Final Project: A Simple, Memory-Efficient, Bayesian
Part-of-Speech Tagger with constant time update.

David Stein, Brian Cass

May 13, 2011

Abstract

In this project we explore a Bayesian part-of-speech (POS) tagging technique with a focus on low memory profile and computational demands. We achieve this by representing our beliefs about a word and its corresponding part-of-speech as a probability density function (PDF) and a confidence value instead of a tag. By computing trigrams and bigrams as combinations of parts-of-speech instead of combinations of words, we reduce the memory requirement to the size of generating n-gram priors, and demonstrate a linear solution in the size of the vocabulary. By utilizing confidence metrics, we can achieve arbitrary accuracy by choosing to skip difficult words, and instead reducing search space for more sophisticated taggers by restricting possible options to a subset of possible tags in those cases.
1 Introduction

Automatic part-of-speech tagging is a relatively well founded and explored field. Marcus et. al. presented the Penn Treebank in ’94, after four years of manual tagging, allowing for large scale testing and training of automated parsers using that dataset [?, ?]. Within two years, Ratnaparkhi introduced an entropy model for tagging that achieved better than 97% accuracy, which remains a base standard for parsers to date [?, ?]. The maximum entropy model has been improved multiple times since then [?], and many approaches with comparable accuracy using Bayesian priors and Markov chains were introduced [?, ?, ?, ?]. This year Mihalcea established an upper bound on tagging accuracy on the Penn Treebank using combined results from established techniques of just over 98%. Finally, Manning, who presented some of the earlier results on maximum-entropy tagging, began exploring the possibility of a return to the linguistic approach to part-of-speech modeling [?, ?].

Despite these advances in automated tagging to the point of near-proven exhaustion of the space, most of the high accuracy taggers presented require a large amount of either memory or computational power (for example, the Stanford tagger takes between 200 and 1000 MB of active memory). None of the taggers we explored have a more elegant failure mode than selecting the wrong answer. As low-capability devices connected to the cloud are becoming more prevalent again, there is some value in presenting algorithms that can do most computations locally and choose difficult tagging tasks to elevate to an external tool. Our approach is
to compute an n-dimensional vector representing both the strength of our belief and the relative confidence of any one part-of-speech (section 2.1). Using this approach, we can achieve comparable accuracy to other Bayesian approaches using less training data, are able to identify which tags are wrong with a reasonable degree of confidence, and can compute the trade-off between accuracy and number of words left untagged.

2 Methodology

We use a simple combination of Bayesian learning and n-grams to compute a PDF over all parts-of-speech for each word. The basic intuition behind our algorithm is that, given a belief state for each word, we can improve our beliefs without specific knowledge about neighbors by assuming every input is as grammatical as our training set.

2.1 Problem Formulation

We are given some tagged training set of sentences, where each word is labeled with its part-of-speech. After training, when given a new sentence $s$ we would like to compute some hypothesis vector for each word $s_i \in s$, $\hat{H}_{s_i}$, such that each element in the vector corresponds to a part of speech. We define $\hat{H}$ to be the probability vector for each hypothesized part-of-speech, and $||\hat{H}||$ to be our confidence in the correctness of the hypothesis. For the sake of clarity and because we would like our approach to be language independent, we use $AA$, $BB$, and $CC$ to denote
generic parts-of-speech, which include our special character \textit{EPSILON} used for the boundaries of sentences.

\section*{2.2 Algorithm}

We now present our tagging algorithm, which is at a high level essentially a simple Bayesian tagger (Algorithm 1)

\textbf{Algorithm 1} Generating Tag PDFs
\begin{algorithmic}[1]
\State \textbf{for} \textit{s} \textit{i} \textbf{\in} \textit{s} \textbf{do}
\State $\mathcal{H} \leftarrow \text{Compute priors (section 2.2.1)}$
\State \textbf{end for}
\State \textbf{for} \textit{n} \in \text{gramsizes} \textbf{do}
\State \textbf{for} \textit{s} \textit{i} \textbf{\in} \textit{s} \textbf{do}
\State update $\mathcal{H}_{s_i}$ (section 2.2.2)
\State \textbf{end for}
\State \textbf{end for}
\State \textbf{Return values based on application (section 2.2.3)}
\end{algorithmic}

\subsection*{2.2.1 Computing Priors}

We update our beliefs using a simple \textit{n}-gram approach, using parts-of-speech as opposed to words as our elements. Using our training set, we can compute the probability of certain part-of-speech pairs:

$$p(AA, BB|AA) = \frac{c(AA, BB)}{\sum_{YY \in \text{POS}} c(AA, YY)} \quad (1)$$

And so on symmetrically for tri-grams and \textit{n}-grams. The benefit of this is that storing this information varies with the number of parts-of-speech in the language
(which is orders of magnitude smaller than the actual vocabulary of the language),
converges to steady state with relatively small training sets, and requires essen-
tially constant storage space. (*Note: in the English language storing this data as
big-ints for up to trigrams takes only $4 \times (36^3 + 36^2 + 36)$ bytes, or about $0.4$ mB,
which is significantly smaller than the capacity of many micro-controllers*)

Having established this $n$-gram data, we need only provide some way to gen-
erate priors for each word. This area can be optimized for each language (for
example, in English using capitalization, prefixes, suffixes, and sentence length
can provide priors to many words in most sentences with virtually no data re-
quirements), however in the approach we investigate here we build our priors by
using the frequency each word is tagged, adding some small value $\delta$ to each term
to allow for results with few exemplars, causing out-of-vocabulary examples to
default to a uniform pdf:

$$p(AA|w) = \delta + \frac{c(w \text{ tagged } AA)}{c(w)} \forall AA$$  \hspace{1cm} (2)

### 2.2.2 Updating Beliefs

After establishing priors at each step, we would like to update those beliefs based
on our part-of-speech. To to this, we want to generate a new hypothesis based on
surroundings:

$$\mathcal{H}'_{s_i, AA} = \sum_{BB \in POS} \mathcal{H}_{s_{i-1}, BB} \cdot p(BB, AA|BB)$$  \hspace{1cm} (3)
We can do this both forwards and backwards, and after each different length of $n$-gram, recompute $\mathcal{H}_s$, for each word by taking a scaled sum of both hypothesis vectors:

$$\mathcal{H} = \left(\frac{1}{n}\right) \mathcal{H}' + \left(\frac{n - 1}{n}\right) \mathcal{H}$$  \hspace{1cm} (4)

You may note this does not necessarily sum to 1, we allow this because we can treat the magnitude of the vector as a measure of our confidence in the PDF provided by normalizing $\mathcal{H}$.

### 2.2.3 Formatting output

Because we return a PDF for each word, the output of the algorithm is application driven. By establishing thresholds for both the magnitude of $\mathcal{H}$ and the ratio of the highest values in the vector, we can decide when we have the confidence to return a best option, and otherwise return either a set of possibilities with higher probability or the suggestion to reanalyze using a more computationally demanding algorithm.

### 2.3 Analysis

The time required to build up the training data is $O(V)$, where $V$ is the size of the vocabulary. Additionally, the space required by this algorithm is also $O(V)$. The build time is dominated by the computation of priors, and otherwise is both constant and very small. Similarly, the runtime for sentence tagging after training is limited by the lookup or computation time of the priors and each word and
because our tagger returns PDFs rather than tags, a one-to-one comparison with other taggers is difficult. Here we discuss some of the preliminary results of using the system proposed above.

By training our system on a sample of 90% of the Penn Treebank and running it on a sample from the remaining sentences, we found that we have a tagging accuracy of approximately 83% without using the cutoff values, but identify the correct tag within three guesses 96.8% of the time, which is comparable to state-
Figure 3: The selection of what tags are most accurate is both a function of our confidence $||H||$ and our number of high probability candidates. Here we see results choosing different cutoffs for both values. (y axis: words, x axis (confidence, ratio) pairs. We would like to maximize the blue regions and minimize the red ones.

of-the-art systems. However, by introducing thresholds based on the confidence ($||H||$) and ratio between the most probably POS’s, we can discard low probability results and greatly increase the accuracy of tagged words (figure 4).

These results were found with bi-gram combinations, which in our implementation took approximately 5k of memory, (which is much smaller than many micro-controllers on the market [?]). We also stored our words as a trie mapping to compressed PDFs, which occupied 6.1MB of space after training on the entire Penn Treebank, but could be reduced significantly by removing entries with small (1 or 2) numbers of exemplars, which had very low confidence and would have marginal effect on our algorithm.
Figure 4: By selectively choosing not to tag data, our algorithm can identify tags that do not have high confidence, which could allow for more accurate parsing despite relatively low confidence in a tag.

4 Conclusion and Possible Extensions

4.1 Applications and Extensions

We have presented a lightweight tagging algorithm that identifies both likely tags and its confidence. This may have application in small devices with limited computational power or memory, especially devices like smart phones that have the option to elevate more difficult tasks to the cloud when available. We could also develop a more accurate second parser to specifically deal with the most frequent cases rejected by our first parser.

Additionally, work should be done to analyze the effectiveness of training data size on accuracy. While we postulate that larger training data should have minimal effect on results, we lack sufficient test data from varied databanks to draw conclusions. In a similar vein, this algorithm is Bayesian and could be modified to update itself in a semi-supervised manner.
4.2 Remarks

We treat this as a generic, language independent algorithm, and many of the errors are artifacts of this. For example, setting our confidence to a unit vector for POS’s like ‘.’ and ‘CD’ and zero otherwise is both easy to do and would greatly improve our priors. Likewise, adding intelligence about the language we are working with is a simple extension that would allow this to run faster and more accurately. Using a confidence measure to avoid assignment to the least certain words, part-of-speech tagging can be made much more accurate and less resource intensive, potentially even allowing integration into micro-controllers. By only looking into POS combinations, we reduce memory greatly without sacrificing accuracy.

References


