Hidden Markov Models

Sam Roweis

1 What Are HMMs ?

Hidden Markov Models (HMMs) are a class of models for mimicing the probability density of a sequence of observed symbols. They are essentially stochastic finite state machines which output a symbol each time they depart from a state. By specifying the state transistion probabilities between states and the symbol generation model for each state, we can attempt to capture the underlying structure in a large set of symbol strings.

In general, the operational paradigm is as follows: select a starting state according to some fixed probability distribution. At each time step, generate an output symbol by invoking the generative model of the current state, and then transition to a new state according to a static transistion probability matrix.

Let us define some notation for future convenience. Assume there are N possible states s_n in our model, and denote the set of all possible states by $\mathcal{S}$ where $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$. To this set, we will add a special state s_0 in which the model always starts as an implementational convenience – the use of this state will be explained later. We may choose to use one of the states (usually s_N) as an *end state* – in this case, whenever the model reaches this state, it stops. Let there be P possible output symbols a_p which comprise the output alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_P\}$. To this set, we add the special symbol a_0 which represents the null output – in other words if the model generates the symbol a_0 it simply does not output anything. Each state s_j has an output distribution defined by the vector A_j such that the probability of emitting symbol a when we are in state s_j is given by $A_j(a)$. Of course we require that:

$$\sum_{a \in \mathcal{A}} A_j(a) = 1 \qquad \forall s_j \in \mathcal{S}$$

There is also a matrix T of transistion probabilities defined so that T_{jk} is the probability of moving into state s_k if the model is currently in state s_j. Note that rows of T must sum to unity (although columns may not) and also that the diagonal elements of T need not be zero – we allow self transitions.

In order to eliminate the need to store the distribution of starting states separately, we make use of the special state s_0. The model always starts in this state, and never returns there. Hence the entire first column of T is filled with zeros, and the first row of T is just the distribution over the starting states. That is, T_{0j} is the probability that the model will start in the "real" state s_j. Also, no symbol is output when we depart from this state; hence $A_0(a)$ is 1 for $a = a_0$ and is 0 otherwise. All this just makes it easier to incorportae the starting state probabilities directly into the workings of the model without having to store them separately and do a special start-up step when we run the model.

Formally, then, we operate the model as follows. Denote the current state by x. Set $x = 0$. Loop until x is the end state or until time runs out: { Generate a symbol a according to probabilities $A_x(a)$. Pick the next state y accoring to probabilities T_{xy}. Set x=y.}.

2 Using many models to do recognition

Now we will examine how to use several HMMs in conjunction to perform a classification or recognition task.

Consider: given a large number of sequences each of which have been labelled as belonging to exactly one of several classes, how can we construct a system that tells us to which class an arbitrary new sequqnce (previously unseen) belongs. In other words, how can we use the examples of the various classes to extract salient features of those classes which we then make use of during recognition.

Hidden Markov models can be applied to this problem as follows: Use all of the sequences labelled as belonging to class C to train a generative HMM for class C. Do this for each class, so that we now have as many HMMs as classes. Now given an arbitrary sequence, one thing we could do would be to let all of the HMMs run freely for some long period of time, observing the sequences that they produced, and then ask which one produced our arbitrary sequence the greatest number of times. The class for which this HMM was the model would be the class into which we classified the arbitrary sequence. Conceptually this is fine, however in practice this is usually unrealistic because it would take forever to get good estimates. We would like some way to answer the classification question without having to run all our models explicitly and observe them for long periods of time. The answer is to compute the *likelihood* that each model has of generating the given sequence. and pick the class whose model has the largest computed likelihood. We now show how to do this computation efficiently.

Consider some sequence of symbols σ which is τ time steps long:

$$\sigma : \sigma_1, \sigma_2, \ldots, \sigma_\tau$$

There are many possible state trajectories that could have caused this sequence; we wish to find all of them and add up all their probabilities. Denote the set of state trajectories which could possibly have caused σ by $I : \{i_1, i_2, \ldots, i_Q\}$. Here, each element i of I is a state trajectory τ states long which represents the states the system visited (in order) while producing the symbol sequence σ. The state visited at any time t in a trajectory i is denoted by $i(t)$. Now define the likelihood that a particular model M caused the sequence σ:

$$L^\sigma(M) = Prob[\sigma_1, \sigma_2, \ldots, \sigma_\tau \mid model\ M]$$

$$= \sum_{i \in I} T_{0i(1)} A_{i(1)}(\sigma_1) T_{i(1)i(2)} A_{i(2)}(\sigma_2) \ldots T_{i(\tau-1)i(\tau)} A_{i(\tau)}(\sigma_\tau)$$

This seems like it is going to be an extremely hard factorial computation, requiring $O(\tau N^\tau)$ multiplies and $O(N^\tau)$ additions, not including the (non-trivial) amount of work required to compute which sequences should be in the set I. The exponential nature of this cost would make computing many such likelihoods prohibitive, and thus severly limit the practicality of using HMMs as recognizers. Luckily, however, there is an excellent trick (due originally to ?) which greatly saves on the cost of working this out:

Define $\alpha_j(t)$ as the incremental probabilites of having reached state s_j at time t having emitted the correct symbols up till then (including σ_t):

$$\alpha_t^\sigma(j) = Prob[\sigma_1, \sigma_2, \ldots, \sigma_t \quad \text{and state at time } t = s_j]$$

Now induction comes to our rescue:

$$\alpha_1^\sigma(j) = T_{0j} A_j(\sigma_1), \qquad \alpha_{t+1}^\sigma(k) = \left(\sum_{s_j \in \mathcal{S}} \alpha_t^\sigma(j) T_{jk} \right) A_k(\sigma_{t+1})$$

Which enables us to easily compute the desired likelihood $L^\sigma(M)$ since we know we must end in some possible state:

$$L^\sigma(M) = \sum_{s_k \in \mathcal{S}} \alpha_\tau^\sigma(k)$$

This can be done very cheaply, in only $O(N)$ multiplies and $O(N^2\tau)$ additions, and we do not need to compute the members of the set I.

3 Training a HMM

We have seen how if a single HMM represents a generative model for a class of sequences that we hope captures some of the structure inherent in the class then a group of HMMs can be useful in a recognition task. We now need to examine how to revise the parameters of a single HMM given some example sequences in order to make it a good generative model.

The idea is to update the parameters so that the model is more likely to generate the example sequences. This can either be done in an online way as we get each example, or once for all the examples together. Ideally, we would like to find the *maximum likelihood* model directly but there is currently no know way to do this efficiently. However, the *Baum-Welsh* algorithm gives us a prescription that always increases the likelihood of the model.

The alogrithm has a forward pass and a backward pass which are just like the E and M steps in EM. For each example sequence σ, do the following:

Forward pass: For each state s_j at each time t, compute the incremental probabilities $\alpha_t^\sigma(j)$ as explained above. **Backward pass:** For each state s_j at each time t, compute $\beta_t^\sigma(j)$ as the probability of starting in state s_j at time t and emitting the correct symbols after that point (including σ_t):

$$\beta_t^\sigma(j) = Prob[\sigma_t, \sigma_{t+1}, \ldots, \sigma_\tau \quad \text{and state at time } t = s_j]$$

which can once again be computed with a nice induction:

$$\beta_\tau^\sigma(j) = A_j(\sigma_\tau) \quad \forall s_j \in \mathcal{S}, \qquad \beta_t^\sigma(k) = A_k(\sigma_t) \sum_{s_j \in \mathcal{S}} T_{kj} \beta_{t+1}^\sigma(j)$$

Now compute $\gamma_t^\sigma(i,j)$ which is the expected number of transitions from state s_i to s_j that begin at time t (given the model M and the example sequence σ):

$$\gamma_t^\sigma(i,j) = \frac{\alpha_t^\sigma(i) T_{ij} \beta_{t+1}^\sigma(j)}{L^\sigma(M)}$$

The total expected number of state transistions from state s_i to state s_j, n_{ij} for the particular example σ is given by the sum over all starting times:

$$n_{ij}^\sigma = \sum_{t=1}^{\tau} \gamma_t^\sigma(i,j)$$

Finally, compute $\gamma_t^\sigma(j)$ which is the probability that the model is in state s_j at time t (given the model M and the example sequence σ):

$$\gamma_t^\sigma(j) = \frac{\alpha_t^\sigma(j) \beta_t^\sigma(j)}{A_j(\sigma_t) L^\sigma(M)}$$

The new model parameters are obtained by taking the averages over all example sequences σ of each of the following quantities:

$$T'_{0j} = \gamma_1(j), \qquad T'_{ij} = n_{ij} \bigg/ \sum_{s_j \in \mathcal{S}} n_{ij}\ , \qquad A'_j(a) = \sum_{t|\sigma_t = a} \gamma_t(j) \bigg/ \sum_{t=1}^{\tau} \gamma_t(j)$$

Each of these update rules has a nice interpretation. The new value for an initial state probability T_{0j} is just the average probability that the model will be in the state s_j at $t = 1$, the first time step. The new transistion probability T_{ij} is simply the ratio of the expected number of state transitions from s_i to s_j to the expected total number of transistions out of state s_i. The new probability $A_j(a)$ of generating the symbol a when leaving state s_j is just the ratio of the probability that the model was in state s_j *and* generated the symbol a to the total probability that the model was in state s_j.

Baum, Soules, and Weiss [1] have proven a theorem (originally proposed by Baum and Eagon in [2]) which guarentees that following this procedure will *increase* the liklihood of the model generating the example sequences (unless the current model defined a critical point of the liklihood function in which case the liklihood stays the same but does not decrease). That is, $< L^\sigma(M') >$ is guarenteed to be greater than $< L^\sigma(M) >$ where the angled brackets denote averages over all example sequences.

4 Extending the Symbol Generation Models

In all of the discussion above, I have assumed that the values of $A_j(a)$ were the actual probabilities of generating the symbol a when leaving state s_j. However, it is possible for each state to use a more complex symbol generation model, such as a gaussian[1] (or fancier) distribution or even a feed-forward neural network. Then each state has its own private values for the parameters of this model (for example, the means and variances or the network weights) which are used whenever that state must generate a symbol. While this makes HMMs potentially much more powerful, it also makes them much more of a hassle to work with. To answer the classification question: Which model is most likely to have generated this sequence ?, we still must be able to compute the individual symbol generation probabilities $A_j(a)$ from these more complex models. Even worse than this, updating our model parameters given some example sequences now becomes very difficult since we must find some to backpropagate through each state's generative model.

5 References

[1] L.E. Baum, T. Petrie, G. Soules, N. Weiss, Maximization Technique Occuring in the Statistical Analysis of Probabilistic Functions of Markov Chains, *The Annals of Mathematical Statistics 1970*, Volume 41 Number 1, pp. 164-171

[2] L.E. Baum, J.A. Eagon, An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology, *Bulletin of the American Mathematical Society*, May 1967, Volume 73, pp. 360-363.

[1] In fact, we can think of a mixture of gaussians model as a very simple HMM that has as many states as gaussians in the model. We start in state 0, go to one of the states, generate a data point according to the private gaussian parameters of that state, and then return to the start state. The inital state probabilities T_{0j} are just the mixing coefficients of the gaussians, and the mean and variance of each gaussian are just the model parameters for each state.