Machine Learning and Real-world Data

Example Sheet 3 - Zejia Yang

February 28, 2024

Markov Assumption

1. Limited Horizon: The transition probability depends only on the current state. It simplifies the process of modelling probabilities and enables efficient computations. When making this assumption, we have to ensure that HMM captures the dynamics of the system being modelled.

$$P(X_t|X_1...X_{t-1}) = P(X_t|X_{t-1})$$

.

2. Time Step Independence: Transition and emission probability are the same across all time steps. It allows us to use the computed probabilities to predict future states and output reliably.

$$P(X_t = s_i|X_{t-1} = s_j) = P(X_2 = s_i|X_1 = s_j)$$

$$P(O_t = k_i|X_t = s_j) = P(O_1 = k_i|X_1 = s_j)$$

3. There is an assumption about output independence, but I thought it isn't a Markov Assumption,

HMM Artificial data

1. S: a set of N emitting hidden states, a start state s_0, an end state s_f.
2. K: an output alphabet of M observations ("vocabulary"), a start symbol k_0,an end symbol k_f.
3. transition probabilities A. $A_{ij} = P(X_t = s_j|X_{t-1} = s_i)$.
4. emission probabilities B. $B_{ij} = P(O_t = k_j|X_t = s_i)$.

Algorithm 1 Generating Artificial Data using HMM

Initialize:
  Set initial hidden state to be s_0.
  Set initial output to be k_0.
  Set time step $t = 0$.
repeat
    Increment time step: $t = t + 1$.
    Transition to a new hidden state based on A.
    Generate an observation based on B for the current hidden state.
until desired sequence length or time horizon is reached
  Append s_f to the end of the hidden state sequence.
  Append k_f to the end of the output sequence.
Return the sequence of generated observations and its hidden state.

Transition into the final state may influence the results when final state is attached with meanings (a specific outcome). Also, it depends on the system you are modelling; Examples include modelling sequences of speech acts in dialog systems, where the final state signifies the end of an sentence/document and the likelihood of reaching this state greatly affects the performance.

Smoothing in HMMs

1. Situations like:
   (a) You already have many high-quality data sequences. It accurately monitors and reflects the possibility. However, we won't know until testing. Thus, I believe it is dependent on the size of your available data. If you already have a large amount of data, smoothing may be counterproductive.
   (b) The number of states is relatively small. The data is likely to accurately estimate the transition probabilities between states. In such cases, additional smoothing may not provide significant benefits, particularly if the training data accurately represents the underlying process.
   (c) You have a large number of states/outputs, but you are confident that for each state, the transition will prioritise a few rather than distributing them evenly. Smoothing in this case may significantly alter your probabilities, resulting in a distorted distribution.
2. In the case of protein model, emission probabilities may be a better candidate for smoothing. Smoothing helps to address the sparsity issue by assigning non-zero probabilities to unseen observations, thus preventing zero probabilities for unseen amino acids and improving the robustness of the model. On the other hand, smoothing of transition probabilities may not be as critical, especially if the training data is sufficiently large and representative of the underlying process. In such cases, the transition probabilities may already be well-estimated from the data, and additional smoothing may not provide significant benefits while increasing computational complexity.

Viterbi and Forward algorithm

1

The recursion formula:

$$\alpha(j,t) = \sum_{i=1}^{N} \alpha(i,t-1) a_{ij} b_j(O_t)$$

1. $\alpha(j,t)$: the probability of being in state j after seeing the first t observations.
2. a_{ij}: transition probability from state i to state j.
3. $b_j(O_t)$: The observation likelihood of O_t at state j.

Given the Markov assumption, the next state only depends on the current state. We can derive $\alpha(j,t)$ by summing up the probability of transition from all the available states at $t-1$ given the next observation state.

2

Compared with the Viterbi algorithm, here is a summation instead of a maximum. It is because we want to calculate the probability of yielding an output sequence (O). Hence, we need to compute $P(O) = \sum_Q P(O|Q)P(Q)$.

Parts of Speech tagging with HMM

1

X_a: The observation sequence is interpreted as "We" (personal pronoun), "can" (auxiliary verb), and "fish" (verb). X_b: The observation sequence is interpreted as "We" (personal pronoun), "can" (noun), and "fish" (noun).

2

$$\begin{aligned} P(X_a) &= P(s_f|s_1)P(s_1|s_4)P(s_4|s_3)P(s_3|s_0) \\ &= a_{1f}a_{41}a_{34}a_{03} \\ &= 0.15 \times 0.73 \times 0.4 \times 0.6 \\ &= 0.02628 \end{aligned}$$

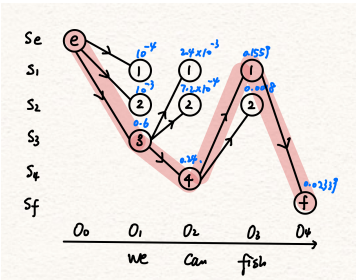
$$\begin{aligned} P(X_b) &= P(s_f|s_2)P(s_2|s_1)P(s_1|s_3)P(s_3|s_0) \\ &= a_{2f}a_{12}a_{31}a_{03} \\ &= 0.2 \times 0.63 \times 0.4 \times 0.6 \\ &= 0.03024 \end{aligned}$$

$$\begin{aligned} P(X_a, O) &= P(k_f|s_f)P(s_f|s_1)P(fish|s1)P(s_1|s_4)P(can|s_4)P(s_4|s_3)P(we|s_3)P(s_3|s_0)P(k_0|s_0) \\ &= a_{1f}\ b_1(fish)\ a_{41}\ b_4(can)\ a_{34}\ b_3(we)\ a_{03} \\ &= 0.15 \times 0.89 \times 0.73 \times 1 \times 0.4 \times 1 \times 0.6 \\ &= 0.02339 \end{aligned}$$

$$\begin{aligned} P(X_b, O) &= P(k_f|s_f)P(s_f|s_2)P(fish|s2)P(s_2|s_1)P(can|s_1)P(s_1|s_3)P(we|s_3)P(s_3|s_0)P(k_0|s_0) \\ &= a_{2f}\ b_2(fish)\ a_{12}\ b_1(can)\ a_{31}\ b_3(we)a_{03} \\ &= 0.2 \times 0.75 \times 0.63 \times 0.1 \times 0.4 \times 0.6 \\ &= 0.00378 \end{aligned}$$

3

Apply the data to run the programme. Here's the result we've got: The black edge is the best path to



the vertex at time t, and the number on top of it is the possibility of the best path. The red-highlighted path is the overall path found on a backward pass.

4

The result from running Viterbi is $e \to 3 \to 4 \to 1 \to f$, which corresponds to personal pronoun $\to$ auxiliary verb $\to$ verb. The model arrives at the correct disambiguation (POS).

5

To estimate the emission probability matrix B from a labelled sample of text, count the occurrences of each word in each part of speech and normalise the counts to get probabilities. For example, given a labelled sample of text, we count how many times each word appears as a specific part of speech and divide by the total number of occurrences of that part of speech. Additionally, we can consider adding smoothing to improve the accuracy.

6

In (a), the probability of encountering the word "can" in the state sequence X_b is relatively low, at $b_1(can) = 0.1$. This observation aligns with the common usage of "can" as an auxiliary verb rather than a main verb. According to statistical principles such as Zipf's Laws, the frequency of a word tends to decrease exponentially as its rank increases, indicating a prevalence of high-frequency words and a long tail of low-frequency ones. Additionally, Heap's Law suggests that vocabulary size expands with text length, implying the continual emergence of new word types beyond the training corpus.Given these insights, it's likely that the emission probabilities of words exhibit sparsity, with many zero probabilities. This sparsity could potentially compromise the robustness of our model. To address this issue, we can implement smoothing techniques. Another idea, the independence of output assumption.

Viterbi with higher order HMMs

1. We need to keep N^N states for an N order HMM.
2. For an k order HMM with N states, M vocabulary and t length, The time complexity is $\Theta(tN^k)$, and the space complexity is also $\Theta(t(N^k + M^k))$. It indicates that although the asymptotic complexity of Viterbi is linear with the t (the length of the sequence). It is exponential with the order of HMM.