March 16, 2015

Natural Language Models and Interfaces: Assignment Part A

Cornelis Boon - 10561145, Markus Pfundstein - 10452397,
Thomas Meijers - 10647023

Abstract

Using Viterbi's algorithm, a trigram language model and a word-tag pair lexical model, we have implemented a Part-of-Speech(POS) tagger. This POS tagger attempts to find the most likely sequence of POS tags for a given sentence. To increase the accuracy of the POS tagger, we smoothe both models using Good-Turing discounting. Without smoothing, the POS tagger has an accuracy of almost 70% on the corpus that we have tested on. With smoothing, the accuracy goes up to more than 90%.

1. Introduction

In the field of linguistics, scientists have always been interested in the syntax of a language. A way to this is to look at the word classes of words in sentences and see if there is some sort of structure or order. These word classes are also known as Part-of-Speech tags (POS tags). In school, we are taught what these are and how to find them. How to identify nouns, verbs, propositions and such. However, to do this by hand, and to for many texts, it requires many laborous hours and one would hope for a more efficient solution.

In this assignment we have built a Part-of-Speech(POS) tagger. This Part-of-Speech tagger will try to assign POS tags to given sentences. This way, many hours of work and annotation can be done in a matter of minutes.

Other applications than simply tagging texts would be to look for underlying structure in languages. It could also be used in semantics and other linguistic fields.

In this paper, we will look at how correct our POS tagger is by looking at its accuracy and how much the accuracy increases when the models that the POS tagger uses, are smoothed.

2. Approach

The POS tagger we have implemented in python. (see Appendices for details on how to run).

2.1. Language model & Lexical model

For both models we have written a separate class. Each class contains the relevant n-gram model as well as an n-gram model of an order lower such that probabilities can be calculated for the sequences of words and tags.

2.2. Smoothing

To increase the accuracy of the estimates and to be able to incorporate 'unseen events', word-tag pairs or trigram-combinations of POS tags that haven't been seen in the training corpus, we use Good-Turing smoothing. For the language model, we smooth over the trigrams that have appeared up to 4 times. For this we use the following formula:

$$r^* = \frac{(r+1)\frac{n_{r+1}}{n_r} - r\frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} \tag{1}$$

With: r, the frequency of a trigram; n_r, the amount of trigrams that have a frequency; and k, the max frequency that gets smoothed. (In this case, k is 4) For unseen events, we give them a probability with the following formula:

$$P(0)_{GT} = \frac{n_1}{n_0 \cdot N} \tag{2}$$

With, N being the total number of seen events and and n_0 being the estimated number of unseen events, which is currently equal to $V^3 - N$, with V the number of unique words in the corpus.

For the lexical model, we only smoothe over the word-tag pairs that have appeared once. This is due to that a lot of POS tags have very low frequency. We also set the estimate of unseen events to one.

$$1^* = \frac{1}{2} \tag{3}$$

And the probability is thus given by:

$$P(0)_{GT} = \frac{1}{2}\frac{n_1}{n_0} \tag{4}$$

2.3. Viterbi

The Viterbi algorithm is used to obtain the most likely sequence of POS tags for a given series of words using a Hidden Markov Model (HMM). The observations of the HMM are the words and the hidden states are the bigram POS tags . It is linear in the number of words and therefore well suited for the problem. It takes as input a sentence and returns a sequence of POS tags. It is a dynamic programming algorithm which means that intermediate results get stored in a lookup table. This allows for rapid computation as calculations that have been done already won't be done again.

The probability of a state s_i at time t can be calculated by the following formula:

$$\pi_t(s_i) = P(word_i|s_i) \cdot max_{s_j}(P(s_i|s_j) \cdot \pi_{t-1}(s_j)) \tag{5}$$

The probability $P(word_i|s_i)$ is calculated by counting how likely the emission of a word is, given a POS tag. Hence the formula is:

$$P(word_i|s_i) = \frac{count(word_i, s_i)}{count(s_i)} \tag{6}$$

The transition probability $P(s_i|s_j)$ is calculated using a trigram model over the POS tags, thus each state s_j is a bigram, and s_i is a trigram which extends the bigram s_j by a new POS tag. The probability is calculated using the formula:

$$P(s_i|s_j) = \frac{count(s_i)}{count(s_j)} \tag{7}$$

2.4. Experiments

We have used section 2-21 of the WSJ[1] as our training set, from which build our language and lexical models. On section 23 of the WSJ, we have tested the POS tagger by letting the POS tagger tag each sentence. We have only considered sentences that are 15 words long or shorter in this experiment. To evaluate the 'correctness' of the POS tagger, we looked at the accuracy of the POS tagger. This was done by comparing the estimated sequence of tags with the annotated tags in section 23. Only fully correct sentences were considered to be accurate estimates.

[1]Wall Street Journal corpus

3. Results

GT Smoothing	**Accuracy**
No	68.06%
Yes	92.52%

Please see the attachmented files *nosmooth_out.txt* and *GTsmooth_out.txt* for further information. They contain every sentence and its predicted POS tags, one for the unsmoothed run and one for the smoothed run.

4. Discussion

The obtained results from the POS-tagger are in line with the expectations (based on results of fellow-students and communicated by the Teaching Assistents). The accuracy of 68.06% without smoothing appear quite low but are explained by the fact that when a new word is encountered, the obtained POS sentence will be empty (as the probability will be 0%). This does not give us 1 missing POS tag but n, where n is the length of the sentence, which has a severe impact on the accuracy.

This fact is also shown that by using Good-Turing smoothing in our language and lexical models the accuracy increases with more than 24%. The obtained accuracy is also one that is relatively good, as our simple implementation of Viterbi and GT smoothing comes reasonably close to state of the art POS-taggers, which obtain an accuracy of around 97% (See Manning (1)). A side note on this is that our tagger is trained and tested only on the WSJ corpus, using other corpora and testing on other texts might yield different results. However, for this assignment, training and testing on the WSJ corpus is more than sufficient.

The efficiency of our implementation is one thing we did not focus on as the implementation of the algorithms took away most of our time. A full run on the WSJ testing corpus with GT smoothing takes roughly 10 minutes. From a commercial point of view this would be unacceptable, but in the context of this assignment, we assume that this is acceptable.

Appendices

A. Run instructions

```
usage: a1_step4.py [-h] [-train-set TRAIN_FILE] [-test-set TEST_FILE]
                   [-smoothing SMOOTH] [-test-set-predicted TEST_SET_PRED]

Assignment A, Step 4

optional arguments:
  -h, --help            show this help message and exit
  -train-set TRAIN_FILE
                        Path to training file
  -test-set TEST_FILE   Path to test file
  -smoothing SMOOTH     Use good-turing (yes/no)
  -test-set-predicted TEST_SET_PRED
                        Path to file with predictions
```

[1] Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *Computational Linguistics and Intelligent Text Processing*, pages 171–189. Springer, 2011.