Dynamic Programing

Problem: A sneaky gambler visits a casino to play a game involving a die. The casino uses only a fair die, but the gambler brings with him an identical looking loaded die and can swap which die he uses without anyone without anyone noticing. He may therefore swap back and forth between die in order to hide his usage of the loaded die. The only discernable difference is the apparent probability of the values he rolls over lengths of time.

The probability the gambler switches die between rolls in either direction is 0.05. The probability of getting each value from the fair and loaded dice are given here:

	Fair	Loaded
1	1/6	1/10
2	1/6	1/10
3	1/6	1/10
4	1/6	1/10
5	1/6	1/10
6	1/6	1/2

Given a sequence of rolls the gambler makes, we want to know at which points he used the loaded die. Obviously we can't determine this for certain, but we can determine the most likely sequence of die swaps the gambler used to get the given sequence of rolls. How might we do this?

Viterbi Algorithm

Think of this problem in terms of a state diagram. Each die is a state, with some transition probability to the other state, and each state has a probability of outputting each value. We can associate each output in our sequence with being in some state in our state diagram at that time. Our sequence of outputs is thus a result of transitioning from some previous state in our diagram to some state s, and then emitting a value from s, where the transitions and emissions occur with the given probabilities. If we associate each output in the sequence with the act of emitting that output from some state at that time, then our goal is to find the traversal through this state diagram over time that maximizes the probability of generating the observed output sequence.

To generalize and formalize, we want to find a path π^* that maximizes the total joint probability:

$$P(x, \pi) = a_{0\pi_1} * \prod_i e_{\pi_i}(x_i) \times a_{\pi_i \pi_{i+1}}$$

where $a_{0\pi_1}$ is the probability of starting in state π_1, $e_{\pi_i}(x_i)$ is the probability of outputting x_i in state π_i, and $a_{\pi_i \pi_{i+1}}$ is the probability of transitioning from state π_i to state π_{i+1}.

How do we solve this with Dynamic Programming? Notice that if the most probable traversal through the state diagram uses state k at that time i, it must also use an optimal traversal through the states to generate the output and get to state k at time i. (The proof for this is a cut-and-paste argument.) The probability of this traversal for the subsequence is the same as the probability for a path through the states given above.

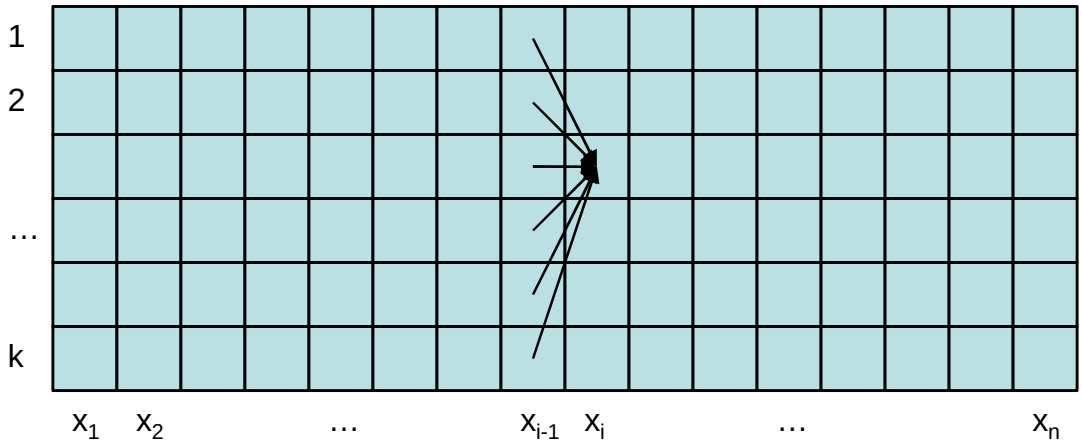

Figure 1: Model of the computation for the Viterbi algorithm. The output sequence is placed along the horizontal axis, while the states are aligned along the vertical axis. The dependencies for computing the probability of being in some state at time i from the probabilities of being in each state at time $i-1$ are shown.

Therefore for all i we can compute the $V_k(i)$, the probability that we were in state k at time i given the output sequence $x_1x_2...x_i$ by computing $V_k(i) = \max_j(a_{jk}V_j(i-1)) * e_k(x_i)$, where j is a possible state of the state machine at time $i-1$. By applying this formula for all states k and increasing times i we can model the computation as progressing through the chart in figure 1.

From this strategy for computing these probabilities, we get most of a dynamic programming algorithm to compute the value we want. We first initialize the entry in figure 1 for each state at time 1 to be the probability of starting in that state times the probability that x_1 was emitted from that state. We compute the rest of the values in this chart using the above recursive formula. To terminate we choose the maximum value over the states k of $V_k(n)$, and follow the pointers in the chart backwards to extract the actual path. (Note that in practice these probabilities become very small after only a few iterations, so the log of the probabilities is used instead.)

The space consumption of this dynamic program is $O(Kn)$ for K states and n outputs, as seen from the chart. Each entry in the chart requires looking at all K entries in the previous column of the chart, so the total runtime of this algorithm is $O(K^2n)$.

This algorithm is very useful for answering these sorts of queries for Hidden Markov Models. A Hidden Markov Model (HMM) models a process whose workings are unknown to us as a state machine with transition and emission probabilities. Once an HMM is trained we can use the Viterbi algorithm to extract the most likely path through the states of the HMM that generated some given output sequence, and from these states make a prediction concerning what happened to generate that output sequence.