TR-IT-0063

A Markov Model Part of Speech Tagger

Jared C Saia
Advanced Telecommunications Research Labs
2-2 Hikaridai, Seika-cho Soraku-gun, Kyoto 619-02 Japan
saia@cs.stanford.edu

August 2, 1994

Abstract

This paper describes a statistical part of speech tagger. First, equations for part of speech tagging are derived from the source-channel model. Then an implementation is described and finally there is some discussion of results and possible improvements.

1 Introduction

Recently the use of stochastic models has become increasingly popular in the language modeling community. These models allow for more robust and flexible analysis than do traditional symbolic methods. The basic idea of these statistical methods is to use a set of training data to construct a probability model. The probability model is then used to calculate the probabilities of various analyses and the one with the highest probability can be considered the "correct" analysis. In this paper a method for stochastically finding parts of speech of words in a sentence is described. This method treats human language as a Markov process. That is, the assumption is made that the probability of any given word is dependent only on the last n words. In addition it is assumed that the probability of a part of speech can be determined by looking only at a window of the last m parts of speech and words. This assumption is made in order to create a reasonably accurate probability model from a finite amount of data. There are other statistical methods such as decision trees which can calculate probabilities without making explicit Markov assumptions(see Black, E. (1992) Decision Tree Models Applied to the Labeling of Text with Parts-of-Speech,Speech and Natural Language Workshop Proceedings, 117-121.)

The task of finding parts of speech when given a stream of words can be best visualized using the noisy channel paradigm. We imagine that someone has sent a message to us which has been corrupted by transmission and that we want to recover the original message by considering what we have heard. In our case the message originally sent to us is a stream of parts of speech. This message has been corrupted and has reached us as a stream of words. We want to recover the stream of tags by considering the stream of words.

We can use probability theory to determine the most likely message said given what we have heard. For example, in the part of speech case, the probability that a given message was produced and that we heard it is P(W,T) where W is the stream of words and T the stream of tags. This probability for any W and T is equivalent to the probability that the stream of tags was said at the source end of the channel times the probability that the stream of tags was received given that the words were said i.e.

$$P(W,T) = P(T)P(W \mid T) \tag{1}$$

This equation with varying degrees of Markov simplification is the basis for all the models described in this paper.

At this point, using equation 1, we can formally define the tagging problem. We have a fixed vocabulary of possible words and tags which the random variables w and t can take on. We are given a sentence W which is defined as a sequence of N values of the variable w. Given this sequence we attempt to find the sequence of tags T which maximizes equation (1). That is we try to find:

$$argmax_T P(W,T)$$

In the simplest case we make the assumption that the probability of each tag is independent of the past words and tags and is only dependent on the current word. That is to say that the probability of a tag in a sentence is simply proportional to the frequency with which that tag appears with the current word.

To get the most likely tag sequence in this case we have

$$P(W,T) = P(W)P(T \mid W)$$

using Bayes rule on equation (1). Since we are maximizing over the sequence of tags, we can eliminate $P(W)$ in the above equation since it remains constant for any tags we may consider. So using the Markov assumption we have

$$argmax_{t_{1,N}} \prod_{i=1}^{N} P(t_i \mid w_i) \tag{2}$$

This is the unigram case.

In this paper we also consider a more complex Markov model. We assume the probability of the tag depends on the last two tags but that the current word is still dependent only on the current tag. For the trigram case we can derive an equation from the right side of equation (1) making the following assumptions:

$$P(T) = \prod_{i=1}^{N} P(t_i \mid t_{i-1}, t_{i-2})$$

and

$$P(W \mid T) = \prod_{i=1}^{N} P(w_i \mid t_i)$$

which give us :

$$argmax_{t_{1,N}} \prod_{i=1}^{N} P(t_i \mid t_{i-1}, t_{i-2}) P(w_i \mid t_i) \tag{3}$$

This is the trigram case.
In order to make this equation work for t_1 and t_2 we define dummy presentence tags t_0 and t_{-1} to condition on.

2 Implementation

When implementing a model which makes use of the above equations one must deal with two main problems. The first problem is to find in a reasonable amount of time the one tag sequence out of all possible such sequences which has the highest probability. The second problem concerns creating accurate probability models given sparse training data. The basic details of implementing Markov Model part of speech taggers have been covered before in past literature but since the approach to solving these two problems often differentiates implementations, the solutions used in this paper's tagger are discussed below.

The computational time taken to find the most probable tag sequence is not large in the unigram case; it takes only O(N) time where N is the number of tags in the tag alphabet. To calculate for the trigram case, we can use the Viterbi algorithm, a dynamic programming method which will find the solution in $O(N^x)$ time for a x-gram grammar. For the trigram case it still takes $O(N^3)$ but this can be further shortened by pruning during the search without much decrease in accuracy. In this implementation we kept for each step of the Viterbi algorithm only the top ten highest probability sequences. This resulted in significant speed up and resulted less than 1% reduction in accuracy.

The second problem of creating useful models from sparse data is not so easily solved. When we use training data to construct probability models, there are often events which do not occur in the training data but may occur in data that we later test on. For example, we don't always have a good estimate for $P(w_i \mid w_{i-1}, w_{i-2})$ since not all possible trigrams appear frequently, but we must come up with some useful default probability for this case.

We are faced with two conflicting requirements. First of all we don't want to give 0 probability to unseen events because there is often some small probability they may occur. Secondly, we only have a limited amount of probability mass to distribute over possible events. The probability of all events must sum to 1 in order to maintain the model invariant.

There are many approaches to these conflicting requirements. The one used here is linear interpolation. This method involves deriving a smoothed model from the weighted sum of other probability models. For example, in the trigram case we have:

$$\tilde{P}(t_i \mid t_{i-1}, t_{i-2}) = \lambda_1 P(t_i \mid t_{i-1}, t_{i-2}) + \lambda_2 P(t_i \mid t_{i-1}) + \lambda_3 P(t_i) + \lambda_4 \frac{1}{numtags}$$

To smooth $P(t \mid w)$ for unknown words we have:

$$\tilde{P}(t \mid w) = \lambda_1 P(t \mid w) + \lambda_2 P(t) + \lambda_3 \frac{1}{numtags}$$

($P(w \mid t)$ is smoothed by rewriting it using Bayes Theorem to incorporate $P(t \mid w)$)

As long as the λs in the above models sum to 1 and as long as each term after the λ is a probability model, the resulting smoothed function is a probability model. In addition, there are mathematical techniques which can be used to find useful λ values.

The technique used in this paper to find useful λs is Hidden Markov Models. In this case, we set aside a certain amount of the training data as smoothing data. We then treat this data as output from a HMM. The "hidden" states in the HMM are thus the probability models. Using the Forward-Backward algorithm we can get good estimates of transition probabilities between the "hidden" probability models. If we set up the states of the HMM correctly then the transition probabilities are exactly the same as our λ values. Given initial λ values, the Forward-Backward algorithm is guaranteed to give λs which will increase the likelihood of the smoothing data according to the current models. We continue this successive improvement until the likelihood of the smoothing data no longer increases.

3 Results and Discussion

The tagger implemented in this paper was tested on the Penn Treebank. This is a corpus of hand tagged sentences taken from various sources. There are less than 50 unique tags used in this corpus. First an attempt was made to replicate the 96% accuracy for trigram tagging that is the current state of the art(see Weischedel (1992) "Part of Speech Tagging", Association of Computational Linguistics).

This degree of accuracy was achieved under the same conditions as Weischedel when using equivalent training data(about 990K words). There are some differences in Weischedel's implementation and this one however. For example, Weischedel uses a backing off smoothing technique for dealing with the sparse data problem. Surprisingly, these differences do not seem to have much effect on accuracy.

It is worth remarking however that for the results published by Weischedel et. alia, they first randomize the sentences in the Penn Treebank, then split the Treebank up into training, smoothing and testing data. In addition, unknown(previously unseen) words are ignored when reporting results. In the tagger implemented in this paper, testing both on randomized and linear data revealed that in the randomized case the accuracy was about 4% higher.

This difference can be attributed to the fact that the style of sentences from the same document are likely to be similar and that if the data is randomized, certain trigram patterns and word tag pairs are more likely to be contained in both the training and testing data. In the linear case, it is possible to come across a style of writing which is poorly represented in the training data.

When both considering unknown words and using linear data, the accuracy drops to 88%. Since when a part of speech tagger is used in a language modeling system, we can't ignore unknown words and will often come across writing styles which are not represented in the training data, this 88% accuracy is probably closer to the "true" state of the art.

4 Conclusion

In this paper, a variation on the standard Markov Model Tagger was created and compared with other Markov Model Taggers which have been implemented. The accuracy of the tagger compared favourably with results reported by others when tested under the same conditions. However when the testing conditions were more stringent, the accuracy of the tagger dropped considerably. This suggests that there is much room for improvement on the part of speech tagging task. In particular better modeling of unknown words and a model which is flexible enough to adjust to varied styles should greatly increase the accuracy of the tagger.