Topic Two: Hidden Markov Models and the Viterbi Algorithm

We start with an example, the problem of speech recognition.

A phoneme is an “elementary particle” of human speech, so to speak. Spoken words of a particular language are understood by a listener by parsing them into a sequence of phonemes. There is no general consensus what exactly the phonemes are, even for a specific language such as English. For all practical purposes we can think of them as the basic sounds of a language, which have the property that replacing one phoneme in a word with a different phoneme would change the meaning of the word. For example, if you replace the last phoneme /s/ in “kiss” with phoneme /l/ you obtain a different word, “kill”. However, we identify variants of the same sound as a single phoneme, for example the “k” sound in words “cat”, “kit” and “skill” are considered all to be slight variants of a single basic phoneme /k/, and replacing one such variant with another would not cause the listener to mistakenly understand the word being pronounced.

So let us now consider a sentence being spoken, for example “I love algorithms”. We understand this sentence by breaking the speech into constituent phonemes. I tried to reproduce the International Phonetic Alphabet (IPA) transcription in LaTeX, but could only get an approximation; so the utterance of “I love algorithms” would look approximately like this: \textit{al\ lav\ algol\@mz}.

Humans are very good at understanding speech, but machines got good at that only quite recently. Of course, machines do not have a direct access to the sequence of the phonemes spoken; they only have access to the resulting (sampled) sound waveforms, and from such waveforms they have to “decide” (if this is the right word for what speech recognition machines do) what phonemes were spoken. In order to do that a machine has to:

1. partition the waveform into disjoint pieces, so that each piece should correspond to a single phoneme (this is tricky in itself, figuring out where the boundaries between the waveforms of adjacent phonemes are);
2. then it has to do feature extraction from each piece, thus producing features vectors \(\vec{f}[1], \vec{f}[2], \ldots\) (this is usually done by taking the DFT of each piece and then looking for features in the frequency domain);
3. finally, it has to decide for each feature vector \(\vec{f}[i]\) to which phoneme it corresponds, i.e., it has to decide what phonemes would produce as their signatures each of the feature vectors \(\vec{f}[i]\).

Note that some phonemes are only slightly different and they also vary a lot from speaker to speaker, so it is hard to determine with absolute certainty from a feature vector \(\vec{f}[i]\) what phoneme “caused” a waveform that resulted in that particular feature vector.

One can model human speech as a Markov chain, whose states are the phonemes, and an utterance is an evolution of a Markov chain through such states, i.e., phonemes. This Markov chain is hidden from the listener.
So given a sequence of such manifestations $\vec{f}[1], \vec{f}[2], \ldots, \vec{f}[10]$ how can we decide what sequence of phonemes caused it, in a way that maximizes the likelihood that we are correct? This is what the Viterbi algorithm does. It is a dynamic programming algorithm which produces a sequence of states that are most likely to have as a consequence the sequence of “observations” $\vec{f}[1], \vec{f}[2], \ldots, \vec{f}[10]$. The Viterbi algorithm has a huge range of applications, including in speech recognition, also for decoding the transmitted convolutional codes in presence of noise in communication channels, which is used in digital telecommunications such as the CDMA mobile phone network and 802.11 wireless LANs; it is also used in many other fields such as bioinformatics and computational linguistics.

**General setup.** We are given a finite Markov chain, consisting of a set of its states $S = \{s_1, s_2, \ldots, s_K\}$ with initial probabilities $\pi_1, \pi_2, \ldots, \pi_K$, where $\pi_i$ is the probability that the chain starts in state $s_i$, and its matrix $M = \{p(i,j) : 1 \leq i, j \leq K\}$ of size $K \times K$ of transition probabilities $p(i,j) = p(s_i \rightarrow s_j)$ (so if at an instant $t$ the chain is in state $s_i$, then $p(i,j)$ is the probability that at the next instant $t+1$ the chain will be in state $s_j$). We have no direct access to the states of such a Markov chain, but we do have a set of observables $O = \{o_1, o_2, \ldots, o_N\}$ which are the manifestations of the states $S = \{s_1, s_2, \ldots, s_K\}$ and the *emission matrix* $E$ of size $K \times N$ where entry $e(i,k)$ represents the probability that when the chain is in state $s_i$ the observable outcome will be $o_k$.

Assume now that we are given a sequence of observable outcomes $(y_1, y_2, \ldots, y_T)$; the goal is to find the sequence $(x_1, x_2, \ldots, x_T)$ of states of the Markov chain for which the probability of such a sequence to be the cause of observing the sequence of outcomes $(y_1, y_2, \ldots, y_T)$ is the largest. In other words, we are looking for the *most likely cause* of the sequence of observations $(y_1, y_2, \ldots, y_T)$. In our speech recognition example we are given a sequence of feature vectors $(\vec{f}[1], \vec{f}[2], \ldots, \vec{f}[10])$ extracted from the sound waveform and we have to find what sequence of phonemes $(\phi_1, \phi_2, \ldots, \phi_{10})$ is the most likely cause of the sequence of the feature vectors $(\vec{f}[1], \vec{f}[2], \ldots, \vec{f}[10])$.

---

1We will abuse the notation for technical convenience and sometimes write $p(s_i, s_j)$ and sometimes just $p(i,j)$ and again sometimes $p(s_i \rightarrow s_j)$ for the probability of the Markov chain transitioning from state $p_i$ into state $p_j$. Similarly, sometimes we write $e(i,j)$ and sometimes $e(s_i, o_j)$ for the probability that state $s_i$ causes the observable outcome $o_j$. 

---

![Figure 1: Hidden Markov Model (HMM)](image-url)
Theoretically, we could solve this problem by brute force: we could go through all possible sequences of states \( \vec{x} = (x_1, x_2, \ldots, x_T) \) of the same length as the sequence of observables \( \vec{y} = (y_1, y_2, \ldots, y_T) \) and compute the corresponding probabilities that each of such sequences might be the cause for the given sequence of observables \( \vec{y} = (y_1, y_2, \ldots, y_T) \) and then select the sequence \( \vec{x} \) for which such probability is the largest. This could, theoretically, be done as follows. We first find the probability \( P(\vec{x}) \) that a sequence \( \vec{x} \) occurred at all, as the following product:

\[
P(\vec{x}) = \pi(x_1) \cdot p(x_1 \rightarrow x_2) \cdot p(x_2 \rightarrow x_3) \cdots p(x_{T-1} \rightarrow x_T)
\]

because \( \pi(x_1) \) is the probability that the starting state (phoneme in our example) is \( x_1 \), times the probability \( p(x_1 \rightarrow x_2) \) that system transitions from state \( x_1 \) into state \( x_2 \) and so forth.

Next, we compute the probability \( P^*(\vec{x}, \vec{y}) \) that this particular sequence of states \( \vec{x} = (x_1, \ldots, x_T) \) will cause the sequence of observable outcomes \( \vec{y} = (y_1, \ldots, y_T) \), as the product of the probabilities \( e(x_1, y_1) \cdot e(x_2, y_2) \cdots e(x_T, y_T) \) where each \( e(x_i, y_i) \) is the "emission" probability that state \( x_i \) produces the observable manifestation \( y_i \).

Finally, the total probability \( P(\vec{x}, \vec{y}) \) that sequence \( \vec{x} \) is the cause of the sequence of observable outcomes \( \vec{y} \) is the product of the probabilities \( P(\vec{x}) \) that sequence \( \vec{x} \) occurred at all and the probability \( P^*(\vec{x}, \vec{y}) \) that, if it did occur, it also caused the right observable outcome \( \vec{y} \), so in total \( P(\vec{x}, \vec{y}) = P(\vec{x}) \cdot P^*(\vec{x}, \vec{y}) \).

So given the sequence of outcomes \( \vec{y} = (y_1, y_2, \ldots, y_T) \) observed, we could now simply pick the sequence of states \( \vec{x} \) for which the value of \( P(\vec{x}, \vec{y}) \) is the largest. However, this is not a feasible algorithm: if the total number of states is \( K \) and the observed sequence is of length \( T \) then there would be in total \( N^T \) many sequences to try, i.e., exponentially many which is no good for real time (or almost real time) algorithms such as speech recognition or decoding the convolutional codes used in mobile telephony. Instead, the Viterbi algorithm solves this problem in time \( O(T \times K^2) \) using dynamic programming.

The Viterbi Algorithm We solve all subproblems \( S(i, k) \) for every \( 1 \leq i \leq T \) and every \( 1 \leq k \leq K \):

Subproblem \( S(i, k) \): Find the sequence of states \((x_1, \ldots, x_i)\) such that \( x_i = s_k \) and such that the probability of observing the outcome \((y_1, \ldots, y_i)\) is maximal.

Thus, we solve subproblems for all truncations \((x_1, \ldots, x_i)\), \(1 \leq i \leq T\), but also insisting that the last state \( x_i \) of such a truncated subsequence be state \( s_k \).

Let us denote by \( L(i, k) \) the largest possible probability for which there exists a sequence \((x_1, x_2, \ldots, x_i)\) ending with \( x_i = s_k \) causing the subsequence of observable outcomes \((y_1, y_2, \ldots, y_i)\); such probabilities must satisfy the following recursion:

\[
L(1, k) = \pi_k \cdot e(x_k, y_1) \\
L(i, k) = \max_{m \in S} \left(L(i-1, m) \cdot p(s_m \rightarrow s_k) \cdot e(s_k, y_i)\right) \\
\sigma(i, k) = \arg \max_{m \in S} \left(L(i-1, m) \cdot p(s_m \rightarrow s_k) \cdot e(s_k, y_i)\right).
\]

Here \( \sigma(i) \) stores the value of \( m \) for which \( L(i-1, m) \cdot p(s_m \rightarrow s_k) \cdot e(s_k, y_i) \) is the largest which allows us to reconstruct the sequence of states which maximize the probabilities we are tracking. We now obtain the solution for our original problem by backtracking:

\[
x_T = \arg \max_{m \in S} (L(T, m)) \\
x_{i-1} = \sigma(i, x_i), \quad i = T, T-1, \ldots, 2.
\]

A Python code for this algorithm can be found at, https://en.wikipedia.org/wiki/Viterbi_algorithm, together with some examples and even animations. Please take a look to make sure you learn this most important algorithm with so many diverse applications.

Nets class we will study voting algorithms.