Chapter 3

Hidden Markov Models (HMMs)

3.1 Hidden Markov Models (HMMs)

There is a variant of the notion of DFA with output, for example a transducer such as a gsm (generalized sequential machine), which is widely used in machine learning.

This machine model is known as hidden Markov model, for short HMM.

There are three new twists compared to traditional gsm models:

(1) If the set of states is denoted $Q = \{q_1, \ldots, q_n\}$, then the transitions between states are labeled with probabilities rather than symbols from an alphabet. For any two states $q_i$ and $q_j$, the edge from $q_i$ to $q_j$ is labeled with a probability $A(i, j)$.
(2) There is a finite set $O = \{O_1, \ldots, O_m\}$ of possible outputs (called the observation space) that can be emitted, and for every state $q_i$, there is a probability $B(i,j)$ that output $O_j$ is emitted (produced).

(3) Sequences of outputs $O = (O_{\omega_1}, \ldots, O_{\omega_T})$ (with $\omega_t \in \{1, \ldots, m\}$ for $t = 1, \ldots, T$) emitted by the model are directly observable, but the sequences of states $S = (q_{i_1}, \ldots, q_{i_T})$ (with $i_t \in \{1, \ldots, n\}$ for $t = 1, \ldots, T$) that caused some sequence of output to be emitted are not observable.

In this sense the states are hidden, and this is the reason for calling this model a hidden Markov model.

HMM’s are among the most effective tools to solve the following types of problems:
(1) **DNA and protein sequence alignment** in the face of mutations and other kinds of evolutionary change.

(2) **Speech understanding systems**: When we talk, our mouths produce sequences of sounds from the sentences that we want to say. This process is complex.

Multiple words may map to the same sound, words are pronounced differently as a function of the word before and after them, we all form sounds slightly differently, and so on.

All a listener can hear (perhaps a computer system) is the sequence of sounds, and the listener would like to reconstruct the mapping (backward) in order to determine what words we were attempting to say.

For example, when you “talk to your TV” to pick a program, say *game of thrones*, you don’t want to get *the price is right*. 
(3) Optical character recognition (OCR). When we write, our hands map from an idealized symbol to some set of marks on a page (or screen).

The marks are observable, but the process that generates them isn’t.

A system performing OCR, such as a system used by the post office to read addresses, must discover which word is most likely to correspond to the mark it reads.
**Example 3.1.** Say we consider the following behavior of some professor at some university.

On a *hot day* (denoted by Hot), the professor comes to class *with a drink* (denoted D) with probability 0.7, and *with no drink* (denoted N) with probability 0.3.

On the other hand, on a *cold day* (denoted Cold), the professor comes to class *with a drink* with probability 0.2, and *with no drink* with probability 0.8.

Suppose a student intrigued by this behavior recorded a sequence showing whether the professor came to class with a drink or not, say NNND.

Several months later, *the student would like to know whether the weather was hot or cold the days he recorded the drinking behavior of the professor.*
Now the student heard about machine learning, so he constructs a probabilistic (hidden Markov) model of the weather, namely based on some experiments, he determines the probability of a transition from a hot day to another hot day, or to a cold day, and the probability of a transition from a cold day to another cold day, or to a hot day.

He also knows that when he started his observations, it was a cold day with probability 0.45, and a hot day with probability 0.55.

The above data determine an HMM depicted in Figure 3.1.
The portion of the state diagram involving the states Cold, Hot, is analogous to an NFA in which the transition labels are probabilities; it is the underlying Markov model of the HMM.

For any given state, the probabilities sum to 1.

The start state is a convenient way to express the probabilities of starting either in state Cold or in state Hot.
Also, from each of the states Cold and Hot, we have emission probabilities of producing the output N or D, and these probabilities also sum to 1.

We can also express these data using *matrices*.

If we associate Cold with the index 1 and Hot with the index 2, then the matrix

\[
A = \begin{pmatrix}
0.7 & 0.3 \\
0.25 & 0.75
\end{pmatrix}
\]

describes the transitions of the Markov model.

The vector

\[
\pi = \begin{pmatrix}
0.45 \\
0.55
\end{pmatrix}
\]

describes the probabilities of starting either in state Cold or in state Hot.
If we associate $N$ with the index 1 and $D$ with the index 2, then the matrix
\[ B = \begin{pmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{pmatrix} \]
describes the emission probabilities.

*The student would like to solve what is known as the decoding problem.*

Namely, *given the output sequence NNND, find the most likely state sequence of the Markov model that produces the output sequence NNND.*

Is it (Cold, Cold, Cold, Cold), or (Hot, Hot, Hot, Hot), or (Hot, Cold, Cold, Hot), or (Cold, Cold, Cold, Hot)?

Given the probabilities of the HMM, it seems unlikely that it is (Hot, Hot, Hot, Hot), but how can we find the most likely one?
Let us consider another example taken from Stamp [?].

**Example 3.2.** Suppose we want to determine the average annual temperature at a particular location over a series of years in a distant past where thermometers did not exist.

Since we can’t go back in time, we look for indirect evidence of the temperature, say in terms of the *size of tree growth rings*.

For simplicity, assume that we consider the two temperatures Cold and Hot, and three different sizes of tree rings: small, medium and large, which we denote by S, M, L.

The HMM shown in Figure 3.2 is a model of the situation.
Suppose we observe the sequence of tree growth rings (S, M, S, L).

What is the most likely sequence of temperatures over a four-year period which yields the observations (S, M, S, L)?

Going back to Example 3.1, we need to figure out the probability that a sequence of states $S = (q_{i_1}, q_{i_2}, \ldots, q_{i_T})$ produces the output sequence $O = (O_{\omega_1}, O_{\omega_2}, \ldots, O_{\omega_T})$. 
Denote our set of states by $Q = \{q_1, q_2\} = \{\text{Cold, Hot}\}$ and our set of outputs by $O = \{O_1, O_2\} = \{N, D\}$.

Then the probability that we want is just the product of the probability that we begin with state $q_{i_1}$, times the product of the probabilities of each of the transitions, times the product of the emission probabilities, namely

$$\Pr(S, O) = \pi(i_1)B(i_1, \omega_1) \prod_{t=2}^{T} A(i_{t-1}, i_t)B(i_t, \omega_t).$$

In our example, $(\omega_1, \omega_2, \omega_3, \omega_4) = (1, 1, 1, 2)$, which corresponds to NNND.

The brute-force method is to compute these probabilities for all $2^4 = 16$ sequences of states of length 4 (in general, there are $n^T$ sequences of length $T$).
For example, for the sequence $S = (\text{Cold, Cold, Cold, Hot})$, which corresponds to the sequence of indices $(1, 1, 1, 2)$, we find that

$$
\Pr(S, \text{NNND}) = \pi(1) B(1, 1) A(1, 1) B(1, 1) A(1, 1) B(1, 1) A(1, 2) B(2, 2) \\
= 0.45 \times 0.8 \times 0.7 \times 0.8 \times 0.7 \times 0.8 \\
\times 0.3 \times 0.7 = 0.0237.
$$

A much more efficient way to proceed is to use dynamic programming.

For $t = 1, 2, 3, 4$, for every state $i = 1, 2$, we compute $\text{score}(i, t)$ to be the highest probability that a path of length $t - 1$ ending in $q_i$ produces the output sequence $(O_{\omega_1}, \ldots, O_{\omega_t})$, and for $t \geq 2$, we let $\text{pred}(i, t)$ be the state that precedes $q_i$ in a best path of length $t - 1$ ending in $q_i$. 
Initially, we set

\[ \text{score}(j, 1) = \pi(j)B(j, \omega_1), \quad j = 1, 2, \]

and since \( \omega_1 = 1 \) we get \( \text{score}(1, 1) = 0.45 \times 0.8 = 0.36 \)
and \( \text{score}(2, 1) = 0.55 \times 0.3 = 0.165 \).

Next we compute \( \text{score}(1, 2) \) and \( \text{score}(2, 2) \) as follows.

For \( j = 1, 2 \), for \( i = 1, 2 \), compute temporary scores

\[ t\text{score}(i, j) = \text{score}(i, 1)A(i, j)B(j, \omega_2); \]

then pick the \textit{best} of the temporary scores,

\[ \text{score}(j, 2) = \max_i t\text{score}(i, j). \]

Since \( \omega_2 = 1 \), we get \( t\text{score}(1, 1) = 0.36 \times 0.7 \times 0.8 = 0.2016 \),
\( t\text{score}(2, 1) = 0.165 \times 0.25 \times 0.8 = 0.0330 \), and
\( t\text{score}(1, 2) = 0.36 \times 0.3 \times 0.3 = 0.0324 \),
\( t\text{score}(2, 2) = 0.165 \times 0.75 \times 0.3 = 0.0371 \).
Then

\[
\text{score}(1, 2) = \max\{\text{tscore}(1, 1), \text{tscore}(2, 1)\}
= \max\{0.2016, 0.0330\} = 0.2016,
\]

and

\[
\text{score}(2, 2) = \max\{\text{tscore}(1, 2), \text{tscore}(2, 2)\}
= \max\{0.0324, 0.0371\} = 0.0371.
\]

Since the state that leads to the optimal score score(1, 2) is 1, we let \(\text{pred}(1, 2) = 1\), and since the state that leads to the optimal score score(2, 2) is 2, we let \(\text{pred}(2, 2) = 2\).
We compute $score(1, 3)$ and $score(2, 3)$ in a similar way.

For $j = 1, 2$, for $i = 1, 2$, compute

$$tscore(i, j) = score(i, 2)A(i, j)B(j, \omega_3);$$

then pick the best of the temporary scores,

$$score(j, 3) = \max_i tscore(i, j).$$

Since $\omega_3 = 1$, we get

$$score(1, 3) = \max\{0.1129, 0.074\} = 0.1129,$$

and

$$score(2, 3) = \max\{0.0181, 0.0083\} = 0.0181.$$ 

We also get $pred(1, 3) = 1$ and $pred(2, 3) = 1$.

Finally, we compute $score(1, 4)$ and $score(2, 4)$ in a similar way.
For $j = 1, 2$, for $i = 1, 2$, compute

$$t\text{score}(i, j) = score(i, 3)A(i, j)B(j, \omega_4);$$

then pick the best of the temporary scores,

$$score(j, 4) = \max_i t\text{score}(i, j).$$

Since $\omega_4 = 2$, we get

$$score(1, 4) = \max\{0.0158, 0.0009\} = 0.0158,$$

and

$$score(2, 4) = \max\{0.0237, 0.0095\} = 0.0237,$$

and $\text{pred}(1, 4) = 1$ and $\text{pred}(2, 4) = 1$.

Since $\max\{score(1, 4), score(2, 4)\} = 0.0237$, the state with the maximum score is $q_2 = \text{Hot}$, and by following the predecessor list we find the most likely path to produce the sequence NNND to be

$$(q_1, q_1, q_1, q_2) = (\text{Cold, Cold, Cold, Hot}).$$

The method we just described is known as the Viterbi algorithm.
Definition 3.1. A hidden Markov model, for short HMM, is a quintuple $M = (Q, O, \pi, A, B)$ where

- $Q = \{q_1, \ldots, q_n\}$ is a finite set of states, with $n$ states.
- $O = \{O_1, \ldots, O_m\}$ is a finite output alphabet (also called set of possible observations), with $m$ observations.
- $A = (A(i, j))$ is an $n \times n$ matrix called the state transition probability matrix, with
  \[
  A(i, j) \geq 0, \quad 1 \leq i, j \leq n, \quad \text{and} \quad \sum_{j=1}^{n} A(i, j) = 1,
  \]
  for $i = 1, \ldots, n$.
- $B = (B(i, j))$ is an $n \times m$ matrix called the state observation probability matrix (also called confusion matrix), with
  \[
  B(i, j) \geq 0, \quad 1 \leq i, j \leq n, \quad \text{and} \quad \sum_{j=1}^{m} B(i, j) = 1,
  \]
  for $i = 1, \ldots, n$. 


Examples of HMMs are shown in Figure 3.1, Figure 3.2, and Figure 3.3.

Note that an output is emitted when visiting a state, not when making a transition, as in the case of a gsm. So the analogy with the gsm model is only partial; it is meant as a motivation for HMMs.

If we ignore the output components $O$ and $B$, then we have what is called a *Markov chain*. 
There are three types of problems that can be solved using HMMs:

1. **The decoding problem**: Given an HMM $M = (Q, O, \pi, A, B)$, for any observed output sequence $O = (O_{\omega_1}, O_{\omega_2}, \ldots, O_{\omega_T})$ of length $T$, find a most likely sequence of states $S = (q_1, q_2, \ldots, q_T)$ that produces the output sequence $O$.

   More precisely, this means finding a sequence $S$ such that the probability

   $$ \Pr(S, O) = \pi(i_1)B(i_1, \omega_1)\prod_{t=2}^{T} A(i_{t-1}, i_t)B(i_t, \omega_t) $$

   is **maximal**.

   This problem is solved effectively by the *Viterbi algorithm*. 
(2) **The evaluation problem**: Given a finite collection \{M_1, \ldots, M_L\} of HMM’s with the same output alphabet \(O\), for any output sequence \(O = (O_{\omega_1}, O_{\omega_2}, \ldots, O_{\omega_T})\) of length \(T\), find which model \(M_\ell\) is most likely to have generated \(O\).

More precisely, given any model \(M_k\), we compute the probability \(tprob_k\) that \(M_k\) could have produced \(O\) along any path.

Then we pick an HMM \(M_\ell\) for which \(tprob_\ell\) is maximal. We will return to this point after having described the Viterbi algorithm.

A variation of the Viterbi algorithm called the *forward algorithm* effectively solves the evaluation problem.
The training problem: Given a set \( \{O_1, \ldots, O_r\} \) of output sequences on the same output alphabet \( O \), usually called a set of *training data*, given \( Q \), find the “best” \( \pi, A, \) and \( B \) for an HMM \( M \) that produces all the sequences in the training set, in the sense that the HMM \( M = (Q, O, \pi, A, B) \) is the most likely to have produced the sequences in the training set.

The technique used here is called *expectation maximization*, or *EM*. It is an iterative method that starts with an initial triple \( \pi, A, B \), and tries to improve it.

There is such an algorithm known as the *Baum-Welch* or *forward-backward algorithm*, but it is beyond the scope of this introduction.

Let us now describe the Viterbi algorithm in more details.
3.2 The Viterbi Algorithm and the Forward Algorithm

Given an HMM $M = (Q, O, \pi, A, B)$, for any observed output sequence $O = (O_\omega^1, O_\omega^2, \ldots, O_\omega^T)$ of length $T$, we want to find a most likely sequence of states $S = (q_i^1, q_i^2, \ldots, q_i^T)$ that produces the output sequence $O$.

For this, we need to find a sequence $S$ such that the probability

$$Pr(S, O) = \pi(i_1)B(i_1, \omega_1) \prod_{t=2}^{T} A(i_{t-1}, i_t)B(i_t, \omega_t)$$

is maximal.

In general, there are $n^T$ sequences of length $T$.

This problem can be solved efficiently by the method of dynamic programming.
For any \( t, 1 \leq t \leq T \), for any state \( q_j \in Q \), we compute \( \text{score}(j, t) \), which is the largest probability that a path of length \( t - 1 \) ending with \( q_j \) has produced the output sequence \((O_{\omega_1}, O_{\omega_2}, \ldots, O_{\omega_t})\).

The point is that if we know \( \text{score}(k, t - 1) \) for \( k = 1, \ldots, n \) (with \( t \geq 2 \)), then we can find \( \text{score}(j, t) \) for \( j = 1, \ldots, n \), because the probability associated with the path \((q_{i_1}, \ldots, q_{i_{t-1}}, q_j)\) is

\[
\text{tscore}(i_{t-1}, j) = \text{score}(i_{t-1}, t - 1)A(i_{t-1}, j)B(j, \omega_t),
\]

so to maximize this probability we just have to find the maximum of the probabilities \( \text{tscore}(i_{t-1}, j) \) over all \( i_{t-1} \), that is, we must have

\[
\text{score}(j, t) = \max_k \text{tscore}(k, j).
\]

To get started, we set \( \text{score}(j, 1) = \pi(j)B(j, \omega_1) \) for \( j = 1, \ldots, n \).
The algorithm goes through a *forward phase* for $t = 1, \ldots, T$, during which it computes the probabilities $score(j, t)$ for $j = 1, \ldots, n$.

When $t = T$, we pick a state $q_{iT} = q_j$ such that $score(j, T)$ is *maximal*.

The machine learning community is fond of the notation

$$j = \arg \max_k score(k, T)$$

to express the above fact.

This gives us *the last state* in an optimal sequence that yields the output sequence $\mathcal{O}$.

The algorithm then goes through a *backward phase*. 
To to this, when we compute

\[ \text{score}(j, t) = \max_k \text{tscore}(k, j), \]

we also record which state \( q_k = q_{i_{t-1}} \) was the last state of the best sequence \( (q_{i_1}, \ldots, q_{i_{t-1}}, q_j) \) for which \( \text{tscore}(k, j) \) is maximal, as \( \text{pred}(j, t) = k \).

This state may not be unique, we just pick one of them.

Again, this can be expressed by

\[ \text{pred}(j, t) = \arg \max_k \text{tscore}(k, j). \]

The predecessors \( \text{pred}(j, t) \) are only defined for \( t = 2, \ldots, T \), but we can let \( \text{pred}(j, 1) = 0 \).
The Viterbi algorithm is shown below.

The input to the algorithm is $M = (Q, O, \pi, A, B)$ and the observed sequence $\mathcal{O} = (O_{\omega_1}, O_{\omega_2}, \ldots, O_{\omega_T})$ of length $T$.

The output is a sequence of states $(y_1, \ldots, y_T) = (q_{I_1}, \ldots, q_{I_T})$, where $(I_1, \ldots, I_T)$ is the sequence of indices of the states.
The Viterbi Algorithm

begin
  for $j = 1$ to $n$ do
    $score(j, 1) = \pi(j)B(j, \omega_1)$
  endfor;

(* forward phase to find the best (highest) scores *)

  for $t = 2$ to $T$ do
    for $j = 1$ to $n$ do
      for $k = 1$ to $n$ do
        $tscore(k) = score(k, t - 1)A(k, j)B(j, \omega_t)$
      endfor;
      $score(j, t) = \max_k tscore(k)$;
      $pred(j, t) = \arg\max_k tscore(k)$
    endfor
  endfor;

(* backward phase to find the optimal path *)

  $I_T = \arg\max_j score(j, T)$;
  $y_T = q_{I_T}$;

  for $t = T$ to $2$ by $-1$ do
    $I_{t-1} = pred(I_t, t)$;
    $y_{t-1} = q_{I_{t-1}}$
  endfor
end
If we run the Viterbi algorithm on the output sequence \((S, M, S, L)\) of Example 3.2, we find that the sequence \((\text{Cold, Cold, Cold, Hot})\) has the highest probability, 0.00282, among all sequences of length four.

One may have noticed that the numbers involved, being products of probabilities, become quite small.

Indeed, underflow may arise in dynamic programming. Fortunately, there is a simple way to avoid underflow by taking logarithms.

It immediately verified that the time complexity of the Viterbi algorithm is \(O(n^2T)\).

Let us now to turn to the second problem, the \textit{evaluation problem}. 
This time, given a finite collection \( \{M_1, \ldots, M_L\} \) of HMM’s with the same output alphabet \( O \), for any observed output sequence \( \mathcal{O} = (O_{\omega_1}, O_{\omega_2}, \ldots, O_{\omega_T}) \) of length \( T \), find which model \( M_\ell \) is most likely to have generated \( \mathcal{O} \).

More precisely, given any model \( M_k \), we compute the probability \( tprob_k \) that \( M_k \) could have produced \( \mathcal{O} \) along any path.

Then we pick an HMM \( M_\ell \) for which \( tprob_\ell \) is maximal.

It is easy to adapt the Viterbi algorithm to compute \( tprob_k \). This algorithm is called the forward algorithm.

Since we are not looking for an explicity path, there is no need for the backward phase, and during the forward phase, going from \( t - 1 \) to \( t \), rather than finding the maximum of the scores \( tscore(k) \) for \( k = 1, \ldots, n \), we just set \( score(j, t) \) to the sum over \( k \) of the temporary scores \( tscore(k) \).

At the end, \( tprob_k \) is the sum over \( j \) of the probabilities \( score(j, T) \).
The input to the algorithm is $M = (Q, O, \pi, A, B)$ and the observed sequence $O = (O_{\omega_1}, O_{\omega_2}, \ldots, O_{\omega_T})$ of length $T$.

The output is the probability $tprob$.

The Forward Algorithm

begin
  for $j = 1$ to $n$ do
    $score(j, 1) = \pi(j)B(j, \omega_1)$
  endfor;
  for $t = 2$ to $T$ do
    for $j = 1$ to $n$ do
      for $k = 1$ to $n$ do
        $tscore(k) = score(k, t - 1)A(k, j)B(j, \omega_t)$
      endfor;
      $score(j, t) = \sum_k tscore(k)$
    endfor
  endfor;
  $tprob = \sum_j score(j, T)$
end
We can now run the above algorithm on $M_1, \ldots, M_L$ to compute $tprob_1, \ldots, tprob_L$, and we pick the model $M_\ell$ for which $tprob_\ell$ is maximum.

As for the Viterbi algorithm, the time complexity of the forward algorithm is $O(n^2T)$.

**Example 3.3.** To illustrate the forward algorithm, assume that our observant student also recorded the drinking behavior of a professor at Harvard, and that he came up with the HHM shown in Figure 3.3.

![Figure 3.3: Example of an HMM modeling the “drinking behavior” of a professor at Harvard.](image-url)
However, the student can’t remember whether he observed the sequence NNND at Penn or at Harvard.

So he runs the forward algorithm on both HMM’s to find the most likely model. Do it!