Part-of-Speech Tagger for Swedish

Simon STÅHL
Computer Science, Lund University
sys03sis@ludat.lth.se

Abstract

A Part-of-Speech (POS) tagger is a tool that automatically resolves the ambiguities that would occur if a text was tagged with the help of a dictionary. Automatic tagging of texts is used in many applications (grammar checkers, etc.), and quite high accuracy can be achieved.

This document describes a stochastic POS tagger that uses a unigram version of the Viterbi algorithm. The overall idea behind the stochastic POS tagger and the Viterbi algorithm is also described.

The unigram tagger is evaluated using a small corpus of just below 100 000 words, and the results indicate that a larger corpus would have yielded greatly improved accuracy.

1 Introduction

There are many applications that needs a text to be tagged with Parts-of-Speech (POS), the lexical categories of words and symbols in a text, and there are several ways to do this tagging. The most primitive approach is to determine the POS of a word by looking it up in a dictionary. This unfortunately leaves us with a lot of ambiguities, since many words have more than one possible POS.

Early POS taggers resolved these ambiguities by using hand coded rules, but writing these rules is both time demanding and complex. Newer rule based taggers derives rules automatically from a hand annotated corpus.

Other POS taggers uses stochastic models. The ambiguities is resolved using statistics, derived from a hand annotated corpus. The probabilities of the possible tag sequences of a given word sequence is calculated, and the one with the highest probability is chosen. This is approximated using N-grams (bigrams and trigrams mainly), since statistics of long sequences is impossible to obtain.

The Viterbi algorithm can be used to optimize the probability calculation of the tag sequences by discarding sub-sequences that can not be part of the tag sequence with the highest probability. This makes the stochastic POS tagger less time and memory consuming than if it would have to calculate all possible paths.

This document describes a stochastic POS tagger, using the Viterbi algorithm. The current implementation uses only unigrams, which makes the result noticeably less correct than a bigram or trigram implementation.

Chapter two describes the POS tagger with its statistics and probabilities. In the third chapter the Viterbi algorithm is described, and in chapter four the results are discussed.

2 The POS tagger

2.1 Statistics

All statistics are derived automatically from a hand annotated corpus. This training of the POS tagger only needs to be done once, since the statistics is saved to file to be used when tagging. The statistics derived is:

C_n – The number word tokens.
$C(w,t)$ – Occurences of word w tagged with t.
$C(t)$ – Occurences of the tag t.
$C(t_1,t_2)$ – Occurences of the tag bigram t_1,t_2.
$C(t_1,t_2,t_3)$ – Occurences of the tag trigram t_1,t_2,t_3.

Since the current implementation only uses unigrams, no bigram or trigram statistics is saved, as would be the case in a more advanced implementation.

2.2 Probabilities

The probability of a tag sequence T for a given word sequence is:

$$P(T)P(W|T)$$

That is, the probability of the tag sequence in it self multiplied with the probability of the word sequence knowing the tag sequence. The tag sequence with the highest probability is chosen to tag the word sequence with.

The probabilities $P(T)$ and $P(W|T)$ are approximated using N-grams, most often trigrams, backing off to bigrams (and unigrams) in the case of missing data. The trigram approximations of the probabilities are:

$$P(T) \approx P(t_1) P(t_2|t_1) \prod_{i=3}^{n} P(t_i|t_{i-2}, t_{i-1})$$

$$P(W|T) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

These probabilities are estimated with the statistics from the hand annotated corpus:

$$P(t_i) = \frac{(C(t_i))}{C_n}$$

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

$$P(t_i|t_{i-2}, t_{i-1}) = \frac{(C(t_{i-2}, t_{i-1}, t_i))}{(C(t_{i-2}, t_{i-1}))}$$

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

Since the current implementation only uses unigrams, the probability *P(T)* is approximated further, on the expense of less correct tagging. The unigram approximation of *P(T)* is:

$$P(T) \approx \prod_{i=1}^{n} P(t_i)$$

The resulting unigram approximation of a tag sequence is:

$$P(T) P(W|T) \approx \prod_{i=1}^{n} P(t_i) P(w_i|t_i)$$

The unigram approximation only selects the most common tag of a word, and does not take any previous tags into consideration when selecting tags.

3 The Viterbi algorithm

The Viterbi algorithm is a dynamic programming algorithm that optimizes the tagging of a sequence, making the tagging much more efficient in both time and memory consumption.

In a naïve implementation we would calculate the probability of every possible path through the sequence of possible word-tag pairs, and then select the one with the highest probability. Since the number of possible paths through a sequence with a lot of ambiguities can be quite large, this will consume a lot more memory and time than neccesary.

Since the path with highest probability will be a path that only includes optimal subpaths, there is no need to keep subpaths that are not optimal. Thus the Viterbi algorithm only keeps the optimal subpath of each node at each position in the sequence, discarding the others.

4 Results

The tagger was tested with a corpus of just below 100 000 Swedish words. The original idea was to test it with the full corpus (with approximately 1.2 million words), but this unfortunately proved to be too time demanding, both when training the tagger and when the statistics was loaded before tagging.

A test set of just below 25 000 words was tagged by the tagger, that had been trained on the small (100 000 words) corpus, and the results were compared with a copy of the test set that was hand annotated. The tagger was able to tag 74.6% of the words correctly, which is way below what could be expected. As a comparison, Sjöbergh (2003) report an 87.3% accuracy when using an unigram tagger trained on a corpus of 1.1 million words.

The reason for this difference in accuracy is probably mainly because of sparse data, many words in the test set are not found in the small corpus, and this is not handled by the tagger. 71.6% of the erroneous taggings were, as a matter of fact, tagged with the tag UKN, which means that the tagger was not able to find any possible tags for that word. This indicates that a larger corpus would have given a result closer to that of Sjöbergh (2003).

5 Conclusion

The implemented POS tagger only uses unigram probabilities, which means that it never takes any sequence of tags into consideration, only selects the most common tag of a word. It does not try to tag unknown words in any way, it just tags them as UKN, unknown, and it was trained on a small corpus of just below 100 000 Swedish words. But with the limitations of both the tagger and the small corpus in mind, it gave relatively good results, 74.6% correct tagged words.

The large percentage, 71.6%, of the erroneous taggings, that were tagged with the tag for unseen word, indicates that a larger corpus would give a higher percentage of correctly tagged words. A faster version of the tagger would have been able to be trained on the larger corpus, but the current version would have taken days to process it.

Extending the tagger to use bigrams and trigrams would also improve the correctness, and would be the next natural step. This would take sequences of tags into consideration, which is important in natural languages.

The handling of unseen words is non-existent in the current implementation, and even a naïve algorithm to handle this would improve the results.

References

Pierre Nugues. 2004. *An Introduction to Language Processing with Perl and Prolog.* Lund University, Lund, Sweden.

Johan Carlberger and Viggo Kann. 1999. *Implementing an efficient part-of-speech tagger.* Royal Institute of Technology, Stockholm, Sweden.

Jonas Sjöbergh. 2003. *Combining POS-taggers for improved accuracy on Swedish text.* KTH Nada, Stockholm, Sweden.

University of Leeds, Viterbi algorithm: http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/viterbi_algorithm/s1_pg1.html

Wikipedia, Viterbi algorithm: http://en.wikipedia.org/wiki/Viterbi_algorithm

Luz Abril Torres Méndez. 2000. *Viterbi Algorithm in Text Recognition.* McGill University, Montreal, Quebec, Canada: http://www.cim.mcgill.ca/~latorres/Viterbi/va_main.html