Detailed Explanation and Implementation of the Baum-Welch Algorithm for Protein Secondary Structure Prediction

Michele Copetti

May 9, 2024

1 Introduction

HMMs are particularly useful in modeling systems where we observe sequences of symbols generated by processes that operate on states hidden from the observer. The Baum-Welch algorithm, an application of the Expectation-Maximization (EM) technique, is used to find the unknown parameters of an HMM when only the output observations are known. This is crucial for applications like protein secondary structure prediction, where the actual states (secondary structure types such as helices, sheets, or coils) are not directly observable from the amino acid sequence.

2 Theory of Hidden Markov Models

An HMM is defined by:

- A set of states $S = \{s_1, s_2, \ldots, s_N\}$.
- A set of possible observations $V = \{v_1, v_2, \ldots, v_K\}$.
- Transition probabilities $A = [a_{ij}]$ where $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$.
- Emission probabilities $B = [b_j(k)]$ where $b_j(k) = P(o_t = v_k \mid q_t = s_j)$.
- Initial state probabilities $\pi_i = P(q_1 = s_i)$.

3 Baum-Welch Algorithm

The Baum-Welch algorithm iteratively adjusts the parameters of the HMM to maximize the likelihood of the observed data. It comprises two main steps: the Expectation (E) step and the Maximization (M) step.

3.1 Forward Algorithm

The forward probabilities $\alpha_t(j)$ are defined as the probability of the observed sequence up to time t and being in state j at time t. The recursion is given by:

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \right] b_j(o_t)$$

Initiated by $\alpha_1(i) = \pi_i b_i(o_1)$ for all i.

3.2 Backward Algorithm

The backward probabilities $\beta_t(i)$ indicate the probability of the observed sequence from time $t+1$ to the end, given state i at time t. The recursion is:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

Starting with $\beta_T(i) = 1$ for all i.

3.3 Expectation Step

This step involves calculating two sets of probabilities based on the current parameter estimates: $\xi_t(i,j)$ and $\gamma_t(i)$. These probabilities are crucial for the subsequent Maximization step.

3.3.1 Calculation of $\xi_t(i,j)$

The probability $\xi_t(i,j)$ represents the expected number of transitions from state i at time t to state j at time $t+1$, given the entire observed sequence. It is computed as:

$$\xi_t(i,j) = \frac{\alpha_t(i) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) \cdot a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)}$$

This expression uses the forward probability $\alpha_t(i)$, the backward probability $\beta_{t+1}(j)$, the transition probability a_{ij}, and the emission probability $b_j(o_{t+1})$.

3.3.2 Calculation of $\gamma_t(i)$

The probability $\gamma_t(i)$ indicates the expected number of times the system is in state i at time t, given the entire observed sequence. It can be derived from the forward and backward probabilities as follows:

$$\gamma_t(i) = \frac{\alpha_t(i) \cdot \beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j) \cdot \beta_t(j)}$$

This formula ensures that $\gamma_t(i)$ is the normalized product of the forward probability $\alpha_t(i)$ and the backward probability $\beta_t(i)$, reflecting the joint probability of being in state i at time t given the observed sequence.

3.3.3 Why These Calculations Matter

The calculations of $\xi_t(i,j)$ and $\gamma_t(i)$ are foundational for the Baum-Welch algorithm. They allow us to estimate how often particular transitions occur and how often each state is visited within the observed data. This information is then used in the Maximization step to adjust the model's parameters in a way that maximizes the likelihood of the observed data under the model.

These quantities represent the expected number of transitions from state i to state j and the expected number of times state i is visited, respectively.

3.4 Maximization Step

Using the expected frequencies computed in the E-step, the parameters of the model are updated as follows:

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$b_j(k) = \frac{\sum_{t=1}^{T} \gamma_t(j) 1_{\{o_t=k\}}}{\sum_{t=1}^{T} \gamma_t(j)}$$

Here, $1_{\{o_t=k\}}$ is an indicator function that is 1 if $o_t = k$ and 0 otherwise.

4 Implementation

Each function and loop in the code directly corresponds to a part of the mathematical formulation described above.

5 Conclusion

The Baum-Welch algorithm provides a robust framework for training HMMs in situations where only observations are known, and the states are hidden.

6 Expectation-Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm is a powerful iterative method used for estimating the parameters of statistical models when the model depends on unobserved latent variables. EM is particularly useful in scenarios where direct computation of maximum likelihood estimates is not feasible due to the complexity introduced by the latent (non-observable) variables.

6.1 General Framework

The EM algorithm alternates between two steps—Expectation (E-step) and Maximization (M-step)—to find parameter estimates that maximize the likelihood of the observed data. The algorithm starts with initial guesses for the parameters and iteratively improves them based on the log-likelihood function.

6.1.1 Expectation Step (E-step)

In the E-step, the EM algorithm computes the expected value of the log-likelihood function, with respect to the conditional distribution of the latent variables given the observed data and the current estimates of the parameters. This step essentially fills in the best guesses of the missing data using the current model parameters. Mathematically, it calculates the expectation:

$$Q(\theta, \theta^{(t)}) = \mathbb{E}_{Z|X,\theta^{(t)}}[\log L(\theta; X, Z)]$$

where: - θ represents the parameters to be optimized, - $\theta^{(t)}$ is the parameter estimate at iteration t, - X represents the observed data, - Z represents the latent variables, - $L(\theta; X, Z)$ is the complete-data likelihood.

6.1.2 Maximization Step (M-step)

Following the E-step, the M-step involves updating the parameters to maximize the expected log-likelihood found in the E-step. This is usually more straightforward since it treats the estimates of the latent variables as if they were observed:

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta, \theta^{(t)})$$

This step updates the parameter estimates to those that maximize the expected log-likelihood, thereby improving the fit of the model to the data.

6.2 Convergence of the EM Algorithm

The EM algorithm is guaranteed to converge to a local maximum (or saddle point) of the likelihood function. At each iteration, the log-likelihood of the observed data either increases or remains the same, because the E-step computes a function that bounds the log-likelihood from below, and the M-step maximizes this function with respect to the parameters.

6.3 EM Algorithm in Hidden Markov Models

In the context of Hidden Markov Models (HMMs), the latent variables are the unknown states of the system, and the observed data are the symbols or signals produced by the system. The Baum-Welch algorithm is a specialized form of the EM algorithm tailored for HMMs, where: - The E-step corresponds to calculating the forward and backward probabilities, which are used to compute

the expected number of times each transition and emission occur. - The M-step involves recalculating the transition and emission probabilities to maximize the expected complete-data log-likelihood.

7 Viterbi Algorithm

The Viterbi Algorithm is a dynamic programming algorithm used to find the most probable sequence of states, known as the Viterbi path, in a Hidden Markov Model (HMM). This algorithm is particularly useful in scenarios where the goal is to determine the sequence of states that most likely resulted in a given sequence of observations. The primary purpose of the Viterbi algorithm is to decode the observed data by finding the sequence of hidden states that has the highest probability of producing the observed sequence.

7.1 Mathematical Formulation

The Viterbi algorithm uses dynamic programming to compute the most likely path through the model's states given the observed sequence. Let us denote:

- $O = \{o_1, o_2, \ldots, o_T\}$ as the sequence of observations.
- $Q = \{q_1, q_2, \ldots, q_N\}$ as the set of states in the model.

We define $V_t(j)$ as the probability of the most probable state path responsible for the first t observations that ends in state j. The Viterbi algorithm proceeds as follows:

7.1.1 Initialization

$$V_1(j) = \pi_j b_j(o_1) \quad \text{for } 1 \leq j \leq N$$

where π_j is the initial probability of state j and $b_j(o_1)$ is the probability of observing o_1 from state j.

7.1.2 Recursion

For each subsequent observation t from 2 to T, the probabilities are updated as follows:

$$V_t(j) = \max_{1 \leq i \leq N} (V_{t-1}(i) \times a_{ij}) \times b_j(o_t)$$

where a_{ij} is the probability of transitioning from state i to state j.

7.1.3 Termination

The probability of the most likely path P that explains the observed sequence is:

$$P = \max_{1 \leq j \leq N} V_T(j)$$

7.1.4 Path Backtracking

To find the actual path, we need to keep track of the state that achieved the maximum at each step. Let $\psi_t(j)$ be the state that maximized $V_t(j)$:

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} (V_{t-1}(i) \times a_{ij})$$

After computing $V_t(j)$ and $\psi_t(j)$ for all t and j, the most likely sequence of states (the Viterbi path) is obtained by backtracking from $\psi_T(j)$ where j is the argument that maximized $V_T(j)$.

7.2 Why It Works

The Viterbi algorithm is an application of dynamic programming that builds solutions to complex problems by combining solutions to subproblems. By storing the probabilities of paths ending in each state up to each point in time, it effectively reduces the complexity of the problem from exponential (involving all possible paths) to polynomial (only tracking the most likely path). This efficiency makes it practical for real-world applications where sequences may be long and state spaces large.

7.3 Asymptotic Complexity

The dynamic programming table has dimensions $N \times T$, where N is the number of states and T is the length of the observation sequence. For each cell in the table, the algorithm evaluates all possible transitions from each possible previous state. Thus, the computation for each cell involves:

$$\max_{1 \leq i \leq N} (V_{t-1}(i) \times a_{ij})$$

This calculation is performed N times (once for each possible previous state) for each of the N current states and repeated for each of the T time steps. Thus, the complexity of filling the dynamic programming table is $O(N^2T)$. Once the table is computed, the algorithm backtracks from the last observation to determine the most probable state sequence. This backtracking process traverses from $t = T$ back to $t = 1$, yielding a complexity of $O(T)$. The overall complexity of the Viterbi algorithm is dominated by the dynamic programming table computation, leading to a total complexity of:

$$O(N^2T) + O(T) \approx O(N^2T)$$