HIDDEN MARKOV MODELS AND SEQUENCE ALIGNMENT

- Swarbhanu Chatterjee.

Hidden Markov models are a sophisticated and flexible statistical tool for the study of protein models. Using HMMs to analyze proteins is part of a new scientific field called *bioinformatics,* based on the relationship between computer science, statistics and molecular biology. Because of large government-sponsored projects like the *Human Genome Project* in the United States, there has been an exponential increase in the quantity of data available about proteins, DNA, and RNA. Traditional lab methods of studying the structure and function of these molecules are no longer able to keep up with the rate of new information. As a result, molecular biologists have turned to statistical methods capable of analyzing large amounts of data, and to computer programs which implement these methods

Hidden Markov models (HMMs) offer a more systematic approach to estimating model parameters. The HMM is a dynamic kind of statistical profile. Like an ordinary profile, it is built by analyzing the distribution of amino acids in a training set of related proteins. However, an HMM has a more complex topology than a profile. It can be visualized as a *finite state machine.*

Finite state machines typically move through a series of states and produce some kind of output either when the machine has reached a particular state or when it is moving from state to state. The HMM generates a protein sequence by *emitting* amino acids as it progresses through a series of states. Each state has a table of amino acid *emission probabilities* similar to those described in a profile model. There are also *transition probabilities* for moving from state to state. < please refer to Figure 1 >

Figure 1 shows one topology for a hidden Markov model. Although other topologies are used, the one shown is very popular in protein sequence analysis. Note that there are three kinds of states represented by three different shapes. The squares are called *match states*, and the amino acids emitted from them form the conserved primary structure of a protein. These amino acids are the same as those in the common ancestor or, if not, are the result of substitutions. The diamond shapes are *insert states* and emit amino acids which result from insertions. The circles are special, silent states known as *delete states* and model deletions. Transitions from state to state progress from left to right through the model, with the exception of the self-loops on the diamond insertion states. The self-loops allow deletions of any length to fit the model, regardless of the length of other sequences in the family.

Any sequence can be represented by a path in the model. The probability of any sequence, given the model, is computed by multiplying the emission and transition probabilities along the path. In order to avoid floating point errors etc during computations, its best to transform the probabilities to their logarithms. The resulting score is called the raw score of the sequence, given the HMM.

It might be the case that there may be many paths for going from one sequence to another. Then we would have to add up the probabilities of each path. A brute force calculation of this problem is computationally unfeasible, except in the case of very short sequences. Two good alternatives are to calculate the sum over all paths inductively using the *forward algorithm,* or to calculate the most probable path through the model using the *Viterbi algorithm*. Both algorithms are described below

The Viterbi model tries to first find out the most probable path and then the probability of the sequence given the HMM model is computed by multiplying all probabilities along the path. What the algorithm basically does is that at every stage in its journey, it tries to figure out the most probable path leading from a particular amino acid's emission to another's. It selects the most probable path to go from one amino acid emission to another amino acid emission after having compared the probability scores corresponding of each path. The algorithm does the above every time it wants to go from one amino acid to another. The resultant path that stretches along the whole sequence of amino acids is said to be the most probable path.

The algorithm employs a matrix. The columns of the matrix are indexed by the states in the model, and the rows are indexed by the sequence. Deletion states are not shown, since, by definition, they have a zero probability of emitting an amino acid. The elements of the matrix are initialized to zero. Then the scheme which has been described in general terms in the above paragraph is used to get a matrix with zeroes in all positions except for the ones that correspond to the most probable path. From the final matrix, we can not only read out the most probable path but also by multiplying the non-zero entries in the matrix, and taking the log, we can calculate the *raw score*.

In the forward algorithm, the strategy is to sum over all path inductively and after that decide which is the most probable path and what its raw score is. Therefore, what this algorithm does is that at every stage it tries to sum over the probabilities of all the paths leading from one amino acid emission to another amino emission < instead of taking the max as the Viterbi algorithm did >. It does this recursively for each leap from one amino acid emission to another. Therefore, at the end of the day, we end up with *the summed over* probability of generating the required sequence of amino acids.

In conclusion, we note that the Viterbi algorithm tells us only the most probable path and its raw score while the forward algorithm investigates the entire phase space with regard to possible paths and gives us the total probability of generating the string of amino acid emissions. The Viterbi algorithm is more economical in terms of computation and is therefore faster because it investigates only a part of the phase space corresponding to the possible paths.

The making of an HMM

Another tricky problem is how to create an HMM in the first place. It is necessary to estimate the amino acid emission distributions in each state and all state-to-state transition probabilities from a set of related training sequences.

If the state paths for all the training sequences are known, the emission and transition probabilities in the model can be calculated by computing their *expected value*. This is done by observing the number of times each transmission or emission occurs in the training set and dividing by the sum of all the transmission probabilities or all the emission probabilities.

If the state paths are unknown, finding the best model given the training set is an optimization problem which has no closed form solution. It must be solved by iterative methods.

The algorithms used to do this are closely related to the scoring algorithms described above. The goal is to find model parameters which maximize the probability of all sequences in the training set. In other words, the desired model is a model against which all the sequences in the training set will have the best possible scores. < This is however not a consistent thing to do because it will completely depend on the suitability of the training sequence. There would be no real justification for choosing a particular training set other than it gives the answers we are looking for and it will give rise to a guessing game >

The parameters are re-estimated after every iteration by computing a score for each training sequence against the previous set of model parameters.

Let us now describe two algorithms for the above –
The *Baum-Welch* algorithm is a variation of the forward algorithm described earlier. It begins with a reasonable guess for an initial model and then calculates a score for each sequence in the training set over all possible paths through this model . During the next iteration, a new set of expected emission and transition probabilities is calculated, as described above for the case when state paths are known. The updated parameters replace those in the initial model, and the training sequences are scored against the new model. The process is repeated until *model convergence,* meaning there is very little change in parameters between iterations.

The *Viterbi* algorithm is less computationally expensive than Baum-Welch. As described in the previous section, it computes the sequence scores over the most likely path rather than over the sum of all paths and therefore saves in computation and is faster but since it doesn't investigate the entire phase space, it is not as rigorous.

However, there is no guarantee that a model built with either algorithm has parameters which maximize the probability of the training set. As in many iterative methods, convergence indicates only that a local maximum has been found. Several heuristic methods have been developed to deal with this problem. One approach is to start with several initial models and proceed to build several models in parallel. When the models converge at several different local optimums, the probability of each model given the training set is computed, and the model with the highest probability wins. Another approach is to add *noise,* or random data, into the mix at each iteration of the model building process. Typically, an *annealing schedule* is used. The schedule controls the amount of noise added during each iteration. Less and less noise is added as iterations proceed. The decrease is either linear or exponential. The effect is to delay the convergence of the model. When the model finally does converge, it is more likely to have found a good approximation to the global maximum.

Sequence Weighting

The selection of a suitable training set is very important when it comes to building a good HMM. If there is a small group of sequences in the training set which are highly similar, the model will overspecialize to the small group. To prevent this, sequence weighting is done in which sequences which do not belong to the highly similar group are given a greater weight, and therefore a greater importance. In order to do this one way is to first construct an alignment tree for the different sequences from which we can read off the sequences which don't belong to the highly similar groups and therefore should be given greater weight. But in order to build this tree we would have to use some other method of sequence alignment.

There is a way to do things so that we don't have to use a different alignment technique for constructing the tree. This approach is *called the maximum discrimination* weighting. In this method, weights are estimated iteratively while the model is being built. After each iteration in the model building process, the sequences in the training set are scored against the current model. Weights are assigned to each sequence, with poorly scoring sequences receiving the highest valued weights. During the next iteration, these sequences get more importance than sequences which had good scores during the previous round. The process repeats until the model converges

A third scheme of weighting is based on trying to make the statistical spread in the model as uniform as possible. There are also other schemes. It is not possible to say which is the best possible scheme. The scheme that we choose has to be specific to the model and the group of proteins we want to investigate.

Uses of HMMs

HMMs are used for classifying sequences. The method is to build to build an HMM with a training set of known members of the class, and then to compute the score of all sequences in the database. HMMs are also used to create a multiple alignment from a group of unaligned sequences.

References :
1. R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Computer Applications in the Biosciences,* 12:95-107. 1996.
2. S. Henikoff and J.G. Henikoff. Position-based sequence weights. *Journal of Molecular Biology,* 243:574-578. 1994.
3. C:\WINNT\Profiles\schatter\Desktop\Goldenfeld\Hw3\generation5_org - An Introduction to Markov Models.htm

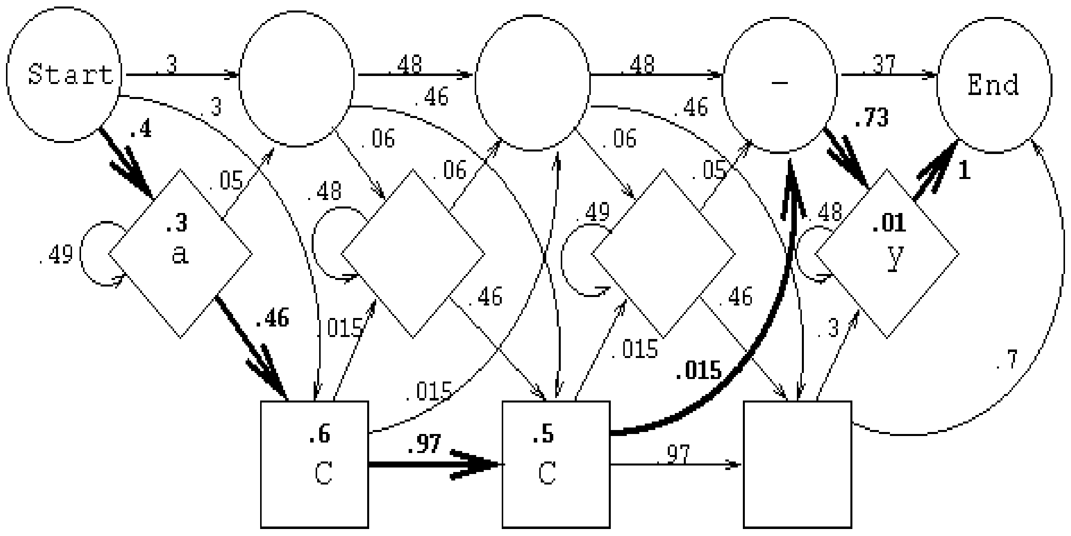

Figure 1. A possible hidden Markov model for the protein ACCY. The protein is represented as a sequence of probabilities. The numbers in the boxes show the probability that an amino acid occurs in a particular state, and the numbers next to the directed arcs show probabilities which connect the states. The probability of ACCY is shown as a highlighted path through the model.