, with initial guesses about the underlying model:

T	Fair	Loaded
Fair	.90	.10
Loaded	.10	.90

E	1	2	3	4	5	6
Fair	.167	.167	.167	.167	.167	.167
Loaded	.10	.10	.10	.10	.10	.50

- We observe a sequence of rolls: 3,4,6,4,6,6,2,6,3,4,1,5,3

Viterbi training at the casino

- We observe a sequence of rolls: 3,4,6,4,6,6,2,6,3,4,1,5,3
- With Viterbi estimation with the current model, we get: LLLLLLLLFFFFF
- Count transitions t and emissions e, add pseudo-counts

t+1	Fair	Loaded
Fair	4+1	0+1
Loaded	1+1	7+1

e+1	1	2	3	4	5	6
Fair	1+1	0+1	2+1	1+1	1+1	0+1
Loaded	0+1	1+1	1+1	2+1	0+1	4+1

Viterbi training at the casino

- Normalize to obtain estimated transition and emission probabilities $T_{kl} = \frac{t_{kl}+1}{\sum_{l'}(t_{kl'}+1)}$ $E_k(b) = \frac{e_k(b)+1}{\sum_{b'}(e_k(b')+1)}$

t+1	Fair	Loaded
Fair	4+1	0+1
Loaded	1+1	7+1

e+1	1	2	3	4	5	6
Fair	1+1	0+1	2+1	1+1	1+1	0+1
Loaded	0+1	1+1	1+1	2+1	0+1	4+1

T	Fair	Loaded
Fair	.83	.17
Loaded	.2	.8

E	1	2	3	4	5	6
Fair	.18	.09	.27	.18	.18	.09
Loaded	.07	.14	.14	.21	.07	.36

- We observe some more rolls: 5,3,4,2,1, 6,1,6,6,2,6,5

Viterbi training at the casino

- All rolls seen so far: 3,4,6,4,6,6,2,6,3,4,1,5,3,5,3,4,2,1, 6,1,6,6,2,6,5
- Viterbi estimation with the new model gives: LLLLLLLFFFFFFFFFFLLLLLLL
- Count transitions and emissions in all rolls seen so far, add pseudo-counts

t+1	Fair	Loaded
Fair	9+1	1+1
Loaded	1+1	13+1

e+1	1	2	3	4	5	6
Fair	2+1	1+1	3+1	2+1	2+1	0+1
Loaded	1+1	2+1	1+1	2+1	1+1	8+1

Viterbi training at the casino

- Normalize to obtain estimated transition and emission probabilities

t+1	Fair	Loaded
Fair	9+1	1+1
Loaded	1+1	13+1

e+1	1	2	3	4	5	6
Fair	2+1	1+1	3+1	2+1	2+1	0+1
Loaded	1+1	2+1	1+1	2+1	1+1	8+1

T	Fair	Loaded
Fair	.83	.17
Loaded	.125	.875

E	1	2	3	4	5	6
Fair	.187	.125	.25	.187	.187	.063
Loaded	.095	.14	.095	.14	.095	.43

- Casino closes, so we do not get more rolls, but we can continue training with the current data

Viterbi training at the casino

- All rolls seen so far: 3,4,6,4,6,6,2,6,3,4,1,5,3,5,3,4,2,1, 6,1,6,6,2,6,5
- Viterbi estimation with the new model gives: LLLLLLLLFFFFFFFFFFFLLLLLLL
- This turns out to be the same predicted sequence as in previous step, so our model stays the same

T	Fair	Loaded
Fair	.83	.17
Loaded	.125	.875

E	1	2	3	4	5	6
Fair	.187	.125	.25	.187	.187	.063
Loaded	.095	.14	.095	.14	.095	.43

- In general, with a longer sequence, more iterations could be needed for convergence

Viterbi training: convergence

- **If no more data arises** Viterbi training algorithm will eventually converge (and stop)
  - Each update of the parameters increases the probability of the most probable paths,
    - so the algorithm will never revisit a previous solution
  - There is only finite (but large) number of Viterbi paths to consider,
    - so we will eventually run out of solutions that we have not considered

Note: Accuracy of estimation depends on the amount of training data

True Model	Fair	Loaded
Fair	.95	.05
Loaded	.10	.90

True	1	2	3	4	5	6
Fair	.17	.17	.17	.17	.17	.17
Loaded	.10	.10	.10	.10	.10	.50

300 rolls	Fair	Loaded
Fair	.73	.27
Loaded	.29	.71

300 rolls	1	2	3	4	5	6
Fair	.19	.19	.23	.08	.23	.08
Loaded	.07	.10	.10	.17	.05	.52

30000 rolls	Fair	Loaded
Fair	.93	.07
Loaded	.12	.88

30000 rolls	1	2	3	4	5	6
Fair	.17	.17	.17	.17	.17	.17
Loaded	.10	.11	.10	.11	.10	.48