DSE 308: PROJECT REPORT

ANKUR GURIA, 17039
ankur17@iiserb.ac.in

OBJECTIVE:

Implement a Hidden Markov Model-based part of speech tagger on Penn Treebank Corpus and analyze the error and success of the tagger.

MOTIVATION:

Back in primary school, we've discovered the distinctions between the different sections of speech marks, such as nouns, verbs, adjectives, and adverbs. Associating each word in a sentence with a proper POS (part of speech) is known as POS or POS annotation. POS tags are also referred to as word classes, morphological classes, or lexical tags. Back in time, the POS annotation was performed manually by human annotators, but being such a laborious job today we have automated tools that are capable of marking any word with an appropriate POS tag in a background. Nowadays, manual annotation is usually used to annotate a limited corpus to be used as training data for the creation of a new automated POS tagger. Manually annotating existing multi-billion-word corpora is unrealistic and automatic labeling is used instead. POS tags provide a lot of information about a term and its neighbors. Their implementations can be used in a number of functions, such as information processing, parsing, Text to Speech (TTS) applications, content extraction, linguistic study for companies. They are often used as an intermediate stage for higher-level NLP activities such as parsing, semantics analysis, translation, etc. Hidden Markov models are known for their applications for improved learning and time pattern detection, such as voice, handwriting, gesture recognition, follow-up musical scores, partial discharges, and bioinformatics. Their implementations can be used in a number of functions, such as information processing, parsing, Text to Speech (TTS) applications, content extraction, linguistic study for corpora They are also used as intermediate measures for higher-level NLP tasks such as parsing.

THEORY:

The Hidden Markov Models (HMM) is a statistical model for modeling generative sequences characterized by an underlying process generating an observable sequence. HMMs have various applications such as speech recognition, signal processing, and some low-level NLP tasks such as POS tagging, phrase chunking, and extracting information from documents. HMMs are also used in converting speech to text in speech recognition. HMMs are based on Markov chains. A Markov chain is a model that describes a sequence of potential events in which the probability of an event is dependent only on the state which is attained in the previous event. The Markov model is based on a Markov assumption in predicting the probability of a sequence. If state variables are defined as $q_1, q_2, \dots, q_i$ a Markov assumption is defined as

$$\text{Markov Assumption: } P(q_i = a | q_i \dots q_{i-1}) = P(q_i = a | q_{i-1})$$

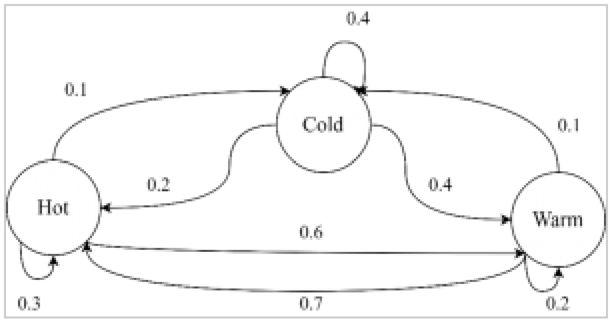

Figure 1. A Markov chain with states and transitions

Figure 1 shows an example of a Markov chain for assigning a probability to a sequence of weather events. The states are represented by nodes in the graph while edges represent the transition between states with probabilities. A first-order HMM is based on two assumptions. One of them is Markov assumption, that is the probability of a state depends only on the previous state as described earlier, the other is the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or observations

$$\text{Output Independence: } P(o_i | q_1 \dots q_i, \dots q_t, o_1 \dots o_i, \dots o_T)$$

An HMM consists of two components, the A and the B probabilities. The A matrix contains the tag transition probabilities $P(t_i \vee t_{i-1})$ and B the emission probabilities $P(w_i \vee t_i)$ where w denotes the word and t denotes the tag. The transition probability, given a tag, how often is this tag is followed by the second tag in the corpus is calculated as:

$$P(t_i | t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

The emission probability, given a tag, how likely it will be associated with a word is given by:

$$P(w_i | t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

Figure 2 shows an example of the HMM model in POS tagging. For a given sequence of three words, “word1”, “word2”, and “word3”, the HMM model tries to decode their correct POS tag from “N”, “M”, and “V”. The A transition probabilities of a state to move from one state to another and B emission probabilities that how likely a word is either N, M, or V in the given example.

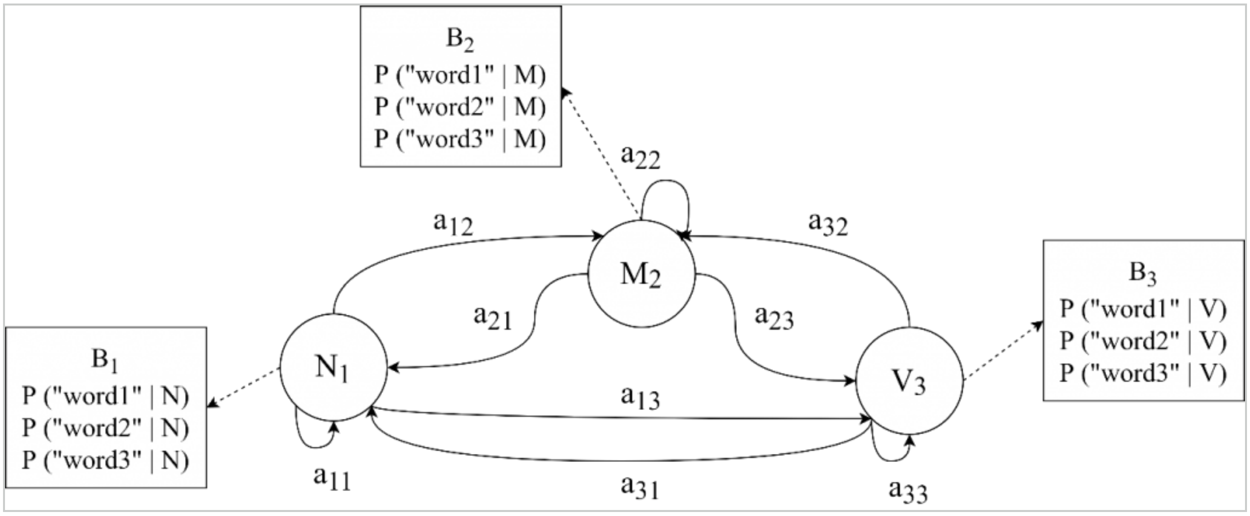

Figure 2. A Hidden Markov Model with A transition and B emission probabilities.

HMM TAGGER

The process of determining hidden states to their corresponding sequence is known as decoding. More formally, given A, B probability matrices and a sequence of observations $O = o_1 \ldots o_2, \ldots o_T$, the goal of an HMM tagger is to find a sequence of states $Q = q_1 \ldots q_2, \ldots q_T$. For POS tagging the task is to find a tag sequence t_1^n that maximizes the probability of a sequence of observations of n words w_1^n:

$$t_1^n = \max_{t_1^n} P(t_1^n | w_1^n) \approx \max_{t_1^n} \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1})$$

The Viterbi Algorithm

The decoding algorithm for the HMM model is the Viterbi Algorithm. The algorithm works as setting up a probability matrix with all observations o_t in a single column and one row for each state q_i. A cell in the matrix $v_t(j)$ represents the probability of being in state j after first t observations and passing through the highest probability sequence given A and B probability matrices. Each cell value is computed by the following equation:

$$v_t(j) = \max_{q_1 \cdots q_{t-1}} P(q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j | (A, B))$$

Figure 3 shows an example of a Viterbi matrix with states (POS tags) and a sequence of words. A greyed state represents zero probability of word sequence from the B matrix of emission probabilities. Highlighted arrows show word sequence with correct tags having the highest probabilities through the hidden states.

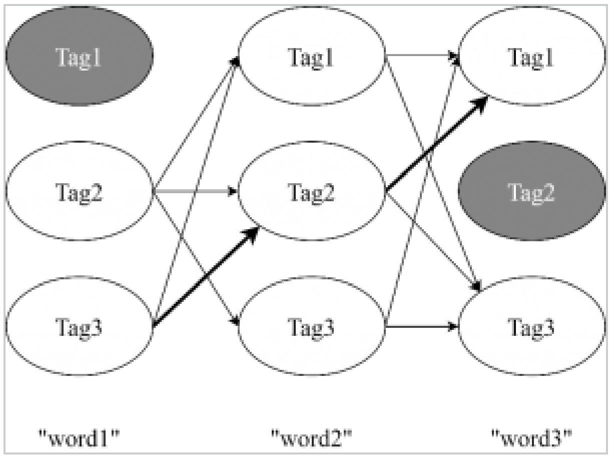

Figure 3. Viterbi matrix with possible tags for each word.

The Viterbi algorithm works recursively to compute each cell value. For a given state q_j at time t, the Viterbi probability at time t, $v_t(j)$ is calculated as:

$$v_t(j) = \max_{i=1}^{n} v_{t-1}(i) a_{ij} b_j(o_t)$$

The components used to multiply to get the Viterbi probability are the previous Viterbi path probability from the previous time $v_{t-1}(i)$, a_{ij} the transition probability from the previous state q_i to the current state q_j and $b_j(o_t)$ the state observation likelihood of the observation symbol o_t given the current state t.

DATA SET USED:

There are many different English-language tagsets used in the marking of several corpora. We have used the Penn Treebank tagset is such an essential tagset. This tagset also distinguishes unique character tags and punctuation tags separately from other POS tags. The tagset from Nivre et al., which is called the Universal POS tagset, is used to tag words from different languages. This tagset is part of the Universal Dependencies project and includes 16 tags and numerous functions to support multiple languages.

EXPERIMENTS:

- We have first tried the original Viterbi Algorithm.
- Then we have tried to optimize the Viterbi Algorithm by mixing it with rule-based tagging to handle unknown words i.e. words that are not present in the training corpus. Rule-based POS tag models add a series of handwritten rules and use

contextual information to assign POS tags to terms. These laws are also referred to as context frame rules. One such law may be If an ambiguous/unknown word ends with the suffix 'ing' and is preceded by a verb, label it as a verb."

OBSERVATIONS:

The optimized Viterbi algorithm had an accuracy of 96.8% and performed better than the original Viterbi algorithm which had an accuracy of 91.78%. Adding the rule-based tagger did improve the performance of the HMM tagger. Comparing the performance of the two models separately on a test sentence gave the following results:

Test sentence: “Will can see Marry”

Viterbi with rule-based tagging results: [('Will', 'NOUN'), ('can', 'VERB'), ('see', 'VERB'), ('Marry', 'NOUN')]
Viterbi algorithm results: [('Will', '.'), ('can', 'VERB'), ('see', 'VERB'), ('Marry', '.')]

CONCLUSION:

We have discussed POS tagging, a text analysis technique used to extract the association between adjacent words in a sentence. POS tagging addresses ambiguities for computers that understand natural language. A significant number of errors emerge from the natural language understanding (NLU) module in conversational systems. POS tagging is one method used to eliminate these mistakes in conversation schemes. Such devices in safety-critical sectors such as healthcare can have a safety-related effect due to errors in understanding natural language.