Hidden Markov Models for Part-of-Speech Tagging in Natural Language Processing

Tanika Jangam

Introduction

From the early days of hard coding grammars to modern day deep learning with large datasets of sentences, NLP has continuously evolved and has become more complex over time. One of the foundations of NLP that was created to provide a powerful framework to model human language are Hidden Markov Models (HMMs), which allow us to represent many different grammatical structures that occur in the English language by pairing hidden states with observable events. Despite the complexity of the human language, HMMs provide an elegant and simplified representation of the underlying syntax and grammars that a language is based on.

In this paper, we'll highlight one of the most important applications of HMM in NLP: Part of Speech (POS) tagging- the process of assigning parts of speech to each word in a sentence. Although this may seem trivial to do at first, POS tagging provides the foundation for more advanced tasks like parsing, information extraction and machine translation. Although modern deep learning technology is more complex, POS tagging using HMMs previously enabled both accuracy and efficiency in designing advanced NLP techniques.

The motivation behind this paper is to understand the linear algebra theory that the HMMs are based on and to show how POS tagging is imperative even today. Finally, I am to demonstrate how this technique creates the building blocks of modern NLP methods.

This paper is structured as follows: first in the "preliminaries" section, I'll review the linear algebra definitions that are needed to understand the logic behind HMMs. Then, we'll move onto the central application, where I'll show how exactly HMMs are used for POS tagging

by using fundamental equations, decoding algorithms, and concrete examples to fully understand how POS tagging is implemented through HMMs. Then, I'll conclude by summarizing the paper and explaining how HMMs are relevant in NLP today.

Preliminaries: Linear Algebra Concepts

Hidden Markov Models are usually written through matrix and vector operations, so the following definitions will help clarify the linear algebra concepts and computations that are referenced in the rest of this paper.

1. **State Probability Vector π:**

   First, in HMMs, we have probability distributions (the probability that a certain event occurs- or in this case, the probability that a certain state corresponds to a certain part of speech). These are usually represented as vectors (1-dimensional matrices).

   For example, let's say we have a set of hidden states $S = \{s_1, s_2, ..., s_N\}$, with each state corresponding to a part-of-speech tag. The probability of being in a certain state at a certain time can be expressed as a vector:

   $$\pi = [\pi_1, \pi_2, ..., \pi_N],$$

   where $\pi_i = P(state = s_i)$. This vector represents the initial state probabilities (probabilities of tags for the first word) or the distribution of states at any given step. For example, if $\pi = [\pi_N, \pi_V] = [.6, .4]$, that means without seeing any words, there is a 60% chance the first word will be a noun and a 40% chance the first word is a verb.

1. **Matrices and Transition/Emission Probabilities:**

   HMMs are based on two sets of probabilities:

   - **Transition Probabilities,** which show how states change over time. This is represented as a matrix as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix},$$

     Where $a_{ij} = P(s_j | s_i)$. For example, if we have:

$$A = \begin{bmatrix} P(N|N) & P(V|N) & P(D|N) \\ P(N|V) & P(V|V) & P(D|V) \\ P(N|D) & P(V|D) & P(D|D) \end{bmatrix} = \begin{bmatrix} 0.4 & 0.5 & 0.1 \\ 0.2 & 0.7 & 0.1 \\ 0.3 & 0.2 & 0.5 \end{bmatrix}.$$

     If the current tag is a Noun, by looking at the first row, there's a 40% chance the next word will also be a noun (P(N|N)=.4), 50% chance it'll be tagged as a verb, (P(V|N)=.5), and so on.

     When you multiply the state probability vector(π, as we defined above) by the transition matrix, you get the probability distributions of the next state .

   - **Emission Probabilities,** which show the probability of observing each word in a certain hidden state/tag. Let's say we have an alphabet of M unique words $V = \{w_1, w_2, ..., w_M\}$. The emission probabilities are given by:

$$B = \begin{bmatrix} b_1(w_1) & b_1(w_2) & \cdots & b_1(w_M) \\ b_2(w_1) & b_2(w_2) & \cdots & b_2(w_M) \\ \vdots & \vdots & \ddots & \vdots \\ b_N(w_1) & b_N(w_2) & \cdots & b_N(w_M) \end{bmatrix},$$

where $b_i(w_k) = P(word = w_k|s_i)$. Each row in the emission matrix probability distribution over the words given a state. For example, let's say we the states [N,V,D] (noun, verb, determiner, respectively) and our words are [the, cat, runs]:

$$B = \begin{bmatrix} P(\text{the}|N) & P(\text{cat}|N) & P(\text{runs}|N) \\ P(\text{the}|V) & P(\text{cat}|V) & P(\text{runs}|V) \\ P(\text{the}|D) & P(\text{cat}|D) & P(\text{runs}|D) \end{bmatrix} = \begin{bmatrix} 0.1 & 0.7 & 0.2 \\ 0.05 & 0.05 & 0.9 \\ 0.8 & 0.1 & 0.1 \end{bmatrix}.$$

If the current hidden state is a noun, the probability of observing the word "the" is 10%, "cat" is 70%, and runs is 20%.

Main Application: HMM-Based Part-of-Speech Tagging

1. Conceptual Framework of HMM POS Tagging:

In POS tagging, we start with a sentence $W = w_1, w_2, ..., w_T$ with T words. The goal is to assign a sequence of POS tags (our hidden states) $S = \{s_1, s_2, ..., s_T\}$ to these words. The hidden states are tags like Noun (person, place, object), Verb (action), Adjective (descriptive word), Determiner (word that precedes a noun, like "the" in "the cat"), etc. The words (w_i) are our observations, and we assume this Markov property:

$$P(s_t|s_1, s_2, ..., s_{t-1}) = P(s_t|s_{t-1}).$$

This means that based on the immediate previous state (s_{t-1}), we can predict the tagging process of the current state (s_t). For example, if our first word in the vector is "the" (a determiner), we

can predict that there is a high probability a noun will follow, such as “cat”, while there’s a low probability that the next state is a verb, as a phrase such as “the running” don’t fit the grammatical structure of the english language.

The emission probabilities link each state (POS tag) to the likelihood of a certain word occurring. For instance, a determiner state might highly favor words like “the,” “a,” and “this,” while a Noun state might be more likely to predict words like “cat,” “MATH401,” or “student.”

In reality, the prediction of the current state is much more complex and can be based off of numerous states before it, but simplifying our prediction to just the immediate previous state helps make linear algebra computations easier.

The overall joint probability of a certain sequence of states and observations are given by:

$$P(S, W) = \pi_{s_1} * b_{s_1}(w_1) * \prod_{t=2}^{T} a_{s_{t-1} s_t} b_{s_t}(w_t)$$

By counting how often a certain tag follows another tag (to estimate a_{ij}, the transmission probabilities), and how often a certain word appears in a given tag (estimates $b_i(w_k)$, the emission probabilities), allows us to produce estimates for the likelihood of a certain word being a certain state/POS tag.

2. The Training Process

Let’s say that we have a large database that provides sentences with words tagged with their POS labels. We can estimate:

- **Initial Probabilities (π_i):** The number of times each tag starts a sentence. If a tag s_i starts a sentence c_i times out of a total of C_{starts} sentences, then:

  $\pi_i = \frac{C_i}{C_{starts}}$ = number of times a tag starts a sentence / total number of sentences

- **Transition Probabilities** (a_{ij})**:** Counting the transitions between POS tags. If we see the pair (s_i, s_j) C_{ij} times, and the tag s_i occurs C_i times, then

$$a_{ij} = \frac{C_{ij}}{C_i}$$

- **Emission Probabilities:** Count how often a tag emits a particular word. If $C_i(w_k)$ is the count of word w_k with tag s_i and C_i is the total count of tag s_i:

$$b_i(w_k) = \frac{C_i(w_k)}{C_i}$$

Example Walkthrough:

Let's say we have two tags: Noun (N) and Verb (V), and have a vocabulary with three words: "dog," "barks," "runs."

Let's say our estimated probabilities are:

- **Initial Probabilities:** $\pi = [\pi_N, \pi_V] = [.6, .4]$
- **Transition Probabilities:** A=[P(N|N)=0.5P P(V|N)=0.5]

  [(N|V)=0.4 P(V|V)=0.6]

- **Emission Probabilities:** B=[P(dog|N)=0.5 P(barks|N)=0.3 P(runs|N)=0.2]

  [P(dog|V)=0.1 P(barks|V)=0.7 P(runs|V)=0.2]

We will use these probabilities for the following step.

3. Decoding: The Viterbi Algorithm

Once we've found the HMM parameters are estimated (π=the probability of being a certain state, A= transmission matrix, B = emission matrix), we face the main decoding problem: given a sentence of words W, find the most likely tag sequence S.

The Viterbi algorithm is a "dynamic programming" approach, which means it predicts the next tag in the sequence by backtracking through the chosen states to find the most likely sequence of tags.

Here is the algorithm, and I'll subsequently provide a continuation of our example:

- **Initialization** (t=1): $V_1(i) = \pi_i b_i(w_1)$
- **Recursion** (from t=2,3,..,T): $V_t(j) = (max_i V_{t-1}(i) a_{ij}) b_i(w_t)$
- **Termination**: $P' = max_j V_T(j)$

These series of steps/computations (based off of the linear algebra concepts defined from the preliminaries section) allows us to pick the best sequence of POS tags for any given sentence efficiently.

Going back to our example, consider that we want to tag the sentence "dog barks." We use the steps of the Viterbi decoding algorithm as detailed above.

- **Initialization** (t=1, so predicting POS tag for the first word, a.k.a. dog):
  - $V_1(N) = \pi_N b_N(dog) = .6 * .5 = .3$
  - $V_1(V) = \pi_V b_V(dog) = .4 * .1 = .04$

So after seeing "dog," we are more likely to be in state N (Noun) since $V_1(N) > V_1(V)$.

- **Recursion** for the second word "barks": For state N at time t=2:

- $V_2(N) = max\{V_1(N)a_{NN}, V_1(V)a_{VN}\}b_N(barks) = max\{.3 *.5, .04 *.4\} *.3 = .045$

- For state V at t=2 (second word- barks)
  - $V_2(V) = max\{V_1(N)a_{NV}, V_1(V)a_{VV}\}b_V(barks) = max\{.3 *.5, .04 *.6\} *.7 = .105$

At the end of this computation, we see that is $V_2(V)$=.105 is higher than, $V_2(N) = .045$ predicting that the best final state for the word "bark" is Verb,. By looking back, we find that at the first word, we were in a Noun state, and at the second word, we moved to a Verb state. Thus, "dog" is tagged as Noun and "barks" is tagged as Verb, providing the POS sequence (N, V).

Although this example walkthrough is simplified, it demonstrates how HMM parameters (the probability distribution of states, the transition probabilities, and the emission probabilities) allows us to assign the most likely POS tags to each word in a sentence.

5. Complexities and Limitations

HMM based tagging is computationally efficient because it's based on the number of POS tags (usually in the tens), and the length of the sentence. Since both the number of states/POS tags (N) and average sentence length (T) isn't large for most usual NLP tasks, HMM tagging happens relatively efficiently.

Additionally, although HMM's were well adopted before neural networks were used, they have limitations. Because we've assumed that the probability of a given state/tag depends only on the state immediately preceding it, the HMM won't be able to predict words that are more nuanced or are dependent on the context of the sentence. For example, consider the sentence:

"The teacher who supervises many students **is** conducting a seminar today."

In this example, the subject, “The teacher” appears in the beginning, and the main verb “is” occurs later on in the sentence. However, there is a phrase in between- “who supervises many students”. Despite the word “students” being plural, the main verb still needs to agree with “the teacher”, which is singular. An HMM would look at the immediate previous word “students” as a sign for a plural verb, and can incorrectly give a plural verb “are” rather than “is”. In order for the HMM to correct this, the model needs to consider more not just the immediate previous word, but the broader context of the sentence.

Conclusion

By representing language as a sequence of hidden states (POS tags) that emit observable words, HMMs allow us to represent the grammatical and syntactical structure of human language into a probability model based on linear algebra concepts— vectors for state probability distributions (π), the transitions matrix (A) and emissions matrix (B). From these concepts, we can do NLP related tasks such as finding the most likely POS tag sequence through the Viterbi Algorithm.

Even though modern neural networks for NLP surpass HMMs in accuracy and efficiency, understanding HMMs for POS tagging is important for understanding the foundation for more complex NLP techniques. HMMs show how ideas as intricate as the structure of human language can be predicted and calculated through linear algebra concepts that set the trajectory for advanced neural techniques used today.

Sources

1. Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37(6), 1554–1563.
2. Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing* (2nd ed.). Prentice Hall.
3. Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.
4. Cutting, D., Kupiec, J., Pedersen, J., & Sibun, P. (1992). A practical part-of-speech tagger. *Proceedings of the Third Conference on Applied Natural Language Processing*, 133–140.
5. Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.
6. Brants, T. (2000). TnT–a statistical part-of-speech tagger. *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP)*, 224–231.
7. Goldwater, S., Griffiths, T. L., & Johnson, M. (2006). Interpolating between types and tokens by estimating power-law generators. *Advances in Neural Information Processing Systems (NIPS)*.
8. Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.