POS Tagging Using Viterbi Algorithm

Sunish Sheth
Natural Language Processing
The University of Texas at Dallas
Richardson, TX 75080 USA
sss140830@utdallas.edu

Abstract
POS Tagging is one of the important part of Natural Language Processing. POS tagging is basically identifying the Parts of Speech of the words in a given sentence. This part of speech is necessary to know the meaning or the sense of the sentence, also termed as disambiguate the sentence. Many applications and algorithms need the words in the sentences to be POS tagged as their basic building block before further processing. So, one may see the importance of POS tagging. This paper discusses the process of POS tagging using Viterbi algorithm, which is an extension on Hidden Markov Model (HMM). Thus reducing the complexity. Brown corpus has been used for Training and Testing the Model being created using Viterbi Algorithm.

1 Introduction

In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context—i.e., its relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.
Earlier, I was researching on the topic of Question Answering, I found out that the basic building block of this process was to have the entire corpus being POS Tagged to help understand the meaning of the sentences in the corpus. Also the question being asked has to be tagged to understand its sense and find a suitable answer. There are many libraries which help get this done, for example, NLTK. So for POS Tagging to be successful, a huge amount of training is required. A good training data helps us create a better model.

The Brown University Standard Corpus of Present-Day American English (or just Brown Corpus) was compiled in the 1960s by Henry Kučera and W. Nelson Francis at Brown University, Providence, Rhode Island as a general corpus (text collection) in the field of corpus linguistics. The corpus originally (1961) contained 1,014,312 words sampled from 15 text categories, made up of 500 samples from randomly chosen publications. Each sample is 2,000 or more words (ending at the first sentence-end after 2,000 words, so that the corpus contains only complete sentences). The tagged Brown Corpus used a selection of about 80 parts of speech, as well as special indicators for compound forms, contractions, foreign words and a few other phenomena, and formed the basis for many later corpora such as the Lancaster-Oslo-Bergen Corpus.
Note that there are some words in the tagged Brown corpus contain combined tags. For instance, the word "wanna" is tagged VB+TO, since it is a contracted form of the two words, want/VB and to/TO. Also some tags might be negated, for instance "aren't" would be tagged "BER*", where * signifies the negation. Additionally, tags may have hyphenations: The tag -HL is hyphenated to the regular tags of words in headlines. The tag -TL is hyphenated to the regular tags of words in titles. The hyphenation -NC signifies an *emphasized* word. Sometimes the tag has a FW- prefix which means foreign word [1].

2 Related Work

Over the period, there have been many research on POS Tagging and many ways have been devised to improve the accuracy and avoiding overfitting of the training data.
In the mid 1980s, researchers in Europe began to use hidden Markov models (HMMs) to disambiguate parts of speech, when working to tag the Lancaster-Oslo-Bergen Corpus of British

English. HMMs involve counting cases (such as from the Brown Corpus), and making a table of the probabilities of certain sequences. For example, once you've seen an article such as 'the', perhaps the next word is a noun 40% of the time, an adjective 40%, and a number 20%. Knowing this, a program can decide that "can" in "the can" is far more likely to be a noun than a verb or a modal. The same method can of course be used to benefit from knowledge about following words.
More advanced ("higher order") HMMs learn the probabilities not only of pairs, but triples or even larger sequences. So, for example, if you've just seen a noun followed by a verb, the next item may be very likely a preposition, article, or noun, but much less likely another verb.
In 1987, Steven DeRose[2] and Ken Church[3] independently developed dynamic programming algorithms to solve the same problem in vastly less time. Their methods were similar to the Viterbi algorithm known for some time in other fields. DeRose used a table of pairs, while Church used a table of triples and a method of estimating the values for triples that were rare or nonexistent in the Brown Corpus (actual measurement of triple probabilities would require a much larger corpus). Both methods achieved accuracy over 95%. DeRose's 1990 dissertation at Brown University included analyses of the specific error types, probabilities, and other related data, and replicated his work for Greek, where it proved similarly effective.
The methods already discussed involve working from a pre-existing corpus to learn tag probabilities. It is, however, also possible to bootstrap using "unsupervised" tagging. Unsupervised tagging techniques use an untagged corpus for their training data and produce the tag set by induction. That is, they observe patterns in word use, and derive part-of-speech categories themselves. For example, statistics readily reveal that "the", "a", and "an" occur in similar contexts, while "eat" occurs in very different ones. With sufficient iteration, similarity classes of words emerge that are remarkably similar to those human linguists would expect; and the differences themselves sometimes suggest valuable new insights. These two categories can be further subdivided into rule-based, stochastic, and neural approaches.
Some current major algorithms for part-of-speech tagging include the Viterbi algorithm, Brill Tagger, Constraint Grammar, and the Baum-Welch algorithm (also known as the forward-backward algorithm). Hidden Markov model and visible Markov model taggers can both be implemented using the Viterbi algorithm. The rule-based Brill tagger is unusual in that it learns a set of rule patterns, and then applies those patterns rather than optimizing a statistical quantity. Unlike the Brill tagger where the rules are ordered sequentially, the POS and morphological tagging toolkit RDRPOSTagger stores rules in the form of a Ripple Down Rules tree.
Many machine learning methods have also been applied to the problem of POS tagging. Methods such as SVM, Maximum entropy classifier, Perceptron, and Nearest-neighbor have all been tried, and most can achieve accuracy above 95%[1].

3. Approach

POS tagging of assigning a part-of-speech to each word in a sentence. (automatically or manually). It is useful in resolving lexical ambiguity. POS tagging is useful in Informational Retrieval (IR), Text to Speech, Question Answering, Word sense disambiguation and many other places where it is used as a preprocessing step of parsing. POS tagging is hard mainly because of "**Ambiguity"**. The major ambiguity found in tagging a word to its part of speech is "Noun" gets tagged to Verb most of the times and vice-versa[1].
Example:
Input → And now for something completely different
Output → [('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely', 'RB'), ('different', 'JJ')][2].

3.1 Hidden Markov Model

A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (*hidden*) states. An HMM can be presented as the simplest dynamic Bayesian network.
In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but the output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states. The adjective 'hidden' refers to the state sequence through which the model passes, not to the parameters of

the model; the model is still referred to as a 'hidden' Markov model even if these parameters are known exactly.

A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other. Recently, hidden Markov models have been generalized to pairwise Markov models and triplet Markov models which allow consideration of more complex data structures and the modelling of nonstationary data.

Hidden Markov Model consists of three components:

- Ps(Si): Probability of system starting in state Si
- Ps(Sj|Si): Probability of the system transitioning from state Si to Sj.
- Pe(Xj|Si): Probability of the system emitting output Xj in state Si[3].

3.1.2 Architecture

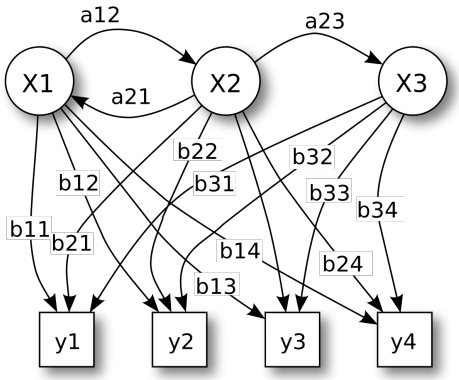

Figure 1: Probabilistic parameters of a hidden Markov model.

The diagram below shows the general architecture of an instantiated HMM. Each oval shape represents a random variable that can adopt any of a number of values. The random variable $x(t)$ is the hidden state at time t (with the model from the above diagram, $x(t) \in \{x_1, x_2, x_3\}$). The random variable $y(t)$ is the observation at time t (with $y(t) \in \{y_1, y_2, y_3, y_4\}$). The arrows in the diagram (often called a trellis diagram) denote conditional dependencies.

From the diagram, it is clear that the conditional probability distribution of the hidden variable $x(t)$ at time t, given the values of the hidden variable x at all times, depends *only* on the value of the hidden variable $x(t-1)$; the values at time $t-2$ and before have no influence. This is called the Markov property. Similarly, the value of the observed variable $y(t)$ only depends on the value of the hidden variable $x(t)$ (both at time t). [3]

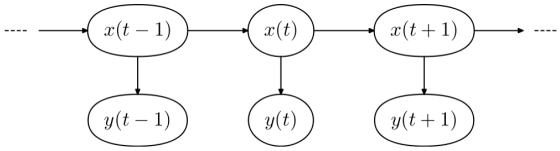

Figure 2: General architecture of an instantiated HMM

3.2 Viterbi Algorithm

The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states – called the Viterbi path – that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models.It is now also commonly used in speech recognition, speech synthesis, diarization, keyword spotting, computational linguistics, and bioinformatics. For example, in speech-to-text (speech recognition), the acoustic signal is treated as the observed sequence of events, and a string of text is considered to be the "hidden cause" of the acoustic signal. The Viterbi algorithm finds the most likely string of text given the acoustic signal [4].

3.2.1 Algorithm

Suppose we are given a hidden Markov model (HMM) with state space S, initial probabilities π_i of being in state i and transition probabilities $a_{i,j}$ of transitioning from state i to state j. Say we observe outputs $y_1, \ldots, y_T$. The most likely state sequence $x_1, \ldots, x_T$ that produces the observations is given by the recurrence relations:[7]

$$V_{1,k} = \mathrm{P}(y_1 \mid k) \cdot \pi_k$$
$$V_{t,k} = \max_{x \in S} \left(\mathrm{P}(y_t \mid k) \cdot a_{x,k} \cdot V_{t-1,x} \right)$$

Here $V_{t,k}$ is the probability of the most probable state sequence $\mathrm{P}(x_1, \ldots, x_T, y_1, \ldots, y_T)$ responsible for the first t observations that have k as its final state. The Viterbi path can be retrieved by saving back pointers that remember which state x was used in the second equation. Let $\mathrm{Ptr}(k, t)$ be

the function that returns the value of $x$used to compute $V_{t,k}$if $t > 1$, or $k$if $t = 1$.
Then:

$$x_T = \arg\max_{x \in S}(V_{T,x})$$
$$x_{t-1} = \mathrm{Ptr}(x_t, t)$$

Here we're using the standard definition of arg max.
The complexity of this algorithm is $O(T \times |S|^2)$.[4]

3.2.2 Concrete Example

Consider a village where all villagers are either healthy or have a fever and only the village doctor can determine whether each has a fever. The doctor diagnoses fever by asking patients how they feel. The villagers may only answer that they feel normal, dizzy, or cold.
The doctor believes that the health condition of his patients operate as a discrete Markov chain. There are two states, "Healthy" and "Fever", but the doctor cannot observe them directly, they are hidden from him. On each day, there is a certain chance that the patient will tell the doctor he/she is "normal", "cold", or "dizzy", depending on her health condition.

```
states = ('Healthy', 'Fever')
observations = ('normal', 'cold', 'dizzy')
start_probability = {'Healthy': 0.6, 'Fever': 0.4}
transition_probability = {
   'Healthy' : {'Healthy': 0.7, 'Fever': 0.3},
   'Fever' : {'Healthy': 0.4, 'Fever': 0.6}
   }
emission_probability = {
   'Healthy' : {'normal': 0.5, 'cold': 0.4, 'dizzy': 0.1},
   'Fever' : {'normal': 0.1, 'cold': 0.3, 'dizzy': 0.6}
   }
```

Figure 3: The observations (normal, cold, dizzy) along with a hidden state (healthy, fever) form a hidden Markov model (HMM)

In this piece of code, start_probability represents the doctor's belief about which state the HMM is in when the patient first visits (all he knows is that the patient tends to be healthy). The particular probability distribution used here is not the equilibrium one, which is (given the transition probabilities) approximately {'Healthy': 0.57, 'Fever': 0.43}. The transition_probability represents the change of the health condition in the underlying Markov chain. In this example, there is only a 30% chance that tomorrow the patient will have a fever if he is healthy today. The emission_probability represents how likely the patient is to feel on each day. If he is healthy, there is a 50% chance that he feels normal; if he has a fever, there is a 60% chance that he feels dizzy.

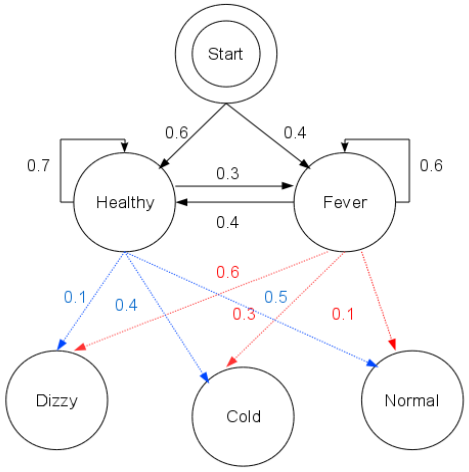

Figure 4: Graphical representation of the given HMM

The patient visits three days in a row and the doctor discovers that on the first day she feels normal, on the second day she feels cold, on the third day she feels dizzy. The doctor has a question: what is the most likely sequence of health conditions of the patient that would explain these observations? This is answered by the Viterbi algorithm.

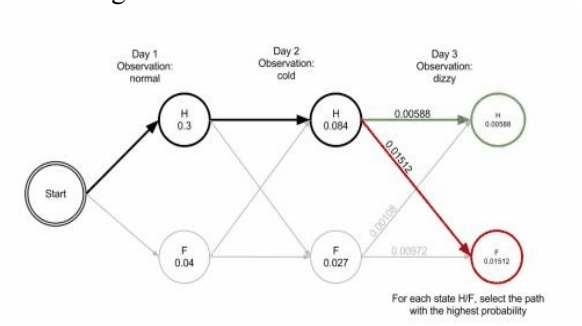

Figure 5: trellis diagram for the Viterbi algorithm. After Day 3, the most likely path is ['Healthy', 'Healthy', 'Fever']

This reveals that the observations ['normal', 'cold', 'dizzy'] were most likely generated by states ['Healthy', 'Healthy', 'Fever']. In other words, given the observed activities, the patient was most likely to have been healthy both on the first day when she felt normal as well as on the second day when she felt cold, and then she contracted a fever the third day[4].

3.3 Implementation

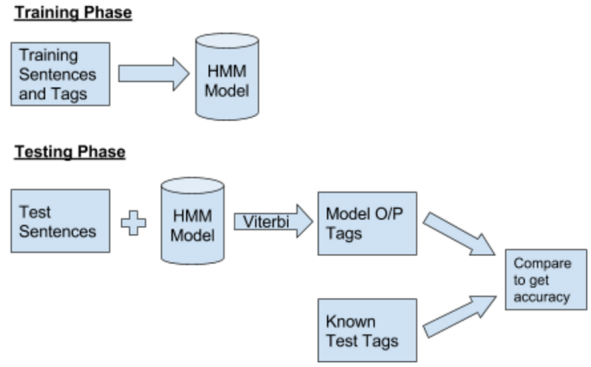

Figure 6 : Flow Diagram for the implementation of POS Tagging process

Brown corpus has been used to train the model. Sentences and tags from the corpus has been used to create State Transition Matrix and Observation Matrix. The following three matrices have been obtained from training data.

- *Ps(Ti)*: Probability of system starting in tag Ti
- *Ps(Tj|Ti)*: Probability of the system transitioning from tag Ti to Tj.
- *Pe(Wj|Ti)*: Probability of the system emitting output Wj in tag Ti.[7]

Now in the testing phase, we use the above created model with the test sentences and apply Viterbi algorithm to get the O/P tag set.

Let W1, W2….,Wn be the sequence of words from test set and T1, T2,....., Tm be the sequence of tags that we can use to classify each word. These tags are the states in Viterbi Algorithm and the words are the observations.

Let $V_t(j)$ is the probability that HMM is in state j after seeing the first t observations. Therefore, $V_{t-1}(j)$ is the previous path probability of the previous time step. a_{ij} is the transition probability from previous state q_i to current state q_j. Now $b_j(O_t)$ is the state observation likelihood of the observation symbol O_t given the current state j.

Now we can use the following steps to get our O/P tag set.

1. Initialization

$$v_1(j) = a_{0j} b_j(o_1) 1 \le j \le N$$

$$bt_1(j) = 0$$

2. Recursion

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

$$bt_t(j) = \operatorname{argmax}_{i=1}^{N} v_{t-1}(i)\, a_{ij} b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$

3. Termination

The best score: $P* = v_t(q_F) = \max_{i=1}^{N} v_T(i) * a_{i,F}$

The start of backtrace: $q_T* = b_{tT}(q_F) = \operatorname{argmax}_{i=1}^{N} v_T(i) * a_{i,F}$

Figure 7 : Steps from Viterbi Algorithm

I compared the model o/p tag set with the available tag set of the sentence to calculate the accuracy of the Viterbi model [5].

4. Experiments

To create the model, I have made use of the brown corpus. The corpus is divided into training and testing data. The corpus consists of 28618 sentences with 579513 tagged words which is used for training. Testing data has 2390 sentences and 36394 tagged words. We will use this to find tags using our model and Viterbi algorithm and compare it against labelled tags given from the corpus. Thus help us find the accuracy of the model.

The following is result from the model I created.

Total Tags	Correct Tags	InCorrect Tags	Accuracy Rate
36394.0	33217.0	3177.0	91.270%

Table 1 : Experimental Results

From the experiment it was found that the tag MOD (modal auxiliary) has incorrectly classified to some other tag. One another tag which is maximum incorrectly classified is NP (Proper Noun or part of name phrase) being tagged as N (Noun).

The system is not that strong enough as its not trained with huge data set so if we provide some unknown word which is not in the training it is not able to predict the tag properly.

5. Conclusion

From the above experiment various conclusions can be stipulated. The system does not handle emotional tagging. It cannot detect sentences that are negative, positive or imperative. The system is less accurate to the words that are unseen. The training data must be much bigger than the current brown corpus to see all the unseen scenarios and make a better model. Currently, the system behaves well when the data is known and thus, performs really good with test data.

Currently the model developed is done using simple Viterbi algorithms. Smoothing techniques such as Laplace smoothing can be applied to make the model much better [6]. We can also use multiple models and then use the majority tag to increase the accuracy.

References

[1]. Wikipedia page for POS tagging: https://en.wikipedia.org/wiki/Part-of-speech_tagging

[2]. http://www.nltk.org/

[3]. Wikipedia page for Hidden Markov Model:https://en.wikipedia.org/wiki/Hidden_Markov_model

[4]. Wikipedia page for Viterbi Algorithm: https://en.wikipedia.org/wiki/Viterbi_algorithm

[5]. Class Lecture notes by Dr. Dan I. Moldovan on Hidden Markov Model: http://www.hlt.utdallas.edu/~moldovan/CS6320.16S/Lecture6%20HMM.pdf

[6]. Class Lecture notes by Dr. Dan I. Moldovan on POS Tagging: http://www.hlt.utdallas.edu/~moldovan/CS6320.16S/Lecture5%20POS.pdf

[7]. Project report from Sander Parawira, a student of Stanford University on Hidden Markov Model and Viterbi Algorithm: http://nlp.stanford.edu/courses/cs224n/2010/reports/parawira.pdf