Analytics Vidhya · Follow publication

Baum-Welch algorithm for training a Hidden Markov Model — Part 2 of the HMM series

How to train a Hidden Markov Model and use it for filtering and smoothing?

7 min read · Jul 21, 2019

Raymond Kwok Follow

Part 1: Architecture of the Hidden Markov Model
Part 2: Algorithm to train a HMM: Baum-Welch algorithm

this article, we will talk about the algorithm for training up a HMM, before making use of it for prediction.

Baum-Welch algorithm

Also known as the forward-backward algorithm, the Baum-Welch algorithm is a dynamic programming approach and a special case of the expectation-maximization algorithm (EM algorithm). Its purpose is to tune the parameters of the HMM, namely the state transition matrix A, the emission matrix B, and the initial state distribution π_0, such that the model is maximally like the observed data.

There are a few phases for this algorithm, including the initial phase, the forward phase, the backward phase, and the update phase. The forward and the backward phase form the E-step of the EM algorithm, while the update phase itself is the M-step.

Initial phase

In the initial phase, the content of the parameter matrices A, B, π_0 are initialized, and it could be done randomly if there is no prior knowledge about them.

Forward phase

In the forward phase, the following recursive alpha function is calculated. For the deviation of the function, I would strongly recommend this YouTube video as the speaker presented it clearly and explained it very well.

$$\alpha(X_k) = P[Y_{0:k}, X_k] = \sum_{X_{k-1}} \alpha(X_{k-1}) P(X_k|X_{k-1}) P(Y_k|X_k)$$

There are a few points to make here:

1. The alpha function is defined as the joint probability of the observed data up to time k and the state at time k
2. It is a recursive function because the alpha function appears in the first term of the right hand side (R.H.S.) of the equation, meaning that the previous alpha is reused in the calculation of the next. This is also why it is called the forward phase.
3. The second term of the R.H.S. is the state transition probability from A, while the last term is the emission probability from B.
4. The R.H.S. is summed over all possible states at time k -1.

It should be pointed out that, each alpha contains the information from the observed data up to time k, and to get the next alpha, we only need to reuse the current alpha, and add information about the transition to the next state and the next observed variable. This recursive behavior saves computations of getting the next alpha by freeing us from looking through the past observed data every time.

By the way, we need the following starting alpha to begin the recursion.

$$\alpha(X_0) = P[Y_0, X_0] = P[Y_0|X_0]P[X_0]$$

the starting alpha is the product of probabilities of the emission and the initial state

Backward phase

Please refer to this YouTube video for the deviation of the following formula.

$$\beta(X_k) = P[Y_{k+1:T}|X_k] = \sum_{X_{k+1}} \beta(X_{k+1})P(X_{k+1}|X_k)P(Y_{k+1}|X_{k+1})$$

Similar points could be made here:

1. The beta function is defined as the conditional probability of the observed data from time k+1 given the state at time k
2. It is a recursive function because the beta function appears in first term of the right hand side of the equation, meaning that the next beta is reused in the calculation of the current one. This is also why it is called a backward phase.
3. The second term of the R.H.S. is the state transition probability from $\boldsymbol{A}$, while the last term is the emission probability from $\boldsymbol{B}$.
4. The R.H.S. is summed over all possible states at time k +1.

Again, we need the ending beta to start the recursion.

$$\beta(X_T) = 1$$

Hold on! Why the alpha and the beta functions?

Good question!

Firstly, as mentioned, they are both recursive functions, which means that we could reuse previous answer as the input for the next answer. This is what dynamic programming is about — you could save time by reusing old result!

Secondly, the formula in the forward phase is very useful. Suppose you have a set of well-trained transition and emission parameters, and given that your problem is to, in real-time, find out the mysterious hidden truth from observed data. Then you actually could do it like this! When you get one data point (data point *p*), then you could put it into the formula which will give you the probability distribution of the associated hidden state, and from which you could pick the most probable one as your answer. And the story does not stop here, as you get the next data point (data point *q*), and you put it again into the formula, it will give you another probability distribution for you to pick the best choice, but this is not only based on data point *q* and the transition and emission parameters, but also the data point *p*. Such use of the formula is called ***filtering***.

Thirdly, and continuing the above discussion, that suppose you collected many data points already, and because you know that the earlier the data point, the less observed data the choice of your answer based on. Therefore you would like to improve that by somehow 'injecting' information from the later data into the earlier ones. This is where the backward formula comes into play. Such use of the formula is called ***smoothing***.

Fourthly, this is about the combination of the last two paragraphs. With the help of the alpha and the beta formula, one could determine the probability distribution of the state variable at any time *k* given the whole sequence of observed data. This could also be understood mathematically.

$$P[X_k|Y_{0:T}] = \frac{\alpha(X_k)\beta(X_k)}{P[Y_{0:T}]} \propto \alpha(X_k)\beta(X_k)$$

The denominator term is a normalization constant and is usually dropped like this because it does not depend on the state, and therefore it is not important when comparing the probability of different states at any time k.

Lastly, the result from the alpha and the beta functions are useful in the update phase.

Update phase

$$\eta(X_k) = P[X_k|Y_{0:T}] = \frac{\alpha(X_k)\beta(X_k)}{\sum_{X_k}\alpha(X_k)\beta(X_k)}$$

$$\xi(X_k, X_{k+1}) = P[X_k, X_{k+1}|Y_{0:T}]$$

$$= \frac{\alpha(X_k)\beta(X_{k+1})P[X_{k+1}|X_k]P[Y_{k+1}|X_{k+1}]}{\sum_{X_k}\alpha(X_k)\beta(X_{k+1})P[X_{k+1}|X_k]P[Y_{k+1}|X_{k+1}]}$$

For the deviation of the above formulas, if you have watched the YouTube videos that I suggested for the forward and backward formula, and you can understand them, then probably you will have no problem to derive these two yourself.

The first formula here is just repeating what we have seen above, and to recap, it is to tell us the probability distribution of a state at time *k* given all observed data we have. The second formula, however, tells us a bit different thing which is the joint probability of two consecutive states given the data. They make use of the alpha function, the beta function, the transition and the emission that are already available. These two formulas are further used to finally do the update.

$$\pi_0^* = \eta(X_0)$$

$$a_{ij}^* = P[X_k = j|X_{k-1} = i] = \frac{\sum_k \xi(X_k = j, X_{k-1} = i)}{\sum_k \eta(X_{k-1} = i)}$$

$$b_{ij}^* = P[Y_k = j|X_k = i] = \frac{\sum_k \eta(X_k = i) \times 1_{Y_k = j}}{\sum_k \eta(X_k = i)}$$

The deviation steps are not shown here, because mathematics is not the intention of this article, but showing the formulas themselves would be useful for us to see how they are being re-used through the steps.

It was mentioned that the Baum-Welch algorithm is a case of EM algorithm. Here I will explain why very briefly. The alpha and the beta function form the E-step because they predict for the expected hidden states given the observed data and the parameter matrices A, B, π_0. The update phase is the M-step because the last three update formulas are so derived that the L.H.S. parameters will best fit the expected hidden states given the observed data.

Summary

The Baum-Welch algorithm is a case of EM algorithm that, in the E-step, the forward and the backward formulas tell us the expected hidden states given the observed data and the set of parameter matrices before-tuned. The M-step update formulas then tune the parameter matrices to best fit the observed data and the expected hidden states. And these two steps are then iterated over and over again until the parameters converged, or until the model has reached some certain accuracy requirement.

Like any machine learning algorithm, this algorithm could be overfitting the data, as by definition the M-step encourages the model to approach the observed data as good as possible. Also, although we have not talked too much about the initial phase, it indeed affects the final performance of the model (as a problem of trapping the model in local optimum), so one might want to try different ways of initializing the parameters and see what works better.

Machine Learning | Markov Chains | Baum Welch | Hidden Markov Models