Hidden Markov Models (2)

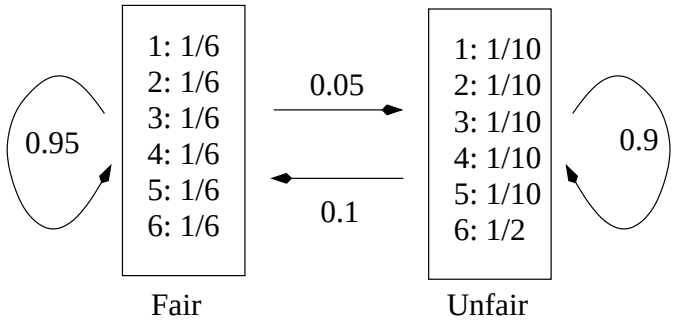
Definition (Hidden Markov Model (HMM))

A *HMM* is a graphical model $M = (\mathcal{S}, Q, A, e)$ consisting of

- an alphabet $\mathcal{S}$,
- a set of states Q,
- a matrix $A = \{a_{k\ell}\}$ of transition probabilities $a_{k\ell}$ for $k, \ell \in Q$, and
- an emission probability $e_k(b)$ for every $k \in Q$ and $b \in \mathcal{S}$.

Example fair/loaded dice

Casino uses two dice, *fair* and *loaded*:



Casino guest only observes the number rolled:

6 4 3 2 3 4 6 5 1 2 3 4 5 6 6 6 3 2 1 2 6 3 4 2 1 6 6...

Which dice was used remains hidden:

F F F F F F F F F F F F U U U U U F F F F F F F F F F...

Generation of simulated data

We can use HMMs to generate data:

Algorithm (Simulator)

Start in state 0.

While we have not reentered state 0*:*

Choose a new state using the transition probabilities

Choose a symbol using the emission probabilities and report it.

Generation of simulated data (2)

We use the *fair/loaded* HMM to generate a sequence of states and symbols:

```
Symbols: 24335642611341666666526562426612134635535566426666366642 53
States : FFFFFFFFFFFFFFUUUUUUUUUUUUUUUUUUUFFFFFFFFFUUUUUUUUUUUUUUFFFF

Symbols: 35246363252521655615445653663666511145445656621261532516435
States : FFFFFFFFFFFFFFFFFFFFFFFFFFFUUUUUUUFFUUUUUUUUUUUUUUFFFFFFFFF

Symbols: 5146526666
States : FFUUUUUUUU
```

How probable is a given sequence of data?

If we can observe only the symbols, can we reconstruct the corresponding states?

Three Key Questions for HMMs

Let M be a HMM, x a sequence of symbols.

(Q1) Determine the probability that M generated x: $P(x) = P(x \mid M)$: *forward algorithm*

(Q2) For x, determine the most probable sequence of states through M: *Viterbi algorithm*

(Q3) Given x and perhaps some additional sequences of symbols, how do we train the parameters of M? *Baum-Welch* algorithm or *Viterbi training*

The probability, given states and symbols

Definition (Path)

A *path* $\pi = (\pi_1, \pi_2, \ldots, \pi_L)$ is a sequence of states in the model M.

Suppose we are given a sequence of symbols $x = (x_1, \ldots, x_L)$ and a path $\pi = (\pi_1, \ldots, \pi_L)$ through M. The joint probability is:

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

with $\pi_{L+1} = 0$.

Unfortunately, in practice we usually do not know the path through the model.

Probability of a sequence of symbols

- Problem: We have observed a sequence x of symbols and would like to know the probability given the HMM M.
- Example: The sequence of symbols $(6, 5, 2)$ has a number of explanations within the dice-model:

Probability of a sequence of symbols

- Problem: We have observed a sequence x of symbols and would like to know the probability given the HMM M.
- Example: The sequence of symbols $(6, 5, 2)$ has a number of explanations within the dice-model:
- $(0, 0, 1)$, $(0, 1, 0)$, $(0, 0, 1)$,$(1, 1, 0)$,
  $(1, 0, 1)$, $(0, 1, 1)$, $(1, 1, 1)$, $(0, 0, 0)$

Probability of a sequence of symbols

- Problem: We have observed a sequence x of symbols and would like to know the probability given the HMM M.
- Example: The sequence of symbols $(6, 5, 2)$ has a number of explanations within the dice-model:
- $(0, 0, 1)$, $(0, 1, 0)$, $(0, 0, 1)$,$(1, 1, 0)$,
  $(1, 0, 1)$, $(0, 1, 1)$, $(1, 1, 1)$, $(0, 0, 0)$
- First idea: we could iterate over all possible state paths and sum the probabilities.

Probability of a sequence of symbols

- Problem: We have observed a sequence x of symbols and would like to know the probability given the HMM M.
- Example: The sequence of symbols $(6, 5, 2)$ has a number of explanations within the dice-model:
- $(0, 0, 1)$, $(0, 1, 0)$, $(0, 0, 1)$,$(1, 1, 0)$, $(1, 0, 1)$, $(0, 1, 1)$, $(1, 1, 1)$, $(0, 0, 0)$
- First idea: we could iterate over all possible state paths and sum the probabilities.
- This would take 2^T time.

Computing $P(x \mid M)$

Suppose we are given an HMM M and a sequence of symbols x. The probability that x was generated by M is given by:

$$P(x \mid M) = \sum_{\pi} P(x, \pi \mid M),$$

summing over all possible state sequences π through M.

Forward algorithm

The value of $P(x \mid M)$ can be efficiently computed using the *forward algorithm*.

This algorithm is obtained from the Viterbi algorithm by replacing max by a sum. More precisely, we define the *forward-variable*:

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k),$$

which equals the probability that the model reports the prefix sequence $(x_1, \ldots, x_i)$ and is in state $\pi_i = k$ at position i.

We obtain the recursion:

Forward algorithm

The value of $P(x \mid M)$ can be efficiently computed using the *forward algorithm*.

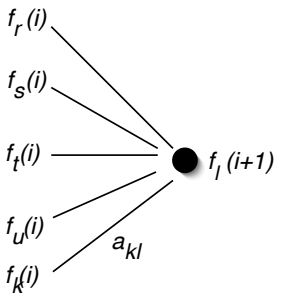
This algorithm is obtained from the Viterbi algorithm by replacing max by a sum. More precisely, we define the *forward-variable*:

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k),$$

which equals the probability that the model reports the prefix sequence $(x_1, \ldots, x_i)$ and is in state $\pi_i = k$ at position i.

We obtain the recursion:

- $f_\ell(i+1) = e_\ell(x_{i+1}) \sum_{k \in Q} f_k(i) a_{k\ell}$.



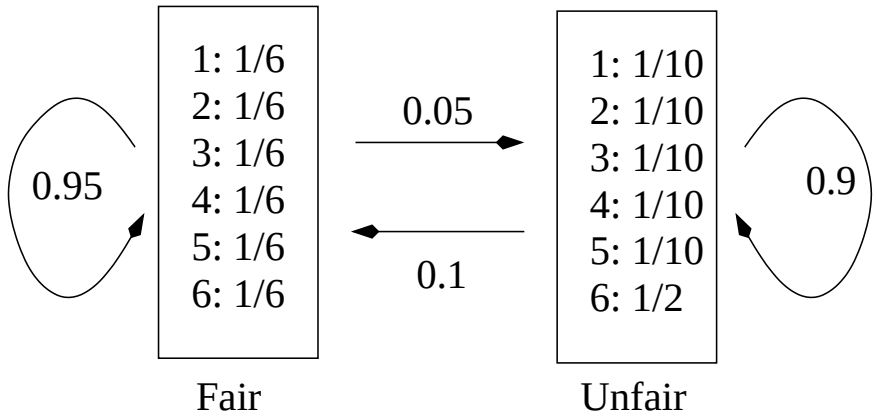
Algorithm (Forward algorithm)

Input:	*HMM* $M = (\mathcal{S}, Q, A, e)$ *and sequence of symbols* x
Output:	*probability* $P(x \mid M)$
Initialization $(i = 0)$*:*	$f_0(0) = 1$, $f_k(0) = 0$ *for* $k \neq 0$.
For all $i = 1 \ldots L$, $\ell \in Q$*:*	$f_\ell(i) = e_\ell(x_i) \sum_{k \in Q} (f_k(i-1) a_{k\ell})$
Result:	$P(x \mid M) = \sum_{k \in Q} (f_k(L) a_{k0})$

Implementation hint: Logarithms can not be employed here easily, but there are so-called "scaling methods".

This solves key question Q1!

Forward algorithm - an Example



- sequence: $(5, 2, 6, 6)$
- e_0: $1/6$ $1/6$ $1/6$ $1/6$ $1/6$ $1/6$
  e_1: $1/10$ $1/10$ $1/10$ $1/10$ $1/10$ $1/2$
- a_0: 0.95 0.05
  a_1: 0.1 0.9
- f_0: 1 0.1583 0.025 0.004 0.00065
  f_1: 0 0.005 0.0012 0.001 0.0006

The most probable path

To solve the decoding problem, we want to determine the path π^* that maximizes the probability of having generated the sequence x of symbols, that is:

$$\pi^* = \arg\max_{\pi} P(x, \pi).$$

This *most probable path* π^* can be computed recursively.

Definition (Viterbi variable)

Given a prefix $(x_1, x_2, \ldots, x_i)$, the *Viterbi* variable $v_k(i)$ denotes the probability that the most probable path is in state k when it generates symbol x_i at position i. Then:

$$v_\ell(i+1) = e_\ell(x_{i+1}) \max_{k \in Q}(v_k(i) a_{k\ell}),$$

with $v_0(0) = 1$, initially.

(Note: We have: $\arg\max_\pi P(x, \pi) = \arg\max_\pi P(\pi \mid x)$, because $P(x, \pi) = P(\pi \mid x)P(x)$.)

The Viterbi algorithm

Algorithm (Viterbi algorithm)

Input:	*HMM* $M = (\mathcal{S}, Q, A, e)$ *and symbol sequence* x
Output:	*Most probable path* π^*.
Initialization $(i = 0)$*:*	$v_0(0) = 1$, $v_k(0) = 0$ *for* $k \neq 0$.
For all $i = 1 \ldots L$, $\ell \in Q$*:*	$v_\ell(i) = e_\ell(x_i) \max_{k \in Q}(v_k(i-1)a_{k\ell})$ $\mathrm{ptr}_\ell(i) = \arg\max_{k \in Q}(v_k(i-1)a_{k\ell})$
Termination:	$P(x, \pi^*) = \max_{k \in Q}(v_k(L)a_{k0})$ $\pi^*_L = \arg\max_{k \in Q}(v_k(L)a_{k0})$
Traceback:	
For all $i = L-1, \ldots, 1$*:*	$\pi^*_i = \mathrm{ptr}_{\pi^*_{i+1}}(i+1)$

Implementation hint: instead of multiplying many small values, add their

Viterbi-decoding of the casino example

We used the *fair/loaded* HMM to first generate a sequence of symbols and then use the Viterbi algorithm to decode the sequence, result:

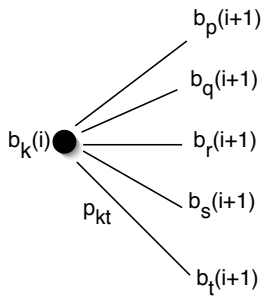
```
Symbols: 24335642611341666666526562426612134635535566462666636664253
States : FFFFFFFFFFFFFFUUUUUUUUUUUUUUUUUUUFFFFFFFFFFUUUUUUUUUUUUUUFFFF
Viterbi: FFFFFFFFFFFFFFUUUUUUUUUUUUUUUUUUUFFFFFFFFFFUUUUUUUUUUUUUUFFFF

Symbols: 352463632525216556154456536636665111454456566212615325164 35
States : FFFFFFFFFFFFFFFFFFFFFFFFFFFFUUUUUUUUFFUUUUUUUUUUUUUUFFFFFFFFF
Viterbi: FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Symbols: 5146526666
States : FFUUUUUUUU
Viterbi: FFFFFFUUUU
```

Backward algorithm

The backward-variable contains the probability to start in state $p_i = k$ and then to generate the suffix sequence $(x_{i+1}, \ldots, x_L)$:
$b_k(i) = P(x_{i+1} \ldots x_L \mid \pi_i = k)$.

Algorithm (Backward algorithm)

Input:	*HMM* $M = (S, Q, A, e)$ *and sequence of symbols* x
Output:	*probability* $P(x \mid M)$
Initialization $(i = L)$*:*	$b_k(L) = a_{k0}$ *for all* k*.*
For all $i = L-1 \ldots 1$, $k \in Q$*:*	$b_k(i) = \sum_{\ell \in Q} a_{k\ell} e_\ell(x_{i+1}) b_\ell(i+1)$
Result:	$P(x \mid M) = \sum_{\ell \in Q} (a_{0l} e_\ell(x_1) b_\ell(1))$



Summary of the three variables

Viterbi	$v_k(i)$	probability with which the most probable state path generates the sequence of symbols $(x_1, x_2, \ldots, x_i)$ and the system is in state k at time i.
Forward	$f_k(i)$	probability that the prefix sequence of symbols $x_1, \ldots, x_i$ is generated, and the system is in state k at time i.
Backward	$b_k(i)$	probability that the system starts in state k at time i and then generates the sequence of symbols $x_{i+1}, \ldots, x_L$.

Posterior probabilities

Suppose we are given an HMM M and a sequence of symbols x. Let $P(\pi_i = k \mid x)$ be the probability that symbol x_i was reported in state $\pi_i = k$. We call this the *posterior probability*, as it computed *after* observing the sequence x.

We have:

$$P(\pi_i = k \mid x) = \frac{P(\pi_i = k, x)}{P(x)} = \frac{f_k(i)b_k(i)}{P(x)},$$

as $P(g, h) = P(g \mid h)P(h)$ and by definition of the forward- and backward-variable.

Training the parameters

How does one generate an HMM?

First step: Determine its "topology", i.e. the number of states and how they are connected via transitions of non-zero probability.

The topology is usually designed "by hand".

Second step: Set the parameters, i.e. the transition probabilities $a_{k\ell}$ and the emission probabilities $e_k(b)$.

We will now discuss the second step. Given a set of example sequences, our goal is to train the parameters of the HMM using the example sequences, e.g. to set the parameters so as to maximize the probability with which the HMM generates the given example sequences.

Training when the states are known

Let $M = (\mathcal{S}, Q, A, e)$ be a HMM.

Suppose we are given a list of sequences of symbols $x^1, x^2, \ldots, x^n$ and a list of corresponding paths $\pi^1, \pi^2, \ldots, \pi^n$. (E.g., DNA sequences with annotated $\mathtt{C}p\mathtt{G}$-islands.)

We want to choose the parameters (A, e) of the HMM M *optimally*, such that:

$$P\big(x^1, \ldots, x^n, \pi^1, \ldots, \pi^n \mid M = (\mathcal{S}, Q, A, e)\big) =$$

$$\max_{(A', e')} P\big(x^1, \ldots, x^n, \pi^1, \ldots, \pi^n \mid M = (\mathcal{S}, Q, A', e')\big).$$

In other words, we want to determine the so-called *Maximum Likelihood Estimator* (*ML-estimator*) for (A, e).

ML-Estimation for (A, e)

(Recall: If we consider $P(D \mid M)$ as a function of D, then we call this a *probability*; as a function of M, then we use the word *likelihood*.)

ML-estimation:

$$(A, e)^{\mathrm{ML}} = \arg \max_{(A', e')} P(x^1, \ldots, x^n, \pi^1, \ldots, \pi^n \mid M = (S, Q, A', e')).$$

To compute A and e from labeled training data, we first determine the following numbers:

$\hat{a}_{k\ell}$: Number of observed transitions from state k to ℓ

$\hat{e}_k(b)$: Number of observed emissions of b in state k

We then set A and e as follows:

$$a_{k\ell} = \frac{\hat{a}_{k\ell}}{\sum_{q \in Q} \hat{a}_{kq}} \qquad \text{and} \qquad e_k(b) = \frac{\hat{e}_k(b)}{\sum_{s \in S} \hat{e}_k(s)}. \qquad (*)$$

Training the *fair/loaded* HMM

Suppose we are given example data x and π:

```
Symbols x: 1 2 5 3 4 6 1 2 6 6 3 2 1 5
States pi: F F F F F F F U U U U F F F
```

State transitions:

$\hat{a}_{k\ell}$	0	F	U
0			
F			
U			

$\rightarrow$

$a_{k\ell}$	0	F	U
0			
F			
U			

Emissions:

$\hat{e}_k(b)$	1	2	3	4	5	6
0						
F						
U						

$\rightarrow$

$e_k(b)$	1	2	3	4	5	6
0						
F						
U						

Pseudocounts

One problem in training is *overfitting*. For example, if some possible transition $k \mapsto \ell$ is never seen in the example data, then we will set $\bar{a}_{k\ell} = 0$ and the transition is then strictly forbidden.

Also, if a given state k is never seen in the example data, then $\bar{a}_{k\ell}$ is undefined for all ℓ.

To solve this problem, we introduce *pseudocounts* $r_{k\ell}$ and $r_k(b)$, and define:

$$\begin{aligned} \hat{a}_{k\ell} &= \text{number of transitions from } k \text{ to } \ell \text{ in the example data } + r_{k\ell} \\ \hat{e}_k(b) &= \text{number of emissions of } b \text{ in } k \text{ in the example data } + r_k(b). \end{aligned}$$

Small pseudocounts reflect "little pre-knowledge", large ones reflect "more pre-knowledge".

The *Laplace rule* is to use a pseudocount of 1 everywhere.

Viterbi training

In unsupervised training, the usual strategy is to iteratively improve the parameters of the HMM.

One such algorithm is the *Baum Welch* algorithm, which is based on the general technique of *expectation maximization*. It aims at optimizing the log-likelihood score.

Viterbi training (2)

Here is a conceptually slightly simpler algorithm, called *Viterbi training*:

Algorithm (Viterbi training)

Let $M = (\mathcal{S}, Q, A, e)$ be a HMM and assume we are given training sequences $x^1, x^2, \ldots, x^n$. *For a number of different random seeds, assign random, non-zero transition and emission probabilities to the HMM. Then, iteratively improve the parameters as follows:*

1. *Use the Viterbi algorithm to compute a state path π^i for each of the training sequences x^i.*
2. *Use this data (and pseudo-counts) to modify the parameters of the HMM as in supervised training.*
3. *Repeat until converged.*

Return the parameters (A, e) that produce the highest probabilities on the training data.

Baum-Welch algorithm

(In the lecture we didn't actually do this but rather we looked at Viterbi training.)

Let $M = (\mathcal{S}, Q, A, e)$ be a HMM and assume we are given *training sequences* $x^1, x^2, \ldots, x^n$. The parameters (A, e) are to be iteratively improved as follows:

- Based on $x^1, \ldots, x^n$ and $\pi^1, \ldots, \pi^n$ and the current value of (A, e) we estimate *expectation values* $\overline{a_{k\ell}}$ and $\overline{e_\ell(b)}$ for $\hat{a}_{k\ell}$ and $\hat{e}_\ell(b)$.

- We then compute (A, e) from $\bar{A}$ and $\bar{e}$ using equation $(*)$.

- This is repeated until the log-likelihood score cannot be improved.

(This is a special case of the so-called *expectation maximization* (EM) technique.)

Algorithm (Baum-Welch algorithm)

Input: HMM $M = (\mathcal{S}, Q, A, e)$, training data $x^1, x^2, \ldots, x^n$,
Output: HMM $M' = (\mathcal{S}, Q, A', e')$ *with an improved score.*
Init.: Randomly assign A *and* e
repeat
  for each *sequence* x_j **do**
    for each *position* i **do**
      for each *state* k **do**
        Compute $f_k(i)$ *for* x^j *with the forward algorithm.*
        Compute $b_k(i)$ *for* x^j *with the backward algorithm.*
  for each *state* k **do**
    for each *state* ℓ **do**
      Compute $\overline{a_{k\ell}} = \sum_j \frac{1}{f_0(L)} \sum_i f_k^j(i) a_{k\ell} e_\ell(x_{i+1}^j) b_\ell^j(i+1)$
    for each *symbol* b **do**
      Compute $\overline{e_k(b)} = \sum_j \frac{1}{f_0(L)} \sum_{\{i|x_i^j=b\}} f_k^j(i) b_\ell^j(i)$

Why do we use the following expression to compute the expectation for $\hat{a}_{k\ell}$ in the algorithm?

$$\bar{a}_{k\ell} = \sum_{j=1}^{n} \frac{1}{f_0(L)} \sum_{i=1}^{L^j} f_k^j(i) a_{k\ell} e_\ell(x_{i+1}^j) b_\ell^j(i+1)$$

For a single sequence x and a single position i, the expected number of transitions from $\pi_i = k$ to $\pi_{i+1} = \ell$ is given by:

$$P(\pi_i = k, \pi_{i+1} = \ell \mid x, (A, e)) = \frac{f_k(i) a_{k\ell} e_\ell(x_{i+1}) b_\ell(i+1)}{P(x)}.$$

$$\left(\text{This follows from:} \quad P(\pi_i = k \mid x) = \frac{P(\pi_i = k, x)}{P(x)} = \frac{f_k(i) b_k(i)}{P(x)}. \right)$$

Convergence:

One can prove that the log-likelihood-score converges to a local maximum using the Baum-Welch algorithm.

However, this doesn't imply that the parameters converge!

Local maxima can be avoided by considering many different starting points.

Additionally, any standard optimization approaches can also be applied to solve the optimization problem.

Summary

A Hidden Markov model (HMM) is an example of a machine-learning datastructure that can be used to classify sequences. It is applicable when:

- there are a sequential dependencies in the data, and
- there is a sufficient supply of training sequences.

Training is known as *supervised*, if annotated training data are available, or *unsupervised*, if the training data are not annotated.

HMMs are used in the detection of C*p*G-islands and, more importantly, as profile HMMs to define protein families. Another application is in gene prediction.