Hidden Markov models are a widely used tool in NLP (and beyond). They define probability distributions over sequential data with labels and observations at each position. Formally, let Q be a set of normal states, not including a special start state q_0 and a special final state q_f. Let Σ be an alphabet of observable symbols. A single "event" generated by an HMM equates to a random walk among the states, starting in q_0 and ending in q_f; at every state along the walk excepting q_0 and q_f, a symbol from Σ is emitted. The random walk has a **Markov property**; the state we move to at time t depends only on the state we were in at time $t-1$. In the most common setting, we assume that the symbols (in Σ) are observed, but the state sequence is **hidden** and must be inferred.

The model is defined by two sets of conditional probability distributions. The **transition** distributions define the probability of moving from each state $q_i \in Q \cup \{q_0\}$ to another state $q_j \in Q \cup \{q_f\}$. Note that $\sum_{q_j \in Q \cup \{q_f\}} \gamma_{i,j} = 1 \ \ \forall\ q_i \in Q \cup \{q_0\}$. We write the transition probability from state q_i to state q_j, i.e. $p(q_j \mid q_i)$ as $\gamma_{i,j}$, and the entire set of transition probabilities is sometimes written as a matrix $\mathbf{\Gamma} = [\gamma_{i,j}]$. The **emission** distributions define the probability of emission of each symbol $w \in \Sigma$ from each state $q_i \in Q$, i.e., $p(w \mid q_i)$. (The initial and final states are silent, i.e., they emit nothing.) The probability $p(w \mid q_i)$ is sometimes written as $\eta_{i,w}$, where i indexes the state q_i and w is a symbol in Σ.

Using an HMM to **label** (or **infer**) an observation sequence in Σ^n is not trivial. Finding the *most probable* sequence of states corresponding to a given observation sequence $\boldsymbol{w} = \langle w_1, w_2, \ldots w_n \rangle$ requires the **Viterbi algorithm**, a dynamic programming algorithm not terribly different from the minimum edit distance algorithm. The recursive equations:

$$
\begin{aligned}
V[0, q_0] &= 1 \\
V[t, q_j] &= \max_{q_i \in Q \cup \{q_0\}} V[t-1, q_i] \cdot \gamma_{i,j} \cdot \eta_{j,w_t} \\
\text{goal} &= \max_{q_i \in Q} V[n, q_i] \cdot \gamma_{i,f}
\end{aligned}
$$

Here, $V[t, q_j]$ is the joint probability of the most probable tag sequence that starts in q_0, ends in q_j, and has generated the sequence of observations $w_1, w_2, \ldots, w_t$. Once we have solved for "goal," we need to backtrack (just like in minimum edit distance algorithms) to recover the best state sequence. To make this work, each "max" step needs to remember which state was the "arg max" in each max step.

Our working example—and one that is helpful in remembering how HMMs work—is to let the states correspond to POS tag N-grams, and the emissions correspond to words. The resulting model is a very commonly used POS tagger that can be understood in terms of a noisy channel, FSTs, Naïve Bayes, or linguistically-inspired language model. HMMs are also used for named entity recognition, Chinese word segmentation, and information extraction (among many others).